Image compression: On-demand systems seeking minimal latency.

Kobe Vrijsen

January 24, 2023

Abstract

Latency and loading times in on-demand settings such as web pages and network streaming and how to minimise it. Image downloads generally take up a large portion of loading times and this paper will discuss and test several methods and formats that aid to reduce latency. Analysing variable factors and their effects on loading times. Discussing both common formats, including PNG, Webp and JPEG, as well as lesser-known compressors like Qlic. Is there an excelling format or does it differ on a per use case basis. This paper aims to provide insight and hopes to assist making choices regarding image compression and how to use it.

Keywords Image compression, Image formats, Latency, Loading times, Web pages.

Contents

Chapter 1

Introduction

On-demand systems seeking minimal latency. Time is key, specifically as little of it as possible. Can image compression be utilised to improve client-side latency? If so, what factors play part and which method offers the best solution? Or is it perhaps determined by its use case?

How compression affects performance as well as overall loading times will be the main focus of this paper. If compression is beneficial, what factors need to be taken care of and kept in mind to get to the best possible solution for our system?

1.1 The system

1.1.1 On-demand systems

Web pages, network streams, games, chats rooms, the list goes on. All are examples of systems that are, either in part or in full on-demand systems. What defines an on demand system is that resources to be requested are unknown before before they are requested. As soon as it is required or has to be shown, a request must be sent out which is responded to with the necessary data. This is the basic concept of on-demand systems. The time it takes from request to result is the heart of the experience. Every tool in the pipeline can be a potential bottleneck. This paper will focus on the data requested and how to provide an optimal solution to reduce the overall latency to a minimum at the best of our abilities.

1.1.2 Images

Continuing on the request data. Two-dimensional graphical raster data will be the focus from here on out: I.e. images, photos and textures. Images, as they will be referred to, commonly take up large portions of overall requested data. This is dues to their large quantity data required to correctly reproduce all details. A raw image can easily reach tens of megabytes. Compared to other request data, which do not often exceed the megabyte mark, images are very much in need of data saving measures.

One does not have to look far, there's already an uncountable amount methods and formats that are used or have been proposed. Some have been useless, others have brought improvements, while some have set a true precedent going beyond what was previously thought possible. It has

become a rather rare occurrence for any person to encounter a normal raw image. Image compression has become trivial enough for it to be a fair waste to not make use of it.

1.1.3 Requests

A request is made with the goal to ultimately receive a response. While for may years in human history severely slow and lossy mouth to mouth communication had been all there was, in this day and age there is a strong sentiment for requests to be as close to instantaneous as possible while transmitting the response in a perfectly intact state.

Firstly a request is sent out over a network, before a response is received there is a moment of silence. Neither light travels instantaneous nor will our data. This term is referred to as network latency or response time. From a client's perspective, this can unfortunately not be changed. Because of this the frame of reference in this paper will begin at the time the first glimpse of the response is received.

Next, the response will carry data that is not of no size. It will take time to receive the full response. This time is determined by the capability of the network, called the network bandwidth or speed, as well as the size of the total response size.

1.1.4 Processing

Data received from out response isn't just any data that can be copied straight to the screen. At least not when talking about compression. A compressed set of data has to be processed and inflated back to its original full sized content and possibly have some additional changes to its layout applied. This will inevitably take time. This may be referred to as processing time, decompression time, initialisation, loading time, etc. After the data transmission time itself, the download, this phase will most likely affect the final loading time, or latency, the most.

1.2 Existing methods

There are several methods of compression available. There's general compression algorithms that can be used as well as image specific compression algorithms. In the latter category there's also a distinction between lossless and lossy compression. As opposed to lossless, lossy compression results in a reduction of detail.

1.2.1 No compression

There are several formats used to save images that do not apply compression at all. Each one may have its reasons to do so. A few are listed below but will not be included in the benchmark. They will, however, serve as source files instead for our test cases.

PPM Portable Pixmap Format, PPM [Poskanzer & Henderson. 2021]. An extremely simple format saving essentially only the raster size and pixel data.

BMP A simple format to store bitmap data [Leonard. 2016].

1.2.2 General compression

General compression algorithms focus on compressing any kind of binary data. While not optimised for images, this does not hinder their ability to do so. In fact, formats like GIF, PNG and more formats than one might expect all default to this method internally. To get a frame of reference against image specific compression algorithms, the following algorithms will also be included in the benchmark:

LZ77 Lempel-Ziv [Ziv & Lempel. 1977]. Commonly referred to as Deflate. This is the default algorithm for containers like ZIP and and PNG. It is also often used to compress on-the-fly network streams. An old and all-round acceptable algorithm.

LZMA Lempel–Ziv–Markov chain algorithm. Developed by the same author as the 7-zip compression tool [Pavlov. 2022]. An extension of the default Lemper-Zif algorithm, it tries to improve compression by tweaking its values.

GNU gzip A compression algorithm and tool created by the Free Software Foundation [FSF. 2020]. Commonly used on Linux like operating systems and by developer communities. A common algorithm used to compress tarballs and also used to compress on-the-fly network streams.

bzip2 An implementation of the Burrows-Wheeler algorithm [Seward. 2019].

PPMd Prediction by Partial Matching. An implementation by Shkarin Dimitri [Shkarin. 2003].

1.2.3 Image compression

Image compression comes in many forms. From general lossless compression to image specific lossy compression. Each with its own advantages and disadvantages. We'll mostly be discussing the formats and algorithms commonly used on the web today as well as some up and coming algorithms with promising features.

Web

Common formats fount on the web and in browsers are include:

PNG Portable Network graphics, PNG. A very common lossless image format. This is a commonly used format for the lossless storage of images [W3 Consortium. 2022].

Jpeg Developed by the Joint Photographic Experts Group, another very common format that withstood time [JPEG. 1992]. Unlike any other formats listed in this paper. Jpeg does not allow lossless compression of an image. While at its best it comes close, it cannot be considered lossless.

GIF GIF, a bitmap image format developed for motion graphics, image sequences. [**?**]. Gif only stores images with a 256 colour palette. While lossless, the 256 colour limitation is generally not considered true colour and will thus not be regarded as lossless in this paper.

WebP WebP, An image format for the Web. A format developed and advertised by Google llc., Alphabet Inc [Goolge llc. 2022]. For this reason WebP is implemented within the chromium engine and thus subsequently widely supported across browsers and can be seen from time to time. Since this format is advertised for use on the web and over the network it is a reasonable guess to expect it to perform well throughout the tests. WebP implements the VP8 and VP9 video coding format to achieve it's compression.

WebP 2 WebP 2, WP2. As the name suggests, the followup op WebP [Google llc. 2022]. Currently an experimental format. A potential improvement to the current version of WebP.

HEVC High Efficiency Video Coding, HEVC, also known as H.265, and High Efficiency Image Format or Coding, HEIF and HEIC. You may recognise H.265 from Mpeg-4 video files. It is commonly used to compress video footage. It offers acceptable reductions in size as well as it commonly being implemented in hardware to aid compression and decompression.

Videos are essentially a sequence of images. that means it can also be used to compress still images. This is where the file format HEIF or HEIC comes from. Newer version of the HEIC format also support AV1. More about it in the next paragraph.

AV1 AOMedia Video format, AV1, and AV1 Image File Format, AVIF. AV1 is based on HEVC, as discussed in the previous paragraph, HEVC is mostly a proprietary format. For this reason efforts have been made to evolve the format into an open format. The result is AV1.

AV1 is also developed to compress video but can also be used to compress just images and these are saved in an AVIF file. Unlike HEIC and JXL. AVIF files are picking up in browser support and newer versions of some browsers, including Chrome, Firefox and Safari, already support this format [Mozilla Foundation. 2023]. A formal specification has been produced by [Han et al. 2021].

Jpeg XL JpegXL. A royalty free format developed in the hopes of outperforming common formats and provide a universal alternative. It is based on the proposed format FUIF and the existing format FLIF. A specification is produced by the Joint Photographic Experts Group [ISO/IEC JTC 1/SC 29. 2022].

Niche

While well known formats are often good enough. There are a number of lesser known compressors out there, including some tailored towards images. Potentially, they may outperform other compressors in a handful of test cases. The following algorithms have been included to see how they would mix in and whether they offer any substantial advantages.

[Rhatushnyak. 2022] Quick and Fast Lossless Image Compression, FLIC, QLIC, QLIC2 & QIC. These four algorithms have been created by Ratushnyak Alexander under Graystone Compression Technologies Inc. some time around the year 2010. While not at all recognised or well known, researching the topic of lossless image compression brought these little known formats to my attention. According to the sizable benchmark orchestrated by Ratushnyak himself, his formats appear to outperform conventional formats. Most notably in speed [Rhatushnyak. 2022].

For this reason I have included these formats in my tests to see how they would perform in relation to more widely known formats. Other than the author of these formats, little is known about the format itself or implementation. They also do not appear free to use without restrictions. All of these formats have been provided with a single executable able to compress and decompress PPM images.

EMMA In 2020, MSU hosted a Competition rewarding contestants for making the best compression algorithms, Global Data Compression Competition 2020 [MSU Media Group. 2020]. The tests were split into several categories. One of which were images. They were further subdivided in speed, balanced and compression. In regards to the overall best compression ratios, EMMA took the crown.

Kvick Part of the Global Data Compression Competition mentioned before, in the rapid category the leading algorithm was Kvick. It also got itself an honourable mention in the balanced test.

Chapter 2

Case study

Before gathering data, a model will have to be worked out to decide on what data has to be collected and what to look out for. As well as mapping that data to a satisfactory and concluding model. After this step a methodological model and with its constraints can be put forward to collect the necessary data.

2.1 Subjectivity

"Lossy"

Lossy compression, as has been mentioned several times, achieves its goals by discarding detail. This in the hopes of not discarding too much while still retaining visual recognition for the end user. While this allows for great reduction in size and processing time, it does come with less detail. Therein lies the problem, how much or how little detail, other than all of it, is within "acceptable levels". Is "acceptable" even enough?

