Conversation held on superuser.com leading to a constructive answer from Warren Young with notions of video compression and quality explained.

How to create an uncompressed AVI from a series of 1000's of PNG images using FFMPEG

Question

How can I create an uncompressed AVI from a series of 1000's of PNG images using FFMPEG?

I used this command to convert an input.avi file to a series of PNG frames:

```
ffmpeg -y -i input.avi  -an -vcodec png  -s 1024x768 pic%d.png`
```

Now I need to know how to make an uncompressed AVI video from all those PNG frames. I tried this:

```
ffmpeg -i pic%d.png -y -f avi -b 1150 -s 1024x768 -r 29.97 -g 12 -qmin 3 -qmax 13 -ab 224 -ar
44100 -ac 2 test.avi
```

But the resulting video loses a lot of quality relative to the original AVI.

Asked by user995074 the 14th of October 2011 at 9:00

Edited by Warren Young the 17th of October 2011 at 14:47

Answer

There are several ways to get an "uncompressed" AVI out of ffmpeg, but that's not as well defined a term as you might assume. A better term is "lossless," though there's some wiggle room in its definition, too.

I'm going to use 720p HD video here to anchor the discussion. (1280x720p, 24fps) Everything I've written below applies just as well to other frame sizes and frame rates.

Fully Uncompressed

If your definition of "uncompressed" is the form the video is in right before it's turned to photons by a digital display, the closest I see in the ffmpeg -codecs list are -vcodec r210 and -vcodec v308. The difference between them comes down to the bit depth.

- **R210** is 4:4:4 YUV at 10 bits per pixel, so it comes to **690 Mbit/s** for 720p in my testing. (That's about a third of a terabyte per hour, friends!)

  R210 may be the same thing as the Blackmagic codec, which is -vcodec blackmagic in recent versions of ffmpeg. Blackmagic Design actually offers several codecs (PDF documentation) which vary in compression level and bit depth. These codecs can be used in either AVI or QuickTime containers, though to read them in normal video apps you'll probably have to have the proprietary Blackmagic codecs installed, and that requires product registration.

- **V308** is the same thing, but at 8 bpp, so it comes to **518 Mbit/s** in my testing. The link sends you to an Apple QuickTime page, but it may be that the Blackmagic codec pack will let you use this in other apps inside an AVI container.

There's also -vcodec ayuv, but that will be 25% larger than V308 for no benefit, since you probably don't need the alpha channel. If you do need the alpha channel, see QuickTime Animation below. It's more likely to be compatible with the other software you'll be using.

So, putting all this together, if your PNGs are named frame0001.png and so forth:

```
ffmpeg -i frame%04d.png -f avi -vcodec v308 -video_size 1280x720 output.avi
```

This frame size argument may or may not be required.

Compressed RGB, But Also Lossless

The closest codec ffmpeg supports to what you actually are asking is **Apple QuickTime Animation**, via `-vcodec qtrle`

I know you said you wanted an AVI, but the fact is that you're probably going to have to install a codec on a Windows machine to read any of the AVI-based file formats mentioned here, whereas with QuickTime there's a chance the video app of your choice already knows how to open a QuickTime Animation file.

`ffmpeg` will stuff `qtrle` into an AVI container for you, but the result may not be very widely compatible. In my testing, QuickTime Player will gripe a bit about such a file, but it does then play it. Oddly, though, VLC won't play it, even though it's based in part on `ffmpeg`. I'd stick to QT containers for this codec.

The QuickTime Animation codec uses a trivial RLE scheme, so for simple animations, it should do about as well as Huffyuv below. The more colors in each frame, the more it will approach the bit rate of the fully uncompressed options above. In my testing using a Pixar-style 3D cartoon movie, however, I was able to get ffmpeg to give me a **250 Mbit/s** file in RGB 4:4:4 mode, via `-pix_fmt rgb24`.

This is one of the reasons I've made a point of talking about the inappropriateness of the term "uncompressed." This format is in fact compressed, yet will give identical pixel values to your PNG input files, for the same reason that PNG's lossless compression doesn't affect pixel values.

The `ffmpeg` QuickTime Animation implementation also supports `-pix_fmt argb`, which gets you 4:4:4:4 RGB, meaning it has an alpha channel. This is a better option, for compatibility reasons, than the raw `-vcodec ayuv` option above.

There are variants of QuickTime Animation with *fewer* than 24 bits per pixel, but they're best used for progressively simpler animation styles. `ffmpeg` appears to support only one of the other formats defined by the spec, `-pix_fmt rgb555be`, meaning 15 bpp big-endian RGB. It would be fine for most screencast captures, for example.

Putting all this together:

```
ffmpeg -i frame%04d.png -f mov -vcodec qtrle -pix_fmt rgb24 output.mov
```

Effectively Lossless: The YUV Trick

Now, the thing about RGB and 4:4:4 YUV is that these encodings are very easy for computers to process, but they ignore a fact about human vision, which is that our eyes are more sensitive to black and white differences than color differences.

Video storage and delivery systems therefore almost always use fewer bits per pixel for color information than for luminance information. The most common schemes are 4:2:0 and 4:2:2.

The data rate of 4:2:0 YUV is just 33% higher than for black and white (Y only) uncompressed video, or half the data rate of 4:4:4 RGB or YUV. The old DV camera standard used 4:1:1 instead, which has the same uncompressed data rate as 4:2:0, but the color information is arranged differently.

