Turning C++ into Rust:

Learnings from Porting Image Format Code on a Weekend

Benedikt Mandelkow **13.10.2020** **Rust-Saar Meetup 4u16**

whoami

computer science student

- currently focussing on computer graphics and image processing at university
- programming language enthusiast
  - active rust user for about 2 years

timeline

- png
- compression
  - DIY
  - zstd
- zpng
  - build/ libraries
  - zpng_rs
  - comparison
    - serialisation, aliasing, out of memory, iterators
  - cpp+rust

png
overview

- header which identifies format and then repeated chunks e.g. width and height and pixel data
- uses DEFLATE compression

The IHDR chunk must appear FIRST. It contains:

```
Width:              4 bytes
Height:             4 bytes
Bit depth:          1 byte
Color type:         1 byte
Compression method: 1 byte
Filter method:      1 byte
Interlace method:   1 byte
```

http://www.libpng.org/pub/png/spec/1.2/PNG-Chunks.html

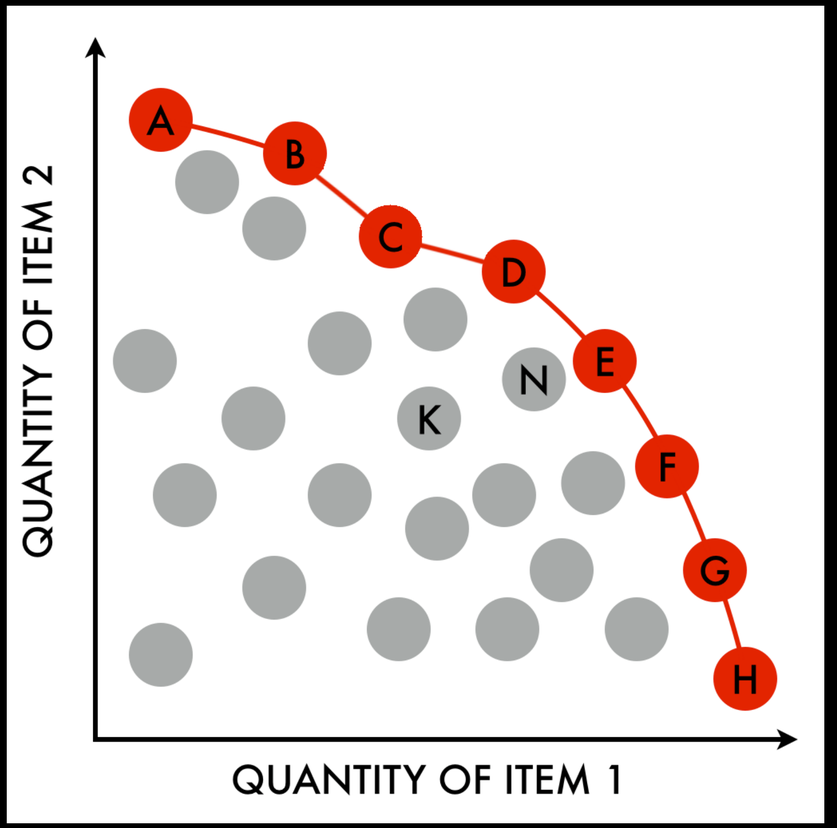
- *https://tools.ietf.org/html/rfc2083*
- *https://www.hackerfactor.com/blog/index.php?/archives/252-PNG-and-Cameras.html*
- *https://www.hackerfactor.com/blog/index.php?/archives/894-PNG-and-Hidden-Pixels.html*

compression

- lossless vs lossy
- try yourself:
  - Run Length Encoding
  - LZW
  - Huffman *(https://fasterthanli.me/articles/huffman-101)*

zstd
properties

- from Yann Collet, previously did LZ4
  - blog: *https://fastcompression.blogspot.com/*
- license: BSD + GPLv2
- performance:
  - „Zstandard reaches the current Pareto frontier, as it decompresses faster than any other currently-available algorithm with similar or better compression ratio.“ *https://en.wikipedia.org/wiki/Zstandard*



https://en.wikipedia.org/wiki/Pareto_efficiency#/media/File:Pareto_Efficient_Frontier_1024x1024.png

zstd
usage

- Arch: „Recompressing all packages to zstd with our options yields a total ~0.8% increase in package size on all of our packages combined, but the decompression time for all packages saw a ~1300% speedup.“ *https://www.archlinux.org/news/now-using-zstandard-instead-of-xz-for-package-compression/*

