What is the difference between YUV and YCrCb?

YUV and YCbCr are both difference encodings of RGB signals used in video.
YUV is found in legacy analog video.
YCbCr is found in digital video formats.

Y is a weighted sum of RGB,
UV and CbCr are both weighted differences between RB and Y.
The weights are designed such that UV and CbCr end up being zero or constant for neutral colors (black, gray and white colors where R=G=B).
This means that the luminance information is mostly carried in the Y channel, and the color information is carried in the other two.
Video transmission and compression can take advantage of this. The color information can be subsampled (transmitted at a lower resolution and using less bandwidth) with minimal loss in image quality.

YUV is a color space system in which colors are defined by using one luminance value and two chrominance values. This is also called as a luma-chroma system. Y is the Luminance component while UV are the Chrominance components.

The YCbCr system is the color space used in digital electronic systems such as digital TVs. In this case, the Cb and Cr are the chroma values. They are the digital counterparts of YPbPr which was predominantly used for analog TVs.

However, in the general sense YUV usually denotes YCbCr. They are used interchangeably though there is a subtle difference in the way they are encoded

Most consumer video codecs imply a 4:2:0 format. Actually Main/Main10 HEVC only supports 4:2:0.

If I understand correctly, you want to convert an H.265 video file to a raw YUV video file, or raw YUV video frame sequence.

ffmpeg can do either for you. If you want to use the same chroma subsampling and bit depth as the H.265 source:

```
ffmpeg -i input.mp4 output.yuv
```

What is the difference between transcoding and encoding?

Most of the time you are transcoding from one format to another. For example, cameras rarely save your videos in uncompressed form, there are very few cameras that allow you to do that and it is typically done with an external device with large storage capacity that attaches via SDI or HDMI. Examples of camera-compressed formats are DV, ProRes, MPEG-2 and H.264.
What is perceived as encoding is taking an SDI or HDMI connector from a camera (playing its saved content from tape or an SD card) and hooking it up to an Encoder that only has the ability to de-serialize SDI or HDMI (which are serial and uncompressed formats for video) and then convert it to a target standard such as MPEG-2 or H.264.
When you are truly encoding is when you start from a scan of a film (as long as each frame was saved in uncompressed form) and then create a compressed rendition of your film.

So, in summary, Transcoding is when you start with a compressed format such as DV/ProRes/MPEG-2/H.264 and then convert your video stream to MPEG-2 or H.264 (which are the most common end formats) and you usually convert to a lower bitrate than what you start with. True Encoding is when you start with the pristine output of a CCD/CMOS via SDI/HDMI in live form or from a Film Scan.

Two different things.
Encode: You start with raw, uncompressed video and you create a coded bitstream. Example: you capture raw video from a camera and you create an mp4 file.

Transcode: You start with a coded video in one format and you create a video in another coded format. Example: you start with .flv and you create mp4.
Transcode typically means decode + encode. (it can also be the same format, but different resolution, bit rate, etc.; example from HD video to SD video).

Simple explanation with no acronyms:

Encoding is when a raw and uncompressed video source is compressed using a codec.

Transcoding is when a previously encoded video source is converted into another format.

Generally speaking transcoding is usually used for the generation of multiple video qualities of a single source, to enable smooth playback whilst viewing a video online.

n short:

Encoding is to take an uncompressed input (SDI, YUV, etc.) and created an compressed output files (e.g. h.264 mp4).

Transcoding is when you take this (high quality) compressed file and compress it in multiple versions, qualities, etc. for e.g. web delivery/streaming. Here you can e.g. see a guide how to select bitrates/resolutions for a multi-bitrate streaming format such as MPEG-DASH and HLS

Movies comes inbuilt with encoding and transcoding options via which you can easily upload video files and stream the same via the website or mobile app. Muvi supports uploading in the following video file formats – MP4, MOV, MKV, FLV, VOB, M4V, AVI, 3GP, MPG. Once the video file is uploaded, Muvi encodes the same to different resolution versions for lower Internet speeds and screen sizes (mobile) automatically. The right video will be delivered automatically depending on speed and screen size. Auto Bitrate Conversion to 4K, 2K, 1080, 720, 640, 480, 360, 240, 144

What is the difference between encoding, decoding, and transcoding a digital data stream?

Encoding means to encode/compress raw (uncompressed) data stream. (Imagine **folding** a paper having some printed picture.)
Decoding means to decode/uncompress encoded (compressed) data stream. (Imagine **unfolding** a paper having some printed picture.)
Transcoding means to convert an encoded data stream in particular format to a new encoding/compression format. (Imagine folding a folded paper having some printed picture in a new way.)

What is encoding and decoding in communication?

Encoding means the creation of a messages (which you want to **communicate** with other person). On the other hand **decoding** means listener or audience of **encoded** message. So **decoding** means interpreting the meaning of the message. ... You will interpret and understand the message, what just been said.

All **communication** begins with the sender. The first step the sender is faced with involves the **encoding process**. In order to convey meaning, the sender must begin **encoding**, which means translating information into a message in the form of symbols that represent ideas or concepts.

The **decoding** of a message is how an audience member is able to understand, and interpret the message.

Imagine yourself talking (interpersonal with no mechanical medium) with your friend Shawn and you said 'I like you'.

You have already interpreted the message i.e you like him and now you have to encode it. Encoding is a process where you turn the message into a particular format(words in this case) which can be received by the receiver through a particular medium(air in this case)

You are converting the message into words and then transferring it to Shawn.

Then Shawn has to decode it i.e he has to receive the words and recognize. If he doesn't recognize then there is no question of decoding and the following process fails. If he does recognize, then he interprets the message.

In case of non-verbal communication, you raise your hand palm facing shawn, which essentially means you want him to stop talking or walking. You have interpreted that you want him to stop and thus you have encoded the message into raising your hand. Now Shawn receives it, decodes the message that you've raised the hand and interprets that you want him to stop.

This is how Encoding and Decoding works.

What is video encoding?

Video encoding is the process of compressing and potentially changing the format of video content, sometimes even changing an analog source to a digital one. In regards to compression, the goal is so that it consumes less space. This is because it's a lossy process that throws away information related to the video. Upon decompression for playback, an approximation of the original is created. The more compression applied, the more data is thrown out and the worse the approximation looks versus the original.

Why is encoding important?

Now there are two reasons why video encoding is important. The first, especially as it relates to streaming, is it makes it easier to transmit video over the Internet. This is because compression reduces the bandwidth required, while at the same time giving a quality experience. Without compression, raw video content would exclude many from being able to stream content over the Internet due to normal connection speeds not being adequate. The important aspect is bit rate, or the amount of data per second in the video. For streaming, this will dictate if they can easily watch the content or if they will be stuck buffering the video.

The second reason for video encoding is compatibility. In fact, sometimes content is already compressed to an adequate size but still needs to be encoded for compatibility, although this is often and more accurately described as transcoding. Being compatible can relate to certain services or programs, which require certain encoding specifications. It can also include increasing compatibility for playback with audiences.

The process of video encoding is dictated by video codecs, or video compression standards.

What are codecs?

Video codecs are video compression standards done through software or hardware applications. Each codec is comprised of an encoder, to compress the video, and a decoder, to recreate an approximate of the video for playback. The name codec actually comes from a merging of these two concepts into a single word: enCOder and DECoder.

Example video codecs include H.264, VP8, RV40 and many other standards or later versions of these codecs like VP9. Although these standards are tied to the video stream, videos are often bundled with an audio stream which can have its own compression standard. Examples of audio compression standards, often referred to as audio codecs, include LAME/MP3, Fraunhofer FDK AAC, FLAC and more.

These codecs should not be confused with the containers that are used to encapsulate everything. MKV (Matroska Video), MOV (short for MOVie), AVI (Audio Video Interleave) and other file types are examples of these container formats. These containers do not define how to encode and decode the video data. Instead, they store bytes from a codec in a way that compatible applications can playback the content. In addition, these containers don’t just store video and audio information, but also metadata. This can be confusing, though, as some audio codecs have the same names as file containers, such as FLAC.

What’s the best video codec?

That’s a loaded question which can’t be directly answered without more information. The reason is that different video codecs are best in certain areas.

For high quality video streaming over the Internet, H.264 has become a common codec, estimated to make up the majority of multimedia traffic. The codec has a reputation for excellent quality, encoding speed and compression efficiency, although not as efficient as the later HEVC (High Efficiency Video Coding, also known as H.265) compression standard. H.264 can also support 4K video streaming, which was pretty forward thinking for a codec created in 2003.

