3D Colour Histograms

Callum McGregor

# Abstract

3D colour histograms show how the colours of an image are distributed within a colour space. The aim of this project was to develop a program that allows one to observe the effects of colour alterations, such as brightness, contrast, and saturation on a histogram. The project was developed as a web page and most of the code was written in JavaScript, in particular using the WebGL and Three.js rendering libraries. A short and iterative development cycle was adopted, accompanied by unit tests written using the test driven development paradigm. This ensured that development was organised into manageable pieces. Once completed, the final product was evaluated against the original design and the potential of histograms as a visualisation tool was assessed. The product met all of the requirements set in the design and experimentation showed that the colour controls had significant and distinct effects on the histogram; compared to conventional 2D visualisations, the 3D histogram provided perspectives and benefits unobtainable by any one other representation. These results are evidence that 3D colour histograms can be used to better understand the effects changing an image’s colour and that there is a viable case for their use in image editing software. There is also the potential for improvements in the product’s performance and for future work: in particular the histogram may be developed into a tool that allows colours to be directly manipulated to recreate conventional controls and even perform operations currently difficult to achieve.

I certify that all material in this dissertation that is not my own work has been identified:

April 2016

Table of Contents

1 Introduction 4

2 Summary of Literature Review 4

2.1 Colour Spaces 4

2.2 Colour Histograms 5

2.3 Technologies 5

2.4 Colour Quantisation 6

2.5 Colour Adjustments 6

3 Summary of Specification and Design 6

3.1 Project Introduction 6

3.2 System Architecture 7

3.3 Component Design 7

3.3.1 Model 8

3.3.2 View 9

3.3.3 Controller 9

3.4 User Interface 9

3.5 Testing and Evaluation 10

3.6 Time Plan 10

4 Development and Testing 11

4.1 Development and Testing Methodology 11

4.2 Points of Note 13

5 Description of the Final Product 14

5.1 System Architecture 14

5.2 Component Design 14

5.2.1 Model 15

5.2.2 View 16

5.2.3 Controller 17

5.3 User Interface 17

5.4 Product Behaviour and Performance 17

6 Evaluation of the Final Product 18

6.1 Basic Requirements 18

6.2 System Architecture 18

6.3 Component Design 19

6.4 User Interface 20

6.5 Product Behaviour and Evaluation 20

7 Critical Assessment of the Project as a Whole 22

7.1 Pre-Development 23

7.2 Development Cycle 23

7.3 Testing 24

7.4 Time Plan 24

7.5 Results and Evaluation 24

8 Future Work 25

9 Conclusion 25

Bibliography 27

Appendices 29

Project Diary 29

Colour Space Transformation and Colour Alteration Formulae 34

# Introduction

During my summer internship of 2015 I worked on the visualisation of colour spaces, and it was there that I was introduced to 3D colour histograms by Javier Villarroal’s website (2013), which allows the user to view the histograms of stock images. Five of these stock images caught my eye: they were pictures of the same scene but with various adjustments made to the image. The differences between these histograms fascinated me and inspired this project.

The aim of this project is to develop a product that will display the histogram of an image and update it as colour alterations are applied with the potential of it being used as a visualation in image editing software. I will then use this tool to investigate whether these adjustments have a significant and useful effect on the histogram in a way that provides a unique visualisation, unavailable in tools currently available such as 2D histograms. If I conclude that the histogram is a useful visualation I will begin to explore the potential of using it as a tool to manipulate colours rather than just visualising them.

This document has been designed to be self-containing and none of my previous pieces of work are required in the reading of this report. Section 2 summarises the literature review and section 3 summarises the specification and design document, which defines the scope of the project, outlines the project design, and the development and testing methodologies. Section 4 discusses the development and testing methods that actually took place in the project with sections of my development diary provided as illustrations. Following a similar structure to section 3, section 5 describes the design of the final product as well as its behaviour and performance; this is then compared to the original design in section 6, which provides justification for points at which the final product departs from the original design also evaluates the product by using it to investigate the histogram’s effectiveness as a visualisation tool. Section 7 provides a critical assessment of the processes that took place throughout the entire project, including lessons learnt and changes that would be made with hindsight. Finally, loose ends, improvements, and the future potentials are discussed in section 8.

# Summary of Literature Review

The literature review analysed and drew conclusions from sources gathered during research for the project. Five key areas emerged from this research: colour spaces, colour histograms, technologies, colour quantisation, and colour alterations. This section of the report summarises the literature review, introducing each topic and stating the conclusions drawn, with the aim of providing the knowledge necessary in the reading of this report.

## Colour Spaces

Colour spaces are a method of representing colour in a coordinate system. Research into this area was important as the spaces in which to display the histogram had to be chosen and as each has its own properties. It was also necessary to transform between colour spaces so research into colour space transforms was necessary.

From Andrew Steer’s introduction to colour science (2008) and Phil Cruse’s overview of the CIE-L\*a\*b\* space (2015) it was concluded that sRGB and CIE-L\*a\*b\* would be suitable spaces in which to display the histogram: sRGB because it is a standard and easily understood, and CIE-L\*a\*b\* because it better preserves perceived uniformity of colour distribution. Hoffmann’s papers on CIE-XYZ and L\*a\*b\* (CIE Color Space, 2015) (CIELab Color Space, 2015) also helped to draw this conclusion, but her papers also provided information on colour space transformations; alongside Ryan Juckett’s (2010) and Irotek’s (2014) websites, Hoffmann’s papers provided the knowledge and formulae necessary to transform colours between the sRGB, CIE-XYZ, CIE-L\*a\*b\*, and HSL spaces (CIE-XYZ is used as an intermediate colour space when transforming between sRGB and CIE-L\*a\*b\*).

## Colour Histograms

The 2D colour histogram of an image plots light levels against the number of pixels with that light level. Luminosity histograms plot the absolute lightness while colour histograms contain three plots, one for each RGB colour channels. 3D histograms plot colours in 3 dimensions: axes represent primary colours in the chosen colour space and frequency is represented by the size of plots.

|  |  |
| --- | --- |
| Figure : 2D Luminosity and Colour Histogram  (https://upload.wikimedia.org/wikipedia/commons/f/f5/Odd-eyed\_cat\_histogram.png) | Figure : 3D Colour Histogram  (http://rsb.info.nih.gov/ij/plugins/images/3d-inspector.jpg) |

Most sources were on 2D histograms and their applications in photography, such as Practical Photography Tips (PracticalPhotographyTips). These sources helped build an understanding of histograms but were of limited value. Sources on 3D histograms were all examples of other programs, including Villarroal’s (2013), Third Avenue Design’s web page (Third Ave Design) web page, and Barthel’s Java application (2007). Through inspection of these sources, the best features were compiled and incorporated into this project.

## Technologies

Prospective technologies for the project were also researched. From previous experience, WebGL and Three.js, two JavaScript rendering libraries, were known to be appropriate tools.

From a comparison of the Khoronos Group’s and Parisi’s WebGL tutorials (Khoronos Group, 2015) (Parisi, 2010) and Three.js tutorials (mrdoob, 2015), it was concluded that while established, WebGL’s code was too verbose compared to Three.js, which wrapped up WebGL’s functionality into a concise and easily used format. Research also uncovered GLSL, a scripting language run on the client’s GPU to improve performance. GLSL also had the benefit that it could be incorporated into Three.js (mrdoob, ShaderMaterial).

## Colour Quantisation

If the exact colours from an image were plotted on a histogram their frequencies would be too low. Colour quantisation is a method of compressing an image by reducing the number of colours and it was found that it could be applied to allocate colours into ranges before plotting them on the histogram.

Segenchuk’s webpage (1997) explains various quantisation techniques and from this it was concluded that the simpler techniques were suitable for the project because they sampled the image in a uniform grid, unlike other methods. Eric Liao’s GitHub page (2013) provided a pseudo-code example of an appropriate quantisation algorithm that could be applied in the project.

## Colour Adjustments

The colour adjustments made available to the user had to be chosen carefully and those chosen should have the best chance of having a significant effect on the histogram.

Apple’s iPhoto (2015) provides various colour controls and a 2D colour histogram. Experiments were conducted into which colour controls had the greatest effect on the histogram. Exposure (brightness), contrast, saturation, and lift/gamma/gain were identified as having the most potential. Hurkman’s web page (2013) provided formulae for applying lift/gamma/gain but no formulae for the other controls were found during research.

# Summary of Specification and Design

The specification and design document was written in preparation for the development of the project. This section of the report summarises the design for the sake of completeness and so that the description of the final product’s design in section 5 may be evaluated against it in section 6.

## Project Introduction

During my summer internship I discovered Javier Villarroal’s 3D colour histogram (H3Stogram - 3D Interactive Color Histogram, 2013) and it motivated me to investigate the effects of colour alterations on histograms. The aim of this project is to develop a program that allows me to investigate whether colour alterations have a significant and useful effect on histograms and whether they would provide a unique perspective on colour spaces.

The target audience of the project is users of image editing software that would benefit from a visualisation of colour spaces and the effects of colour adjustments on the distribution of colours within them. The project requires a basic knowledge of colour spaces and colour manipulation in order to be able to fully appreciate they benefits of the histogram.

