**V4L2**

**By**

**KATTHI SHASHIKALA**

**Contents**

**1. Video Encoder flow**

**2. Video Decoder flow**

**3. What is RGB**

**RGB storage format**

**4. What is YUV**

**Types of YUV**

**5. Chroma subsampling**

**6. What is Bitstream**

**What are the bitstream/Video formats**

**H264**

**7. Video Container formats**

**What are types of container formats**

**8. What is metadata**

**9. Compression:**

**10. What is Pixel**

**11. What is Video Frame**

**12. What is Aspect ratio**

**13. 720p, 480p, 1080p -> What is P**

**14. 1080i --> What is I**

**15. Resolutions and types of resolutions**

**16. What is High FPS video**

**17. What is low FPS video**

**18. What is timelapse video**

**19. What is Hardware Video decoder/encoder**

**20. What is Sofware Video Decoder/encoder**

**21. AV sync (Audio Video Sync); What is? When AV sync issue can come**

**22. No sound in a video issue**

**23. What is Video upscaling**

**24. What is Video downscaling**

**25. What is Macro block**

**26. What ffmpeg**

**27. What is ffprobe**

**28. What is I frame**

**29. What is P frame**

**30. What is B frame**

**31. What is V4l2**

**Structure of a V4L driver**

**Common implementation of a v4l2 application**

**32. Transcoding**

**33. What is O/p Plane**

**34. What is Capture Plane**

**35. What is qbuf**

**36. What is dqbuf**

**37. How to run test app and collect the logs for test app commands**

**Full log**

**Video Training**

Encoding-->Video recording

Decoding-->Video playback

The digital video is encoded to meet proper formats and specifications for recording and decoded(decode the formats and specifications of encoded data)to playback through the use of video encoder and decoder

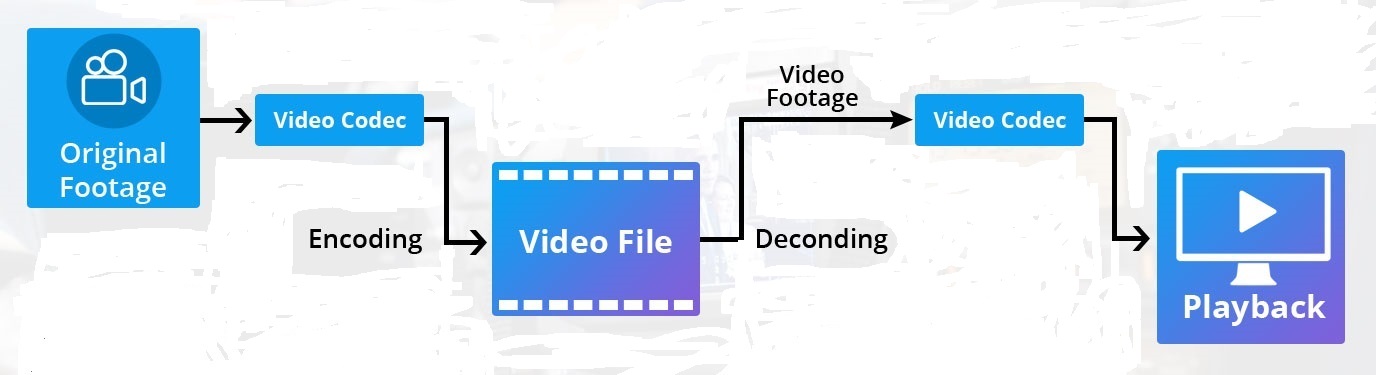


Fig: Encoding and Decoding of video file

**Source link:** <https://www.muvi.com/blogs/best-video-codec-for-streaming.html>

**Video Encoder flow:**

camera HW (BAYER format) --> camera ISP (BAYER -> RGB -> YUV) --> Video Encoder (YUV -> bitstream standard formats H264, H265) --> File (H264 -> mp4)

**Video Decoder flow:**

File (mp4 -> H264) --> Video Decoder (H264 -> YUV) --> Display(YUV -> RGB) --> (RGB) Display HW

**What is RGB?**

RGB (red, green, and blue) refers to a system for representing the colors to be used on a computer display.

Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum.

In this model, the colors red (R), green (G) and blue (B) are added together at different intensities to produce millions of different colors on modern video display screens.

Levels of R, G, and B can each range from 0 to 100 percent of full intensity.

Each level is represented by the range of decimal numbers from 0 to 255 (256 levels for each color) and 0 is taken to be black,255 is taken to be white.

Equivalent to the range of binary numbers from 00000000 to 11111111, or hexadecimal 00 to FF.

The total number of available colors is 256 x 256 x 256, or 16,777,216 possible colors.

It is three plane mode R and G and B.

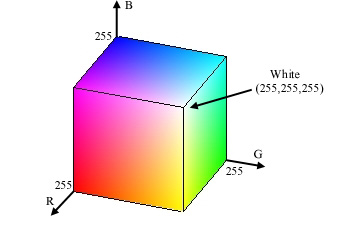


Fig :RGB-planes

**Source link:** <https://msu.edu/~schanerr/Color_Figures.html>

R + G + B = RGB

1,1,1,1,1 1,1,1,1,1 1,1,1,1,1 3,3,3,3,3

1,1,1,1,1 1,1,1,1,1 1,1,1,1,1 3,3,3,3,3

1,1,1,1,1 1,1,1,1,1 1,1,1,1,1 3,3,3,3,3

1,1,1,1,1 1,1,1,1,1 1,1,1,1,1 3,3,3,3,3

**RGB storage format:**

1. RGB555 is a 16 bit RGB format. Each component is represent by 5 bits, and the remaining one is not used.

2. RGB565 is also a 16 bit RGB format, but R used5 bits, G used 6 bits, and B used 5 bits.

3. RGB24 is a 24 bit RGB format. Each component used 8 bits with a range of 0-255.

4. RGB32 is a 32 bit RGB format with 8 bits per component and the remaining 8 bits as Alpha channels.

This alpha channel can change the transparency of the picture

**What is YUV?**

YUV is a color space typically used for color image/video processing.

It encodes a color image/video taking into account properties of the human eye that allow for reduced bandwidth for chroma components without perceptual distortion.

It was first used in analog TV broadcasting. Color spaces YUV, YIQ, YCbCr, and YPbPr all belong to the YUV family.

Y' stands for the luma component (the brightness) and U and V are the chrominance (color) components and It`s a two-plane mode, Y and UV.

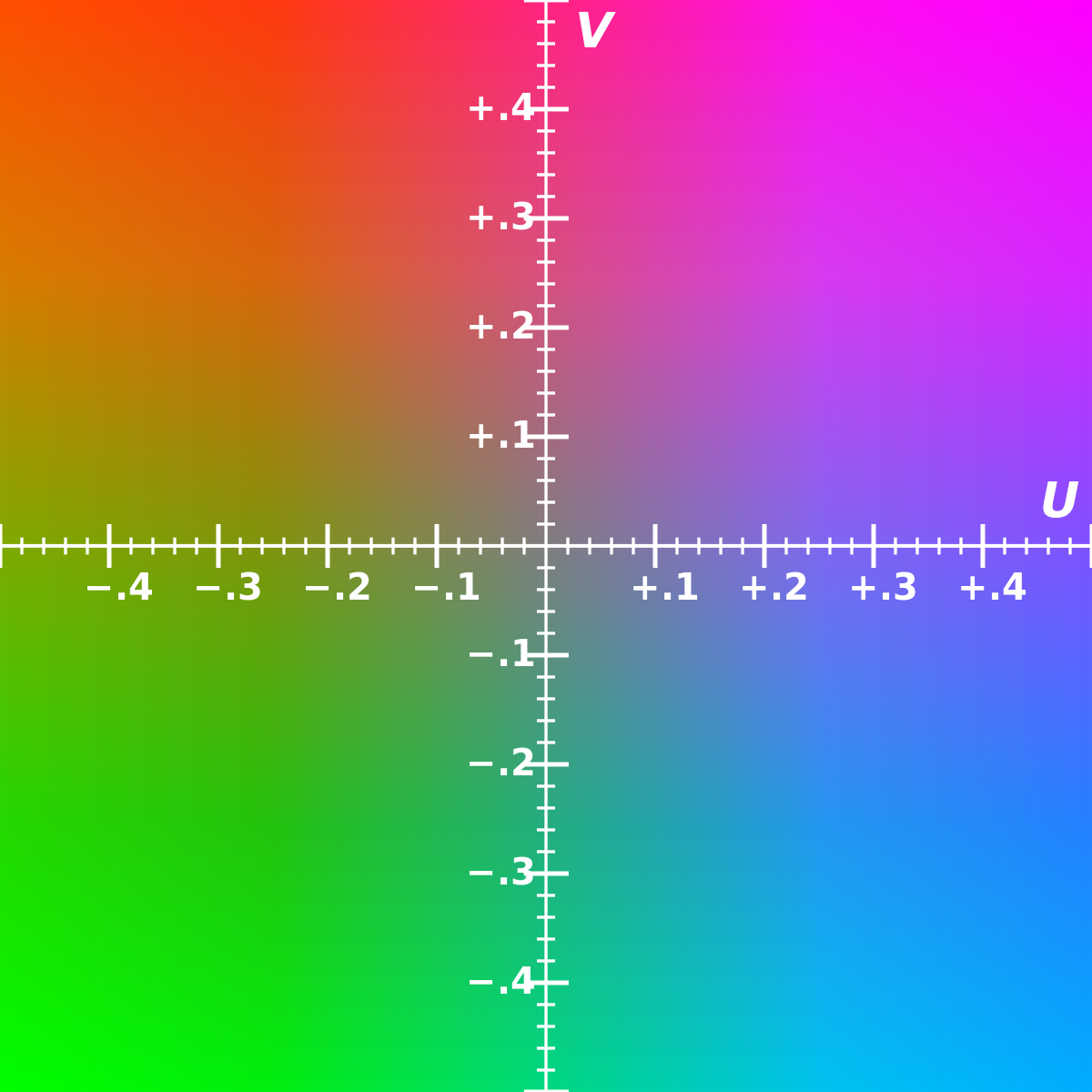


Fig:YUV-planes

**Source link:** <https://commons.wikimedia.org/wiki/File:YUV_UV_plane.jpg>

Y + U + = YUV

1,1,1,1,1 2,2,2,2,2 3,3,3,3,3

1,1,1,1,1 2,2,2,2,2 3,3,3,3,3

1,1,1,1,1 2,2,2,2,2 3,3,3,3,3

1,1,1,1,1 2,2,2,2,2 3,3,3,3,3

**Chroma subsampling:**

Chroma subsampling is used to reduce the amount of data in a video signal while having little or no visible impact on image quality.

