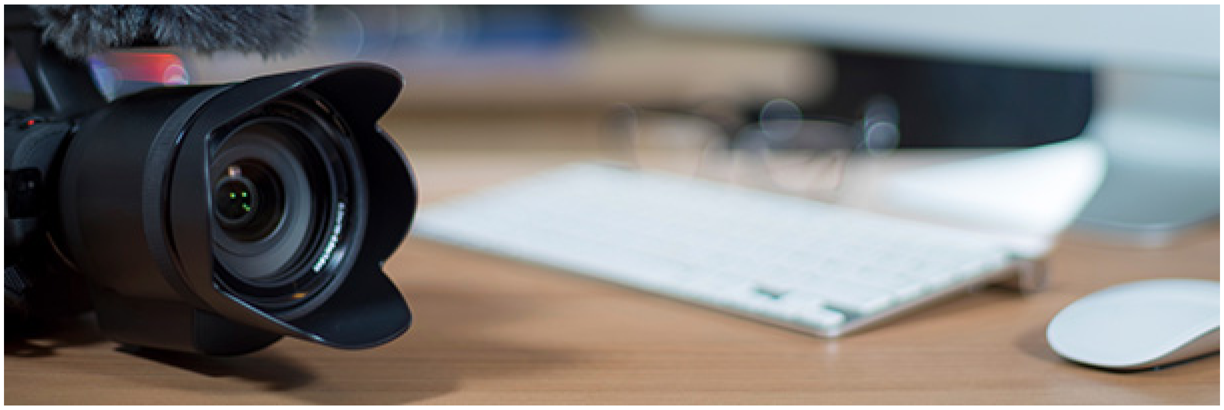
Video Compression Secrets - Smaller Files, Better Quality UPDATED (Oct 15)



by Stephen Haskin, October 19, 2015

"The best codec to keep file sizes small is Windows Media (WMV). Microsoft has done an excellent job of combining small file size and very good temporal quality. The format isn't necessarily good for iOS devices, but if you can manage to get your projects on YouTube, you will be able to deliver everywhere. And that's a plus, in my opinion."

In 2012, YouTube reported that users were uploading 48 hours of video to the service *every minute*. That's *over 69,000 hours of video a day*, every day. Today, in 2015, YouTube is reporting that *300 hours of video are uploaded every minute.* That's equal to an almost unimaginable number of petabytes of video uploaded every day (a petabyte is a million gigabytes). YouTube has well over 10 million servers! The only way this works is because YouTube compresses all that video when you upload yours. But you aren't dealing in petabytes of video so why does this matter to you?

Most of us already use video in our eLearning. In the "most of us" category, you are compressing your video whether you know it or not. While some of us use our own servers for video rather than YouTube's, Vimeo's, or some other video service, we all have file-size limitations and bandwidth issues.

How you compress your video and what form you compress it to makes a difference. But what do we know about compressing video except to use the Adobe Media Encoder or a standalone program like Sorenson Squeeze (or DivX, Microsoft, Apple, etc.) and several other video compressors? Does the file type you create matter? Yes. Does the file size matter? Yes. Let's talk about file size first.

What affects file size?

Any timeline-based project (Figure 1) must be compressed in order to put it on YouTube, Vimeo, your servers, or wherever you need to store and access it. Furthermore, while you can stream an uncompressed video file (AVI), it's not easy. The bandwidth requirements are such that your IT department will not have anything nice to say about it.

The video you upload to YouTube gets compressed the way YouTube wants to compress it, no matter how you compressed it originally. Although the server sizes seemingly approach infinity at YouTube, when it comes to uploading, the only worry about file size is how long it might take to upload a large file. It's the streaming part that matters. YouTube, and a few of the other video services, also automatically deliver different video codecs to different platforms like iOS or Windows. If you're delivering to multiple platforms, it's worth a look.