The basic requirements of the project are that it will allow the user to: upload an image from their local file space; view the image’s 3D colour histogram in the sRGB and CIE-L\*a\*b\* spaces; alter the image’s brightness, saturation, and contrast; and view the effects of these adjustments on both the image its histogram.

## System Architecture

The project’s architecture is designed with the Model View Controller (MVC) design pattern. MVC separates the parts of the program responsible for processing and displaying data: the model processes data and contains the program’s business logic; the view handles presentation of data, in this case the histogram; and the controller acts as a liaison between the model and view, prompting the model to update its data and instructing the view to refresh its view of data. The basic UML diagram of an MVC application can be seen in Figure 3:1. MVC was chosen because the separation of presentation and business logic allows flexibility and is a tried and tested design.

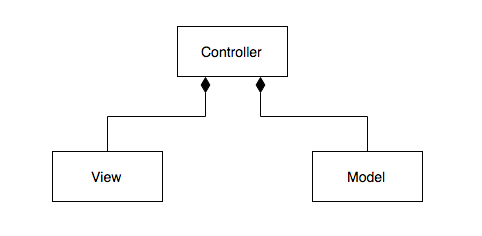


Figure : Model View Controller UML Diagram

## Component Design

The project is broken down into concise component tasks. The data flow diagram in Figure 3:2 illustrates how data is passed between tasks and how they fit into the MVC design.

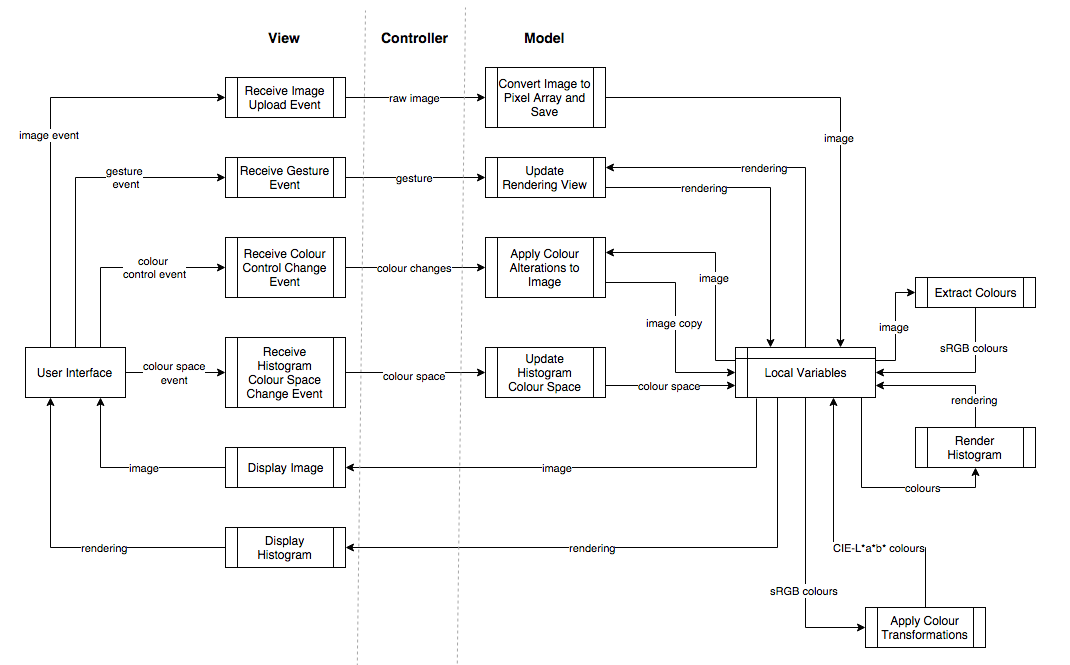


Figure : Data Flow Diagram

### Model

Tasks in the model include: extracting colours from an image, transforming between colour spaces, applying colour alterations, and generating the histogram rendering. State need not persist beyond the life of the program so data is stored in fields instead of databases.

Transforming between the sRGB and CIE-L\*a\*b\* spaces is achieved with an intermediate space, CIE-XYZ, as explained in section 2.1. Converting sRGB to XYZ is done by linearisation followed by a transformation matrix: Figure 3:3 and Figure 3:4 show formulae and equations for these two operations. XYZ is transformed into L\*a\*b\* with another formula, seen in Figure 3:5.

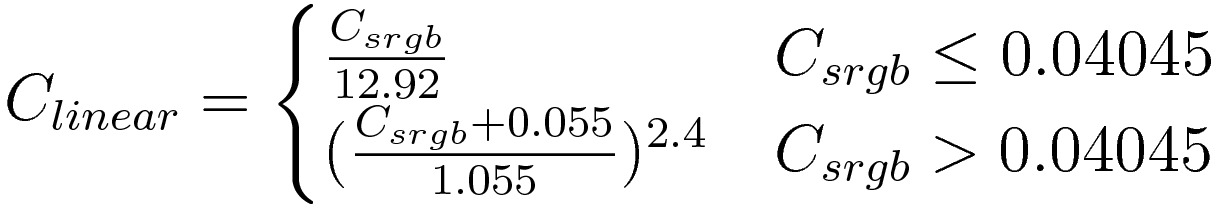


Figure : sRGB Linearisation Formula (Juckett, 2010)

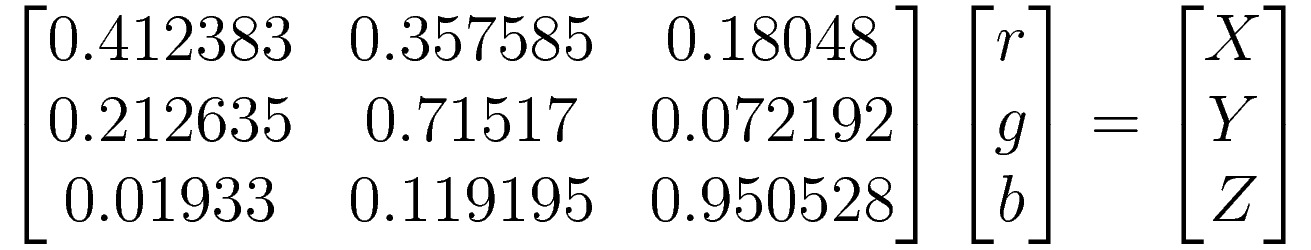


Figure : sRGB to CIE-XYZ Transformation Matrix (Juckett, 2010)

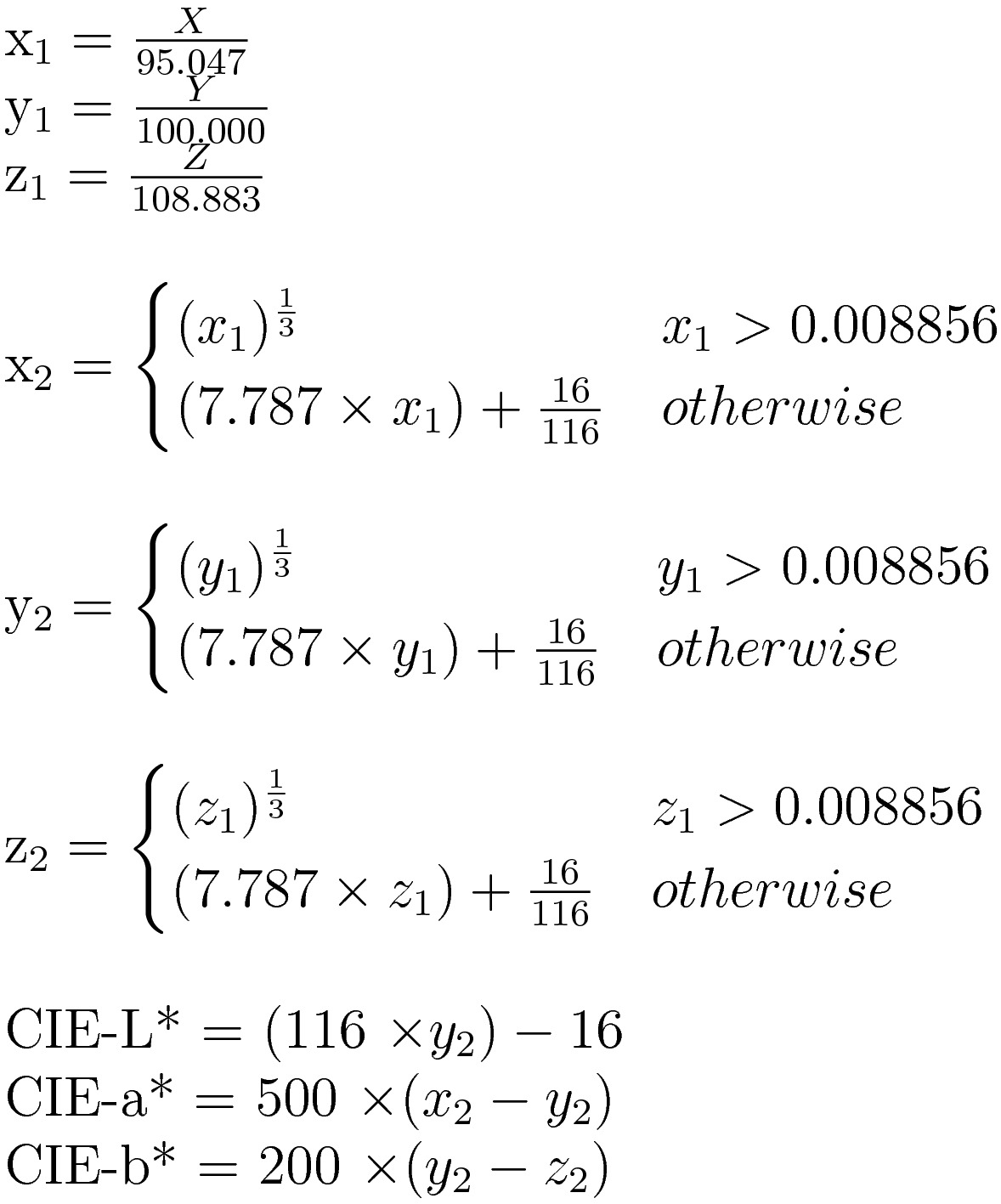


Figure : CIE-XYZ to CIE-L\*a\*b\* formula (Irotek Group Ltd., 2014)

