3D Colour Histograms

Callum McGregor

# Abstract

Abstract body.

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

1.1 Project Statement and Motivation 4

2 Summary of Literature Review 4

2.1 Introduction 4

2.2 Colour Spaces 4

2.3 Colour Histograms 5

2.4 Technologies 6

2.5 Colour Quantisation 6

2.6 Colour Adjustments 7

3 Summary of Specification and Design 7

3.1 Project Definition and Requirements 7

3.2 System Architecture 7

3.3 Component Design 8

3.4 UI Design 9

3.5 Testing and Evaluation 9

3.6 Time Plan 10

4 Development 10

5 Testing 14

6 Description of the Final Product 15

6.1 System Architecture 15

6.2 Component Design 15

6.2.1 View 16

6.2.2 Model 17

6.2.3 Controller 18

6.3 User Interface 18

7 Evaluation of the Final Product 19

7.1 Introduction 19

7.2 Requirements 19

7.3 System Architecture 20

7.4 Component Design 20

7.5 User Interface 21

7.6 Experimentation 21

7.7 User Testing 21

8 Critical Assessment of the Project as a Whole 21

8.1 Pre-Development 21

8.2 Development Cycle 22

8.3 Time Plan 22

8.4 Testing 22

8.5 Conclusion 23

9 Future Work 23

10 Conclusion 23

Bibliography 24

Appendices 27

# Introduction

## Project Statement and Motivation

Over the summer of 2015 I worked on the visualisation and comprehension of colour spaces. During my research I was introduced to 3D colour-histograms by Javier Villarroal’s website (2013), a site that allows the user to view the histograms of stock images. Five of these stock images are pictures of the same scene with alterations made to the images colour. The effects of changing an image’s brightness, contrast, and saturation on the histograms fascinated me and this curiosity is what brought me to the question that inspired my project: could a 3D colour-histogram be used as a visualisation or editing tool in image editing software in such a way that provides either a unique perspective or controls that are not currently available in most image editing software?

My project is a program used to attempt to answer me question. If I deem that the histogram is an appropriate visualation I will begin to explore the use of the histogram as a tool in image editing by developing my program into an application: I speculate that a histogram could be used to either alter the mood or temperature of an image by shifting colours towards parts of the space or to isolate ranges of colours in an image, allowing them to be edited independently.

If my colour histogram is successful I will expand it to being a tool of image editing itself. Rather than selecting an area of an image spatially and applying changes to the isolated area a histogram could be used to select the colours that are to be altered: this approach would be quicker than selecting spatially and allow greater control over effects such as mood and temperature. In addition, colour histograms are often bi-modal because the background and foreground contain different colour sets, it should therefore be possible to separate the foreground and background of an image using a colour histogram.

# Summary of Literature Review

The literature review summarised the research carried out in order to aid the designing and development of the project. Five key areas of research emerged: colour spaces, colour histograms, technologies, colour quantisation, and colour alterations. This section will summarise the literature review, introducing each topic and summarising conclusions drawn with the aim of providing the background knowledge necessary to read this report.

## Colour Spaces

Colour spaces are a method of representing and communicating colour in a coordinate system. I would have to choose which colour space or spaces in which to display the histogram and as each colour space has it’s own benefits and drawbacks, research was important to choose appropriately. It was also necessary to transform between colour spaces if I chose to use a colour space other than sRGB (the colour space that colours would most likely be inputted in) or as I later discovered, when applying colour adjustments: so research into how to transform between colour spaces was necessary.