The main formats such as:

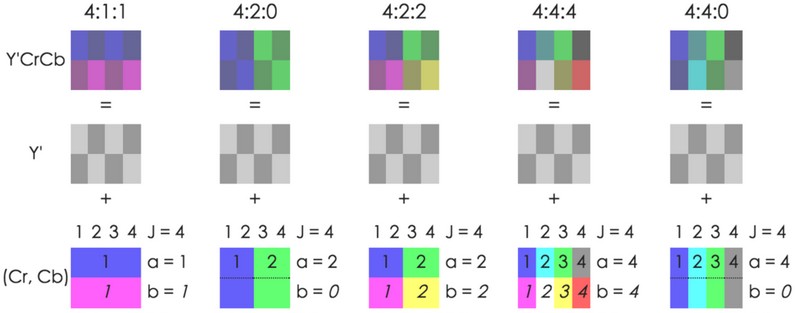
ycbcr 4:2:0, ycbcr 4:2:2, ycbcr 4:1:1, ycbcr 4:4:4

Fig:YCbCr-formats

**Source link:** <https://medium.com/@sddkal/chroma-subsampling-in-numpy-47bf2bb5af83>

**Compression:**

To get the smaller size with same quality we need compression.

RGB 🡪YUV444 🡪 Conversion

YUV444 🡪YUV420 🡪 Chrome Subsampling

YUV 🡪 H264 🡪 This is specific to standard

**Bitstream format?**

A bitstream format is the format of the data found in a stream of bits used in a digital communication or data storage application.

The term typically refers to the data format of the output of an encoder, or the data format of the input to a decoder when using data compression.

Video Encoder (YUV -> bitstream standard formats H.264, H.265)

Video Decoder (H.264 -> YUV)

**These are the some bitstream formats**:

H.261,H.262,H.263,H.264 / AVC,H.265 / HEVC,H.266 / VVC,DV,VC-1,VC-2,VC-3,VC-5,VP3,VP6,VP7,VP8,VP9,AV1.

**H.264(advanced video coding)**

it is proprietary code.

It uses low bit rate than H.265.

The decoder operates with a sequence of bits received in a specific format. The binary stream is structured and divided into packets. On the upper level, there is separation of the stream on NAL-packets.

**NAL** stands for Network Abstraction Layer.

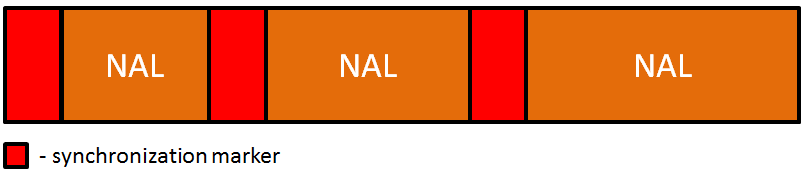


Fig: Stream separation on NAL packets.

NAL-type defines what data structure is represented by current NAL-packet. It can be slice, or parameter set, or filler and so on.

The first byte of a NAL-packet is a header that contains information about the type of packet

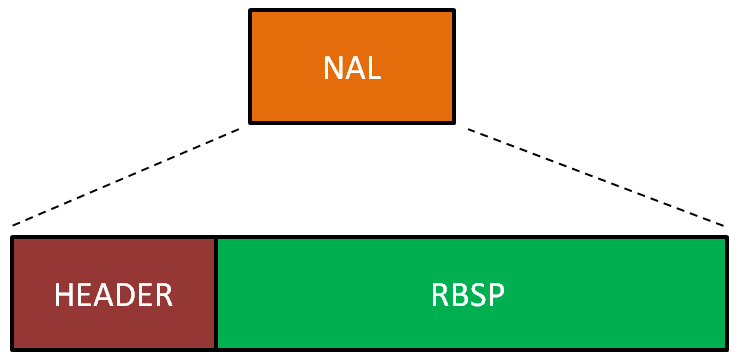


Fig: NAL structure

NAL-packet identified as **RBSP** (Raw Byte Sequence Payload). RBSP describes a row of bits specified order of SODB (String Of Data Bits).  
According to the ITU-T specification if SODB empty (zero bits in length), RBSP is also empty.

The first byte of RBSP (most significant, far left) contains the eight bits SODB; next byte of RBSP shall contain the following eight SODB and so on, until there is less than eight bits SODB. This is followed by a stop-bits and equalizing bit

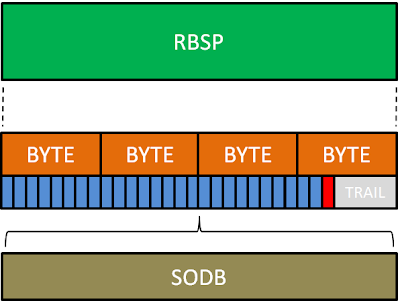


Fig: Raw Byte Sequence Payload (RBSP)

Any coded image contains slices, which in turn are divided into **macroblocks**. Most often, one encoded image corresponds to one slice. Also, one image can have multiple slices.

The slices are divided into the following types:

|  |  |
| --- | --- |
| **Type** | **Description** |
| 0 | P-slice. Consists of P-macroblocks (each macro block is predicted using one reference frame) and / or I-macroblocks. |
| 1 | B-slice. Consists of B-macroblocks (each macroblock is predicted using one or two reference frames) and / or I-macroblocks. |
| 2 | I-slice. Contains only I-macroblocks. Each macroblock is predicted from previously coded blocks of the same slice. |
| 3 | SP-slice. Consists of P and / or I-macroblocks and lets you switch between encoded streams. |
| 4 | SI-slice. It consists of a special type of SI-macroblocks and lets you switch between encoded streams. |
| 5 | P-slice. |
| 6 | B-slice. |
| 7 | I-slice. |
| 8 | SP-slice. |
| 9 | SI-slice. |

Table :Slice types

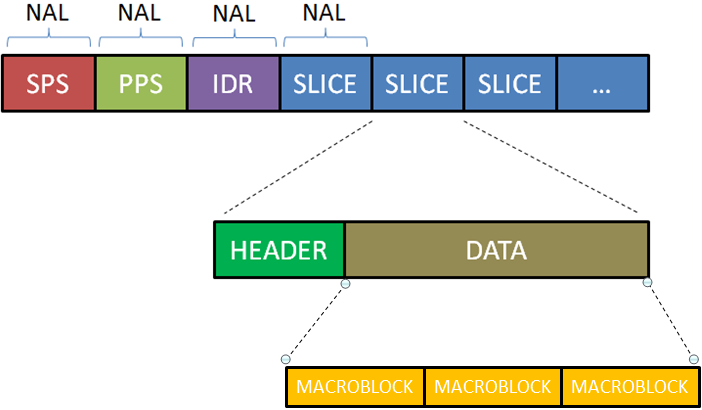


Fig: Detailed H.264 stream

**Source link:** <http://gentlelogic.blogspot.com/2011/11/exploring-h264-part-2-h264-bitstream.html>

**H.265(high efffiency video coding):**

it is proprietary code.

we can get 50% smaller size of file than H.264.

For live streaming it is better than H.264,but we need more hardware resource than H.264.

**VP8:**

It is open source/google

We can get smaller file size than H.264.

**VP9:**

open source by google

we can get 50% smaller size than VP8

It uses webm and IVF containers.

**Video container formats(file)?**

The container is the wrapper for the streams. It presents a single interface that media players and tools can interact with. It contains your video, audio, and metadata (vital data such as captions, SEO, and vital information that pieces the video together for playback).



Fig: Video container format

**Source link:** <http://www.pitivi.org/manual/codecscontainers.html>

It can also be called a file extension since they are often seen as file names, such as AVI, MOV, or MP4. While AVI and MOV 3GP,3G2,ASF,AVI,DMF,EVO,F4V,FLV,MP4,PS,TS,MXF,M2TS,Video file.

