H.264
FOR THE REST OF US

BY
KUSH AMERASINGHE

A layman's guide to video compression and F4V, an explanation of F4V-specific Adobe® Media Encoder and Adobe® Flash® Media Encoding Server settings, and a case study of estimating an acceptable bit rate for a given video.

Adobe

Introduction

"Video compression is like packing suitcases"

H.264 is taking the web video world by storm. This relatively new technology is as intriguing as it is mystifying. As a person who didn't have the patience to understand all the intricacies of how this marvel actually works, yet who was keen to get the best it has to offer, I had my own awkward share of learning experiences (that made me feel like a teenager again). But, seeing how hard it is to get the rundown on this subject (at the time of this writing), I thought there might be other people out there who might benefit if I shared what I have learned so far.

The aim of this text is not to go into a comprehensive explanation of how H.264 works or all its possible applications. Rather, I'll try to explain the underlying parameters in a less less technical fashion so that the average web video publisher can make more informed decisions when choosing various options and determining the optimal values to critical parameters of H.264 specifically applied to Adobe® Flash® Player.

The main components of this text include:

- A layman's guide to video compression and F4V
- Explanation of F4V-specific Adobe Media Encoder and Adobe Flash Media Encoding Server settings
- A case study of estimating an acceptable bit rate for a given video.

The versatility of H.264

H.264 is a highly sophisticated, yet versatile technology intended to serve a wide variety of applications ranging from highly compressed, low-frame-size teleconferencing to large-format, cinema-quality file encoding. Using a technology with such a vast usage range for the relatively narrow field of web-based video (which is constantly being redefined regarding where in the quality spectrum it falls as Internet bandwidth and technology adaptation improves) can be challenging. The degree of control H.264 offers coupled with the complexity of the compression methods themselves can be overwhelming to newcomers and seasoned web video producers alike.

The suitcase metaphor

I like to think of video compression as somewhat similar to packing suitcases for a trip. I hope this is something most of you can relate to. Even if you are not a globe-trotter, at some point you must have faced with the challenge of getting the most amount of cloths you can carry in a given suitcase while preserving the best condition of the cloths when you get to your destination. I will keep coming back to this analogy to describe several aspects of video compression where applicable.

H-dot-who?

Codecs

A *codec* is a particular technology or method used to compress an electronic signal, such as a video or audio recording. The various codecs are often simply different ways of doing the same thing, yet one codec may have no direct relation to another. Each codec has its advantages and disadvantages. Usually newer codecs provide better efficiency and quality, while older codecs reach a wider audience. The most commonly used video codecs to compress video for Adobe Flash are H.264, On2 VP6, and Sorenson Spark. H.264 was introduced in Adobe Flash Player 9.0.r115. On2 VP6 was the older codec introduced in Flash Player 8. Sorenson Spark is the oldest of the codecs supported in Flash Player, introduced in Flash Player 6.

Compatibility

At the time of this writing, some version of Flash Player was installed in approximately 98% of the computers connected to the Internet in the world. Out of this, the proportion of users still using Flash Player 6 or older is negligible in most geographies. The adoption of new Flash Player versions capable of playing the latest H.264 codec is about 90% of this user base. This makes H.264-encoded video for Flash the most attractive option for general video distribution applications.

"The vast majority of Internet video consumers have already adopted the new versions of Flash Player capable of playing H.264 video"

Choosing a codec

There are other factors to consider when choosing a codec. While some users have the software capability to play back H.264, their practical CPU capabilities and available network bandwidth, along with the particular settings (or range of settings in case of multi-bit encoding) actually used to serve the videos, could greatly affect the quality of the experience these users receive. If your target audience is a highly specialized group (say, from a particular country or region, age group, or other such group), the proportion of such users in such a group may (or may not) be different from that in the general audience. For example, the relative higher quality-to-data rate ratio of H.264 generally comes at the price of relatively higher demands of processing power; such processing power may not be affordable by the audience for which you intend your video.

Also, since codecs are simply different ways of compressing the signal, rather than a linear series of incrementally improving specifications, some methods may better suit specific needs than others. For example, an FLV file encoded using On2 VP6 can contain an alpha channel that could be used to create interactive transparent video to be used in a Flash application, while H.264 in Flash currently doesn't support alpha channels.

Yet, I believe that H.264 is the overall winner so far in terms of its advantages and the ability to efficiently distribute high-quality, large-format video (even compared to current competing codecs not available in Adobe Flash Player). For this reason, I will focus mainly on the use of this technology specifically applied to Flash Player.

So where would the suitcase story fit in to the codec aspect of the compression world? At first, I was inclined to think of it as your choice for what brand of case to use. Then I realized the variety of factors involved in the choice of codec is almost as diverse as in the decision-making process involved in choosing how to go on your trip to begin with—by plane, ship, train, and so on. Each has its own cost, time, effort, and accessibility factors that depend a lot on your particular destination. Just as some destinations have multiple, sensible options to get there, some simply won't be reachable with certain modes of travel. Just like some modes of travel invented later in history made it more practical to reach certain destinations, certain video applications are only possible after certain codecs were invented!

Containers

It is important to understand what compression algorithms have to put up with (aside from the nagging roommate?)

It is important to remember, however, that video data in a given video codec could be delivered in many different types of containers or "wrapper" formats along with other forms of data (for example, the audio track in an audio codec, metadata, and so on). When considering the ability of a certain device to interpret a certain type of video, you should make sure it supports the specific codec(s) + container combination for it to be fully compatible.

Compression is not mean, just misunderstood!

Many people think compression is a nasty process that completely degrades their precious video creation into something that is much less of what it was originally. You can't really blame them, because it is quite true. Yet if you take some time to see the world from a compressor's point of view, you not only will start to appreciate the wonders and great practical qualities compression can add to your video publishing process, you will actually get along well with compression and be good friends! H.264 is one of the most complicated codecs in wide use today, and it can be particularly challenging to produce. Yet, given what it can achieve, the extra effort to get to a basic understanding of the philosophies behind how it behaves could make you a better compressionist.

Before you begin to think about how to get the best results from a particular combination of compression settings, consider factors that make it more challenging or easier for the compression algorithm (method) itself.

Some of this is obvious. For example, the longer the duration of a video, the more data is needed to transmit that video to the user (just like how you would need to carry more clothes for a longer vacation). Similarly, the larger the frame size or resolution of the video, the higher the amount of data it takes to store or transport (a large adult's wardrobe for the same duration of stay takes much more space than that of a small child).

These temporal and spatial dimensions of a video are key aspects on which the H.264 compression focuses to reproduce the closest possible video playback to the original file with the smallest possible amount of data (more about this later).

Other factors also dictate how compression-friendly the video content is. The nature of the content itself can make a video much easier to compress while maintaining reasonable quality, or too difficult to compress adequately. (Think of how certain articles of clothing can easily fold into thin layers, while some are impossible to fold or press without ruining their pristine quality—say, faded T-shirts versus suits and hats.)

Another crucial factor that influences compression efficiency is how dynamic or static the content of the video is in terms of subject motion and/or camera motion. This is somewhat independent of the frame rate (how many frames are played per second), since H.264

has sophisticated ways to detect and efficiently pack repetitive or redundant frames, even though reducing the frame rate can reduce the amount of information that needs to be packed. (A vacation with a busy agenda that involves different activities needing several changes of clothes per day demands more clothes than one where you only need one set of clothes each day.)

The balancing act

I think that there are three major axes of compromise that you have to balance carefully to achieve the best encoding experience for you and the best playback experience for your audience.