As noted, though, a more advanced video compression standard is already available in HEVC. This codec is more efficient with compression, which would allow more people to watch high quality video on slower connections. It’s not alone either. In 2009, Google purchased On2, giving them control of the VP8 codec. While this codec wasn't able to take the world by storm, it was improved upon and a new codec, dubbed VP9, was released. Netflix tested these later formats versus H.264, using 5,000 12 second clips from their catalog. From this, they found that both of the codecs were able to reduce bitrate sizes by 50% and still achieve similar quality to H.264. Of the two, HEVC outperformed VP9 for many resolutions and quality metrics. The exception was at 1080p resolution, which was either close and in some scenarios had VP9 as more efficient.

By these tests, wouldn't this make HEVC the best codec? While technically it's superior to H.264, it overlooks a key advantage of the older codec: compatibility. H.264 is widely supported across devices, for example it wasn't until iOS 11 in late 2017 that iPhones could support HEVC. As a result, despite not being as advanced, H.264 is still favored in a lot of cases in order to reach a broader audience for playback.

Note, the H.264 codec is also sometimes called X.264. However, this is not the same codec but actually a free equivalent of the codec versus the licensed H.264 implementation.

What's the best audio codec?

Like video, different audio codecs excel at different things. AAC (Advanced Audio Coding) and MP3 (MPEG-1 Audio Layer 3) are two lossy formats that are widely known among audio and video enthusiasts. Given that they are lossy, these formats, in essence, delete information related to the audio in order to compress the space required. The job of this compression is to strike the right balance, where a sufficient amount of space is saved without notably compromising the audio quality.

Now both of these audio coding methods have been around for awhile. MP3 originally came on the scene in 1993, making waves for reducing the size of an audio file down to 10% versus uncompressed standards at the time, while AAC was first released in 1997. Being a later format, it's probably not surprising that AAC is more efficient at compressing audio quality. While the exact degree to that statement has been debated, even the creators of the MP3 format, The Fraunhofer Institute for Integrated Circuits, have declared AAC the "de facto standard for music download and videos on mobile phones"… although the statement conveniently happened after some of their patents for MP3s expired (and also led to a bizarre number of stories claiming MP3 was now dead, an unlikely outcome from this). So while MP3 has much more milage with device compatibility to this day, AAC benefits from superior compression and is a preferable method for streaming video content of the two. Not only that but a lot of delivery over mobile devices, when related to video, depends on the audio being AAC. IBM's video streaming and enterprise video streaming offerings are an example of that, although can transcode the audio to meet these specifications if needed.

Now AAC and MP3 are far from the only formats for digital audio. There are many other examples, both lossy like WMA (Windows Media Audio) and lossless like APAC (Apple Lossless Audio Codec). One of these formats is FLAC (Free Lossless Audio Codec), which is lossless. This means the original audio data can be perfectly reconstructed from the compressed data. While the size of the audio track is smaller than WAV (Waveform Audio File Format), an uncompressed format, it still requires notably more data for an audio stream compared to lossy formats like AAC and MP3.

As a result, while loseless is seen on physical media like Blu-rays, it’s less common for streaming where size is important.

So what are the recommended codecs?

Favoring compatibility, H.264 and AAC are widely used, IBM’s video streaming and enterprise video streaming offerings support both the H.264 video codec and the AAC audio codec for streaming. While neither is cutting edge, both can produce high quality content with good compression applied. In addition, video content compressed with these codecs can reach large audiences, especially over mobile devices.

Compression techniques

Now that we have examined codecs a bit, let’s look into a few compression techniques. These techniques are utilized by codecs to intelligently reduce the size of video content. The goal is to do so without hugely impacting video quality. That said, certain techniques are more noticeable to the end viewer than others.

Image resizing

A common technique for compression is resizing, or reducing the resolution. This is because the higher the resolution of a video, the more information that is included in each frame. For example, a 1280×720 video has the potential for 921,600 pixels in each frame, assuming it’s an i-frame (more on this in a bit). In contrast, a 640×360 video has the potential for 230,400 pixels per frame.

So one method to reduce the amount of data is to “shrink” the image size and then resample. This will create fewer pixels, reducing the level of detail in the image at the benefit of decreasing the amount of information needed. This concept has become a cornerstone to adaptive bitrate streaming. This is the process of having multiple quality levels for a video, and it’s common to note these levels based on the different resolutions that are created.

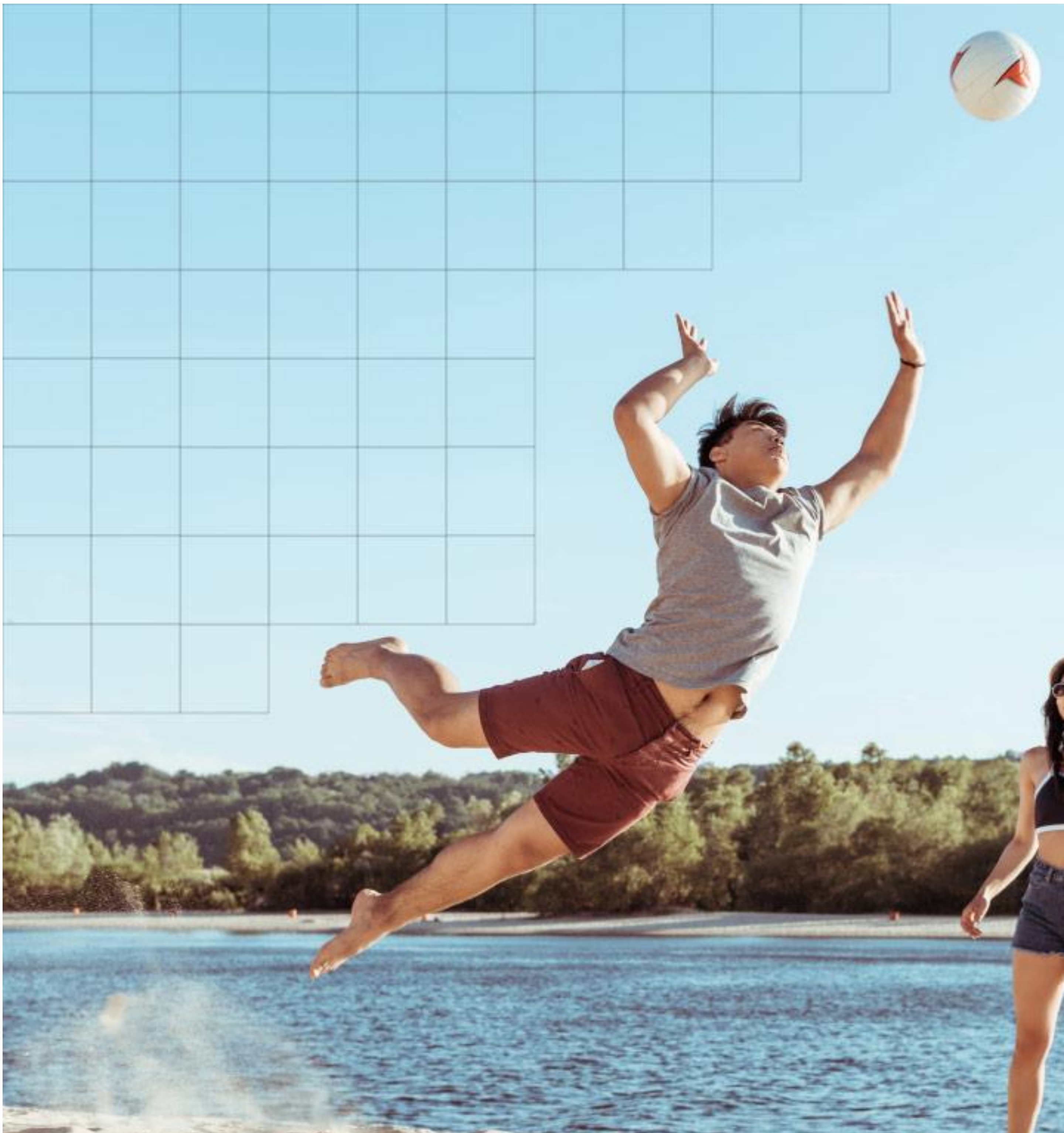
An artifact of resizing can be the appearance of “pixilation”. This is sometimes called macroblocking, although usually this is more pronounced than mere pixilation. In general this is a phenomena where parts of an image look blocky. This can be a combo of a low resolution image and interframe, where details in the video are actually changing but areas of the video as part of the interframe process are being kept. While this is reducing the amount of data the video requires, it’s coming at the cost of video quality.

Note that this resizing process is sometimes referred to as scaling as well. However, scaling sometimes takes on an uncompressed meaning. For example, sometimes it’s used to describe taking the same size image and presenting it in a smaller fashion. In these scenarios, the number of pixels aren’t changing and so there is no compression being applied.

