A BEGINNER'S EXPERIMENT IN DECOMPRESSING LOSSLESS JPEG2000

by Winnie Schwaid-Lindner

Tisch School of the Arts, Moving Image Archiving and Preservation '18

wschlin@gmail.com

www.wn.ie

TABLE OF CONTENTS

INTRODUCTION

Lossless JPEG2000 is a compression/decompression algorithm (also known as “codec”) that is almost ubiquitous in the digital video world. It has been used by anyone from the Library of Congress to private users, who have selected it based on the fact that the compression within the file is “lossless”.

Video compression as a whole is essentially achieved by taking shortcuts through the data that is stored - videos consist of bits and bytes, and a codec bypasses the bits and bytes that humans aren’t likely to notice missing. This makes the file smaller without visibly compromising the quality of the video. However, codecs that are “lossy” (as opposed to “lossless”) permanently lose the data that has been cut out, and it is impossible to take the compressed file and get all of the original data back. In comparison, “lossless”, or “mathematically lossless” codecs are reversible. The algorithm that is applied can be turned back around, so that the data that was originally cut out can be recovered.

However, just because a codec is mathematically lossless does not mean that the reversal process is painless or easily accomplished. Although experts on video compression will likely know how to start the reversal process, the prevalence of lossless JPEG2000 as a codec suggests that many of these users are not at all experts, and may be hobbyists or beginners instead.

As the writer of this paper falls more into the latter category than the former, this project is the attempt (and moderate success) of lossless JPEG2000 reversal. It will discuss the methodology, thought process, difficulties, and results of the decompression achieved, in addition to outlining the steps that a beginner may take to recreate my results, and similarly reverse their own video file.

METHOD OVERVIEW AND PREAMBLE

Software

As a beginner, I started this project with a bare minimum of command line interface (CLI) experience. The application for the CLI on the computer I have used for this project is called "Terminal", as I am using a Macbook Pro running macOS Sierra (10.12.4). Using Terminal, I was able to download and implement FFmpeg - a open source project designed for the conversion of video and audio. FFmpeg was key in the reversal process, and as it is a well established tool, there is significant documentation and help available online from experts.

In addition, I dabbled in the use of OpenJPEG - a similarly open-source JPEG2000 tool that contains a decompression feature.

In order to ensure that I was using these tools correctly, I consulted with two software-development experts. They were instrumental in double checking my work and ensuring that I was using the tools to my full capacity as a beginner.

It should also be noted that the Macbook in use contains 500gB total storage, with 231gB available at the start of this project.

Preparation for Decompression

In order to ensure that any errors I experienced were the result of my work and the software I used, I controlled for the video itself by taking 5 clips from the Netflix production *Meridian.* This video is nearly 12 minutes long and designed for algorithmic testing. Most importantly for this project, *Meridian* is encoded in JPEG2000, and downloadable in the file format .mxf (.mxf is very common wrapper that is often used in the production industry[1]). *Meridian* contains many intentional artistic choices that are hard to compress, such as several seconds of cigarette smoke and several minutes of low lighting.

The 5 clips chosen are of some of the most difficult content for compression (including the smoke and lighting), and were chosen for the next phase of this plan, which will be to recompress the designated clips using common lossy and lossless codecs (including recompressing back into JPEG2000), and noting their variations.

The clip that was used most heavily for this portion of the project was a 5 second long extraction of the *Meridian* credits. These are written with a complex font, while the background is black, and was

[1] mediacom.dk. "Technical Specifications: Commercial File Delivery." Accessed May 5, 2017. http://www.mediacom.dk/media/3339612/TV%202%20-%20Technical%20specifications.pdf.

chosen based on the hope that the low variation in colors (being that there is only one shade of black and one of white) would take up less storage space.

This extraction step would likely not be used by anyone wishing to recreate my work and reverse their own file, as they would likely want their whole video.