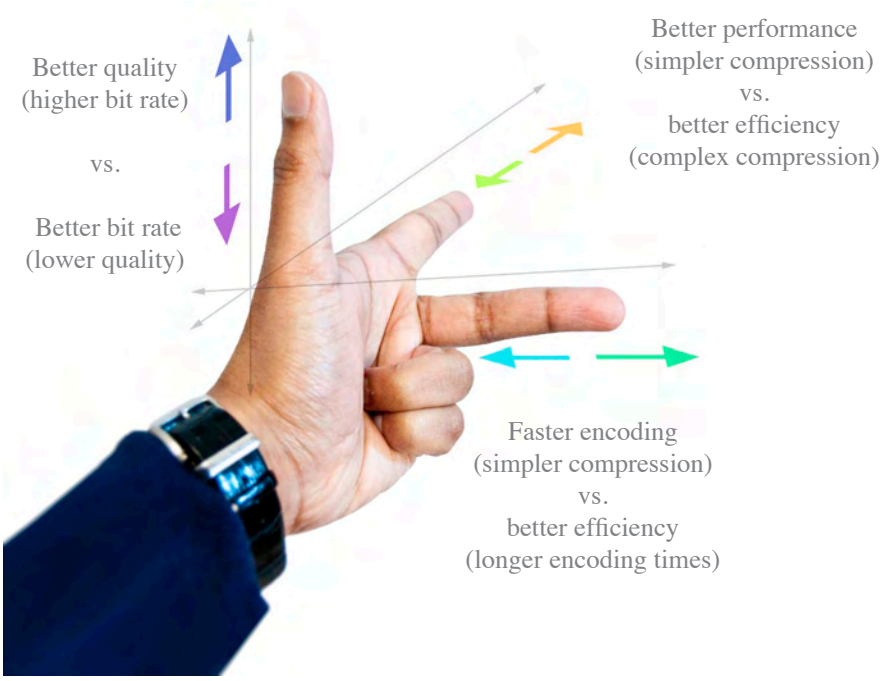


Figure 1 : The three dimensions of compression considerations. There is no right or wrong, but rather, give and take when it comes to deciding in which quadrant of the compression considerations chart you want to target.

Quality versus data rate

The first is the balance between quality and data rate. Those of you who are old enough to remember the days when the Internet started to use still images for the first time (for those of you who are not—yes, there was a time when the Internet was text-only) also remember that one of the great dilemmas was to compress the pictures just right so they would load reasonably fast without looking completely ruined. This is still a good practice despite the wider adoption of broadband. By now, both web developers and website users have an implicit understanding of where this balance lies. Since large-format video on the web is relatively new, it is not surprising that web weaving veterans feel a sense of "deja vu."

Similar to still images, in the video world, increasing the data rate during the compression phase allows you to pack higher (humanly perceived) visual quality into a given video. However, the increased data rate could either degrade or sometimes deny playback due to bandwidth limitations of the target user base. Meanwhile, reducing the data rate limits the video quality; past a certain point, it degrades the video to the point of being unrecognizable, despite smoother playback and/or load speeds—just as trying to cram too many clothes into a small suitcase can crumple and ruin them (especially if you take drastic measures such as those discussed later on) while bigger suitcases cost more and sometimes maybe

Figure 2 : Compression artifacts. When you compress something too much, the effects of compression start to become obvious!

banned if they don't fit size or weight criteria. (Multiple bit rate hosting allows you to serve the same video in a range of options transparent to the user, which could solve this issue in cases where your target audience has a wide range of bandwidths.) Note also that the encoded data rate impacts the amount of data served from a website, which in turn can effect hosting costs.

Efficiency versus playback performance

Complex compression methods such as H.264 could take up a significant amount of processing power compared to simpler methods when decoding during playback. Yet, using more complex compression methods can yield better quality at the same data rate.

So, the second axis of compromise is performance versus efficiency, meaning you could gain greater efficiency for getting more quality out of lower data rates if you are willing to expend more processing power and vice versa (somewhat independently in the encoding and the decoding sides of the equation). This is an important point to remember, since H.264 has many parameters that deal with this aspect, and understanding this philosophy of efficiency versus CPU overhead is crucial to making sense out of the flexibility these settings offer.

Compression efficiency versus encoding speed

While there is some correlation between added decoding complexity and corresponding encoding processing power demands, in general, you spend more time encoding than decoding. This makes sense in most cases, where playback needs to be in real time while

encoding happens on your own terms (except in live streaming). Not all extra measures that increase encoding time in return for better bit-rate-to-quality ratio result in harder-to-decode video. If you can afford it, you should generally spend as much time as it takes to get the best possible compression efficiency you can during encoding. Yet, there can be situations where certain features only give marginal or virtually no noticeable improvement to the visible quality for a given bit rate while making encode times significantly longer. In those cases, you should carefully evaluate the balance between practicality and end results.

Understanding the performance balance

Balancing those last two axes are similar to how, if you spend more time folding and organizing your clothes, you could fit more things into a given suitcase, while a quick, careless dumping would save time at the cost of space efficiency. Further, if you take extra steps to make your clothes pack better, such as steam pressing them one by one, you could achieve even better results at the cost of consuming more time. You can take it a step further by using some sort of vacuum shrink-wrapping process to make the clothes even more compact. This would take even more time and money. However, ironing the clothes will increase the time it takes to pack, but won't really have any impact on the amount of time it takes to unpack once you get to your destination; while shrink-wrapping clothes in plastic will also affect how much time and effort it takes to unpack. This is also true in the world of H.264, where certain settings would make a video take longer to encode, but the playback would not be impacted, while there are certain other settings that significantly impact playback performance due to increased overhead while decoding.

The losses and gains of lossy compression

There are two major groups of data compression in general: lossless compression and lossy compression. Lossless compression is when the process of compressing and then decompressing results in a file identical to the original. In other words, the data is rearranged and processed in such a way that it takes less spaces yet all the information is still available when it is reconstructed. (An example of lossless compression is a ZIP file—your file is the same when you unzip it.) In lossy compression, however, the data is further reduced in such a manner that it takes significantly less space than lossless compression alone, simply by throwing away some or most of the information in the original file.

The idea behind lossy compression is to do it in such a way that the discarded missing information will not be obvious to most viewers (especially if they don't have anything with which to compare it), yet save a great deal of data volume during transmission. When this is pushed too far, however, the effects of such destructive reduction do become obvious to even the untrained eye. These unwanted side effects due to over-compression are called *compression artifacts*.

To make it more interesting, the compression algorithm itself is not humanly aware when these artifacts become obvious to the point where further reduction of data does more harm than good. Similarly, the algorithm is not aware whether there is any added value in increasing the quality further at the cost of more data, since the human eye may not pick up the added detail or fidelity to the original. Even worse, this level of acceptable or excellent quality is highly subjective, not only from application to application (for example, video conferencing versus watching movies), but also among target audiences and even individuals with different expectation levels. This is the point where video compression leans more towards being an art than a science.

Figure 3 : Lossy compression is destructive. Packing your clothes the lossy compression way means you won't have the same clothes when you get to your destination!

This is also the point where the suitcase metaphor becomes rather bizarre. Yet for the sake of consistency, imagine you decide to remove some of the bulky buttons that are getting in the way of neatly folding a certain article of clothing. To be even more drastic, you may decide to cut off the long sleeves of a shirt with a pair of scissors to make it less of a burden on your load, yet it is still wearable (at least in theory). In either case, note how the resulting clothes you would unpack at your destination are not the same as the ones you packed; and the change is permanent, just as there is no coming back from a lossy-compressed file to the original. Unlike a desperate packer who still has some sense left in his or her mind about keeping the clothes still usable when unpacked, the completely mindless computer compression methods would not know where to stop and may go so far as to unweave a skirt into a ball of yarn that could, in theory, get to the destination taking less space, yet have no or little use once it gets there!

Even the slightest changes that any lossy compression makes to a video could result in problems later on when it is compressed again. For this reason, you should always use the highest possible quality file available all the way up to the last output stage. (Adobe native format editing tools make it quite practical.) Even a video that seems to be high in visible quality or looks practically identical to the original will have missing information and unusual hidden patterns and properties not so obvious to the naked eye when compressed. Therefore, given the choice, you should always opt for the least processed file, even if the compressed file "looks" as if there is no loss in quality, as problems that lie hidden to the human eye may be amplified in the recompression stage.

