Video Encoding Settings for H.264 Excellence
Video Encoding Settings for H.264 Excellence

Quality is not an act, it is a habit.
— Aristotle

by Jason Robert Carey Patterson, Apr 2012

This document describes in detail a set of resolutions, bitrates and settings used for high-quality H.264 video encoding, and the reasoning behind those choices. Video encoding is a game of tradeoffs, and these settings represent a balance which is very good, and difficult to improve upon.

Familiarity with basic video-encoding terms and technology is assumed when reading this document, and an understanding of the nitty-gritty details might also be helpful.

Resolutions & Bitrates

We provide video files at 7 different standard widescreen resolutions...

- **240p** (424x240, 0.10 megapixels)
- **360p** (640x360, 0.23 megapixels)
- **432p** (768x432, 0.33 megapixels)
- **480p** (848x480, 0.41 megapixels, "SD" or "NTSC widescreen")
- **576p** (1024x576, 0.59 megapixels, "PAL widescreen")
- **720p** (1280x720, 0.92 megapixels, "HD")
- **1080p** (1920x1080, 2.07 megapixels, "Full HD")

Encoding at such a wide range of resolutions is based on the assumption a web video embedding mechanism will be used that is capable of detecting the viewer’s Internet connection speed and choosing the appropriate video file based on that link speed, along with the screen size and playback capabilities of the browser – thus supplying each different viewer with the best resolution and bitrate he/she can use. Simpler video embedding may also be used, of course, right down to a basic HTML5 `<video>` tag pointing to a single version of the video, but the whole point of encoding at a variety of resolutions and bitrates is intended for an intelligent embedding mechanism to make sensible use of those many different versions, supplying high-quality HD video to those viewers with fast enough Internet links, while falling back gradually for viewers with increasingly slower links.

For each resolution, we use a bitrate which is the lowest sensible 64% cut (80% of 80%) of a common Internet link speed (see below) that still achieves "very good" visual quality, with no major visible compression artifacts. Just like saving a still image for use on a website, we put quality first and only compress as much as possible without introducing any noticeable degradation (hopefully). If that means using a higher bitrate for a given resolution than some other websites, then so be it – the ’net can take it, and bandwidth is less and less of a problem every day. It’s unwise to
push bitrates too low and risk delivering a blurry, unprofessional video, which other sites such as YouTube routinely do.

For most of the resolutions we also provide a higher quality (HQ) version encoded at a somewhat higher bitrate, for the benefit of users with sufficiently fast Internet links. The visual differences from the normal, "very good" quality version to the HQ version are generally quite small, such as less blur during rapid motion, less risk of banding in dark scenes, and less risk of crystallizing during difficult fades. Nonetheless, we might as well take advantage of the user's link speed for improved quality from fewer compression artifacts, assuming the user's link is not fast enough to get up to the next higher resolution, which would be a significant step up in general sharpness and clarity.

Finally, we also encode a "superbit", ultimate-quality version at full HD 1080p resolution using a very high bitrate similar to Blu-ray. The bitrate chosen is 20 Mbps, which is a safe 80% of the maximum peak bitrate allowed for H.264 level 4.0. The superbit version should be almost lossless, practically indistinguishable from the original master – a "transparent" encoding, as it's known. It acts both as the best local playback (non-web) version and as a long-term master for future transcoding (eg: YouTube uploads, burning to DVD/Blu-ray, re-encoding with future codecs etc).

Audio is encoded at 44.1 kHz in AAC format, mono at 64 kbps or stereo at 128 kbps, both of which are excellent quality, practically indistinguishable from the original master. The lower resolutions all use identical audio settings to allow switching cleanly between them in an adaptive streaming scenario without any audible "pops". The 128-kbps versions are a stereo equivalent of the same settings. Switching from mono to stereo with the same settings should be almost seamless, and should only happen once in most cases, since a user is unlikely to need to switch back down if their link is ever over 5 Mbps (making that changeover at a lower speed, such as 3 or 4 Mbps, would be more problematic – the faster the link, the more likely it is to be well above what we need). The superbit, ultimate-quality version uses the maximum possible AAC bitrate of 320 kbps since it acts as a long-term master and we want the minimum possible quality loss if we have to re-encode from it (although 256 kbps would be sufficient and is considered effectively lossless for later re-encoding purposes).

The exact bitrates chosen are...

<table>
<thead>
<tr>
<th>Name</th>
<th>Resolution</th>
<th>Link (Mbps)</th>
<th>Bitrate (Mbps)</th>
<th>Video (kbps)</th>
<th>Audio (kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>240p</td>
<td>424x240</td>
<td>1.0</td>
<td>0.64</td>
<td>576</td>
<td>64</td>
</tr>
<tr>
<td>360p</td>
<td>640x360</td>
<td>1.5</td>
<td>0.96</td>
<td>896</td>
<td>64</td>
</tr>
<tr>
<td>432p</td>
<td>768x432</td>
<td>1.8</td>
<td>1.15</td>
<td>1088</td>
<td>64</td>
</tr>
<tr>
<td>480p</td>
<td>848x480</td>
<td>2.0</td>
<td>1.28</td>
<td>1216</td>
<td>64</td>
</tr>
<tr>
<td>480p HQ</td>
<td>848x480</td>
<td>2.5</td>
<td>1.60</td>
<td>1536</td>
<td>64</td>
</tr>
<tr>
<td>576p</td>
<td>1024x576</td>
<td>3.0</td>
<td>1.92</td>
<td>1856</td>
<td>64</td>
</tr>
<tr>
<td>576p HQ</td>
<td>1024x576</td>
<td>3.5</td>
<td>2.24</td>
<td>2176</td>
<td>64</td>
</tr>
<tr>
<td>720p</td>
<td>1280x720</td>
<td>4.0</td>
<td>2.56</td>
<td>2496</td>
<td>64</td>
</tr>
<tr>
<td>720p HQ</td>
<td>1280x720</td>
<td>5.0</td>
<td>3.20</td>
<td>3072</td>
<td>128</td>
</tr>
<tr>
<td>1080p</td>
<td>1920x1080</td>
<td>8.0</td>
<td>5.12</td>
<td>4992</td>
<td>128</td>
</tr>
<tr>
<td>1080p HQ</td>
<td>1920x1080</td>
<td>12.0</td>
<td>7.68</td>
<td>7552</td>
<td>128</td>
</tr>
<tr>
<td>1080p Superbit</td>
<td>1920x1080</td>
<td>N/A</td>
<td>20.32</td>
<td>20000</td>
<td>320</td>
</tr>
</tbody>
</table>

...and for older, non-widescreen content (with only 75% as many pixels)...
<table>
<thead>
<tr>
<th>Name</th>
<th>Resolution</th>
<th>Link (Mbps)</th>
<th>Bitrate (Mbps)</th>
<th>Video (kbps)</th>
<th>Audio (kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>240p</td>
<td>320x240</td>
<td>1.0</td>
<td>0.64</td>
<td>576</td>
<td>64</td>
</tr>
<tr>
<td>360p</td>
<td>480x360</td>
<td>1.2</td>
<td>0.77</td>
<td>704</td>
<td>64</td>
</tr>
<tr>
<td>480p</td>
<td>640x480</td>
<td>1.5</td>
<td>0.96</td>
<td>896</td>
<td>64</td>
</tr>
<tr>
<td>480p HQ</td>
<td>640x480</td>
<td>2.0</td>
<td>1.28</td>
<td>1216</td>
<td>64</td>
</tr>
<tr>
<td>576p</td>
<td>768x576</td>
<td>2.3</td>
<td>1.47</td>
<td>1408</td>
<td>64</td>
</tr>
<tr>
<td>576p HQ</td>
<td>768x576</td>
<td>2.5</td>
<td>1.60</td>
<td>1536</td>
<td>64</td>
</tr>
<tr>
<td>720p</td>
<td>960x720</td>
<td>3.0</td>
<td>1.92</td>
<td>1856</td>
<td>64</td>
</tr>
<tr>
<td>720p HQ</td>
<td>960x720</td>
<td>4.0</td>
<td>2.56</td>
<td>2432</td>
<td>128</td>
</tr>
<tr>
<td>1080p</td>
<td>1440x1080</td>
<td>6.0</td>
<td>3.84</td>
<td>3712</td>
<td>128</td>
</tr>
<tr>
<td>1080p HQ</td>
<td>1440x1080</td>
<td>9.0</td>
<td>5.76</td>
<td>5632</td>
<td>128</td>
</tr>
<tr>
<td>1080p Superbit</td>
<td>1440x1080</td>
<td>N/A</td>
<td>20.32</td>
<td>20000</td>
<td>320</td>
</tr>
</tbody>
</table>

Each bitrate has been chosen to be 64% (80% of 80%) of a common Internet link speed (1, 1.5, 2, 2.5, 3, 4, 5 Mbps etc) in order to make full – but safe – use of that common connection speed. Unlike still images, where we want the smallest good-looking file to have the web page finish loading as quickly as possible, for video we only care that it downloads fast enough to play. Downloading a video faster than we need for playback doesn’t make any difference to the end user, but using more of his link speed to improve the quality does, so we actually want to avoid under-shooting the user’s available bandwidth for video, and *increase the size/bitrate as much as is safely possible*. On the other hand, pushing too hard and using too much of the user’s link speed might result in pausing (for buffering), which is much worse from an end user’s point of view than slightly lower quality.