Interframe and video frames

One video compression technique that might not be widely realized is interframe. This is a process that reduces “redundant” information from frame to frame. For example, a video with an FPS (frames per second) of 30 means that one second of video equals 30 frames, or still images. When played together, they simulate motion. However, chances are elements from frame to frame within those 30 frames (referred to as GOP, Group of Pictures) will remain virtually the same. Realizing this, interframe was introduced to remove redundant data. Essentially reducing data that would be used to convey that an element has not changed across frames.

To better understand this concept, let’s visualize it.



Here is a still image of volleyball being thrown and spiked. There is quite a bit of movement in this example, with the volleyball being thrown up, the ball being spiked, sand flying and elements moving from the wind like trees and water. Regardless, there are parts that can be reused, specifically areas of the sky. These are seen in the blocked off area, as these elements don't change. So rather than spend valuable data to convey that parts of the sky hasn't changed, they are simply reused to conserve space. As a result, only the below element of the video is actually changing between frames in this series.

This technique, built into video codecs like H.264, explains why videos with higher motion take up more data or look worse when heavy compression is applied. To execute the technique, the process utilizes three different types of frames to achieve this: i-frames, p-frames and b-frames.

I-frame
Also known as a keyframe, this is a full frame image of the video. How often the i-frame appears depends on how it was encoded. Encoders, like the Telestream Wirecast and vMix, will allow you to select the keyframe interval. This will note how often an i-frame will be created. The more often an i-frame is created, the more space this requires. However, there are benefits to doing periodic i-frames every 2 seconds, the largest of these is due to adaptive streaming, which can only change quality settings on an i-frame.

P-frame
Short for predictive frame, this is a delta frame that only contains some of the image. It will look backwards toward an i-frame or another p-frame to see if part of the image is the same. If so, that portion will be excluded to save space.

B-frame
Short for bi-directional predictive frame, this is a delta frame that also only contains some of the image. The difference between this and a p-frame, though, is that it can look backwards or forwards for other delta frames or i-frames when choosing what details to leave out as they exist in another frame. As a result, b-frames offer improved compression without detracting from the viewing experience. However, they do require a higher encoding profile.

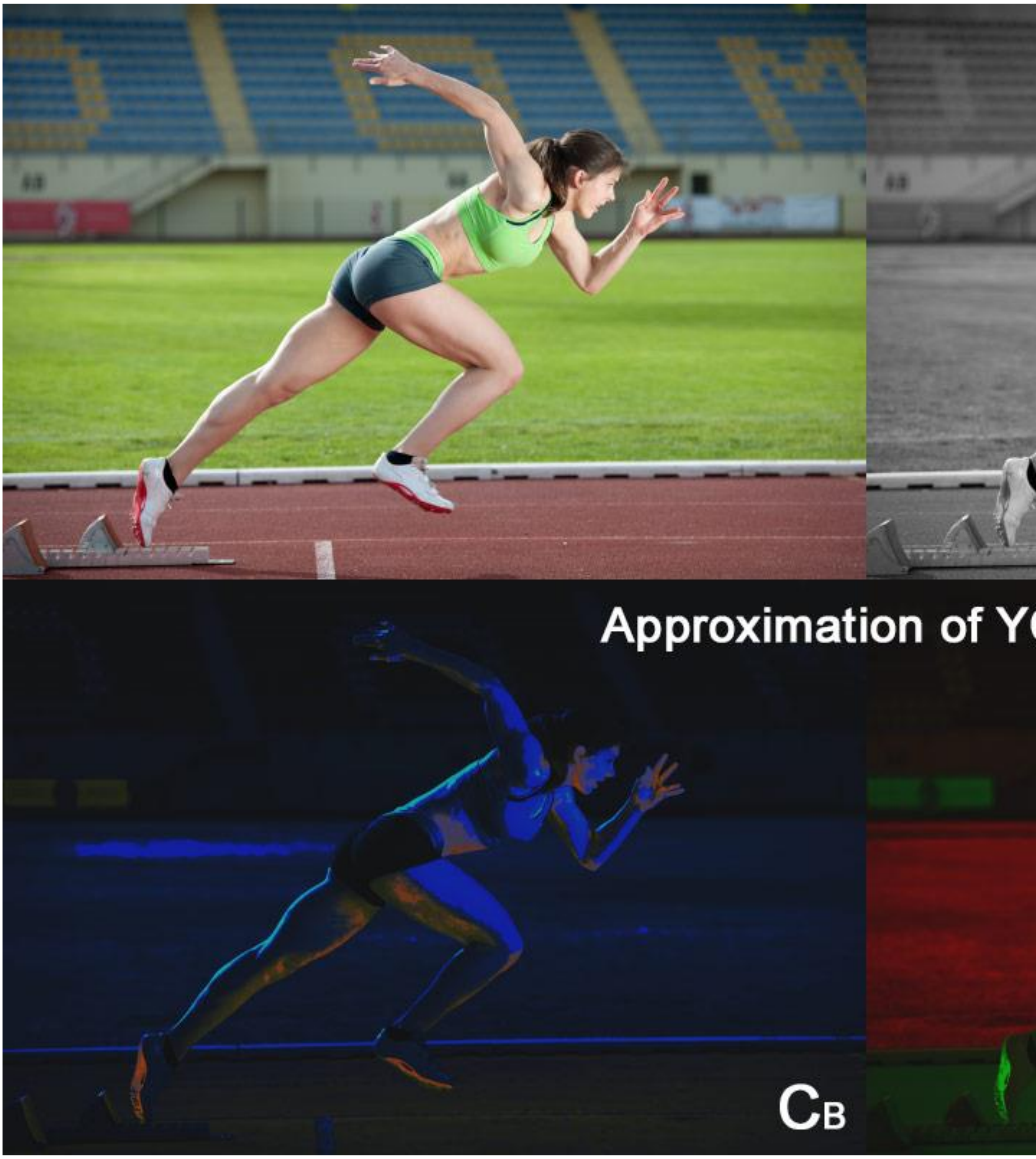
For more details on this entire process, check out our Keyframes, InterFrame & Video Compression article.

Chroma subsampling

Ever seen a gradient (where the colors slowly transition from one to the other) that had a harsh, unnatural look to it? This might be the result of reducing the amount of colors in the image.

As might be expected, the more color information that is required to represent an image, the more space it'll take. As a result, one way to compress data for video is to discard some of this color information. This is a process called chroma subsampling. The principal idea behind it is that the human eye has a much easier time detecting differences in luminance than chroma information. As a result, it tries to preserve the luminance while just sacrificing color quality.

In a nutshell, this is achieved through converting a RGB (Red, Green, Blue) spectrum into YC_BC_R. In doing this, luminance is separated (noted as "Y") and can be delivered uncompressed. Meanwhile, chroma information is contained in C_BC_R, with the potential to reduce them to save space. This is a hard process to describe, but essentially think of the "Y" as a grayscale version of the image. This is overlapped with a C_B "version", which will contain shades of yellow and blue, and a C_R "version", that will contain shades of green and red.



Add all of these elements together and you have a complete, colored image.

When it comes to compressing this, it’s generally noted in a fashion similar to 4:2:2 versus an uncompressed version at 4:4:4. In the latter example, chroma subsampling hasn’t been applied and each has the same sample rate, so no compression is applied. Now each of the numbers lines up with an area of YC_BC_R. So 4:2:2 means that the grey scale version (Y) is not being compressed as it’s still 4, while C_B and C_R are each being reduced to 2 so they have half the sample rate as the luminance. As a result, the amount of bandwidth required is being dropped by 33% ([4+2+2] / [4+4+4] = 66% data versus the original). Other variants exist, for example 4:1:1, which reduces the bandwidth even more, and 4:2:0, which actually alternates betweenC_B and C_R on horizontal lines to compress.

Altering frame rates

Another method to compress video is to reduce the amount of frames per second. This reduces the amount of data that a video will require, as less information is needed to convey each second. This can be a bit more destructive than other methods, though, as lowering the frame rate too much will result in video that lacks fluid movement.

Summary

Now one should know the basics of encoding and why it's done. They should also walk away knowing how content is compressed while not hugely impacting perceived quality. Action items can include using recommended codecs and setting a few elements that are better suited for your distribution method. An example of the latter would be a keyframe interval of 2 seconds to be mindful of adaptive streaming.

Ready to take your broadcasting to the next step? If so, be sure to also check out The Definitive Guide to Enterprise Video eBook to understand what enterprise video can do for you, how fast it is growing and what requirements make it agile and cost efficient.

Note that we did not cover interlacing in this article. The reason being, even though this is a compression technique, is that it's not widely used on content streamed over the Internet, and is something found more in traditional televised content. That said, for those curious on it and also deinterlacing, i.e. trying to reverse the process, please reference this Interlaced Video & Deinterlacing for Streaming article.

What is video encoding? Explain with an example?

