H.264 Video Encoding Guide

The goal of this H.264 video encoding guide is to inform new users how to create a high-quality video using the encoder x264.

There are two rate control modes that are usually suggested for general use: Constant Rate Factor (CRF) or Two-Pass ABR. Rate control decides how many bits will be used for each frame. This will determine the file size and also how quality is distributed. To know more about what the different rate control modes do see this post.

Contents

If you need help compiling and installing libx264 see one of our compiling guides.

Constant Rate Factor (CRF)

Use this mode if you want to keep the best quality and don't care about the file size.

This method allows the encoder to attempt to achieve a certain output quality for the whole file when output file size is of less importance. This provides maximum compression efficiency with a single pass. By adjusting the so-called quantizer for each frame, it gets the bitrate it needs to keep the requested quality level. The downside is that you can't tell it to get a specific filesize or not go over a specific size or bitrate, which means that this method is not recommended for encoding videos for streaming.

1. Choose a CRF value

The range of the CRF scale is 0–51, where 0 is lossless, 23 is the default, and 51 is worst quality possible. A lower value generally leads to higher quality, and a subjectively sane range is 17–28. 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.

> **Note:** The 0–51 CRF quantizer scale mentioned on this page 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.

2. Choose a preset and tune

Preset

A preset is a collection of options that will provide a certain encoding speed to compression ratio. A slower preset will provide better compression (compression is quality per filesize). This means that, for example, if you target a certain file size or constant bit rate, you will achieve better quality with a slower preset. Similarly, for constant quality encoding, you will simply save bitrate by choosing a slower preset.

Use the slowest preset that you have patience for. The available presets in descending order of speed are:

- `ultrafast`
- `superfast`
- `veryfast`
- `faster`
- `fast`
- `medium` – default preset
- `slow`
- `slower`
- `veryslow`
- `placebo` – ignore this as it is not useful (see FAQ)

You can see a list of current presets with `-preset help` (see example below). If you have the `x264` binary installed, you can also see the exact settings these presets apply by running `x264 --fullhelp`.

Tune

You can optionally use `-tune` to change settings based upon the specifics of your input. Current tunings include:

- `film` – use for high quality movie content; lowers deblocking
- `animation` – good for cartoons; uses higher deblocking and more reference frames
- `grain` – preserves the grain structure in old, grainy film material
- `stillimage` – good for slideshow-like content
- `fastdecode` – allows faster decoding by disabling certain filters
- `zerolatency` – good for fast encoding and low-latency streaming
- `psnr` – ignore this as it is only used for codec development
- `ssim` – ignore this as it is only used for codec development

For example, if your input is animation then use the `animation` tuning, or if you want to preserve grain in a film then use the `grain` tuning. If you are unsure of what to use or your input does not match any of tunings then omit the `-tune` option. You can see a list of current tunings with `-tune help`, and what settings they apply with `x264 --fullhelp`.

Profile

Another optional setting is `-profile:v` which will limit the output to a specific H.264 profile. Omit this unless your target device only supports a certain profile (see Compatibility). Current profiles include: `baseline`, `main`, `high`, `high10`, `high422`, `high444`. Note that usage of `-profile:v` is incompatible with lossless encoding.

List presets and tunes

To list all possible internal presets and tunes:

```
ffmpeg -hide_banner -f lavfi -i nullsrc -c:v libx264 -preset help -
```

> **Note:** Windows users may need to use `NUL` instead of `-` as the output.

CRF Example

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

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

Note that in this example the audio stream of the input file is simply stream copied over to the output and not re-encoded.

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

Use this method if you are targeting a specific output file size, and if output quality from frame to frame is of less importance. This is best explained with an example. Your video is 10 minutes (600 seconds) long and an output of 200 MiB is desired. Since `bitrate = file size / duration`:

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

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

Two-Pass Example

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.

For example:

```
ffmpeg -y -i input -c:v libx264 -b:v 2600k -pass 1 -c:a aac -b:a 12
ffmpeg -i input -c:v libx264 -b:v 2600k -pass 2 -c:a aac -b:a 128k
```

> **Note:** Windows users should use `NUL` instead of `/dev/null` and `^` instead of `\`.

As with CRF, choose the slowest `-preset` you can tolerate, and optionally apply a `-tune` setting and `-profile:v`.