Hypothetically one could make a lossy compression algorithm by taking an image as input and save a black images as its result. Disregarding all subjective matter, who is there to say this image is or is not good enough. There is no true objective way to determine whether a lossy compression algorithm is any good at any level. The best, and possibly only, thing that can objectively be agreed upon is a lossless, that is a perfect one-to-one, mapping of data from source to result.

What this means is the JPEG is already eliminated as a possible candidate in our case study. In addition, GIF, while it implements lossless compression internally it can only save images that use up to a 256 unique colours. Since most images consist of many more colours, GIF will also not included in this case study.

2.2 Analysis

2.2.1 Network speed

Network speed is an important factor in any on-demand system. network speed is directly proportional to the decrease in download time. Unlike network speed or download time, the execution of decompression is mostly unrelated to variable factor. Since the combination of both determines

the latency experienced by the user, it is important to analise the effects of a variable network speed in combination with the decompression time given a certain image and algorithm.

In order to better understand the effects, lets pose the following hypothetical model:

$$T = (size * ratio)/speed + latency \tag{2.1}$$

Where $size$ is the total size of the request, $ratio$ is the achieved compression ratio, $speed$ is the network carriage capacity over a set frame in time and $latency$ is introduced by the decompression algorithm by performing the decompression. Here it can also be reasonably assumed that the latency is in constant relationship to the size on the same device with the same data. More about this at **??**.

The most variable constant in our equation will be the client network carriage capacity. Assuming the decompression time is more or less constant is relation to the size, the following two scenarios can be analised.

$$T_0 = size_0/speed \tag{2.2}$$

$$T_1 = (size_0 * ratio)/speed + latency \tag{2.3}$$

Where $ratio$ is the compression ratio achieved. T_0 is the time it takes without compression, T_1 shows the time with compression. Taking a skeptical compression ratio of 90% and the have the latency be about about 10e7-th of the size in bytes in seconds. Given 1MB, this equates a reduced size of 900KB and an additional latency of 100ms, the curve in figure 2.1 can be derived if by taking T_1 over T_0. This puts T_1's performance in relation to the default compression-less T_0.

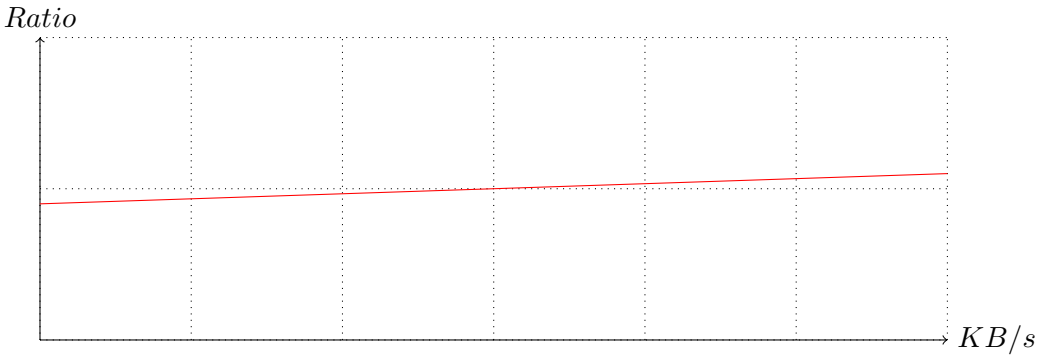


Figure 2.1: Comparison to default. T_1/T_0

Figure 2.1 reveals an interesting relationship. Notice the line trends upwards starting out positive, but crosses 1 at about 1MB/s. This is due to the network being capable enough of loading the entire file quicker than the decompression algorithm has time to decompress the file. Analising this relationship the following can be noted. The compression will only improve the overall time if the network speed is below a certain threshold.

$T_1 < T_0$ holds true if and only if:

$$speed < size_0 * (1 - ratio)/latency \tag{2.4}$$

$$speed < (size_0 - size_1)/latency \tag{2.5}$$

This formula gives us an interesting way to decide if a given algorithm is worth it. While any algorithm that even marginally reduces size will provide an improvement in worst case scenarios, if it only does so with limited network speed it's not worthwhile to use the algorithm.

now, what good does it make to compare each algorithm to none at all. Discussing the relationship between algorithms and make comparisons between them is also possible. Taking the previously discussed formulas, compare two different algorithms as follows:

$$T_1 = (size_0 * ratio_1)/speed + latency_1 \tag{2.6}$$

$$T_2 = (size_0 * ratio_2)/speed + latency_2 \tag{2.7}$$

Plotting the curve in figure 2.2. the compression ratio in example 1 is 80% and 60% in example 2. The latency in example 2 is twice the latency of 1; 100ms and 200ms respectively. The size of the original file is 1MB.

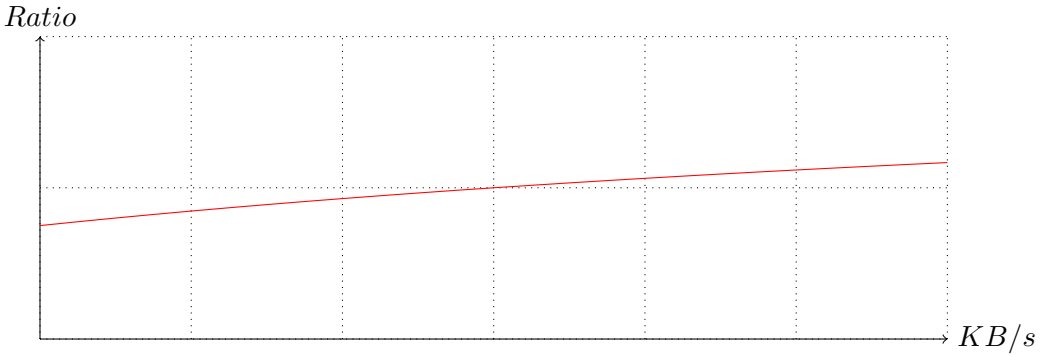


Figure 2.2: Comparing two functions. T_2/T_1

As can be seen here, T_2 appears to be faster than T_1 when the network speed is below, in this example, about 2MB per second. The following derived formula can calculate the tipping point and below which the latter algorithm wins over the former.

$$speed < size * (ratio_1 - ratio_2)/(latency_2 - latency_1) \tag{2.8}$$

This is logical as when T_1 is substituted to be that of no compression, I.e. T_0, it reduces to formula 2.4 again. The tipping point varies based on the difference in compression ratio and latency as expected. A better compression ratio and a lower latency also positively impact the performance compared to the other. However, it also in relation to the total size. For a larger file, a better compression ratio can beat a faster algorithm.

2.2.2 Time complexity

Up until now, the size and latency has been assumed to be constant. While true for a specific image, the values might differ greatly amongst them. First of all, the initial size of the image may

determine the total amount of latency added. Secondly, the contents or complexity of the image may also have an effect on latency.

For this reason the following model to analise the effect can be posed:

$$latency = O(size) * factor * (1 \pm error) + base \tag{2.9}$$

Algorithms are commonly described by their O time complexity, this forms the main term of our model. The $factor$ determines the slope of the complexity increase. The $error$, or deviation, can also be taken into account. Some algorithms may perform close to their theoretical model while others might be greatly unpredictable. lastly it must also acknowledged that some algorithms require some extra time to get ready or clean up. this is expressed through the trailing $base$ term.

Big O notations range from $O(1)$ to $O(\infty)$ but the easiest to understand is $O(n)$. This is a linear time complexity. For an arbitrary value 10 the time would be twice as much as that of 5. This is the complexity that had been assumed while discussing network bandwidth.

To calculate if an algorithms offers less latency than a second algorithm, and under which initial values. The following statement can be posed:

$$O_1 * factor_1 + base_1 < O_2 * factor_2 + base_2 \tag{2.10}$$

This equation can be lightly simplified as follows:

$$O_1 < O_2 * F + \Delta_b / factor_1 \tag{2.11}$$

$$F = factor_2 / factor_1 \tag{2.12}$$

$$\Delta_b = base_2 - base_1 \tag{2.13}$$

The error is left out of the equation in this section as it adds complexity and is based on practical testing. The base time is not left out because this values tends to greatly impact the result similarly to the slope factor.

To calculate the tipping point. O_1 and O_2 may be filled in and the equation can subsequently be solved for the $size$ of the input. However, since some O functions can be quite complex the equation might not be easily solvable; E.g. substituting O_1 for x and O_2 for $ln(x)$, seemingly simple formulas, the equation becomes almost unsolvable. Additionally, for some formulas there may be multiple intersecting points. For this reason it is recommended to use graphing software to locate and determine an approximation of the intersections.

Simplifications

This equation can be simplified under certain conditions. This would make solving the problems a lot easier.

Where O_2 and O_1 describe the same complexity graph. The equation reduces in complexity:

$$O < -\Delta_b / \Delta_f \tag{2.14}$$

$$\Delta_f = factor_2 - factor_1 \tag{2.15}$$

This equation, unlike equation 2.11, is far easier to solve for most complexity functions. Note that if the factor of function 2 is smaller than 1, the function is never smaller than function 2 unless the counteracts this difference.

Where O_2 describes $O(1)$, or O_1 in which case the terms are switched, further simplifications can be applied to the equation. This can alternatively be described as only the base component being present. Although it is unlikely that any algorithm to be tested has such a time complexity.

$$O_1 < F + \Delta_b / factor_1 \tag{2.16}$$

Given 2.9 it is possible that the base is so large that any effect of the previous term is negligible. However this almost always means that that algorithm is an extremely poor performer.

Similarly, where the factor or growth of the first terms can be so large that the base shows almost no effect. This would simplify 2.11 to:

$$O_1 < O_2 * F \tag{2.17}$$

This also greatly reduces complexity and allows the equation to be solved more easily. For this equation where O_1 and O_2 also describe the same complexity, something interesting happens:

$$1 < F \tag{2.18}$$

I.e. if function 1 has a weaker slope than function 2, the function is the smallest regardless; The function with the weakest slope is a winner.

Conclusion

Whether any of the simplified equations can be used to present predictions within margin remains to be seen. It is required to run several tests to measure both basic performance as well as complexity and slope.