- Linux: „With Linux 5.9 comes the ability to compress the Linux kernel image / initrd with Zstd for yielding faster boot speeds but at a compression ratio between Gzip and XZ/LZMA.“ *https://www.phoronix.com/scan.php?page=news_item&px=Zstd-Firmware-Compress-Patch*

zpng

https://github.com/catid/Zpng

Zpng

Small experimental lossless photographic image compression library with a C API and command-line interface.

It's much faster than PNG and compresses better for photographic images. This compressor often takes less than 6% of the time of a PNG compressor and produces a file that is 66% of the size. It was written in just 500 lines of C code thanks to Facebook's Zstd library.

The goal was to see if I could create a better lossless compressor than PNG in just one evening (a few hours) using Zstd and some past experience writing my GCIF library. Zstd is magical.

I'm not expecting anyone else to use this, but feel free if you need some fast compression in just a few hundred lines of C code.

Thanks to the author: Christopher A. Taylor

How to build

master Zpng / apps / thirdparty /

catid Fix header include. Remove extra image test files

..

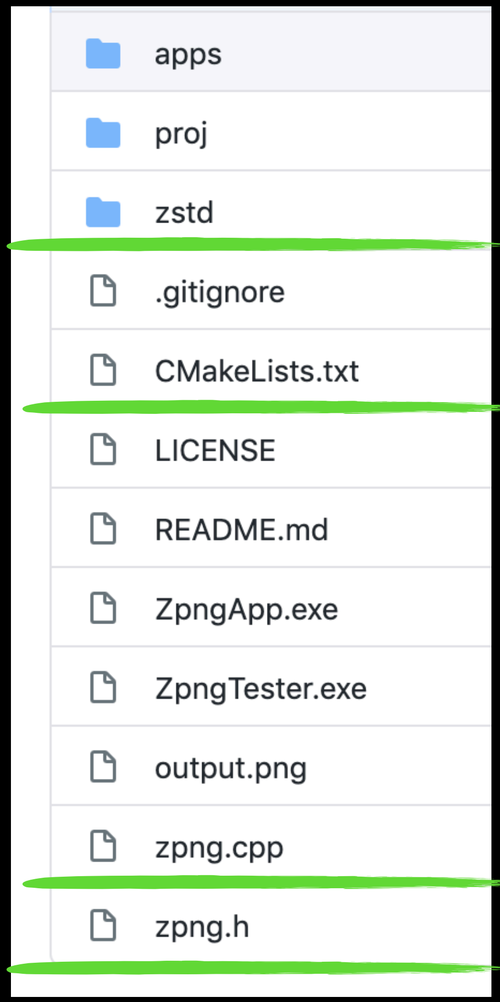
stb_image.h

stb_image_write.h

https://wiki.alopex.li/LetsBeRealAboutDependencies

```
$ otool -L Zpng/build/zpng
Zpng/build/zpng:
    /usr/local/opt/llvm/lib/libc++.1.dylib [..]
    /usr/lib/libSystem.B.dylib [..]

$ cargo tree | wc -l
106
```