So, we start with bitrates which match the most common link speeds, then leave 20% headroom for *protocol overhead* (which can be as much as 16% on ADSL) and contention during busy times. *Assuming 80% of peak link performance has long been a good real-world guide*, and US FCC data from 2011 confirms that’s still the case (see chart). We then only use *80% of the remaining 80%* (so just 64% of the original advertised link speed), leaving the extra 20% as ”*bitrate headroom*” to allow the video to download safely ahead of what the target bitrate needs on average, to cover bitrate fluctuations and spikes during hard-to-encode parts of the video, such as rapid motion. Again, experience shows 80% is a good, safe but not overly conservative choice (some assume as much as 90% is safe, while Netflix seems to use a conservative 75%).

Internet link speeds continue to rise rapidly, so while our chosen bitrates are higher than some other video websites, for quality’s sake, they’re still quite reasonable. Based on Akamai data from 2010, the average real-world downloading speed (after protocol overhead) is already 8+ Mbps in Japan, South Korea and Hong Kong, 4.6 Mbps in the USA and Canada, somewhere around 4 Mbps in Western Europe, 2.9 Mbps in Australia and 2.6 Mbps in Russia. Even 3G cellphone networking is around 2 Mbps on average, although it’s highly variable. The *average American can therefore already view the 720p high-definition versions* of our videos without waiting, and the average Australian or Russian the 480p versions. The average in such statistics is skewed by the high speeds, of course, since it’s an exponential curve, but even so, about one third of Internet connections in modern countries are over 5 Mbps real-world downloading speed, which is enough for the 720p HQ versions, and 70% are over 2 Mbps and therefore can definitely view the 480p versions without waiting. Even in Australia, where broadband speed is more uneven and the average lags behind most modern countries, *government statistics from 2011* indicate 89% of users can view the 360p versions without any waiting (1.5+ Mbps link speed), and 45% can instantly view the full 1080p versions (8+ Mbps link speed).
It's useful to compare our chosen video bitrates with the major online video providers. All use H.264 as their preferred video codec. The bitrates below have been personally verified for iTunes, YouTube and Vimeo, are official statements from Hulu, the BBC and ESPN, and are taken from the user interface of Netflix...

<table>
<thead>
<tr>
<th>Provider</th>
<th>Video Bitrate (kbps)</th>
<th>~240p</th>
<th>360p</th>
<th>480p</th>
<th>720p</th>
<th>1080p</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC</td>
<td></td>
<td>800</td>
<td>1500</td>
<td>3200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESPN</td>
<td></td>
<td>~400</td>
<td>1400</td>
<td>2000</td>
<td>2800</td>
<td></td>
</tr>
<tr>
<td>Hulu</td>
<td></td>
<td>~480</td>
<td>700</td>
<td>1000</td>
<td>2500</td>
<td>3200</td>
</tr>
<tr>
<td>iTunes</td>
<td></td>
<td></td>
<td>1500</td>
<td>4000</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>Netflix</td>
<td></td>
<td>~250</td>
<td>560</td>
<td>1050</td>
<td>2350</td>
<td>4800</td>
</tr>
<tr>
<td>Vimeo</td>
<td></td>
<td>800</td>
<td>2000</td>
<td>4500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YouTube</td>
<td></td>
<td>250</td>
<td>500</td>
<td>1000</td>
<td>2000</td>
<td>3500</td>
</tr>
<tr>
<td>Our Bitrate</td>
<td></td>
<td>576</td>
<td>896</td>
<td>1216</td>
<td>2496</td>
<td>4992</td>
</tr>
</tbody>
</table>

Or, more graphically...

Looking in terms of resolution...

- **At 240p we use a higher bitrate than everyone else** in order to guarantee a minimum acceptable level of professional-looking quality, rather than looking ridiculously blurry/blocky, a la YouTube. Even at this bitrate, overall blurring and lack of detail is an issue, but serious compression artifacts are generally only noticeable during high motion and fades, which are a definite struggle. Other than that, the content looks reasonably good, just at a very small, low resolution.

- **At 360p our bitrate is almost double YouTube and Netflix, 28% above Hulu**, and roughly equal with (12% above) the quality-oriented BBC and Vimeo, again to provide a professional level of quality. Fades are still very problematic, but the rest of the time the quality is actually quite good. At this resolution and the next, ESPN uses an insanely high bitrate, which should be ignored (see below).

- **At 480p we're right in the middle of the pack**, half way between YouTube/Netflix/Hulu on the low side and the BBC/iTunes on the high side. At this point, the quality meets our desired standard of being "very good", with few visible compression artifacts and no major ones. It would pass for standard-definition digital television, despite being about a quarter the bitrate!
• At **480p HQ we use the same bitrate as the BBC and iTunes**, and just a little (12%) below Netflix, the only other provider to offer a second, higher-quality 480p version. The 26% increase in bitrate helps to reduce any remaining visible compression artifacts, so video quality is even better, with only the most difficult content, such as high-detail fades, causing any noticeable glitches.

• At **576p none of the major providers offer a comparable version**, so it's been left out of the chart for simplicity. Quality-wise, the same comments as for 480p apply here, and below, with very good quality at the lower bitrate, and even better at the higher bitrate. We choose to include 576p versions, unlike the other providers, because the 3-4 Mbps range is a reasonably common connection speed and we can offer 45% more pixels with that extra bandwidth, rather than simply throwing it away.

• At **720p we're again right in the middle of the pack**, half way between YouTube/Vimeo on the low side and the BBC on the high side, and roughly equal with Netflix and Hulu.

• At **720p HQ we use roughly the same bitrate as the BBC and Hulu**, a little (15%) below Netflix, and well below the crazy bitrate of iTunes (but see below).

• At **1080p we're well above YouTube, slightly above Vimeo and equal with Netflix and iTunes**, since there’s no point trying to push lower – not many links exceed 7 Mbps but not 8 Mbps, so dropping to 4352 wouldn't achieve much except reduced quality.

• At **1080p HQ none of the major providers currently offer a comparable version**, so it’s been left out of the chart for simplicity, although similarly high bitrates are available from niche vendors and are common for high-quality 1080p encodings in the pirate world. The increase in bitrate for 1080p HQ is a generous 50%, compared to 17-26% for the other HQ versions, since this is the final version we offer, and we want to ensure it has excellent quality.

Looking in terms of link speed...

- for very slow links we provide a reasonably good, but small, 240p version at 640 kbps
- for 1.5+ Mbps links we provide a quite good 360p version
- for 1.8+ Mbps links we provide a very good 432p version
- for 2+ Mbps links we provide a very good 480p version, and even better at 2.5+ Mbps
- for 3+ Mbps links we provide a very good 576p version, and even better at 3.5+ Mbps
- for 4+ Mbps links we provide a very good 720p version, and even better at 5+ Mbps
- for 8+ Mbps links we provide a very good 1080p version, and excellent at 12+ Mbps

As the chart shows, there are really 3 camps of providers. First, there are the **providers whose bitrates seem too low: YouTube and Vimeo**, plus Netflix and Hulu at the lower resolutions. They aren't as concerned about quality as they are about making sure it plays without waiting at all costs, even if the quality is poor. Our chosen bitrates are significantly higher than both YouTube and Vimeo at all resolutions, due to our goal of very good visual quality with no major visible compression artifacts. At the lower resolutions, we also use higher bitrates than Netflix and Hulu, again for quality’s sake, although they’re equal at higher resolutions. Interestingly, Netflix and Hulu are the only others to offer multiple bitrates at some resolutions (480p and 720p) to make full use of the user’s Internet link speed for higher quality.

Second, there are the **providers who more-or-less agree with our chosen bitrates: Netflix and Hulu at the higher resolutions, and the BBC**. We’re in near-perfect agreement with Netflix on appropriate transition points to 480p and 720p, and in near-perfect agreement with the BBC on high-quality bitrates.