2.2.3 Streaming algorithms

Streaming algorithms, as opposed to non-streaming algorithms, can process sequenced information from start to finish in a few passes, typically just one, with either no or minimal arbitrary lookup [Wikipedia. 2022]. A sliding window also largely counts as single pass. Such algorithms can start their operation even before all data is available. In 2.2.1 latency has been discussed, after the calculation of the download time a fixed latency was applied. This assumes the full response is required before the algorithm can be run. What streaming algorithms change in this regard is that they are already processing the data during the download. In the best case scenario this reduces the latency to a fraction of their otherwise significant value. Whereas in the worst cases, such as the download being faster than the algorithm itself, it does not change anything. This means that a streaming algorithm has a major advantage over non-streaming algorithms.

Calculating this benefit is not straightforward and the effects should thus be tested through practical tests. The expectation is that there will be either no impact or a positive improvement.

2.2.4 System resources

System resources will also affect the performance of each algorithm. While its easy to say it causes equal slowdown to each algorithm, this may be wrong. There are multiple factors that can influence performance. This includes but is not limited to: processor spreed, processor cores, memory latency, thread scheduling, memory capacity, processor architecture, etc. It is almost guaranteed when listing all factor to forget some. The expectation is for it to majorly affect results in a negative way. The extent of this is uncertain.

2.3 Methodology

2.3.1 Software

Each algorithm's performance will be measured through their reference implementation, where available. An executable provided by either the official maintainers or a reputable source. For general compression algorithms 7-zip was used. For image compression if such an implementation is hard to get by or is not available Imagemagick [ImageMagick Studio llc. 1999] was used. This turned out to be only PNG.

Furthermore. Each executable will be executed on a single thread with no memory limit. This is to eliminate executables that haven't implemented parallel execution. Either for the fear of inefficiently implementing their algorithms or incapability of fully utilising the system. The other reason is that not all executables will be able to utilise the system to their best extent. This is why a single thread is the best option to stack the executables against each other.

All formats and algorithms have been tested using this dedicated executable. Each executable receives the raw image as a PPM container together with the supplied parameters for that algorithm specifically which may include effort level and quality. Not all executables accept a PPM input, in which case the file is converted to either a bitmap, BMP, or PNG (Effort 0) in this respective order; Notable exceptions are `cwebp.exe` which accepts BMP and `avifenc.exe` which accepts PNG.

Here is a list of all executables used and their supplied parameters

Lossless

Compressors developed and optimised for images.

```
# PNG
# Using imagemagick
convert -define png:compression-level=E
convert

# JpegXL
cjxl -q 100 -e E
djxl
```

```
# AVIF
# Only accepts .png
avifenc -l -s (10 - E)
# Only outputs .png
avifdec --png-compress 0

# WebP
# Only accepts .bmp
cwebp -lossless -m E
# Does output .ppm
dwebp -ppm

# WebP2
cwp2 -q 100 -effort E
dwp2 -ppm

# QIC, QLIC, QLIC and FLIC
qic c
qic d

# kvick
kvick c i
kvick d i

# EMMA
emma_c
emma_d
```

Where E is the effort level, depends on the executable. Generally ranging from 0 to 10.

PNG, Convert tool part of Imagemagick [ImageMagick Studio llc. 1999]. Qic, Qlic, Qlic2 and Flic as testd on the Lossless Photo Compression Benchmark [Rhatushnyak. 2022]. Kvick and EMMA submitted to the Data Compression Global Competition [MSU Media Group. 2020].

General

Binary compressors irrespective of content type.

```
# PPMd
7z a -mmt=1 -mx=E -m0=PPMD
7z e

# BZip2
7z a -mmt=1 -mx=E -m0=BZip2
7z e
```

```
# LZMA2
7z a -mmt=1 -mx=E -m0=LZMA2
7z e

# Deflate
7z a -mmt=1 -mx=E -m0=Deflate
7z e

# GZip
# Output file extension: .tgz
7z a
7z e
```

Where E is the effort level, depends on the executable. Generally ranging from 0 to 10.

The 7-zip compression tool: A versatile tool implementing many common compression algorithms [Pavlov. 2022].

Lossy

Compressors developed for images, hoping to improve by discarding detail.

```
# JpegXL
cjxl -q Q -e E
djxl

# AVIF
# Only accepts .png
avifenc -q Q -s (10 - E)
# Only outputs .png
avifdec --png-compress 0

# WebP
# Only accepts .bmp
cwebp -q Q -m E
# Does output .ppm
dwebp -ppm

# Webp2
cwp2 -q Q -effort E
dwp2 -ppm

# Jpeg
cjpeg -quality Q -optimize
```

```
djpeg

# MozJpeg
cjpeg-static -quality Q -optimize
djpeg-static
```

Where E is the effort level, depends on the executable. Generally ranging from 0 to 10. Q is the quality level, depends on the executable. Generally ranging from 0 to 100.

JXL reference implementation [JPEG XL Project. 2022]. AVIF reference implementation [AOMedia. 2023]. WebP reference implementation [Goolge llc. 2022]. WebP2 reference implementation [Google llc. 2022]. Jpeg-Turbo [The libjpeg-turbo Project. 2022]. Mozjpeg [Mozilla Foundation. 2022].

2.3.2 Benchmark

To make sure that reading and writing files to the disk is not a bottleneck of great proportions, it was decided to perform the tests on a RAM disk. This is, similar to a normal disk, a space in RAM memory where the file files are read and stored. This mitigates the bottleneck caused by sending huge amounts of data to and from the drives which would add latency and bandwidth limitations. The program used to achieve this has been ImDisk [w77. 2022].

To perform the benchmarks a script can be used to ease and automate the process. For the benchmarks in this paper PowerShell was chosen because it is easy to develop simple scripts able to access executables on the system with the additional benefit of having measuring functionality and data manipulation [Microsoft. 2022]. The script used to perform the benchmarks can be found at appendix E.

While executing the benchmark data is collected: the time it takes for the compressor to execute and complete its operation, the time it takes for the decompressor to execute its operation, the size of the compressed output, the size of the decompressed output, an optionally the difference in colour.

Measuring the time for the executable to operate includes: The executable starting, it loading the source file, performing its operation, and writing the results out to its destination. By performing the benchmark on a RAM disk the reading and writing should correlate to the internal copy of buffers between different stacks and components in most applications.

Image set

The performance is most likely determined by the size of the source image as well as its contents. To test this a benchmark will be performed using a large set of different images ranging from small to large with a variety of contents.

As a basis for the set of test images. The images used in the benchmark performed by [Anonymous. 2020] will be used. The total size of the library total just over 16 thousand items.

As a reduced set, to perform a benchmark large enough to show trends, the game screenshot set will be used. Games offer a wide variety of styles, colours and detail, as well as text and different edges. This may form a good basis for a test requiring various images. This set consists of about 500 images.

To perform quick tests a set of 7 images is used. All images used can be found in appendix A;

Effort

Some algorithms allow an effort or speed level to be specified. To test the effects of this some tests on different error levels are performed. The algorithms in question with their respective effort levels are: AVIF $[0, 10]$, JXL $[1, 9]$, WEBP $[0, 6]$, WP2 $[0, 9]$ and PNG $[0, 9]$. A benchmark will also be performed to test the result of this.

Chapter 3

Experiment

3.1 Hurdles

3.1.1 Time

The essence of this case study, time, also played part in the benchmarks. Performing the small test, benchmark times exceeding a quarter of an hour were not uncommon. Since this is a benchmark of seven images, extrapolating this time to the medium test set of 500 images makes that test worth optimising.

Performing the benchmark on the large sixteen thousand images set falls outright out of the question. This test could easily exceed the times denoted in multiples of days. For this reason this test has not been performed.

To realise reasonably attainable benchmark times, it was decided to leave the algorithm EMMA out of further tests. It was shown in the small test that EMMA took considerably more time to both compress and decompress the image. For this reason it was not deemed feasible to also include it in further benchmarks as it would account for a sizable amount of time with the now known result of it likely being last.

3.1.2 Errors

During the benchmarks a few errors have also presented itself. Notably one of the tested executables, Kvick, presented quite a few. For a large portion, about half, Kvick did not produce a result. The algorith will still be included in the results but conclusions cannot be taken in relation the the other algorithms for its lack in successful performance. Other executables did not present any errors and will all be included.

3.2 Theory

3.2.1 Complexity

As described in **??**, a complexity function can be estimated from the data collected from 2.3.2. As described in **??** this is a linear function. By calculating a linear approximation of the source image

size plotted to decompression time $factor$, $base$, and $error$ as described by formula 2.9 can be derived.

A distribution of measured performance and complexity graph estimation for each algorithm included in the test can be found at **??**. Two unique graphs are shown in figure **??**.

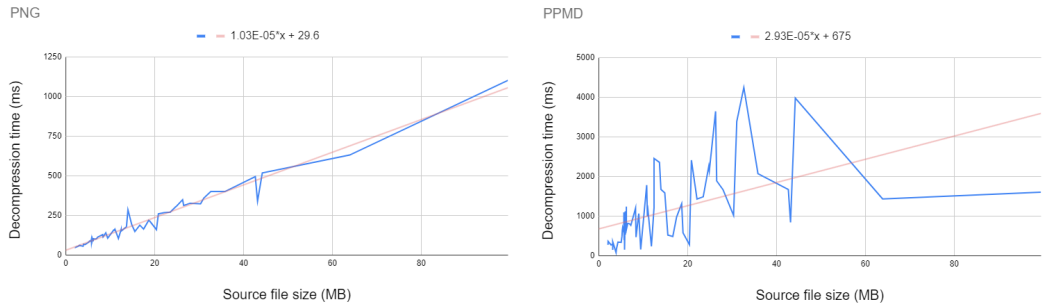


Figure 3.1: Complexity graph. PNG (left) & PPMd (right)

As can be seen, there is a sizable difference between algorithms in terms of their growth and stability. PNG on the left shows a linear growth relatively predictably. PPMd on the right, in comparison, shows a rather steep growth and appears to be fairly unpredictable.