All colour adjustments are applied to colours in the sRGB space. Brightness is adjusted by adding or subtracting a constant value to or from each colour channel. Contrast is adjusted using the formula in Figure 3:6, which calculates a contrast factor before applying it to each colour channel. Saturation is adjusted by transforming into the HSL space and adding or subtracting to or from the S value. The transformation formulae for sRGB and HSL can be seen in Figure 3:7 and Figure 3:8.

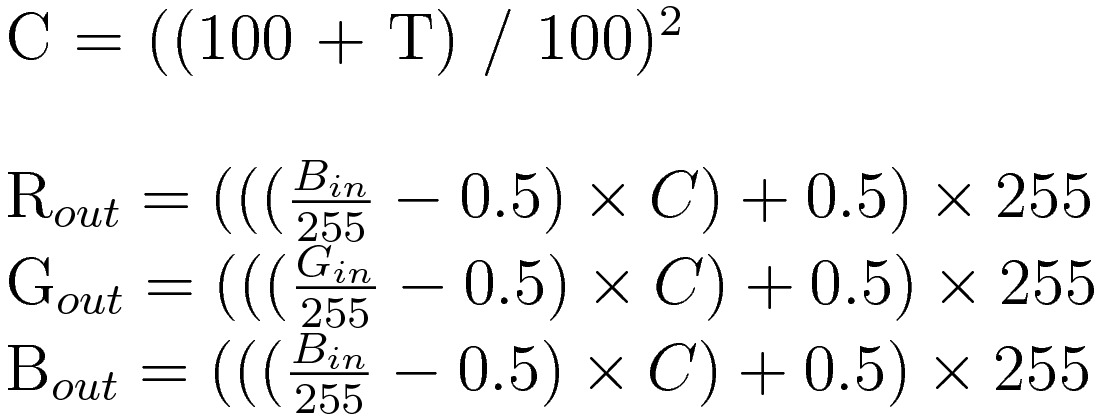


Figure : Contrast Formula (Esterhuizen, 2013)

|  |  |
| --- | --- |
| Figure : sRGB to HSL (RapidTables, 2015) | Figure : HSL to sRGB (RapidTables, 2015) |

### View

Events are fired by the user interface in the event of the user uploading an image, performing a gesture on the rendering canvas, changing the histogram colour space, or adjusting the colour controls. In each case the event is caught by the view and relevant information is sent to the controller. The other task of the view is to update its representation of the model’s data, in this redrawing the image and its histogram.

### Controller

Figure 3:2 does not explicitly define tasks for the controller and instead implied tasks lie where data is passed between the view and model. These implied tasks either receive events from the view or fetch data from the model when its state changes and passes it to the view.

## User Interface

Figure 3:9 shows a sketch of the user interface design, which consists of the histogram surrounded by the image, an image upload button, three colour control sliders, and a colour space radio button. Uploading an image, adjusting colour controls, or changing the colour space will fire events, handled by the view. Additionally, events are fired when the user left or right click drags or scrolls over the rendering, providing rotation, panning, and zooming controls on the rendering.



Figure : User Interface Sketch

## Testing and Evaluation

The program will be tested with a suite of unit tests written with the test driven development paradigm, which states that tests should be written to fail followed by the code to make them pass. Tests should be written to cover a broad range of scenarios, focusing on edge cases that are the most likely to cause erroneous behaviour. Intentionally erroneous data should also be tested to ensure that the program handles it correctly. It is not possible to formally test the rendering but instead manual tests should be conducted, in which the output will be compared against expected results.

The project was founded on the idea that colour alterations might have a significant and useful effect on histograms. While not a formal hypothesis, these ideas will be tested in the evaluation of the project, along with comparing the final product against the project requirements and design in section 3.1. To avoid bias, impartial people should be used to evaluate the project.

## Time Plan

The time plan, shown in Figure 3:10, defines tasks down the left-most column and highlights the weeks in which they should be completed in the subsequent columns. Development is broken down into the first 13 tasks and the final 3 are writing this report and preparing for the poster session and demonstration. Formal deadlines are explicitly marked and informal deadlines are implied at the end of each task’s allotted time.

Dividing development into tasks allows a short and iterative development cycle to be adopted: each cycle consists of implementing a feature, testing it, and a brief reflection and evaluation. Short cycles make development flexible, as frequent adjustments to the time plan or design can be made, and prevent one from going off topic.



Figure : Time Plan Gantt chart

# Development and Testing

The development and testing methodology is defined in section 3.6 and this section of the report describes the development and testing process that actually took place, provides illustrations, and describes points of interest during development. A development diary containing a detailed record of progress was kept and it is used to provide evidence: a complete copy of this diary can be found in the appendices.

## Development and Testing Methodology

Short and iterative development cycles consisting of implementation, testing, and reflection were kept throughout the project. The diary entry for week 6 of term 2 illustrates how the task of implementing sRGB to CIE-L\*a\*b\* transformations was completed during the scheduled week:

Week 6 (15/2/2016 – 21/2/2016)

* Used simplified colour transformations (found at <http://www.easyrgb.com/index.php?X=MATH&H=07#text7>) to convert from RGB to Lab
* Refactored ColorRGB class to incorporate both RGB and Lab values into a single object
* New Color class also includes the code to convert between colours spaces

While the time plan was never altered there were times when it was not strictly adhered to and the project fell behind schedule. The diary entry for week 9 of term 2 shows that the goal of implementing colour adjustment formulae was not met that week:

Week 9 (7/3/2016 – 13/3/2016)

* Little progress made …
* Secondly, adjusting brightness results in strange artefacts (extreme colours) to appear
* Are these a result of a poor brightness algorithm? Find out by changing the colour of the image

Despite this small setback, the next diary entry shows how development was back on track by the following week:

Week 10 (14/3/2016 – 20/3/2016)

* The image now changes colour with the histogram …
* Saturation adjustment now also works, using HSL in the same way to brightness …
* Contrast adjustments added using the algorithm found here: <http://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/>
* All colour controls now added: only question now is how to handle clipping

Alongside implementing a feature, unit tests would be written in what is the second part of the development cycle. Test driven development was initially adhered to, shown by the diary entry for week 3 of term 2:

Week 3 (25/1/2016 – 31/1/2016) …

* Began looking into testing, allowing me to start TDD and unit tests for existing code
* Configured WebStorm to use Karma to run QUnit tests (JetBrains, 2015), this took a day and was a lot of faff but nothing important to conclude ...
* Colour quantisation function in the model is the first to be written using TDD

Unfortunately the amount of code written with test driven development decreased as the project progressed, but rather tests were written at the end of the cycle. While vital features were tested, some remained untested.

While it was possible to write formal tests for most tasks in the model, those handling images were untestable because files uploaded through the HTML5 File element could not be replicated in tests for security reasons. The histogram rendering could not be formally tested either due to its nature, along with the GLSL scripts which performed some colour transformations. Manual inspections of the rendering had to be made to ensure it was correct. The sRGB space was easy to inspect as one could simply look to see if the vertices were the correct colours and those in between were interpolated correctly. It is also unlikely that the sRGB colours would be rendered incorrectly as there the framework works in sRGB. The CIE-L\*a\*b\* space was harder to test because the framework does not interpolate colours correctly so they must be defined pixel by pixel. By rendering a 2D plane at L=50, the result can be compared to Figure 4:1, a correct representation of the L\*a\*b\* space. GLSL is required to render in the CIE-L\*a\*b\* space without performance issues and while moving calculations to the GPU improves the program’s performance, it makes more code untestable, so there was a balance struck between testable and quick code.