Finally, there are the **providers whose bitrates seem too high: Apple’s iTunes movies and TV shows, and ESPN** especially at the lower resolutions. There are technical explanations for both cases. iTunes is a bit sneaky about it, but it doesn't actually offer true 848x480p as an option, instead using 640x480p **anamorphically scaled** in the horizontal dimension to widescreen. Apple does this for user simplicity – just one “SD” which works on all of Apple’s devices, even the

UPDATE: Providers sometimes change the bitrates they offer based on experience and feedback from their users. For example, a couple of years after this article was written, Netflix split their 1080p 4800kbps offering into two normal/HQ variants, at 4300kbps
old iPhone 1 and earlier video iPods. Lowering the resolution like this means offering only 75% as many pixels, which of course means Apple's bitrate at 480p actually covers only 75% as many pixels, or put another way, it effectively uses a 33% higher bitrate per pixel. iTunes also uses a higher bitrate than we do at 720p, again by about 30%. At 720p iTunes doesn't sacrifice resolution like it does at 480p, thankfully, but instead it just throws bandwidth at the problem, using a whopping 4 Mbps for 720p video! In both cases, this is primarily because Apple needs to use high bitrates to compensate for the poor quality of the standard QuickTime H.264 encoder, which consistently comes last in codec comparisons and is visually worse even at a glance – our 480p encodings at 1216 kbps are better than theirs at 1500 x 1.33 = 2000 kbps, a 64% higher per-pixel bitrate! ESPN's bitrates are also higher than ours at all resolutions except the very smallest version (where they're willing to sacrifice a lot of quality to ensure it plays without waiting at all costs, for those crazy sports fans), but the overly high bitrates are particularly obvious at the lower mainstream resolutions. This is because ESPN still uses old RTMP adaptive streaming (you thought that was dead, didn't you?), forcing them to use less flexible constant-bitrate encoding (yikes!).

**Encoder Software - x264**

The choice of video encoder software has an extremely large impact on final quality, probably more than any other choice except bitrate. **We use and recommend the excellent, open-source x264 encoder**, which is essentially the gold standard of H.264 video encoding today, and has been for several years. Since 2006, x264 has consistently won the annual MSU MPEG-4 AVC/H.264 Video Codecs Comparison competition every year, along with numerous other codec comparisons and reviews. Its nearest rival is generally the MainConcept H.264 encoder used in applications such as Adobe Media Encoder and Microsoft Expression Encoder. While MainConcept is also a very good encoder, x264 reliably produces slightly better quality at any given target bitrate, both subjectively (IMHO) and as objectively measured by whatever metric is being used in the comparison (PSNR, SSIM etc).

In our particular case, we use the `x264Encoder` QuickTime codec, which provides x264 in the form of a plug-in component that integrates into the Mac OS X QuickTime video library, allowing easy use from most Mac applications including our batch-encoding tool of choice, Compressor. We could also use x264 through other, non-QuickTime-based encoding tools, such as Handbrake, but they may not provide full access to all of x264's settings, and they probably wouldn't also support other QuickTime codecs such as Flip4Mac WMV which we also use for our overall encoding system. Compressor with x264Encoder fits our requirements nicely, although we do lose a few percent in performance because both Compressor and QuickTime 7 (on which it's based) are only 32-bit.

The high quality of x264 encoding is primarily due to...

- **aggressive motion-estimation search**, which helps find as much temporal and spatial redundancy in the image as possible, using a large number of initial candidate predictors followed by a complex, uneven multi-hexagon search (with early exit for speed), followed by sub-pixel refinement using full rate-distortion optimization to account for the real, final cost-vs-benefit of each choice

- **excellent bitrate control/distribution**, using macroblock-level analysis ("MB-tree") to track the degree of referencing of each macroblock through the actual motion vectors from future frames, allowing the encoder to only lower the quality in the areas of each frame which are changing rapidly (not referenced much in the future), rather than lowering the quality of the whole frame as in most encoders – essentially traditional bitrate control but applied at the level of each 16x16 macroblock rather than at the whole-frame level – which helps maintain clear, stable backgrounds in the presence of moving foreground objects

- **intelligent, adaptive, variable use of B-frames**, rather than just using a fixed pattern like IBBPBBPBBB as in most encoders, to make better use of the available bitrate by inserting the more expensive but higher-image-quality I- and P-frames where they're of most benefit to serve as reference frames, which is good at all times but is particularly important during fades (one of the hardest things to compress well)
- **adaptive quantization**, which varies the quantizer for each individual macroblock within each frame to avoid blur/blocking in flat areas containing fine/edge detail, such as calm water, sky and animation

- **full rate-distortion optimization** used for motion-vector refinement, macroblock partitioning (subdividing each macroblock, balancing the cost of additional motion vectors against the benefit of the less complex residual image left to encode), and final quantization (the key lossy step!), which selects locally-optimal motion vectors, macroblock partitioning and quantization based on cost-vs-benefit using the real, actual cost of each possible choice when that choice is processed right through to final entropy encoding, versus the image-quality benefit as measured by the RDO metric (see below)

- a "psycho-visual" rate-distortion optimization metric, which tries to match perceived visual quality better by de-emphasizing blurry "low-error but low-energy" choices, rather than using simpler metrics like sum of absolute differences (SAD), peak signal-to-noise ratio (PSNR) or structural similarity of images (SSIM), which all tend to lean towards low numeric pixel differences but too much blur

x264 is also fast, with SIMD vector instructions used for most primitive operations, along with good multi-threading which achieves a near-linear speedup during the second encoding pass on multi-core and multi-processor systems (video encoding is naturally a highly parallel problem, of course, which makes parallelizing it pretty easy). x264 is very fast for a software encoder, and actually competitive with dedicated encoding hardware if fast settings are used, while achieving better quality. We, of course, use much slower settings to achieve the highest possible quality, but we still appreciate the speed of x264.

**Others Encoders:** Even if you do use a different encoder, such as MainConcept, you should still find this document useful, as most encoders offer similar settings based on the underlying H.264 format itself and the nature of video encoding in general. A word of warning, though – **do not use Apple's standard H.264 encoder, the one that comes built into QuickTime**, as it isn't very good and consistently comes last in encoder comparisons, especially at low bitrates.

**Encoding Strategy & Passes (--pass)**

Having decided on resolutions, bitrates, and encoder software, the next most important setting is the overall encoding strategy. There are essentially 4 possibilities...

- **constant quality** – uses a fixed quantizer and lets the resulting file size/bitrate fluctuate arbitrarily (like saving JPEG images at a certain quality, the file size will be different for different images depending on their complexity)

- **constant bitrate** – makes each frame use the same number of bits and lets the resulting quality fluctuate arbitrarily (would be like saving JPEG images all to the same file size, the quality would be worse for some images than others depending on their complexity)

- **single-pass variable bitrate** – varies both the quality and bitrate from one frame to the next, giving the harder frames more bits in order to optimize for overall quality at a target average bitrate, but with no knowledge of the overall content of the video except for a small lookahead window of a few frames

- **2-pass variable bitrate** – varies both the quality and bitrate from one frame to the next, giving the harder frames more bits to achieve the maximum overall quality at a target average bitrate, knowing the complete content of the video in advance to allocate bits most appropriately across the whole video, perhaps subject to a local peak bitrate constraint

The constant-quality, fixed-quantizer strategy is simple, but is of little interest for Internet video because the bitrate is too variable and the final file's average bitrate is unpredictable. Ultimately, we're targeting particular Internet link speeds, after all. The constant-bitrate approach solves this problem, of course, and can be made to work reasonably well, but it must typically use an excessively high bitrate to guarantee acceptable quality in the hard-to-encode parts of the video (as in the ESPN example, above), because every frame gets an equal bit allocation, whether it's easy or hard to encode. Typically, this means setting the bitrate high enough to handle the worst case, which means it's too high, and wasteful, 99% of the time.
Single-pass, variable-bitrate encoding is fast and produces reasonably good quality, but the **single pass means the encoder must guess what might be coming** in the future and make reasonable allowances "just in case", since the encoder isn't psychic and doesn't know the content of the rest of the video. It won't know one part is a particularly easy scene, while another part is a hard scene to encode, so easy-to-encode scenes tend to get too many bits relative to the harder scenes, where those bits would have done more good.

With **2-pass encoding, the encoder makes an entire pass through the video before writing a single bit** to the output file, precisely in order to learn exactly where the bits would be spent most effectively (which scenes are the hard ones etc). Unless you're in a high-volume situation where reducing the encoding time really matters, or a live situation where you can't know what's coming, if you can afford to wait, the **longer encoding time of 2-pass variable-bitrate encoding is definitely worth it** and results in significantly better quality.

**All:** 2-pass variable-bitrate encoding, to achieve the best possible quality at the target bitrates and use every bit as wisely as possible.

### H.264 Profile (--profile)

The **H.264 profile** is a compatibility issue. It defines the capabilities required of the player, that is, which features of the full H.264 format the player must support in order to play the file. Naturally, the encoder can't use any features which the player isn't guaranteed to have, so using a higher profile uses more features to give better quality at the same target bitrate, but prevents some older or lower-end players from playing the file altogether...

- **Baseline profile is a subset of the normal H.264 format** which drops the most CPU-intensive features, primarily B-frames, weighted predictions and **CABAC arithmetic coding** for the final entropy coding step. Removing these features allows H.264 videos to be played by quite low-performance processors, but sacrifices quite a lot of quality to achieve that. Baseline is the only profile supported on very old handheld devices such as the early video iPods and the original iPhone 1/3G.

