FIC File Format Specification

Jeremy Berchtold

February 2019

Introduction

FIC (Fast Image Compression) is a compression algorithm focused on compression/decompression speed over a better compression ratio. This document outlines the file format used to store images using FIC.

Outline

A FIC file contains the following elements in order: header, edge map, pixel data.

Header

The header contains a magic number and specifies the width and height of the image.

Edge Map

The edge map specifies which pixels are on edges and therefore don't use delta compression.

Pixel Data

The pixel data contains the colors of all pixels, some stored using raw 8-bit values per channel, and some stored using delta compression.

Header

The header is 12 bytes total and is divided into three sections: the magic number, the width, and the height. All of these values are 32-bits. The magic number is always the following bytes: `0x00`, `0x46`, `0x49`, `0x43`. The width and height are the width and the height of the image in pixels stored as 32-bit unsigned integers.

Edge Map

The edge map is a 2-dimensional array of bits. The dimensions are `width` and `height` giving it a total size of `(width*height)/8` bytes.

2-dimensional array

It is stored in row-first ordering, meaning the first `width` bits are the first row `(y=0)`, the next `width` bits are the second row `(y=1)`, and so on. The formula for calculating the bit index of the bit corresponding to a pixel at (x, y) is `bitIndex = (y*width + x)`. Keep in mind this is the bit index, not the byte index.

Bit value

If the bit has a value of `1`, the corresponding pixel is on an edge, and therefore will be stored using raw uncompressed values. If the bit has a value of `0`, the corresponding pixel is not on an edge, and therefore will be stored using delta compression.

Pixel Data

The pixel data, also known as the body of the file, stores the actual pixel values for the image. The pixel values are stored in a 2-dimensional array, similar to the edge map.

2-dimensional array

Like the edge map, the pixel data is stored row first. This means the first `width` pixel values will be the first row `(y=0)`, the next `width` pixel values will be the second row `(y=1)`, and so on. Random access is not possible because with delta compression some pixel values will use different amounts of bits, so decode time for a single pixel is $O(n)$.

Raw pixel values

If a pixel is on an edge (as specified by the edge map), it is stored in a raw uncompressed format. 8-bits are used for each channel and the data is stored in RGB order.

Compressed pixel values

If a pixel is not on an edge (as specified by the edge map), it is compressed using delta compression. The deltas are calculated relative to either the pixel to the left or the pixel above the current pixel. The first bit of the data for a compressed pixel will be `1` if the deltas are relative to the pixel above; and the bit will be `0` if the deltas are relative to the pixel to the left. Following the first

bit is a set of 3 prefix-free codes representing compressed deltas for each of the channels RGB, in that order.

Delta compression

Deltas are compressed so deltas with a smaller magnitude result in shorter prefix-free codes and larger deltas will have longer codes. The idea behind this is that smaller deltas will be more common in images than large jumps in color. The compressed delta is divided into three parts: index, sign, and offset. The the index is a string of `1` bits terminated by a `0`. The number of `1` bits in this set of bits is the value of the index. Next is the sign bit, which is `1` if the value is non-positive and `0` if the value is positive. Lastly is the offset, which stores an extra bit of precision. The formula to calculate the value of the delta from the fields is dependent on the value of the sign bit.
If the sign bit is `1`, then `delta = -2*index + offset`.
If the sign bit is `0`, then `delta = 2*index + 1 + offset`.
Using a C-style ternary operator expression, this condenses to
`delta = (sign ?  -2*index :  2*index + 1) + offset`.