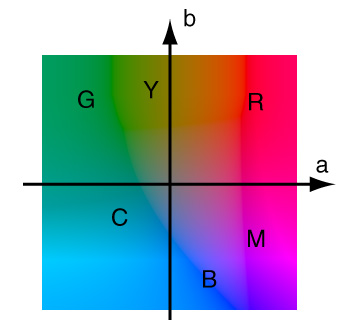


Figure : CIE-L\*a\*b\* at L=50

(http://www.arch.virginia.edu/~km6e/arch551/content/lectures/lec-05/page-2.html)

## Points of Note

The design specifies that the program be structured with the MVC design pattern. During development the design was adjusted while ensuring it was still MVC. The first week’s task was to restructure code written in previous months into the MVC design. Alex Netkachov’s web page (Model-View-Controller (MVC) with JavaScript, 2015) provided a JavaScript template of the MVC design and was used to restructure the project. At this point it was also noticed that the original design was not truly MVC. Tasks responsible for rendering the histogram should be moved from the model to the view, as they were concerned with representing data and not processing it: the diary entry for week’s 1 and 2 of term 2 shows evidence of this:

Weeks 1 & 2 (11/1/2016 – 24/1/2016)

* Prototype code was refactored into MVC using Alex Netkachov’s (2015) template and old (non-MVC) code achieved into “prototype” directory …
* Original DFD diagram had to be changed:
  + All rendering is now done in the view, as it regards the presentation of data and should have no effect on the representation of data in the model
  + Following on from above, a custom colour class was created, allowing me to define colours how I want, independently of the rendering framework, which should be isolated to the view

At the end of development the program was due to be optimised, a process that differed from the normal development cycle. Using Google Chrome’s in-browser profiler it was observed that displaying images on the web page was taking an unreasonable length of time, 24 to 25 seconds for 305 by 235 pixel image. Sampling a smaller number of pixels was tried but resulted in blocky images and fewer colours in the histogram. Instead, the values of all pixels were collected from the DOM at once instead of in many single pixel queries. This reduced computation time from 25 seconds to 0.6 seconds. After this the next longest task was instantiating colour objects. Currently, a new object is created for each pixel, which is wasteful. Alternative solutions include creating an object for each unique colour or representing an image’s colours as a 1D array of primitives. There was no time to explore these solutions but they are mentioned as future work in section 8.

# Description of the Final Product

In this section the design and function of the final product will be described. The original design in section 3 will be referred to when appropriate to avoid repetition. In section 6 this description of the final product will be evaluated against the design in section 3. A complete collection of the formulae used in the final product can be found in the appendices.

## System Architecture

The product was developed as a web page and the program written in JavaScript, implementing the Model View Controller (MVC) design pattern. An updated version of the UML diagram in Figure 3:1 is seen in Figure 5:1. The controller holds an instance of the model and view, allowing events fired by the view to be caught and methods called on the model to update its state accordingly. The view contains an instance of the model so that it can listen directly to events and update its representation of the data.

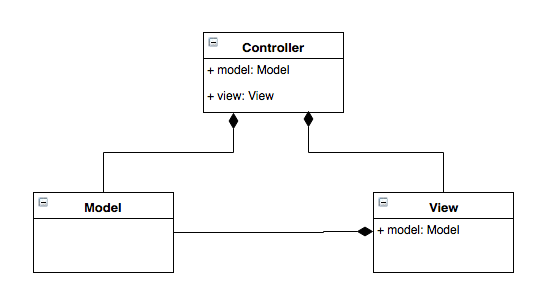


Figure : MVC UML Diagram

## Component Design

The program is divided into component tasks, which are grouped into the MVC design. The data flow diagram in Figure 5:2 is an updated version of that in Figure 3:2.

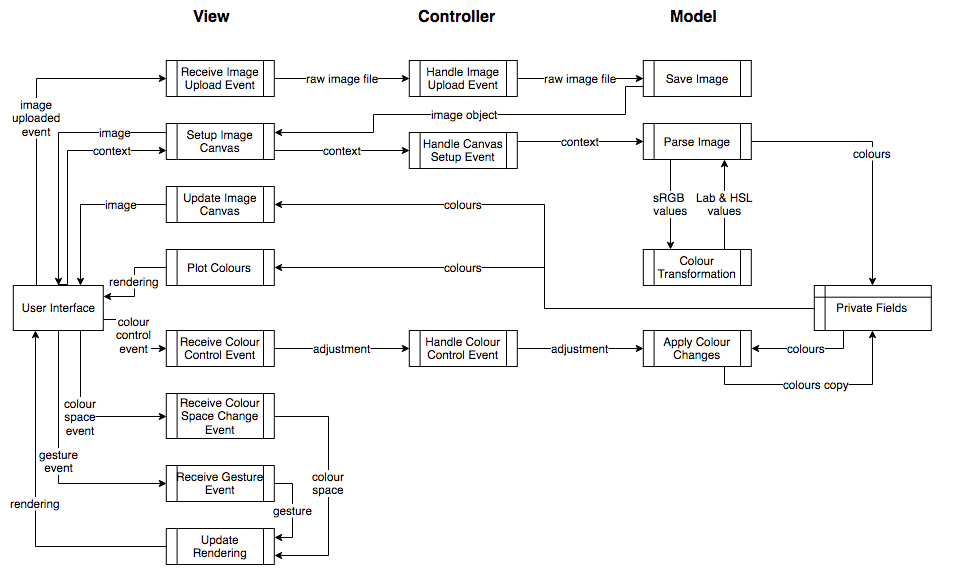


Figure : Data Flow Diagram

### Model

As well as holding state, the model is responsible for extracting colours from an image, converting sRGB colours to the CIE-L\*a\*b\* and HSL spaces, and applying colour alterations. The model also fires events when its state changes.

While the view converts an image into an appropriate format (see 5.2.2), the model’s Parse Image task extracts colours. The model receives the image canvas’ context, which can be queried to find the colour of each pixel. A colour object is created to represent each pixel and the sRGB colour is also converted into the L\*a\*b\* and HSL spaces for later use. The list of colours stored as a private field of the model.

Colour transformations are implemented with various formulae. The formulae used to convert between sRGB and HSL are the same as those in Figure 3:7 and Figure 3:8. Formula for transforming from CIE-XYZ to CIE-L\*a\*b\* are as defined in Figure 3:5 and the formula to transform from sRGB to CIE-XYZ, which differs from the original design, is seen in Figure 5:3.

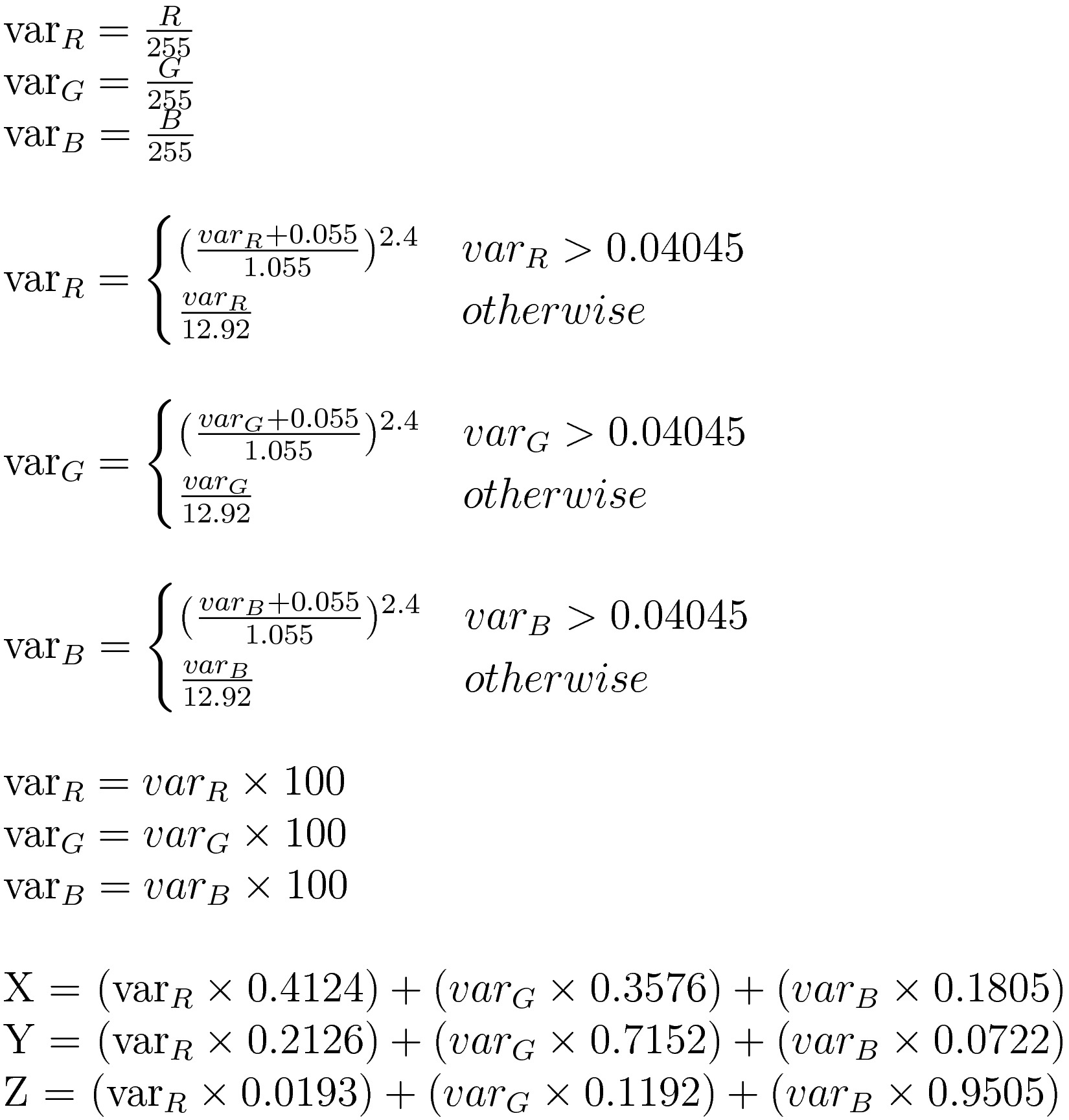


Figure : sRGB to CIE-XYZ formula

Colour alterations are applied in the Apply Colour Changes task. Brightness is applied as explained in 3.3.1 and the formulae for adjusting contrast and saturation are as they are in Figure 3:6, Figure 3:7, and Figure 3:8. An extra step is taken after these formulae are applied to clip colour channels that fall outside of valid ranges to the nearest limit. Altered colours are saved separately to the original colours because colour adjustments are not reversible if clipping occurs and all colour alterations are applied to the original unaltered colours.