Obsessive compressive disorder

Elaborate compression schemes such as H.264 shine at what they do because of how obsessed they are by nature to get the visual quality across with the least amount of data possible. Another notable aspect of H.264's compression scheme is how it uses references to other areas of the video clip to define a new area without having to redefine it from scratch.

For example, the amount of data it takes to define an image of a car that is moving across the screen from one frame to the other with a new set of data for each frame is much higher compared to defining it once and reusing that data to say, "OK, put that same car a little towards the right in this frame." Then what about the area that used to be covered behind the car that is now visible because the car moved? No problem: "See how it looked like before the car entered the frame to begin with," and so on.

This level of obsession to reduce space by going to such extremes can only be illustrated by a suitcase packer who is desperate to save space to the point of simply omitting certain items of clothing by writing a note on how to recreate a certain outfit, such as "buy another pair of pants that is this style yet the same color as that jacket" or packing only one of each sock instead of the whole pair, with a note saying, "Try to find another sock just like this one!"

This type of referencing can occur in both the spatial aspect of compressing individual frames and the temporal aspect of compressing motion over several frames.

Figure 4 : H.264 compression goes to great lengths to reduce what needs to go in the suitcase. Instead of packing both socks in a pair, why not carry only one sock with a note to make a copy of it when you get there?

Taming the knobs

They say power is nothing without control—and not understanding what all the controls are that the H.264 encoders allow you to set could sometimes leave you powerless and lost! Now that you have a basic idea of what goes on in the mysterious, yet enchanting world of H.264 compression, I'll run through the controls and settings for the most commonly used encoders offered by Adobe.

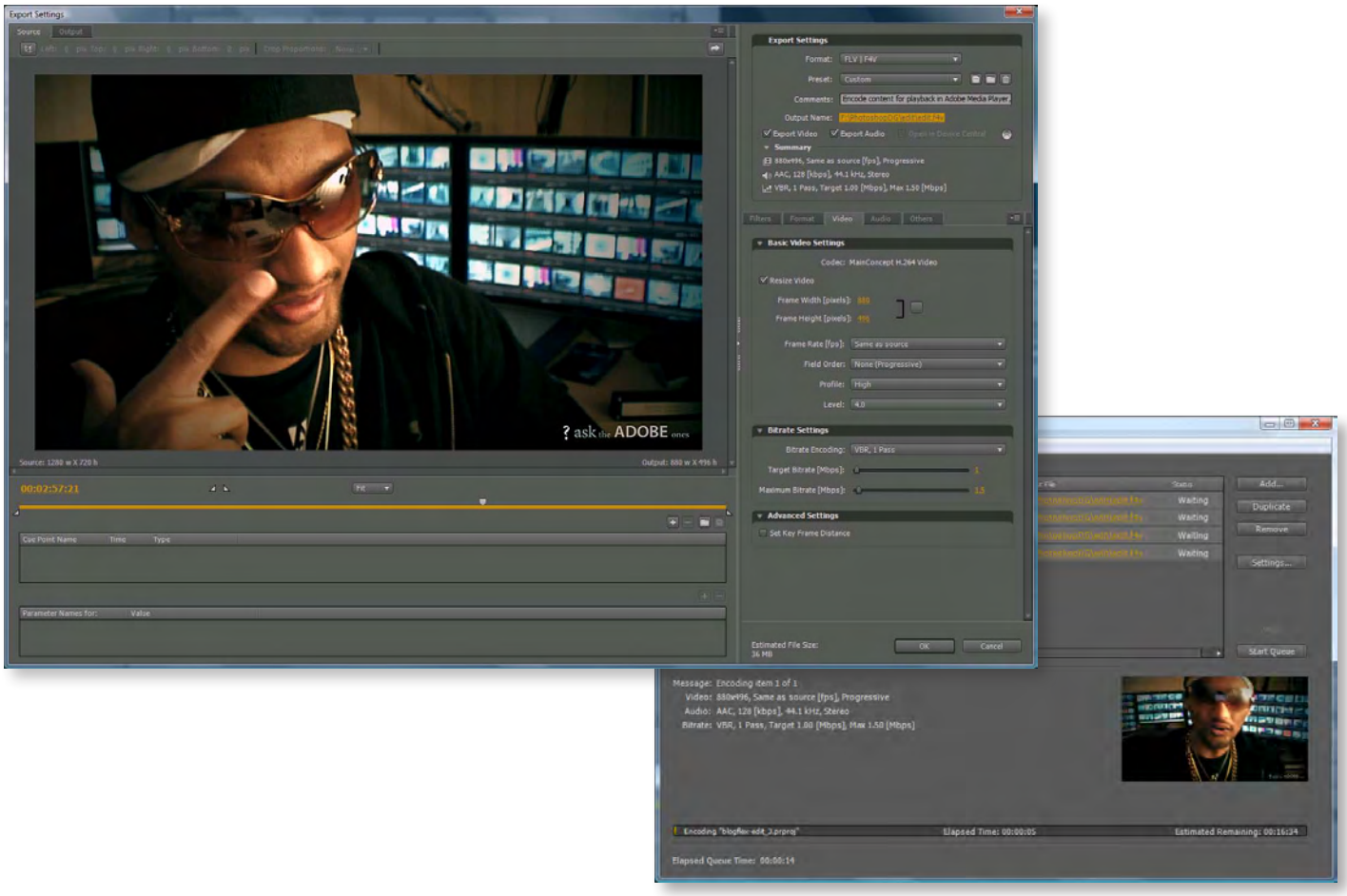


Figure 5 : Knowing what the knobs do helps Extra-H look good on the web.
AME about to compress a video of me playing my alter ego, "Extra-H," in "Ask the Adobe Ones."

Adobe® Media Encoder

Adobe Media Encoder (AME) is included as part of the Adobe Creative Suite 4 bundle of tools and will automatically launch when you try to output a video file format from a tool such as Adobe Premiere Pro CS4. While Adobe Media Encoder can encode many different formats, here the focus is on H.264-specific settings.

When you export media from Adobe Premiere Pro CS4, under Format Settings, you get a wide selection of major formats available (such as QuickTime, AVI, FLV and others). This format list is the same series of options you get when you add a raw file directly to the AME and set the format using the pop-up menu in the Format column. (You can set these settings for each entry in the AME queue independently.) This setting determines the file format or the container for the encoded media.

AME export settings

Format

You may notice that there are several H.264-related format options. If you are exporting to a standard format such as Blu-ray or iPod, select a template from the Format setting and then choose the appropriate subset of that format from the preset pop-up menu. For web delivery, the settings may need some tweaking, depending on your particular case.

While Adobe Flash Player supports a wide variety of standards, certain data the F4V container is capable of holding is not properly interpreted in iTunes and QuickTime. Hence there is a distinction between generic H.264-based files and those intended for Flash Player with the use of the F4V extension.

Presets

The presets are predefined combinations of encoding settings tailored to the format you selected. In case of the FLV/F4V format category, there are some presets using the FLV format, in which case you can choose either On2 VP6 or Sorenson Spark as the codec; and some designed for F4V, such as the Main Concept H.264 codec. These presets can be used either as-is or as a starting point for creating your own set of optimum encoding settings. Other than the codec setting itself, several other codec-specific options will be available for each file type (FLV and F4V).

In the Premiere Pro Export dialog box, you can tweak the settings of a given preset after selecting it. In the case of the AME queue, you can select a preset using the pop-up menu in the Preset column; then, if you click the underlined preset name itself, a dialog box similar to that of the Premiere Pro Media Export dialog box will appear allowing you to change any settings preloaded by the preset you selected. (Other than tweaking settings, you can also select a different format and/or preset, and then even go on to save the settings into your own preset name in this dialog box.)

Regardless of the preset you choose, you can switch between the FLV or F4V subformats in the Format tab before you go on to tweak video and audio settings on their respective tabs. For this H.264 discussion, look into the F4V-related settings.