Lossless H.264

You can use `-crf 0` to create a lossless video. Two useful presets for this are `ultrafast` or `veryslow` since either a fast encoding speed or best compression are usually the most important factors.

Fast encoding example:

```
ffmpeg -i input -c:v libx264 -preset ultrafast -crf 0 output.mkv
```

Best compression example:

```
ffmpeg -i input -c:v libx264 -preset veryslow -crf 0 output.mkv
```

Note that lossless output files will likely be huge, and most non-FFmpeg based players will not be able to decode lossless. Therefore, if compatibility or file size are an issue, you should not use

lossless.

> **Tip:** If you're looking for an output that is roughly "visually lossless" but not technically lossless, use a `-crf` value of around 17 or 18 (you'll have to experiment to see which value is acceptable for you). It will likely be indistinguishable from the source and not result in a huge, possibly incompatible file like true lossless mode.

Overwriting default preset settings

While `-preset` chooses the best possible settings for you, you can overwrite these with the `x264-params` option, or by using the libx264 private options (see `ffmpeg -h encoder=libx264`). This is not recommended unless you know what you are doing. The presets were created by the x264 developers and tweaking values to get a better output is usually a waste of time.

Example:

```
ffmpeg -i input -c:v libx264 -preset slow -crf 22 -x264-params keyi
```

> **Warning:** Do not use the option `x264opts`, as it will eventually be removed. Use `x264-params` instead.

Additional Information & Tips

CBR (Constant Bit Rate)

There is no native or true CBR mode, but you can "simulate" a constant bit rate setting by tuning the parameters of a one-pass average bitrate encode:

```
ffmpeg -i input.mp4 -c:v libx264 -x264-params "nal-hrd=cbr" -b:v 1M
```

In the above example, `-bufsize` is the "rate control buffer", so it will enforce your requested "average" (1 MBit/s in this case) across each 2 MBit worth of video. Here it is assumed that the receiver / player will buffer that much data, meaning that a fluctuation within that range is acceptable.

CBR encodes are usually inefficient if the video is easy to encode (e.g., empty or black frames).

Constained encoding (VBV / maximum bit rate)

Use this mode if you want to constrain the maximum bitrate used, or keep the stream's bitrate within certain bounds. This is particularly useful for online streaming, where the client expects a certain average bitrate, but you still want the encoder to adjust the bitrate per-frame.

You can use `-crf` or `-b:v` with a maximum bit rate by specifying both `-maxrate` and `-bufsize`:

```
ffmpeg -i input -c:v libx264 -crf 23 -maxrate 1M -bufsize 2M output
```

This will effectively "target" `-crf 23`, but if the output were to exceed 1 MBit/s, the encoder would increase the CRF to prevent bitrate spikes. However, be aware that `libx264` does not

strictly control the maximum bit rate as you specified (the maximum bit rate may be well over 1M for the above file). To reach a perfect maximum bit rate, use two-pass.

In another example, instead of using constant quality (CRF) as a target, the average bitrate is set. A two-pass approach is preferred here:

```
ffmpeg -i input -c:v libx264 -b:v 1M -maxrate 1M -bufsize 2M -pass
ffmpeg -i input -c:v libx264 -b:v 1M -maxrate 1M -bufsize 2M -pass
```

Low Latency

x264 offers a `-tune zerolatency` option for low latency streaming.

Compatibility

All devices

If you want your videos to have highest compatibility with ancient devices (e.g., old Android phones):

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

This disables some advanced features but provides for better compatibility. Typically you may not need this setting (and therefore avoid using `-profile:v` and `-level`), but if you do use this setting it may increase the bit rate compared to what is needed to achieve the same quality in higher profiles.

iOS

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`

QuickTime

QuickTime only supports YUV planar color space with 4:2:0 chroma subsampling (use `-vf format=yuv420p` or `-pix_fmt yuv420p`) for H.264 video.

Blu-ray

See Authoring a professional Blu-ray Disc with x264.

Pre-testing your settings

Encode a random section instead of the whole video with the `-ss` and `-t`/`-to` options to quickly get a general idea of what the output will look like.

- `-ss`: Offset time from beginning. Value can be in seconds or HH:MM:SS format.
- `-t`: Duration. Value can be in seconds or HH:MM:SS format.
- `-to`: Stop writing the output at specified position. Value can be in seconds or HH:MM:SS format.