3.2.2 Model

Latency in formula 2.1, as predicted by the complexity model in the previous section, can be substituted. This allows us to plot several graphs and draft a theoretical model to extrapolate this data to a broader spectrum of real world possibilities.

By applying both the complexity and network models the following formula can be derived:

$$T = (size_0 * ratio)/speed + size_0 * factor + base \tag{3.1}$$

While this formula allows plotting a graph, it may be difficult to see any changes in performance. Performance is usually measured against the default value, in this case it is the value as described by formula 2.2. Here, plots show the ratio of T over its default counterpart.

Given a network speed of 650 KB/s, figure 3.2 shows the theoretical performance given a variable source file size. Given a source file of 5 MB, figure 3.3 shows the theoretical performance given a variable network speed.

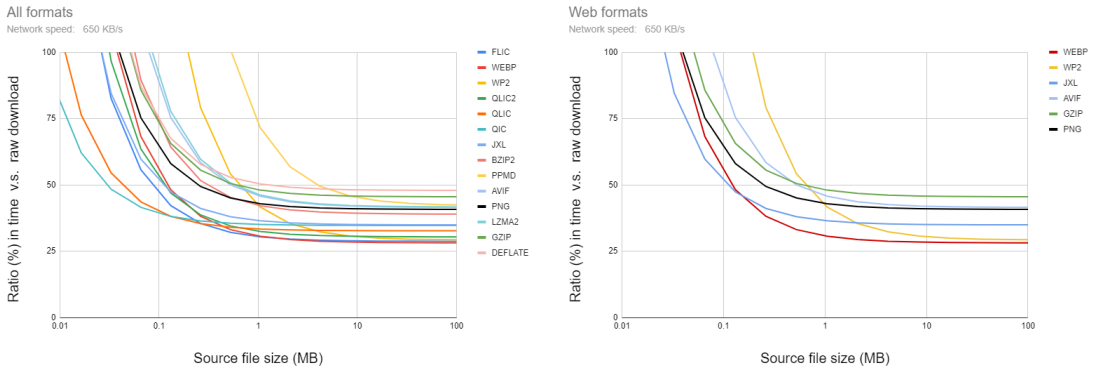


Figure 3.2: Performance model $size \in [10KB, 100MB]$ $speed = 650KB/s$

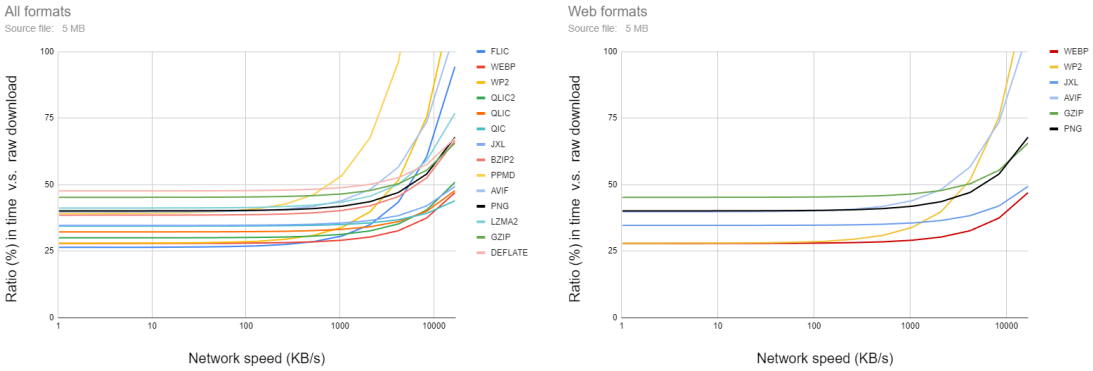


Figure 3.3: Performance model $speed \in [1KB/s, 1.7MB/s]$ $size = 5MB$

3.3 Real data

3.3.1 Measurements

The model put forward in figure 3.2 is theoretical. The benchmark that has been conducted can also be plotted to the same graph. The measurement from a few algorithms are plotted in the following graph.

The first thing to note is that there is no data before a source file of a size smaller than about 2 MB. The reason for this is that no image tested in the benchmark is smaller than this. Secondly the performance is, as expected, heavily dependant on the image and great unpredictability can be noted. However, the plots seem to have a general relative order, WebP finds itself consistently below the others, whereas Gzip lags behind. This is in line with the theoretical model.

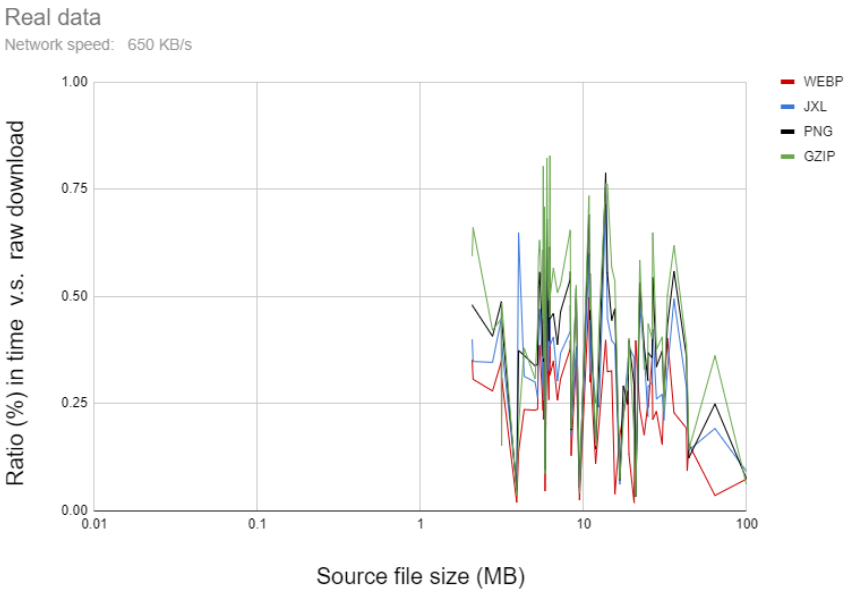


Figure 3.4: Real measurements

3.3.2 Ranking

By averaging all this data out for every algorithm. The charts shown in figure 3.5 can be built up. Separate values are displayed for the download time, the former including decompression time, and the former including decompression time. The right chart in figure 3.5 shows common web formats only.

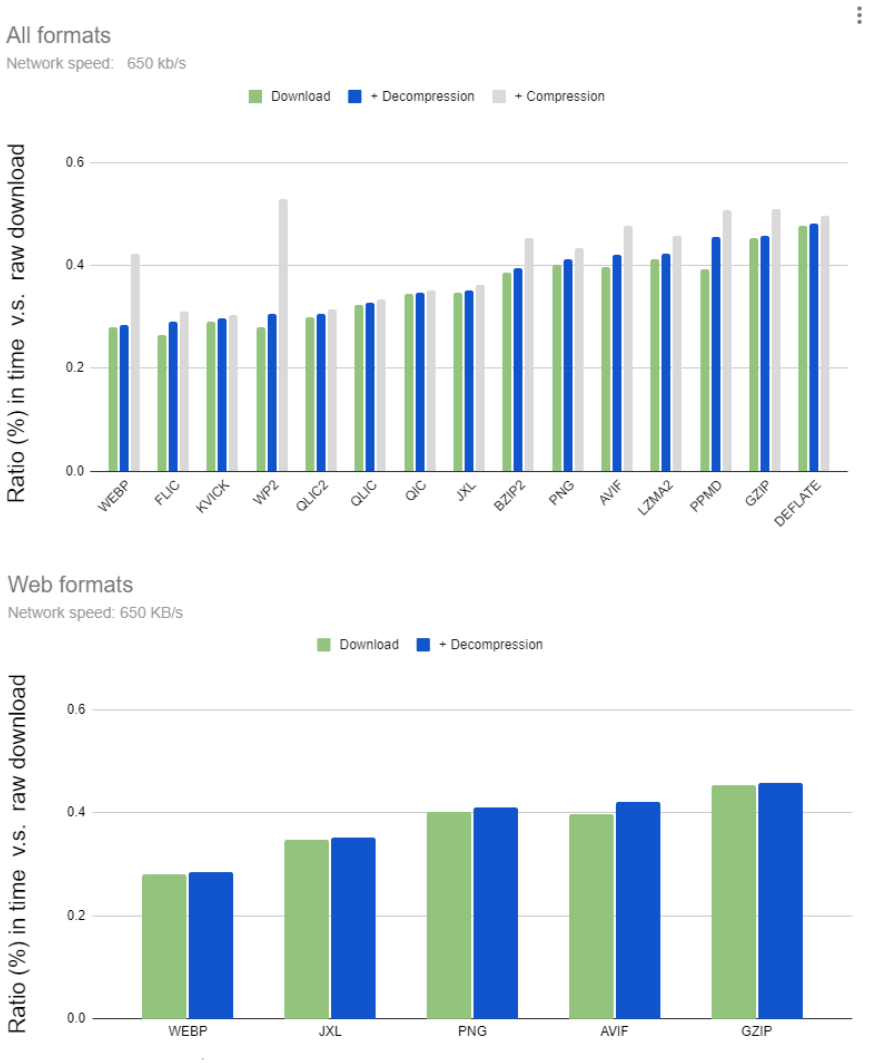


Figure 3.5: Ranking of formats

3.3.3 Effort

The former testing data shows the result for the medium test conducted with a fixed effort level. Some algorithms allow higher effort levels and this may have effects on compression ratio and decompression time. Two values that influence their performance as measured in this paper. The plot in figure 3.6 shows the differences between effort levels for the algorithms that allow it.

The algorithms appear to gain performance when effort levels are increased. However, when compared to the growth in compression time this option becomes rather unappealing as seen in figure 3.7. Note the vertical axis is scaled logarithmically. In this small test, AVIF soared to a compression time of over 300 seconds at its peak.

Disregarding compression times and solely focusing on download and decompression performance allows drafting update charts when applying the improvement measurements to figures 3.2 and 3.3. For each algorithm, the maximum gain in effort is taken and applied the measured values.

The same can be done for the real data in figure 3.5 resulting in figure3.8.