- **Main profile was the original H.264 format,** and is the profile **officially** supported on certain popular older Apple devices such as the iPhone 4, iPad 1 and Apple TV 2, although the hardware decoder in the A4 processor inside those devices actually supports high profile (see below). Why Apple doesn't state that in the specs is a mystery, but the specs are wrong – iTunes will happily sync high-profile videos to those devices (and even to the iPhone 3GS) as long as they're under the resolution and bitrate limits (see **H.264 level** below). iTunes uses **high profile for its 720p resolution videos**, as does YouTube, and all play just fine on those devices. In reality, **hardly any devices actually support just main profile, it's always either baseline or high profile,** because high profile came out just 18 months after the original H.264 standard, way back in 2004, well before H.264 had really "caught on" and **way** before many hardware decoders were designed for it.

- High profile contains a couple of extra features added soon after the original H.264 standard was released, based on practical experience with it. You can **think of high profile as H.264 version 1.1**. It was originally part of the more esoteric **fidelity-range extensions**, which included the niche profiles (see below), but high profile is far more useful because it adds one key feature – the option to adaptively use an 8x8 block size for the DCT, rather than just 4x4. This yields a modest but noticeable improvement in compression efficiency, and therefore quality at a given bitrate, without significantly impacting playback complexity or CPU load. High is the profile supported on Blu-ray, the **iPhone 4S**, iPad 2 and Apple TV 3 (and later models, naturally), as well as all mainstream desktop/laptop computers: **QuickTime 7.2** (Jul 2007), **Flash 9.3** (Dec 2007), Windows Media Player 12 (Jul 2009 as part of Windows 7), or any non-ancient version of VLC, MPlayer etc (libavcodec circa 2006).

- There are also some niche profiles, such as **high10** which provides 10 bits per color component, and **high444** which provides full 4:4:4 chroma sampling. These aren't widely supported for playback, but were intended for "high-fidelity" uses such as long-term archival storage of masters. In particular, high444 has an optional lossless mode by eliminating quantization, although the resulting bitrate is huge, comparable to ProRes. Archiving using these profiles is overkill and wasteful, and probably unwise given the lack of common playback support, which is why they aren't widely used.
x264's default is high profile, except for the x264Encoder QuickTime plug-in's iPod presets which use baseline profile. Handbrake's iPhone 4, iPad 1 and Apple TV 2 presets also use high profile.

**240p/360p**: baseline profile, in order to safely play on anything, since these versions are the ultimate fallbacks and might be delivered over the web to who-knows-what device, including the original iPhone 1/3G.

**432p+**: high profile (default), even though it's not officially supported on the iPhone 4, iPad 1 and Apple TV 2, because it's known to work and produces slightly better quality (if iTunes, YouTube and Handbrake can do it, so can we!).

COMPATIBILITY: We assume the web page video embedding mechanism being used is intelligent enough to not embed a video with a resolution greater than that of the screen (with certain special exceptions, such as the iPhone 1/3G supporting 360p video on a 480x320 screen). Therefore, the above settings mean we assume any device with a screen resolution of more than 640x360 is also capable of H.264 high profile, but anything below that might only support baseline profile. This seems like a fairly safe bet, although there may be a very small number of old Android, Windows or Blackberry phones/tablets which have screen resolutions above 640x360 but only support baseline profile (do any such cases actually exist?). No new devices should be getting designed with only baseline profile support today, but if any common cases of this issue do ever occur it may be possible to detect them in web page video embedding mechanisms and only use the 360p version in that case.

QUALITY: For cases where the player supports high profile but the Internet link is slow, such as a regular laptop on a slow connection, better video quality could be achieved with a second 240p/360p version which used high profile. Unfortunately, supporting different versions of a video in the same format, at the same resolution and the same bitrate, based on some other, secondary criteria, isn't possible in most web page video embedding mechanisms. That degree of device- and feature-specific decision making would need some way to indicate the features required for each video file variant (essentially sub-formats), perhaps through a special file naming scheme, along with some kind of device detection mechanism and feature capability database on the JavaScript side, both of which are fragile and become unwieldy as the number of features and devices increases. We have to draw the line somewhere. Fortunately, over 70% of users have Internet links fast enough to get to at least the 480p version, where we do use high profile.

### H.264 Profile-Implied (various)

For 240p/360p, only using baseline profile implies disabling the following on-by-default features, which we also turn off explicitly for completeness...

- **B-frames** (frame reordering in the QuickTime dialog box) – allows B-frames (bidirectional frames).
- **Weighted predictions** (--weightp & --weightb) – allows referring to different reference frames using different weightings which are then combined, especially useful for fades.
- **8x8 DCT blocks** (--8x8dct) – allows use of 8x8 blocks for DCT transforms, rather than just 4x4 blocks.
- **CABAC entropy coding** (--cabac) – uses more space-efficient but CPU-intensive arithmetic coding for the final lossless compression step of the encoding pipeline, rather than simpler variable-length coding, for 10-20% better overall compression.

COMPATIBILITY: A handful of players don't correctly handle weighted predictions, including very old versions of Flash, CoreAVC, the MediaTek hardware decoders in some old LG, Phillips and Oppo Blu-ray players, and the Sony PlayStation Portable (PSP). Fortunately, as of mid-2010 practically all such bugs had been ironed out, except the PSP which we don't support anyway for other reasons, so we can safely use weighted predictions, which are crucial for good fades.

### H.264 Level (--level)

The H.264 level is a compatibility issue related to speed and resolution. Whereas the H.264 profile (above) defines the video compression features the player must support, the H.264 level defines the
peak bitrate the player can handle, along with the maximum resolution, and the maximum number of reference frames held in memory during playback (see later section). x264's default is to automatically set the output file's level based on the peak bitrate, resolution, number of reference frames, and other settings. There's no reason to change this setting, but it's a good idea to document the levels used.

**All:** automatic (default), which results in level 2.1 for 240p, 3.0 for 360p/432p, 3.1 for 480p/576p/720p and 4.0 for 1080p.

**Peak Bitrate (--vbv-maxrate & --vbv-bufsize)**

Limiting the peak bitrate is a necessary evil, required in order to limit the severity of spikes and overruns of the target bitrate during hard-to-encode sections of the video, such as rapid motion. Video codecs like H.264 naturally vary the bitrate over the course of the video to dedicate more bits to the frames that need them most, which is critical to achieving high quality. In a web playback scenario, however, whether it's adaptive streaming or the more usual progressive download while playing, we can't let the bitrate fluctuate too wildly, because severe spikes or sustained overruns of the target bitrate might cause playback to pause (for buffering), and pausing is much worse from an end user's point of view than a bit of blurring during a high-motion scene.

It's a tradeoff – too strict a peak bitrate limit will reduce quality a lot by removing the ability of the variable bitrate to use the bits most wisely (bitrate variations are a good thing), but too generous a limit will make pausing for buffering more likely, especially for the slower connections within each resolution's target link-speed range, because the player won't have downloaded far enough ahead to cover a big spike, even with our 20% bitrate headroom (ie: we know the user's Internet link can download at least 1.25x the target average bitrate, see earlier section on resolutions and bitrates).

x264's default is not to limit the peak bitrate at all, except for the x264Encoder QuickTime plug-in's iPod presets which limit it to 10 Mbps with a 256k buffer, which is the limit of the hardware decoder in the video iPods and iPhone 1/3G (ie: H.264 level 3.0, see H.264 level above). Most people and encoding tools recommend a peak bitrate of double the target average bitrate, a few only 1.5x, and some no limit at all.

**240p/360p:** double the target bitrate with a 256k buffer (works out to 1.8/1.14 seconds), to allow for the original iPhone 1/3G and similar devices.

**432p-1080p:** double the target bitrate with a 1.5-second buffer (1.5x peak bitrate), which is generous on the assumption our 20% bitrate headroom will cover most spikes and general fluctuations, so only really problematic, sustained overruns need to be clipped by the encoder (at a loss of quality).

**1080p Superbit:** 25 Mbps to keep within H.264 level 4.0 (works out to 1.25x the already very high target bitrate), with a 30-Mbit buffer which is the minimum size allowed for Blu-ray players and therefore should be safe for most 1080p hardware decoders (works out to 1.2 seconds).

**PAUSING RISK:** The settings for 432p-1080p are fairly generous and probably cause almost no spike clipping for most content at the encoder level, leaving only our 20% bitrate headroom. If anything, we're leaning slightly towards better overall quality at the risk of possible pausing for buffering on the very slowest links within each link-speed range. If the user was to jump into the middle of the video and land in a high-motion scene which temporarily uses double the target bitrate, the player would start playback then suddenly pause and have to wait while it buffered. A setting of something like 1.5x the target bitrate and a 1-second buffer would be a safer, more conservative setting, although even that wouldn't completely prevent the "jump into high-motion scene" risk, which is almost unavoidable, really.