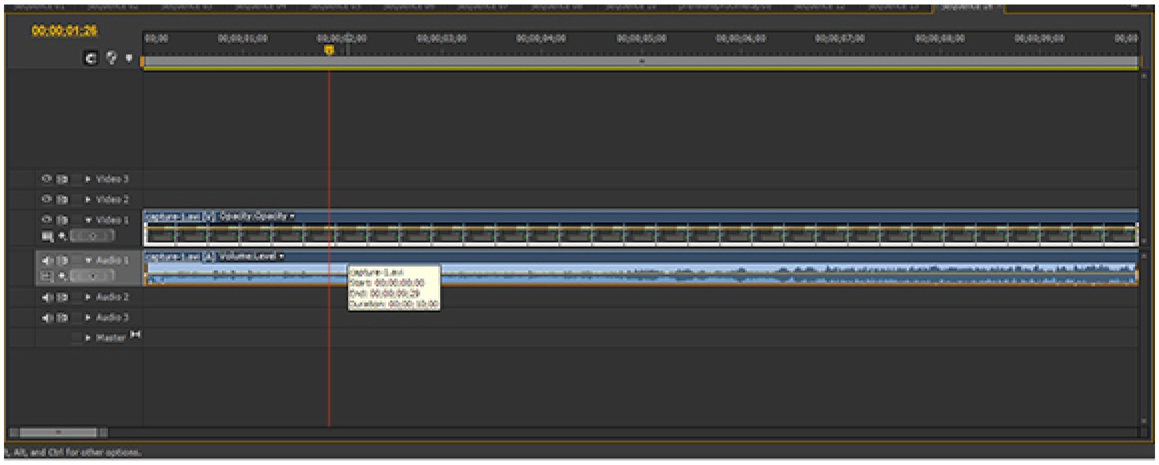


Figure 1: **Timeline sample**

Video file size depends on many variables: high definition (HD) vs. standard definition (SD), frame rate, color depth, and even the amount of movement in the video. There are three measurements that count for a lot in determining the final file size.

First, the actual pixel dimensions of the finished product (Figure 2). Standard video (SD) is generally 720 X 480 pixels—that equals 345,600 total pixels per frame. If you can do native HD video, the dimensions are 1,920 X 1,080, which equals 2,073,600 pixels. That's a lot more pixels per frame to stream.