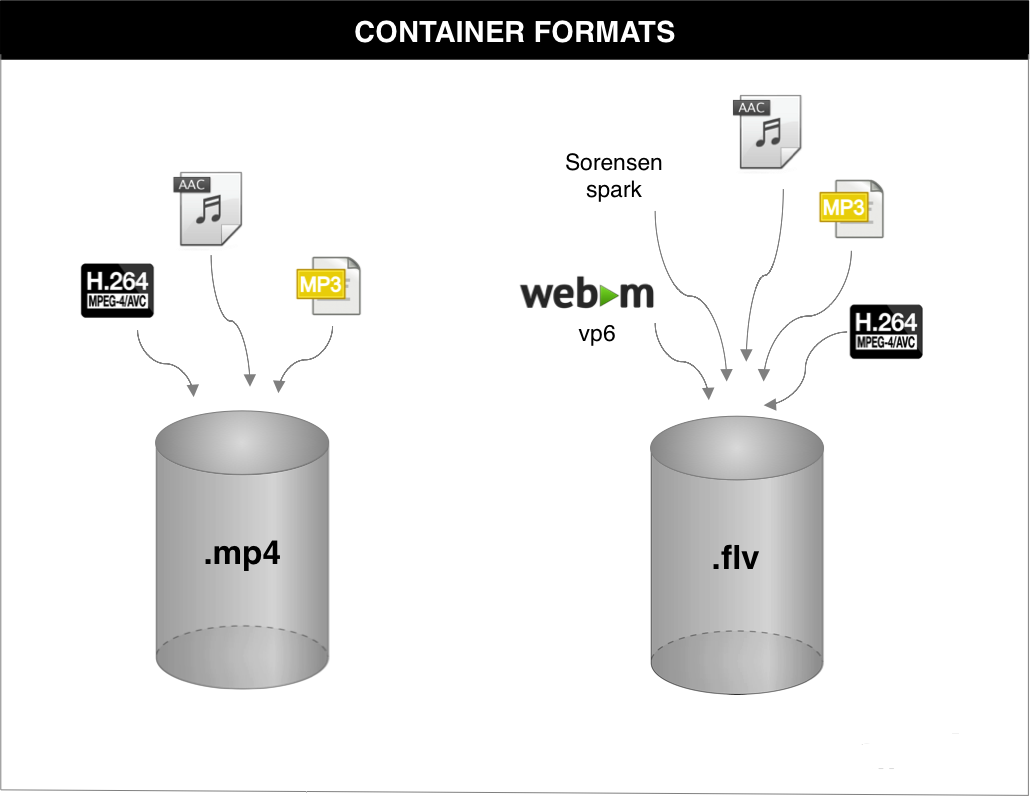


Fig: Different container formats

**Source link:** <https://streamshark.io/blog/understanding-codecs-and-formats/>

**What is Metadata?**

Metadata is data about data that includes additional information about your video that puts the actual video file into a context and enhances its presentation. It includes, among other things, title, description, and publishing time.

If we take one file, It contain information like:

**type:** video/audio

**format:** h264, h265

**fps:** 30 --> Frames per second

**Resolution:**1280x720=WidthxHeight

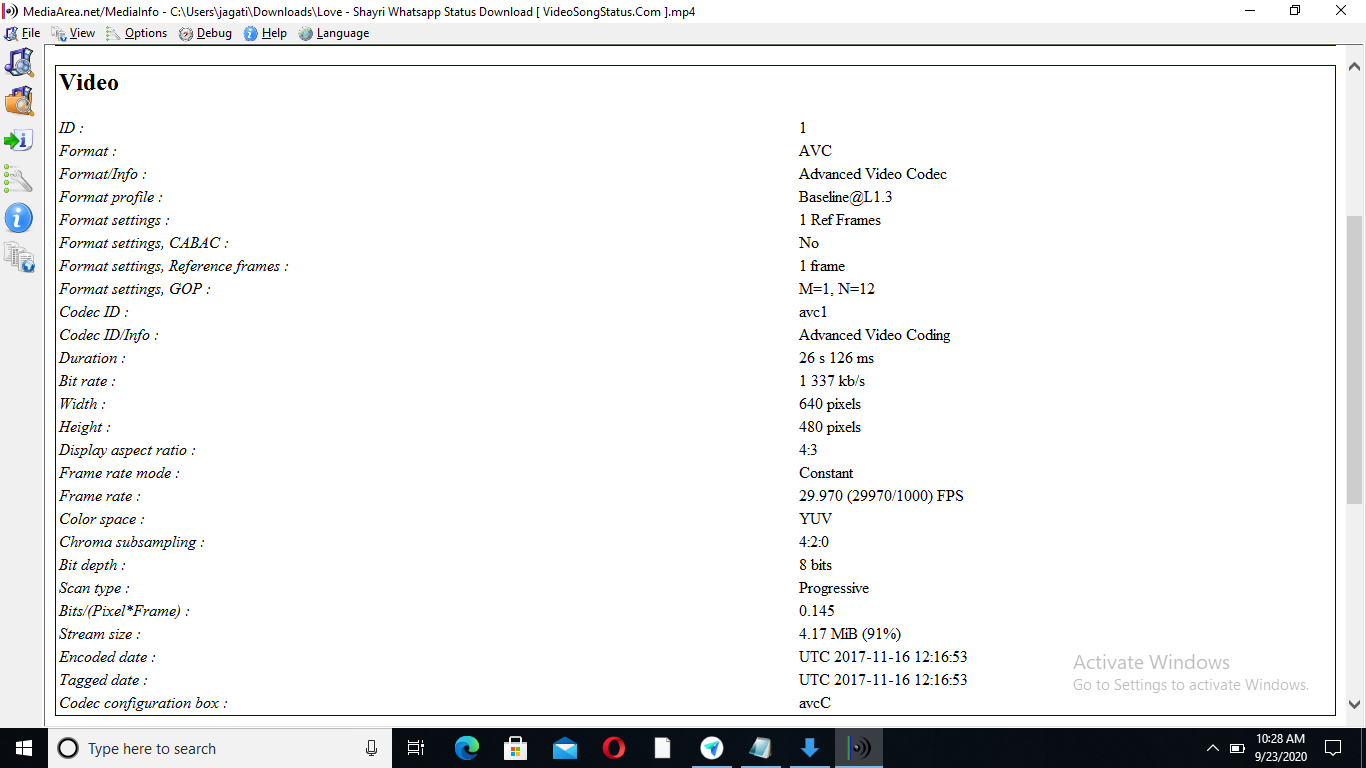


Fig: Meta data

**What is Pixel?**

Pixel is the small single component of a digital image.

A pixel is represented by a dot (or) square on a computer monitor display screen, the quantity and size and color combination of pixels will varies with display resolution.

For example, a computer with a display resolution of 1280X768 will produce a maximum of 98,3040 pixels on a display screen.

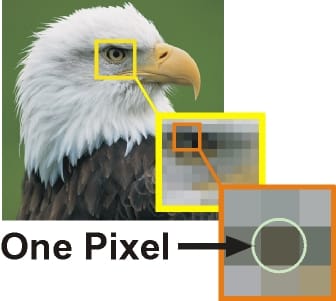


Fig: Pixel in a digital image

**Source link:** <https://www.tes.com/lessons/_kjvlPLmCriLSw/minecraft-selfies>

**What is Video Frame?**

Frame is the one of the many still images which compose the complete moving picture.

Video frames were represenated as analog waveforms in which varying voltages represented the intensity of light in an raster scan across the screen.

Frame rate is the no.of frames displayed per unit of time is measured in frames per second(fps) also measured in hertz(HZ).

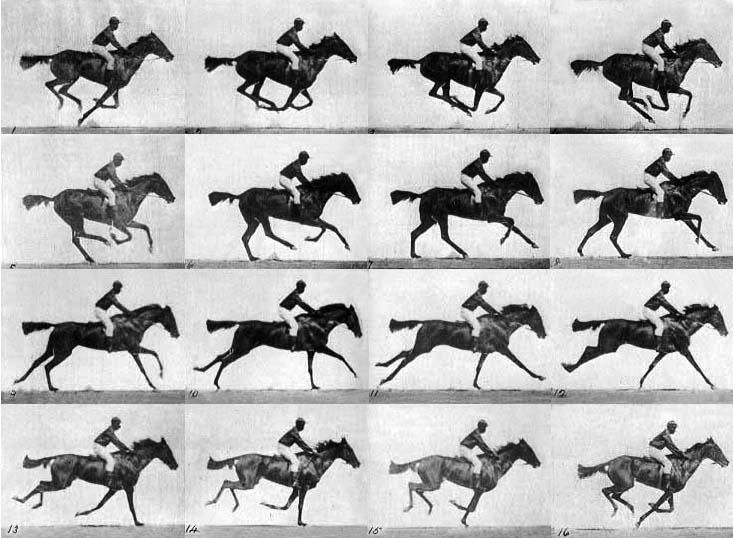


Fig: Frames in a video of horse riding

**Source link:** <https://nofilmschool.com/2015/12/check-out-15-minute-history-lesson-frame-rates>

**720p, 480p, 1080p -> What is P ?**

p stands for progressive scan.

Means entire frame transmitted at once,all the lines in frame are drawn at once to fill screen.



Fig: Progressive scan

**1080i --> What is I ?**

I stands for intrelaced scan

Means half the frame is transmitted at a time like odd(1.3.5...) or even(2,4,6...) lines transmits at a time in frame

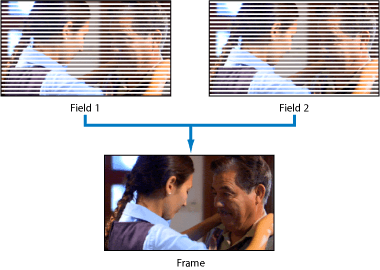


Fig: Interlaced scan

**Source link:** <https://www.synopi.com/interlaced-and-progressive-video/>

**What is I frame?**

Intra coded frame

It constructed from itself only and it is used as a reference for other frames

It never constructed from other frames.