**Quantizer-Curve Compression (--qcomp)**

The quantizer is the critical value in the main lossy step of quantization during video (and image) compression, and the value is varied on a block-by-block and frame-by-frame basis to control the quality. Lower quantizers remove less of the minor values in the DCT matrix of frequency coefficients for each block, preserving more of the original signal's frequency distribution, meaning
A high-quality video encoding will vary both the bitrate and the quantizer from one frame to the next. During hard-to-encode frames involving rapid motion, for example, it is necessary to use more bits, as you would expect with variable-bitrate encoding, but it's wise to also lower the quality by increasing the quantizer, rather than maintaining an unnecessarily high quality by throwing excessive bits at those difficult frames and thus wasting those bits, because the rapid motion will largely hide any improved quality, when those bits could have been used to improve quality elsewhere in other frames of the video. On the other hand, allowing the quantizer to go too high risks reducing the quality too much, making the quality drop visible and resulting in noticeably "bad" sections during hard-to-encode parts of the video, like rapid motion and fades. The human eye is particularly drawn to such "bad" sections, noticing them as glitches which stand out from the video's normal overall quality.

The tradeoff between varying the bitrate and varying the quantizer, that is, throwing more bits at difficult frames or reducing their quality instead, is controlled by the degree of "compression" applied to the quantizer curve -- the curve of the changing quantizer over time (at the per-macroblock level in x264, rather than whole frame). The setting varies from 0 to 1. At 0, the bitrate is not allowed to vary at all, producing a constant-bitrate encoding with the consequence of quality dropping severely during hard-to-encode sections of the video. At the other extreme of 1, the bitrate is allowed to vary as wildly as necessary (subject to the peak bitrate, see above) to maintain high quality during hard-to-encode sections, possibly wasting those bits because hard-to-encode sections of the video are usually rapid motion or fades, and thus by their very nature are transient, with individual frames changing so quickly that high individual frame quality is difficult for humans to see. The default value of 0.6 is supposed to represent a reasonable tradeoff, but there appears to be very little evidence to support that particular value, with it having essentially been inherited from the older XviD encoder (x264's predecessor).

A detailed discussion of quantizer compression showed that higher values than 0.6 produce considerably better results, assuming 2-pass encoding and a sensible peak bitrate setting, and our own testing easily confirms this. The difference is especially obvious for low-bitrate encodings, where the default value of 0.6 often produces "good looking static and slow motion scenes, and very bad fast moving scenes." It seems to be generally agreed that a higher value than 0.6 is better for low-bitrate encodings -- even the people who argue for the default staying at 0.6 admit that. And of course for high-bitrate encodings it doesn't really matter, precisely because they're high-bitrate so quality is far less of an issue -- it's going to look great either way, and there's likely to be no noticeable difference between 0.6 and 0.9. The bottom line is that variable bitrate is absolute magic for video encoding, and the last thing we want to do is suppress that magic.

All: 0.9.

Minimum Quantization (--qpmin)

Raising the minimum quantizer from its default of 0 (really 1) shouldn't be required, and in fact would only force lower-than-ideal quality in some areas of the frame under the rare conditions where a very low quantizer could be used for very high-detail blocks. Unfortunately, some players, such as QuickTime, have bugs playing back files with quantizers below 3, which is why the x264Encoder QuickTime plug-in defaults to a minimum quantizer of 4, and Handbrake similarly defaults to 3. Fortunately, raising the minimum quantizer to 3 has essentially no effect on real-world quality, as any quantizer below about 10 is practically lossless.

All: 3, for QuickTime compatibility.

Per-Macroblock Bitrate-Control Lookahead (--rc-lookahead)

x264 uses sophisticated macroblock-level bitrate control ("MB-tree") which tracks the degree of referencing of each macroblock through the actual motion vectors from future frames, allowing the encoder to only lower the quality in the areas of each frame which are changing rapidly (not
referenced much in the future), rather than lowering the quality of the whole frame as in most encoders – essentially traditional bitrate control but applied at the level of each 16x16 macroblock rather than at the whole-frame level. This particularly helps to maintain clear, stable backgrounds in the presence of moving foreground objects.

Using **a longer "lookahead" for this per-macroblock bitrate analysis increases quality** by allowing more effective fine-grained, per-macroblock use of the available bitrate, but **encoding will take longer and use significantly more memory**, particularly at high resolutions (there are over 8000 16x16 macroblocks in a single 1920x1080 high-definition frame, after all). x264's default is 40 frames, with diminishing returns as the distance increases to about 60 frames, and no practical gain beyond that.

**All:** 60.

### Scene-Change Detection (**--scenecut**)

Detection of **when the scene has changed, and thus when it's a good time to insert a keyframe (I-frame),** is a critical issue. x264's scene-change detection setting works counter-intuitively, the opposite of a change threshold – **higher values increase the number of scene changes detected.** x264's default is 40, but strangely the x264Encoder QuickTime plug-in's presets all use 80, which would result in a lot of unnecessary scene changes being detected.

**All:** 40 (x264 default).

### Maximum Keyframe Interval (**--keyint**)

The maximum amount of time between keyframes (I-frames) has a major impact on quality, which makes it one of the most important settings to tune, and one of the most difficult decisions. The encoder will try to use keyframes at **scene changes**, of course (see above), but for a lot of content this value is important because many scenes are longer than 5 or even 10 seconds. Having **too many keyframes severely reduces quality**, because the efficiency of reusing image areas from previous frames is completely lost at each keyframe – the encoder has to "start over" at every keyframe. Therefore, we want as few keyframes as possible to achieve the highest quality for the given target bitrate.

On the other hand, we **still want enough keyframes that seeking and fast-forwarding behavior is good**, because players can only jump directly to keyframes "under the hood" during playback, and will usually only display the keyframes during fast-forwarding and rewinding at higher speeds (at low speeds such as 2x or 3x they can often play every frame). Jumping to an arbitrary point in the timeline therefore becomes more sluggish the fewer keyframes there are, because more intervening delta-frames need to be decoded just to reconstruct the final target frame, even though those intervening frames between the previous keyframe and the target frame won't actually be displayed. If the video is deployed using **adaptive streaming**, where the player might dynamically switch between different versions during playback based on the available network bandwidth, then such switching can also only occur at the keyframes (of the stream being switched to), so again we don't want the keyframes too far apart.

x264's default is 250 frames (8.3 seconds at 30fps), but for no good reason the x264Encoder QuickTime plug-in's default is only 60 frames (2 seconds). The x264 team have recently been using 500 frames (16.7 seconds) for their settings at the annual MPEG-4 AVC/H.264 Video Codecs Comparison competition, but that's definitely pushing things a bit too far. Digital TV normally uses 1 or 2 seconds, but that's deliberately quite short to make channel switching fast and to have quick error recovery in case of interference. DVD uses a very short ~0.5 second keyframe interval, and Blu-ray 1 second, because they use very high bitrates (so quality isn't an issue) and they want to guarantee good fast-forwarding behavior despite being read from a relatively slow optical disc. The most common recommendation for Internet video is 10 seconds.

**All:** 12 seconds (360 frames at 30fps), which is enough to cover the vast majority of scenes and result in only "natural" scene-change keyframes, and about as far as we can push it. Fast-forwarding while seeing only one frame for every 12 seconds of video is just barely okay, as a worst-case scenario. Assuming fast-forwarding at 10x speed, that means 6 seconds per minute with 5
keyframes, resulting in 1.2 seconds on-screen per keyframe. Jumping to arbitrary points in the timeline definitely feels sluggish as well, but not too painful, even at 1080p, and it will get better in time with faster computers.

**Minimum Keyframe Interval (--min-keyint)**

The H.264 format actually supports two different types of I-frames – traditional keyframe "IDR" I-frames which represent suitable restart points (IDR stands for "instantaneous decoder refresh"), and other, lesser, non-IDR I-frames in which frames after the non-IDR I-frame can still refer to frames before the I-frame, meaning the non-IDR I-frame can't be used as a restart point, but it's still encoded as an I-frame! This latter case is only really useful for handling extreme flashes and other sudden, single-frame mass changes, which are very rare.

The minimum keyframe interval controls which of these two types of I-frames is used at each occurrence, by preventing full IDR I-frames from being placed closer than this number of frames apart. x264's default is 10% of the maximum keyframe interval, which would often be about 1 second, and makes sense since there's no need to have two restart points closer than that. Unfortunately, it's known that some players have trouble when fast-forwarding or scrubbing in a video with non-IDR I-frames, including some recent versions of QuickTime and Flash, which is why the x264Encoder QuickTime plug-in's default is 1, making every I-frame a full IDR I-frame.

All: 1 (x264Encoder default, not x264 default), for compatibility with QuickTime, Flash and various other players.

**Motion-Estimation Search Pattern (--me)**

Ultimately, high-quality video encoding at lowish bitrates is all about finding similar image areas in previous frames, and reusing them. The search pattern is the pattern used during motion estimation to search for the most similar area to each macroblock in each possible reference frame, in order to select the best motion vector for each macroblock. This is where a great deal of encoding time is spent. More thorough search patterns will find better matches, producing better motion vectors, leading to a less complex residual image left to encode after motion compensation, and therefore better quality at the given target bitrate. Of course, a more thorough search also takes a lot longer during encoding.

