FFMPEG Notes and Tips

Once support for AV1 comes to FFMPEG I will recompile it to include AV1 support. It will, supposedly, provide better compression than HEVC/H265 and **not be encumbered by patents** to the level of H265 and H264.

For an idea of the licensing nightmare HEVC has created see this article in The Register

There are times when I dearly wish I had a GUI to do some of the work, particularly with feature-rich applications like FFMPEG but we don't always have the chance or the choice. While there are tools like IFFMPEG they are not as comprehensive as I'd like them to be so we need to learn at least the basic of the command line.

I've chosen to install FFMPEG via Homebrew rather than compile it directly via XCode. The command will install FFMPEG with the optional Open H264 support enabled.

```
brew install ffmpeg --with-openh264
```

Once it is installed you get the `ffmpeg` command available. It'll be the basis for what follows.

Basic use

These are some of the basic commands that I use.

The first command gathers information about the video:

```
ffmpeg -i tears_of_steel_1080p.mov
```

And the result will be something like this. Note that the streams descriptions have hard returns added for readability, your result will look different.

```
  libavutil      55. 78.100 / 55. 78.100
  libavcodec     57.107.100 / 57.107.100
  libavformat    57. 83.100 / 57. 83.100
  libavdevice    57. 10.100 / 57. 10.100
  libavfilter     6.107.100 /  6.107.100
  libavresample   3.  7.  0 /  3.  7.  0
  libswscale      4.  8.100 /  4.  8.100
  libswresample   2.  9.100 /  2.  9.100
  libpostproc    54.  7.100 / 54.  7.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'tears_of_steel_1080p.mov':
  Metadata:
    major_brand     : qt
    minor_version   : 512
    compatible_brands: qt
    encoder         : Lavf53.32.100
  Duration: 00:12:14.17, start: 0.000000, bitrate: 6361 kb/s
    Stream #0:0(eng): Video: h264 (Main) (avc1 / 0x31637661), yuv420p,
    1920x800 [SAR 1:1 DAR 12:5], 6162 kb/s, 24 fps, 24 tbr, 24 tbn, 48 tb
    Metadata:
      handler_name    : DataHandler
      encoder         : libx264
    Stream #0:1(eng): Audio: mp3 (.mp3 / 0x33706D2E), 44100 Hz, stereo, s
    Metadata:
      handler_name    : DataHandler
```

Resizing the video

For this first example, we'll resize the video to (WxH) 640x480.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv -s 640x480 agent_327_480p.mp
```

Changing the video data rate

FFMPEG allows me to change the bit rate for both audio and video separately. In this example, I will only change the video bitrate to 2 megabits without changing the audio bitrate at all.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -b:v 2m \
  agent_327_480p.mp4
```

Changing the audio data rate

The two attributes that we want to work with for audio are `-b:a` and `-ab`. There is an older argument `-ar` that will accomplish the same thing but the recommendation is to use `-b:a` to be consistent with the video attribute.

The `-b:a` attribute controls the audio sampling frequency. In this example, I've set the value to 48000.

With `-ab` we get control of the audio bit rate (expressed in bits per second).

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -b:a 48000 -ab 120k  \
  agent_327_480p.mp4
```

Changing the aspect ratio of a video

The aspect ratio of an image describes the proportional relationship between its width and its height. It is commonly expressed as two numbers separated by a colon, as in 16:9.

In FFMPEG, this one is an easy one to change; use the `-aspect` flag and the aspect ratio than you want to use.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -aspect 16:9 \
  agent_327_480p.mp4
```

Changing the frame rate of a video

Frame Rate is the frequency (rate) at which consecutive images called frames appear on a display. Values above 12 are perceived by humans as motion.

To change it use the `-r` value

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -r 25 \
  agent_327_480p.mp4
```

Converting from one format to another

To make sure that support for all the codecs I want FFMPEG with support for `libvpx` and opus (the default audio format for VP9) using the following command:

```
brew reinstall ffmpeg --with-openh264 --with-x265 --with-tools --with-lib
```

This makes sure that all the codecs are installed and available for the following sections. **This is a one-time operation**. Once the codecs are installed you don’t need to reinstall them again

Converting to VP9

Let’s say that your client delivers a video in MP4 format and you need to deliver it as a high-quality VP9 video. Based on We’ll look

- 1-pass average bitrate
- 2-pass average bitrate
- Constant quality
- Constant bitrate

1-pass average bitrate

The simplest way to encode VP9 is the simple variable bitrate (VBR) mode VP9 offers by default. This is also sometimes called “Average Bitrate” or “Target Bitrate”. In this mode, it will simply try to reach the specified bit rate on average, e.g. 2 MBit/s.