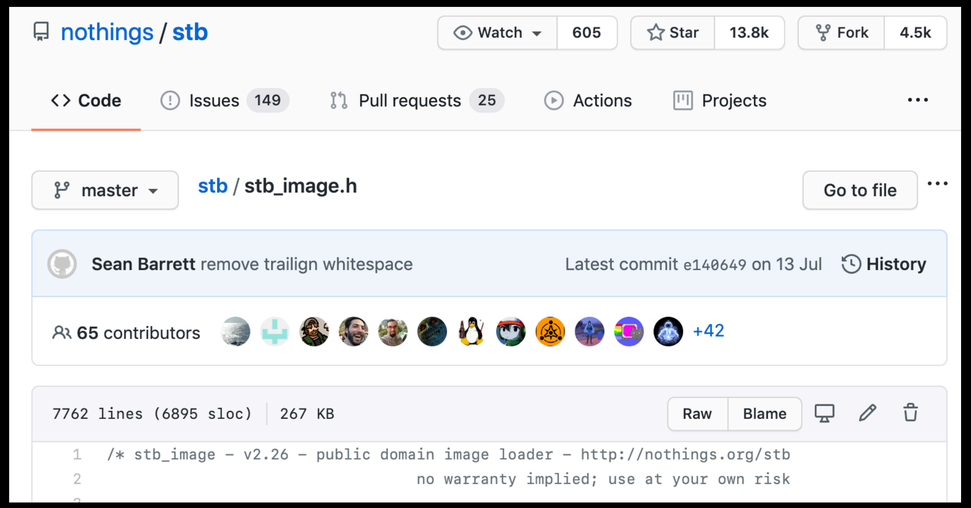


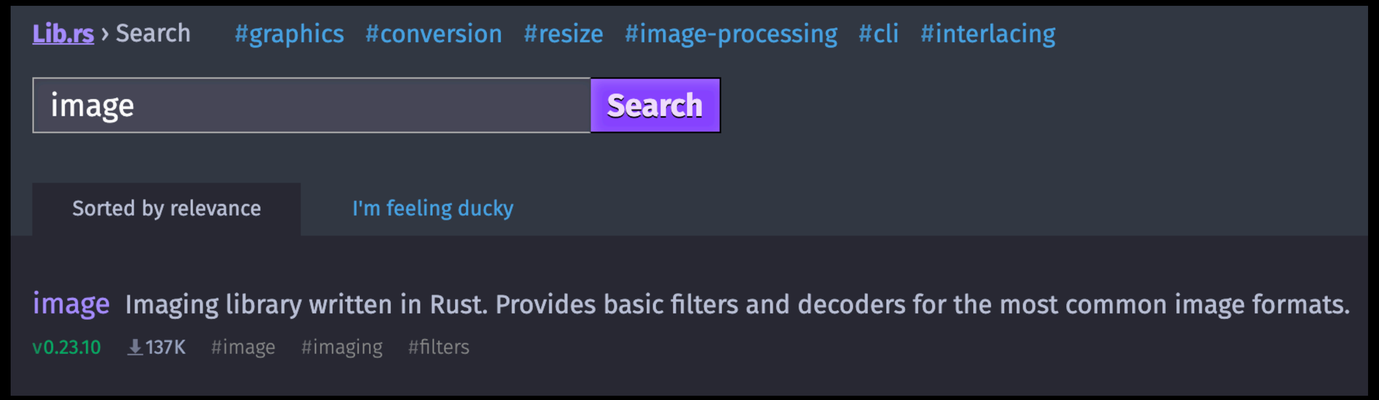

```
i~/Downloads
λ git clone https://github.com/catid/Zpng                                    10ms
Cloning into 'Zpng'...
remote: Enumerating objects: 83, done.
remote: Total 83 (delta 0), reused 0 (delta 0), pack-reused 83
Unpacking objects: 100% (83/83), 700.38 KiB | 1.85 MiB/s, done.
i~/Downloads
λ cd Zpng/ && mkdir build && cd build                                    1s 466ms
i~/D/Z/build
λ  (master|✓ )  cmake .. -GNinja                                            38ms
-- The C compiler identification is AppleClang 10.0.1.10010046
-- The CXX compiler identification is AppleClang 10.0.1.10010046
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/clang - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/benediktmandelkow/Downloads/Zpng/build
i~/D/Z/build
λ  (master|…14) ninja                                                       3s 557ms
[22/27] Building CXX object CMakeFiles/unit_test.dir/apps/zpng_test.cpp.o
../apps/zpng_test.cpp:281:43: warning: comparison of integers of different signs: 'unsigned int' and 'int' [-Wsign-compare]
                decompressResult.Channels != comp ||
                ~~~~~~~~~~~~~~~~~~~~~~~~~ ^  ~~~~
../apps/zpng_test.cpp:282:47: warning: comparison of integers of different signs: 'unsigned int' and 'int' [-Wsign-compare]
                decompressResult.HeightPixels != y ||
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ^  ~
../apps/zpng_test.cpp:283:46: warning: comparison of integers of different signs: 'unsigned int' and 'int' [-Wsign-compare]
                decompressResult.WidthPixels != x ||
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ^  ~
../apps/zpng_test.cpp:284:46: warning: comparison of integers of different signs: 'unsigned int' and 'int' [-Wsign-compare]
                decompressResult.StrideBytes != x * comp ||
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ^  ~~~~~~~~
4 warnings generated.
[27/27] Linking CXX executable unit_test
i~/D/Z/build
λ  (master|…18)                                                             12s 659ms
```