**What is P frame?**

Predictive frame

It is constructed from previous i and p frames.

It contains previous frame changes.

It can construct from another P-frame.

**What is B frame?**

Bi-directional predictive frame.

It is constructed from past and future frames.

It is not constructed from another B-frame means never used as a reference.

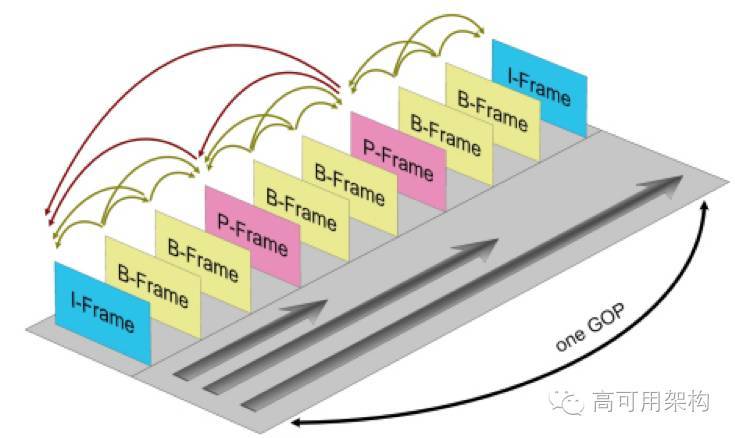


Fig: I and B and P frames in a video

**Source link:** <http://www.bnoack.com/index.html?http&&&www.bnoack.com/video/GOP.html>

In [video coding](https://en.wikipedia.org/wiki/Video_coding), a **group of pictures**, or **GOP structure**, specifies the order in which [intra-](https://en.wikipedia.org/wiki/Intra-frame) and [inter-frames](https://en.wikipedia.org/wiki/Inter-frame) are arranged.

The GOP is a collection of successive pictures within a coded video stream.

Each coded video stream consists of successive GOPs, from which the visible frames are generated.

Encountering a new GOP in a compressed video stream means that the decoder doesn't need any previous frames in order to decode the next ones, and allows fast seeking through the video.

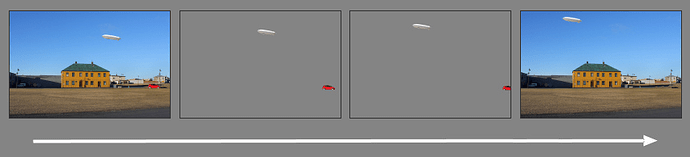


Fig: GOP with two I-frames and two B-Frames

**What is High FPS video ?**

High fps means anything >30fps is usually reserved for recording busy scene with a lot of motion, such as video games athletics or anything you want to show in slow motion.

24 fps is standard one.

30 fps is used in live TV broadcasts and excellent for live sports.

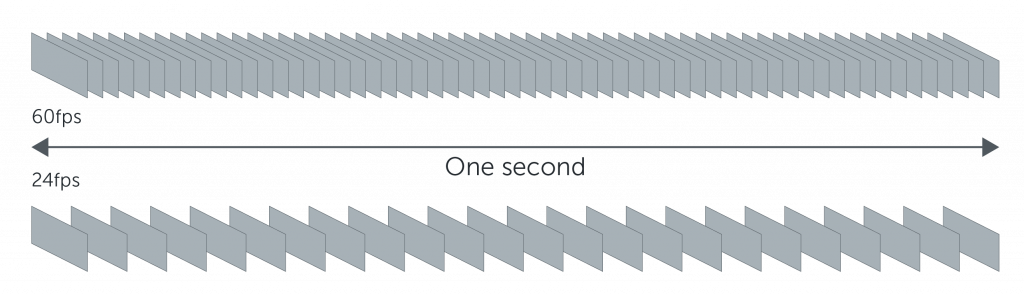


Fig:Framerates

**Source link:** <https://www.techsmith.com/blog/frame-rate-beginners-guide/>

======== Normal Video ===========

720p@30fps --> 1280x720 @ 30 fps

If Total video length = 1 min

1 min => 60 secs

30 fps

1 sec => 30 frames

1 frame => 1/30 => 33 msecs

60 secs ==> 1800 frames

Example: Decoder (H264 -> YUV)

1800 frames should be processed in 1 min

If we take video of 1 min with 720p(resolution) and 30fps(frame per sec),it will contain total of 1800 frames(60secs(1 min)x30fps) and one frame will be process in 1/30fpsx1000=33msec,so total 1800 frames will process in 1 min with frame difference is 33msec.

======== High FPS Video =============

720p@120 fps

Total video length = 1 min

1 min => 60 secs

120fps

1 sec = 120 frames

1 frame => 1/120 = 8.33 mSecs

60 secs => 7200 frames

720p@30fps speed play ==> 4 mins

7200 frames @ 30 fps ==> 4 mins

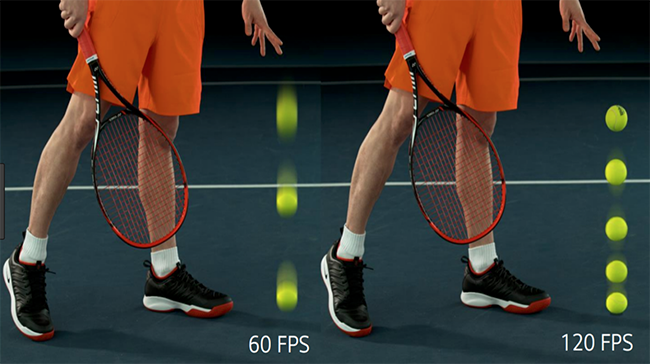


Fig: Difference between 60fps and 120fps

**Source link:** <https://www.techradar.com/news/hfr-explained-high-frame-rate-is-coming-to-tvs-heres-what-you-need-to-know>

**What is timelapse video ?**

In timelapse the frequency at which film frames are captured is much more spread out than the frequency used to view the sequence.

when played at normal speed,time appears to be moving faster and thus lapsing

======== Time Lapse Video ===========

720p@30fps --> 1280x720 @ 30 fps

day 1: Everyday morning 11AM => 1 frame

.

.

.

day 600 : 1 frame

600 frames@30fps

**Source Link:** <https://www.youtube.com/watch?v=8ARlYDhkCGY>

**Resolution?**

Resolution means the no.of pixels per unit of area.

**Width=720x16/9=1280(Hight x Aspect ratio)**

**720p = 1280 x 720** - is usually known as HD or "HD Ready" resolution

**1080p = 1920 x 1080** - is usually known as FHD or "Full HD" resolution

**1440p = 2560 x 1440** - is commonly known as QHD or Quad HD resolution, and it is typically seen on gaming monitors and on high-end smartphones.

1440p is four times the resolution of 720p HD or "HD ready."

To make things even more confusing, many premium smartphones feature a so-called 2960x1440 Quad HD+ resolution, which still fits into 1440p.

**4K or 2160p = 3840 x 2160** - is commonly known as 4K, UHD or Ultra HD resolution.

It is a huge display resolution, and it is found on premium TVs and computer monitors. 2160p is called 4K because the width is close to 4000 pixels. In other words, it offers four times the pixels of 1080p FHD or "Full HD."

**8K or 4320p = 7680 x 4320** - is known as 8K and it offers 16 times more pixels than the regular 1080p FHD or "Full HD" resolution.

For now, you see 8K only on expensive TVs from Samsung and LG.

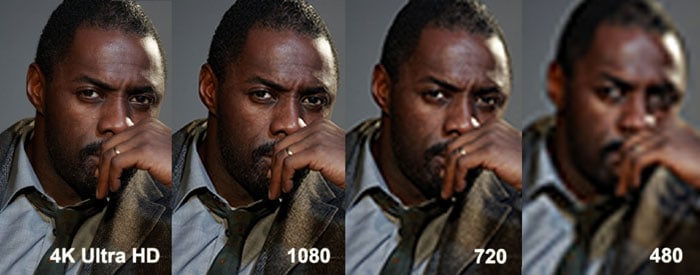


Fig: Resolutions

**Source link:** <https://i0.wp.com/feelproductions.com/wp-content/uploads/HD-vs-4k.jpg>



Fig: Resolutions-WidthxHeight

**Source link:** <https://www.solidtyper.com/blog/720p-vs-1080p-vs-4k-vs-8k/>

**Aspect ratio?**

The term aspect ratio was initially used in motion pictures, indicating how width of the picture was in relation to its height.

Movies were initially in **4:3** aspect ratio.

As display technology improved, TV and monitor manufacturers began to move toward widescreen displays as well.

Originally "widescreen" referred to anything wider than the typical 4:3 display, but it quickly came to mean a 16:10 ratio and later **16:9**.

Depending on the aspect ratio of your display, you can use only resolutions that are specific to its width and height.

**Some of the most common resolutions:**

4:3 aspect ratio resolutions: 640×480, 800×600, 960×720, 1024×768, 1280×960, 1400×1050, 1440×1080, 1600×1200, 1856×1392, 1920×1440, and 2048×1536.

16:10 aspect ratio resolutions: 1280×800, 1440×900, 1680×1050, 1920×1200, and 2560×1600.