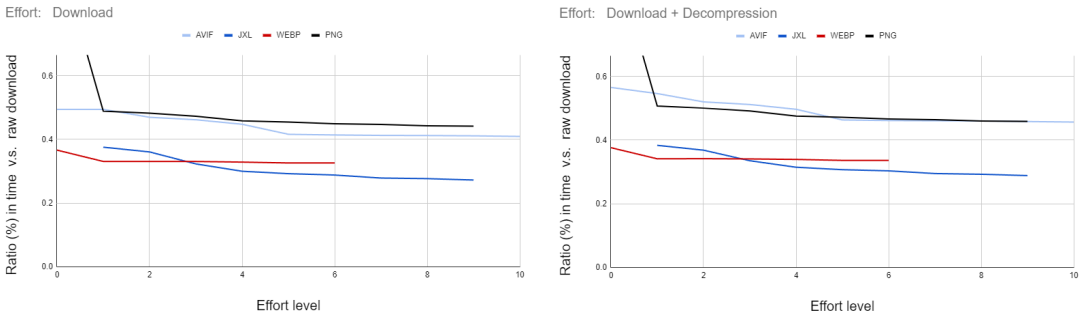


Figure 3.6: Effects of effort

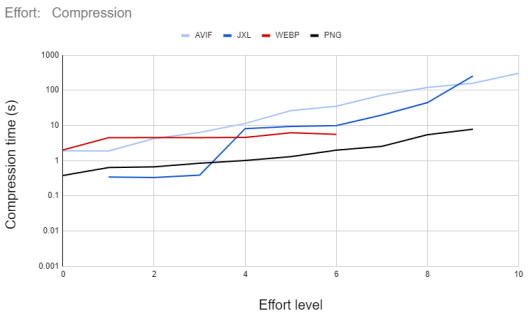


Figure 3.7: Effects of effort on compression time

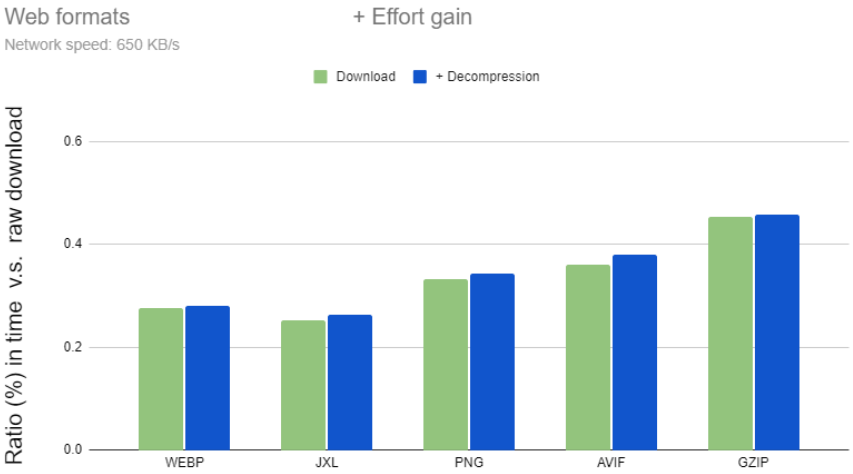


Figure 3.8: Real data with additional effort gains

Chapter 4

Conclusion

4.1 Model

The model put forward in 2.2 were able to show interesting data. However, it has been shown by figure 3.4 that this is a rather unreliable model. While not incorrect, the great discrepancies in real measurements given the same parameters tell that more and well tailored tests should be conducted to fit a use case.

4.2 Real data

A fair amount of testing data has been used to form the chart in figure 3.5. From this graph we can see that WebP consistently performs well, together with the formats Flic and Kvick. Although Kvick had an error rate that was rather high this result cannot be taken as any conclusion. WebP 2 also performed well, falling just behind that of WebP. Since this format is still experimental this is all that can be said about it. Compression times were rather high for both WebP end WebP 2.

After the strong performers we find Jpeg XL. When also accounting for effort as shown in figure 3.8, Jpeg XL managed to become a strong competitor. In this whole study the winner in absolute terms. But as shown in figure 3.7, the compression time needed to achieve this performance is rather costly compared the the other options available.

The first general compression algorithm presents itself after Jpeg XL. PNG roughly shows the same performance as it implements a general compression algorithms as well, this result was to be expected.

AVIF appeared to not perform well in the tests performed. Why this is the case is uncertain. In the back the remaining general algorithms can be found. While still providing improvements over the default, they are as expected not optimal for images.

4.3 Conclusion

Jpeg XL holds the winning title in this study. However, it requires a high level of effort to achieve this. **WebP** has shown balance and provides better performance where compression time is not unlimited. The most common lossless format, PNG, is shown to not be the optimal solution for a fast delivery of images. Furthermore, some niche formats have also shown competitive results.

Bibliography

[Poskanzer & Henderson. 2021] Poskanzer, J., Henderson, B. (2021). Netbpm. *https://netpbm.sourceforge.net*

[Leonard. 2016] Leonard, S. (2016). Windows Image Media Types. RFC 7903. *10.17487/RFC7903*

[Han et al. 2021] Han, J., Li, B., Mukherjee, D., Chiang, C., Grange, A., Chen, C., Su, H., Parker, S., Deng, S., Joshi, U., Chen, Y., Wang, Y., Wilkins, P., Xu, Y. & Bankoski, J. (2021). A Technical Overview of AV1. *10.1109/JPROC.2021.3058584*

[W3 Consortium. 2022] W3 Consortium (W3C). (2022). Portable Network Graphics (PNG) Specification (Third Edition). *https://www.w3.org/TR/png/*

[Rhatushnyak. 2022] Rhatushnyak, A. (2022). Lossless Photo Compression Benchmark. *http://qlic.altervista.org/LPCB.html*

[Anonymous. 2020] Anonymous. (2020). Lossless Image Formats Comparison (Jpeg XL, AVIF, WebP 2, WebP, FLIF, PNG, ...) v1.27. *https://docs.google.com/spreadsheets/d/1ju4q1WkaXT7WoxZINmQpf4ElgMD2VMlqeDN2DuZ6yJ8*

[Wikipedia. 2022] Wikipedia. (2022). Streaming algorithm. Wikimedia Foundation. *https://en.wikipedia.org/wiki/Streaming_algorithm*

[Mozilla Foundation. 2023] Mozilla Foundation. (2023). Image file type and format guide. The MDN Web Docs Project. *https://developer.mozilla.org/docs/Web/Media/Formats/Image_types*

[Mozilla Foundation. 2022] Mozilla Foundation. (2022). Improved JPEG encoder. *https://github.com/mozilla/mozjpeg*

[The libjpeg-turbo Project. 2022] The libjpeg-turbo Project. (2022). SIMD-accelerated libjpeg-compatible JPEG codec library. *https://libjpeg-turbo.org*

[Goolge llc. 2022] Google llc. (2022). WebP. An image format for the Web. *https://developers.google.com/speed/webp*

[Google llc. 2010] Goolge LLC. (2010). Comparative study of WebP, JPEG and JPEG 2000. *https://developers.google.com/speed/webp/docs/c_study*

[AOMedia. 2023] Alliance for Open Media. (2023). libavif - Library for encoding and decoding .avif files. *https://github.com/AOMediaCodec/libavif*

[JPEG. 1992] JPEG. (1992). Information technology — Digital compression and coding of continuous-tone still images *ISO/IEC 10918*

[MSU Media Group. 2020] MSU Media Group. (2020). Data Compression Global Competition. Video Lab. Graphics and Media Lab. MSU Media group. Moscow State University. *https://globalcompetition.compression.ru*

[Pavlov. 2022] Pavlov, I. (2022). 7-Zip & Lempel–Ziv–Markov chain algorithm (LZMA). *https://www.7-zip.org*

[Ziv & Lempel. 1977] Ziv, J., Lempel, A. (1977). A universal algorithm for sequential data compression. *10.1109/TIT.1977.1055714*

[Seward. 2019] Seward, J. (2019). bzip2. *https://sourceware.org/bzip2/*

[Shkarin. 2003] Shkarin, D. (2003). PPM: One step to practicality. Institute for Dynamics of Geospheres. *https://compression.ru/compression.ru/ds/*. *http://compression.graphicon.ru/download/articles/ppm/shkarin_2002dcc_ppmii_pdf.rar*

[FSF. 2020] Free Software Foundation. (2020). GNU Gzip. *https://www.gnu.org/software/gzip/*

[Google llc. 2022] Google llc. (2022). WebP 2: experimental successor of the WebP image format. Git repositories on chromium. *https://chromium.googlesource.com/codecs/libwebp2/*

[JPEG XL Project. 2022] JPEG XL Project. (2022). JPEG XL image format reference implementation. *https://github.com/libjxl/libjxl*

[ImageMagick Studio llc. 1999] ImageMagick Studio LLC. (1999). ImageMagick® is a free and open-source software suite for displaying, converting, and editing raster image and vector image files. *https://imagemagick.org*

[ISO/IEC JTC 1/SC 29. 2022] The Joint Photographic Experts Group (JPEG) committee (ISO/IEC JTC 1/SC 29/WG 1). (2022). Information technology — JPEG XL image coding system. *ISO/IEC 18181*

[Microsoft. 2022] Microsoft. (2022). PowerShell. *https://microsoft.com/powershell*

[w77. 2022] w77. (2022). ImDisk Toolkit: Ramdisk for Windows and mounting of image files. *https://sourceforge.net/projects/imdisk-toolkit/*