libraries

images

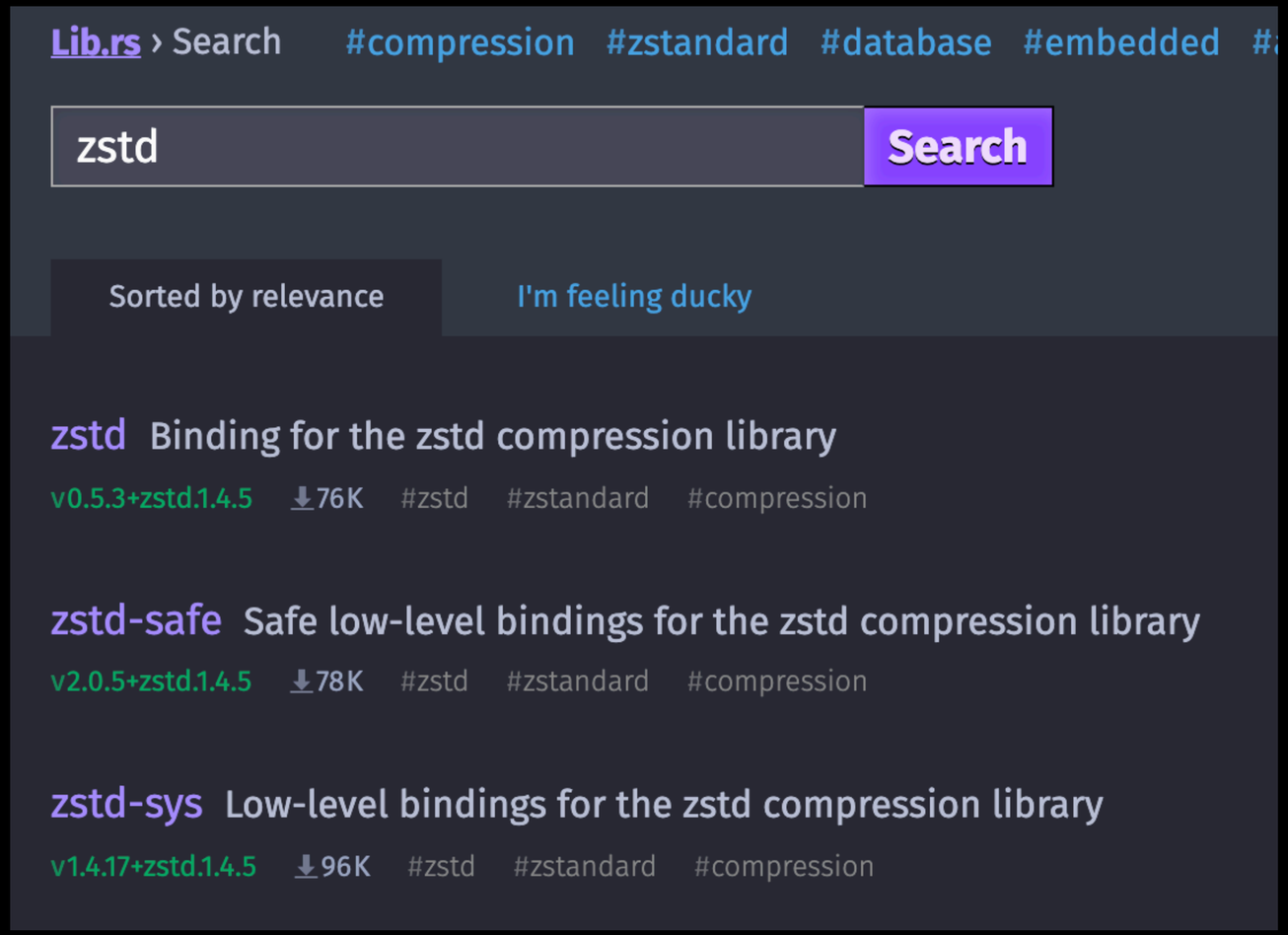
- stb image
- image





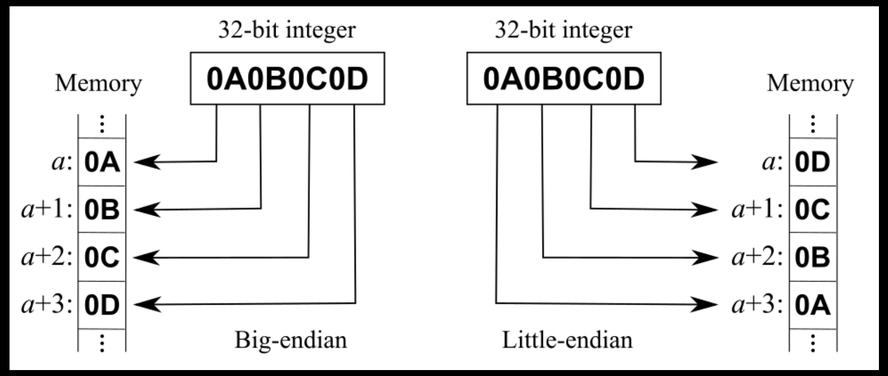
lib.rs

zstd


Lib.rs › Search
#compression
#zstandard
#database
#embedded
zstd
Search
Sorted by relevance
I'm feeling ducky
zstd Binding for the zstd compression library
v0.5.3+zstd.1.4.5 ⬇76K #zstd #zstandard #compression
zstd-safe Safe low-level bindings for the zstd compression library
v2.0.5+zstd.1.4.5 ⬇78K #zstd #zstandard #compression
zstd-sys Low-level bindings for the zstd compression library
v1.4.17+zstd.1.4.5 ⬇96K #zstd #zstandard #compression

endianness

image: *https://en.wikipedia.org/wiki/Endianness*




```rust
header.extend_from_slice(&u16::to_le_bytes(ZPNG_HEADER_MAGIC));
header.extend_from_slice(&u16::to_le_bytes(imageData.WidthPixels));
header.extend_from_slice(&u16::to_le_bytes(imageData.HeightPixels));
header.extend_from_slice(&u8::to_le_bytes(imageData.Channels));
header.extend_from_slice(&u8::to_le_bytes(imageData.BytesPerChannel));

debug_assert_eq!(&header.len(), &(ZPNG_HEADER_OVERHEAD_BYTES as usize));
```

```cpp
// original code

ZPNG_Header* header = (ZPNG_Header*)output;
header->Magic = ZPNG_HEADER_MAGIC;
header->Width = (uint16_t)imageData->WidthPixels;
header->Height = (uint16_t)imageData->HeightPixels;
header->Channels = (uint8_t)imageData->Channels;
header->BytesPerChannel = (uint8_t)imageData->BytesPerChannel;
```

- rust needs: `let a: u8 = d.wrapping_add(prev[i]);`
- in cpp unsigned integer overflow is defined behaviour
  https://stackoverflow.com/a/16188846

aliasing

c++

"Yes, incrementing 32-bit integers in a std::vector is up to eight times faster than incrementing 8-bits on a popular compiler with common optimization settings.“

```
void vector8_inc(std::vector<uint8_t>& v) {
    for (size_t i = 0; i < v.size(); i++) {
        v[i]++;
    }
}

void vector32_inc(std::vector<uint32_t>& v) {
    for (size_t i = 0; i < v.size(); i++) {
        v[i]++;
    }
}
```