From Andrew Steer’s introduction to colour science (2008) and Phil Cruse’s overview of the CIE-L\*a\*b\* space (2015) I concluded that sRGB and CIE-L\*a\*b\* would be two suitable colour spaces in which to display a histogram: sRGB because it’s a universal standard and one that most users will understand without needing it to be explained, and CIE-L\*a\*b\* because it preserves perceived uniformity of colour difference better most other colour spaces. Hoffmann’s two papers on the CIE-XYZ and CIE-L\*a\*b\* colour spaces (CIE Color Space, 2015) (CIELab Color Space, 2015) also helped draw this conclusion, but more importantly her papers provided useful information on colour space transformations. Hoffmann’s papers, alongside Ryan Juckett’s (2010) and Irotek’s (2014) websites, provided me with the understanding and formulae necessary to transform colours between the sRGB, CIE-XYZ and CIE-L\*a\*b\* colour spaces: CIE-XYZ is used as an intermediate colour space when transforming between sRGB and CIE-L\*a\*b\*.

## Colour Histograms

2D histograms plot light levels against the number of pixels in an image with that light level. Luminosity histograms plot absolute lightness values while colour histograms comprise of three plots on the same axes, one for each RGB colour channel: §Figure 2.1 shows an example of a luminosity and colour histogram on the same axes. 3D histograms on the other hand assign one axis to each dimension of the colour space and represent frequency by the size of the plot, usually represented as sphere or cube, an example of which can be seen in §Figure 2.2.

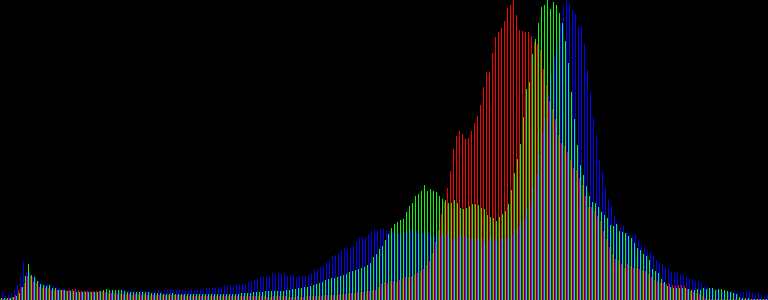


Figure : 2D Colour Histogram

(https://upload.wikimedia.org/wikipedia/commons/f/f5/Odd-eyed\_cat\_histogram.png)

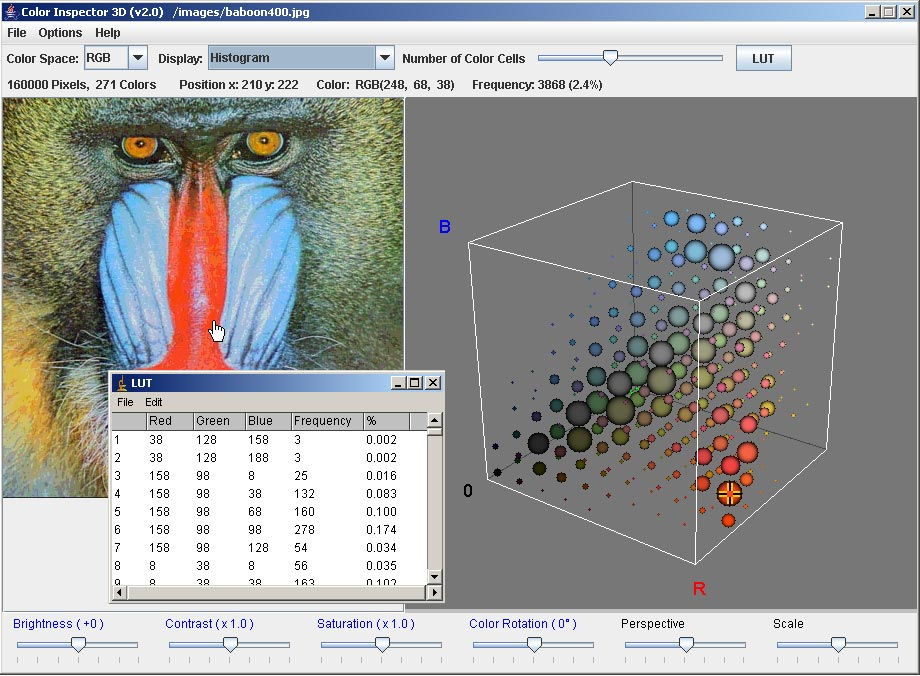


Figure : 3D Colour Histogram

(http://rsb.info.nih.gov/ij/plugins/images/3d-inspector.jpg)

