# Cloud-based RAW Image Editing

Ryan Collins

Department of Computer Science, Durham University

## Introduction

This poster describes the design and implementation of a Cloud-based system for processing and editing RAW images. Previous RAW image editing software is designed for native use and most of the current software available is proprietary. This paper shows the design and implementation of a system that's not proprietary, and allows for editing of RAW images through our Cloud-based RAW image editing service.

## RAW Images Explained

RAW isn't a file format in itself, but rather an umbrella term for a set of many different formats that store the raw camera sensor data. There are a set of different proprietary file formats, that all store similar information, but in different ways. When an image is captured, the camera sensor uses charge buildup to represent the amount of light that is incident on the sensor (using a phenomenon known as the photoelectric effect). Colour data itself isn't stored at all in the RAW image. Colour is achieved by putting a filter over the sensor, such that each individual part of the sensor captures only red, green or blue light, which then allows one to build up a full-color image. The process of creating a full-color image from the camera sensor is called demosaicing. There are several different steps to decoding RAW images, in addition to demosaicing which was explained above. These are:

- **White Balance** A light source doesn't necessarily emit white light, but light of some tint. This process corrects for this tint, to ensure the white colours look white in the image, rather than a tinted white/orange.
- **Gamma Correction** Cameras typically represent color changes linearly, with a gamma of 1.0. However, this isn't necessarily pleasing to the eye, so this can be changed to change how tones are reflected. This can be used to simulate the tonal output of film cameras, for example.
- **Sharpening/Noise Reduction** Sometimes, noise can creep into the image, particularly if the image itself is shot with large ISO levels (ISO is a sensitivity setting that can be set when taking a photo, to allow the camera sensor to be more sensitive to light). This can be corrected during the editing process. Also, sometimes edges might need to be enhanced to bring out the detail in the image. This can again be used for stylistic purposes, but this is ultimately the decision of the image editor.

## Test of Exposure Adjustment

Figure: Test of Exposure Adjustment (a) Exposure Level Small: 0.0001 (b) Exposure normal: 1.0 (c) Exposure level high: 5000

## Test of Colour Adjustment

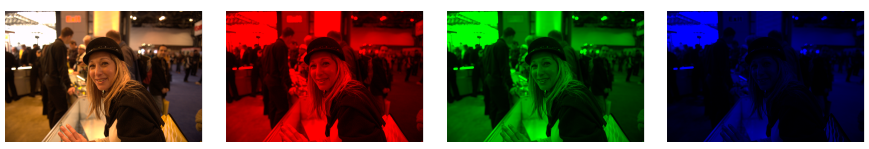

Figure: Test of Colour Balance (a) No Adjustment (b) Red only (green and blue zero) (c) Green only (red and blue zero) (d) Blue only (red and green zero)

To test the functionality of color adjustment, the output of each channel individually is tested. If the outcome is only the specified channel, then the color balance component of the system works.

## Test of Gamma Correction

Figure: Test of Gamma Correction output. (a) $\gamma = \mathbf{0.3}$ (b) $\gamma = \mathbf{1.0}$ (c) $\gamma = \mathbf{1.7}$

## System Architecture

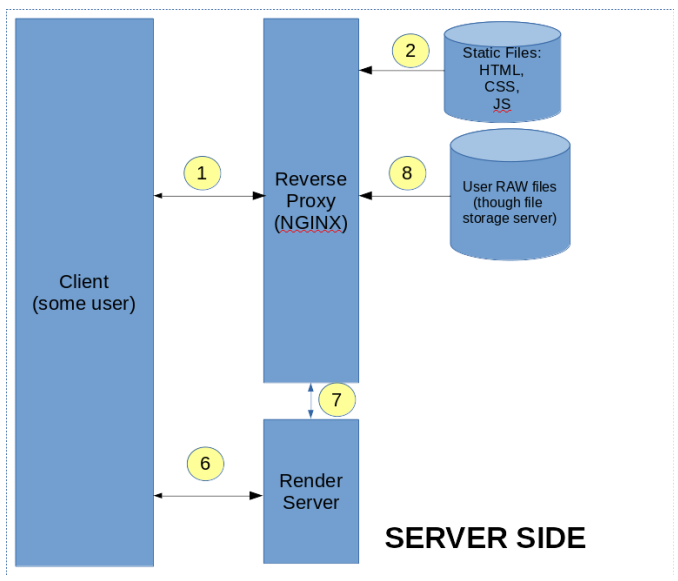


Figure: Overall System Architecture

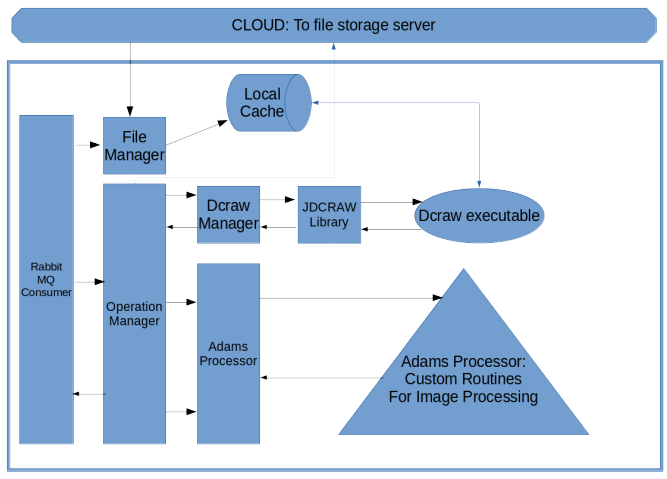


Figure: Render Server Architecture

## System Workflow and Architecture

The system receives a set of instructions, encoded using JavaScript Object Notation (JSON). These instructions outline what RAW image to use as the source, along with the settings to use for rendering. Due to the proprietary nature of RAW files, creating a custom RAW parser from scratch was not advisable, due to a large number of formats, and different structures and information contained within these formats. While Adobe's DNG standard was considered, one could be required to convert other RAW files to DNG before editing, which becomes difficult on certain types of operating systems as Adobe's DNG converter is only compatible with Windows and Mac. instead, portability was designed for, by using a C executable called dcraw, containing no dependencies, to process the RAW file and output as an uncompressed TIFF for further editing in Java through custom image processing operations. Java was used as the primary language for image manipulations, alongside links to the dcraw C executable. A message queue was used to spread out requests across the system. A Node.js server is used as the interface between the user and the message queue, and each render server fetches unique jobs from the message queue. The entire system was deployed and connected using Docker and Docker Compose.

## Evaluation

Cloud-based RAW image editing is useful for some applications, especially in providing an API to other applications that wish to work with RAW images. However, in a real-time setting, the current implementation isn't ideal due to network overheads, coupled with lack of multi-threading. Improvements can be made by creating a custom Image representation, which can allow multi-threading and prevent various memory leaks that occur in the current system.

## Conclusion

Overall, the Cloud-based RAW image editing idea is feasible and works well. While the performance isn't as good as native RAW editors, RAW image editing as a service works, and providing some delay is allowed, the system works.