`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.

Custom preset file

Refer to the `-vpre` output option in the documentation.

FAQ

Will two-pass provide a better quality than CRF?

No, though it does allow you to target a file size more accurately.

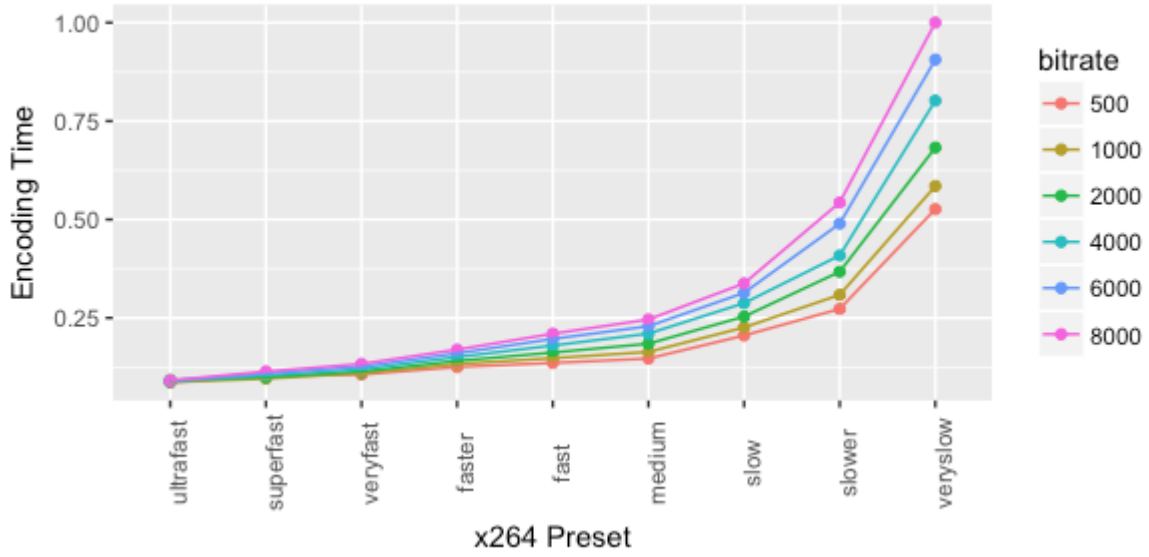
Why is `placebo` a waste of time?

It helps at most ~1% in terms of quality, compared to the `veryslow` preset at the cost of a much higher encoding time. It's diminishing returns: `veryslow` helps about 3% compared to the `slower` preset, `slower` helps about 5% compared to the `slow` preset, and `slow` helps about 5-10% compared to the `medium` preset.

How do the different presets influence encoding time?

This depends on the source material, the target bitrate, and your hardware configuration. In general, the higher the bitrate, the more time needed for encoding.

Here is an example that shows the (normalized) encoding time for a two-pass encode of a 1080p video:



Going from `medium` to `slow`, the time needed increases by about 40%. Going to `slower` instead would result in about 100% more time needed (i.e. it will take twice as long). Compared to `medium`, `veryslow` requires 280% of the original encoding time, with only minimal improvements over `slower` in terms of quality.

Using `fast` saves about 10% encoding time, `faster` 25%. `ultrafast` will save 55% at the expense of much lower quality.

Why doesn't my lossless output look lossless?

Blame the RGB to YUV color space conversion. If you convert with `-pix_fmt yuv444p` it should still be lossless (which is the default now).

Will a graphics card make x264 encode faster?

For x264 specifically, probably not. x264 supports OpenCL for some lookahead operations.

See HWAccelIntro for information on supported hardware H.264 encoders and decoders.

Encoding for dumb players

You may need to use `-vf format=yuv420p` (or the alias `-pix_fmt yuv420p`) for your output to work in QuickTime and most other players. These players only support the YUV planar color space with 4:2:0 chroma subsampling for H.264 video. Otherwise, depending on your source, `ffmpeg` may output to a pixel format that may be incompatible with these players.

Additional Resources

- Constant Rate Factor Guide
- Understanding Rate Control Modes

Вложения (1)

Последнее изменение: 01 нояб. 2017 г., 2:53:26

t a g s

avc h.264 x264