First, let's look at our objective here. Video Encoding or rather compression is one of the most complex components in any multimedia system. Video or rather visual content is very huge. Human eye perceives colors based on the light falling on the rods and cones within our eye. This simple 2D picture (a.k.a. **frame**) is what is typically referred to as **texture content**. Video is like a flip book i.e. a series of pictures being recognized at a particular speed or more commonly known as **rate**. So, the eye actually is recognizing these 2D textures at a particular rate (Frame Rate) and transmitting this information to the brain to interpret the same as moving data.

Now, let's look why is compression required? Let's take our common HD TV or a HD movie screen where we watch movies. We feel that the pictures are in continuous motion. For simplicity, I will consider that the rate is 30 frames per sec (Movie screens employ 24 fps).

Let's calculate the amount of data.

Total Data per second = HD frame x 30 fps = 1920 x 1080 x 3 x 30 (where HD = 1920 x 1080 resolution, each pixel carrying color information being represented by 3 bytes - 24bpp)
= **~178 MBytes / second**

If we consider a 3 hour movie, the total storage required is **~1878 GBytes**. That's a lot of space..

If we have to transmit the data over a network or store it in a device, we have literally a very huge problem at hand !!!

Video Compression/Encoding addresses this. It employs a certain set of algorithm to represent the same content in a much reduced space with almost the same quality as the raw data.

Let's now look at how Video Encoding works. Most of the video encoders use a similar set of processing steps. I would recommend to refer to this picture for following explanation. The overall guiding principle of visual compression is reduce the amount of storage by harnessing the redundancies.

First, let's look at a simple picture. The picture is broken down into blocks of 16x16 (a.k.a. Macroblocks / MB) to ease processing. Each MB is connected to it's local neighbors as they are all like little mosaic pieces of a bigger picture.

One of the first steps in Visual compression is to harness the **correlation between the spatial neighbors**. This is also the same guiding principle of **Intra Prediction**.

Similarly, 2 successive pictures also have a lot in common, **correlation between temporal neighbors** is harnessed through **Motion Estimation**.

Out of either of these 2 processes, we get set of **predictors / motion vectors** and differences w.r.t the reference i.e. **delta error signal,** which is nothing but texture content in differential form.

So, we have achieved one set of reduction in space. Next, we need to code these different components in such a way that it can be accurately reconstructed using a reverse process.

The data error signal needs to be compressed next. To achieve this, another correlation function is employed to identify unique identifiers to represent the data. This is achieved by a **transformation** function, which is mostly a form of **DCT**.

Out of transform, we have a set of unique identifiers and small differences around them. These are next **quantized** to fit into a representative set of levels. What this process also does is to convert the little fringe differences to zero.

We can also note that there is a small loss of information, but nothing which would impact the overall look and feel of the video.

The quantized coefficients would be a set of unique identifiers and a string of zeroes. These are then coded using special coding schemes to optimize the number of bits to be transmitted. A simple rule of thumb for encoding is "**Use minimal number of bits for most commonly occurring elements**" (Note: Same principle can be found in telegraph where "e" is represented by a dot). This process is called **Variable Length Encoding**, the output of which is the actual bitstream that is stored or transmitted.

This bitstream is subjected to a reverse process called “**Decoding**” to recreate the actual content. This sums up the overall video compression steps.

Note: There is one more aspect of compression where RGB24 data is converted to YUV420 which in itself reduces the data by half. However, since this topic is covered already in different threads, I gave the same a skip.

What is the difference in working on H.264 and H.265 when encoding. How is H.265 so efficient and compresses files very small sizes?

H.264 (aka MPEG-4 AVC) is currently a mainstream video compression format. It is widely used in Blu-ray discs, internet sources like videos in YouTube and iTunes Store, web software, and also HDTV broadcasts over terrestrial, cable and satellite, while, H.265 (also known as HEVC, short for High Efficiency Video Coding) is a video compression standard whose predecessor is H.264/MPEG-4 AVC. H.265 HEVC ensures to deliver video quality identical to H.264 AVC at only half the bit rate.
H.264 has been a huge success. It’s a flexible codec standard that’s used by streaming services, satellite providers, and for Blu-ray discs. It’s scaled remarkably well since it was first proposed and is capable of handling 3D, 48-60 fps encodes, and even 4K. The Blu-ray disc standard doesn’t currently include provisions for some of these technologies, but the H.264 codec itself is capable of handling them.
The problem with H.264, however, is that while it can handle these types of encodes, it can’t do so while simultaneously keeping file sizes low. A new standard is necessary to push file/stream sizes back down while driving next-generation adoption, and that’s where H.265 comes in. It’s designed to utilize substantially less bandwidth thanks to advanced encoding techniques and a more sophisticated encode/decode model.
H.265 was built to match the capabilities of future screens and includes support for 10 bit color and high frame rates. This is early days — support and capability of the current alpha are limited to 8-bit color and YUV output, but we still wanted to take the alpha technology out for a spin.

What is H.264?

H264 (aka MPEG-4 AVC) is currently a mainstream video compression format. It is widely used in Blu-ray discs, internet sources like videos in YouTube and iTunes Store, web software, and also HDTV broadcasts over terrestrial, cable and satellite.

What is H.265?

H.265 (also known as HEVC, short for High Efficiency Video Coding) is a video compression standard whose predecessor is H.264/MPEG-4 AVC. H.265 HEVC ensures to deliver video quality identical to H.264 AVC at only half the bit rate.

H.265 vs H.264: Differences between H.265 and H.264

In general, H.265 has several big advantages over H.264, including better compression, delicate image and bandwidth saving. For more detailed differences, please read H.265 vs H.264 comparison table.

Cameras have dedicated video processing chips that are able to apply effects using hardware, not software, and the output of that processing feeds in to a fixed video encoding block. If you tried to get the system to do anything else it wouldn't be able to because the components are often plumbed directly together.

Computers are general purpose processing systems, they are bulky and inefficient at doing specific task but if you want to do word processing on a camera you will struggle. An increasing number of computer CPUs have video co-processing or video processing instructions and you can buy a GPU to accelerate general purpose video encoding. It used to be, before computing power was like it is, that you had to do off-line processing but now you can process faster than real time if you have a high power computer

For starters, a camera is processing RAW data. RAW isn’t actually a video file, it’s just a pixel by pixel readout of the camera sensor. The RAW readout from a sensor takes up a massive amount of storage space if you were to store it (which some high end cameras like Arri and Red do), but the data is actually very easy to process as long as you can handle the throughput. Think of it like Simple English Wikipedia. The articles are longer because they don’t use complex words, but they’re easier to understand.

The other reason is that the processors in cameras are custom built to process the video data. Processors in computers are general purpose processors that have been programmed to handle video encoding. You can get external h.264 encoders like the Blackmagic Design H.264 PRO Recorder, which will process h.264 encoding in real-time or faster because it is custom built to do that, and only that.

Cameras are built with specialized hardware for encoding. This is faster than doing so in software, but has certain limitations:

1. The number of formats and resolutions supported is more limited.
2. A certain pre-set list of special effects is included, but there is no real way to add more, and the tweaking options available are generally very limited.
3. Upgradability is low: there’s generally no way to add new encodings, nor to support newer versions of encodings.

It’s possible to buy specialized hardware that can be added to a computer for encoding, but it tends to be quite expensive, to have the same types of limitations as above, and to require specialized software to work with the hardware.

The camera has a dedicated hardware encoder and firmware that is fast enough to encode the video stream in real-time and store it to disk. It’s vastly simplified by not having to deal with a large number software-defined parameters, layers, etc. nor decoding (a video editing package is both decoding the source and encoding the output at the same time)

How do I change the picture quality or bitrate, while a video is encoding?

Many different ways to inprepret the question. For the framerate it can change at any time if wanted 60>3fps if needed and anywhere else.

You can't mix codecs sadly so you'll have to decide there which one to use. Now I dont think it's possible to change resolution on the fly as the entire thing is in one resolution, upscaling the 720p or downscaling the 1080p is the compromise you have to make. Either way will cause more sharpening, more blur or lower bitrate.

Another thing is that you can set in the encoder to change encoding parameters for a certain timestamp or to a area of time, though it really isn't nessesary as the encode will look weird in some places then. This will change the bitrate but not the resolution. Neither do I know the parameters off head, you'll have to check the encoder documentation to figure it out.

What is the difference between an I-Frame and a Keyframe in video encoding?

Summary: an “I-frame picture” (MPEG-2) (also known as an “intra-only frame” (VP9) or a “coded picture in which all slices are I or SI slices” (H.264)) is a compressed frame that doesn’t depend on the contents of any earlier decoded frame.

In video coding, a “key frame” (VP8, VP9) (also known as an “IDR picture” (H.264) or a “stream access point” (MPEG-4 part 12)) is a place in the video where the decoder can start decoding. It will start with a coded I-frame, but with additional restrictions to make sure it and subsequent frames can be decoded.