### View

Tasks in the view either receive events from the user interface or update the representation of the model’s data, which in this case is the histogram, rendered using the WebGL and Three.js JavaScript libraries.

Most tasks that receive events from the user interface fire another event that is caught by the controller. An exception to this is the Receive Colour Space Change Event and Receive Gesture Event tasks that do not affect the model’s state so instead update the user’s view of the histogram locally within the view. Other tasks that are responsible for updating the user’s view of the image and its histogram are Update Image Canvas and Plot Colours, which listen to events fired by the model when and update the view’s representation of the model’s data.

The final responsibility of the view is converting the image into a format from which the value of pixels can be queried. The Setup Image Canvas task receives an Image object from the model and puts it into an HTML5 Canvas element. As well as displaying the image to the user, the context of the canvas allows the values of pixels to be inspected and is passed back to the model.

### Controller

The controller contains an instance of both the model and the view. Tasks in Figure 5:2 depict where the controller listens to events from the view and passes them onto the model. The controller does not however handle communications from the model to the view because the view listens directly to the model.

## User Interface

The user interface is a web page, the original design of which is seen in Figure 3:9 and the final result in Figure 5:4. The histogram is in the centre of the page in front of a neutral grey background; the image is to its left and a section containing colour controls, the colour space radio button, and a file upload button runs along the top of the page.

The button labelled “Choose File” allows the user to upload an image from their local file system, which is then displayed on the canvas on the right. The colour space of the histogram can be changed with the radio button. The brightness, contrast, and saturation of the image can be changed with the sliders, causing the image and its histogram to update. Finally the user can gesture on the histogram, dragging with the left or right mouse button held and scrolling to rotate, pan, or zoom.

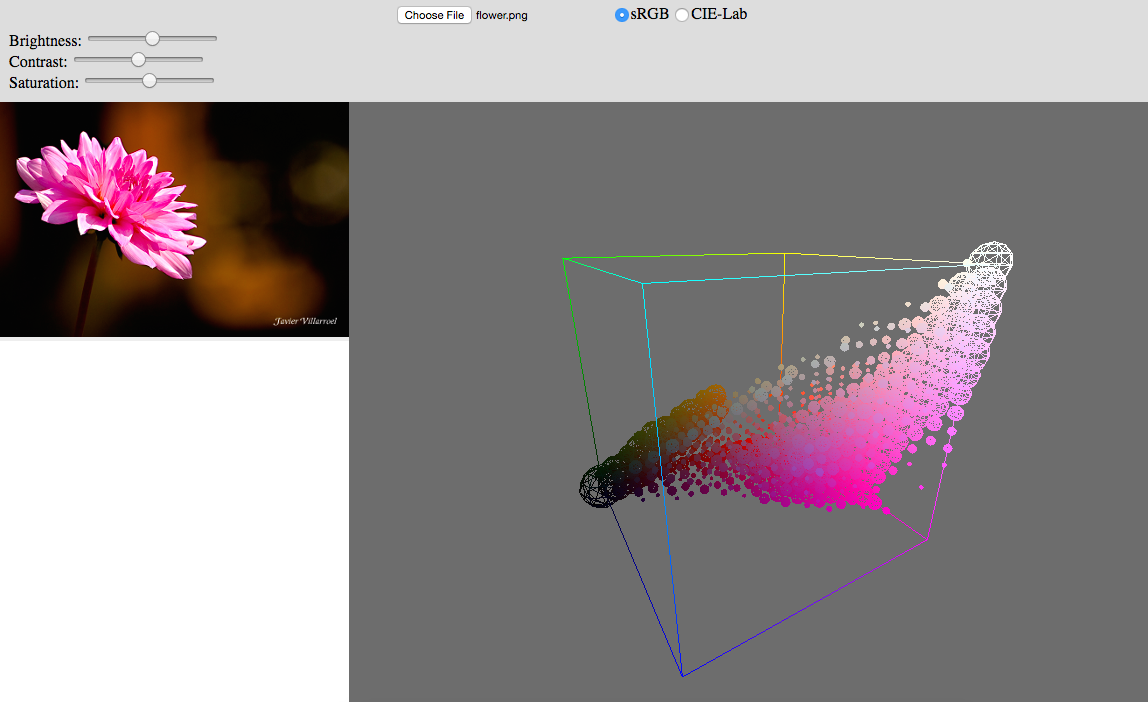


Figure : User Interface

## Product Behaviour and Performance

When first loaded, the web page will appear with a blank canvas on the left and an empty histogram on the right. In the sRGB space by default, the histogram will appear empty but the outline of the space will appear in front of a neutral grey background, chosen to display colours as accurately as possible. 3 edges of the cube (those going from black to red, green, and blue) represent the 3 axes of the colour space, but the additional edges are added for completeness. The CIE-L\*a\*b\* colour space is not a cube like sRGB but rather a cylinder. The limits of L\*a\*b\* are less well defined so only the coordinate axes are drawn. The user is able to move the camera in the rendering by left or right-click dragging, and scrolling, which rotate, pan, and zoom respectively.

The first action the user takes is uploading an image by clicking the button labelled “Choose File”. This prompts a window to open, allowing the user to browse their local file system and select a file. Once confirmed, the window closes and the uploaded image will appear in the canvas on the left and the histogram in the canvas on the right. This will take a few seconds.

Each axis of the histogram is divided into discrete bins. If there are colours that fall within a bin a wireframe sphere is rendered on the centre of that bin and coloured accordingly; the more points that fall within a bin, the large the plot becomes.

While initially in the sRGB space, the histogram can also be viewed in L\*a\*b\* by changing the selection of the radio button. This will redraw the histogram axes in the new coordinate system along with all of the plots within. This operation should be quick because the colours are transformed into L\*a\*b\* space in advance.

Finally the brightness, contrast, and saturation of the image can be adjusted with the sliders. Initially in the centre position, moving a slider will cause the image and its histogram to be redrawn with the appropriate colour alterations applied. The distribution of points within the histogram can be seen to move and the appearance of the image change. This process will take a while.

# Evaluation of the Final Product

In this section the final product, outlined in section 5, will be evaluated by comparing it against the original design in section 3 and using it to attempt to resolve the questions posed in 3.1. At the points where the final product departs from the original design, changes will be justified and shown to still meet the project requirements.

## Basic Requirements

The final product meets all of the basic requirements defined in the design: it allows the user to upload an image, view its histogram in the sRGB and CIE-L\*a\*b\* spaces, adjust the image’s brightness, contrast, and saturation, and view the effects of these changes on the image and its histogram.

## System Architecture

This is a comparison between the system architecture design defined in 3.2 and the system architecture of the final product, described in 5.1. The final product implemented the MVC design pattern but some changes were made. Unlike in Figure 3:1, Figure 5:1 shows that the view contains an instance of the model. This change occurred as I learnt more about how to use the MVC pattern, but the current design can still be improved. After reviewing my code it is my opinion that user interaction should go into the controller, not the view. While this change has no immediate effect on the program’s behaviour, if the project were to progress this change would prevent problems arising later.

## Component Design

Here I compare the design in 3.3 and the actual system in 5.2. The final product contains all of the tasks outlined in the design but some have been moved and reorganised.

Tasks responsible for rendering the histogram were moved from the model to the view: I made this decision after coming to the conclusion that as a representation of the data, the histogram did not belong in the model. Effects of this reorganisation were that colours must be fetched by the view from the model and the histogram’s colour space is stored as state in the view.

In the design, the model was responsible for converting raw images into collections of pixels whose values could be queried. After I discovered that the HTML5 canvas element would perform this conversion I moved this task to the view. This change is not in accordance with the MVC design because it places data processing in the view but given the convenience of the canvas element I decided this was an acceptable change. There is an artefact of the previous design left in the model: the Save Image task takes a raw image file from the view, wraps it up in a JavaScript Image object, and sends it back. This task is data processing so traditionally would reside in the model but it is questionable whether it should be moved to the view along with the rest of the tasks that it interacts with. This is a weak point in the design of my final product.

The design specifies that the conversion between sRGB and CIE-XYZ is done using a linearisation function followed by a transformation matrix (see Figure 3:3 and Figure 3:4) but during development I discovered a simpler way to represent this operation: Irotek’s web page (Irotek Group Ltd., 2014) defines a formula for converting from sRGB to CIE-XYZ, seen in Figure 5:3. Initially I was unsure whether the two were equivalent but as shown in Figure 6:1, the formula is mathematically identical to a linearisation function followed by a matrix multiplication.