AME video settings

Size

If the "Resize Video" option is selected, you can define the width and height of your video frames in pixels. If this option is not selected, the video will be encoded in its native size. While encoding at the native size will retain the maximum quality so long as you provide a high enough bit rate for it to be interpreted without noticeable artifacts, in practice you would probably want to reduce the video frame size in pixels to make it more practical for the H.264 compressor to pack it into a data stream or file small enough to be delivered over the average Internet connection. Generally, you should avoid resizing videos larger than their original size, since it makes no sense to increase the burden on the compression while degrading the initial precompression image quality.

The number of pixels in a video frame is a multiple of its frame width and height. Therefore, increasing either would have a dramatic effect on the image "area" or the number of total pixels. Increasing both would have an even greater impact. For example, you would think of a 640 × 480 video as being "twice as large" in comparison to a 320 × 240 video. In fact,

doubling both width and height *quadruples* the pixel count. This is an important point to remember when trying to come up with relative bit rates that work well for a certain size, since pixel count is a more sensible factor to consider than comparing either the width or height of a video. However, while in general the more pixels in your video, the higher the burden on the compression method, H.264 compression does employ complex algorithms that make this rule of thumb somewhat less applicable in certain cases. For example, a frame that has a small logo in the middle of a solid color background would compress equally as well as the same logo pasted on a much larger background—as long as the logo is not proportionately larger—despite the higher pixel count in the latter.

Other than the relative effect of how much of an impact the size of a video makes on the bit rate, the actual value of pixels can also play a role. This has to do with how the compression works in blocks or tiles rather than the entire image. For example, frame sizes that are evenly divisible by 16 are preferred over those that are not (divisible by 8 is the next best thing).

When resizing, it is important to keep the *image aspect ratio* (the proportion of width to height) the same as the original source to prevent aesthetically degrading (squashing or stretching) the image. The human eye is sensitive to such subtle changes in proportion, especially when objects that are expected to be proportionate (such as the wheels of a car) become stretched disproportionately. The most commonly used image aspect ratios in video are 4:3 and 16:9. Other aspect ratios exist in film, for example, yet when they are "made for TV," these images are often prepared with one of the basic screen aspect ratios.

Another size consideration is that, if your source video has a non-square pixel aspect ratio (not 1), the encoded video size needs to be translated into what it would be in a square pixel frame.

Pixels have an aspect ratio too?

While the *image aspect ratio* refers to the relative width to height proportion of an image (such as 16:9), the *pixel aspect ratio* refers to the proportional scale of each pixel in it. Certain traditional TV formats use non-square pixels. For example, with HDV (unlike full HDTV) images actually consist of 1440 × 1080 pixels. This image is then disproportionately stretched to a 1920 × 1080 screen area when played back on a HD-capable television.

When you specifically select the F4V format, there is no setting to define the pixel aspect ratio (unlike the case when defining generic H.264 settings). This is because Flash Player performs best when the video contains square pixels, or a pixel aspect ratio of 1. Videos with non-square pixels take additional scaling (stretching) steps at playback, and the quality/performance/bit-rate compromises are not worth such additional processing. Since the pixel aspect ratio is going to be set as 1 when you choose F4V, you should make the image size reflect that change if your original source had a non square pixel value. For example, 1440 × 1080 HDV should be 1920 × 1080, because 1080 × (16/9) = 1920; and a 720 × 576 PAL DV wide screen corresponds to 1024 × 576 in square pixels: 576 × (16/9) = 1024.

This is an exception to the rule about not scaling video size up, since the other option would be to keep the real pixel width and reduce the height to reflect the correct image aspect ratio, resulting in vertical detail loss.

It is best to make any changes to the video frame size only at the final encoding stage and keep the size the same as the native size all throughout the video editing process, using the native editing capabilities of the Adobe video tools.

Frame rate

The *frame rate* is how many still images are displayed in the video to give the illusion of moving images. The human eye can be fooled to perceive such motion to be convincing enough at around 24 frames per second (the typical film frame rate). PAL (common in Europe and some parts of Asia) uses 25fps, while NTSC standard (used in the US and Japan, for example) uses 29.97fps. As frame rates get lower, the motion seems jittery, especially if the subject changes location in the frame rapidly. However, early CD-ROM–based video clips used frame rates as low as 15 to 10fps to play back video, and some subject matter such as computer screen capture tutorials could be experienced reasonably well at frame rates as low as 5fps. In contrast, computer gaming–specific display devices boast large frame rates such as 120fps. Finally, there is a growing trend among video content creators to mimic a "cinematic feel" using 24fps.

Unless the source video was processed with a specialized in-between frame creation process such as the Premiere Pro Time Warp effect, generally you should never exceed the frame rate of the source video. In fact, the best results will be achieved if the frame rate is kept the same as your original source. Since this depends a lot on what sort of camera is used, it is important to know this information prior to your encoded frame rate decision. Just like frame size, the frame rate is best kept native throughout the editing process.

While in the ideal world, native frame rates give the best possible results, in practice the frame rate generally burdens the bit rate significantly. Again, the complexity of the temporal compression algorithms of H.264 makes it harder to draw a strictly linear relationship between frame rate and its impact on the bit rate. The nature of the video content can drastically change how well the video is compressed without noticeable artifacts. For example, a steady camera with a person simply talking on screen ("talking head video") is much easier to compress compared to an action movie sequence with rapidly changing shots, each containing fast motion. In the world of H.264, the frame rate should be thought of as more of a ceiling. Higher frame rates allow for smoother movements at the cost of lower quality-to-bit-rate economy, while reducing the frame rate reduces the burden, allowing more "breathing space" to pack more visual quality (compromising temporal quality).

If you do reduce the frame rate, try to do it in even steps such as halving to prevent any audio-video synchronization problems that could result in odd frame rate conversions.

Field order

The *field order* is yet another relic of the past that continues to over-complicate the digital video world. Unlike film, where each picture frame is flashed on the screen in such rapid succession that we do not notice the flicker from one to the next (an illusion of continuous movement), traditional television technology created a moving image by means of an electron beam that scanned continuous rows from top to bottom fast enough to be perceived by the naked eye as a full frame. To pack more motion information into each frame, every other row of "pixels" on the screen was scanned into a field (for example, scan the first, third, fifth, and so on to complete one field; when complete, return to the top and scan the second, fourth, sixth, and so on until the second field is complete). The two fields were thus "interlaced" to form a video frame. If the picture changed between these two scans during the recording process, the images on the odd and even rows, or fields, would be different within a single frame! To make matters worse, PAL and NTSC scan the fields in different order (odd/even, also called upper/lower). When these standards became digital (DV), the field order conventions changed again. For this reason, you should choose your field settings during the editing process using the Premiere Pro project presets for the standard you are working with.

However, when it comes to encoding for the web, the final output should be full frames without any split motion within each frame. If your original video source does indeed consist of fields rather than progressive frames, make sure it is *de-interlaced* before encoding it. Do not de-interlace video that is either already de-interlaced or didn't have fields to begin with; doing so may introduce unwanted jaggedness rather than improving the image. Once your output is free of fields, use the field order setting of "None" or progressive on the final encode.

Profiles

Profiles are a series of feature sets aimed at different applications. While there are many profiles within the H.264 standard, the most commonly used profiles today are Baseline, Main, and High. It helps to understand the issue of efficiency at the cost of performance when choosing a profile. Profiles with more capabilities tend to achieve better quality for a given bit rate while consuming more resources to implement.

Since these profiles are simply different sets of capabilities rather than options of a single linear setting (not to be confused with a low/mid/high quality setting), these can't be compared to one another on a continuous scale. The following descriptions of each profile provide only a relative comparison, since the complexity of the factors that make them either more efficient (and hard to handle) or less efficient (while being easy to compute)—together with the way each of their capabilities get utilized in various different scenarios on a given piece of video—can vary unpredictably. Which one of the profiles you should be using greatly depends on the application, and there is no overall winner.