Full answer: there are two concepts at play here.

(1) Whether the coded frame depends on the contents of any previously decoded frame.

Video encoding schemes save bits by taking advantage of the similarities between nearby frames. Generally speaking, each frame gets divided into little rectangular pieces (known as macroblocks). For each macroblock, the coded frame first instructs the decoder on how to form a prediction about what the pixel values will be, and then (optionally) tells the decoder what the difference is between the prediction and the intended output.

The prediction can generally be formed in one of two ways: (1) based on some part of the same frame (usually required to be above or to the left of the macroblock in question, so it will have already been encoded), or (2) based on some part of a decoded frame that the decoder has already received and saved for later use.

A macroblock that is predicted in the first way (within the same frame) is called an “intra-coded macroblock.” And a coded picture that contains only intra-coded macroblocks (i.e., none of the macroblocks depend on any picture contents outside the current frame) is known as an “intra-coded picture” or “I-picture” in MPEG-2 part 2 (H.262) video, or an “intra-only frame” in VP9. A similar concept is known as an “I-slice” in MPEG-4 part 10 (H.264) video.

Bottom line: All of these terms mean what people say when they say “I-frame”—a coded frame that (unlike most frames in a compressed video) isn’t predicted from the pieces of any earlier-coded frame.

(Ultrapedantic note: No video format that I'm aware of has a precise thing called an "I-frame" — unfortunately, interlacing makes the terminology more complicated. In H.262 / MPEG-2 part 2, they want to be clear about whether they're referring to a single intra-coded picture that happens to code a frame [a frame that might be interlaced or might be progressive], or if they just mean a frame coded in a way that doesn't depend on any decoded pictures outside itself, whether that means one intra-coded picture for the whole frame, or two coded pictures each coding one field. MPEG-2 uses "I-frame picture" for the first concept, and "coded I-frame" for the second concept. H.264 gets rid of these terms and defines "I-slice" instead. VP9 calls it an "intra-only frame.")

(2) Whether a decoder can start playing the video at a given point.

Just because a coded frame is completely intra-coded (i.e. it's an I-frame picture (H.262 / MPEG-2 part 2) or intra-only frame (VP9) or consists solely of an I-slice (H.264 / MPEG-4 part 10), it doesn't mean that:

- ⓾ it stands alone (i.e., that frame can be correctly decoded without the decoder having seen some earlier part of the bitstream), or
- ⓾ it represents an entry point in the video, so that if the decoder starts at the intra-coded frame, it can then successfully decode all the subsequent frames.

Even though an intra-only frame doesn't depend on the image contents of a previously decoded frame, it can still depend on the decoder having been set up with a particular state (e.g. probability tables for compression or quantization matrices) by earlier parts of the bitstream. So even in MPEG-2, an I-frame picture cannot necessarily be decoded all by itself.

Furthermore, just because a bitstream has one I-frame picture, subsequent frames can continue to depend on the contents of earlier frames—even ones decoded long before the I-frame picture was inserted into the bitstream. So even if the decoder can decode the I-frame picture, that doesn't mean it can decode any subsequent frames in the video.

To represent a place where the decoder can actually join the video and start decoding, MPEG-2 uses the term "point of random access," H.264 uses the term "instantaneous decoding refresh (IDR) picture," and VP8 and VP9 use the term "key frame." These are places in the video that not only start with an intra-only frame, but also make sure that the intra-only frame can be decoded with the "default" decoder state, and that every subsequent frame can be decoded without reference to anything that came before the decoder joined.

The general term (defined in MPEG-4 part 12) is a "stream access point."

Why is transcoding necessary?

Suppose you get a notification from Netflix today- 'Ghoul now streaming on Netflix'. You were waiting for this eagerly.

You hurry up, finish whatever you are doing and open Netflix to watch the content. But you are highly disappointed when you see the video still buffering even after 5 hours and consuming your datapack for the day fast and not adjusting to your screen size.

This happens when the content from the content provider is streamed without using a transcoder.

What is **encoding?**

Taking content and compressing it or encoding it in the required format.

What is **transcoding?**

Taking already compressed(encoded) audio or video files, decompressing/ decoding it, trans-sizing it and trans-rating it and then recompressing/re-encoding the files is called transcoding.

Trans-sizing means changing screen resolution from,say, a pixels* b pixels to c*d pixels. For example: from 1920*1080p to 1280*720p.

Trans-rating means changing bitrate(no. of bits transferred per second) of video. Example: Converting a video from,say, 6Mbps to 2 Mbps.

The **Netflix** problem: Lets understand that problem now.

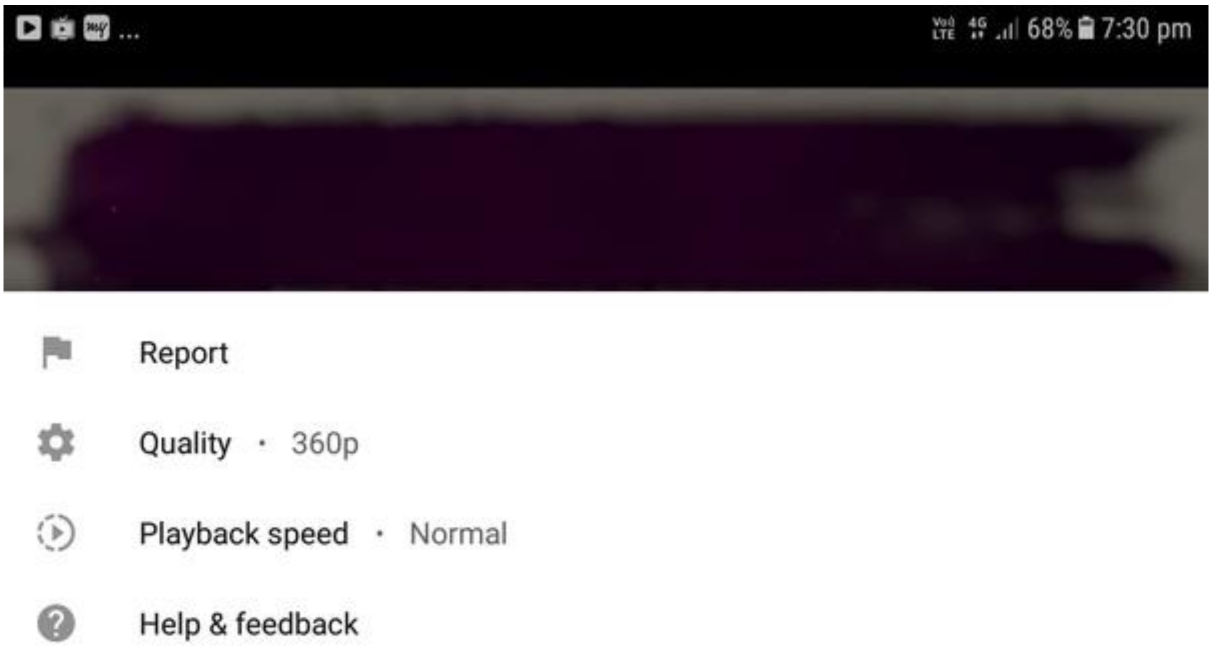
When a content provider like Netflix streams its episodes over the internet, it doesn’t stream audio and video content in just one format, rather it streams it's same video content in different frame and bit rate sizes. Why so?

This is because streaming the content only in one format makes it available only to the audience who have that format. Now, all the audience doesn't have the same data speeds, not all the users are going to view the content on the same size screen. For example: in a video streamed only in 1080p HD, audience having that frame size and likely bit rate will enjoy the series but audience with 720p and lower bit rate of a few Mbps will not be able to watch it properly which is what happened with you in the beginning.

So what is the **solution?**

Streaming the same episode’s audio and video content in more than one format is the solution so that the end users (users like us) can watch the episodes in any format, in any desired quality.

