Project Plan: Cloud-based RAW Image editor

Ryan Collins
Supervisor: Tom Friedetzky

October 22, 2017

1 Description

This project is about creating a RAW image editor for the Cloud. While technologies exist to process normal images (e.g. ImageMagick), there aren't really many Cloud-based tools for editing RAW images, either using some web application, or through a web API.

There will be two main outcomes from this project:

RAW editing web service The creation of a RAW image editing service, that is given a RAW image URL, along with a set of instructions to carry out, so that an image can be rendered.

A RAW Image Editor Interface This is a UI that uses our image editing service to process images, and provide some form of interface to the user to carry out instructions
We are creating two systems here rather than one, because there are cases where the web service will also prove useful independently of the web editor. An example is within Content Management Systems, in the processing of RAW images through a content management system. This would allow the user to bypass editing of RAW images within an off the shelf program, before uploading the exported file to the content management system, to then be processed again to fit the content. This would improve the editor's workflow. This is only one of many possible scenarios.

2 Preliminary Preparations

There are numerous questions that we need to answer, before designing our system, so that we have an idea of what we are building, as well as ensuring that we can design such a system properly, and learn from the mistakes of others who have found faults in their own designs. These questions include:

- How can one programatically read RAW files, and process them? Specifically, on GNU/Linux platforms?

- What are the most common image operations within RAW editors?
- How is a RAW file structured, and what information does it contain?
- Implementation wise, what optimisations can be made to improve the performance of our image processing system?
- What are the constraints that we need to work with?
- Regarding the implementation, what technologies/programming languages/libraries should be used, and which components should be written by us?
- How can we interface with between the client and the server? What protocols exist?
- Are there any existing systems that follow this design?

3 Research Question

4 Deliverables

4.1 Our deliverables

Our deliverables, as taken from the Specification document, are as follows:

Minimum Objectives

- Load DNG RAW files by upload
- Exposure adjustment
- Noise reduction methods (Gaussian, mean, median)
- Web Interface interacting with an image processing server
- Non-destructive image adjustment (i.e. no reduction in quality over time)

Intermediate Objectives

- Modern, user friendly User Experience
- Spot healing implemented
- Haze removal
- cropping, rotating and exporting to other formats

Advanced Objectives

- Addressing potential scalability issues
- Implement basic previews using built in JPEG to improve QoS

4.2 Achieving our deliverables

Load DNG RAW files by upload Achieved by using an already existing storage API (such as OpenStack Swift), and linking this with our web frontend. In reality, we could use any potential storage mechanism, such as NGINX/Apache, or even just a service like Google Drive. OpenStack Swift is more like other Cloud-based storage services like Amazon's S3, so this should aim to mirror a real-world implementation.

Exposure Adjustment We could potentially use the dcraw-based exposure adjustment to do this, along with multiplying a number by the values in an image matrix.

To go further, histogram equalisation can be brought into this, to allow for automatic exposure adjustment, based on the image itself. This feature would need to be implemented using the secondary (and custom written) processor within the render server.

Noise reduction methods Noise reduction methods such as Mean and Gaussian can be implemented using convolution. The appropriate kernels are needed to achieve these, but once we have this component, we can fairly easily Convolve thanks to Java's built-in image processing functionality.

Median is slightly more difficult, as it's not directly linear (so we therefore cannot use convolution). Therefore, we will need to process the image ourselves, and consider parallelising our code to ensure that this operation is achieved more promptly. Research into better algorithms/implementations for this should also be considered, as by changing the algorithm slightly, we can optimise for speed.

Web Interface More research into the best approach regarding server to client communication is needed, to determine whether web-socket, or REST is the best design.

In terms of the implementation, the initial plan was to use Vue.JS but due to the heft of a potential JS framework application, we should also consider implementation is Vanilla JS/jQuery.

Frontend design should also be important, but not the major aspect of our system. The priority should be a basic web interface that's somewhat usable, coupled with a functional backend.

Non-Destructive Image adjustment By design, the system will non destructively edit images, providing we use uncompressed TIFF output from dcraw. Furthermore, we should use the RAW file as the basis for all image manipulation.

Spot Healing Implemented More research into spot healing algorithms needs to be undertaken, and then optimisation of these algorithms for our application needs to be carried out.

Only then, can implementation be considered.

Haze Removal Currently we have a few sources for potential Haze Removal algorithms. More research needs to be done to determine which is more suitable for our application. Or potentially, one could implement both?

Cropping, rotating and exporting Cropping is a relatively trivial task, as it just involves choosing a selection of pixels to keep, and the rest can be thrown away.

Rotation is more difficult, as this requires a transform using sine and cosines to new pixel locations along a space (in this case, the output image).

Exporting to other formats, while it might seem trivial, is mostly a library task. While it's an important part component of a photo editor, we will need to consider the different attributes to each of the major file formats (such as TIFF, PNG, JPEG).

Scalability Issues A more theoretical analysis should be undertaken to help prevent this on a theoretical level (in terms of the maximum number of connections).

A more applied test would be useful to determine how well our system performs with different algorithms. This might require a written tool to bypass the web frontend, access the render server directly, issuing commands using web socket.

5 Gantt Chart