Baseline: This profile is generally targeted at light applications such as video conferencing or playback on mobile devices with limited processing power. It provides the least efficient compression among the three choices, and at the lowest CPU overhead on decoding.

Main: This profile has more capabilities than Baseline, which generally translates to better efficiency; yet it comes at the cost of a relatively higher CPU overhead (though less than the High profile). This profile is usually used in medium-quality web video applications.

High: This is the most efficient profile among the three. It has the most capabilities that pack more quality into a given bit rate, yet it is also the hardest to process because of these added operations. Though originally intended only for high-definition applications such as Blu-ray, this profile is increasingly becoming popular for web video applications as well due to the increase in the processing power available to the average user.

Level

The level setting is a restriction on the rate of chunks the decoding process could run into. In practice, this translates into a frame size and frame rate combination restriction. The higher the level, the higher this restriction is set. Due to the vast spectrum of applications H.264 is used for (from tiny mobile video to full-quality cinema), it is sometimes necessary to filter out the range of permissible content for a given device or platform, depending on its processing capabilities. For example, a small device with limited processing capability should be prevented from attempting to play back a large feature film that involves large amounts of data and high-intensity processing despite the fact that the device may be able to interpret the data (albeit at an impractical pace) simply because it can recognize the file format.

For example, a level 1 setting restricts the amount of pixel "bundles" (macroblocks) so as to allow only a video that is barely 200 pixels or so wide running at 15fps, while level 3 would accommodate a 720 × 480 video at 30fps.

Since the level defines a limit and not an absolute settings combination, it is possible to use a level that is more than what is required. However, doing so may restrict your video from being played back on a device that is actually capable of playing it, if it mistakes the video for a much higher-volume file. Therefore, increase the level step by step until you are able to achieve the desired frame size/frame rate/bit rate combination without making it unnecessarily high.

While level settings are crucial to devices, video playback on desktop computers usually doesn't pay attention to the level setting in the file being played.

Video bitrate mode: VBR versus CBR

VBR, or Variable Bit Rate encoding, allows you to define a general average value or target stream rate in conjunction with a maximum value. The idea behind this is to use efficient compression to maintain high quality while allowing for occasional spikes of data due to difficult-to-compress segments of video. Generally, VBR is more efficient compared to CBR, or Constant Bit Rate encoding, when it comes to packing a file with the maximum quality for a given amount of data storage overall. However, allowing for these unpredictable spikes of data to maintain a constant rate of image quality could result in interrupted playback if the spikes become too frequent or the maximum limit is set too high. Therefore, VBR is commonly used for progressive downloads and file-based video on the web. However, with current broadband services capable of bursting data to much higher rates than what they can maintain at a constant level, VBR maybe a viable option for streaming as well in some cases.

CBR is traditionally used in streaming media and other applications where a constant predictable stream of data is essential. This predictability comes at a price of not letting H.264 use its adaptive compression capabilities to deliver constant quality to its fullest potential. CBR, in a way, trades consistency in quality to gain predictability in smooth playback without interruptions or pauses.

Though the general rule says that VBR is for progressive downloads and CBR is for streaming media, experimenting with both may yield contradictory results in a given case. Therefore, it is important to experiment with your specific content in your specific environment.

Number of passes

This setting determines whether the video is encoded in just one pass (compression run) or if the encoder revisits the video from beginning to end a second time to find ways to pack the data even more. This can be applied to either CBR or VBR. The number of passes is one of those factors that can result in better "packing" while not having any impact on how easy or hard it is to unpack the video. Generally, two passes take almost twice as much encoding time, but result in relatively better quality-to-bit-rate efficiency. However, in most cases the doubling in encoding time doesn't get you twice the quality. Therefore, choose two-pass if you want the best possible quality at all costs since this added investment is only on the encoding side and not at the expense of added processing power at playback (unlike some other parameters that impact both the encoding and decoding). However, if you believe the slight increase in bit-rate economy is not worth the added time spent on encoding, you may choose one-pass. Again, experiment to establish what works best for your particular case.

Set key frame distance

Key frames are full frames directly derived from the original source without the use of references to other frames within the video. The key frame distance, or how often these key frames appear in the video, can affect how closely the encoded video resembles the original uncompressed source. The frequency of the key frames can also affect how well the video is "scrub-able". Selecting this option lets you adjust this setting manually. Generally, the optimal key frame distance depends on the amount of motion in the video and the frame rate. Usually, it is set to one to three seconds, translated into frames using the frame rate. (For example, for a 30fps video, one second is 30 frames.)

Adobe® Flash® Media Encoding Server

Adobe Flash Media Encoding Server (FMES) is an enterprise-level encoding platform specifically designed for formats supported by Adobe Flash Player. FMES also provides a much richer assortment of H.264-specific controls compared to Adobe Media Encoder, which means greater flexibility and power to come up with the optimal combination of settings tailor-made to your specific situation.

FMES settings

AME is aimed at individuals and groups producing video on an ongoing basis (as in a post-production facility), while FMES is an enterprise-level solution for large-scale transcoding (such as a TV network publishing to the web).

Most of the basic settings at the beginning are similar to those in Adobe Media Encoder (see the previous section about AME settings for explanations of those settings).

Here are some of the additional H.264-related settings found in FMES that are not found in AME.

Stream-Basic

Stream Type: The Stream Type can be set to F4V or standard MPEG-4 System. F4V is more suitable for content targeting Flash Player.

Use streaming mode: This can be set either to true or false. Setting this to true brings to the front of the file a set of data that contains information about how to interpret the file, which is favorable for streaming.

Video-Basic and Bitrate

The Video Basic and Bitrate settings are similar to those used in AME, even though the naming may be slightly different. The "Maximum bitrate" can only be defined in VBR mode, because CBR only has a target rate (simply called "Video Bitrate" here).

H.264 Encoder Settings - Basic

Under this category, the now-familiar Profiles and Levels can be selected (see explanation of them under the AME settings section). Just like the "Maximum bitrate" setting, certain settings under this category may not be available, depending on the settings you have chosen earlier. For example, selecting the Baseline profile disables the CABAC Entropy Coding Mode option, because the Baseline profile does not provide that functionality.