```
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 \
-b:v 2M \
output-1-pass.webm
```

2-pass average bitrate

In order to create more efficient encodes when a particular target bitrate should be reached, you should choose the two-pass encoding.

For two-pass, you need to run ffmpeg twice as shown below; The differences between the two passes are:

- In pass 1 and 2, use the `-pass 1` and `-pass 2` options, respectively
- In pass 1, output to a null file descriptor, not an actual file. (This will generate a logfile that ffmpeg needs for the second pass)
- In pass 1, you need to specify an output format (with `-f`) that matches the output format you will use in pass 2
- In pass 1, specify the audio codec used in pass 2; in many cases, `-an` in pass 1 will not work
- In pass 1 specify a fast encode (`speed 4`) and in pass 2, a slower encode (`speed 1`). This should speed up the overall encode process

```
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 -b:v 2M -pass 1 \
-c:a libopus -speed 4 -f webm /dev/null && \
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 -b:v 2M -pass 2 \
-c:a libopus -speed 1 output.webm
```

Converting to H264 (AVI to H264)

When working with AVC/H264 video, for the most part, we need to work with Constant Rate Factor (CRF).

CRF is the default quality (and rate control) setting for the x264 and x265 encoders. Values range between 0 and 51, where lower values would result in better quality and larger file sizes. Higher values mean more compression, but eventually, the video quality will suffer noticeably degrade.

Pick the CRF values

The range of the CRF scale is `0–51`, where 0 is lossless, 23 is the default, and 51 is worst quality possible (see the note below for the difference between 8 and 10-bit encoding and the CRF values). A subjectively sane CRF range is 17–28 with a default of 23. Consider 17 or 18 to be visually lossless or nearly so; it should look the same or nearly the same as the input but it isn't technically lossless.

The range is exponential, so increasing the CRF value +6 results in roughly half the bitrate/file size, while -6 leads to roughly twice the bitrate.

Choose the highest CRF value that still provides an acceptable quality. If the output looks good, then try a higher value. If it looks bad, choose a lower value.

The `0–51` CRF quantizer scale mentioned on this section only applies to 8-bit x264. When compiled with 10-bit support, x264's quantizer scale is `0–63`. You can see what you are using by referring to the ffmpeg console output during encoding (`yuv420p` or similar for 8-bit, and `yuv420p10le` or similar for 10-bit). 8-bit is more common among distributors.

Presets

A preset is a set of predefined options that will provide a certain encoding speed to compression ratio. A slower preset will provide better compression. If you target a certain file size or constant bit rate, you will achieve better quality with a slower preset at the expense of a bigger file size.

Use the slowest preset that you have the patience to wait for. As with many things, you'll have to test the presets to see which one works best for your project.

The available presets in descending order of speed are:

- ultrafast
- superfast
- veryfast

- faster
- fast
- medium – default preset
- slow
- slower
- veryslow

You can see a list of current presets with `-preset help`

CRF Example

This command encodes a video with good quality, using the `slow` preset to achieve better compression:

```
ffmpeg -i input.avi -c:v libx264 -preset slow -crf 22 -c:a copy output.mk
```

Note that in this example the audio stream of the input file is simply not re-encoded, we just copy the stream over to the output.

If you are encoding a set of videos that are similar, apply the same settings to all the videos: this will ensure that they will all have similar quality.

Two-Pass

Two-Pass encoding is more complicated but it works better when trying to target a specific file size with frame output quality being a secondary concern. For this example we'll use the following formula to calculate bitrate:

```
(200 MiB * 8192 [converts MiB to kBit]) / 600 seconds = ~2730 kBit/s tota
2730 - 128 kBit/s (desired audio bitrate) = 2602 kBit/s video bitrate
```

You can also forgo the bitrate calculation if you already know what final (average) bitrate you need.

For two-pass, you need to run ffmpeg twice, with almost the same settings, except for:

- In pass 1 and 2, use the -pass 1 and -pass 2 options, respectively.
- In pass 1, output to a null file descriptor, not an actual file. (This will

generate a logfile that ffmpeg needs for the second pass.)

- In pass 1, you need to specify an output format (with -f) that matches the output format you will use in pass 2.
- In pass 1, specify the audio codec used in pass 2; in many cases, -an in pass 1 will not work.