Below pictures illustrate transcoding benefits:


68% 7:30 pm

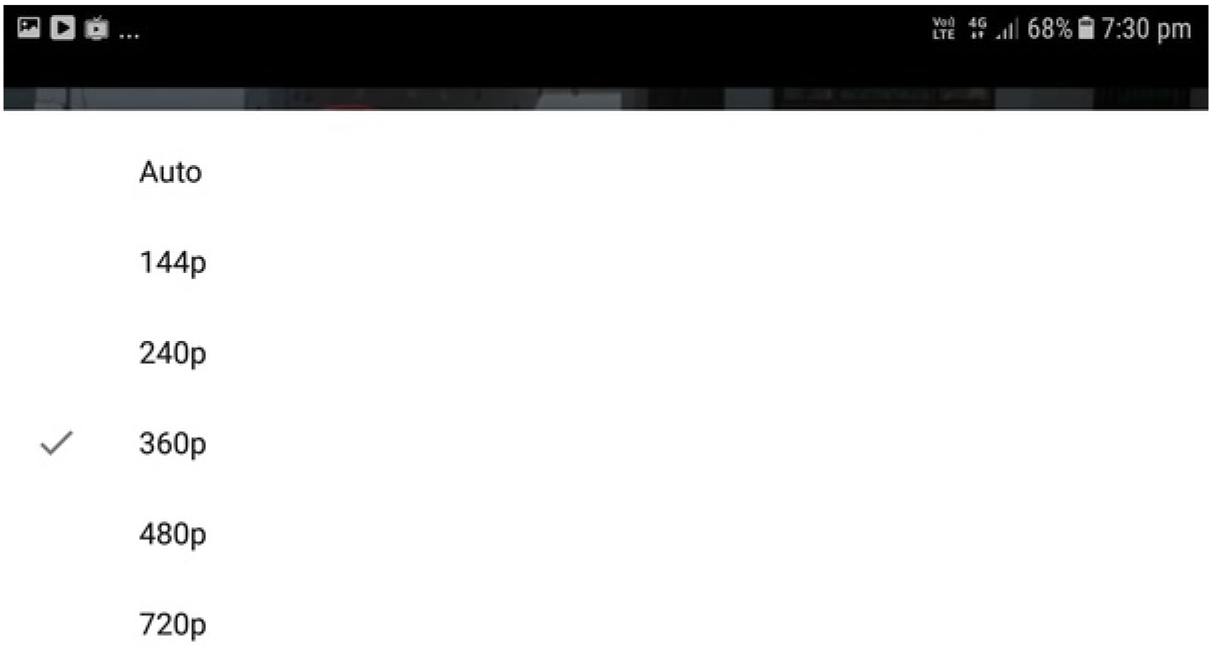
Report

Quality · 360p

Playback speed · Normal

Help & feedback

(Image source: My phone)


68% 7:30 pm

Auto

144p

240p

✓ 360p

480p

720p

(Image source: My phone)

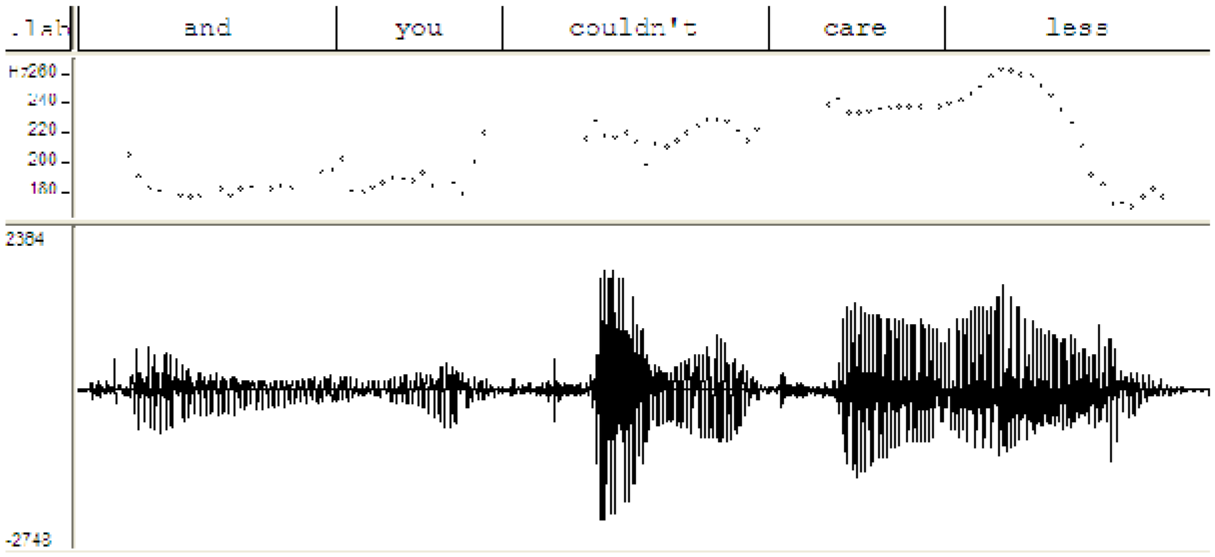
This is why the transcoding is necessary. To stream the same content over the internet throughout the world in many different formats and bitrate sizes so that the users can enjoy the content at their convinient data speeds and frame size.

Why is encoding needed?

When you transfer data from your computer to another computer it cannot transfer the data you see directly, for example if you are sending “Hello World” from one computer to another computer, it cannot transfer it as “Hello World”, it converts the “Hello World” in to binary , might be something like this

“01001000011001010110110001101100011011110010000001010111011011110111001001101100
0110010000001010” and transfers it because wires can transmit only binary data.

Or when you talk over phone or internet your voice needs to be encoded to voice frequency(which is stored in binary)



When it reaches back again to another computer it will decode the data received in to original format, binary is converted back to “Hello world!” or back to voice data

Next Story is how did you get that binary shown above from “Hello world”, this is where encoding and decoding comes. you can assign values to the characters and transfers them, just like below.

Char - HEX - binary

a - 0 - 0000000

b - 1 - 00000001

c - 2 - 00000010

d - 3 - 00000011

e - 4 - 0000100 ….. so on

so you can define you own encoding set like above and tell the person sitting over other computer that you have used the above encoding format so that when he decodes the binary he received he perfectly gets “Hello World”.

Since this is common use case, ANSI made a standard encoding formats for this purpose and asked us to use it.

The final story is to understand the difference between different encodings like ASCII, UTF-8 and so on. so initially the communication data used to contain only the characters that we see on keyboards, so ASCII supports it and contains 256 characters where each byte representing a character, so there is a limitation for ASCII encoding where you can’t send Chinese, Japanese or Telugu, so UTF-8 is introduced which supports almost all the languages in the world, UTF-8 encoding is very smart and uses memory efficiently, and so other encodings. All the web browsers uses UTF-8 encoding, so it is able to support different languages and so we are able to see

Facebook emojis and lot other. so if browser uses ASCII, we would end up seeing only 256 characters. look this video for better understanding.

Why is 10-bit video better than 8-bit video?

A subpixel is effectively a single-color LED with controllable brightness. For every pixel, you have red, green and blue sub-pixels, hence the term RGB. Despite this lack of variety, you can actually produce millions of colors by varying the intensity of each subpixel: having all three of them at full blast will create white. Having red and green on at full blast with no blue will get you yellow. By varying the intensity of each sub-pixel, you can get a huge variety of colors.

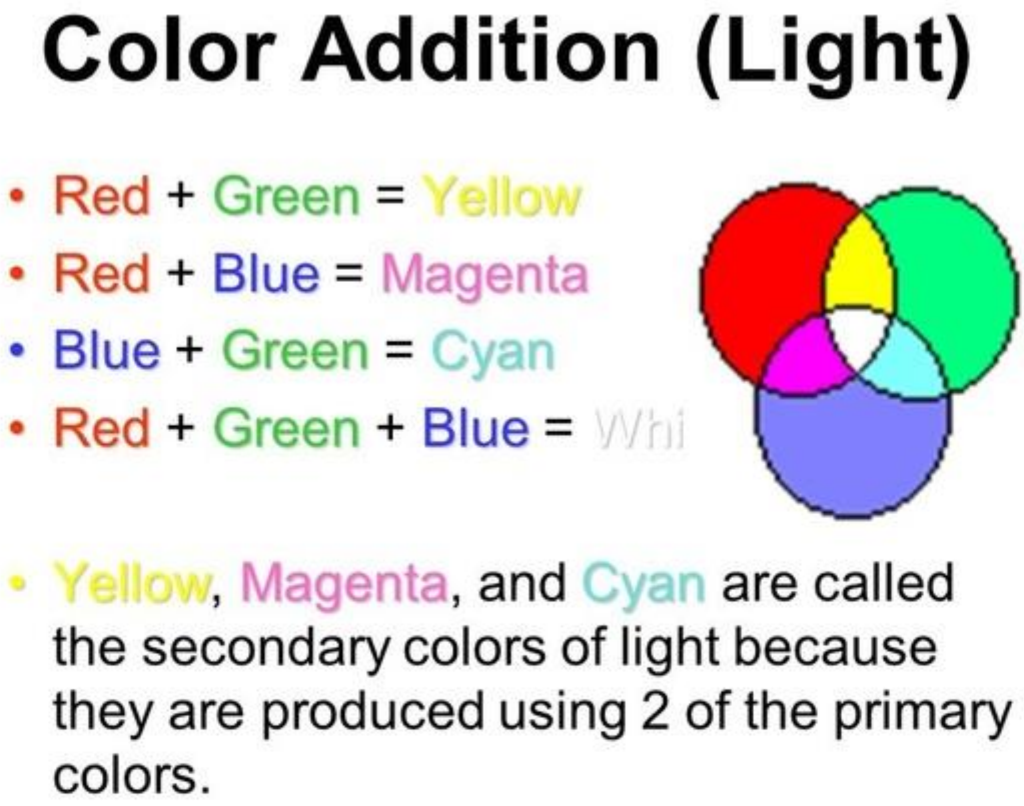

Color Addition (Light)
• Red + Green = Yellow
• Red + Blue = Magenta
• Blue + Green = Cyan
• Red + Green + Blue = Whi
• Yellow, Magenta, and Cyan are called the secondary colors of light because they are produced using 2 of the primary colors.

image: Light and Color

The controller hardware in the display takes care of dealing with those lights. But to do that, it must first receive video information from a computer, Blu-ray player, etc.