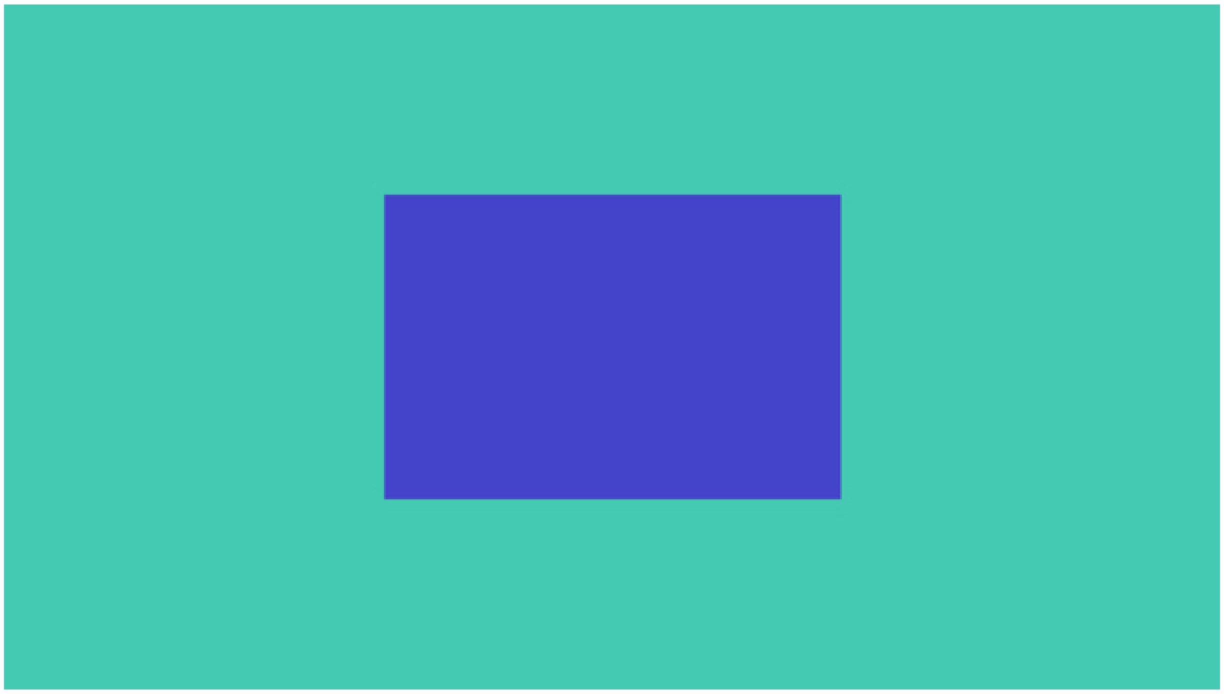


Figure 2: **Standard video frame size (blue center area) vs. native HD video frame size (green outer area)**

The second measurement is the length of the video. When anything changes on the video timeline you're creating, the file size changes after you render your video. Change something else on your timeline and the resulting file size changes again. And so on.

The third measurement is frame rate. Most of the video we do is about 30 frames per second (fps). If you cut the number of frames in half, the video file size will be cut in half.

There are more variables, though. Here are some of them, including the three just mentioned:

- Pixel dimensions of the video
- Frame rate (15, 24, 25, 30, or whatever)
- How frequent are your key frames (the frame where the entire frame is recorded)
- Progressive or interlaced frames
- Constant or variable bitrate streaming
- Buffer size
- Audio sample rate
- Render quality

There's even more, if you want to dig into it. No wonder video quality can be so variable…and it's very confusing to have to finish your video in order to stream it.

An explanation of containers and codecs

When it comes to your rendered video, there are two kinds of software in play. The first is the development environment or format. This is a container that the server sends to the browser. The viewer's browser needs to have the correct decoder add-on installed in order for viewers to see the video. There are several of these containers, but these days we use H.264, which is the MP4 format. But there's still a lot of QuickTime (MOV), or Windows Media Video (WMV) as well. Both QuickTime and Windows also use the MP4 codec (see below) as well. It can be confusing, and to be sure there are more, but I'm going to keep this explanation simple to keep the complexity down.

The second part of a video is the *codec* inside the format. A codec consists of two parts—the encoder, which is what you use while you're encoding the video from your timeline or file. The second part is the decoder; this part resides on the viewer's computer as an add-in to decode the video inside the container. The codec is what actually encodes and compresses the video when you're rendering it.

A very common codec is H.264. It's called a block codec because it looks for differences in blocks of video as it encodes and plays back. H.264 is used for HD video compression. Now, there is no reason H.264 can't compress standard-definition video—note that this is true for all codecs. Most of my clients work in SD video or NTSC to keep streaming rates lower. You should also note that if your finished video is going into an Articulate Studio or Storyline project, Articulate will not show HD video...yet. In fact, the dimensions for Articulate have been 640 X 480, an unusual size that fills the template screen.

A second widely used codec is Windows Media. This codec can be used with H.264 and is the standard codec for Blu-Ray discs. Really. It can have other codecs in it besides H.264, but for purposes of this article I'm going to use the WMV because it is easy to encode from the timeline in Premiere Pro.

Conceptually speaking, compressing video is actually pretty simple. If you look at a video from one frame to the next, some pixels change and some are exactly the same as in the frame before. Why encode a pixel that's exactly the same as a pixel in the frame before? That's a waste of bytes. So the software essentially ignores the pixels that don't change and changes the ones that do change. (If only it were that simple. There are very complex algorithms that go along with this, but the essence is the same.)