I found that most sources were concerned with only 2D histograms and how they could be applied to photography, such as Practical Photography Tips (PracticalPhotographyTips), Rockwell’s webpage (2006) and the Cambridge in Colour webpage (McHugh, 2015). These sources helped me to understand colour histograms, but they were of limited use to my project. Sources relating to 3D colour histograms were all implementations, including Villarroal’s online histogram (2013) that inspired my project, Third Avenue Design’s web-based histogram (Third Ave Design), Caballero & Belmonte’s real-time video histogram (Caballero & Belmonte), and Barthel’s Java application (2007). By inspecting these sources I was able to infer how best to display a 3D histogram and which features from each to include in my project.

## Technologies

I also researched technologies that I could use to develop my project. I had previous experience using WebGL, a JavaScript library based on OpenGL, to do 3D renderings of colour spaces, augmented with another library, Three.js. I was heavily biased to developing my project as a web page using these frameworks but did some research to confirm that they were appropriate.

I had done a lot of the research into technologies and frameworks prior to my project when I was first working with colour spaces; from this experience I knew that the JavaScript libraries WebGL and Three.js were a good bet. Using the Khoronos Group’s and Parisi’s tutorials (Khoronos Group, 2015) (Parisi, 2010) I came to the conclusion that while WebGL is a well established rendering framework, it’s API was verbose and difficult to use. I concluded that Three.js (mrdoob, 2015), a framework that wraps up WebGL’s functionality so that it is easier to read and write, would hopefully result in better code and faster progress. I was also aware from previous experience that GLSL could be used to run small scripts on the client’s GPU much quicker than on the CPU.

## Colour Quantisation

If the colours extracted from an image were plotted directly onto a histogram they would be too sparsely dispersed for any useful inferences to be made. Colour quantisation is used to compress an image by reducing the number of colours while loosing as little visual information as possible. I realised that quantisation could be applied to the colours extracted from an image to allocate them into bins that could then be plotted onto a histogram.

Khouloud Meskaldji et. al.’s paper on the effects of colour quantisation on histograms (2009) was found to be irrelevant to my project despite it’s title. Segenchuk’s webpage (1997) goes over the details of a number of quantisation techniques with increasing complexity and from this I was able to conclude that only the simplest techniques were suitable for my project. This was because the more complex methods wouldn’t sample the image from a uniform grid but instead gave more resolution to higher density areas, which helps reduce loss of visual information when the image is being recreated with a smaller palette of colours, but for my histograms I would need to sample colours in a uniform grid. Eric Liao (2013) provides a pseudo-code implementation of one of these simpler quantisation algorithms on GitHub.

## Colour Adjustments

A key feature of my project is to allow the user to adjust colour controls and view the resulting changes to the image and it’s histogram. This is an area in which I had the least previous experience so I had to research both which colour adjustment controls would be most useful to include in my project and how they would be implemented.

I began by using Apple’s iPhoto (2015), which includes a 2D colour histogram, to experiment with which colour controls have the greatest impact on the histogram of an image. I made the assumption that large effects on a 2D histogram would translate to large effects on a 3D histogram. I identified exposure (brightness), contrast, and saturation, and lift/gamma/gain to be the two sets of controls that would be most useful to include in my project. Hurkman’s web page (2013) illustrates how lift/gamma/gain can be applied using formulae. From previous experience I also identified ASC-CDL, a summary of which was provided by Nikolai (2013), as potential formulae for colour alterations. At this point I needed to do further experimentation before I could make any more conclusions so no more research was undertaken.

# Summary of Specification and Design

The Specification and Design document was written before the second term in preparation for the development stage of my project. It outlines the definition and requirements of my project, the overall system architecture, specification and relation of tasks within the system, the user interface, a plan of testing and evaluation, and finally a time plan for the remaining weeks of the project. This section of the report summarises this original design for the sake of completeness and also so that the final product may be evaluated against it. A design of the final product with a similar structure to this section can be seen in §6, and this is evaluated against the original design in §7.