Most displays on the market will only allow each sub-pixel to take on one of 256 light intensity levels, ranging from fully off (0) to fully on (255). Times that across 3 subpixel channels (Red, Green and Blue) and you get the ~16.8 million colors possible on such a display. Computers and media players deal with this color information digitally: it takes 8bits to represent each subpixel's brightness (2^8 = 256) and 24bits to represent all 3 subpixel channels that make up a single pixel (2^24 =~16.8 million).

This is where the term "8bit" picture comes from: displays which support 8bit color will have 3 subpixel channels with brightness levels each represented by an 8bit number. The number of brightness levels per subpixel directly affects the number of colors each pixel can represent, as ???????????=(??????????????)3

. Given that computers don’t typically interface with displays at the sub-pixel level (instead dealing with pixels), you’ll also hear “24bit color” being thrown around when talking about those displays: that’s simply the per-subpixel bit count times the number of channels, of which there are 3. That aggregate figure can also be used to figure out total number of colors more directly (2^24), and can avoid some finicky encoding schemes in older systems.

Given that the amount of light levels each sub-pixel can take on directly influences the total number of colors a display can represent, why not go for more than 256? Break beyond 8-bit color?

That’s precisely what 10-bit displays do: each sub-pixel can now accommodate any one of 1024 different light levels. More light levels per subpixel means more colors per pixel. More light levels means more granularity in terms of the available colors. You need 10 bits to represent these sub-pixels, hence the term “10bit color”. Over 3 channels, we get 30bits per pixel, which translates to just over 1 billion colors, which is a fair bit higher than the ~16.8 million on our 8bit display.

10bit screens can display far more colors than 8bit ones, which can have a positive effect on your user experience: overall, the colors being displayed will be closer to what was intended by the content creator.

Why is RGB so popular?

Because you are on a computer, so you may get some bias on color system used in computers. Really on video YUV [with non standard definition of UV], or other similar format are used.

On some industries, Pantone is the popular choice to define a colour.

In general: if you do not have a luminous colour, so the most common case: the colour you see is determied also by the type (“colour”) of the light (illuminant), you need to specify colour in much better details (3 parameters are not enough).

With videos, a YCrCb (or similar) is better, because one could split Y with the other channels, and so using different compression on the different channels. This give a much better compression with same quality.

Screens uses RGB, so we are used (e.g. for the web) to use RGB colour codes. The fact that more natural systems (e.g. HSV) are not much standardized on computers (or better: we use transformations which are good for colour pickers, but wrong on the real sense of HSV, so losing the “natural” part.

Note: on web, now one should use CSS, and modern CSS preprocessors added function to handle colour also in HSV systems, because it is much more generic and easy: you just define one basic colour, and you get automatically the dark, the unsaturated, etc. variations (think about active buttons, menus, etc., or the disabled one, etc.). It is much more easier than to calculate every time. [In any case, at the end, one should fine tune the most important of them, possibly with RGB values), The RGB system for encoding colors is popular because that's generally the set of primaries that will result in the greatest range of available colors (i.e., the maximum color gamut) possible in a given additive color (usually this means "light emitting") display device. Of course, there are a lot of possible choices for the specific red, green, and blue primaries to use in a given system, which is why specifications for standard color systems such as sRGB, AdobeRGB, Rec. 2020, etc., are necessary.

What is the RGB colour model in a computer?

RGB (red, green, and blue) refers to a system for representing the colors for use on a computer display. pink, inexperienced, and blue can be combined in diverse proportions to attain any color in the visible spectrum. tiers of R, G, and B can every variety from zero to a hundred percent of full depth. every stage is represented by way of the variety of decimal numbers from zero to 255 (256 stages for every colour), equal to the range of binary numbers from 00000000 to 11111111, or hexadecimal 00 to FF. the overall range of available shades is 256 x 256 x 256, or 16,777,216 feasible colours.within the Hypertext Markup Language (HTML), the shade for a web page background or textual content font is specific via an RGB price, expressed with six digits in hexadecimal format. the first and 2nd digits represent the red level; the third and fourth digits represent the inexperienced degree; the 5th and 6th digits constitute the blue level. so that it will genuinely display the colours for all possible values, the pc show machine must have 24 bits to explain the coloration in every pixel. In show structures or modes that have fewer bits for showing shades, an approximation of the required color will be displayed.

Why in RGB the biggest number for any particular color is 255?

All the number crunching in all other replies is absolutely correct.
The binary approach of computer counting loves the powers of 2.

But none of them explains: why 0 to **255** and not **64** (2^6) or **1024** (2^10) ?

The human eye consists of nerves (rods and cones) which are sensitive to either luminance (L) or colors (with an a and b value). Hence the 'mother' of all visual numbers: the **L*ab** system ! Slightly different variations of this system are called HSB or HSL (Hue, Saturation, Brightness or Lightness). And all of these can also be recalculated into the mixing of primary colors like RGB light (Red, Green, Blue) or even CMY inks (Cyan, Magenta, Yellow).

But whatever system you use, the light-sensitive nerves (rods) in our eyes can only distinguish about **150** gradations, from the darkest to the lightest. And most of them are at their best in the mid-range. (We are far less sensitive to differences in colors.)

So using 2^7 would yield only 128 gradations, which is not even enough for the total number of necessary gradations, not to mention the need for a more refined mid-area. But 2^8 with its **256** gradations is more than enough, and even leaves room for some often applied corrections in the darkest and lightest areas of an image.

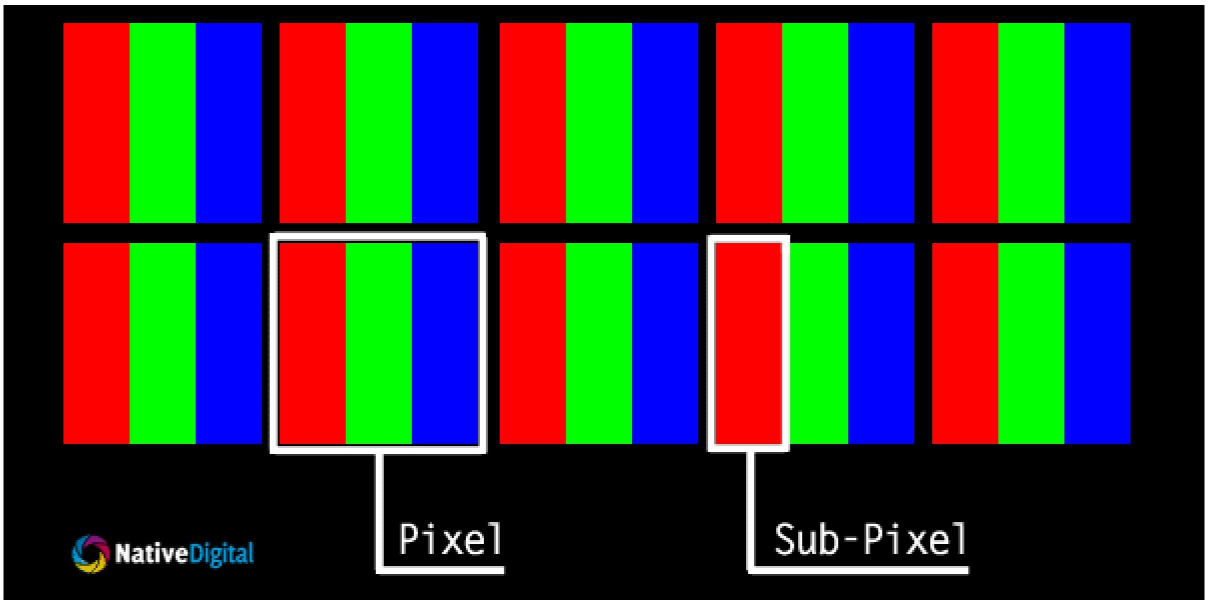
That's why a final result of an image (like a JPEG) will do fine with just 2^8 per RGB channel. But if you're going to retouch the image and want to brighten up dark areas or get some darker tones in a too bright area, you need an image with maybe 2^10 or even more values (gradations) in its channels, to let the result appear smooth enough in these forced areas of darkest and lightest gradations. That's why photographers prefer these "raw" file formats, which use more "bits" in their measurements.