```
ffmpeg -y -i input -c:v libx264 -b:v 2600k -pass 1 -c:a aac -b:a 128k -f r
ffmpeg -i input -c:v libx264 -b:v 2600k -pass 2 -c:a aac -b:a 128k output
```

faststart for web video

You can add `-movflags +faststart` as an output option if your videos are going to be viewed in a browser. This will move some information to the beginning of your file and allow the video to begin playing before it is completely downloaded by the viewer. **It is not required if you are going to use a video service such as YouTube**.

Compatibility: Profiles and Levels

H264 has multiple profiles. Each profile uses a subset of the coding tools defined by the H.264 standard. The tools are algorithms or processes used for video coding and decoding. An encoder will compress video based on a specific profile, and this will define which tools the decoder must use in order to decompress the video. A decoder may support some profiles, while it does not support others. Each profile is intended to be useful to a class of applications.

If you want your videos to have the highest compatibility with ancient devices (e.g., old Android and iOS phones) use a `level 3 baseline` profile like so:

```
-profile:v baseline -level 3.0
```

This disables some advanced features but provides for better compatibility as encoders and decoders **must** support the baseline profile. This setting may increase the bit rate compared to what is needed to achieve the same quality in higher profiles.

For iOS devices look at the table below for the combination of profile and level you need to support your target devices. Again, the lower the level/profile combination the wider support you get but there may be times (especially when doing adaptive streaming with HSL or DASH) that you'll want to go for the higher profile/level combinations.

iOS Compatability (source)			
Profile	**Level**	**Devices**	**Options**
Baseline	3.0	All devices	`-profile:v baseline -level 3.0`
Baseline	3.1	iPhone 3G and later, iPod touch 2nd generation and later	`-profile:v baseline -level 3.1`
Main	3.1	iPad (all versions), Apple TV 2 and later, iPhone 4 and later	`-profile:v main -level 3.1`
Main	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4s and later	`-profile:v main -level 4.0`
High	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4s and later	`-profile:v high -level 4.0`
High	4.1	iPad 2 and later, iPhone 4s and later, iPhone 5c and later	`-profile:v high -level 4.1`
High	4.2	iPad Air and later, iPhone 5s and later	`-profile:v high -level 4.2`

Converting to H265 (AVI to H265)

Constant Rate Factor (CRF)

When working with AVC/H264 video, for the most part, we need to work with Constant Rate Factor (CRF).

CRF is the default quality (and rate control) setting for the x264 and x265 encoders. Values range between 0 and 51, where lower values would result in better quality and larger file sizes. Higher values mean more compression, but eventually, the video quality will noticeably degrade.

Pick the CRF values

Use this mode if you want to retain good visual quality and don't care about the exact bitrate or filesize of the encoded file. The mode works exactly the same as in x264, so please read the H.264 guide for more info.

As with x264, you need to make two choices:

- Choose a CRF. The default is 28, and it should visually correspond to alibx264 video at CRF 23, but result in about half the file size. Other than that, CRF works just like in x264
- Choose a preset. The default is medium. The preset determines how fast the encoding process will be – at the expense of compression efficiency. Put differently, if you choose ultrafast, the encoding process is going to run fast, but the file size will be larger when compared to medium. The visual quality will be the same. Valid presets are `ultrafast`, `superfast`, `veryfast`, `faster`, `fast`, `medium`, `slow`, `slower`, `veryslow` and `placebo`
- Choose a tune. By default, this is disabled, and it is generally not required to set a tune option. x265 supports the following -tune options: psnr, ssim, grain, zerolatency, fastdecode. For example:

```
ffmpeg -i input -c:v libx265 -crf 28 -c:a aac -b:a 128k output.mp4
```

Two-Pass Encoding

This process is very similar to the H264 two-pass encoding with some different parameters.

For two-pass, you need to run ffmpeg twice, with almost the same settings, except for:

- In pass 1 and 2, use the `-x265-params pass=1` and `-x265-params pass=2` options, respectively. For libx265, the `-pass` option (that you would use for libx264) is not applicable
- In pass 1, output to a null file descriptor, not an actual file. (This will generate a logfile that ffmpeg needs for the second pass)
- In pass 1, you need to specify an output format (with -f) that matches the

output format you will use in pass 2
- In pass 1, specify the audio codec used in pass 2; in many cases, -an in pass 1 will not work

The full H265 example looks like this:

```
ffmpeg -y -i input -c:v libx265 -b:v 2600k -x265-params pass=1 -c:a aac -b
ffmpeg -i input -c:v libx265 -b:v 2600k -x265-params pass=2 -c:a aac -b:a
```

As with CRF, choose the slowest -preset you can tolerate, and optionally apply a -tune setting. Note that when using faster presets with the same target bitrate, the resulting quality will be lower and vice-versa.

Passing Options

Generally, options are passed to x265 with the -x265-params argument. For fine-tuning the encoding process, you can pass any option that is listed in the x265 documentation. This is only good if you know exactly what you need to change.

Setting Profiles

Currently, ffmpeg does not support setting profiles with the `profile:v` option, as libx264 does. However, the profile options can be set manually, as shown in this Super User post.

Links and Resources

- FFmpeg and VP9 Encoding Guide
- FFmpeg and H.264 Encoding Guide
- FFmpeg and H.265 Encoding Guide
- CRF Guide (Constant Rate Factor in x264 and x265)
- Understanding Rate Control Modes (x264, x265, vpx)
- What is MPEG-DASH? (2011)
- MPEG DASH
- Guidelines for Implementation: DASH-IF Interoperability Points
- HTTP Live Streaming 2nd Edition

Once support for AV1 comes to FFMPEG I will recompile it to include AV1 support. It will, supposedly, provide better compression than HEVC/H265 and **not be encumbered by patents** to the level of H265 and H264.

For an idea of the licensing nightmare HEVC has created see this article in The Register

There are times when I dearly wish I had a GUI to do some of the work, particularly with feature-rich applications like FFMPEG but we don't always have the chance or the choice. While there are tools like IFFMPEG they are not as comprehensive as I'd like them to be so we need to learn at least the basic of the command line.

I've chosen to install FFMPEG via Homebrew rather than compile it directly via XCode. The command will install FFMPEG with the optional Open H264 support enabled.

```
brew install ffmpeg --with-openh264
```

Once it is installed you get the `ffmpeg` command available. It'll be the basis for what follows.

Basic use

These are some of the basic commands that I use.

The first command gathers information about the video:

```
ffmpeg -i tears_of_steel_1080p.mov
```

And the result will be something like this. Note that the streams descriptions have hard returns added for readability, your result will look different.

```
  libavutil      55. 78.100 / 55. 78.100
  libavcodec     57.107.100 / 57.107.100
  libavformat    57. 83.100 / 57. 83.100
  libavdevice    57. 10.100 / 57. 10.100
  libavfilter     6.107.100 /  6.107.100
  libavresample   3.  7.  0 /  3.  7.  0
  libswscale      4.  8.100 /  4.  8.100
  libswresample   2.  9.100 /  2.  9.100
  libpostproc    54.  7.100 / 54.  7.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'tears_of_steel_1080p.mov':
  Metadata:
    major_brand     : qt
    minor_version   : 512
    compatible_brands: qt
    encoder         : Lavf53.32.100
  Duration: 00:12:14.17, start: 0.000000, bitrate: 6361 kb/s
    Stream #0:0(eng): Video: h264 (Main) (avc1 / 0x31637661), yuv420p,
    1920x800 [SAR 1:1 DAR 12:5], 6162 kb/s, 24 fps, 24 tbr, 24 tbn, 48 tb
    Metadata:
      handler_name    : DataHandler
      encoder         : libx264
    Stream #0:1(eng): Audio: mp3 (.mp3 / 0x33706D2E), 44100 Hz, stereo, s1
    Metadata:
      handler_name    : DataHandler
```

Resizing the video

For this first example, we'll resize the video to (WxH) 640x480.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv -s 640x480 agent_327_480p.mp
```