## Project Introduction

My introduction to colour spaces and 3D rendering was during my summer internship of 2015, where I worked on visualisations of colour spaces. While researching I discovered Javier Villarroal’s 3d colour histogram (H3Stogram - 3D Interactive Color Histogram, 2013), which inspired me to investigate the effects of colour alterations on an image’s histogram.

The experiences from my internship suggested that histograms could be useful as a visualisation or tool in image editing software. The aim of my project is to develop a program to investigate whether colour alterations have a significant effect on an image’s histograms and whether this could mean they are useful as a visualation or tool in image editing, such that they provide a unique perspective, as almost all conventional colour space visualisations take short cuts to render the spaces in 2D and therefore loose information. Such visualisations or tools could be used in image editing software with an intended audience of users who have a firm understanding of colour and would benefit from a unique perspective of colour spaces with no loss of information, unlike their 2D counterparts.

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

## System Architecture

The project’s architecture is to be designed in accordance with the Model View Controller (MVC) design pattern. This design pattern separates out the parts of the program responsible for handling and representing data: the model processes data and contains the business logic of the program; the view handles presentation of data; and the controller acts as a liaison between the model and view, allowing them to be independent. The basic UML diagram of an MVC application can be seen in Figure 3:1. I chose MVC because it’s separation of presentation and business logic is suited for projects with clearly separated user interfaces and data processing components, such as this project.

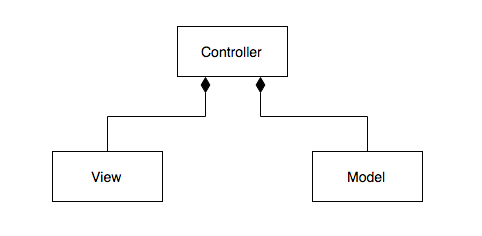


Figure : Model View Controller UML Diagram

## Component Design

The project structure is broken down into component tasks, each of which is defined by a particular job that it performs. The data flow diagram in Figure 3:2 depicts how data is passed between tasks and how tasks fit into the MVC design, defined in §3.2.