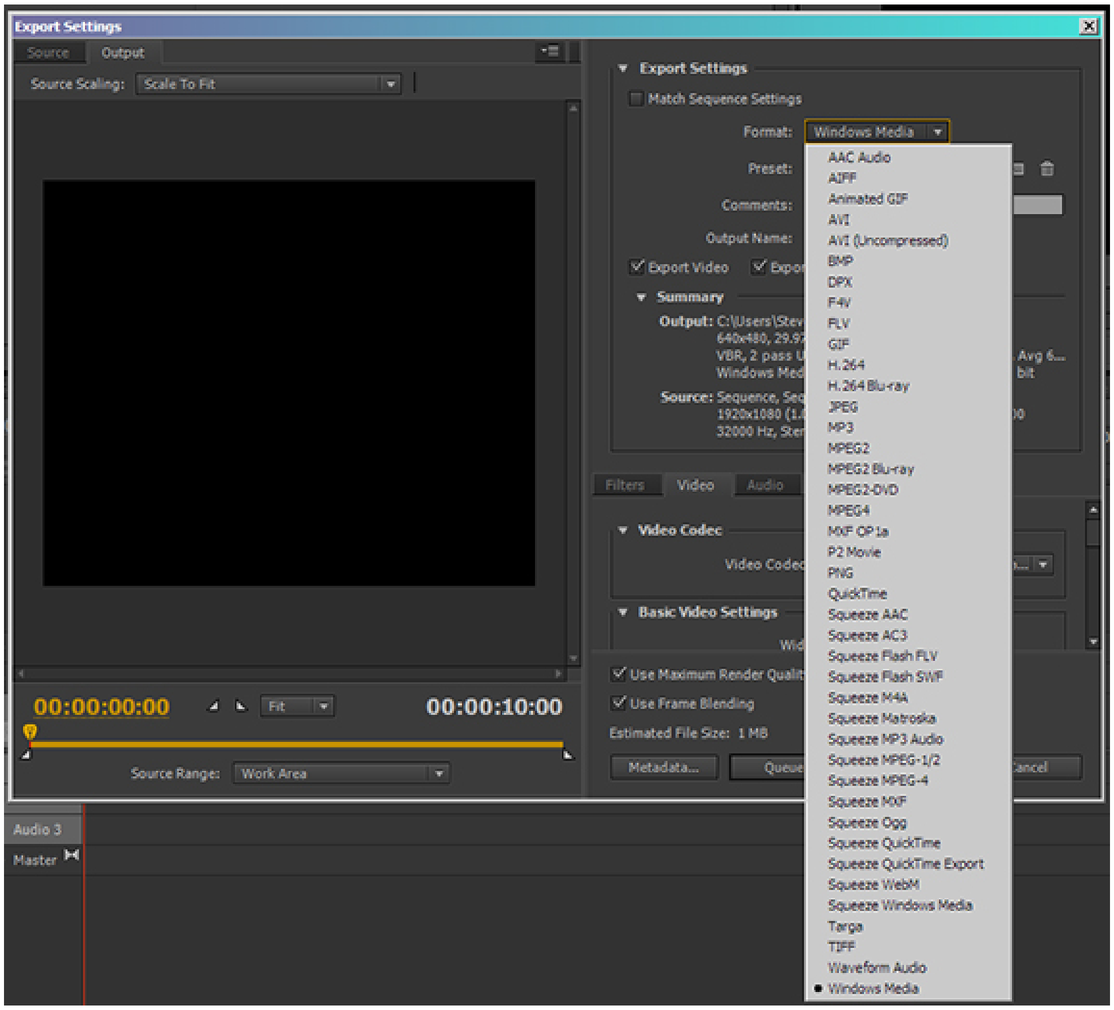
NOTE: I took out references to Flash that were in the first version of this article in the above section because Flash is mostly a dead issue today. The three major browsers (Explorer [now Edge]), Chrome, and Firefox) that dominate the market no longer support Flash, and neither does Safari. The reasons are manifold and complex, but let's just say that the Flash technology didn't keep up with today's mobile devices and computers. Of course, there's the (in)famous spat between Apple and Adobe, but that's a story unto itself. The spat was the beginning of the end for Flash though and it's becoming a tool of the past.