The command I used in Terminal to extract the clip and what each component means is as follows:

```
ffmpeg -i /file/path/INPUT.mxf -ss 00:10:52.0 -t 5 -codec copy
/file/path/OUTPUT.mxf
```

- "Ffmpeg" is required to indicate that you wish to use FFmpeg
- "-i" followed by "/file/path/INPUT.mxf", indicates that INPUT.mxf is the input, or source file.
- "-ss" stands for "slow seek", where FFmpeg will seek through the file until it reaches the following time (in this case, 00:10:52.0 - 10 minutes and 52.0 seconds into the video), before it starts whatever actions are designated.
- "-t" designates the duration. Because -t is followed by 5, I have indicated that I wish to extract 5 seconds starting at the -ss (in this example, I'll get 00:10:52.0 - 00:10:57.0).
- "-codec copy" indicates that I wish to make a copy of the file, and that I don't wish to overwrite INPUT.mxf.
- "/file/path/OUTPUT.mxf" is what I want my file name to be, and where it should be located. If I were to say "/Users/JDoe/Desktop/Video.mxf", a .mxf file named "Video" would appear on my (J Doe's) desktop.

STRATEGIES FOR DECOMPRESSION

After ensuring that the resulting .mxf file had the same lossless JPEG2000 compression (for this file, both are Lossless JPEG 2000 YUV color 4:2:2) and file type as the original using MediaInfo[2], I outlined several strategies through which I could reverse the lossless compression and end up with a resulting uncompressed video file. Tool-wise, I determined that both FFmpeg and OpenJPEG could be used as both are free, open-source, and work on Mac. Strategically, I hypothesized that there were 2 methods to achieve decompression - by converting the video file as-is, or by extracting frames, and then decompressing and wrapping those frames to form a complete video file.

OpenJPEG

At first glance, OpenJPEG appeared to be the perfect software for decompressing my .mxf file. Research and OpenJPEG's documentation both indicated that the software came with a tool designed specifically to decompress lossless JPEG2000 files, by using the simple command "opj_decompress".

However, as software designed for experts, I found OpenJPEG very difficult to use. I required assistance from a developer to even compile (or in non-expert terms, "install") the program, and even after, when attempting to run the program.

After making headway and running files through OpenJPEG, all resulting attempts yielded a frame-rate of 25fps[3] (DVB standard - native to Europe, Asia, Africa, and much of South America, all of which formerly followed the PAL standard). Although this is not an issue within itself, as modern computers will have no issue playing back videos with a wide range of frame rates and video standards, this deviation from the 59.94fps (ATSC standard - North America, Central America, and some of South America, formerly NTSC) when decompressing indicates that the rate of the video was being fundamentally changed in this process.

As a beginner, this process seemed unintuitive, clunky, and although it was likely possible to change the frame-rate, I was intimidated by the high level of difficulty and barrier to entry of getting as far as we did.

---

[2] "MediaInfo - Download." Accessed May 5, 2017. https://mediaarea.net/en/MediaInfo/Download.

[3] OpenJPEG was developed and maintained by a team affiliated with Université catholique de Louvain, in Belgium. The software is actually written to compress, wrap, and do other functions at 25fps. When working with 1080p this gives a total of 25 progressive frames per second. This limitation is due to the lack of a European equivalent to 1080p60fps as defined by the ATSC. Thus, while technically possible through a major rewrite of the code, the inherent flaws in current European broadcast standards have crippled the current codebase for use in a production environment in non-PAL applications.

As one of the most important aspects of this project is that it is recreatable, I made the conscious decision to return to FFmpeg, and determine whether the decompression could be done exclusively there.

FFmpeg

When researching possible options for reversing the compression within FFmpeg, I chose .yuv as a file format for my decompressed video. As the video was already within the YUV colorspace (as seen in the aforementioned lossless JPEG2000 YUV 422 specs), it appeared to be a natural container format.

Research on whether the conversion was possible was similarly promising - there are many help-posts on how to convert and decompress files within FFmpeg, it was just the concatenation of the two that was difficult to find.