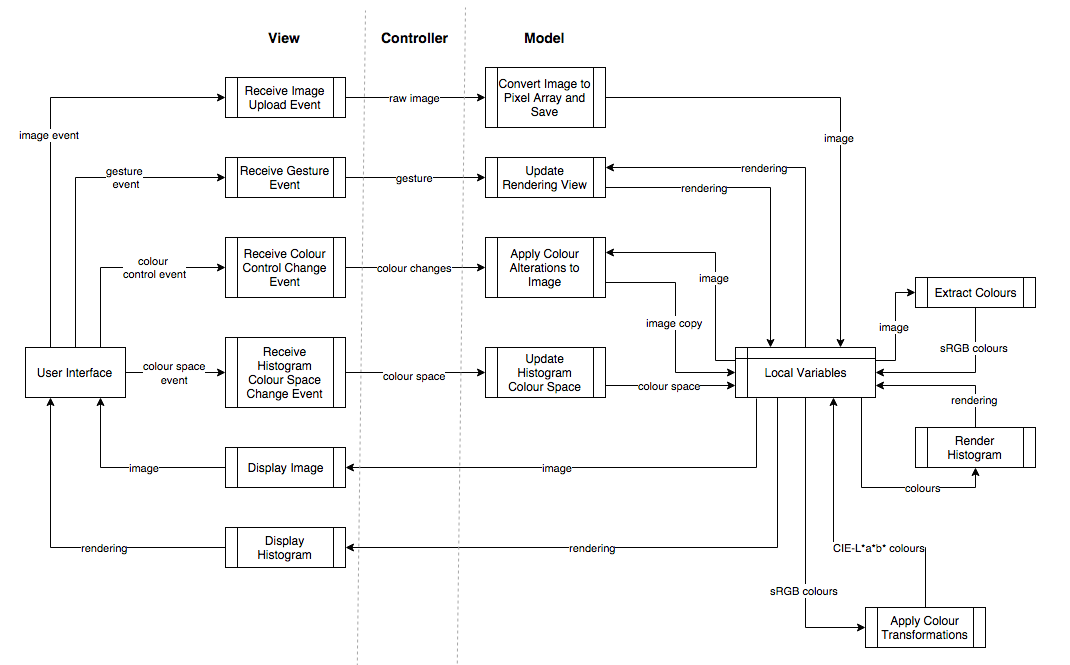


Figure : Data Flow Diagram

Tasks in the model include: extracting sRGB colours from an image; transforming colours between colour spaces; and applying colour alterations to the image. This design specifies that the model is responsible for producing the rendering of the histogram but as seen in §6, this was moved to the view during development. It is also worth noting that because no state needs to persist beyond the lifetime of the program, storing data in databases is unnecessary and using variables and fields will suffice.

Colours will initially be in the sRGB space but it will be necessary to convert these into the CIE-L\*a\*b\* space. As explained in §2.1, the CIE-XYZ space is used as an intermediate step. The sRGB colour must be linearised before a transformation matrix is applied to the colour vector: formulae and equations for these operations can be seen in Figures 3:3 and 3:4. Once in CIE-XYZ space another formula is applied to transform into CIE-L\*a\*b\* space, seen in Figure 3:5.

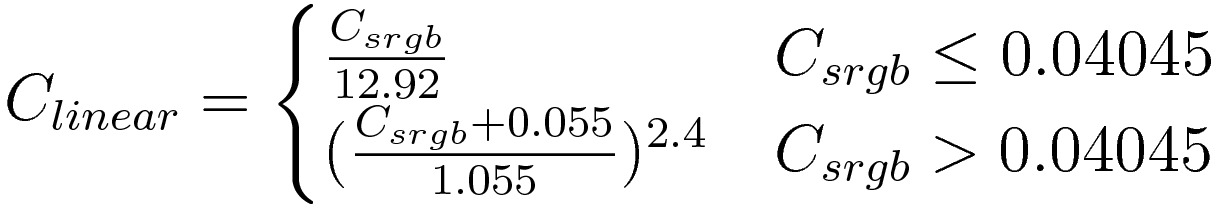


Figure .3 sRGB Linearisation Formula (Juckett, 2010)

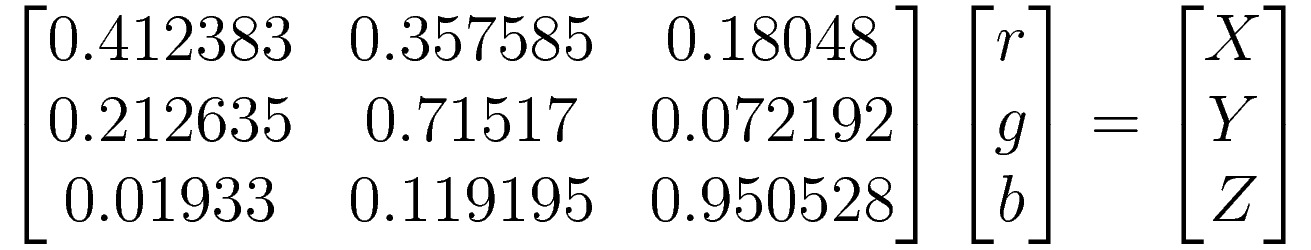


Figure .4 sRGB to CIE-XYZ Transformation Matrix (Juckett, 2010)