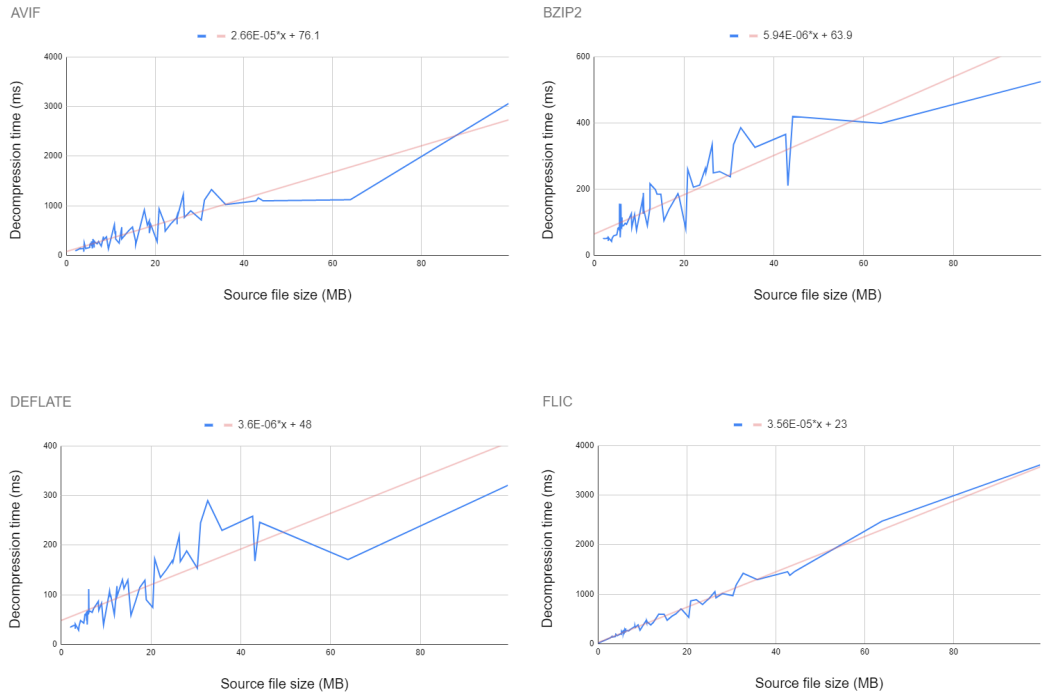
Appendix A

Artifatcs

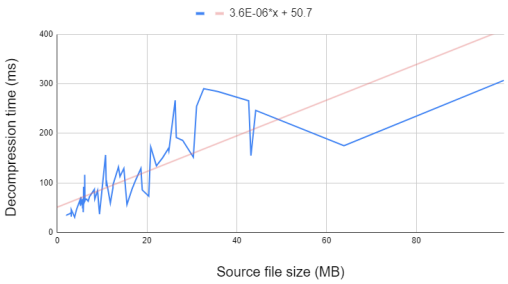
Vrijsen, K. (2023).
Image compression: On-demand systems seeking minimal latency.
https://git.konelinx.be/ImageCompressionODSSML
https://github.com/KoneLinx/ImageCompressionODSSML

Appendix B

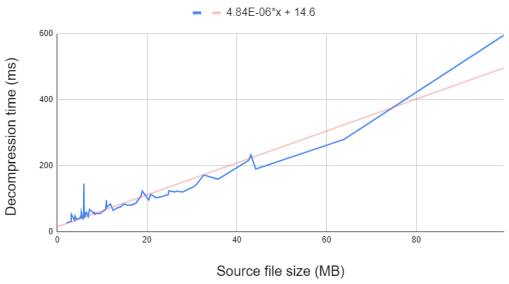
Complexity graphs



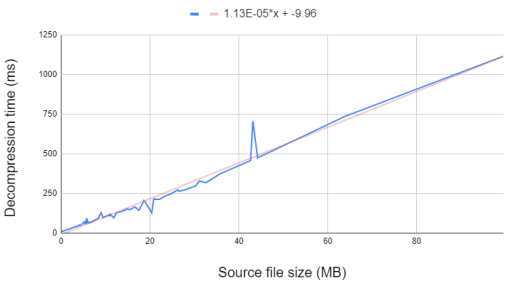
GZIP



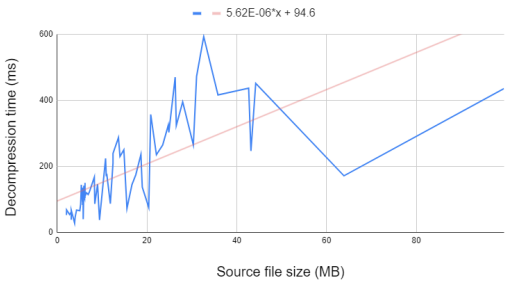
JXL



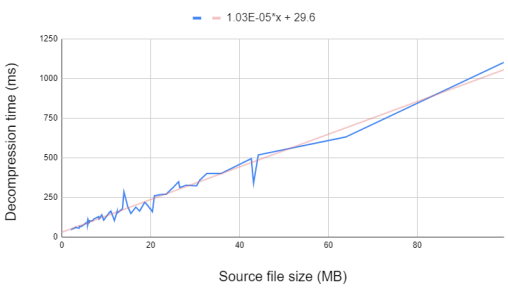
KVICK



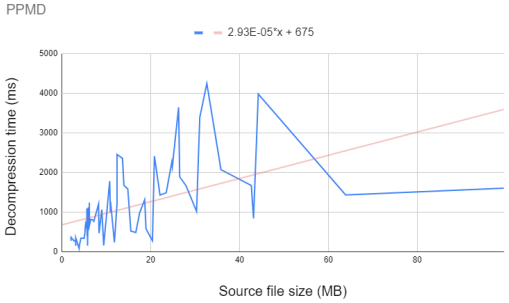
LZMA2



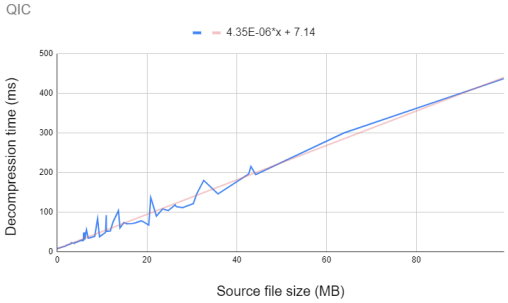
PNG



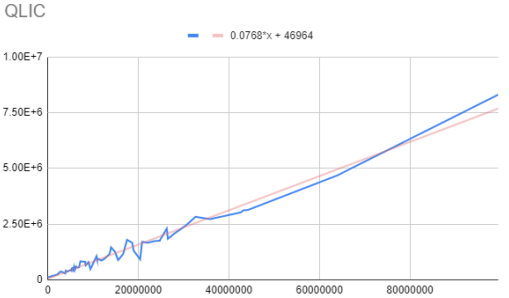
PPMD

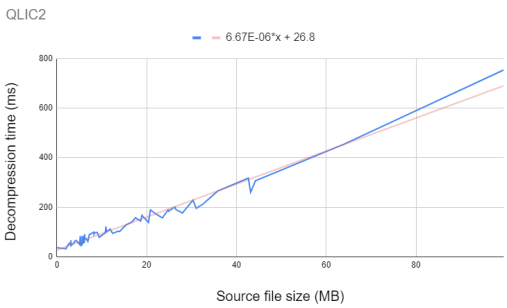


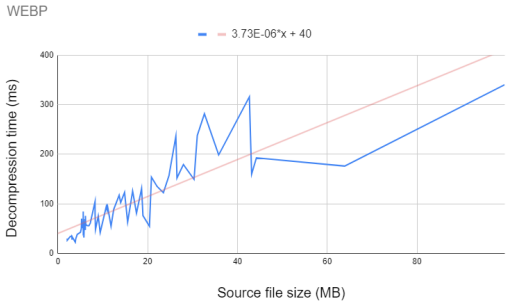
QIC

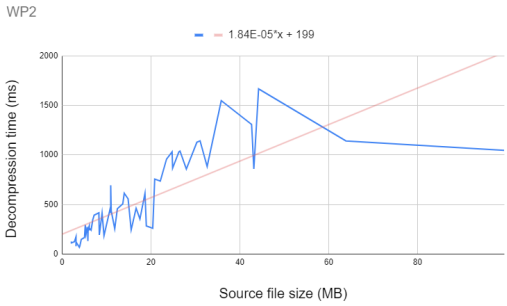


QLIC



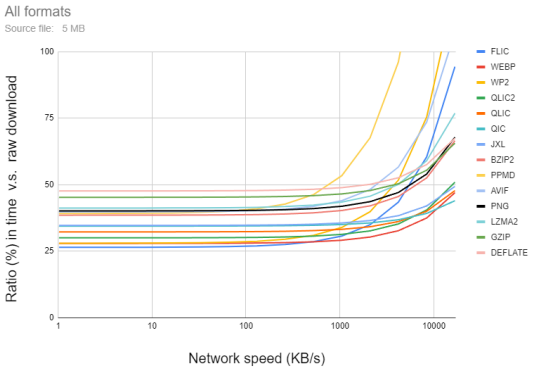

QLIC2
6.67E-06*x + 26.8
Decompression time (ms)
800
600
400
200
0
0
20
40
60
80
Source file size (MB)

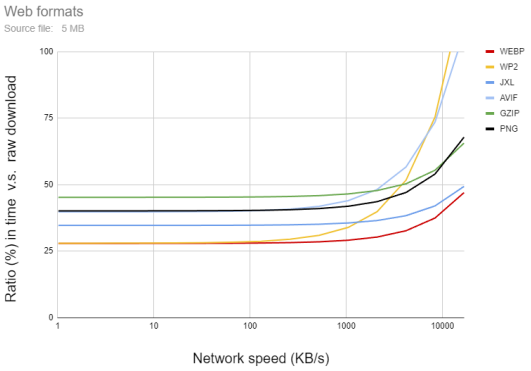

WEBP
3.73E-06*x + 40
Decompression time (ms)
400
300
200
100
0
0
20
40
60
80
Source file size (MB)

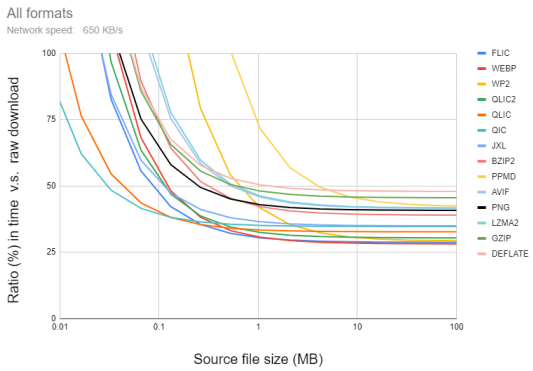

WP2
1.84E-05*x + 199
Decompression time (ms)
2000
1500
1000
500
0
0
20
40
60
80
Source file size (MB)

