LEARN TO PRODUCE VIDEO

WITH  FFMPEG

IN 30 MINUTES OR LESS

BY JAN OZER

Thanks for downloading this chapter, which is Chapter 12 Encoding HEVC from the book Learn to Produce Video with FFmpeg in 30 Minutes or less, which is available in print and PDF versions.

As you'll see, the chapter is short and sweet, covering the effectiveness of the x265 presets in FFmpeg, suggesting an encoding ladder, and supplying command lines to produce that ladder. If you buy the book, you'll be a sent a file with all Windows command lines so you can easily adapt them for your own use.

Here's a high level Table of contents.

The book is targeted towards FFmpeg newbies, and really should get you up and running in 30 minutes or less. Thanks for having a look.

For more information on the book, please click here or navigate to:
http://streaminglearningcenter.com/learnffmpeg.html

Chapter 12: Encoding HEVC

HEVC is the standards-based successor to H.264 that's primarily targeted towards Smart TVs and set-top boxes. As with H.264, there are multiple HEVC codecs out there; the codec that's available in FFmpeg is called x265. As you'll see, working with x265 is very similar to working with x264 except that the encoding times are much longer and the quality is much better.

In this chapter, you'll learn:

- considerations for encoding HEVC, including a look at HEVC encoding profiles
- how to encode using the x265 codec in FFmpeg.

During the chapter, I'll identify FFmpeg commands for various x265 parameters. However, you'll have to apply them in a special way, which I detail towards the end of the chapter.

What is HEVC

HEVC is a standards-based codec created by the same groups that created H.264. Like H.264, you produce HEVC by choosing different profiles, and x265 offers presets and tuning mechanisms.

You can produce x265 in FFmpeg, which I'll demonstrate in this chapter, or with the x265 executable that you can download from http://x265.org/. The quality should be identical, but you'll have to convert your source files to YUV or Y4M formats to input them into x265, which is time-consuming and can consume tons of disk space. With FFmpeg, you can input the same files you've been using for H.264, which is faster and easier.

Like H.264, HEVC is a codec, not a container format. While you can package HEVC in multiple container formats, single files are typically encoded in the MP4 container format, while adaptive bitrate files are typically packaged in Dynamic Adaptive Streaming over HTTP (DASH) format.

Basic HEVC Encoding Parameters

With this as background, let's talk basics of HEVC encoding. At a high level, all non-H.264 specific lessons learned in previous chapters regarding bitrate control, I-, B-, and P-frames, resolution, and frame rate apply here. Beyond these, you'll have to choose a profile, a preset, and if desired, a tuning mechanism. Of course, you'll have to choose the codec first, which you do with the following command string.

```
-c:v libx265
```

Since AAC is the audio codec for HEVC, no changes are necessary there.

HEVC Profiles

Most HEVC encoding tools let you select the Main or Main 10 profile. The Main profile supports 8 bits per sample, which allows 256 shades per primary color, or 16.7 million colors in the video. In contrast, the Main 10 profile supports up to 10 bits per sample, which allows up to 1024 shades and over 1 billion colors. Of course, you'll need a 10-bit display to see the extra colors, which most potential viewers don't have at this point. That's because the real targets of Main 10 output are high-dynamic-range (HDR) displays, which are just starting to ship in quantity in 2017.

In addition, if your video has an 8-bit color depth, which most formats do, encoding in 10-bit won't add the colors and improve video quality. On the other hand, some experts argue that processing in 10-bit color may improve the encoding precision of 8-bit source videos, even if it doesn't add colors. My tests didn't quite confirm this claim, as you can see in Table 12-1.

720p - x265	Main	Main 10	Delta
Tears of Steel	37.05	37.73	1.84%
SIntel	41.37	41.25	-0.29%
Big Buck Bunny	37.21	37.16	-0.13%
Talking Head	41.15	41.15	0.00%
Freedom	39.70	39.57	-0.31%
Haunted	39.56	41.78	5.61%
Average	**39.34**	**39.77**	**1.12%**

Table 12-1. Main 10 delivered slightly higher quality than Main.