Conversion from Video to Video

In an attempt to streamline the process, and therefore make it more repeatable for other non-experts, I determined that by taking my .mxf and decompressing / converting it to a different file type in a single step would be the simplest option. This would be done by using the .mxf as the input, and then indicating that I wanted the video to be changed from lossless JPEG2000 (pixel format YUV 4:2:2) to "rawvideo" (pixel format YUVA 4:4:4), and ending with the intended output.

I hypothesized that the command below (as a synthesis of the above) would therefore successfully decompress and convert the file to a .yuv:

```
ffmpeg -i /file/path/INPUT.mxf -c:v rawvideo -pix_fmt yuv444p -c:v ayuv /file/path/OUTPUT.yuv
```

- "`ffmpeg`" is required to indicate that you wish to use FFmpeg
- "`-i`" followed by "`/file/path/INPUT.mxf`", indicates that INPUT.mxf is the input, or source file.
- "`-c:v rawvideo -pix_fmt yuv444p`" is that determines that the video will be transferred to raw (uncompressed), with the pixel format of YUV 444.

Frame Extraction and Re-Wrapping

If the single step video to video conversion failed, the secondary plan was to add in an extra component to extract the frames of the original file, and then decompress and wrap them as the desired .yuv.

This would take two different commands, one after the other. The first is as follows:

```
ffmpeg -i /file/path/INPUT.mxf
/file/path/output1folder/OUTPUT1_%04d.bmp
```

- "`ffmpeg`" is required to indicate that you wish to use FFmpeg
- "`-i`" followed by "`/file/path/INPUT.mxf`", indicates that INPUT.mxf is the input, or source file.
- "`/file/path/OUTPUT1.bmp`" is what I want my file name to be, and where it should be located. If I were to say "`/Users/JDoe/Desktop/Video.bmp`", a .bmp file named "Video" would appear on my (J Doe's) desktop.
- "`%04d`" in the name of the .bmp output file is called a "decimal wildcard[4]" - it indicates that FFmpeg will assign a number in its place. The 04d notes that there will be 4 numbers following. This means that when you go to /file/path/output1folder/, there will be hundreds of .bmp files of frames, starting with OUTPUT_0001.bmp, and going in order (0002, 0003, etc) until the input file does not have any more frames to extract.

After this has been completed, the second command would be:

```
ffmpeg -i /file/path/output1folder/OUTPUT1_%04d.bmp -vf format=rgb24
-pix_fmt yuva444p -c:v ayuv -r 2997/50
/file/path/OUTPUT2_fromframes.yuv
```

- "`ffmpeg`" is required to indicate that you wish to use FFmpeg

[4] "FFMPEG An Intermediate Guide/Image Sequence - Wikibooks, Open Books for an Open World." Accessed May 5, 2017. https://en.wikibooks.org/wiki/FFMPEG_An_Intermediate_Guide/image_sequence.

- "`-i`" followed by "`/file/path/output1folder/OUTPUT1_%04d.bmp`", indicates that the wildcard .bmp files are the input, or source file. Note that here, the frames that were previously the output in the previous command, are now the input.
- "`-vf format=rgb24 -pix_fmt yuva444p -c:v ayuv`" is required because .bmp files have a native colorspace of RGB24, but the desired decompressed file has a colorspace of YUVA. This string implements a conversion between those two colorspaces.
- "`-r`" specifies the frame rate, while "`2997/50`" is 59.94 in fraction form - due to the OpenJPEG frame-rate issues, I wanted to ensure that the correct rate would be maintained in these conversions.
- "`/file/path/OUTPUT2_fromframes.yuv`" is what I want my file name to be, and where it should be located.

TROUBLESHOOTING

When it came to implementing the above strategies, there were multiple problems on the .yuv commands.

Both commands would create a .yuv, but when running the subsequent .yuv files through MediaInfo, Exiftool (another metadata reader), the only information that was displayed was the amount of space the file took up (approximately 25gB). In comparison, the previous .mxf files would display detailed information about video and audio streams, including the codecs used, the frame rate, and the pixel dimensions of the video.