Changing the video data rate

FFMPEG allows me to change the bit rate for both audio and video separately. In this example, I will only change the video bitrate to 2 megabits without changing the audio bitrate at all.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -b:v 2m \
  agent_327_480p.mp4
```

Changing the audio data rate

The two attributes that we want to work with for audio are `-b:a` and `-ab`. There is an older argument `-ar` that will accomplish the same thing but the recommendation is to use `-b:a` to be consistent with the video attribute.

The `-b:a` attribute controls the audio sampling frequency. In this example, I've set the value to 48000.

With `-ab` we get control of the audio bit rate (expressed in bits per second).

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -b:a 48000 -ab 120k  \
  agent_327_480p.mp4
```

Changing the aspect ratio of a video

The aspect ratio of an image describes the proportional relationship between its width and its height. It is commonly expressed as two numbers separated by a colon, as in 16:9.

In FFMPEG, this one is an easy one to change; use the `-aspect` flag and the aspect ratio than you want to use.

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -aspect 16:9 \
  agent_327_480p.mp4
```

Changing the frame rate of a video

Frame Rate is the frequency (rate) at which consecutive images called frames appear on a display. Values above 12 are perceived by humans as motion.

To change it use the `-r` value

```
ffmpeg -i Agent_327_Operation_Barbershop.mkv \
  -r 25 \
  agent_327_480p.mp4
```

Converting from one format to another

To make sure that support for all the codecs I want FFMPEG with support for `libvpx` and opus (the default audio format for VP9) using the following command:

```
brew reinstall ffmpeg --with-openh264 --with-x265 --with-tools --with-libv
```

This makes sure that all the codecs are installed and available for the following sections. **This is a one-time operation**. Once the codecs are installed you don't need to reinstall them again

Converting to VP9

Let's say that your client delivers a video in MP4 format and you need to deliver it as a high-quality VP9 video. Based on We'll look

- 1-pass average bitrate
- 2-pass average bitrate
- Constant quality
- Constant bitrate

1-pass average bitrate

The simplest way to encode VP9 is the simple variable bitrate (VBR) mode VP9 offers by default. This is also sometimes called "Average Bitrate" or "Target Bitrate". In this mode, it will simply try to reach the specified bit rate on average, e.g. 2 MBit/s.

```
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 \
-b:v 2M \
output-1-pass.webm
```

2-pass average bitrate

In order to create more efficient encodes when a particular target bitrate should be reached, you should choose the two-pass encoding.

For two-pass, you need to run ffmpeg twice as shown below; The differences between the two passes are:

- In pass 1 and 2, use the `-pass 1` and `-pass 2` options, respectively
- In pass 1, output to a null file descriptor, not an actual file. (This will generate a logfile that ffmpeg needs for the second pass)
- In pass 1, you need to specify an output format (with `-f`) that matches the output format you will use in pass 2
- In pass 1, specify the audio codec used in pass 2; in many cases, `-an` in pass 1 will not work
- In pass 1 specify a fast encode (`speed 4`) and in pass 2, a slower encode (`speed 1`). This should speed up the overall encode process