Here, I encoded 8-bit source videos using the Main and Main 10 profiles and otherwise identical features. Although the Main 10 encoded videos averaged slightly higher quality, four of the six videos were either about the same or worse quality.

Before chasing the extra quality some claim Main 10 can deliver, note that if you encode your video using the Main 10 profile, only Main 10 compatible decoders can play the video. Most early HEVC players were not Main 10 compatible, so if you're distributing HEVC videos to the general public, rather than to specific smart TVs or set-top boxes, there is a compatibility risk (see Figure 12-1).

So, I recommend using the Main profile for general-purpose distribution, even if your video is 10-bit in origin. If you're distributing to known Main 10 compatible HEVC decoders, you should consider encoding with Main 10 even if your video is 8-bit in origin. Obviously, if you're encoding HDR video, which is beyond the scope of this tutorial, you'll need to use Main 10.

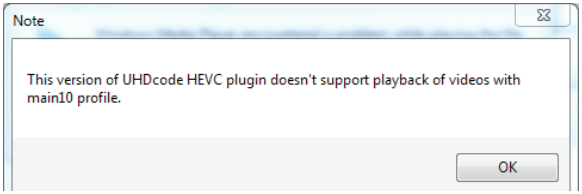

Figure 12-1. Only Main 10 compatible players can play Main 10 files.

> **Note:** *To encode to Main 10 using FFmpeg and x265, you'll have to download or compile a Main 10-specific version. You can't call the Main 10 x265 codec from the standard downloadable version of FFmpeg.*

Quick Summary: HEVC Profiles

> ***1.*** If you're encoding video for general-purpose distribution, use the Main profile for the broadest possible compatibility.
>
> ***2.*** If you're producing for a platform or platforms with known Main 10 compatibility, encode using the Main 10 profile, whether the source footage is 8-bit or 10-bit.

As mentioned, to create Main 10 output with FFmpeg, you'll need an FFmpeg build with 10-bit libx265. Once you have that, you can specify the profile using one of these strings:

```
-profile main

-profile main10
```

x265 Presets

The x265 encoding presets share the same names as the x264 presets, and Table 12-2 shows their comparative quality. As you can see, the Ultrafast preset always produced the lowest quality, and Placebo the highest. Interestingly, quality actually dropped after the Superfast preset, and didn't surpass that level until the Fast preset. Overall, the average difference between the highest and lowest scores was 6.7 percent.

	Ultrafast	Superfast	Veryfast	Faster	Fast	Medium	Slow	Slower	Veryslow	Placebo	Total Delta
Tears of Steel	37.25	38.06	38.04	38.05	38.34	38.39	38.84	38.86	38.93	39.00	4.70%
Sintel	35.87	36.89	36.66	36.67	37.11	37.25	37.74	37.79	37.90	37.97	5.86%
Big Buck Bunny	36.10	37.65	37.61	37.60	37.91	38.26	38.70	38.89	39.03	39.18	8.54%
Freedom	38.16	39.01	38.45	38.46	38.71	38.98	39.36	39.44	39.52	39.58	3.72%
Haunted	41.36	41.77	41.39	41.39	41.55	41.68	41.97	41.92	41.97	42.02	1.60%
Screencam	44.03	46.70	46.55	46.54	46.78	47.12	48.31	48.69	48.99	49.34	12.07%
Tutorial	42.46	47.14	46.46	46.42	46.52	47.19	48.35	47.65	48.02	48.53	14.31%
Average	**38.64**	**39.51**	**39.30**	**39.31**	**39.58**	**39.74**	**40.13**	**40.18**	**40.27**	**40.35**	**6.70%**

Table 12-2. PSNR quality by video file and encoding preset.