In order to determine whether the issue was with the source .mxf files (perhaps the clip extraction had failed), I attempted to play both the .mxf and .yuv files using VLC Media Player[5]. Unfortunately, all files were unable to play, as even though the .mxf files had been compressed, they were still good quality and too large for my computer to handle. In an attempt to otherwise see whether there was video material stored in the video files, I made animated .gif files of the clips. These .gif files would be smaller and viewable, and were created using the following command:

```
ffmpeg -i /file/path/INPUT.yuv -vf scale=500:-1 -t 10 -r 10
/file/path/OUTPUT.gif
```

- "`ffmpeg`" is required to indicate that you wish to use FFmpeg

[5] "Official Download of VLC Media Player, the Best Open Source Player - VideoLAN." Accessed May 5, 2017. https://www.videolan.org/vlc/index.html.

- "`-i`" followed by "`/file/path/output1folder/OUTPUT1_%04d.bmp`", indicates that the wildcard .bmp files are the input, or source file. Note that here, the frames that were previously the output in the previous command, are now the input.
- "`-vf`" stands for video filter, while "`scale`" is the filter that will reduce the pixel dimensions of the video. Because "`scale=500:-1`" the largest dimension of the resulting file will be 500 pixels.
- "`-t`" designates the duration. Because -t is followed by 10, I have indicated that I wish to extract the first 10 seconds of the file.
- "`-r`" designates the frame-rate, so "`-r 10`" sets the rate to 10fps, which is another way to ensure that the file stays small enough to open on an average computer.
- As before, "`/file/path/OUTPUT.gif`" is what I want my file name to be, and where it should be located.

When this command had been implemented for the .mxf files, a gif exactly to the outlined specifications was created. However, when the exact same command was run on the .yuv files, I received the error "`Picture size 0x0 is invalid`". I then added "`-s 3840x2160`", which specifies the picture size, but the error would not change.

After attempting variations within the command such as changing the pixel format or altering the sizes and scales, I determined that the issue must be with the .yuv output, rather than the command itself - why should the exact same language produce different results? It must be related to the only component of the command that changed.

In order to determine whether it was the .yuv component that was the issue in the command, I modified ".yuv" to ".avi", which is the file wrapper native to Windows Media Player. AVI was chosen because it is a wrapper that can contain uncompressed video, and was chosen over .mov because .avi operates in the YUVA colorspace, while .mov does not[6].

This change to .avi cleared up all the errors, and produced playable files, which may be requested by contacting the author of this paper, Winnie Schwaid-Lindner, at wschlin@gmail.com.

[6] "Linux - How to Create an Uncompressed AVI from a Series of 1000's of PNG Images Using FFMPEG - Super User." Accessed May 5, 2017. https://superuser.com/questions/347433/how-to-create-an-uncompressed-avi-from-a-series-of-1000s-of-png-images-using-ff.

The 5 second clip of the credits was decompressed to around 200gB, and the file resulting file was unable to save due to the lack of sufficient local storage on my computer. To put this into relative terms; 40% of this computer's total storage was not sufficient for a 5 second video.

In order to be able to save the resulting files and review their metadata, I created new 1 second long .mxf files of the *Meridian* source material, and then applied the previous efforts to create the resulting YUVA 4:4:4 (uncompressed) .avi files.

RESULTS

Both the .avi files created successful gifs, indicating that they did contain the intended video content, and the metadata displayed similarly indicates that video is contained.