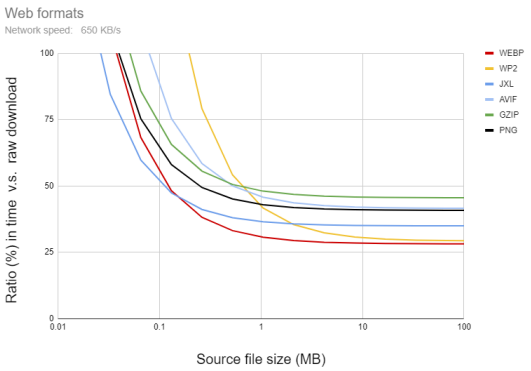
Appendix C

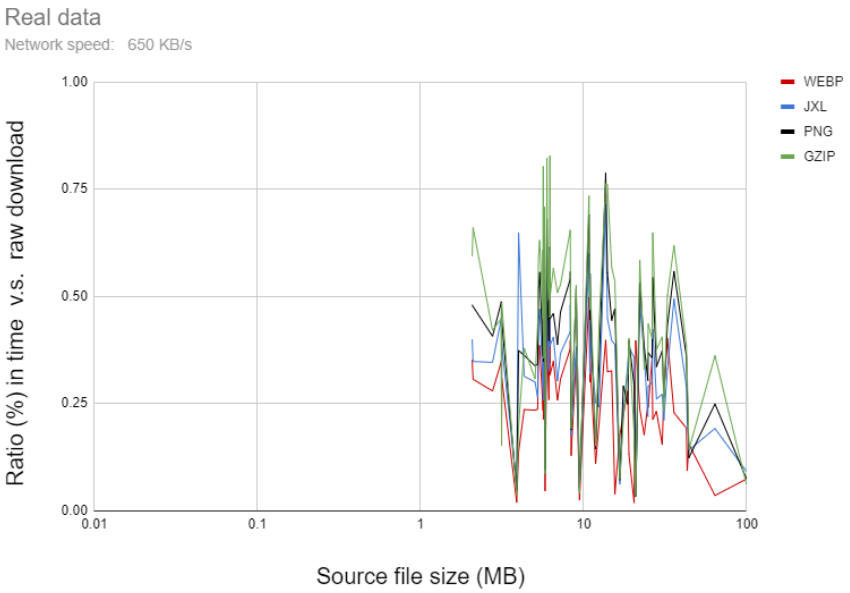
Models

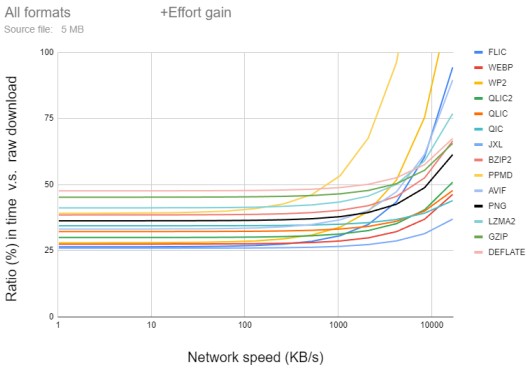


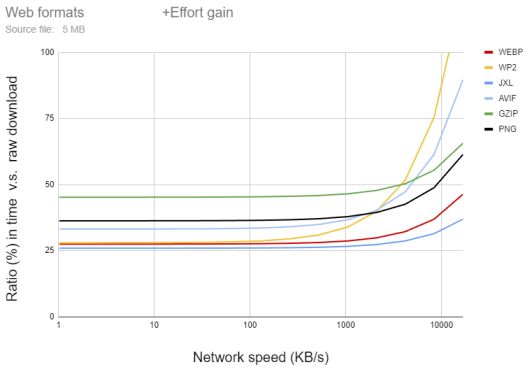


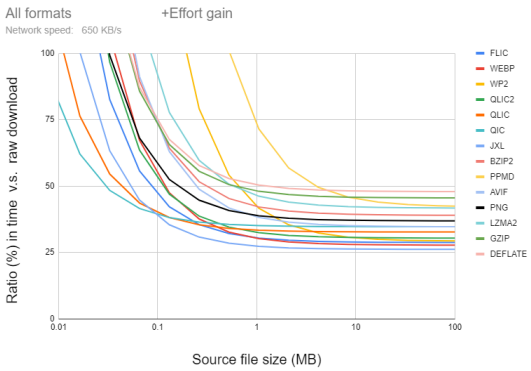


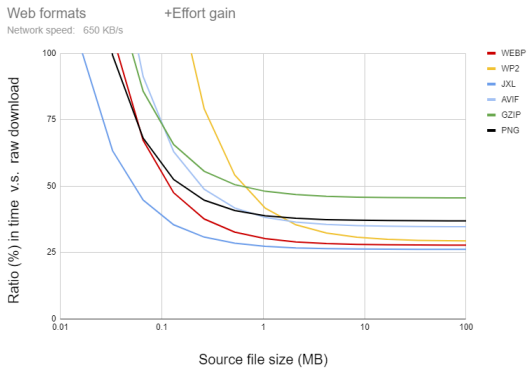



Real data
Network speed: 650 KB/s
Ratio (%) in time v.s. raw download
1.00
0.75
0.50
0.25
0.00
0.01
0.1
1
10
100
Source file size (MB)
WEBP
JXL
PNG
GZIP


All formats
+Effort gain
Source file: 5 MB
Ratio (%) in time v.s. raw download
Network speed (KB/s)
FLIC
WEBP
WP2
QLIC2
QLIC
QIC
JXL
BZIP2
PPMD
AVIF
PNG
LZMA2
GZIP
DEFLATE

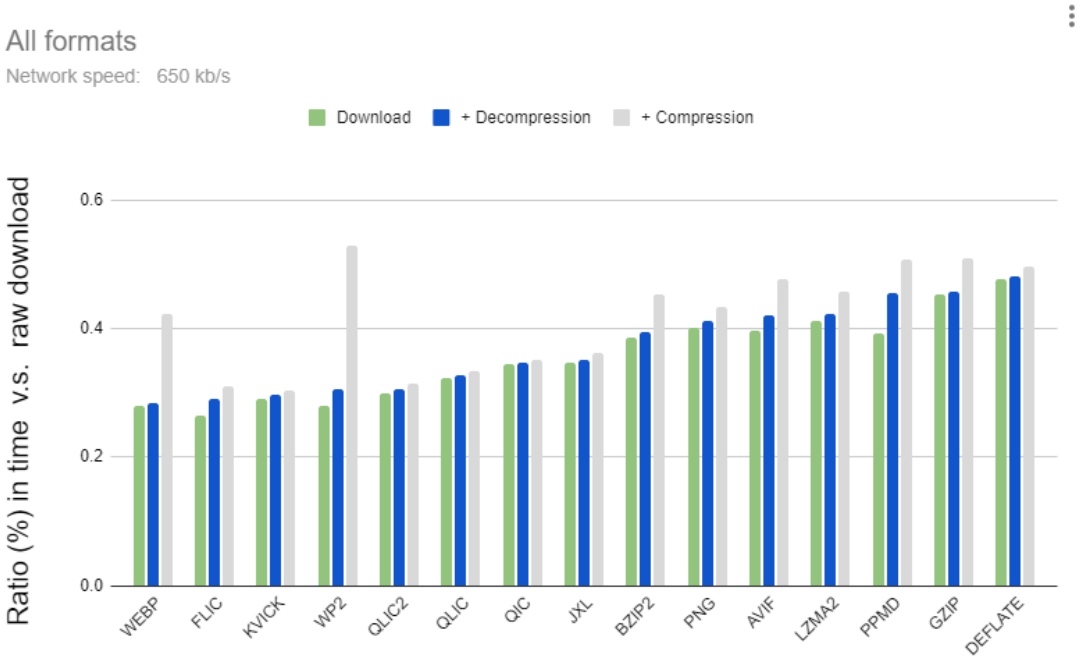

Web formats
+Effort gain
Source file: 5 MB
Ratio (%) in time v.s. raw download
Network speed (KB/s)
WEBP
WP2
JXL
AVIF
GZIP
PNG

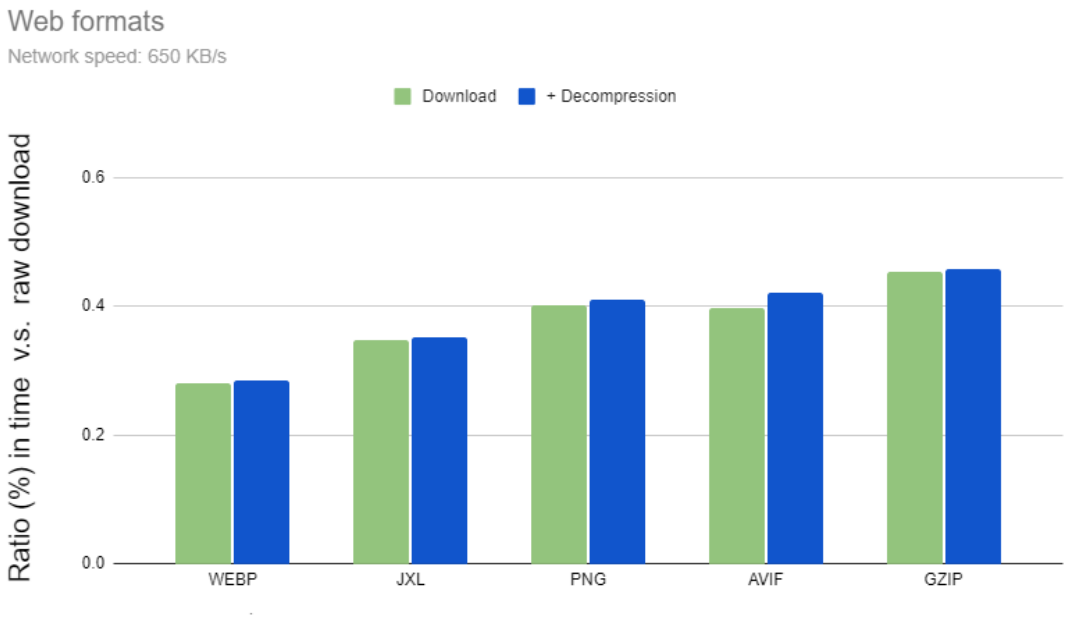

All formats
+Effort gain
Network speed: 650 KB/s
Ratio (%) in time v.s. raw download
Source file size (MB)
FLIC
WEBP
WP2
QLIC2
QLIC
QIC
JXL
BZIP2
PPMD
AVIF
PNG
LZMA2
GZIP
DEFLATE