How does encoding time factor in? This is shown in Figure 12-2. Here I've normalized encoding time on a scale from 0 to 100, with the time of the Ultrafast encode set to 0. The quality side shows the percentage of quality each preset delivers as compared to the Placebo preset that delivers maximum quality or 100%.

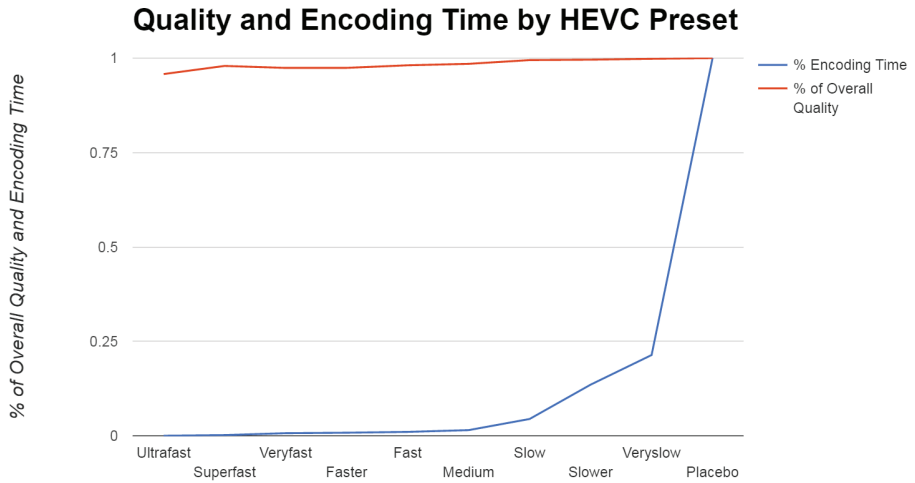

Figure 12-2. Quality versus encoding time by x265 preset.

From an encoding time perspective, the needle barely even moves until the Medium preset, and the quality jump from Medium to Slow makes the Slow preset look like an obvious move. At Slow, you're just under 99.47 percent of total quality, and encoding time for the higher quality presets really starts to jump. If you're running out of capacity, it's worth experimenting with Superfast, as the bang for your encoding time buck is substantial. To put the numbers in perspective, PSNR for Superfast averaged 39.51 dB, while Slow averaged 40.13 dB, which isn't a difference that most viewers would notice.

You choose x265 presets just like x264 presets, using this string:

```
-preset [preset]

-preset veryslow
```

As with x264, if you don't specify a preset, FFmpeg will use the default Medium preset.

> **Tip:** *Note that you can see the exact configuration options used for each preset at [bit.ly/x265_pre](bit.ly/x265_pre), a page created by x265 developer MulticoreWare.*

Our HEVC Encoding Ladder

Table 12-3 shows an encoding ladder for HEVC encodes. Basically, I decreased the data rates for the H.264 encodes by about 50%, eliminated the lowest rung of the ladder and added rungs for 1440p and 2160p. While this is as generic as you can get, it gives us a starting point for the HEVC encodes that I'll demonstrate below.

	Width	Height	Frame Rate	Video Bitrate	Peak Bitrate	Buffer	Profile	Preset	Key-frame	B-frame	Ref frame	Audio Bitrate
270p	480	270	30	220,000	242,000	220,000	Main	Slow	2	3	5	64,000
360p_l	640	360	30	400,000	440,000	400,000	Main	Slow	2	3	5	64,000
360p_h	640	360	30	720,000	792,000	720,000	Main	Slow	2	3	5	96,000
720p_l	1,280	720	30	1,000,000	1,100,000	1,000,000	Main	Slow	2	3	5	128,000
720p_h	1,280	720	30	1,800,000	1,980,000	1,800,000	Main	Slow	2	3	5	128,000
1080p_l	1,920	1,080	30	2,500,000	2,750,000	2,500,000	Main	Slow	2	3	5	128,000
1080p_h	1,920	1,080	30	4,000,000	4,400,000	4,000,000	Main	Slow	2	3	5	128,000
1440p	2,560	1,440	30	6,000,000	6,600,000	6,000,000	Main	Slow	2	3	5	128,000
2160p	3,840	2,160	30	10,000,000	11,000,000	10,000,000	Main	Slow	2	3	5	128,000

Table 12-3. Encoding ladder for HEVC encodes.