The simplest search pattern is a straightforward 4-point diamond shape (left/right/up/down), with the next simplest a 6-point hexagon, then a complex uneven multi-hexagon (UMH), and finally full-blown exhaustive search (which is very slow indeed).

x264's default is a simple hexagon, but it's widely acknowledged that x264's implementation of uneven multi-hexagon search is one of its biggest advantages over other encoders, and usually achieves within 0.5% of full exhaustive search. The quality-oriented presets, including the x264Encoder QuickTime plug-in's "optimized" presets, all use uneven multi-hexagon search.

All: uneven multi-hexagon.

**Motion-Estimation Search Range (--merange)**

As mentioned above, in the end, high-quality video encoding at lowish bitrates is all about finding areas of similarity and reusing them. The larger the search area, the more likely the encoder will find a good match, leaving a less complex residual image to be encoded and therefore producing better quality at the given target bitrate, but the larger search will also take much longer to run. High-definition material benefits from a larger search range more than lower resolutions, naturally, because the same camera or object movement covers more pixels in a high-resolution situation. x264's default is 16 pixels, but note that for uneven multi-hexagon search the search pattern is changed and iterated at several different levels, so the range isn't literally 16 pixels exactly.

Increasing the search range suffers from severely diminishing returns, greatly increasing encoding time for no visible gain in most cases, since areas further away are unlikely to be very similar. On the other hand, a large search range is most useful for high-motion scenes, which are precisely the
scenes that are the hardest to encode, where a noticeable drop in quality is most likely and the need to find any available similarities is the most pressing. So a large search range, while generally of little benefit, is vitally important in those few critical times of rapid motion which make such a big difference to the overall perceived quality of the encoding ("It's not how well you do the easy stuff, it's how well you do the hard stuff").

240p-720p: 32, which is excellent coverage at 720p and overkill for the lower resolutions, but we need every scrap of similarity we can find at the lower resolutions because of their lower bitrates, and encoding is relatively fast at low resolutions anyway.

1080p: 48, to account for the same relative movement within the 1.5x1.5 times larger frame.

**Sub-Pixel Motion-Vector Refinement (--subme)**

Motion-vector refinement controls how much time and effort the encoder should put into macroblock partitioning decisions and final motion-vector refinement for H.264's quarter-pixel motion vectors. A more thorough evaluation of the possible final motion vectors will find better matches, producing better motion vectors, leading to a less complex residual image left to encode after motion compensation, and therefore better quality at the given target bitrate. Naturally, a more thorough evaluation will also take a lot longer during encoding. Ultimately, this setting and the previous two (search pattern and search range) are where the "rubber meets the road" in terms of the quality-vs-encoding-time tradeoff. This is where we pay for the higher quality of our encodings.

Settings 1-5 don't use rate-distortion optimization and are intended for fast, lower-quality encoding situations such as live video conferencing. A setting of 6 enables rate-distortion optimization for I- and P-frames, 7 adds RDO for B-frames, 8 enables RDO refinement for I- and P-frames, 9 adds RDO refinement for B-frames, and a setting of 10 enables quarter-pixel RDO refinement for all frames (requires full RDO, see later section). x264's default is 7.

All: 10.

**Predicted Motion Vectors (--direct)**

The H.264 format allows the encoder to use predicted motion vectors instead of actually encoding each vector explicitly, which saves some bits and thus slightly improves quality. The prediction can be either spatial (predict from neighboring blocks) or temporal (predict from previous frames). x264's default is spatial, but it also offers an automatic mode which selects the best choice for each frame (requires Z-pass encoding).

All: automatic.

**Number of Reference Frames (--ref)**

Unlike previous codecs, H.264 supports multiple reference frames, so each macroblock in each P- or B-frame can refer to a different reference frame with its motion vectors, allowing better matches to be found, resulting in a less complex residual image to encode and thus better quality at the given target bitrate. Allowing just 2 reference frames yields a big quality improvement in certain special cases, such as sudden large-but-temporary changes in the frame. This essentially solves the "camera flash" problem that plagued earlier codecs such as MPEG-2 when dealing with content such as celebrity press events, since the problematic flash frame's content doesn't have to negatively affect future frames, which can now point to the frame before the flash instead.

Increasing the number of reference frames beyond 2 allows even better matches to possibly be found, but naturally suffers from severely diminishing returns after 3 or 4 frames, since frames further away in time are likely to be more and more different and therefore not very useful for finding similarities. Increasing the number of reference frames also dramatically increases the encoding time, since motion-estimation search, which is the slowest part of video encoding, has to occur on all possible reference frames in order to find the best match.

Hardware decoders place a limit on the maximum number of reference frames allowed during playback, indicated by their supported H.264 level (see earlier section). Although the limit on the
number of reference frames notionally relates to "video memory size", in reality any limit on the H.264 level is actually more of a general indicator of decoding speed, based on the expected performance of the memory system under a workload with the given number of reference frames. The more reference frames, the lower the memory locality (reuse) and the larger the active working set, meaning more cache misses and more time spent waiting for main memory, which substantially reduces performance. The hardware's memory latency and bandwidth play a big part here, as does the size and speed of the L2/L3 cache for a regular CPU, or the size of the internal scratch RAM in an SoC video-decoder block. No normal CPUs or hardware SoC decoders currently have enough on-chip memory to avoid the use of main memory altogether at the higher resolutions, except for a GPU-based player and its local video memory on the graphics card (which will have way more than is needed).

Unfortunately, some popular hardware decoders don't smoothly play videos which use the decoder's claimed maximum number of reference frames (eg: the NVIDIA Tegra 2 processor under old versions of Android), resulting in either no playback or severe stuttering during playback if the number of reference frames is too high. This is usually due to inadequate cache size and/or a lack of suitable prefetching to cover main-memory latency. Some software players also struggle with large numbers of reference frames, for similar reasons.

x264's default is 3 reference frames, which is also what YouTube uses. iTunes uses 4 for 1080p, but only 2 for 720p, presumably for compatibility with older, slower computers (Apple also avoids CABAC at 720p, presumably for the same reason, and uses a high bitrate to compensate). Taking this to an extreme, iTunes uses just 1 reference frame for 480p, which is ridiculous since playing 480p H.264 baseline video should be relatively easy for any modern computer. Perhaps Apple are just being ultra-conservative in case something becomes a problem in the future, and they want their fallback SD videos to be as undemanding as possible in terms of required performance, but not using even 2 reference frames is taking things way too far!

All: 4, which should find about as much useful similarity as there is to find, doesn't blow out the encoding time ridiculously, complies with the target H.264 levels, is safe for almost all known hardware decoders (assuming Android 3.1+ on Tegra 2 processors), and shouldn't really stress software players too much.

COMPATIBILITY: The Sony PlayStation Portable (PSP) is a significant device which does not play H.264 video with our chosen 4 reference frames. Given its 480x272 screen resolution, however, combined with the PSP's various video playback limitations, the only version of our videos the PSP could possibly play would be the lowest-quality 320x240 version for non-widescreen content, and no version at all for the by-far more common case of widescreen content. We therefore simply do not support the PSP as a target for our videos.

PERFORMANCE: Testing shows reducing the number of reference frames down to just 2 doesn't significantly improve the playback speed of notoriously slow software players, such as QuickTime 7 on Windows. The difference is barely measurable, a few percent at most, perhaps making a stressful, high-motion 720p HQ video play at 21fps rather than 20fps (where other software players achieve the full 30fps on the same system, as do all hardware decoders). Given the loss of visual quality, the widespread use of hardware-accelerated playback in practice, and thinking long-term, a small gain in playback speed on "bad" players isn't worth it, since it doesn't really fix the problem – playback is still not acceptably smooth, and most people wouldn't notice any real difference in smoothness.

**B-Frames as Reference Frames (--b-pyramid)**

The H.264 format allows B-frames to be used as references for other frames (why wouldn't it?), which is sometimes called B-frame pyramiding because a "pyramid" of B-frames forms, with lower
B-frames pointing up to higher B-frame(s), which then point to one or more P-frames and ultimately to an I-frame (keyframe).

Unfortunately, some players don't correctly handle B-frames as reference frames, causing visual corruption during playback. This is usually because the computationally-intensive step of deblocking during playback (see later section) is being performed "less fully" than it should be on B-frames (the majority of all frames) to save time, or sometimes even skipped altogether. As a result, the not-quite-properly-decoded B-frames don't contain exactly the pixels the encoder expects them to (and the H.264 format officially requires them to!), resulting in incorrect visual results for any subsequent frames which reference B-frames, typically ugly smearing/tearing.

Sadly, the Blu-ray specification doesn't require B-frames to be supported as reference frames, even though that is part of the full H.264 specification, and as a result, some Blu-ray hardware decoders don't support it. Some software players also provide an option to reduce or turn off deblocking for B-frames in order to improve performance. Thus, we cannot rely on that part of the H.264 standard being reliably implemented, and must instead avoid using B-frames as reference frames. Fortunately, the loss of quality caused by not using any B-frames as reference frames is fairly minor, as a similar I/P-frame is almost always available instead.