Figure : sRGB to CIE-XYZ formulae equality

## User Interface

The user interface of the final product contains all of the features specified in the design but the structure of the page has been changed and the quality is not what I had hoped for. The histogram and image are adjacent, as in the design, but all controls now reside along the top of the window: while I would like to reorganise the UI to match the design this change does not have a huge effect. The quality of the UI is however, poor: the divisions holding the image and histogram do not resize to fill the web page while the top panel does, making for an uneven page. The page does not look attractive and merely exists to serve a purpose. This is the case because I prioritised the program over its interface and I do not regret this decision because the UI was not the focus of this project.

## Product Behaviour and Evaluation

The final product contains all the features defined in the design, but the project can not be evaluated solely on this criteria and it must also be decided whether the product allows me to answer the initial questions asked in 3.1: do colour alterations have a significant and useful effect on an image’s histogram, in a way unique to other visualisations in image editing? Impartial members of the target audience would ideally conduct the evaluation of my product but because my target audience is small and there wasn’t sufficient planning of the evaluation I have had to do it myself.

Applying colour alterations to an image and deciding whether there is a significant affect on the histogram will answer the first question and the second question can be answered in two parts: firstly whether the effects on the histogram are useful and secondly whether the 3D colour histogram has any advantage over its 2D counterpart. During development I used the same image of a flower so I will use an unseen image for the evaluation to ensure that the program is not tailored towards a one case. The image I chose was the photograph of Lena Söderberg, a famous test image in image editing. The unedited image and its histogram are shown in Figure 6:2. Figure 6:4 through to Figure 6:8 show the effects of increasing and decreasing brightness, contrast, and saturation. I have only used the sRGB space in these examples as it can be compared to 2D histograms but the same evaluation can be applied to L\*a\*b\*.

When comparing the edited histograms to unedited one it is clear that the histogram is significantly affected by the colour alterations. This answers the first of my two questions: colour alterations do have a significant affect on an image’s histogram.

To answer the second question the effects of colour alterations must be examined more closely. I observed that adjusting brightness moves colours towards the white and black vertices of the colour space. Varying contrast levels pushes colours towards the extremes of the colour space and pulls them into the centre; at first glance this is a similar effect to saturation but on further inspection one can see that contrast repels and attracts colours to and from the centre point of the colour space while saturation attracts and repels colours towards and from the white-black diagonal. These observations make sense in the context of the operation causing them and because the effects are I conclude that the histogram provides a meaningful perspective to these operations.

Finally, my 3D histogram can be compared against a 2D histogram to determine whether the third dimension provides a unique perspective. I am comparing my program against Apple’s iPhoto (Apple Inc., 2015), which provides exposure (brightness), contrast, and saturation controls and a 2D colour histogram. On the 2D histogram, adjusting brightness shifts the three colour channels towards the ends of the scale; varying contrast spreads and contracts colour channels along the x-axis, while maintaining the outline shape of each plot; and saturation spreads or contracts the colours along the x-axis, separating or coalescing the three plots. These observations are harder to relate to the operations causing them than in the 3D space and by comparing these observations to those made on my 3D histogram I conclude that the 2D histogram can show how channels change in relation to each other and the exact luminance level of a colour can be identified; the 3D histogram however offers the ability to view individual colours and the distribution of the colour. It is not possible to accurately represent sRGB in 2 dimensions because it assigns a primary colour to each axis, so a 3D visualisation is necessary to fully understand the space. CIE-L\*a\*b\* can be visualised in 2 dimensions by fixing the lightness value (as seen in Figure 4:1) but while this representation may be accurate it looses information by omitting the third axis. L\*a\*b\* is also a minority and most spaces can not be accurately represented in 2D, meaning a 3D histogram is necessary to accurately represent most space.

|  |  |
| --- | --- |
| Macintosh HD:Users:callum:Desktop:Lenna_0.png  Figure : Lenna Uneditted | |
| Macintosh HD:Users:callum:Desktop:Lenna bright max.png  Figure : Increased Brightness | Macintosh HD:Users:callum:Desktop:Lenna bright down max.png  Figure : Decreased Brightness |

|  |  |
| --- | --- |
| Macintosh HD:Users:callum:Desktop:Lenna cont up.png  Figure : Increased Contrast | Macintosh HD:Users:callum:Desktop:Lenna contr down.png  Figure : Decreased Contrast |
| Macintosh HD:Users:callum:Desktop:Lenna sat up.png  Figure : Increased Saturation | Macintosh HD:Users:callum:Desktop:Lenna sat down.png  Figure : Decreased Saturation |

Another observation that I made was that unexpected colours would often appear at the edges of the colour space. I attribute these anomalies to the naïve clipping algorithm that clips channels to the nearest limit if they fall outside the valid range. Improving the clipping method could reduce these unexpected colours and improve the quality of the colour adjustment controls. An alternative clipping method scales each channel towards the centre of the space proportionally to its size until all channels are within the limits. I did not have time to experiment with clipping during development but it is an area that I would like to explore in the future.

I had initially hoped that the plotting of an image’s histogram and performing colour alterations would be done in real time but this was not achieved. I prioritised creating a working product before optimising, which I think was a wise decision, and made efforts to improve performance at the end of development. In the case of extracting colours from the image I was able to reduce the calculation time down from 25 seconds to 0.6 seconds. I have had more ideas of how to optimise the program and would like to investigate them in future work.

# Critical Assessment of the Project as a Whole

This section of the report will not evaluate the quality of the product, but rather the quality of the work throughout the entire project.

## Pre-Development

During the first term I completed the literature review and specification and design document. The time plan that I produced as part of the project statement and plan helped keep my work on track. Thanks to my previous experience in the area I was able to start my literature review with a number of sources already lined up. It would have been easy to become complacent with the knowledge and experience that I already had but I pushed myself to explore new areas and better understand those that I was familiar with. At the end of my research I had compiled and analysed sources on a variety of topics. I regret the lack of literature sources, as all of my sources were either web pages or online journal entries. The specification and design document was the second piece of work I would submit before I began development. This document reflected both the results from my literature review and targets sets out in my initial project statement. The level of detail was appropriate for a design at that stage, allowing the reader to interpret the design himself or herself independent of any particular implementation. Both the information I gathered in my research and the project design put me in good stead for starting development.

During this time I also wrote some preliminary code with the aim of supporting my literature review and design, allowing me to come to more informed conclusions. The code was based on the work I had done over the previous summer and there was little new material so progress was quick: however the code was written in an ad-hoc manner and not tested. As a result the program was poorly structured and contained bugs, meaning it was difficult to build upon. Intervention had to be taken at the beginning of development to insure that the code reached an acceptable standard. In hindsight, I should have planned better from the beginning and refrained from rushing into writing code.

## Development Cycle

The start of term 2 marked the beginning of development. The specification and design document’s time plan (see Figure 3:10) spanned the final stages of my project and focused on breaking development down into short and iterating cycles: a summary of the methodology can be found in section 3.6 and evidence of how it was executed in section 4.1. I found that short development cycles made the project more manageable and kept my mind focused, as each week I had a single goal to accomplish. Testing in each cycle also gave me constant reassurance that the program was correct and removed the need for a large testing and debugging period at the end of development. Finally, reflection at the end of each cycle allowed me to assess my progress and adjust my time plan accordingly.

In hindsight I did not put enough emphasis on the review in each cycle and I should have used the time to reassess the design. An example of where this would have been beneficial is in my data flow diagrams. As discussed in section 6.2 and 6.3, the structure of my project changed over time; while this itself was not a problem I did not give enough thought to how the project was changing as I went along and consequently at the end of development there were aspects of the project’s structure that were not in accordance with the MVC design. I was however able to reflect at the end of each cycle in my diary. The diary was intended for retrospective analysis to help in the writing of this report but it also provided me with a chance to pause and reflect. While not intentional, this diary benefitted my development.

## Testing

As specified in section 3.5, my code was to be tested with unit tests and the test driven development paradigm. This was a good decision because it incorporates testing into the development and forces complex systems to be broken down into small and manageable components. While initially followed, I often got distracted and forget to test until the end of a cycle. The tests that I wrote covered a wide range of scenarios, focusing on edge cases, and included testing with invalid data. While the tests were good, if I had kept to test driven development, better code would have been produced.

I also regret not doing more research into testing frameworks. The framework that I chose, QUnit, lacked functionality that I required and left holes in my tests. For example, in order to test whether a value is true with a small tolerance I had to define my own test method. Some parts of my program were inevitably going to be hard to test, such as the rendering, but perhaps these tests would have been possible with a different testing framework. In summary, QUnit was not a wise choice of testing framework and more research should have been done.

The parts of my project that were untestable were tested manually. The process of convincing myself that the program was rendering colours correctly (discussed in section 4.1) was only that, convincing myself that it was correct. While formally testing the rendering may not be a realistic goal, more thought should have been put into how the rendering could have been tested.

## Time Plan

I set myself the goal of completing development by Easter so that the remaining time could be spent writing this report and preparation for the poster session and demonstration. I knew from experience that estimating the time needed to complete software development tasks was difficult so I was generous and was prepared to adjust the schedule if necessary. It was not necessary to change the time plan and in section 4.1 I discuss how I feel behind schedule a few times but managed to get back up to date. While this success was party due to a good work ethic, in hindsight the time allocated for certain tasks was overly generous and could have been better spent improving the performance of my program or exploring areas of future development.