If you would pump up the contrast in a 2^8 RGB image (typically a JPEG) too much, banding might appear ! (Using too much JPEG compression also causes similar deteriorations.) Such artifacts are very quickly noticeable in slowly changing large areas, like bright skies, soft shadows, and curved areas in white clothes and skin tones, etc.

It doesn’t. RGB values for 24bit color depth do, but that’s hardly the only option out there.

Most monitors you’ll see today have a color depth of 24bits: that is each pixel can show up to 2^24 (~16.7M) colors. This 24bit value is separated into 3 channels, each one representing either Red, Green or Blue (RGB). Because 24/3= 8, each one of these channels is 8bits wide. That is, each display sub-pixel can take on 2^8 different levels of brightness, which really are 255 different shades of the specific color of said sub-pixel. That is, the R value in a 24bit RGB value directly controls how bright the sub-pixel will be for the affected pixel on your screen.

Given that there are only 8bits per channel, it makes sense that there’s a cap of 255 per color: you simply can’t encode a higher value in an 8bit number (hence why 255 corresponds to max brightness).



But everything i’ve talked about applies to 24bit color. There are other standard out there. For example, you might have heard of 10bit HDR before. That relates to color depth, with each RGB value being 30bits, hence 10 bits per channel.

With 10bits per RGB channel, you can actually have values going up 1023.

Of course, both 10bit 1023 and 8bit 255 are the maximum values for their respective color depths. Both of them correspond to the maximum brightness for a subpixel. But 10bit color offers greater color granularity because there are simply more RGB values.

Because it's a format for computers, and 0-255 is the range a single byte can represent. This allows the color values to be stored in a clean number of bytes, which is a huge benefit for computers to work with. Going with fewer bits per pixel has a noticeably degraded color quality, while going with more will drastically increase the data to represent images and movies without being a noticeable visual improvement

255 fits into an 8 bit number where 0 is the first digit (2^8 -1) = 255

For a 24 Bit number, Red Green Blue can be defined as (0-255)(0-255)(0-255).

For a 32 bit number, an alpha layer (usually transparency but can be reflectiveness) can be added defining how easy it is to see the image behind the current image.

Note: it is just web colours, and it was choose so to have one colour per byte: easier to process.

RGB before web was often cap to 100 or to 1, but using decimal points. Or better: the white was set so that RGB was 100, 100, 100, or 1, 1, 1. We are not blocked on white: we can have more luminous colours (like in real world).

And in fact, we use also values below 0, for colours “out of gamut”. These colour cannot be created by a screen with R, G, and B subpixels which emit the same colour has the choosed RGB standard, but e.g. a better television could display them.

Note: the values are also gamma corrected (it is a must, for so small range).

It's not—it's just and extremely common upper value these days. With 8 bits per color channel (24 bit color), the maximum number of shades per channel is 256 (0–255). On 10-bits-per-channel color systems (30 bit color), which are becoming more common, there are 1024 possible shades per channel.

Each channel (red, green and blue) is a bit of information. A bit is an 8 binary number string. Thus 00000000 through to 11111111 is the full range of a single bit.

Within a bit, 256 variations can be represented—each value can be either 0 or 1. This gives us 16 (8 * 2). And thus, just as 10 * 10 gives us 100—representing the full range of numbers in a decimal 100, 16*16 gives us 256—the total variations capable within a binary bit.

As 0 is considered a value, this gives us a range of 0 to 255.

The colors red, green, and blue are 8-bit each. Since 2^8 is 256, each color has a range of 0-255

Why is color measured on a scale of 0 to 255?

Colour scales and definitions were standardised in the times of 32-bit processors. RGB colour, being the integral and core part of display and graphics processors, has been addressed and defined in the terms of the processors’ arrays.

0–255 is 256 units of colour. 32-bits of each colour with 8-levels of colour grading gives you a full scale of 256 units of each colour definition.

So, Red would be (255,0,0), Green (0,255,0), and Blue (0,0,255), so on for all the colour combinations.

Simply because colour of an object in digital systems is allocated in units of bytes of data in memory.

Now according to the RGB color model we can have Red, Green or Blue at different amounts to give a particular colour. (Remember RGB is an additive colour model).

Each byte has 8 bits and thus using these 8 bits there are 255 possible numbers possible to generate. [Remember (11111111) in binary is equivalent to 255 in decimal system]

Now using this scheme, different amounts of 'Red', 'Green', and 'Blue' gives us the final colour of our object!

Color Spaces in Frame Grabbers: RGB vs. YUV

RGB formats are usually straightforward: red, green, and blue with a given pixel size. RGB24 is the most common, allowing 8 bits and a value of 0-255 per color component. RGB frame grabbing is best if your image processing and/or storage requirements call for it. RGB capture can be saved directly as a bitmap by adding a simple header. Windows XP GDI (Graphic Display Interface) and Linux GUIs work with and display RGB bitmaps.

The problem with RGB, however, is that it is not the best mapping for representing visual perception. YUV color-spaces are a more efficient coding and reduce the bandwidth more than RGB capture can. Most video cards, therefore, render directly using YUV or luminance/chrominance images. The most important component for YUV capture is always the luminance, or Y component. For this reason, Y should have the highest sampling rate, or the same rate as the other components. Generally, 4:4:4 YUV format (equal sample per component) is a waste of bandwidth because the chrominance can be adequately sampled at half the sample rate of the luminance without the eye being able to notice the difference. YUV 4:2:2 and 4:2:0 formats are generally used with the preference on 4:2:2. 4:2:2 means the chrominance components are sampled horizontally at half the rate of the luminance. 4:2:0 means the Cb (blue chrominance) and Cr (red chrominance) components are sub-sampled at a factor of 2 in the vertical horizontal directions.

Capturing to YUV (or YCrCb), besides being more efficient, is best when the images are to be rendered by a standard video display directly. Additionally, some image compression algorithms, such as JPEG, directly support YUV, so there is no need for RGB conversion. In the YCrCb model, 0x80 is the black level for Cr and Cb, and 0x10 is the black level for Y.

Most DirectX 8, 9, and 10 hardware supports direct display of YUV images. With the correct format, you can render directly to the video card without transforming the image with MMX instructions or other (preferably optimized) code.

The most common video cards support packed YUV mode, YUY2 (4:2:2) and/or NV12 (4:2:0) natively. The byte ordering for YUY2 is as follows:

Y0 U0 Y1 V0 Y2 U2 Y3 V2 Y4 U4 Y5 V4

YUY2 format (4:2:2 = width * height Y values, width * height / 2 Cr,Cb values = 2 * W * H total bytes)

NV12 is a quasi-planar format and has all the Y components first in the memory, followed by an array of UV packed components. For a 640x480 NV12 image, the layout is as follows:

```
Y0  Y1  Y3  …   …  …  …
…
…
…           …   …  …  …      Y301798  Y301799
U0  V0  U1  V1  …  …  …
…   …   …   …   …  …  …  …   U76799   V76799
```

NV12 format(640 * 480 Y values, 640 * 480 / 4 Cr and Cb values = 3/2 H * W total)

Many hardware capture boards are most efficient using planar YUV mode. They capture YUV on a plane-by-plane basis. Sensoray's 2255, for example, can capture directly into 4:2:2 YUV422P format. (It also supports YUY2 and RGB formats in the SDK.) The planar format is advantageous when the final YUV format is not known, because each component can be directly referenced as an array. A separate pointer can simply point to each of the component arrays. The packed formats do not allow this. Additionally, from the planar format, it is trivial to create the packed YUV formats such as NV12 or YUY2 with MMX optimized instructions for DirectX VA (DirectX Video Accelerated) or SDL display in Windows or Linux respectively.

```
Y1  Y3  …   …   …  …
…

…           …   …  …  …      Y301798  Y301799
U0  U1  U2  U3  …  …  …      …        …
                …  …  …      U76798   U76799
V0  V1  V2  V3
…   …   …   …   …  …  …  …   V76798   V76799
```

YUV planar (4:2:2), 640 x 480 image

What is 4:4:4 screen, and what is 4:2:2 screen?

In video processing

4:4:4 refers to the case where Luma and Chroma have the same spatial resolution.

4:2:2 refers to the case where Chroma has 1/2 of the Luma resolution horizontally

4:2:0 refers to the case where Chroma 1/2 of the Luma resolution both horizontally and vertically.

Example: 720 x 480

4:2:0 → Chroma (Cb or Cr) is 360 x 240

4:2:2 → Chroma is 360 x 480

4:4:4 → Chroma is 720 x 480