All: off, for compatibility with bad players that don't apply proper deblocking to B-frames.

**P-Frame Early Skip Detection (--fast-pskip)**

P-frame skip detection allows the encoder to skip consideration of macroblocks in P-frames at an early stage of processing if the change is extremely small. This speeds up encoding by about 20% on average, with practically no quality loss in almost all cases using sane bitrates. To quote one of the x264 programmers: "no-fast-pskip does practically nothing. It is a placebo option, and if you think it does something significant, your eyes are probably fooling you and you should stop messing too much with settings you don't understand." x264's default is therefore P-frame skip detection turned on, although the x264Encoder QuickTime plug-in defaults to having it turned off, probably from an old bug where P-frame skip detection was a little too aggressive and didn't handle blue sky areas well, a bug which has long since been fixed.

240p-1080p: on (default).

1080p Superbit: off, to ensure we capture every last tiny detail since this version acts as a long-term master for future re-encoding.

**Adaptive Number of B-Frames (--b-adapt)**

The x264 encoder supports using an adaptive number of B-frames rather than just a fixed pattern like IBPBBPBBPBB, and this setting controls that adaptive decision. x264's default is a fast, simple algorithm (1), but it also supports a slower, higher quality, "optimal" algorithm (2). The slow algorithm is the default for the quality-oriented presets, including the x264Encoder QuickTime plug-in's "optimized" presets. Strangely, the x264Encoder QuickTime plug-in's iPod presets set this to 1 even though they use baseline profile and therefore don't use B-frames.

240p/360p: 0 (B-frames disabled by baseline profile).

432p+: 2 (slow, "optimal").

**Maximum Number of B-Frames (--bframes)**

The x264 encoder will adaptively decide when to use B-frames and how many to use (see above), up to a given limit. Allowing longer sequences of consecutive B-frames is good for quality because B-frames are the most efficient frame type in terms of compression, but considering large numbers of B-frames will slow down encoding significantly, with diminishing returns because the encoder will rarely choose to actually use more than 4 or 5, with 1-3 being much more common.

Long sequences of B-frames also risk a growing propagation of error, resulting in a gradual degradation of quality followed by a slightly visible "pulse" when the video switches back to a higher
quality P- or I-frame. This is similar to the "keyframe pumping" smearing-then-pulse problem seen in the older DivX/XviD codec (H.264's predecessor) during moderate-motion content encoded with very long sequences of P/B-frames caused by an excessively long keyframe interval, although the DivX/XviD problem is much more severe because the keyframe interval is typically at least several seconds, not a fraction of a second, leaving much more time for error-buildup smearing to occur, making the following "pulse" jump in quality far more noticeable.

Note that with a setting of 2, about 67% of all frames will be B-frames, with 3 that rises to a maximum of 75%, 4 to a maximum of 80%, and 5 to around a maximum of 83% if the encoder was actually to frequently use 5 consecutive B-frames, which is unlikely. In other words, the B-frame type dominates all other frame types in all cases (IBBBPBBBPBPPBPPPPPBB), and this setting only increases it relatively slightly as a percentage. x264's default is a maximum of 3 B-frames. YouTube uses just 2 B-frames, presumably for faster encoding. Jan Ozer recommends 2 or 3 (why 2? just encoding time?). Most of the x264 programmers recommend a higher setting when targeting maximum quality, some as high as 8 or even 16, although that does slow down encoding a lot for no real-world gain – an increase in the use of B-frames by less than 1% isn't enough to have a visible effect on quality. Some hardware decoders are known to have problems with long sequences of B-frames, such as certain older ATI/NVIDIA GPUs and the NVIDIA Tegra 2 processor used in many early Android phones and tablets.

240p/360p: 0 (B-frames disabled by baseline profile, displayed as 1 in x264Encoder).

432p-1080p: 3 (default), for compatibility with older ATI/NVIDIA GPUs and Android devices based on the Tegra 2 processor, and possibly other hardware decoders with similar bugs (3 B-frames is a pretty "standard" setting, and a lot of people use it, so it should be very unlikely to break anything).

1080p Superbit: 5, which is about as many as might ever be used in practice.

**Rate-Distortion Optimization (--trellis)**

Rate-distortion optimization (aka: trellis) is a slow but effective brute-force optimization technique which uses a video quality metric to exhaustively measure both the distortion from the original master and the actual cost in bits for each possible choice, processed right through to final entropy encoding, essentially considering every possibility and choosing the best one according to cost-vs-benefit, as indicated by the RDO metric being used. x264 uses a very good, "psycho-visual" RDO metric by default (see below), but can also be instructed to use simpler metrics like PSNR or SSIM.

Rate-distortion optimization was originally targeted at optimizing quantization (--trellis=1, x264's default), but it can also be used from earlier in the encoding pipeline, starting at motion-vector refinement and including macroblock partitioning and block-type decisions (--trellis=2). Rate-distortion optimization was previously not supported for baseline profile in x264, because it only worked with CABAC final entropy encoding, but that's no longer the case in modern versions of x264.

Although RDO does select the optimal choice among the possibilities for things such as macroblock partitioning and quantization, the choice is only optimal in the local sense, for just that one macroblock. It doesn't take into account the overall, more global situation, including where and how that macroblock might be referenced by other blocks in this frame or future frames. Thus, using the term "optimal" for RDO is common but inaccurate. The RDO metric is not a perfect match to human perception either, but even if it was, and even if an exhaustive motion-vector search with infinite range was used, RDO would still not guarantee a truly optimal encoding for even the current frame, let alone the entire video. Nonetheless, RDO does produce very good choices, probably close to optimal in most cases.

**All:** 2 (on for motion-vector refinement, macroblock partitioning and quantization).

**Psycho-Visual Optimization (--psy-rd)**

One of x264's key advantages over other encoders is its use of psycho-visual optimization to improve subjective quality. Psycho-visual optimization tries to better match the human visual system's perception and interpretation of images. The theory is that our brain is geared in such a way that
it doesn't simply want the image to look similar to the original, it wants the image to feel like it has a similar level of complexity – otherwise it looks "worse" even if it's technically closer to the original, numerically speaking.

In other words, we humans would rather see a slightly distorted but equally detailed area on an image than a non-distorted but slightly blurred area. Thus, this optimization works by altering the rate-distortion metric to de-emphasize blurry "low-error but low-energy" choices during quantization, macroblock partitioning and so on, rather than using simpler metrics like peak signal-to-noise ratio (PSNR) or structural similarity of images (SSIM), which tend to lean towards less actual numeric pixel difference but too much blur.

The strength of the psycho-visual optimization effect can be tuned from its default of 1.0, either increased up to around 1.5 to lean towards even more detail, texture, film grain, and ultimately noise artifacts, or decreased to around 0.5 to lean towards more smooth, flat, and ultimately more blurry output. There is also a psycho-visual trellis setting, which is currently disabled (default 0.0) because it's still considered experimental, but which also should default to 1.0 when activated in the future, and seems to work well both for grainy film-based content and for clean computer graphics.

All: 1.0/1.0 (future default, probably).

**DCT-Based Decimation (---dct-decimate)**

DCT-based decimation allows the encoder to skip encoding blocks it deems unnecessary based on a simple DCT threshold test, because the residual to be encoded is very small, and therefore probably not visually noticeable. This speeds up encoding by avoiding the need to perform slow rate-distortion optimization on blocks which will probably get no bits assigned to them anyway, and thus causes very little quality loss in most cases. It also tends to stabilize areas of the image that are changing in very minor ways, which can be a good thing visually if those minor changes aren't "real".

Naturally, it's better for quality to let rate-distortion optimization decide which blocks should be skipped or quantized more aggressively by giving it all the blocks to consider, as RDO will produce an "optimal" choice without relying on an arbitrary (even if very good) threshold test, but DCT-based decimation can often increase encoding speed very substantially with no visible quality loss, especially in cases with stable backgrounds, like "talking head" shots.

Unfortunately, DCT-based decimation can occasionally cause skipping of subtle changes during slow fades or very dark scenes, leading to banding or stepping even at high bitrates. Film grain or camera sensor noise is usually enough of a change that DCT-based decimation won't skip it, assuming the resolution and bitrate are high enough to capture that level of detail, but dark scenes where the grain is invisible, or very clean sources like computer graphics can be a problem for DCT-based decimation. x264's default is to have DCT-based decimation turned on.

All: off, to preserve good slow fades in clean-source situations like computer graphics.

