Cloud-based RAW image editing

Student Name: Ryan Collins (gcdk35)

Supervisor Name: Dr Tom Friedetzky

Submitted as part of the degree of M.Eng Computer Science to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Context/Background

Aims The main aim of this project is to test the feasibility of a Cloud-based RAW image editor.

Method A render server backend will first be implemented as an API, taking in an input as a JSON object, and then processing the image, and then a JavaScript client shall be created to interface with this API.

Proposed Solution A web application that uses dcraw coupled with custom Java code to read RAW images, and allow adjustment of various parameters, with the output being sent back to the user.

Keywords — RAW image editing, dcraw, cloud image editing

I INTRODUCTION

Many photographers use a file format (or rather, a family of very similar file formats) called RAW, which rather than compressing the image and conducting some image manipulation on the camera, store the RAW camera sensor data outputted by the camera sensor, for later processing and editing by a computer. These files can be much larger than the compressed image, but provide a far greater degree of control over the captured image, when compared with a compressed JPEG, along with an increase in quality. A RAW file essentially acts as a digital negative, as the image can be edited constantly without losing any quality between edits. (Verhoeven 2010)

A Project Aim

B Deliverables

II DESIGN

This section presents the proposed solutions of the problems in detail. The design details should all be placed in this section. You may create a number of subsections, each focusing on one issue.

This section should be up to 8 pages in length. The rest of this section shows the formats of subsections as well as some general formatting information. You should also consult the Word template.

A Requirements

B Proposed Extensions

C Architecture

As the system is fairly large, it's important to break it down into individual pieces. These are:

C.1 Client Interface to Render Server Communication

There are several options for communicating between the user interface and the render server.

Representational State Transfer (REST) A Representational State Transfer system would work by sending a request to the server, requesting an image be rendered. This initial request will be replied to with a job number, which will be quoted in further communications. From here, future requests will be made over a regular interval, requesting the information for the specified job. If this job is finished, the result will be returned, but otherwise further requests will need to be made. This is the process of polling.

The XMLHttpRequest object in JavaScript, used to make AJAX requests, was designed by Microsoft in 1999, and later adopted in the 2000s. This method is definitely the most compatible with browsers, being compatible with Edge, Chrome, Firefox, Internet Explorer 7+, Opera, and Safari. (Mozilla 2017)

However, this method does require making many requests and connections to the server, coupled with code to regularly poll at an interval until done. While this will work, it's not the most optimal solution, and the code produced from this would be more complex (code could be needed to issue jobs, recall jobs, and to ensure that the person who requests the result of a job is actually allowed to see the result of the job).

Web Socket Web Socket allows for two way communication between a browser and a server. It acts in a similar way to traditional TCP socket communication, only it incorporates the origin based security model used within web browsers. By using web socket over REST, opening multiple HTTP connections is not needed, as a single connection is maintained at all times. (Melnikov & Fette 2011)

Socket.io Socket.io is an implementation that relies on both REST and Web Socket. If the browser supports web socket, then it is utilised, but in the event that the user's browser does not support new web socket technologies, then it defaults to using REST, and automatically polls regularly for information. This way, we get the best of both options shown above, in such a way that the code itself remains fairly tidy (as the polling nature of REST is abstracted away from our system).

C.2 Render Server

The render server takes instructions given to it (with accordance to our API), and generates the output image based on the RAW image supplied, and the appropriate settings.

C.3 Client-side Interface

The page design of the interface shall follow the design in D. The goal of the client side interface is to allow adjustment of the image parameters, and show a preview whenever a parameter is changed.

To display the image, an HTML5 Canvas will provide a large amount of control to how we can display the image, allowing for features such as zooming, and drawing. This can't be achieved using a standard HTML image.

D User Interface

A sidebar should be used as the main interface for adjusting image parameters. This sidebar should be able to be hidden, showing the image fully underneath. When the sidebar is in the expanded state, the preview image should be displayed fully on screen.

The user interface design is shown in Figure **??**

Within the sidebar, clicking on a navigation item will display a new submenu. If the item is a parameter adjustment, updating the value in the menu will also update the value that is sent to the render server to generate a preview.

E Evaluation

III REFERENCES

References

Melnikov, A. & Fette, I. (2011), 'The WebSocket Protocol', RFC 6455.
URL: *https://rfc-editor.org/rfc/rfc6455.txt*

Mozilla (2017), 'Xmlhttprequest'. Accessed: 2018-01-09.
URL: *https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest*

Verhoeven, G. J. J. (2010), 'It's all about the format unleashing the power of raw aerial photography', *International Journal of Remote Sensing* **31**(8), 2009–2042.
URL: *https://doi.org/10.1080/01431160902929271*