Lossy vs. non-lossy codecs

Lossy is what it seems—it means just that. Some of the data in a video is lost when it's encoded. Can you see the loss? It depends on how good or bad your encoder is. There's actually a lot of superfluous data in a frame of video and if some of it gets lost, so much the better for file size because you can't see it. In analog TV, there was about a 10 percent loss of the picture area due to "scanning" and "over-scanning" of the images. Digital video doesn't have any of the "safe area" problems that the good old analog version had. In analog video, you had to make your screen supers fit within the safe area, somewhat more than 10 percent of the area of the picture. If you ran the super or information to the edge of the screen, chances are that by the time it got to someone's home, it looked like it got cut off…and it did!

The problem with codecs and formats

The problem with all those codecs and formats is there are just too many of them. When you look at the dropdown menu in the Adobe Media Encoder (Figure 3), you'll be presented with no less than 26 media formats.


Export Settings

Source Scaling: Scale To Fit

Match Sequence Settings

Format: Windows Media

✓ Export Video

Summary

Output: C:\Users\Stev…
640x480, 29.97…
VBR, 2 pass U…
Windows Med…

Source: Sequence, Seq…
1920x1080 (1.…
32000 Hz, Ster…

✓ Use Maximum Render Quality
✓ Use Frame Blending

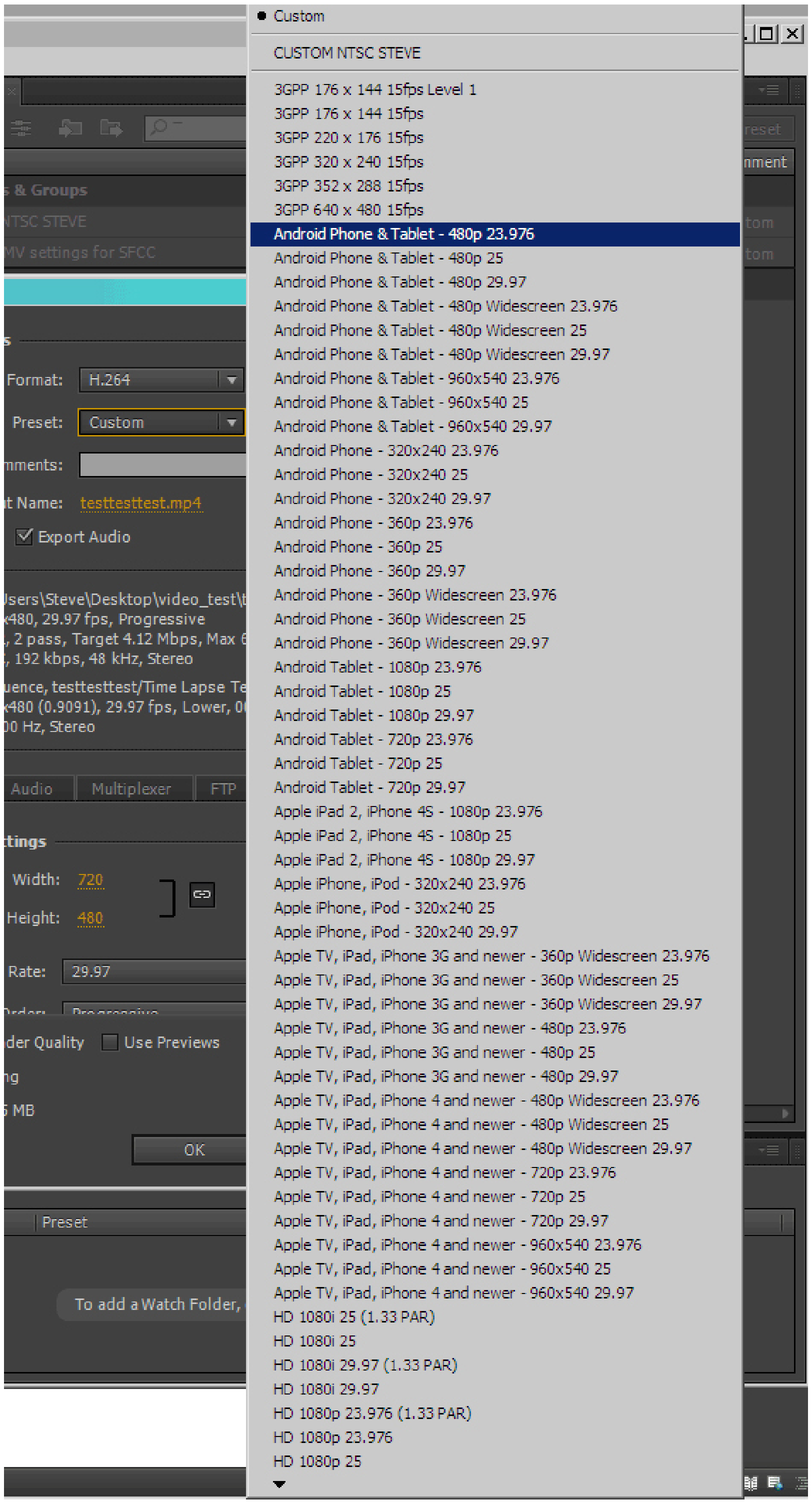
Estimated File Size: 1 MB

00:00:00:00
00:00:10:00

Source Range: Work Area

Figure 3: **Adobe Media Encoder dropdown menu**

And that's 26 formats without all the Sorenson presets included in the dropdown. If you include them, it's over 40 formats to choose from. Everything from uncompressed AVI to Windows Media WMV is available. Depending on the format you select, you'll either have just a few codecs and sizes to choose from (try F4V, which is a Flash video format), or you will get a second dropdown menu with a bewildering number of codecs to select to encode your project (Figure 4).



Format: H.264

Preset: Custom

mments:

t Name: testtesttest.mp4

☑ Export Audio

Jsers\Steve\Desktop\video_test\t
‹480, 29.97 fps, Progressive
, 2 pass, Target 4.12 Mbps, Max 6
, 192 kbps, 48 kHz, Stereo