## Results and Evaluation

I am pleased with the results that my project has yielded: in section 6.5 I was able to conclude that the histogram provides a unique and useful perspective on colour distributions and the effects of colour manipulation. The one caveat to this evaluation is that I was the only one to come to these optimistic conclusions. The target audience (defined in 3.1), whilst broad, is restricted enough that I was not able to gather anyone impartial to evaluate my project. Should I start again I would not change my project’s requirements or goals to accommodate for a wider audience as I believe this project to be interesting, but I would consider my method of evaluation more seriously from the beginning and gather a small group of people to help evaluate the project.

# Future Work

While I have successfully implemented all the features in the design there are still improvements that can be made and other avenues to explore. In this final section of the report I will discuss where I see this project going in the future.

In its current state there are some areas of the project that could be improved. As discussed in section 5.3, the user interface was not a priority and is consequently poorly designed. If I were to publish the project online the user interface would need work to get it to the necessary standards. The performance of the program is another issue that needs attending. Currently uploading an image and adjusting its colours can take an unreasonably long time. Ideally the project would work in real-time so that the user experience is as smooth and uninterrupted as possible. I have begun to optimise the program but there is a lot more that could be done, as discussed in section 6.5. Finally, I also discussed in section 6.5 how the naïve clipping method sometimes causes unexpected colours to appear. The aim of my project was not to implement colour adjustment controls so I am not overly concerned about this, but now that the goal of my project has been met I can begin to explore other avenues, such as how to better implement these tools. Other avenues also opened themselves up to me during development, such as how to best plot colours on the histogram and whether to give the user control over properties such as the opacity and shape of plots. These ideas were not essential to the project at the time but now can be revisited.

Looking beyond the project in its current state, I see the possibility for 3D colour histograms to be used as tools of image editing and not just visualisations. My project was a prototype inspired by the question of whether histograms offer a unique perspective into colour alterations and the results suggest that they do, but could this be carried on to develop them into tools as well? Evaluation of the project in 6.5 show how different colour operations effect the histogram in different ways but what they appear to have in common is moving the distribution of points towards and away from certain areas of the colour space. Would it perhaps be useful to provide attraction and repellent controls on the histogram to directly affect the colours in the image? For the operations provided in this project (brightness, contrast, and saturation) there is little point in this as they are easily to manipulate with conventional controls, but there is the potential for more versatile and varied colour controls using the histogram. Such controls would allow more complex operations to be performed as well providing a good visualisation of the colour distribution.

# Conclusion

During the course of this project I have developed a comprehensive understanding of the different colour spaces and this knowledge has been applied with the skills gained in using 3D rendering libraries to generate a complete working program. Being able to use the tool I had developed to answer the questions that inspired the project was satisfying, but the conclusion that colour alterations do have a significant and useful effect on an image’s histogram is not the end of the story.

The positive results from this project have inspired me to think further into how the effects of colour alterations on histograms can be used in image editing software. Currently, image editing software provides visualisations such as 2D colour histograms, which are useful to identify when colours are being clipped, and cross sections of the CIE-L\*a\*b\* space, which show the distribution of colours. My 3D colour histogram allows the user to identify clipped colours as well as see the distribution of colours without loosing information by reducing the number of dimensions. While a 3D rendering is more complex I can see my project having potential in image editing, allowing users to better understand where the colours in an image lie within the colour space and the effects of their colour alterations.

From the start of the project I was interested in the application of histograms as a tool of colour alterations, allowing the user to directly manipulate the colours that they see in the rendering. Now that the project is drawing to a close this potential is on my mind again and I am interested to whether this kind of tool would be appreciated by members of my project’s target audience. But as well as using the histogram as a tool, many avenues of further work have opened up to me as I have progressed through the project: among them are a variety of different colour controls, variations in how these are implemented, different colour spaces, and more control over the histogram’s aesthetics. Each time one of these potential ideas came up I had to suppress the temptation to pursue it otherwise I would not have been able to complete the project on schedule, but now I am free to explore these new and exciting directions.

# Bibliography

Apple Inc. (2015, March 19). iPhoto. *9.6.1*. Apple Inc.

Barthel, K. U. (2007, September 16). *3D Color Inspecter/Color Histogram.* Retrieved November 12, 2015 from Research Services Branch: http://rsb.info.nih.gov/ij/plugins/color-inspector.html

Caballero, M. L., & Belmonte, N. G. (n.d.). *Real time color 3D histogram analysis*. Retrieved October 3, 2015 from Senchalabs: http://www.senchalabs.org/philogl/PhiloGL/examples/histogram/

Cruse, P. (2015, June 12). *Introduction to Colour Spaces.* Retrieved October 2, 2015 from Colour Phil: http://www.colourphil.co.uk/lab\_lch\_colour\_space.shtml

Esterhuizen, D. (2013, April 20). *C# How to: Image Contrast*. Retrieved January 6, 2016 from Software By Default: http://softwarebydefault.com/2013/04/20/image-contrast/

Hoffmann, G. (2015, September 24). *CIE Color Space.* Retrieved November 5, 2015 from docs-hoffman: http://docs-hoffmann.de/ciexyz29082000.pdf

Hoffmann, G. (2015, September 24). *CIELab Color Space.* Retrieved November 5, 2015 from docs-hoffmann: http://docs-hoffmann.de/cielab03022003.pdf

Hurkman, A. H. (2013, May 20). *What's In a Name?* Retrieved November 9, 2015 from Pro Video Coalition: http://www.provideocoalition.com/whats-in-a-name

Irotek Group Ltd. (2014). *Color conversion math and formulas*. Retrieved October 2, 2015 from EasyRGB: http://www.easyrgb.com/index.php?X=MATH&H=07#text7

JetBrains. (2015, December 23). *Creating JavaScript Unit Tests*. Retrieved January 20, 2016 from JetBrains: https://www.jetbrains.com/webstorm/help/creating-javascript-unit-tests.html

Juckett, R. (2010, May 16). *RGB Color Space Conversion*. From Ryan Juckett: http://www.ryanjuckett.com/programming/rgb-color-space-conversion/

Khoronos Group. (2015, April 26). *Main Page*. Retrieved November 5, 2015 from WebGL Wiki: https://www.khronos.org/webgl/wiki/Main\_Page

Liao, E. (2013, April 18). *Uniform Color Quantization*. Retrieved November 8, 2015 from Github: https://github.com/rcliao/MultimediaSystem/wiki/Uniform-Color-Quantization

McHugh, S. (2015). *Camera Histograms: Luminosity & Color*. Retrieved October 3, 2015 from Cambridge in Colour: http://www.cambridgeincolour.com/tutorials/histograms2.htm

Meskaldji, K., Boucherkha, S., & Chikhi, S. (2009). Color Quantization and its Impact on Color Histogram Based Image Retrieval. In O. A. Abdelmalek Amine (Ed.), *Conférence Internationale sur l'informatique et ses Applications.* *547.* Algeria: http://ceur-ws.org.

mrdoob. (n.d.). *ShaderMaterial*. Retrieved October 2, 2015 from three.js: http://threejs.org/docs/#Reference/Materials/ShaderMaterial

mrdoob. (2015, September 22). *three.js*. Retrieved September 27, 2015 from three.js: http://threejs.org/

Netkachov, A. (2015). *Model-View-Controller (MVC) with JavaScript*. Retrieved January 11, 2016 from Alex@Net: https://alexatnet.com/articles/model-view-controller-mvc-javascript

Nikolai. (2013, May 9). *Color Decision List - introduction*. Retrieved October 2, 2015 from Waldman Media: http://www.niwa.nu/2013/05/color-decision-list-introduction/

Parisi, T. (2010, December 6). *The Lessons*. (T. Parisi, Editor) Retrieved October 2, 2015 from Learning WebGL: http://learningwebgl.com/blog/?page\_id=1217

PracticalPhotographyTips. (n.d.). *Color Histograms*. Retrieved October 31, 2015 from Practical Photography Tips: http://practicalphotographytips.com/Digital-Camera-Basics/color-histograms.html

RapidTables. (2015). *HSL to RGB color conversion*. Retrieved January 6, 2016 from Rapid Tables: http://www.rapidtables.com/convert/color/hsl-to-rgb.htm

RapidTables. (2015). *RGB to HSL color conversion*. Retrieved January 6, 2016 from Rapid Tables: http://www.rapidtables.com/convert/color/rgb-to-hsl.htm

Segenchuk, S. (1997, May 5). *An Overview of Colour Quantization Techniques*. Retrieved November 7, 2015 from Worcester Polytechnic Institute: http://web.cs.wpi.edu/~matt/courses/cs563/talks/color\_quant/CQindex.html

Steer, W. A. (2008, March 15). *Introduction to Colour Science*. Retrieved November 5, 2015 from Techmind: http://www.techmind.org/colour/

Third Ave Design, T. S. (n.d.). *3D Histogram*. Retrieved October 3, 2015 from 3D Histogram: http://3dhistogram.com/?#contact