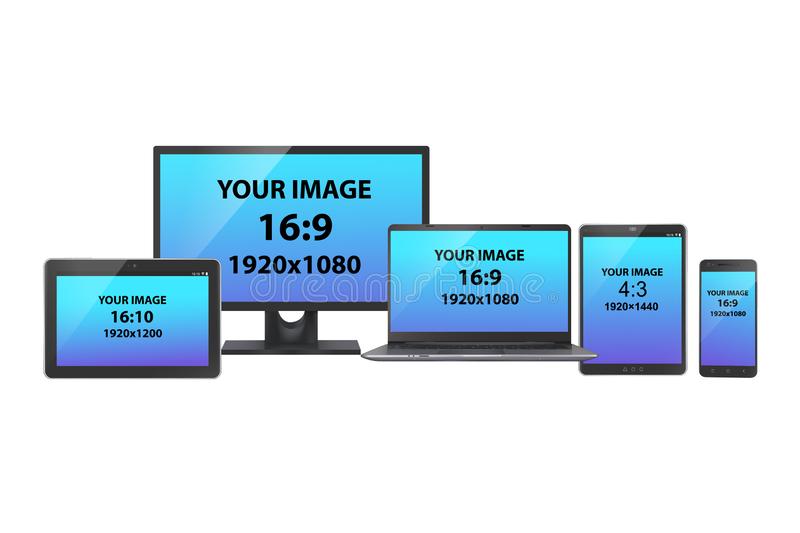
16:9 aspect ratio resolutions: 1024×576, 1152×648, 1280×720 (HD), 1366×768, 1600×900, 1920×1080 (FHD), 2560×1440 (QHD), 3840×2160 (4K), and 7680 x 4320 

Fig: Aspect ratios

**Source link:** <https://www.shutterstock.com/image-vector/set-electronic-devices-pc-monitor-tablets-1359413084>

**What is Video Down scaling?**

We can watch a high-resolution video on a smaller resolution screen.

For example, if you want to watch a video with a 4K resolution on a 720p resolution, that video is shown at 720p resolution, because that is all that your screen can offer.

if the video you want to watch has a higher resolution than that of your device resolution capacity, your device converts the video's resolution to one that fits the resolution of your device,

this is called downscaling.

**What is Video Up scaling?**

We can watch a low-resolution video on a high resolution device.

Upscaling means 720p video content you can see it in full screen mode on a 1080p device.

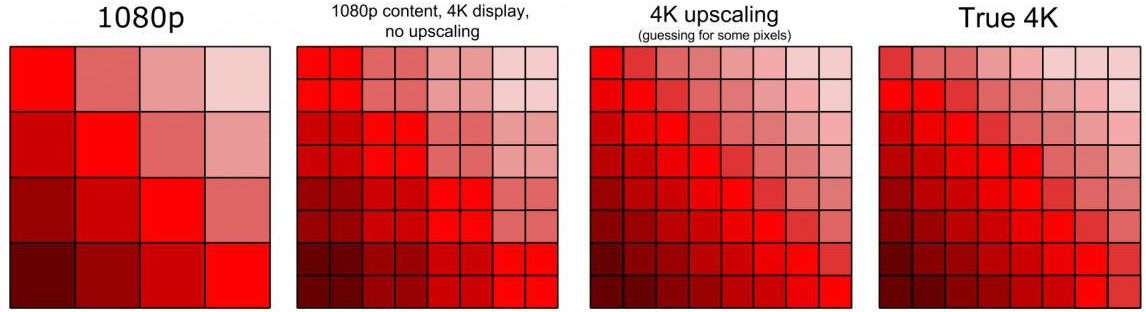


Fig: Upscaling

**Source link:** <https://www.extremetech.com/extreme/310029-how-to-upscale-video-content-to-4k-8k-and-beyond>

**Macro block?**

We break up or partition a frame into rectangular areas known as slices or tiles and each tile can be broken down into further into square units known as macro blocks or coding tree units.

Codec will process one macro block at a time,These macro blocks splitted into quarters and quartes again to process.

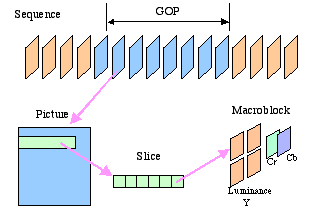


Fig: Macro block

**Source link:** <https://yong.epower.co.nz/Multimedia/MPEGVideo.html>

**What is AV sync (Audio Video Sync)? When AV sync issue can come?**

Audio-to-video synchronization (AV synchronization, also known as lip sync, or by the lack of it: lip-sync error, lip flap) refers to the relative timing of audio (sound) and video (image) parts during creation and post-production (mixing), transmission, reception and play-back processing.

**When it will occur:**

If video is slower than the audio/audio is slower than video,time difference between audio and video.

**No sound in a video issue?**

No sound in a video issue means video will play without audio.

Means in the container format may be audio file is missed.

**Hardware Video decoder/encoder?**

Hardware encoders/decoders are dedicated processors that use a designed algorithm to encode/decode.

**Sofware Video Decoder/encoder?**

Software encoder/decoders are programs that run on laptop/desktop computer

Ex:

FFmpeg

**ffprobe?**

ffprobe gathers information from multimedia streams and prints it in human- and machine-readable fashion.

For example it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it.

**ffmpeg?**

ffmpeg is one such software platform especially for multimedia files. With the help of its expanded libraries we can convert, edit, repair, format any video format. It is basically a command line tool which uses commands to work on audio and video formats.