uence, testtesttest/Time Lapse Te
‹480 (0.9091), 29.97 fps, Lower, 0
00 Hz, Stereo

Width: 720

Height: 480

Rate: 29.97

● Custom
CUSTOM NTSC STEVE
3GPP 176 x 144 15fps Level 1
3GPP 176 x 144 15fps
3GPP 220 x 176 15fps
3GPP 320 x 240 15fps
3GPP 352 x 288 15fps
3GPP 640 x 480 15fps
Android Phone & Tablet - 480p 23.976
Android Phone & Tablet - 480p 25
Android Phone & Tablet - 480p 29.97
Android Phone & Tablet - 480p Widescreen 23.976
Android Phone & Tablet - 480p Widescreen 25
Android Phone & Tablet - 480p Widescreen 29.97
Android Phone & Tablet - 960x540 23.976
Android Phone & Tablet - 960x540 25
Android Phone & Tablet - 960x540 29.97
Android Phone - 320x240 23.976
Android Phone - 320x240 25
Android Phone - 320x240 29.97
Android Phone - 360p 23.976
Android Phone - 360p 25
Android Phone - 360p 29.97
Android Phone - 360p Widescreen 23.976
Android Phone - 360p Widescreen 25
Android Phone - 360p Widescreen 29.97
Android Tablet - 1080p 23.976
Android Tablet - 1080p 25
Android Tablet - 1080p 29.97
Android Tablet - 720p 23.976
Android Tablet - 720p 25
Android Tablet - 720p 29.97
Apple iPad 2, iPhone 4S - 1080p 23.976
Apple iPad 2, iPhone 4S - 1080p 25
Apple iPad 2, iPhone 4S - 1080p 29.97
Apple iPhone, iPod - 320x240 23.976
Apple iPhone, iPod - 320x240 25
Apple iPhone, iPod - 320x240 29.97
Apple TV, iPad, iPhone 3G and newer - 360p Widescreen 23.976
Apple TV, iPad, iPhone 3G and newer - 360p Widescreen 25
Apple TV, iPad, iPhone 3G and newer - 360p Widescreen 29.97
Apple TV, iPad, iPhone 3G and newer - 480p 23.976
Apple TV, iPad, iPhone 3G and newer - 480p 25
Apple TV, iPad, iPhone 3G and newer - 480p 29.97
Apple TV, iPad, iPhone 4 and newer - 480p Widescreen 23.976
Apple TV, iPad, iPhone 4 and newer - 480p Widescreen 25
Apple TV, iPad, iPhone 4 and newer - 480p Widescreen 29.97
Apple TV, iPad, iPhone 4 and newer - 720p 23.976
Apple TV, iPad, iPhone 4 and newer - 720p 25
Apple TV, iPad, iPhone 4 and newer - 720p 29.97
Apple TV, iPad, iPhone 4 and newer - 960x540 23.976
Apple TV, iPad, iPhone 4 and newer - 960x540 25
Apple TV, iPad, iPhone 4 and newer - 960x540 29.97
HD 1080i 25 (1.33 PAR)
HD 1080i 25
HD 1080i 29.97 (1.33 PAR)
HD 1080i 29.97
HD 1080p 23.976 (1.33 PAR)
HD 1080p 23.976
HD 1080p 25