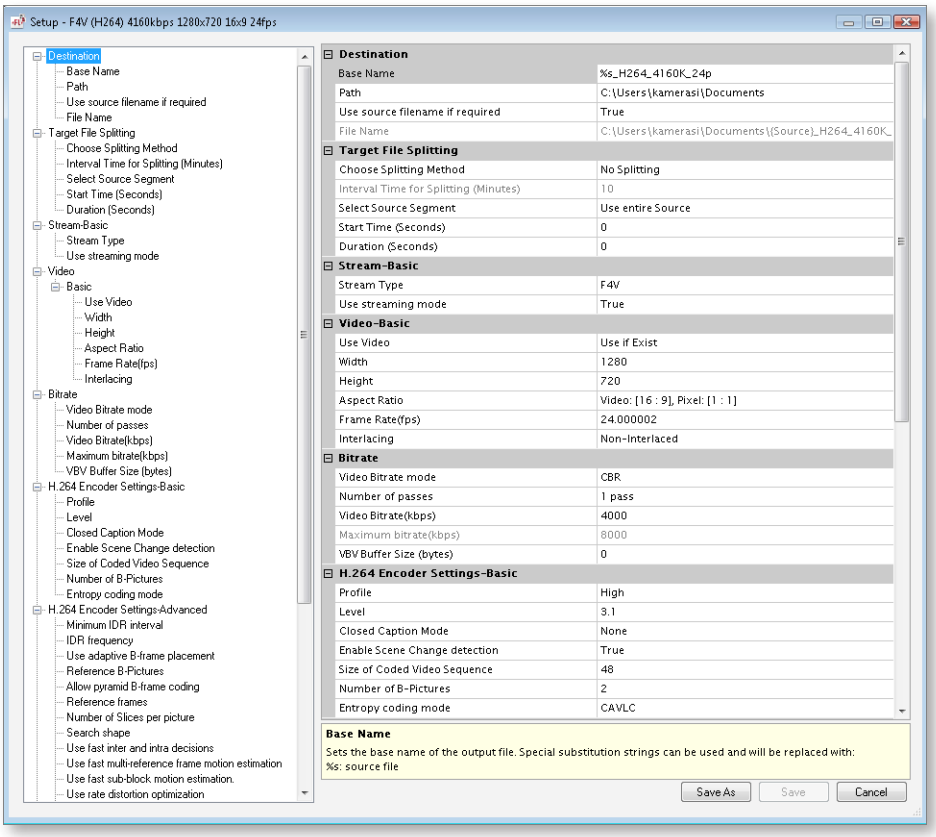


Figure 6 : FMES basic settings

Enable Scene Change detection: Setting this to true forces the encoder to insert a fresh video frame that defines a different type of visual information that has no resemblance to the previous set of frames. This is possible because H.264 can define sets of interrelated frames of varying length (unlike older compressors that use cyclic groups in the temporal compression process). Enabling scene change detection can reduce sudden reduction of quality when the video visually changes unpredictably; yet doing so will reduce the bit rate efficiency and may also increase encoding time.

Size of Coded Video Sequence: This refers to the length of the set of pictures used using referential compression methods. Videos that have fewer changes may permit higher values for this setting while increasing bit rate efficiency. Lower values provide better quality at the cost of lower bit rate efficiency. Setting this value to 1 totally stops advanced temporal compression methods, resulting in a file with higher fidelity to the source yet much lower bit rate efficiency. In most cases, a value 10 times the frame rate is adequate.

Number of B-Pictures (not available in Baseline profile): This refers to the number of referential pictures inserted between actual pictures derived from the source video. Increasing this number increases bit rate efficiency at the cost of reduced seek-ability and increased playback processing power demand.

Entropy coding mode: This mode can be set to either CAVLC (Context-based Adaptive Variable Length Coding) or CABAC (Context-based Adaptive Binary Arithmetic Coding). CABAC is not available when the Baseline profile is selected. When choosing between the two modes while using Main or High Profiles, the important thing to remember is that CABAC is more advanced, and hence gives better bit-rate-to-quality economy at the cost of higher processing power required compared to CAVLC. In general, for higher-quality applications such as large-format web video, try CABAC and see if it performs smoothly in the target playback environment. If the increase in quality comes at a significant reduction in acceptable playback performance, use CAVLC instead. (The increase in processing overhead for CABAC is usually worth the slight but noticeable increase in quality for a given bit-rate.)

H.264 Encoder Settings - Advanced

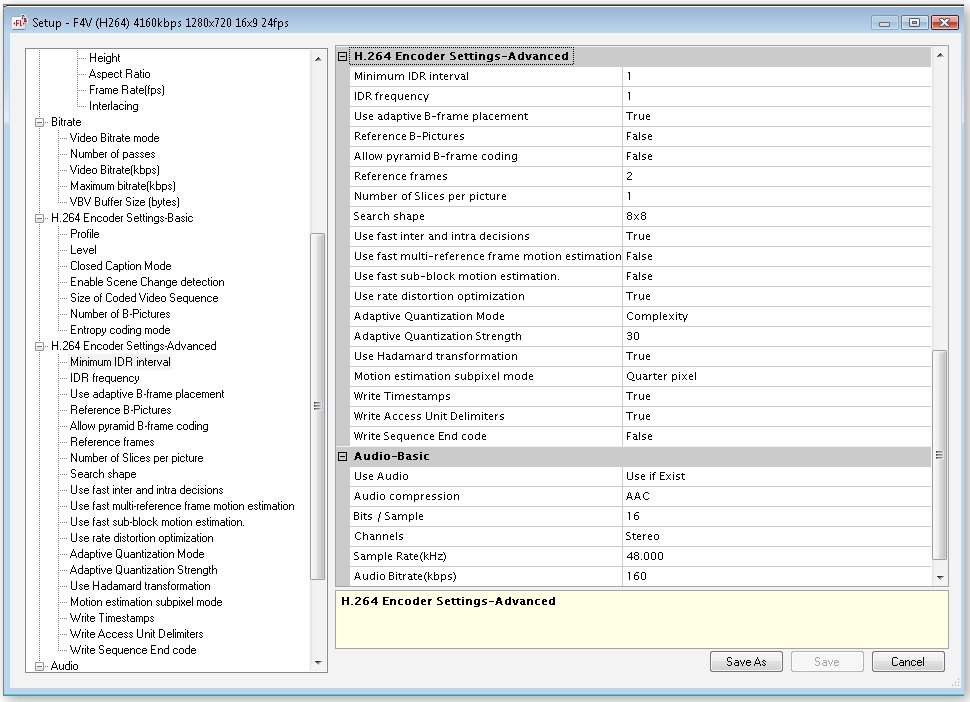


Figure 7 : FMES advanced settings

Let me start this scary section by saying that the best rule of thumb is not to mess with any settings you don't really understand, as they would only add more variables to your experiments. The value in the basic preset you choose will probably serve you well. However, if you do understand how these things work, I'll try to help you align that general knowledge with how it applies to FMES.

One key factor to remember with these settings as well is which of them affect only encoding and which of them affect both encoding and decoding on playback (similar to how certain suitcase-packing techniques result in longer packing time with no impact on unpacking, while others complicate both). In general, you should try to take advantage of as much processing overhead as possible to make your content compressed in the most efficient way during encoding without overwhelming the playback performance from the decoding side. Yet, in some cases, this could result in impractically long encoding times, which may not be worth the small increase in bit-rate economy it gives you.

Finally, remember that most of these techniques and algorithms are somewhat independent from one another, while all work in layers to achieve the common goal of optimizing bit-rate economy. (It's a bit like using "buy another one like this" sticky notes *and* steam pressing *and* plastic shrink wrapping on the same set of clothes to pack the most efficient suitcase ever, while trying not to over-degrade your clothes in either stage of the process at the same time.)

Minimum IDR interval: In old-school compression, there was only one type of key frame (or "I" frames). Now, in H.264, there are two classes of key frames: ("normal") I frames and IDR (Instantaneous Decoding Refresh) frames. The major factor that sets IDR frames apart from the non-IDR, or plain, I frames is that—as the name suggests—it forces the decoder to refresh itself (in other words, forget all about what happened prior to that frame)—hence not referring to any frames before that frame from then on.

Since IDRs lead each group of pictures, the interval between one IDR to the other is the size of the group of pictures. Unlike more traditional codecs' use of consistently sized cyclic groups of pictures, H.264 can have groups of pictures with different frame counts. "Minimum IDR interval" defines the minimum number of frames in a group (meaning

it can fluctuate higher but never lower than this number). Decreasing this value would increase video scrubbing accuracy and error resilience at the cost of bit-rate economy. In some cases, lower values may simplify the compression, resulting in better encoding and playback performance. Usually it's best to leave the interval at 1 and let the encoder decide the lowest interval in different parts of the video. If for whatever reason you would want H.264 to be restricted from inserting IDRs too close together, you can increase this value.

IDR frequency: This is the high end of the automatically determined IDR insertion range. Reducing the IDR frequency setting results in more of the I frames being designated as IDR frames, which results in better random seeking (scrubbing or skipping the video) and error resilience at the cost of bit-rate economy. IDRs may be placed about every 10 frames in most cases. If you require more responsiveness when scrubbing, a lower number or even 1 can be used. (When set to 1, all I frames will be designated IDR frames. However, if set to 0, only the very first frame of the entire file will be designated an IDR frame.)

Use adaptive B-frame placement: Since H.264 is capable of using irregular groups of picture sets, enabling this feature makes full use of this capability by placing B-frames based on what is going on in the particular video frames being encoded. Using this may increase encoding times but may improve bit-rate economy.