Note that we don't have a lot of commercial comparisons we can use because there's very little HEVC-encoded content available for testing. Netflix reportedly encodes *House of Cards* at 16 Mbps, which blows my numbers out of the water, but you must start somewhere.

> ***Tip:*** *Just as I was finishing this book, I wrote an article on HDR production for Streaming Media Magazine. Look for it for a solid overview of HDR production. One fabulous article I found in my research was entitled, HDR Video Part 5: Grading, Mastering, and Delivering HDR, and it includes a detailed description of how to produce HDR with an FFmpeg front end called Hybrid. If you need to learn how to produce HDR with FFmpeg, check out the article at bit.ly/hdr_ffmpeg.*

x265 and FFmpeg

As mentioned above, encoding to x265 is easier in FFmpeg than using the x265 executable because you don't have to pre-convert the files to YUV/Y4M. On the other hand, there really is very limited documentation for the x265 controls available in FFmpeg, which is a pain.

In contrast, x265 is very well documented (see bit.ly/x265_documentation). In theory, you can add any x265 configuration option to an FFmpeg command script by adding `-x265-params` to the FFmpeg command string and adding the x265 parameters after that (see bit.ly/x265_ffmpeg). In practice, however, it's not quite that simple, and I had to evolve into the following approach to produce the desired format. I'm not sure that this is the only way to do it, or even the best way, but it worked for me.

1080p Conversion

Here's a script I used to create the 1080p and lower ladders from Table 12-3 from 1080p source.

```
ffmpeg -y -i TOS_1080p.mov -c:v libx265 -preset slow -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=4000:vbv-maxrate=4400:vbv-bufsize=4000 -an -pass 1 -f mp4
NUL && \

ffmpeg -i TOS_1080p.mov -c:v libx265 -preset slow  -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=4000:vbv-maxrate=4400:vbv-bufsize=4000 -an -pass 2
TOS_1080p_h.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -preset slow  -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=2500:vbv-maxrate=2750:vbv-bufsize=2500 -an -pass 2
TOS_1080p_l.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -s 1280x720 -preset slow -x265-
params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3
:b-adapt=2:bitrate=1800:vbv-maxrate=1980:vbv-bufsize=1800 -an -pass 2
TOS_720p_h.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -s 1280x720 -preset slow  -x265-
params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3
:b-adapt=2:bitrate=1000:vbv-maxrate=1100:vbv-bufsize=1000 -an -pass 2
TOS_720p_l.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -s 640x360 -preset slow  -x265-
params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=
3:b-adapt=2:bitrate=720:vbv-maxrate=792:vbv-bufsize=720 -an -pass 2
TOS_360p_h.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -s 640x360 -preset slow  -x265-
params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=
3:b-adapt=2:bitrate=400:vbv-maxrate=440:vbv-bufsize=400 -an -pass 2
TOS_360p_l.mp4

ffmpeg -i TOS_1080p.mov -c:v libx265 -s 480x270 -preset slow -x265-
params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=220:vbv-maxrate=242:vbv-bufsize=220 -an -pass 2 TOS_270p.
mp4

ffmpeg -i TOS_1080p.mov -vn  -c:a aac -b:a 128k   -pass 2  TOS_audio.mp4
```

Batch 12-1. Converting 1080p source to our HEVC encoding ladder 1080p rungs and below.

The funky things, of course, are that the size and preset are outside the `-x265-params` string while the other parameters follow it, and are tied together with colons. By this point, if you just follow carefully, you should have no trouble duplicating these results.

Note that I've included the -an switch (audio? no!) in the video outputs, and -vn (video? no!) in the final output for audio. Since I didn't need to change either the number of channels or sampling frequency, I didn't specify those in the audio string.