***Figure 4:* Codec formats**

But wait … there's more. Depending on the codec you select, there are other moving targets that will make your video file size larger or smaller, have better or worse temporal quality, or make it able to be seen by any number of devices and browsers. This is where the real problem begins for delivery to multiple device and browser versions. Most of us know that Flash won't play in iOS, but mp4 will when encoded with H.264. But that codec and format combination won't play in older versions of Firefox or Internet Explorer. So you can be forced into a lose-lose situation when you need to play your project across multiple devices and platforms.

Video cards

In the last few years video cards have had a great impact on video editing and rendering. It used to be that your video card didn't make much difference when it came to editing, encoding, and compressing video, but it sure does now. The newest programs from Sorenson and Adobe make use of the GPU (graphics processing unit, aka the chip on your video card) with many video cards.

A decent video card now has about two to four gigabytes of RAM. Three years ago, video cards weren't used for much other than displaying video and rendering video games. The fact is, video cards run significantly faster than an Intel or AMD processor when editing or rendering. And they have more cores. More cores means faster rendering and that means your render times are shorter. Those cores were designed for nothing but video, unlike the general purpose processors that power all the other stuff on our computers. With the launch of Creative Suite 5 (CS5) in 2010, Adobe released the Mercury playback engine that rendered video by using the nVidia video card hardware. What this meant to users was that in the Premiere Pro timeline there was almost real time rendering (well, not exactly real time, but it would require another article to explain that). By using the GPU to do the heavy lifting on timeline rendering, Adobe opened the door to faster video editing and rendering. Over the years, there have been some issues with the Mercury engine and the Dynamic Link engine, which allows compositions in After Effects to be directly connected in Premiere Pro. But all seems to be working fine these days and there are improvements in the 2015 version of the Creative Cloud.

Finally: the rendering process

There are two kinds of video rendering, and the kind of rendering you're planning on or need to do will greatly influence the tool you use. The first (and simplest) rendering is from the timeline in Adobe Captivate, Premiere Pro, Vegas, Final Cut Pro, or whatever timeline-based program you're using. This is direct rendering off the timeline, and you're pretty much stuck using the encoding software that came with your video editing package. This works well for those who are placing their video in PowerPoint, Articulate, or whatever.

When you need to render to multiple platforms, dimensions, and codecs, or you're serving your own video directly, you'll probably render from your timeline to an AVI or uncompressed MOV file first, and then render to the different formats using a tool like Sorenson Squeeze. There are several products in the marketplace, and I don't want to write reviews of them here, but with a separate rendering software package, you can set up a render "queue" with all the file types and dimensions you'll need for your finished product. You can also do that with the Adobe Media Encoder, but a product like Squeeze has more options if you need them.

Methodology

Table 1 shows you the results (size, quality, and time-to-render) that I got from a variety of format and codec combinations. Let me explain what I did before you look at the table.