Villarroal, J. (2013). *H3Stogram - 3D Interactive Color Histogram*. Retrieved September 27, 2015 from H3Stogram: http://h3stogram.herokuapp.com/

# Appendices

## Project Diary

### Weeks 1 & 2 (11/1/2016 – 24/1/2016)

* Prototype code was refactored into MVC using Alex Netkachov’s (2015) template and old (non-MVC) code achieved into “prototype” directory
* Included a custom event class that allows listeners to subscribe to an event and receive notifications with data when the events fire: other implementations of events involved sending notifications through the DOM, which I thought was messy and not true to MVC
* MVC required me to start using JavaScript objects/class, which use a prototype model
* Original DFD diagram had to be changed:
  + All rendering is now done in the view, as it regards the presentation of data and should have no effect on the representation of data in the model
  + Following on from above, a custom colour class was created, allowing me to define colours how I want, independently of the rendering framework, which should be isolated to the view

### Week 3 (25/1/2016 – 31/1/2016)

* Prototype is now completely refactored into an MVC design
* Began looking into testing, allowing me to start TDD and unit tests for existing code
* Configured WebStorm to use Karma to run QUnit tests (JetBrains, 2015), this took a day and was a lot of faff but nothing important to conclude
* Tests written for some existing code, mostly easy to test functions such as the utils and model
* Some functions are noted as being untestable, mostly due to being unable to simulate file inputs by loading a file from local file space: I intend to either modify them to become testable or to define manual tests that can be used instead
* Colour quantisation function in the model is the first to be written using TDD

### Week 4 (1/2/2016 – 7/2/2016)

* Ran into confusion with the values in my LUT and during the conversion between 24-bit and 16-bit
* Resolved confusion: multiply 16-bit value to get 24-bit value
* I initially thought it was a factor of 16 (16 \* 16 = 256) and while this maps 0 to 0, and 1 to 16, 15 (16-bit max value) maps to 240
* I knew the 24-bit range would be divided into 16 ranges (256 / 16 = 16) but I believed these ranges would have a size of 16
* Instead they have a range of 17: 255 / 17 = 15 divisions, plus 1 from 0 to 17, totalling 16 divisions with a range of 17
* Confusion resolved, new code written, and tested

### Week 5 (8/2/2016 – 14/2/2016)

* Noted that the small number of dominant colours were overpowering the majority, making them difficult to see
* Took natural logarithm of values to prevent smaller values from being dwarfed
* It worked, however when a bin had only 1 entry, the size of the plot would be 0 (ln(1) = 0) and the plot is drawn very big for some reason
* These 1 value colours will have to be either ignored or increased by 1. But they are so small that this will hardly be noticed
* I decided to use ln(x+1), which passes through the origin, ensuring that y is only 0 when the value is 0. It does skew values slightly, but I thought this was better than omitting all results with a value of 1
* If a sphere radius is below the minimum, do you omit it or set it to the minimum radius?

### Week 6 (15/2/2016 – 21/2/2016)

* Used simplified colour transformations (found at <http://www.easyrgb.com/index.php?X=MATH&H=07#text7>) to convert from RGB to Lab
* Refactored ColorRGB class to incorporate both RGB and Lab values into a single object
* New Color class also includes the code to convert between colours spaces
* Decided not to port code for colour quantisation into Color class too, because the colours should be stored in max resolution (24-bit) and only quantised when they are to be displayed
* Note: Look into those things that Richard mentioned in the graphics lecture for detecting which parts of code are taking the longest
* Wrote tests for the colour space transformations using test values obtained from same site as formulae
* Is this simple formula accurate enough compared to matrix transformations?
* The owners of the website assure me they are equivalent but I may conduct further tests to assure myself of this
* But for now I will trust their word and continue with development

### Week 7 (22/2/2016 – 28/2/2016)

* Implemented CIE-L\*a\*b\* to sRGB conversions
* More extensive tests are necessary for the conversions
* Began rendering Lab colour space cube, realised that fragment shaders are now necessary are colours are interpolated incorrectly by WebGL
* Rendered RGB cube using shaders (previously done with a series of JS functions) and is now much neater
* Lab cube also rendered using shaders
* However transformations had to be written in GLSL, which is nearly impossible to test, but the computational and structural advantages outweigh this disadvantage
* Changed RGB colour quantisation method to use floating point arithmetic rather than LUTs, which were overkill
* This is opens up the opportunity to perform quantisation within the Color class
* Colour quantisation inside Color class works well and removes complexity in model
* However lab values are not being plotted correctly
* Tests have been written for easily tested functions, but other things are harder to test with unit tests, namely the rendering
* Renderings still require to be tested by hand
* Found a silly bug that was colouring Lab colours incorrectly (while they were in the correct locations): I was using the rgb shaders, not the lab shaders
* Reformatted code so colour space changes work: included adding a static variable to Color
* Refactored Color class so that variables and methods can be private, giving the code more of a OO feel, which I am more familiar and comfortable with
* It also makes testing easier and helps assure that the class is being used correctly

### Week 8 (29/2/2016 – 6/3/2016)

* Antony suggested loosing the cube for the CIE-Lab rendering as the axes are adequate, appropriate, and the cube misleading
* He also suggested the idea of having two independent renderings, side by side, one in RGB and the other Lab
* Limited the OrbitControls’ controls to the div containing the canvas
  + This firstly allows the user to use slide controls (i.e. brightness etc.) without moving the rendering
  + And secondly allows two renderings to be shown side by side
* Implemented UI controls for colour tools in the form of sliders (note: these will not work with IE9 and below)
* Added an HSL field to the Color object, allowing which is to be used to adjust the brightness and saturation of colours
* Brightness can now be changed using the sliders and the effects seen in the histogram
* Results take a number of seconds to calculate, so efficiency and optimisation will certainly need to be addressed
* This website (<http://colorizer.org/>) shows that changing L in Lab changes L in HSL pretty independently, but changing L in HSL changes both a and b in Lab as well as L: this is unexpected and also observed in my visualisation… worth investigation of the transitivity of these transformations!
* I am currently guessing that the range of L in HSL is 0 to 1, and this assumption gives expected results

### Week 9 (7/3/2016 – 13/3/2016)

* Little progress made
* Discussed how to continue for the last few weeks of development
* Prioritise usability and being able to see results and make conclusions from the visualisation
* Make it so that the image’s colours change as those in the histogram do
* Also implement last two colour controls
* Finally, if time, make the histogram quicker
* Secondly, adjusting brightness results in strange artefacts (extreme colours) to appear
* Are these a result of a poor brightness algorithm? Find out by changing the colour of the image
* Can I deal with clippings better?

### Week 10 (14/3/2016 – 20/3/2016)

* The image now changes colour with the histogram
* Increasing brightness almost looks like increasing saturation: very bright colours emerge/colours disappear into black
* Saturation adjustment now also works, using HSL in the same way to brightness
* Noticed that I wasn’t ensuring the S and L values were inside the 0 to 1 range when applying the adjustment: clipping to these ranges appears to give more sensible colour changes
* Contrast adjustments added using the algorithm found here: <http://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/>
* All colour controls now added: only question now is how to handle clipping
* Another useful thing to do would be to plot all the colours in a colour space to view its shape
* I decided it would be easier to simple find an image that contained all RGB colours
* To achieve this I had to fix my webpage so that it didn’t clip the input images to 200x200 pixels
* This worked well and I am able to see the shape of sRGB in Lab, and it is very interesting: not a double-ended cone but more of a squashed and off-centre rhombus
* However in the process of doing this I discovered that images larger than 200x200 took a long time to extract colours from, considerably longer than I had expected
* It is therefore even more important that I now look at how I can make this more efficient

### Week 11 (21/3/2016 – 27/3/2016)

* Began using Google Chrome’s profiler to determine which parts of my program are taking the longest
* I started by trying to optimise the initial uploading of an image and extracting colours
* I soon discovered that it was the getImageData that was taking up the most processor time: 24,000 to 25,000 ms for a 350x235 image
* I tried sampling a quarter of all pixels rather than every one: while this brought the computation down to about 6,500ms it wasn’t ideal and resulted in blocky images
* After a little Googling I realised that it was the getImageData call on the DOM object that was taking the longest, so I decided to get all the pixels in one getImageData call and iterate over the resulting array
* This brought the computation down to a blinding 66.1ms
* The same idea was applied to the putImageData call in the view
* The most expensive process is now the garbage collector, followed by definitions of Color objects: this leads me to believe that the best way to improve the performance of my program now is to reduce the number of objects declared and/or make their instantiation more efficient
* I will also look into what processes are the most expensive when adjusting an image’s colour
* Discussions with Antony in our weekly meeting brought up the topic of how I will evaluate my project in my final report
* I think I will need to drop the idea of a hypothesis and rather present it as a general idea, as a hypothesis requires more detailed testing than I can conduct: in particular a definition of what constitutes a failure

## Colour Space Transformation and Colour Alteration Formulae

|  |  |
| --- | --- |
| CIE-XYZ to CIE-L\*a\*b\* formula (Irotek Group Ltd., 2014) | sRGB to CIE-XYZ formula (Irotek Group Ltd., 2014) |
| Contrast formula (Esterhuizen, 2013) |  |
| sRGB to HSL formula (RapidTables, 2015) | HSL to sRGB formula (RapidTables, 2015) |