**QOIv2 – THE QUITE OK IMAGE FORMAT**

Specification **Version 2.0**, 2022-04-30 – **Unknown6656**, based on the previous QOI format

published by Dominic Szablewski on <https://qoiformat.org/qoi-specification.pdf>

The Version 2.0 of the QOI format (henceforth only referred to as “v2”) is compatible with Version 1.0 (“v1”), which can be found at <https://qoiformat.org/qoi-specification.pdf>.

A QOI file consists of a 14-byte header, followed by any number of data “chunks” and an optional 8-byte end marker. The 8-byte end marker is only required for QOIv1. The header has the same structure for QOIv1 and QOIv2 and is described as follows:

**struct qoi\_header {**

**uint8\_t magic[3]; // "qoi"**

**uint8\_t version; // "f" (0x66) for version 1.0**

**// "2" (0x32) for version 2.0**

**uint32\_t width; // image width in pixels**

**uint32\_t height; // image height in pixels**

**uint8\_t channels; // 3 = RGB, 4 = RGBA**

**uint8\_t colorspace; // 0 = sRGB with linear alpha**

**// 1 = all channels linear**

**};**

The header’s magic number is the constant **3-character ASCII8** string “**qoi**”. It is followed by the **8-bit** version field, which encodes the file’s version number. A value of “**f**” (**0x66**) indicates v1 of the QOI specification. All subsequent bytes are decoded and encoded as specified in <https://qoiformat.org/qoi-specification.pdf>. A value of “**2**” (**0x32**) indicates v2 of the QOI specification.

The QOI header’s **colorspace** and **channels** fields are purely informative and do not change the way data chunks are encoded.

This document assumes from now on that the file conforms the v2 specification and will be encoded/decoded as such. Furthermore, this document assumes that all pixel values are encoded as 32-bit non-premultiplied RGBA structures of the following form (unless noted otherwise):

**struct rgba\_pixel {**

**uint8\_t r; // the red color channel [0..255]**

**uint8\_t g; // the green color channel [0..255]**

**uint8\_t b; // the blue color channel [0..255]**

**uint8\_t a; // the alpha channel [0..255]**

**};**

A RGBA pixel value of **01234567** would therefore represent **{r=0x01, g=0x23, b=0x45, a=0x67}**, which corresponds to the float32 color vector **(.0039, .1373, .2706, .4039)** or the uint8 color vector **(1, 35, 69, 103)**.

Images are encoded row by row, left to right, top to bottom in a row-major order. The decoder and encoder initialize the following tracking variables:

* the previous RGBA pixel value (“**prev**”)
* the RGBA pixel value preceding the previous one (“**pprev**”)
* an empty indexing (running) cache with a capacity of **64** RGBA pixel values (“**cache**”)
* an empty palette with a capacity of **16** RGBA pixel values (“**palette**”)

The variables **prev** and **pprev** are indexed with the RGBA value of **000000ff**, which represents the color **opaque black**. All indices of **cache[]** and **palette[]** are initialized with **00000000**, which represents the color **transparent**.

An image is complete when all pixels specified by **width \* height** have been covered. If an QOI encoder issues a sequence of chunks which cannot fully encode the image, all remaining pixels should be assumed to be **transparent** (**rgba=00000000**). Pixels are encoded as chunks. A chunk has a variable binary length, which is divisable by 8, meaning that a chunk is always byte-aligned. This does **not** mean that the chunk’s underlying fields must be byte-aligned. A chunk can –but is not forced to- encode multiple pixels at once.

The version 2.0 of the QOIF image format specifies the following chunk types:

* a repetition of the previous pixel value **prev**
* a repetition of the pixel value **pprev**
* a color as indexed by **chache**
* a color as indexed by **palette**
* a pixel value specified by the individual channel values for **r,g,b** or **r,g,b,a**
* a 3x 2-bit color difference in regards to **prev**
* a 1-channel color difference in regards to **prev**
* a 2-channel color difference in regards to **prev**
* a luma difference in regards to **prev**
* a luma difference in regards to the pixel value neighbouring to the top of the currently processed pixel
* an average pixel value computed based on neighbouring pixel values
* a difference in hue in regards to **prev**

The following overview describes the binary layout of all chunk types defined in version 2.0 of the QOIF specification:

**.-------------------------------.-------------------------------.-------------------------------.-------------.-------------.**

**CHUNK START <| byte[0] | byte[1] | byte[2] | byte[3] | byte[4] |>**

**<| 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 ... 1 0 | 7 6 ... 1 0 |>**

**'-------------------------------'-------------------------------'-------------------------------'-------------'-------------'**

**+-------+-----------------------+ ' ' ' '**

**OP\_INDEXED | 0 0 | CACHE INDEX | ' ' ' '**

**+-------+-------+-------+-------+ ' ' ' '**

**OP\_2BIT\_DIFF | 0 1 | Δ R | Δ G | Δ B | ' ' ' '**

**+-------+---+---+-------+-------+-----------+-------------------+ ' ' '**

**OP\_2CHN\_DIFF | 1 0 0 | SEL\_CHN\_2 | Δ CHANNEL 1 | Δ CHANNEL 2 | ' ' '**

**+-----------+-----------+-------+-------+---+-------------------+ ' ' '**

**OP\_1CHN\_DIFF | 1 0 0 1 1 0 0 0 |SEL\_CHN| Δ CHANNEL | ' ' '**

**+-------------------------------+-------+-----------------------+-------------------------------+-------------+ '**

**OP\_RGB | 1 0 0 1 1 0 0 1 | RED CHANNEL VALUE | GREEN CHANNEL VALUE |BLUE CHANNEL | '**

**+---------------------------+---+-------------------------------+-------------------------------+-------------+-------------+**