**4x4 Partitions in P-Frames (---partitions p4x4)**

Taking the time to consider the possibility of using the small 4x4 macroblock partition sizes (4x4, 4x8 and 8x4) is known to be useful in I-frames but of little benefit in P-frames (and not supported at all for B-frames in x264). It greatly slows down encoding for an insignificant gain in quality at most normal resolutions, although it can improve quality slightly at low resolutions if the bitrate is high enough to take advantage of it to capture more fine detail. To quote one of the x264 developers: "Its (sic) pretty much useless except at relatively high bitrates, and even then its only particularly helpful at low resolutions, and even then its not that great." x264's default is not to consider 4x4 partitions in P-frames.

240p/360p: on, because we need everything we can get at the very low resolutions/bitrates, especially given we're stuck with H.264 baseline profile at those resolutions, and the smaller block sizes might be slightly useful at low resolutions, plus the encoding-time cost is still reasonable at low resolutions.

432p+: off (default).
Deblocking (--deblock)

Unlike earlier codecs, H.264 expects the player will perform some form of deblocking during playback, blurring the edges of macroblocks to avoid visible blocks in the final image (blur is less noticeable to the human eye than false edges). In fact, **while some good players for earlier codecs could optionally apply deblocking, H.264 actually requires it, taking advantage of that fact to improve overall compression** significantly by using the deblocked frames as the reference frames for future P- and B-frames. Thus, H.264 is sometimes referred to as having "in-loop" deblocking, because it's a mandatory part of the official playback process, not an optional extra.

Deblocking is one of the more computationally-intensive parts of H.264 playback, and while officially it can't be skipped or done "less fully" to save time, **some players perform lesser deblocking on B-frames (the majority of all frames), which is why those players don't support using B-frames as reference frames** for other frames (see earlier section).

The strength and threshold of the deblock smoothing can be set during encoding...

- **Strength**, also called alpha, controls the amount of smoothing along the edges of blocks, with the default of 0 good for most situations. Positive values up to about +2 apply more smoothing and ultimately soften/blur the image, while negative values down to about -3 apply less smoothing and preserve more sharpness/detail, at the risk of some visible blocking artifacts. Nothing below -3 should really be considered.

- **Threshold**, also called beta, determines how flat the blocks must be in order for smoothing to be activated, with 0 a good default which rarely needs to be changed. The actual threshold calculation is complex – to quote an x264 developer: "There is a threshold based on [qp + 2*alpha] and another threshold based on [qp + 2*beta]. The texture/gradient/whatever must pass both thresholds before any filtering is applied." At most, the beta threshold should only be changed very slightly, but a small change to -1 can improve overall sharpness at the risk of some visible blocking.

**240p-1080p**: 0:0 (default).

**1080p Superbit**: -2:-1, to preserve as much sharpness as possible since we have a very high bitrate so there's no risk of visible blocking artifacts.

PERFORMANCE: Reducing the amount of deblocking does improve the playback speed of notoriously slow software players, such as QuickTime 7 on Windows, but the difference is only modest, a few percent, even with the beta threshold dropped to -2. Given the visual ugliness of blocking artifacts, the widespread use of hardware-accelerated playback in practice, and thinking long-term, a small gain in playback speed on "bad" software players isn't worth it.

Colorsce Tagging

A color specified using simple numeric values such as (175, 0, 0) alone is actually somewhat ambiguous. Sure, it means 175/255ths of maximum red, but of what maximum red? Its actual appearance will depend on the screen's brightness, gamma curve (brightness response curve), color saturation, color gamut (overall color range), color temperature (warmth/coolness), white point setting and so on. Thus, we are forced to use the notion of color spaces, saved in files known as color profiles, which should be used to tag all photographically-oriented images and videos, so they can be displayed correctly and look more-or-less the same on different screens.

The main exception to colorspace tagging is images used on websites, where you should convert the images to the web's standard sRGB colorspace, and **not** embed a color profile, in order to guarantee consistent colors in web browsers that aren't colorspace-aware, as well as to reduce file size (embedding a color profile adds an extra 2-10k, which is a lot of overhead for small web

Colorspace and/or gamma tagging for video files inserts a colr (modern) or gama (old, deprecated) tag into the output QuickTime file's header, and a **VUI parameter into the actual H.264 bitstream itself**, to indicate the file's colorspace. Modern, colorspace-aware video player applications can then accurately map the file's colors and brightness levels to the playback screen in order to display the video so it looks more-or-less the same as you intended, and the same on different screens.

Gamma-only tagging is now obsolete and not supported by the VUI parameter of the H.264 format, and any QuickTime-level
gamma tagging is lost in the conversion to the recommended .mp4 container file format, so only modern colorspace tagging is relevant nowadays.

Modern video-file colorspace tagging uses 3 numbers, each an index into a table of known broadcast standards: the RGB color primaries to define what exactly "red", "green" and "blue" really are; the transfer function (the modern equivalent of gamma); and a conversion matrix. Together, these form a "non-constant luminance coding", or NCLC.

Unfortunately, many current video player applications don't respect colorspace tagging, including Flash, Windows Media Player (most configurations, sometimes overridden by GPU drivers), VLC and MPlayer, all of which simply send the colors through either numerically unchanged or using their own, internal, usually fixed and usually "unique" color/gamma correction.

After a considerable amount of experimentation with a variety of player applications, on a variety of different operating systems, phones, tablets and standalone TV screens, and after lots and lots and lots of reading on the subject, there are only 3 colorspace taggings which really matter...

- **No tagging** (default) declares nothing about the video file's colorspace, and leaves it at the mercy of the video player application's whims, which isn't a great idea, even though that's going to be the case for many players anyway since many ignore colorspace tagging.

- **SMPTE-C colorspace (NCLC 6-1-6)** is the current version of the old NTSC standard-definition TV colorspace, also known as "recommendation 601". It has an official white point of D65 (6500K) and a gamma of 2.2, although most consumer NTSC CRTs were usually somewhat darker with a gamma of 2.3-2.5 in practice (even good ones). As a result, many older player applications that aren't colorspace-aware will assume video is encoded expecting a gamma of around 2.3-2.5, and if the screen isn't that dark (or the player simply assumes the screen's gamma is 2.2) the player will darken the video to (incorrectly) compensate – so colorspace-unaware players will often show video a little darker than they should.

- **HDTV colorspace (NCLC 1-1-1)** is the colorspace standard for modern high-definition content and screens, also known as "recommendation 709". Like SMPTE-C, it has an official white point of D65 (6500K) and the gamma is approximately 2.2 overall, though it contains a small linear section near black. This time around, however, the gamma of 2.2 is actually mostly reflected in practice on good LCD/plasma screens, allowing better shadow detail. HDTV also defines slightly different, slightly better red/green/blue color primaries, allowing for slightly more saturated colors. The HDTV colorspace has a very well-specified sibling in the form of the standard sRGB colorspace, which uses the same red/green/blue color primaries and white point as HDTV, with roughly the same approximate gamma of 2.2 (though there are some minor, largely insignificant differences in the precise gamma curves).

It's important to note the SMPTE-C and HDTV colorspaces are slightly different, and in fact different enough to be clearly visible when viewed side-by-side or flipped between (you can read a brief summary of the differences, or more detailed coverage). Nonetheless, it's safe to assume most modern TV screens aspire to match the HDTV/sRGB colorspace as closely as possible, and

The widespread missing colorspace-awareness functionality in video player applications is the main cause of the many articles about "QuickTime's gamma bug", which is ironic since QuickTime Player is one of the few players that does respect colorspace and gamma tagging. The "bug" is actually more a lack of configurability, because the QuickTime H.264 encoder tags all SD-sized encodes with the SMPTE-C colorspace (NCLC 6-1-6), and all HD-sized encodes with the HDTV colorspace (NCLC 1-1-1), with QuickTime performing color conversion as required during the export, so that they end up looking the same in a colorspace-aware player, but different in a player which ignores the colorspace tagging! The QuickTime H.264 encoder should provide an option to set the output file's colorspace explicitly (as x264Encoder does), but sadly it doesn't – the tag itself can be changed by other tools after-the-fact, but the colors will already have been converted into the new colorspace. Nevertheless, QuickTime Player and the QuickTime plug-in are actually doing the "right thing" by respecting the tagging during playback. It's just the rest of the world is doing the wrong thing, including some of Apple's other applications.
assume older SMPTE-C content will map acceptably to the new colorspace – at worst it would look a little washed out (what was that joke about NTSC meaning "never the same color" again?).

Modern computer screens are not as well standardized, but are based on the same technology as LCD HDTVs. All of the major operating systems recommend a gamma of 2.2, although the exact value will vary slightly based on the screen's actual color profile. The red/green/blue primaries and color gamuts of computer screens vary quite considerably, and the color gamut is often somewhat worse than HDTVs, at least good ones, due to the constraints of embedding a good screen with a good backlight into a thin, light laptop enclosure. Again, however, it's safe to assume most computer screens attempt to more-or-less match the HDTV/sRGB colorspace. Most phones and tablets also attempt to adhere to an approximate gamma of 2.2 and similar color primaries to HDTV/sRGB, although once again there's considerable variability among real-world devices.

*All:* HDTV (NCLC 1-1-1), keeping in mind some players and screens will display the video a little more darkly.