4:2:2 is a kind of halfway point between 4:2:0 and 4:4:4. It is twice the data rate of Y-only video, or ⅔ the data rate of 4:4:4.

The point of all this is that if you're starting with a 4:2:0 H.264 file, re-encoding it to 4:4:4 RGB buys you absolutely nothing over 4:2:0 uncompressed YUV, unless you need the RGB encoding to send it to a device or program that doesn't understand YUV, such as an LCD display or Photoshop. Even then, storing the file on disk as 4:2:0 YUV still makes more sense, because the conversion to RGB can happen on the fly and still be effectively lossless.

You really only need 4:4:4 when you're pixel peeping or you're doing pixel-level color changes on the video, and need to preserve exact pixel values. Visual effects (VFX) work is easier to do with a 4:4:4 pixel format, for example, so high-end VFX houses are often willing to tolerate the higher data rates it

requires.

Once you open yourself up to YUV codecs with color decimation, your options open up, too. ffmpeg has many *effectively lossless* codecs. QuickTime Animation above is one of them, but switching to the YUV color space gives you an even better option: **Huffyuv**.

You get this either via -vcodec huffyuv or -vcodec ffvhuff

Huffyuv supports several pixel formats, the most interesting of which for our purposes here are RGB 4:4:4 (a.k.a. RGB24), YUV 4:2:2, and YUV 4:2:0 (a.k.a. YV12). An older version of ffmpeg I tested only worked with YUV 4:2:2, so if you see that your version keeps telling you the pixel format is incompatible and that it's going to use yuv422p instead, upgrade.

- -pix_fmt rgb24 — RGB 4:4:4 is essentially the same thing as QuickTime Animation, except with potentially better compression.

  It is also essentially the same thing as the YUV 4:4:4 mode used by the V308 option above. The color space difference makes no practical difference, except that it may be slower to decode and display on YUV-native systems.

  Due to Huffyuv's lossless compression, I was able to get a test video to compress to about **260 Mbit/s**, with identical visual quality to what you'd get from V308. Although this is scarcely different from what QuickTime Animation gives, I'd expect it to give much better results with some files, particularly live action video.

- -pix_fmt yuv422p — This mode is far more practical for video than RGB24, which is doubtless why the ffmpeg developers chose to implement it first. As you'd expect from the theoretical ⅔ reduction discussed above, my test file encoded to 176 Mbit/s. That's pretty much exactly ⅔, if you take into account the fact that the audio track stayed the same bit rate between these two tests.

- -pix_fmt yuv420p — Newer versions of ffmpeg should support this mode, too, but there seems to be some bug that prevents it from taking effect if you use it with -vcodec huffyuv. However, I found that if you use the ffvhuff codec instead, it defaults to this pixel format, giving a 139 Mbit/s file in my testing. I don't know if it's compatible with the Windows DirectShow codec, since they describe this codec as a "Huffyuv variant".

Putting all this together:

```
ffmpeg -i frame%04d.png -f avi -vcodec huffyuv -pix_fmt rgb24 output.avi
```

ffmpeg also supports decode-only mode for **Lagarith** and encode-only mode for **Lossless JPEG**. These two codecs are actually somewhat similar, and should give files a bit smaller than Huffyuv with the same quality. If the ffmpeg developers ever add Lagarith encoding, it would be a strong alternative to Huffyuv.

I can't recommend Lossless JPEG, though, because the support for this format isn't very good across the industry. There is a commercial Windows codec, but I don't know if it can decode the files ffmpeg creates.

Perceptually Lossless: Or, You Can Probably Get Away With Some Loss

Then there are the codecs that are *perceptually* lossless. Unless you're pixel peeping, you almost certainly cannot tell that these give different visual results than the previous two groups. By giving up on the idea of absolutely zero change between the video capture sensor and the display device, you buy considerable savings:

- **Apple ProRes**: -vcodec prores or -vcodec prores_lgpl — This comes in several variants. ProRes 4444 gives **132 Mbit/s** for our 720p example, and is VFX quality. Stepping down to ProRes 422 HQ, you get the data rate down to **88 Mbit/s**, and almost certainly cannot tell any difference without pixel peeping. There are even more parsimonious variants of ProRes, but they're meant for either SD video or as proxies for full-res files.

The main problem with ProRes is that it doesn't yet have wide support outside the Apple and pro video worlds.

- **Low-loss MJPEG**: `-vcodec mjpeg -qscale 1` or `-vcodec mjpegb -qscale 1` — This is far more common than lossless JPEG, and so it's much easier to get support for it. Expect pretty wide variability in data rates from this codec. A test I just made here gave me **26 Mbit/s** for 720p video. That's high enough compression to make me nervous about lossiness, but it looked pretty good to me. Based on data rate alone, I'd say it's probably on par quality-wise to **12 Mbit/s** MPEG-2 or **6 Mbit/s** H.264.

Putting all this together:

```
ffmpeg -i frame%04d.png -f avi -vcodec mjpeg -qscale 1 output.avi
```

Bottom line on these methods is that unless you're doing something very demanding, "good enough" really is good enough.

Answered by Warren Young the 16th of October 2011 at 21:29

Edited the 23rd of May 2012 at 09:04

source: http://superuser.com/questions/347433/how-to-create-an-uncompressed-avi-from-a-series-of-1000s-of-png-images-using-ff