**OP\_RGBA | 1 0 0 1 1 0 1 |PAL| RED CHANNEL VALUE | GREEN CHANNEL VALUE |BLUE CHANNEL |ALPHA CHANNEL|**

**+-----------------------+---+---+-------+-----------+-----------+-------------------------------+-------------+-------------+**

**OP\_LUMA\_TOP | 1 0 0 1 1 1 | Δ G | ΔR - ΔG | ΔB - ΔG | ' ' '**

**+---------------+-------+-------+-------+-----------+-----------+ ' ' '**

**OP\_PALETTE | 1 0 1 0 | PALETTE INDEX | ' ' ' '**

**+---------------+---+-----------+ ' ' ' '**

**OP\_RUN\_PREV | 1 0 1 1 |PP?| RUN COUNT | ' ' ' '**

**+-----------+---+---+-----------+---------------+---------------+ ' ' '**

**OP\_LUMA\_DIFF | 1 1 0 | Δ G | ΔR – ΔG | ΔB – ΔG | ' ' '**

**+-----------+-----------+---+---+---------------+---------------+ ' ' '**

**OP\_AVG | 1 1 1 0 0 0 |PP?|TL?| ' ' ' '**

**+-----------------------+---+---+ ' ' ' '**

**OP\_REPT\_TOP | 1 1 1 0 0 1 0 |TL?| ' ' ' '**

**+---------------------------+---+ ' ' ' '**

**| 1 1 1 0 0 1 1 # | ' ' ' '**

**<RESERVED> | 1 1 1 0 1 # # # | ' ' ' '**

**| 1 1 1 1 # # # # | ' ' ' '**

**+-------------------------------+---------------------------------------------------------------+---- -- - - - - - - - - -- ----+**

**OP\_EXIF | 1 1 1 1 1 1 1 1 | EXIF DATA LENGTH (IN BYTES) | EXIF DATA |**

**+-------------------------------+---------------------------------------------------------------+---- -- - - - - - - - - -- ----+**

**The processing of EXIF data inside OP\_EXIF chunks is optional, however, the size taken by OP\_EXIF chunks has to be respected. The decoder should continue as normal should it encounter any malformed chunk, thus resulting in a deterministic decoding behavior.**

Each chunk begins with a constant operation code (opcode, OP) which indicates the chunk type. The chunk’s opcode has a typical length of **2-8 bits**. The opcode is followed by the chunk’s remaining fields and flags.

The QOIF image encoder iterates over each pixel in a row-major order and emits the chunk best representing current pixel(s). Each time a chunk is emitted, the last pixel value will be stored in the variable **prev**, while the value of **prev** will be moved to **pprev**. Furthermore, the running cache “**cache**” will be updated as follows:

* The current pixel value **current** (uint32\_t) consisting of the four bytes **r**, **g**, **b**, and **a** is used to compute a deterministic hash value (“**index**”).
* The variable **cache** is updated at **index** to store **current**. This may overwrite any previously stored value.

The cache is updated as follows (note that the **UL** suffix indicates the data type **uint64\_t**):

**uint32\_t current = ...;**

**uint\_8t index = (((current & 0x3f3f3f3fUL) \* 0x0c141c2cUL –**

**((current & 0x00200000UL) << 5)) >> 26) & 63;**

**cache[index] = current;**

The image decoder is required to keep track of the current and previous pixel values in a similar fashion in order to preserve data consistency.

The following sections describe the individual chunk types and their respective encoding:

**OP\_INDEXED**

**7 6 5 4 3 2 1 0**

**+---+-----------+**

**|0 0| index |**

**+---+-----------+**

The **OP\_INDEXED** chunk begins with the constant **2-bit opcode 00**, followed by a **6-bit unsigned integer** number which indicates a cache index in the range of **0-63**.

If the image decoder parses an **OP\_INDEXED** chunk, it sets the currently processed pixel to the value of **cache[index]** and moves onto the next pixel.

**OP\_2BIT\_DIFF**

**7 6 5 4 3 2 1 0**

**+---+---+---+---+**

**|0 1|red|gre|blu|**

**+---+---+---+---+**

The OP\_2BIT\_DIFF chunk begins with the constant **2-bit opcode 01**, followed by three **2-bit** numbers (range **-2** to **+1**), which represent a color change from the current pixel in relation to **prev**. The fields **red**, **gre**, and **blu** represent a value change in the pixel’s color channels red, green, and blue, respectively. The alpha channel remains unmodified. Each value change is encoded as follows:

Binary value Pixel value change

**00 -2**

**01 -1**

**10 0**

**11 +1**

During the encoding process, the binary values are computed as follows:

During the decoding process, the new pixel is computed as follows:

**OP\_2CHN\_DIFF**

**7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0**

**+-----+-----+---------+---------+**

**|1 0 0|selct| Δchn1 | Δchn2 |**

**+-----+-----+---------+---------+**

Execution preference:

RUN\_PPREV

RUN\_PREV

PAL\_INDEXED

INDEXED64

2BIT\_DIFF

AVG

LUMA\_DIFF

2CHN\_DIFF

LUMA\_TOP\_DIFF

HUE\_ROT

1CHN\_DIFF

INDEX\_LUMA

RGB

RGBA