Reference B-Pictures: Enabling this option allows the encoder to use B-frames as reference frames. Note that this is only possible if the number of B-frames per sequence is greater than one. This may increase processing overhead in return for increased bit-rate economy.

Allow pyramid B-frame coding: Enabling this would add an additional level of complexity by letting B-frames refer to other B-frames. This adds more encoding and playback processing overhead to increase bit-rate economy.

Reference frames: Unlike previous settings that affect the frequency of a particular frame type, this setting defines the number of frames to which a resulting frame may refer in order to construct that particular frame. The higher this number, the more efficient the compression would be, but this hinders performance. This number can be between 2 to 16.

Number of Slices per picture: Each frame can be processed as one whole entity (one slice) or it can be split into two or four slices that could each have their own references, regardless what is happening on the other slices. One advantage of splitting frames is the ability of multi-CPU machines to process one slice per processor, effectively reducing the total time it takes to process a single frame; however, this may reduce the quality-to-bit-rate ratio.

Search shape: This refers to the size of the block used for motion estimation. The options are 8 × 8 or 16 × 16. Smaller blocks provide more motion-search accuracy at the cost of higher processing overhead during encoding, with no significant impact on playback performance.

Use fast inter and intra decisions: Enabling this option speeds up encoding times at the cost of bit-rate economy. In most cases, the gains in encoding speed may outweigh the negligible quality loss and slight decrease in bit-rate economy. Disable this option only in special cases when time is not of the essence and the maximum possible quality-to-bit-rate ratio is required. This setting has no significant effect on decoding performance at playback.

Use fast multi-reference frame motion estimation: Turning this feature on can further reduce encoding times at a slight cost of quality and bit-rate economy. In most cases, the benefits outweigh the drawbacks. This setting also has no significant impact on decoding complexity.

Use fast sub-block motion estimation: This additional algorithm uses an adaptive process to use sub-block motion estimation selectively, only when needed to increase encoding speed with negligible drawbacks in terms of quality loss.

Use rate distortion optimization: This method optimizes the bit rate based on the difference between the resulting compressed image and the source. Activating this option can increase encoding time while improving bit-rate economy. This may not directly impact the playback performance.

Adaptive Quantization Mode: Adaptive quantization (AQ) is a smart way of quantizing areas (blocks) of frames depending on the nature of their content. For example, when the area is relatively flat (clear sky or a flat wall, for instance), most areas can be interpreted with less data, while areas that have complex details get higher consideration; so areas that are more likely to produce artifacts are selectively compressed to a lesser degree. AQ has four possible settings:

- **None** turns AQ off
- **Complexity** mode works by taking the amount of detail in areas into consideration
- **Brightness** smoothens out flat dark areas
- **Contrast** tries to give dull less contrasting areas less importance

Complexity mode works for most content; however, you should experiment if your content is of a particular style.

AQ is an added encoding burden that improves bit-rate economy in return, yet it has no significant effect on decoding performance on playback.

Adaptive Quantization Strength: This setting controls the degree to which AQ is applied (0–100). If you feel AQ is oversimplifying the image or see significant artifacts in areas that are subject to AQ, try decreasing it; or if you feel the resulting images have more potential to take advantage of the increased efficiency of AQ, try increasing it.

Use Hadamard transformation: This algorithm further increases compression efficiency in smooth areas and is more effective in images with mostly smooth areas. It may add some processing cost, both on the encoding and decoding sides.

Motion estimation subpixel mode: This determines the thoroughness and accuracy of motion estimation. Full pixel is the easiest yet least effective mode, while Quarter pixel is the highest depth that requires the most processing power to encode. Half pixel is somewhere in between. Finer search depths give better quality-to-bit-rate ratios. This setting only affects encoding and not playback performance.

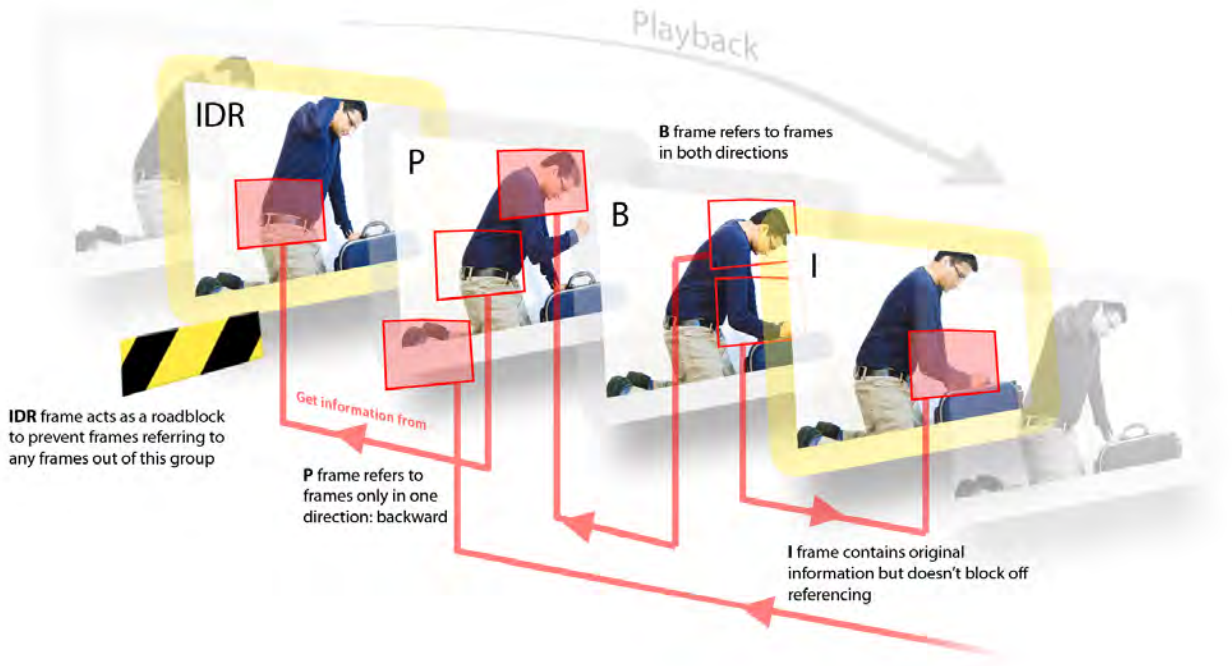
Write Timestamps: Not to be confused as an image overlay function (in other words, this won't cause timecode to appear in your encoded video visually), this is an additional piece of data that helps the decoder keep everything in sync (such as the audio and video). While theoretically this adds more data, its usefulness may outweigh the problems it might cause not to have this information included. So set it to true except in extreme or specialized cases where you would want to eliminate this information for whatever reason.

Write Access Unit Delimiters: The Access Unit Delimiter is an added piece of information written to the file that may help certain types of decoders to detect boundaries between frames, handle frames with fields, and so on. This may be required by some playback standards to maintain compatibility.

Write Sequence End code: The sequence end code is related to the Access Units and may help diagnose playback problems due to incompatibility in certain environments. This is not required (set to false) for most situations.

Still confused about all these different types of frames? Here is an illustration explaining the different kinds of functionality and purpose they have within a group of pictures of a compressed video.

Figure 8 : Frame types



The Kush Gauge:

Is your data economy below or above average?

To help newcomers get an idea of what combination of settings could be considered a little too high on the data rate for that size/type of content, or brilliantly nimble, making the pros look bad, or perhaps a little too stingy, or maybe just about right, I came up with a simple framework based on some experimentation and comparing notes with some others who have been tackling the same issue.

These are purely for reference and should not be treated as rules or one-size-fits-all values, as your particular situation may drastically differ from the situation that I tackled. Yet it could be interesting to compare your own compression trials and resulting optimal values with these; or you could use these as starting points or ranges to work from before discovering your own magic numbers.