„[…] it could in theory change the value of size(), because uint8_t is a typedef for unsigned char, and unsigned char (and char) arrays are allowed to alias any type. […] **So the compiler assumes that every increment v[i]++ potentially modifies size() and hence must recalculate it every iteration.“**

from *https://travisdowns.github.io/blog/2019/08/26/vector-inc.html*

more: A Guide to Undefined Behavior in C and C++, Part 1 *https://blog.regehr.org/archives/213* and *https://blog.regehr.org/archives/959*

aliasing
rust

- rust is more restrictive than c++, more similar to fortran
- „In Rust, this optimization should be sound. **For almost any other language, it wouldn't be** (barring global analysis). This is because the **optimization relies on knowing that aliasing doesn't occur,** which most languages are fairly liberal with. Specifically, we need to worry about function arguments that make input and output overlap, such as compute(&x, &mut x).“ *https://doc.rust-lang.org/nomicon/aliasing.html*
- the optimisations are not yet enabled due to an llvm bug
  - `-Zmutable-alias=no`
  - *https://github.com/rust-lang/rust/issues/54878*

what can be improved?

out of memory (OOM)

```rust
// Stage 1: Decompress back to packing buffer
let mut packing = vec![0; byteCount];
```

```c
// handling in original code:

// Space for packing
packing = (uint8_t*)calloc(1, byteCount);

if (!packing) {
    goto ReturnResult;
}
```

> On the whole, living with the possibility of OOM is easier and less troublesome.
>
> Written on 08 May 2019.

https://utcc.utoronto.ca/~cks/space/blog/linux/StrictOvercommitCanOOM

> Rust in general encourages to *abort* on out-of-memory issues, while this is a big *nono* when the code is used in a system library (such as curl).

https://daniel.haxx.se/blog/2020/10/09/rust-in-curl-with-hyper/

Maybe relevant:

https://github.com/dtolnay/no-panic

> Still other folks resort to more than one of these techniques, and that's pretty much my preference: have the app try to be aware of itself and throttle back if at all possible. Then have it kill itself if it runs away. Then if that doesn't work, have some kind of "container" limit, and finally, if that fails, the whole system has a global limit.
>
> It's all terrible, naturally, but this is what you get when you build on top of systems that assume every single resource will always be available. It doesn't matter whether it's memory, disk, network, or something else entirely. Given enough time and permutations, something will disappear out from under you, and if you have no way to notice it and handle it clearly (perhaps due to terrible layering schemes), the only possible outcome is a mess.
>
> In retrospect, it's amazing some of this stuff works at all.

https://rachelbythebay.com/w/2018/03/31/limit/

your code is not functional enough!!1!

can we reduce bounds checks while still avoiding repeated allocations?

```rust
fn PackAndFilter<const kChannels: usize>(
    input: &[u8],
    width: u16,
    _height: u16,
    _byteCount: usize,
) -> Vec<u8> {
    input
        .chunks(width as usize * kChannels as usize)
        .map(|row| {
            row.chunks(kChannels)
                .scan([0; kChannels], |prev, channel| {
                    Some(prev.iter_mut().zip(channel).map(|(prev_i, curr)| {
                        let d = curr.wrapping_sub(*prev_i);
                        *prev_i = *curr;
                        d
                    }))
                })
                .flatten()
        })
        .flatten()
        .collect::<Vec<u8>>()
}
```

for testing: *https://godbolt.org/*

TODO
possible improvements/ experiments

- do performance measurements/ optimisations
  - cargo flamegraph *https://github.com/flamegraph-rs/flamegraph*
    - *https://github.com/jonhoo/inferno*
  - *https://sled.rs/perf.html*
    - „performance happens naturally when engineers love the codebase and they are aware of which parts of the system can be sped up“
  - *https://github.com/plasma-umass/coz*
- compile to wasm for a web app
- *different filtering approaches*
  - *https://github.com/catid/Zpng/issues/2*
- plot tradeoff between zstd compression settings (time and size) for a representative image dataset
- handle allocations better, size specified by input
- add fuzzing *https://github.com/rust-fuzz*

where is the perfect image format I expected you to present?

- maybe take a look at the upcoming
  - AV1
  - AVIF
- finally friendly licensing for state of the art codecs
- *https://github.com/leandromoreira/digital_video_introduction*
- *https://aomediacodec.github.io/av1-avif/*

Thanks

https://github.com/benmkw/zpng_rs