However, there is an interesting discrepancy in the metadata from the result of the frame extraction when compared to the metadata from the video to video conversion:

Frame to Video (.bmp to .avi)	
Format	AVI
Format/Info	Audio Video Interleave
Format profile	OpenDML
File size	1.85 GiB
Duration	2 s 369 ms
Overall bit rate	6 722 Mb/s
Writing application	Lavf57.71.100
Format	YUV
Codec ID	AYUV
Codec ID/Hint	YUV
Width	3 840 pixels
Height	2 160 pixels
Display aspect ratio	16:9

Video to Video (.mxf to .avi)	
Format	AVI
Format/Info	Audio Video Interleave
Format profile	OpenDML
File size	1.85 GiB
Duration	1 s 1 ms
Overall bit rate	15.9 Gb/s
Writing application	Lavf57.71.100
Format	YUV
Codec ID	AYUV
Codec ID/Hint	YUV
Width	3 840 pixels
Height	2 160 pixels
Display aspect ratio	16:9

Frame rate	59.940 (60000/1001) FPS
Color space	YUVA
Chroma subsampling	4:4:4
Bit depth	8 bits
Compression mode	Lossless
Bits/(Pixel*Frame)	13.521
Stream size	1.85 GiB (100%)

Frame rate	59.940 (60000/1001) FPS
Color space	YUVA
Chroma subsampling	4:4:4
Bit depth	8 bits
Compression mode	Lossless
Bits/(Pixel*Frame)	32
Time code of first frame	00:00:17;12
Time code source	ISMP
Stream size	1.85 GiB (100%)

The fields marked in grey above are those that differ.

The top two differing fields, duration and overall bit rate, are interesting in that the clip was pre-specified to be 1 second long. The .mxf to .avi metadata specifies that that clip is 1 second long, while the table on the left, the .bmp to .avi metadata, is 2.369 seconds. Inversely, the bit rate of the video to video conversion on the right is 2.369 times that on the left.

When looking at the output data from the .bmp to .avi file, the last line reads "`frame= 60 fps=5.9 q=-0.0 Lsize= 1944009kB time=00:00:02.36`", which is interesting considering that the frame-rate portion of the command (-r 2997/50) should be forcing the file to 59.94fps, making it impossible for the 60th frame to occur into the following second. I have tried modifying both the frame-rate and the bit-rate in order to determine what effect that has on the resulting .bmp to .avi file, but have not seen any change thus far.

Furthermore, the metadata of both files notes "Compression mode: Lossless", even though "YUVA 4:4:4" would indicate that there is no "compression mode" on the file.

When using .mov (another wrapper) as an output instead of .avi, one would have to revise the colorspace used, as AYUV is not native to .mov or quicktime. Once this has been taken into account by changing the pix_fmt (from "`yuv444p`" to "`yuva444p`") and the `-c:v (`from "`ayuv`" to "`v408`"), the command should produce similar results.

CONCLUSION / INSTRUCTIONS

Based on these two newly created files and their respective sets of metadata, I will be using video to video decompression/conversion to reverse my lossless JPEG2000 files. Although both strategies required outside assistance and extensive troubleshooting, the resulting one-step video to video command should work on all lossless JPEG2000 files.

Any self-identified beginner can take the following steps to reverse their own lossless JPEG2000 file:

1. Download FFmpeg[7]
2. Open your local Command Line Interface (the application is called Terminal on macOS)
3. Identify the file that you would like to decompress - would you like to convert the entire file, or just a portion?
    a. If only a portion, make a clip by modifying the following command to reflect your file, and then paste it in the CLI and hit enter
        i. `ffmpeg -i /your/file/path/here/INPUT.mxf -ss 00:10:52.0 -t 5 -codec copy /your/file/path/here/OUTPUT.mxf`
            1. Replace "`/your/file/path/here/INPUT.mxf`" with the file path of the video you would like to convert - this can be accomplished by dragging the file into the CLI on a mac.
            2. Replace "`00:10:52.0`" with the timecode your desired clip should start at
            3. Replace "`-t 5`" with "`-t`" and however many seconds you would like your clip to be
            4. Replace "`/your/file/path/here/OUTPUT.mxf`" with the name and the location you want the resulting clip to be.
4. Make sure that you have enough storage space on your computer to decompress the video you want - even a really short video can take up a significant amount of space!

[7] "FFmpeg." Accessed May 5, 2017. https://ffmpeg.org/.