Weighing the factors

Start by considering the factors that lead to needing higher bit rates to achieve a given level of quality:

- number of pixels in each frame
- number of frames per second
- amount of motion in the image (low/mid/high)

Calculating the amount of pixels per frame is easy: simply multiply the width by the height. For example, a 640 × 360 video has 640 × 360 = 230,400 pixels.

The frame rate is immediately known. In this example, assume it's 30fps. This should be the minimum frame rate at which the video is still acceptable (For instance, a computer screen capture demonstration need not use 30fps if only the mouse is moving.)

Consider the amount of motion (call it "motion rank"). As a general rule, try to simplify it into three ranks: Low, Medium, High. To define these ranks in real-world terms:

- **Low motion** is a video that has minimal movement. For example, a person talking in front of a camera without moving much while the camera itself and the background is not moving at all.
- **Medium motion** would be some degree of movement, but in a more predictable and orderly manner, which means some relatively slow camera and subject movements, but not many scene changes or cuts or sudden snap camera movements or zooms where the entire picture changes into something completely different instantaneously.
- **High motion** would be something like the most challenging action movie trailer, where not only the movements are fast and unpredictable but the scenes also change very rapidly.

To make this highly subjective yet crucial factor into a quantifiable number, try giving a multiplication factor to each rank. Since these ranks are not in a linear manner, I chose to give the following corresponding numbers to the ranks: Low = 1, Medium = 2, High = 4. (In other words, a video with a reasonable amount of movement is twice as hard to compress compared to one that has very little to no movement. An extremely fast and unpredictable video would be four times as hard to compress while maintaining the same level of quality.)

Given this relative multiplier based on these factors, I sought to develop a base number from which these multipliers can produce real-world bit-rate estimates. After numerous experiments, I noticed a certain pattern of what could be considered a "constant" or base value (for most commonly used video frame-size and frame-rate ranges). When rounded off, that value is 0.07 bps per pixel, per frame, per motion rank value.

In other words, to estimate the optimal H.264 bit rate value that would give what is considered "good quality" results for a given video, you could multiply the target pixel count by the frame rate; then multiply the result by a factor of 1, 2 or 4, depending on its motion rank; and then multiply that result by 0.07 to get the bit rate in bps (divide that by 1,000 to get a kbps estimate or by 1,000,000 to get a Mbps estimate).

Practical example

1280 × 720 @24fps, medium motion (rank 2):
1280 × 720 × 24 × 2 × 0.07 = 3,096,576 bps = ~ 3000 kbps

If the motion is high (rank 4), it's about 6000 kbps.

On the other hand, if the same clip is still usable at 5fps, and the motion is low:
1280 × 720 × 5 × 1 × 0.07 = 32,256 bps = ~ 320 kbps

A reduction in frame size can dramatically reduce the bit-rate requirements:
640 × 360 × 5 × 1 × 0.07 = 80,640 bps = ~ 80 kbps

This example shows how these factors could account for dramatic bit-rate differences among videos of the same frame size, yet containing different frame rates and degrees of motion.

In case of CBR, a value close to this estimate can be used. In case of VBR, a value that is about 75% of the estimate can be used as a target and a value about 150% of it can be used as the maximum rate. This VBR gap greatly depends on the nature of the content and the ability to absorb the bit rate spikes in the target playback environment.

What about audio?

H.264 is strictly a video codec. How audio information is coupled with the video stream is entirely up to the container (for example, F4V). The AAC (Advanced Audio Coding) codec is generally used with a video stream containing H.264 video. AAC has many improvements over previous popular lossy audio compression technologies such as MP3, including higher efficiency (same quality at lower bit rates) and added features (more channels for implementing surround sound, wider range of sampling rate options, and so on). AAC also has variants: for instance, HE-AAC is for higher-efficiency, lower-bit-rate applications such as streaming.

Just as for video, you should try your best to start from uncompressed or lossless audio information and encode to a lossy audio format only at the final stage.

Mono or stereo?

True stereo sound requires two independent channels of audio. For this reason, you should think of audio in terms of a single track or double track, rather than an added feature. When your bit-rate budget is tight, choose stereo only when the content absolutely requires it. For example, if the video is a music video, stereo sound may be important for a good user experience; but a talking-head video where most of the content is speech can be consumed in mono, even if there is some stereo music in it (during the introduction, for example). If the source material was in mono to begin with, there is absolutely no point in using stereo encoding. (It's a bit like saving a black-and-white picture in a color file format).

Bits per sample

Just as digital pictures are divided into pixels with distinct levels of intensity (such as 256 levels of gray), digital audio has distinct steps that the waveform could be at a given time. Using more bits per sample can define an audio stream that is finer and closer to being real waveforms, since it has higher number of "steps." Generally, 16-bit audio is considered high quality, and all other means of decreasing the bit rate should be considered before resorting to decreasing the bits per sample value.

Sampling frequency

Sampling frequency is the audio equivalent of image resolution. Just as higher resolution images with more pixels can hold finer detail, while lower resolution images can only show larger forms, higher sampling rates enable finer detail or higher-frequency sounds to be stored with high fidelity.

Since the human ear is not sensitive to audio frequencies higher than approximately 20kHz, an audio "resolution" capable of producing such a signal (a wave form consisting of one sample up, then one sample down) is around 40kHz. So a sampling frequency of 44.1kHz is suitable for most applications. The exception is speech-only content, which may be encoded to lower sampling frequencies, such as 22.05kHz, because the highest useful frequencies in speech fall well below the highest human sound perception. Yet suppressing the frequency range may degrade other subtle sounds, such as hisses and breathing noises, which could give the perception of audio quality loss. Generally, sampling frequencies higher than 48kHz are not used except in specific high-fidelity audio applications.

Audio bit rate

Usually, the audio stream consumes a much smaller proportion of the overall bit rate allocation of an audio-video stream or file. Even at considerably low bit rates, reasonable audio quality can be achieved. For example, a stereo music track can be encoded at 96–128kbps with minimal to no noticeable loss in perceived sound quality. In mono sound, bit rates as low as 56–80kbps may still be acceptable, while in particular speech-only audio applications where keeping the sound comprehensible is the only goal (as opposed to maintaining an aesthetically pleasing voice), bit rates can be drastically low.

Audio source considerations

When looking for an optimum audio bit rate for a given situation, start by deciding whether you want the audio track to be mono or stereo (or surround sound!) first. Then try experimenting with different bit rates while keeping the base audio at 16-bit 44.1kHz. If quality suffers considerably when you try to achieve bit rates as low as you really need, only then start making higher-level compromises (aside from the mono versus stereo decision) such as reducing the sampling frequency and the bits per sample.

For best results, make sure the audio is "cleaned up" before crunching it in the encoder. Just as you would color-correct and adjust levels to make sure the visuals are making the best out of the available color range using level/brightness/contrast controls for video, make sure your audio waveform uses most of the available amplitude range without clipping (going out of range, resulting in a flat line in the waveform).

Conclusion

Video compression using H.264 is more of an art than a science, but having a basic understanding of how certain factors influence the outcome can greatly improve your ability to achieve better results. While there are no magic formulas or clear-cut rules, starting off from a sound foundation and then gradually refining your settings through controlled experimentation can yield the optimum results in a given situation.

I hope you found it useful to follow my journey though the turbulent airways of the encoding world. Are you a better packer now?

Acknowledgements

Special thanks to the following people who helped me with this guide:

Benoit Ambry
Karl Soule
Mike Melanson
Desiree Motamedi
Laurel Reitman
Jessica Fewless
Elen Gales (photography)

..and anybody else I forgot.

About the author

Kush Amerasinghe works as a computer scientist for Adobe Systems in California. He currently wears many hats including being a video content producer for Adobe TV (tv.adobe.com) while hosting his own show "Ask the Adobe Ones"—a lighthearted, interactive web show where the audience gets to ask questions from him and other Adobe experts. (It is one of the most subscribed Adobe TV shows on Adobe Media Player.)

Before joining Adobe, Kush worked in many different fields, including multimedia, web, television, visual effects, and education.

He can be reached at: ask@adobe.com