**Instalation of ffmpeg:**

sudo apt install ffmpeg

**Ffmpeg linux commands:**

ffmpeg -i video.mp4 **🡪** It will show the details of video file

#### Converting video files to different formats:

ffmpeg -i video.mp4 out.h264

ffmpeg -i video.mp4 out.yuv

ffmpeg -i out.h264 sample.mp4

ffmpeg -i out.h264 sample.yuv

ffmpeg -i video.mp4 out.h265

ffmpeg -i out.yuv sample.mp4

ffmpeg -i out.yuv out.h264

ffmpeg -i out.yuv out.h265

ffmpeg -i out.h265 out.yuv

ffmpeg -i out.h265 sample.mp4

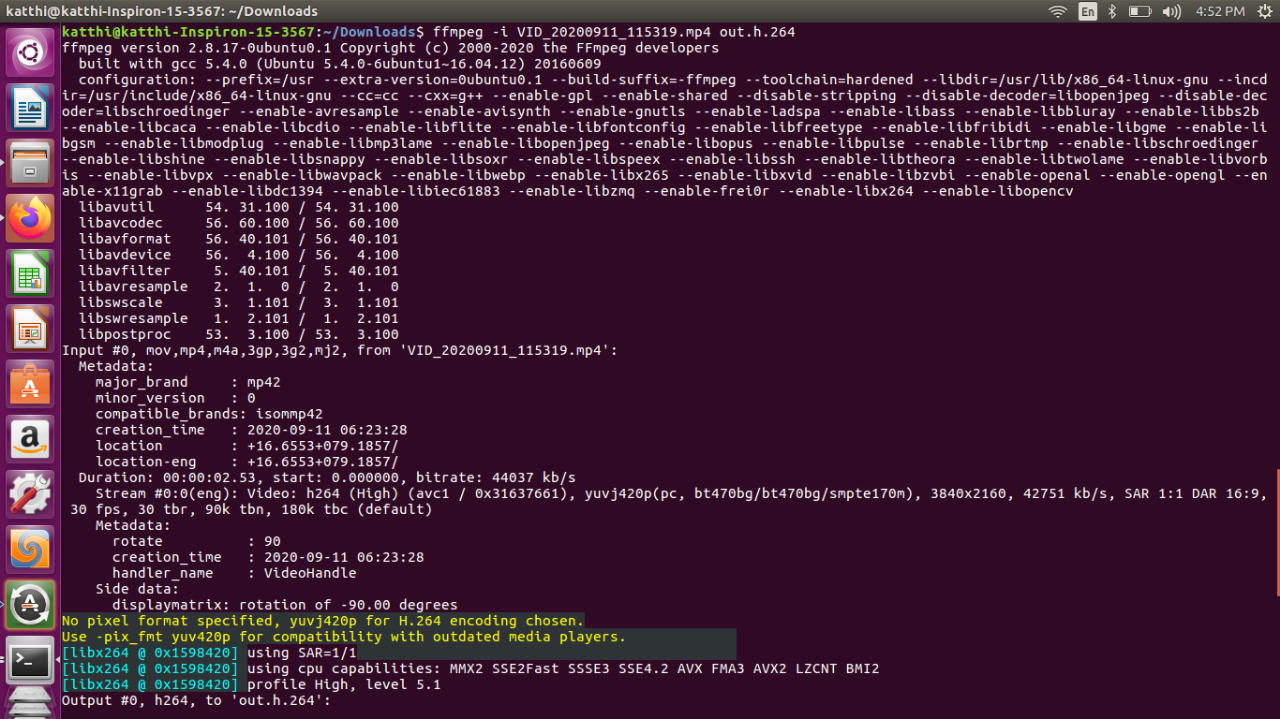


Fig: Conversion of Mp4 video to H264 with ffmpeg

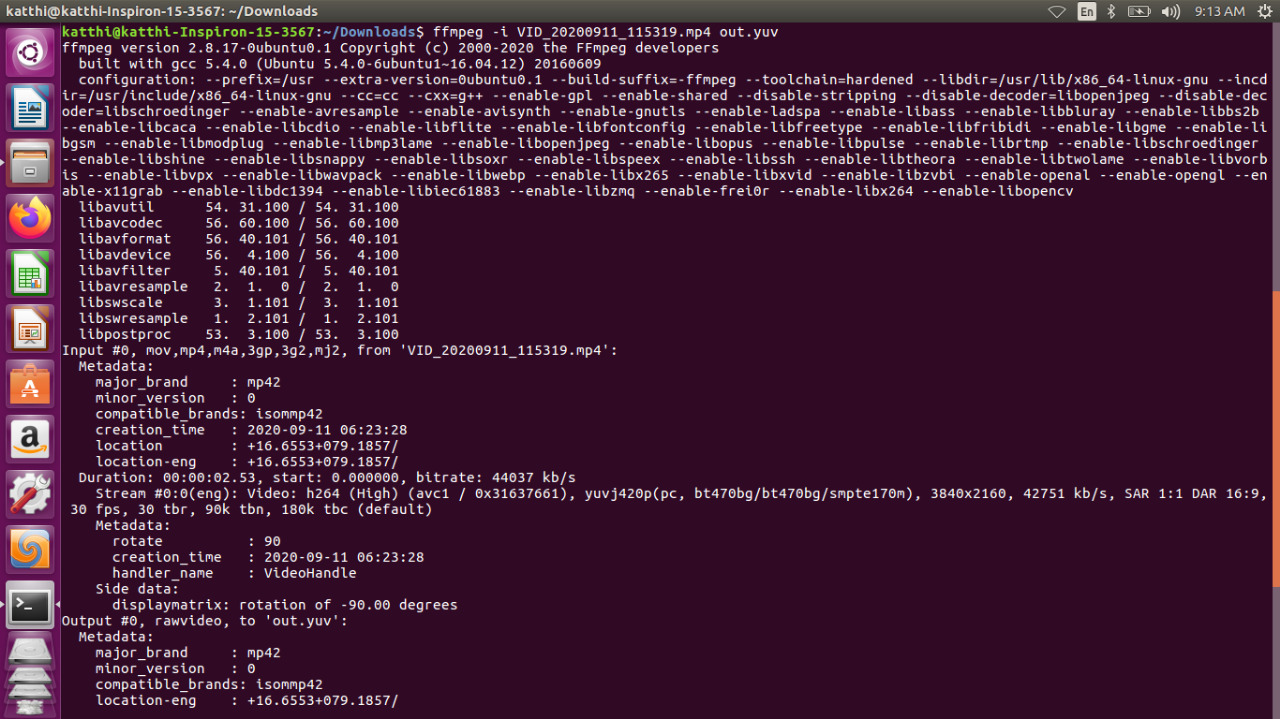


Fig: Conversion of Mp4 video to YUV with ffmpeg

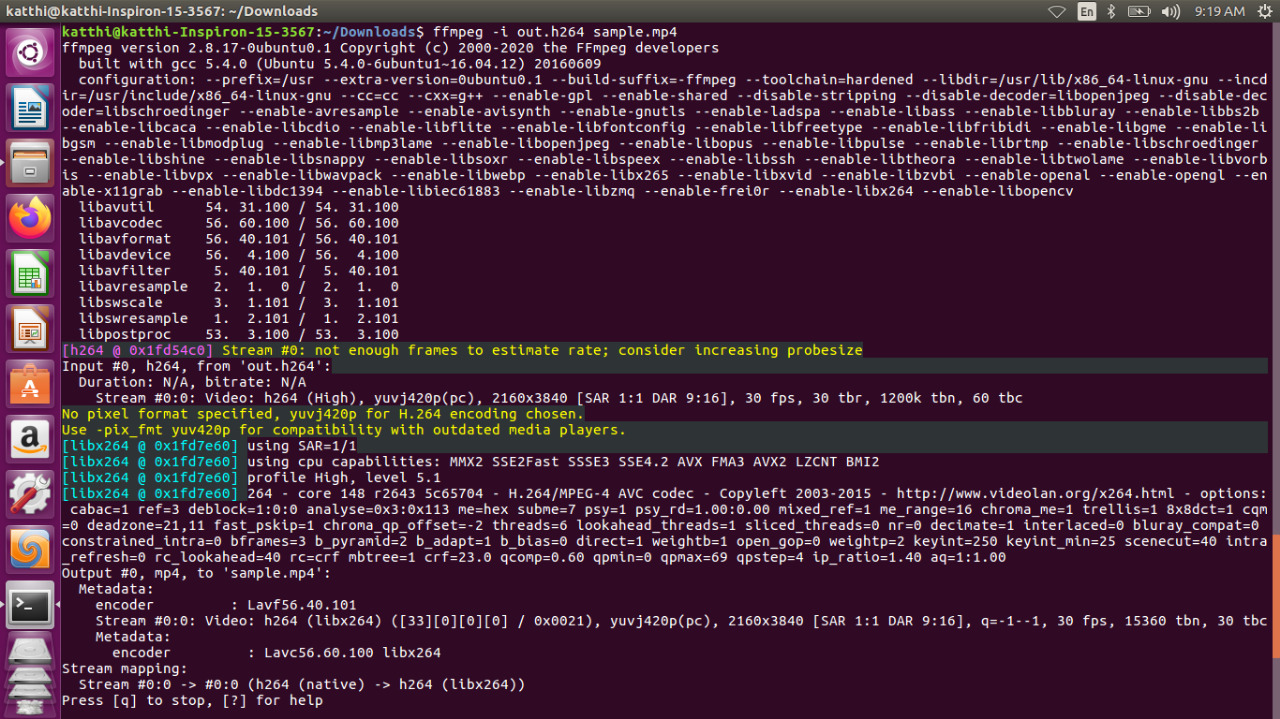
****

Fig: Conversion of H264 video to MP4 with ffmpeg

**Source Link:** <https://ostechnix.com/20-ffmpeg-commands-beginners/>

**What is V4L2?**

**Video4Linux** (**V4L** for short) is a collection of [device drivers](https://en.wikipedia.org/wiki/Device_driver) and an [API](https://en.wikipedia.org/wiki/Application_programming_interface) for supporting real time [video capture](https://en.wikipedia.org/wiki/Video_capture) on [Linux](https://en.wikipedia.org/wiki/Linux) systems. It supports many [USB](https://en.wikipedia.org/wiki/USB) [webcams](https://en.wikipedia.org/wiki/Webcam), [TV tuners](https://en.wikipedia.org/wiki/TV_tuner_card), and related devices, standardizing their output, so programmers can easily add video support to their applications.

V4l2 stands for Video for Linux 2,second version of the V4L API and framework.

Since we don’t have any common framework, we need to do the setup of device instance and connecting to sub-devices themselves for all the drivers. And there is also lot of common code that could never be re-factored due to lack of framework.

To avoid this complexity, introduced a framework with common functionalities for all the drivers which is known as V4L2 framework.

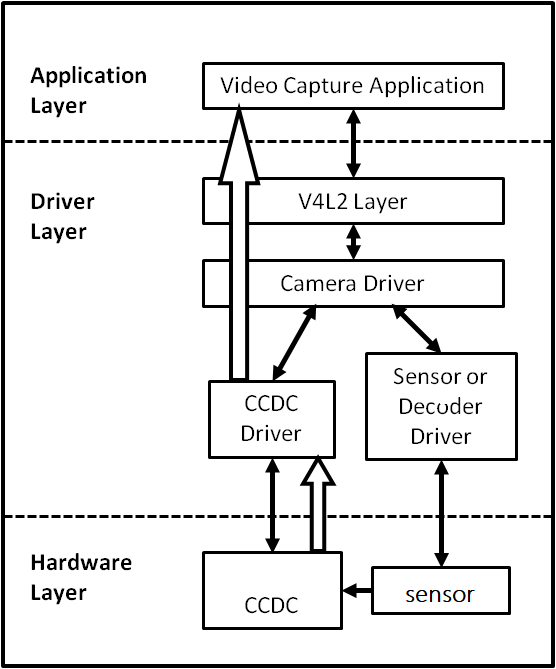
****

Fig:V4L2

**Source link:** <https://processors.wiki.ti.com/images/3/36/AM437x_capture_overview.png>

**Structure of a V4L driver:**

All drivers have the following structure:

1. A struct for each device instance containing the device state.
2. A way of initializing and commanding sub-devices (if any).
3. Creating V4L2 device nodes (/dev/videoX, /dev/vbiX and /dev/radioX) and keeping track of device-node specific data.
4. Filehandle-specific structs containing per-filehandle data;
5. video buffer handling.

V4L2 drivers are implemented as kernel modules, loaded manually by the system administrator or automatically when a device is first discovered. The driver modules plug into the “videodev” kernel module. It provides helper functions and a common application interface

Many drivers support “video\_nr”, “radio\_nr” or “vbi\_nr” module options to select specific video/radio/vbi node numbers

Devices can support several functions. For example video capturing, VBI capturing and radio support.

The V4L2 API creates different nodes for each of these functions.

The V4L2 API was designed with the idea that one device node could support all functions.

**Common implementation of a v4l2 application:**

To open and close V4L2 devices applications use the [open()](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/func-open.html#func-open) and [close()](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/func-close.html#func-close) function, respectively. Devices are programmed using the [ioctl()](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/func-ioctl.html" \l "func-ioctl) function.

[**VIDIOC\_QUERYCAP**](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/vidioc-querycap.html#vidioc-querycap) **ioctl:**

It checks if the kernel device is compatible with this specification, and to query the [functions](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/devices.html#devices)(capabilities) and [I/O methods](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/io.html#io) supported by the device.

All V4L2 drivers must support [ioctl VIDIOC\_QUERYCAP](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/vidioc-querycap.html" \l "vidioc-querycap). Applications should always call this ioctl after opening the device.