After a lot of consideration, I simplified the process as much as I could. It was Occam's razor. Simpler is better. There are just too many variables. And no matter what we do, your results will be somewhat different. Simplest is best … at least in this case.

The method: I used a 10-second video clip from a time-lapse project I did a while back. The clip has some, but not complete, movement in it. However, there are a lot of light changes and that (to a rendered video anyway) is movement. We also rendered as close to 30 fps as we could (some codecs only allowed 29.97 fps) and progressive scan. If your project doesn't have much change or movement, 15 fps might be good enough, but apples-to-apples, we needed to pick something.

To do the rendering, I used a variety of different formats and keyframe rates (not frames per second) so that there is some variation that you might be able to key in on. There are so many formats you can use that I limited it to five, with a two-keyframe interval where that was available as an option. We also limited the number of codecs. It gets silly after a while, and some of the formats can only render one codec. The temporal quality is purely a judgment call on my part. If the video looked substantially different when I looked at it, I marked it down or up. When there was blocky pixilation, it really got marked down. The size speaks for itself.

Table 1: **Results from various combinations of format and codec**

Format	**Codec**	**Overall Size**	**Temporal Quality**	**Time to render (min:sec.)**
.avi	Uncompressed	272Mb	Excellent	0:9.5
.avi	NTSC DV	37Mb	Excellent	0:9.1
.avi	Custom	37Mb	Excellent	0:7.6
.f4v	Match Source	2.28Mb	Fair	0:7.6
.f4v	Custom	7.74 Mb	Good	0:19.4
H.264	Custom-Android	5.29 Mb	Good	0:17.1
H.264	Apple TV	4.22 Mb	Fair	0:8.0
QuickTime	H.264 (keyframe 1)	10.41 Mb	Good	0:6.8
QuickTime	H.264 (keyframe 30)	4.22 Mb	Fair	0:7.8
QuickTime	MP4 (keyframe 1)	14.12Mb	Good	0:9.1
QuickTime	MP4 (keyframe 30)	2.40 Mb	Fair	0:9.3
Sorenson Flash	FLV (720)	2.22Mb	Fair	1:01.8
Squeeze MP4	MP4	4.12Mb	Fair	0:35.3
Squeeze QuickTime	Export JPEG	14.12Mb	Good	0:15.8
Squeeze wmv	2k	1.12Mb	Good to very good	0:53.7
Windows Media	DV NTSC	2.7Mb	Very good	0:19.5
Windows Media	Custom	2.59Mb	Very good	0:20.1
Windows Media	Custom 30p	2.65	Very good to excellent	0:11.2

Conclusions

There are 18 versions of the 10-second video. There could have easily been over 100 versions. It's difficult to make concrete conclusions and recommendations because everyone's video project, bandwidth allowances and needs, and temporal qualities are different.

As a general conclusion, we would say if you can use YouTube, by all means do so. Yes, you get the frame around the picture, but YouTube can adapt to the constantly changing environment in video as well as delivering your content to different platforms. If you're only encoding one or two videos at a time, Sorenson Squeeze is not worth the money. But if you have a training department and are encoding tens or dozens of videos at one time, then it becomes a bargain. I didn't get to test the Sorenson Server, but I'm sure it's good. The idea of encoding on the fly when there's a lot of different formats and devices to deliver to makes a product like Squeeze an attractive alternative to rendering and then showing.

The best codec to keep file sizes small is Windows Media (WMV) with H.264 encoding. Microsoft has done an excellent job of combining small file size and very good temporal quality. The format isn't necessarily good for iOS devices, but if you can manage to get your projects on YouTube (where you can make a video private or even have a password), you will be able to deliver everywhere. And that's a plus, in my opinion.

If you need something to serve on almost any device today, you need a video server that can figure out what device is requesting the video, what the size of the screen is, how fast the user's connection is, etc., etc., etc. In some circumstances you might face, there are innumerable codecs and sizes for a given format. In H.264 format there are 24 presets for Android phones and tablets and 21 presets for iDevices! Sometimes you've got to choose one and just let it go.