


video Understanding components of video tracks

 riptutorial.com/video/example/20669/understanding-components-of-video-tracks

Example#

This example will explore how to see the layout of a video track and how to extract the individual pictures within it.

The sample content used here is Tears of Steel, by Blender Foundation. Specifically, we will use the download titled "HD 720p (~365MB, mov, 2.0)". This is a single file that ends with the extension "mov" and will play in just about any modern media player.

We will use the mp4info and mp4dump tools from the Bento4 suite for track layout and structure analysis and FFmpeg for extracting the individual pictures that make up the video track.

The sample movie uses the "QuickTime" (MOV) packaging format, which is based on the ISO Base Media File Format - an international standard that underlies all packaging formats in the MP4 file format family. This makes it highly compatible with the majority of available tools and enables easy analysis.

Let's first examine the overall structure of the file. All media files based on ISO Base Media File Format are structured as a hierarchy of boxes - a mini-filesystem of sorts. Use the mp4dump utility to extract the box structure, by executing the following command:

```
mp4dump tears_of_steel_720p.mov
```

The output will be similar to the following:

```
[ftyp] size=8+12
  major_brand = qt
  minor_version = 200
  compatible_brand = qt
[wide] size=8+0
[mdat] size=8+371579623
[moov] size=8+598972
  [mvhd] size=12+96
    timescale = 1000
    duration = 734167
    duration(ms) = 734167
  [trak] size=8+244250
    [tkhd] size=12+80, flags=f
      enabled = 1
      id = 1
      duration = 734167
      width = 1280.000000
      height = 534.000000
...
```

This represents the internal structure of the file. For example, you see here a *moov* box that has an 8-byte header and 598972 bytes of contents. This box is a container for various metadata boxes that describe the contents of the file. For more information about the meaning of the various boxes and their properties, see [ISO/IEC 14496-12](#).

The actual media samples themselves - the compressed pictures and audio waveforms - are stored in the *mdat* box, the contents of which are opaque to the mp4dump utility.

To exclude irrelevant data and simplify the analysis workflow - this example focuses on the video track - we now remove the audio track from our sample movie. Execute the following command:

```
ffmpeg -i tears_of_steel_720p.mov -an -vcodec copy video_track.mov
```

Note that the above step will also remove various custom extension elements from the input video, packaging the essence of the visual content into a new container file and discarding anything else. If you do this in a production scenario, ensure that you truly are free to discard all other elements in the input file!

Encoded video tracks are a sequence of pictures. With the H.264 codec used here - and all other commonly used modern codecs - the pictures can be of various different types:

- I-pictures- these are independent pictures, decodable solely using the data contained within the picture.
- P-pictures- these take another picture as a baseline and apply a transform to that image (e.g. "move these particular pixels to the right by 5 pixels").
- B-pictures- similar to P-frames, but bidirectional - they can also reference pictures from the future and define transforms such as "these particular pixels that will be fully visible in 5 frames are now 10% visible".

The exact combination of picture types may be freely chosen by the encoding workflow, creating many optimization opportunities, although certain use cases may constrain the available flexibility by e.g. requiring an I-frame to be present at exactly 2 second intervals.

Execute the following command to see the picture structure of the video track:

```
mp4info --show-layout video_track.mov
```

In addition to presenting a human-readable form of the overall file metadata, you will see a detailed printout of the video track's picture layout.

```

...
00000959 [V] (1) size= 7615, offset=15483377, dts=491008 (39958 ms)
00000960 [V] (1)* size=104133, offset=15490992, dts=491520 (40000 ms)
00000961 [V] (1) size= 16168, offset=15595125, dts=492032 (40042 ms)
00000962 [V] (1) size= 4029, offset=15611293, dts=492544 (40083 ms)
00000963 [V] (1) size= 24615, offset=15615322, dts=493056 (40125 ms)
00000964 [V] (1) size= 4674, offset=15639937, dts=493568 (40167 ms)
00000965 [V] (1) size= 18451, offset=15644611, dts=494080 (40208 ms)
00000966 [V] (1) size= 95800, offset=15663062, dts=494592 (40250 ms)
00000967 [V] (1) size= 30271, offset=15758862, dts=495104 (40292 ms)
00000968 [V] (1) size= 10997, offset=15789133, dts=495616 (40333 ms)
00000969 [V] (1) size= 28458, offset=15800130, dts=496128 (40375 ms)
00000970 [V] (1) size= 9593, offset=15828588, dts=496640 (40417 ms)
00000971 [V] (1) size= 24548, offset=15838181, dts=497152 (40458 ms)
00000972 [V] (1) size= 6853, offset=15862729, dts=497664 (40500 ms)
00000973 [V] (1) size= 27698, offset=15869582, dts=498176 (40542 ms)
00000974 [V] (1) size= 7565, offset=15897280, dts=498688 (40583 ms)
00000975 [V] (1) size= 24682, offset=15904845, dts=499200 (40625 ms)
00000976 [V] (1) size= 5535, offset=15929527, dts=499712 (40667 ms)
00000977 [V] (1) size= 38360, offset=15935062, dts=500224 (40708 ms)
00000978 [V] (1)* size= 82466, offset=15973422, dts=500736 (40750 ms)
00000979 [V] (1) size= 13388, offset=16055888, dts=501248 (40792 ms)
00000980 [V] (1) size= 2315, offset=16069276, dts=501760 (40833 ms)
00000981 [V] (1) size= 21983, offset=16071591, dts=502272 (40875 ms)
00000982 [V] (1) size= 3384, offset=16093574, dts=502784 (40917 ms)
00000983 [V] (1) size= 22225, offset=16096958, dts=503296 (40958 ms)
...

```

Each row in this printout is a picture contained in the video track. Those marked with an asterisk as (1)* are I-pictures. You can see how they are the largest in size, with the others enabling greater compression by referencing existing pictures and only describing the differences.

The listing also contains the offset of the picture data in the video file and the decoding timestamp of the picture, enabling further correlation and analysis. Note that the decoding order/timing of pictures is not necessarily the same as the presentation order/timing! If B-pictures exist in the video, they can only be decoded *after* any pictures they reference, even if they are presented *before* the referenced pictures!

Having gained some insight into the structure of the video track, execute the following command to extract the 30 pictures starting at the 40 second mark as PNG files:

```
ffmpeg -i video_track.mov -ss 00:00:40 -vframes 30 picture%04d.png
```

The extracted pictures will be fully decoded, as they would appear in a video player - it is not possible (without extremely specialized tooling) to get a visual representation of the raw data in P-frames or B-frames.

Observe how the 7th generated picture is a complete scene change in the video. You can easily correlate this with the mp4info output above - the 7th picture starting from the 40 second mark (number 00000966) is far larger in size than the ones near it. Scene

changes are difficult to encode, as they refresh the entire picture and contain much new data. If the encoder is not given enough leniency to optimize for scene changes (i.e. not allowed to generate a large picture), the visual output will be low quality or "blocky" until the next I-picture. By examining the allocation of bandwidth (bytes) to various pictures, you can gain insight into such visual artifacts that may suddenly appear in a video.