4K Scaling Exercise

We covered multiple examples of scaling back in Chapter 5. Since you're most likely to encounter these operations when working with 4K content, I wanted to test and make sure they still work when producing x265.

The following example inputs Tears of Steel at 3840x1714 resolution, with a display aspect ratio of 2.25:1 and outputs the encoding ladder shown in Table 12-3, save the high-quality versions of the 1080p, 720p, and 360p files, which I removed to save space. With the 4K and 2K files, I produced at full resolution (3840x2160, 2540x1440) with letterboxing, essentially using the command string shown in Batch 5-6. In all other resolutions, I produced at full resolution while cropping out the excess pixels as shown in Batch 5-5.

Here are the commands.

```
ffmpeg -y -i TOS_4k.mov -c:v libx265 -preset slow -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=4000:vbv-maxrate=4400:vbv-bufsize=4000 -an -pass 1 -f mp4
NUL && \

ffmpeg -i TOS_4k.mov -c:v libx265 -vf "scale=3840:2160:force_original_as-
pect_ratio=decrease,pad=3840:2160:(ow-iw)/2:(oh-ih)/2" -preset slow
-x265-params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=
3:b-adapt=2:bitrate=10000:vbv-maxrate=11000:vbv-bufsize=10000 -an -pass 2
TOS_4K_lb.mp4

ffmpeg -i TOS_4k.mov -c:v libx265 -vf "scale=2560:1440:force_original_as-
pect_ratio=decrease,pad=2560:1440:(ow-iw)/2:(oh-ih)/2" -preset slow
-x265-params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframe
s=3:b-adapt=2:bitrate=6000:vbv-maxrate=6600:vbv-bufsize=6000 -an -pass 2
TOS_2K_lb.mp4

ffmpeg -i TOS_4k.mov -c:v libx265 -preset slow -vf "scale=1920:1080:force_
original_aspect_ratio=increase,crop=1920:1080" -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=2500:vbv-maxrate=2750:vbv-bufsize=2500 -an -pass 2
TOS_1080p_l.mp4
```

```
ffmpeg -i TOS_4k.mov -c:v libx265  -preset slow -vf "scale=1280:720:force_
original_aspect_ratio=increase,crop=1280:720" -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=1000:vbv-maxrate=1100:vbv-bufsize=1000 -an -pass 2
TOS_720p_l.mp4

ffmpeg -i TOS_4k.mov -c:v libx265  -preset slow -vf "scale=640:360:force_
original_aspect_ratio=increase,crop=640:360" -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=400:vbv-maxrate=440:vbv-bufsize=400 -an -pass 2
TOS_360p_l.mp4

ffmpeg -i TOS_4k.mov -c:v libx265  -preset slow -vf "scale=480:270:force_
original_aspect_ratio=increase,crop=480:270" -x265-params
profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframes=3:b-
adapt=2:bitrate=220:vbv-maxrate=242:vbv-bufsize=220 -an -pass 2
TOS_270p.mp4

ffmpeg -i TOS_4k.mov -c:v libx265 -vf "scale=2560:1440:force_original_as-
pect_ratio=decrease,pad=2560:1440:(ow-iw)/2:(oh-ih)/2" -preset slow
-x265-params profile=main:keyint=48:min-keyint=48:scenecut=0:ref=5:bframe
s=3:b-adapt=2:bitrate=4000:vbv-maxrate=4400:vbv-bufsize=4000 -an -pass 2
TOS_2K_lbx.mp4

ffmpeg -i TOS_4k.mov -vn  -c:a aac -b:a 128k -pass 2  TOS_audio.mp4
```

Batch 12-2. Converting 4K source to our HEVC encoding ladder with letterboxing and cropping.

I checked the files and encoding, and the controls seemed to work as advertised. Give it a try!

So, that's HEVC. In the next chapter, you'll learn the ins and outs of encoding VP9.