5. When you're ready, modify the following command to reflect your file, and then paste it in the CLI and hit enter

    a. To .avi

        i. `ffmpeg -i /your/file/path/here/INPUT.mxf -r 2997/50 -c:v rawvideo -pix_fmt yuv444p -c:v ayuv /file/path/OUTPUT.avi`

            1. Replace "`/your/file/path/here/INPUT.mxf`" with the file path of the video you would like to convert - this can be accomplished by dragging the file into the CLI on a mac.

            2. Replace "`/your/file/path/here/OUTPUT.avi`" with the name and the location you want the resulting clip to be.

    b. To .mov

        i. `ffmpeg -i /your/file/path/here/INPUT.mxf -r 2997/50 -c:v rawvideo -pix_fmt yuva444p -c:v v408 /file/path/OUTPUT.mov`

            1. Replace "`/your/file/path/here/INPUT.mxf`" with the file path of the video you would like to convert - this can be accomplished by dragging the file into the CLI on a mac.

            2. Replace "`/your/file/path/here/OUTPUT.mov`" with the name and the location you want the resulting clip to be.

6. Your lossless JPEG2000 file should now be decompressed!

SOURCES REFERENCED

"Encode/H.264 – FFmpeg." Accessed May 5, 2017. https://trac.ffmpeg.org/wiki/Encode/H.264.

Federal Agencies Digitization Guidelines Initiative. "Digital File Formats for Videotape Reformatting," December 2, 2014. http://www.digitizationguidelines.gov/guidelines/FADGI_VideoReFormatCompare_pt5_20141202.pdf.

"FFmpeg." Accessed May 5, 2017. https://ffmpeg.org/.

"Ffmpeg - Convert H264 Video to Raw YUV Format - Stack Overflow." Accessed May 6, 2017. http://stackoverflow.com/questions/20609760/convert-h264-video-to-raw-yuv-format.

"FFMPEG An Intermediate Guide/Image Sequence - Wikibooks, Open Books for an Open World." Accessed May 5, 2017. https://en.wikibooks.org/wiki/FFMPEG_An_Intermediate_Guide/image_sequence.

"Ffmpeg Documentation." Accessed May 5, 2017. https://ffmpeg.org/ffmpeg.html#Advanced-Video-options.

"Linux - How to Create an Uncompressed AVI from a Series of 1000's of PNG Images Using FFMPEG - Super User." Accessed May 6, 2017. https://superuser.com/questions/347433/how-to-create-an-uncompressed-avi-from-a-series-of-1000s-of-png-images-using-ff.

mediacom.dk. "Technical Specifications: Commercial File Delivery." Accessed May 5, 2017. http://www.mediacom.dk/media/3339612/TV%202%20-%20Technical%20specifications.pdf.

"MediaInfo - Download." Accessed May 5, 2017. https://mediaarea.net/en/MediaInfo/Download.

"Official Download of VLC Media Player, the Best Open Source Player - VideoLAN." Accessed May 5, 2017. https://www.videolan.org/vlc/index.html.

"PIX FMT YUV420P - MultimediaWiki." Accessed May 6, 2017. https://wiki.multimedia.cx/index.php/PIX_FMT_YUV420P.

"Recommended 8-Bit YUV Formats for Video Rendering (Windows)." Accessed May 5, 2017. https://msdn.microsoft.com/en-us/library/windows/desktop/dd206750(v=vs.85).aspx#ayuv.

"Useful FFmpeg Commands For Converting Audio & Video Files." Accessed May 6, 2017. https://www.labnol.org/internet/useful-ffmpeg-commands/28490/.

"Video - FFmpeg BMP to YUV x264 Color Shift - Video Production Stack Exchange." Accessed May 6, 2017. https://video.stackexchange.com/questions/19944/ffmpeg-bmp-to-yuv-x264-color-shift/19958.

"Video - How to Make FFmpeg Use Output Codec and Pix_fmt Same with Input - Stack Overflow." Accessed May 5, 2017. http://stackoverflow.com/questions/25356202/how-to-make-ffmpeg-use-output-codec-and-pix-fmt-same-with-input.