Other features can be queried by calling the respective ioctl.

int ioctl(int fd*,* [VIDIOC\_QUERYCAP](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-querycap.html#c.VIDIOC_QUERYCAP), struct [v4l2\_capability](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-querycap.html#c.v4l2_capability) \*argp)

**fd**

File descriptor returned by [open()](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/func-open.html#func-open).

All V4L2 devices support the VIDIOC\_QUERYCAP ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities.

The ioctl takes a pointer to a **struct**[**v4l2\_capability**](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-querycap.html#c.v4l2_capability) which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an EINVAL error code.

**VIDIOC\_S\_FMT ioctl:**

It will set the frame format like (RGB,YUV,MJPEG….)

int ioctl(int fd, [VIDIOC\_S\_FMT](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-g-fmt.html#c.VIDIOC_S_FMT), struct [v4l2\_format](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-g-fmt.html#c.v4l2_format) \*argp)

**fd**

File descriptor returned by [open()](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/func-open.html#func-open).

**argp**

Pointer to struct [v4l2\_format](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-g-fmt.html#c.v4l2_format).

[**VIDIOC\_ENUM\_FMT**](https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/vidioc-enum-fmt.html#vidioc-enum-fmt) **ioctl:**

To enumerate all image formats supported by video capture, overlay or output devices is available.

int ioctl(int fd*,* [VIDIOC\_ENUM\_FMT](https://www.kernel.org/doc/html/v4.14/media/uapi/v4l/vidioc-enum-fmt.html#c.VIDIOC_ENUM_FMT), struct [v4l2\_fmtdesc](https://www.kernel.org/doc/html/v4.14/media/uapi/v4l/vidioc-enum-fmt.html#c.v4l2_fmtdesc) \*argp)

**fd**

File descriptor returned by [open()](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/func-open.html#func-open).

**argp**

Pointer to struct [v4l2\_fmtdesc](https://www.kernel.org/doc/html/v4.14/media/uapi/v4l/vidioc-enum-fmt.html#c.v4l2_fmtdesc).

To enumerate image formats applications initialize the type and index field of struct [v4l2\_fmtdesc](https://www.kernel.org/doc/html/v4.14/media/uapi/v4l/vidioc-enum-fmt.html#c.v4l2_fmtdesc) and call the [ioctl VIDIOC\_ENUM\_FMT](https://www.kernel.org/doc/html/v4.14/media/uapi/v4l/vidioc-enum-fmt.html" \l "vidioc-enum-fmt) with a pointer to this structure.

**VIDIOC\_REQBUFS ioctl:**

It is used to inform the device about your buffers: how are you going to allocate them and how many are there and this will allow the device to write buffer data correctly(mmap is used to map the buffers)

int ioctl(int fd, [VIDIOC\_REQBUFS](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-reqbufs.html#c.VIDIOC_REQBUFS), struct [v4l2\_requestbuffers](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-reqbufs.html#c.v4l2_requestbuffers) \*argp)

**fd**

File descriptor returned by [open()](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/func-open.html#func-open).

**argp**

Pointer to struct [v4l2\_requestbuffers](https://www.kernel.org/doc/html/v4.15/media/uapi/v4l/vidioc-reqbufs.html#c.v4l2_requestbuffers).

**VIDIOC\_QUERYBUF ioctl :**

It can be used to give the amount of memory it needs means how many bytes it needed for frame format and frame dimensions ,allocate it and the status of a buffer at any time after buffers have been allocated with the [ioctl VIDIOC\_REQBUFS](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-reqbufs.html#vidioc-reqbufs).

int ioctl(int fd, [VIDIOC\_QUERYBUF](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-querybuf.html#c.VIDIOC_QUERYBUF), struct [v4l2\_buffer](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/buffer.html#c.v4l2_buffer) \*argp)

[**VIDIOC\_STREAMON**](https://hverkuil.home.xs4all.nl/codec-api/uapi/v4l/vidioc-streamon.html#vidioc-streamon) **ioctl :**

This ioctl is used to start the applications andit will activate the streaming capability of the device.

int ioctl(int fd, [VIDIOC\_STREAMON](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-streamon.html#c.VIDIOC_STREAMON), const int \*argp)

**fd**

File descriptor returned by [open()](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/func-open.html#func-open).

**VIDIOC\_QBUF** **ioctl:**

Queue the buffer to handle the buffer over to the device ,Applications call this ioctl to en-queue an empty (capturing) or filled (output) buffer in the driver's incoming queue.

The semantics depend on the selected I/O method.

int ioctl(int fd, [VIDIOC\_QBUF](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/vidioc-qbuf.html#c.VIDIOC_QBUF), struct [v4l2\_buffer](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/buffer.html#c.v4l2_buffer) \*argp)

**fd**

File descriptor returned by [open()](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/func-open.html#func-open).

**argp**

Pointer to struct [v4l2\_buffer](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/buffer.html#c.v4l2_buffer).

[**VIDIOC\_DQBUF**](https://hverkuil.home.xs4all.nl/codec-api/uapi/v4l/vidioc-qbuf.html#vidioc-qbuf)**ioctl :**

To de-queue a buffer applications use the this ioctl to retrieving the buffer from the outgoing queue.

Buffers remain locked until de-queued, until the [VIDIOC\_STREAMOFF](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/vidioc-streamon.html#vidioc-streamon) [ioctl or VIDIOC\_REQBUFS](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/vidioc-reqbufs.html" \l "vidioc-reqbufs) ioctl is called, or until the device is closed

int ioctl(int fd, [VIDIOC\_DQBUF](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/vidioc-qbuf.html#c.VIDIOC_DQBUF), struct [v4l2\_buffer](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/buffer.html#c.v4l2_buffer) \*argp)

**fd**

File descriptor returned by [open()](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/func-open.html#func-open).

**argp**

Pointer to struct [v4l2\_buffer](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/buffer.html#c.v4l2_buffer)

[**VIDIOC\_STREAMOFF**](https://hverkuil.home.xs4all.nl/codec-api/uapi/v4l/vidioc-streamon.html#vidioc-streamon) **ioctl :**

It will stop the applications by turn of the streaming mode, after buffers will free and it calls the device close function.

int ioctl(int fd, [VIDIOC\_STREAMOFF](https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/vidioc-streamon.html#c.VIDIOC_STREAMOFF), const int \*argp)

**fd**

File descriptor returned by [open()](https://01.org/linuxgraphics/gfx-docs/drm/media/uapi/v4l/func-open.html#func-open).

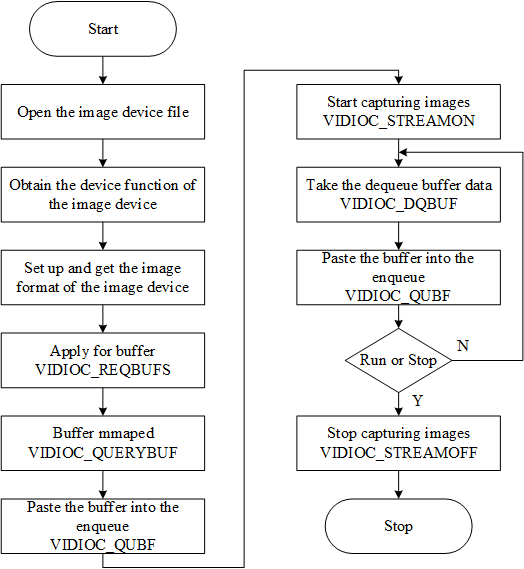


Fig: Flow chart of application implementation

**Source link:** <https://www.researchgate.net/figure/V4L2-image-capture-operation-flowchart_fig1_324172011>

**What is Transcoding?**

[Transcoding](https://bitmovin.com/cloud-transcoding-adaptive-video/) refers to the process of compressing video files as much as possible at minimal quality loss to represent (and transfer) information by using less data.

Essentially, video transcoding is the conversion of a video file from one format to a better-compressed version to ensure consumers can stream content without buffering and at the highest possible qualities.(Both Encoding and Decoding)

Ex:

Video upscale: 640x480 is converting into 1280x720

640x480 🡺 1280x720

**Input:**input.mp4(640x480)

input.h264 🡪 input.yuv 480p 640x480(Decoder)

input\_up.yuv 720p 1280x720 🡪 input\_up.h264 720p(encoder)

V4L devices or other video-related devices (e.g. DRM).Buffers (planes) are allocated by a driver on behalf of an application.