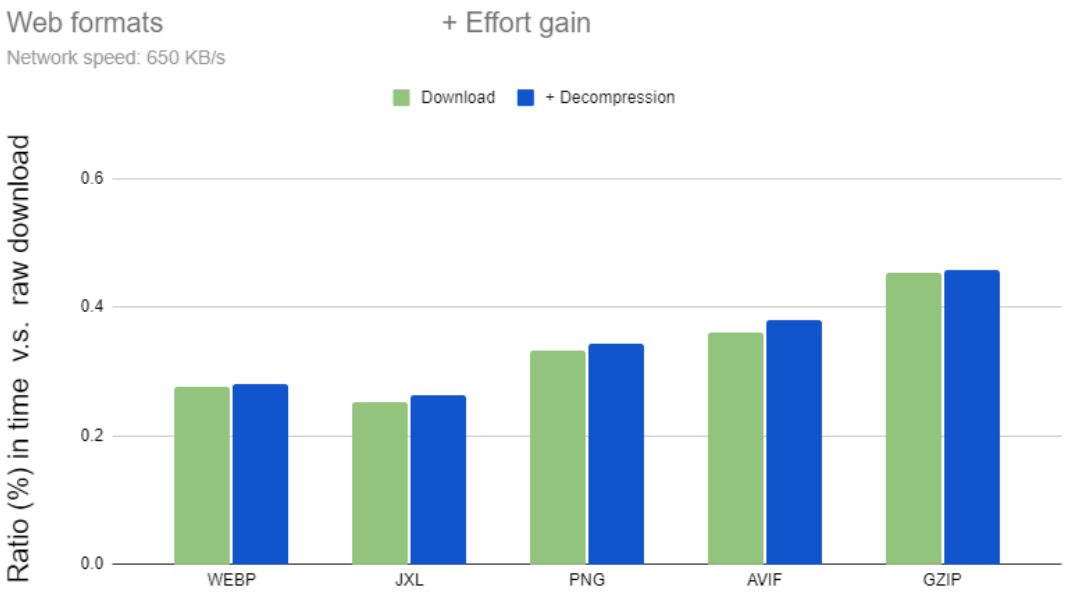

Web formats
+Effort gain
Network speed: 650 KB/s
Ratio (%) in time v.s. raw download
Source file size (MB)
WEBP
WP2
JXL
AVIF
GZIP
PNG

Appendix D

Ranking




Web formats
Network speed: 650 KB/s
Download
+ Decompression
Ratio (%) in time v.s. raw download
0.6
0.4
0.2
0.0
WEBP
JXL
PNG
AVIF
GZIP


Web formats
+ Effort gain
Network speed: 650 KB/s
Download
+ Decompression
Ratio (%) in time v.s. raw download
0.6
0.4
0.2
0.0
WEBP
JXL
PNG
AVIF
GZIP

Appendix E

Scripts

```
function Test-Algorithm {
 param (
  [ValidateScript({ Get-Item $_ })][string]$WorkingDir=".",
  [ValidateRange(1, 1000000000)][int]$RunCount=1,
  [ValidateRange(-1, 11)][int]$Effort=-1,
  [ValidateRange(0, 11)][int]$EffortMin=1,
  [ValidateRange(0, 11)][int]$EffortMax=1,
  [ValidateRange(-1, 100)][int]$Quality=-1,
  [ValidateRange(0, 100)][int]$QualityMin=10,
  [ValidateRange(0, 100)][int]$QualityMax=100,
  [ValidateRange(1, 100)][int]$QualityStep=5,
  [Parameter(Mandatory)][ValidateScript({ Get-Command "Test_$($_)_c"; Get-Command "Test_$($_)_d" })][string]
       $Format,
  [string]$RawExtension=".ppm",
  #[Parameter(Mandatory, ValueFromRemainingArguments)][ValidateScript({ Get-Item $_ })][string[]]$SourcePaths
  [ValidateScript({ Get-Item $_ })][string]$SourcePath
 )
 if ($Effort -ne -1) {
  $EffortMin = $Effort
  $EffortMax = $Effort
 }
 if ($Quality -ne -1) {
  $QualityMin = $Quality
  $QualityMax = $Quality
 }
 $RunTimeTotal = Measure-Command {
  #foreach ($SourcePath in $SourcePaths) {
   $SourceFile = Get-Item "$SourcePath"
   $TmpFilePath = "$($WorkingDir)\source$RawExtension"
   if ($SourceFile.Extension -eq $RawExtension) {
    Copy-Item -Force $SourceFile "$TmpFilePath"
   } else {
    ffmpeg -i "$($SourceFile.FullName)" -compression_level 0 "$TmpFilePath" 2>&1 | Out-Null
   }
   $TmpFile = Get-Item "$TmpFilePath"
   $TmpFileCPath = "$($TmpFile.FullName).tmp"
   $TmpFileDPath = "$($TmpFile.FullName)$RawExtension"
   $OriginDir = Get-Location
   Set-Location $WorkingDir
   $DataFileName = "$($SourceFile.Name).$EffortMin-$EffortMax-1.$QualityMin-$QualityMax-$QualityStep.
        $RunCount"
   $DataFile = New-Item -Force -ItemType File "$DataFileName"
   ## Uncomment for header
   #Set-Content $DataFile "Format,Effort [$EffortMin-$EffortMax],Quality [$QualityMin-$QualityMax],Run,Full
        size (bytes),Compressed size (bytes),Compression time (100ns),Decompression time (100ns)" #,Colour
        difference (red%), Colour difference (green%), Colour difference (blue%)"
   $RunTime = Measure-Command {
    for($e = $EffortMin; $e -le $EffortMax; $e++) {
     for($q = $QualityMax; $q -ge $QualityMin; $q -= $QualityStep) {
      for($run = 1; $run -le $RunCount; $run++) {
       $CTime = Measure-Command {
        & "Test_$($Format)_c" -q $q -e $e -if "$($TmpFile.FullName)" -of "$TmpFileCPath"
       }
       $DTime = Measure-Command {
        & "Test_$($Format)_d" -if "$TmpFileCPath" -of "$TmpFileDPath"
       }
       ## Uncomment to add colour difference tests
```

```
       #$ColourDiff = "$(composite.exe "$($TmpFile.Fullname)" "$TmpFileDPath" -compose difference ppm:- |
            convert.exe - -resize 1x1 -format "%[fx:r],%[fx:g],%[fx:b]" info:-)"
       Add-Content $DataFile "$Format,$e,$q,$run,$($SourceFile.Name),$((Get-Item␣"$TmpFileDPath").Length),$((
            Get-Item␣"$TmpFileCPath").Length),$($CTime.Ticks),$($DTime.Ticks)"#,$ColourDiff"
       Remove-Item "$TmpFileCPath"
       Remove-Item "$TmpFileDPath"
      }
     }
    }
   }
   Remove-Item $TmpFile
   Set-Location $OriginDir
   Move-Item $DataFile ".\TestData.$Format.$($DataFile.Name).$($RunTime.Ticks).$((Get-Date).Ticks).csv"
  #}
 }
}

function Test-AllAllgorithms
{
 param (
  [ValidateScript({ Get-Item $_ })][string]$WorkingDir=".",
  [ValidateRange(1, 1000000000)][int]$RunCount=1,
  [ValidateRange(-1, 11)][int]$Effort=-1,
  [ValidateRange(0, 11)][int]$EffortMin=1,
  [ValidateRange(0, 11)][int]$EffortMax=1,
  [ValidateRange(-1, 100)][int]$Quality=-1,
  [ValidateRange(0, 100)][int]$QualityMin=10,
  [ValidateRange(0, 100)][int]$QualityMax=100,
  [ValidateRange(1, 100)][int]$QualityStep=5,
  [Parameter(Mandatory)][string]$Formats,
  [Parameter(Mandatory, ValueFromPipeline)][ValidateScript({ Get-Item $_ })][string]$SourcePath
 )
 BEGIN {
  $FormatList = $Formats.Split("␣")
 }
 PROCESS {
  foreach ($Fmt in $FormatList) {
   $Ext = ".ppm"
   if ($Fmt.Contains(":"))
   {
    $Fmt = $Fmt.split(":")
    $Ext = $Fmt.get(1)
    $Fmt = $Fmt.get(0)
   }
   Test-Algorithm -Format $Fmt -RawExtension $Ext -WorkingDir $WorkingDir -SourcePath $SourcePath ‘
   -RunCount $RunCount ‘
   -Effort $Effort -EffortMin $EffortMin -EffortMax $EffortMax ‘
   -Quality $Quality -QualityMin $QualityMin -QualityMax $QualityMax -QualityStep $QualityStep
  }
 }
}

# Example algorithm
# Define functions; e.g. JXL
function Test_JXL_c ($e,$q,$if,$of)
{
 cjxl -q $q -e $e "$if" "$of"
}
function Test_JXL_d ($if,$of)
{
 djxl "$if" "$of.ppm"
 Rename-Item "$of.ppm" "$of"
}

# Run benchmark example
# Preferably run on a RAM disk.
(Get-ChildItem $SourceDirectory | Where-Object -Property Extension -EQ .ppm).FullName | Test-AllAllgorithms -
     Formats "PNG␣WEBP:.bpm␣AVIF:.png␣JXL␣WP2" -Effort 0 -Quality 100 -WorkingDir T: -RunCount 3
```

Kobe Vrijsen

January 24, 2023

kobevrijsen@posteo.be

konelinx.be