```
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 -b:v 2M -pass 1 \
-c:a libopus -speed 4 -f webm /dev/null && \
ffmpeg -i input.mp4 \
-c:v libvpx-vp9 -b:v 2M -pass 2 \
-c:a libopus -speed 1 output.webm
```

Converting to H264 (AVI to H264)

When working with AVC/H264 video, for the most part, we need to work with Constant Rate Factor (CRF).

CRF is the default quality (and rate control) setting for the x264 and x265 encoders. Values range between 0 and 51, where lower values would result in better quality and larger file sizes. Higher values mean more compression, but eventually, the video quality will suffer noticeably degrade.

Pick the CRF values

The range of the CRF scale is `0–51`, where 0 is lossless, 23 is the default, and 51 is worst quality possible (see the note below for the difference between 8 and 10-bit encoding and the CRF values). A subjectively sane CRF range is 17–28 with a default of 23. Consider 17 or 18 to be visually lossless or nearly so; it should look the same or nearly the same as the input but it isn’t technically lossless.

The range is exponential, so increasing the CRF value +6 results in roughly half the bitrate/file size, while -6 leads to roughly twice the bitrate.

Choose the highest CRF value that still provides an acceptable quality. If the output looks good, then try a higher value. If it looks bad, choose a lower value.

The `0–51` CRF quantizer scale mentioned on this section only applies to 8-bit x264. When compiled with 10-bit support, x264's quantizer scale is `0–63`. You can see what you are using by referring to the ffmpeg console output during encoding (`yuv420p` or similar for 8-bit, and `yuv420p10le` or similar for 10-bit). 8-bit is more common among distributors.

Presets

A preset is a set of predefined options that will provide a certain encoding speed to compression ratio. A slower preset will provide better compression. If you target a certain file size or constant bit rate, you will achieve better quality with a slower preset at the expense of a bigger file size.

Use the slowest preset that you have the patience to wait for. As with many things, you’ll have to test the presets to see which one works best for your project.

The available presets in descending order of speed are:

- ultrafast
- superfast
- veryfast
- faster
- fast
- medium – default preset

- slow
- slower
- veryslow

You can see a list of current presets with `-preset help`

CRF Example

This command encodes a video with good quality, using the `slow` preset to achieve better compression:

```
ffmpeg -i input.avi -c:v libx264 -preset slow -crf 22 -c:a copy output.mk
```

Note that in this example the audio stream of the input file is simply not re-encoded, we just copy the stream over to the output.

If you are encoding a set of videos that are similar, apply the same settings to all the videos: this will ensure that they will all have similar quality.

Two-Pass

Two-Pass encoding is more complicated but it works better when trying to target a specific file size with frame output quality being a secondary concern. For this example we'll use the following formula to calculate bitrate:

```
(200 MiB * 8192 [converts MiB to kBit]) / 600 seconds = ~2730 kBit/s tota
2730 - 128 kBit/s (desired audio bitrate) = 2602 kBit/s video bitrate
```

You can also forgo the bitrate calculation if you already know what final (average) bitrate you need.

For two-pass, you need to run ffmpeg twice, with almost the same settings, except for:

- In pass 1 and 2, use the -pass 1 and -pass 2 options, respectively.
- In pass 1, output to a null file descriptor, not an actual file. (This will generate a logfile that ffmpeg needs for the second pass.)
- In pass 1, you need to specify an output format (with -f) that matches the output format you will use in pass 2.

- In pass 1, specify the audio codec used in pass 2; in many cases, -an in pass 1 will not work.

```
ffmpeg -y -i input -c:v libx264 -b:v 2600k -pass 1 -c:a aac -b:a 128k -f r
ffmpeg -i input -c:v libx264 -b:v 2600k -pass 2 -c:a aac -b:a 128k output
```

faststart for web video

You can add `-movflags +faststart` as an output option if your videos are going to be viewed in a browser. This will move some information to the beginning of your file and allow the video to begin playing before it is completely downloaded by the viewer. **It is not required if you are going to use a video service such as YouTube**.

Compatibility: Profiles and Levels

H264 has multiple profiles. Each profile uses a subset of the coding tools defined by the H.264 standard. The tools are algorithms or processes used for video coding and decoding. An encoder will compress video based on a specific profile, and this will define which tools the decoder must use in order to decompress the video. A decoder may support some profiles, while it does not support others. Each profile is intended to be useful to a class of applications.

If you want your videos to have the highest compatibility with ancient devices (e.g., old Android and iOS phones) use a `level 3 baseline` profile like so:

```
-profile:v baseline -level 3.0
```

This disables some advanced features but provides for better compatibility as encoders and decoders **must** support the baseline profile. This setting may increase the bit rate compared to what is needed to achieve the same quality in higher profiles.

For iOS devices look at the table below for the combination of profile and level you need to support your target devices. Again, the lower the level/profile combination the wider support you get but there may be times (especially when doing adaptive streaming with HSL or DASH) that you'll want to go for the higher profile/level combinations.

iOS Compatability (source)			
Profile	**Level**	**Devices**	**Options**

Baseline	3.0	All devices	`-profile:v baseline`
			`-level 3.0`

Baseline	3.1	iPhone 3G and later, iPod touch 2nd generation and later	`-profile:v baseline`
			`-level 3.1`

Main	3.1	iPad (all versions), Apple TV 2 and later, iPhone 4 and later	`-profile:v main`
			`-level 3.1`

Main	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4s and later	`-profile:v main`
			`-level 4.0`

High	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4s and later	`-profile:v high`
			`-level 4.0`

High	4.1	iPad 2 and later, iPhone 4s and later, iPhone 5c and later	`-profile:v high`
			`-level 4.1`

High	4.2	iPad Air and later, iPhone 5s and later	`-profile:v high`
			`-level 4.2`

Converting to H265 (AVI to H265)

Constant Rate Factor (CRF)

When working with AVC/H264 video, for the most part, we need to work with Constant Rate Factor (CRF).

CRF is the default quality (and rate control) setting for the x264 and x265 encoders. Values range between 0 and 51, where lower values would result in better quality and larger file sizes. Higher values mean more compression, but eventually, the video quality will noticeably degrade.

Pick the CRF values

Use this mode if you want to retain good visual quality and don't care about the exact bitrate or filesize of the encoded file. The mode works exactly the same as in x264, so please read the H.264 guide for more info.

As with x264, you need to make two choices:

- Choose a CRF. The default is 28, and it should visually correspond to alibx264 video at CRF 23, but result in about half the file size. Other than that, CRF works just like in x264
- Choose a preset. The default is medium. The preset determines how fast the encoding process will be – at the expense of compression efficiency. Put differently, if you choose ultrafast, the encoding process is going to run fast, but the file size will be larger when compared to medium. The visual quality will be the same. Valid presets are `ultrafast`, `superfast`, `veryfast`, `faster`, `fast`, `medium`, `slow`, `slower`, `veryslow` and `placebo`
- Choose a tune. By default, this is disabled, and it is generally not required to set a tune option. x265 supports the following -tune options: psnr, ssim, grain, zerolatency, fastdecode. For example:

```
ffmpeg -i input -c:v libx265 -crf 28 -c:a aac -b:a 128k output.mp4
```

Two-Pass Encoding

This process is very similar to the H264 two-pass encoding with some different parameters.

For two-pass, you need to run ffmpeg twice, with almost the same settings, except for:

- In pass 1 and 2, use the `-x265-params pass=1` and `-x265-params pass=2` options, respectively. For libx265, the `-pass` option (that you would use for libx264) is not applicable
- In pass 1, output to a null file descriptor, not an actual file. (This will generate a logfile that ffmpeg needs for the second pass)
- In pass 1, you need to specify an output format (with -f) that matches the output format you will use in pass 2
- In pass 1, specify the audio codec used in pass 2; in many cases, -an in pass 1 will not work

The full H265 example looks like this:

```
ffmpeg -y -i input -c:v libx265 -b:v 2600k -x265-params pass=1 -c:a aac -b
ffmpeg -i input -c:v libx265 -b:v 2600k -x265-params pass=2 -c:a aac -b:a
```

As with CRF, choose the slowest -preset you can tolerate, and optionally apply a -tune setting. Note that when using faster presets with the same target bitrate, the resulting quality will be lower and vice-versa.

Passing Options

Generally, options are passed to x265 with the -x265-params argument. For fine-tuning the encoding process, you can pass any option that is listed in the x265 documentation. This is only good if you know exactly what you need to change.

Setting Profiles

Currently, ffmpeg does not support setting profiles with the `profile:v` option, as libx264 does. However, the profile options can be set manually, as shown in this Super User post.

Links and Resources

- FFmpeg and VP9 Encoding Guide
- FFmpeg and H.264 Encoding Guide
- FFmpeg and H.265 Encoding Guide
- CRF Guide (Constant Rate Factor in x264 and x265)
- Understanding Rate Control Modes (x264, x265, vpx)
- What is MPEG-DASH? (2011)
- MPEG DASH
- Guidelines for Implementation: DASH-IF Interoperability Points
- HTTP Live Streaming 2nd Edition