**What is O/p Plane?**

O/p plane means input buffer to any conversion(Either encoder/decoder)

**What is Capture Plane?**

Capture plane means output buffer to any conversion(Either encoder/decoder)

Ex:

O/p plane [Decoder] Capture Plane

dev/video0

O/p plane [Encoder] Capture Plane

dev/video1

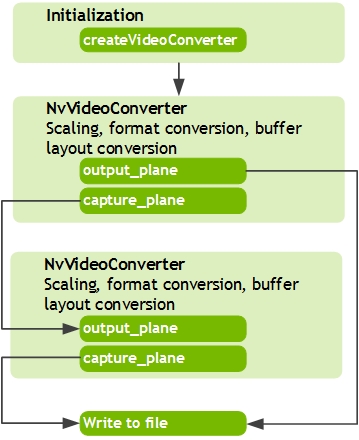


Fig: Output and Capture planes

**Source link:** <https://docs.nvidia.com/jetson/archives/l4t-multimedia-archived/l4t-multimedia-271/ee_video_decode_cuda.html>

**What is Qbuf?**

Queuing the buffers(input buffer), send empty buffers(output buffer) in input queue

**What is Dqbuf?**

Retreiving the buffers, receive acknowledgment of buffer in output queue

Ex:

Decoder is converting H264 -> YUV

In case of a decoder conversion:

ibuf [Decoder] obuf

**O/p Plane** **Capture plane**

qbuf\_ibuf (Queuing the buffers) ---- qbuf\_obuf (send empty buffers)

dqbuf\_ibuf (receive ack of buffer) ---- dqbuf\_obuf (Retreiving the buffers)

**How to run test app and collect the logs for test app commands?**

run.sh 🡪script file contains the below content

echo ""

sudo dmesg –C

sudo ch –c ‘echo “3” >> /sys/class/video4linux/video0/dev\_debug’

sudo ch –c ‘echo “3” >> /sys/module/videobuf2\_v4l2/parameters/debug’

sudo ch –c ‘echo “3” >> /sys/module/videobuf2\_core/parameters/debug’

./yavta /dev/video0 –capture=10

echo ""

echo ""

dmesg

Change the permissions of script file using the chmod command

chmod 777 run.sh

Run the script file and take the logs into log1.txt file

./run.sh | tee log1.txt

**Full Log:**

Below is the log for test application log1.txt file

**/\*** open the video device 0 and query the device capabilities card as : Integrated\_Webcam\_HD and bus as usb-0000:00:14.0-5 \*/

shashi:video\_open:VIDIOC\_QERYCAP

Device /dev/video0 opened: Integrated\_Webcam\_HD: Integrate (usb-0000:00:14.0-5).

/\* video device 0 formats as width=1280, height=720, pixelformat=MJPG \*/

shashi:video\_get\_format:VIDIOC\_G\_FMT

Video format: MJPG (47504a4d) 1280x720

/\* it allocated 8 buffers using malloc \*/

shashi:video\_alloc\_buffers:VIDIOC\_REQBUFS

8 buffers requested.

/\*8 buffers are mapping into process address space using mmap length as 1834789 and different offset address\*/

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 0

Buffer 0 mapped at address 0x7f267834a000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 1847296

Buffer 1 mapped at address 0x7f2677d76000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 3694592

Buffer 2 mapped at address 0x7f2677bb3000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 5541888

Buffer 3 mapped at address 0x7f26779f0000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 7389184

Buffer 4 mapped at address 0x7f267782d000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 9236480

Buffer 5 mapped at address 0x7f267766a000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 11083776

Buffer 6 mapped at address 0x7f26774a7000.

shashi:video\_alloc\_buffers:VIDIOC\_QUERYBUF

length: 1843789 offset: 12931072

Buffer 7 mapped at address 0x7f26772e4000.

/\* buffers are placing into input queue for capturing \*/

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_queue\_buffer:VIDIOC\_QBUF

/\* video device is capturing 10 frames by stream on \*/

shashi:video\_enable:enable ? VIDIOC\_STREAMON : VIDIOC\_STREAMOFF

/\* retreiving the buffer from capture plane \*/

shashi:video\_do\_capture:VIDIOC\_DQBUF

0 (0) [-] 0 54824 bytes 5022.898708 1600148240.715216

/\* sending an empty buffer to capture plane \*/

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

1 (1) [-] 2 53768 bytes 5024.495000 1600148240.915236

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

2 (2) [-] 3 55944 bytes 5024.695035 1600148241.115293

shashi:video\_do\_capture:VIDIOC\_DQBUF

3 (3) [-] 4 56136 bytes 5024.895070 1600148241.315301

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

4 (4) [-] 5 62432 bytes 5025.095106 1600148241.515341

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

5 (5) [-] 6 62152 bytes 5025.295144 1600148241.715378

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

6 (6) [-] 7 64360 bytes 5025.495178 1600148241.915410

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

7 (7) [-] 8 64672 bytes 5025.695232 1600148242.115478

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

8 (0) [-] 9 72200 bytes 5025.895251 1600148242.315512

shashi:video\_queue\_buffer:VIDIOC\_QBUF

shashi:video\_do\_capture:VIDIOC\_DQBUF

9 (1) [-] 10 72296 bytes 5026.095288 1600148242.515516

shashi:video\_queue\_buffer:VIDIOC\_QBUF

/\* video device is disabled by stream off \*/

shashi:video\_enable:enable ? VIDIOC\_STREAMON : VIDIOC\_STREAMOFF

Captured 9 frames in 1.800756 seconds (4.997901 fps, 343624.566571 B/s).

/\* allocated buffers released after stream off \*/

shashi:video\_free\_buffers:VIDIOC\_REQBUFS

8 buffers released.

/\* video device 0 is closed \*/

shashi:video\_close

[28928.813784] usb 1-5: reset high-speed USB device number 2 using xhci\_hcd

[29888.519390] video0: VIDIOC\_QUERYCAP: driver=uvcvideo, card=Integrated\_Webcam\_HD: Integrate, bus=usb-0000:00:14.0-5, version=0x00040f12, capabilities=0x84200001, device\_caps=0x04200001

[29888.519484] video0: VIDIOC\_G\_FMT: type=vid-cap, width=1280, height=720, pixelformat=MJPG, field=none, bytesperline=0, sizeimage=1843789, colorspace=0, flags=0x0, ycbcr\_enc=0, quantization=0, xfer\_func=0

[29888.524120] video0: VIDIOC\_REQBUFS: count=8, type=vid-cap, memory=mmap

[29888.524151] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=0, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x0, length=1843789

[29888.524157] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524227] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=1, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x1c3000, length=1843789

[29888.524233] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524286] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=2, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x386000, length=1843789

[29888.524292] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524339] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=3, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x549000, length=1843789

[29888.524344] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524389] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=4, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x70c000, length=1843789

[29888.524394] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524448] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=5, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0x8cf000, length=1843789

[29888.524453] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524511] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=6, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0xa92000, length=1843789

[29888.524516] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29888.524571] video0: VIDIOC\_QUERYBUF: 00:00:00.00000000 index=7, type=vid-cap, flags=0x00012000, field=any, sequence=0, memory=mmap, bytesused=0, offset/userptr=0xc55000, length=1843789

[29888.524577] timecode=00:00:00 type=0, flags=0x00000000, frames=0, userbits=0x00000000

[29889.137787] video0: VIDIOC\_STREAMON: type=vid-cap

[29889.818249] video0: VIDIOC\_STREAMOFF: type=vid-cap

[29889.820902] video0: VIDIOC\_REQBUFS: count=0, type=vid-cap, memory=mmap

**References**

Link: https://www.synopi.com/interlaced-and-progressive-video/

Link: https://www.youtube.com/watch?v=8ARlYDhkCGY

Link: <https://ostechnix.com/20-ffmpeg-commands-beginners/>

Link:https://www.cnblogs.com/kevin-heyongyuan/articles/11070935.html

Link: <http://gentlelogic.blogspot.com/2011/11/exploring-h264-part-2-h264-bitstream.html>

Link: <http://www.bnoack.com/index.html?http&&&www.bnoack.com/video/GOP.html>

Link: https://www.solidtyper.com/blog/720p-vs-1080p-vs-4k-vs-8k/

Link: <https://www.techsmith.com/blog/frame-rate-beginners-guide/>

Link: https://www.digitalcitizen.life/what-screen-resolution-or-aspect-ratio-what-do-720p-1080i-1080p-mean

Link: <https://www.kernel.org/doc/html/v4.16/media/uapi/v4l/v4l2.html#v4l2spec>

Link: <https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/videodev2.h#L942>

Link: <https://docs.nvidia.com/jetson/archives/l4t-multimedia-archived/l4t-multimedia-271/ee_video_decode_cuda.html>

Link: <https://www.youtube.com/watch?v=gxefuXizO04>

Link: https://www.youtube.com/watch?v=H\_o5h5SK\_70

Link: https://en.wikipedia.org/wiki/Comparison\_of\_video\_container\_formats

Link: https://www.youtube.com/watch?v=J9RemuYxA4s

Link: <https://www.youtube.com/watch?v=XvoW-bwIeyY>

Link: <https://www.muvi.com/blogs/best-video-codec-for-streaming.html>

Link: <https://msu.edu/~schanerr/Color_Figures.html>

Link: <https://i0.wp.com/feelproductions.com/wp-content/uploads/HD-vs-4k.jpg>

Link: <https://www.tes.com/lessons/_kjvlPLmCriLSw/minecraft-selfies>

Link: <https://nofilmschool.com/2015/12/check-out-15-minute-history-lesson-frame-rates>

Link: <https://commons.wikimedia.org/wiki/File:YUV_UV_plane.jpg>

Link: <http://www.pitivi.org/manual/codecscontainers.html>

Link: <https://www.shutterstock.com/image-vector/set-electronic-devices-pc-monitor-tablets-1359413084>

Link: <https://medium.com/@sddkal/chroma-subsampling-in-numpy-47bf2bb5af83>

Link: <https://streamshark.io/blog/understanding-codecs-and-formats/>

Link: <https://www.techradar.com/news/hfr-explained-high-frame-rate-is-coming-to-tvs-heres-what-you-need-to-know>

Link: <https://www.extremetech.com/extreme/310029-how-to-upscale-video-content-to-4k-8k-and-beyond>

Link: <https://yong.epower.co.nz/Multimedia/MPEGVideo.html>

Link: <https://www.researchgate.net/figure/V4L2-image-capture-operation-flowchart_fig1_324172011>

Link: <https://processors.wiki.ti.com/images/3/36/AM437x_capture_overview.png>