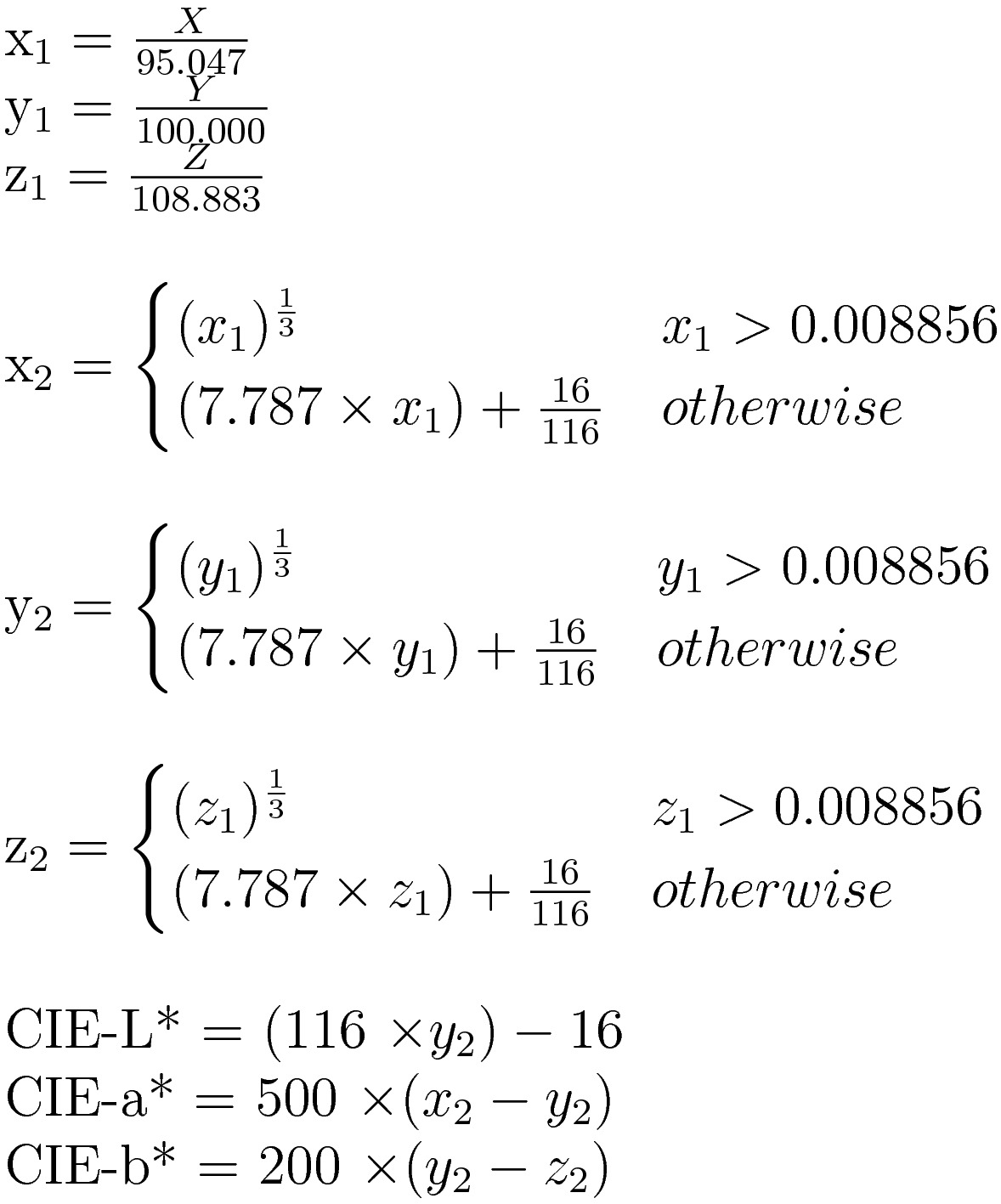


Figure .5 CIE-XYZ to CIE-L\*a\*b\* formula (Irotek Group Ltd., 2014)

The brightness, saturation, and contrast adjustments are applied to the colours in the model. Brightness is altered by adding or subtracting a constant value to each colour channel, but formulae of contrast and saturation are more complicated. 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 first transforming the RGB colour into HSL space and then adding or subtracting a saturation factor to the S value. The formulae to transform between sRGB and HSL can be seen in Figures 3:7 and 3:8.

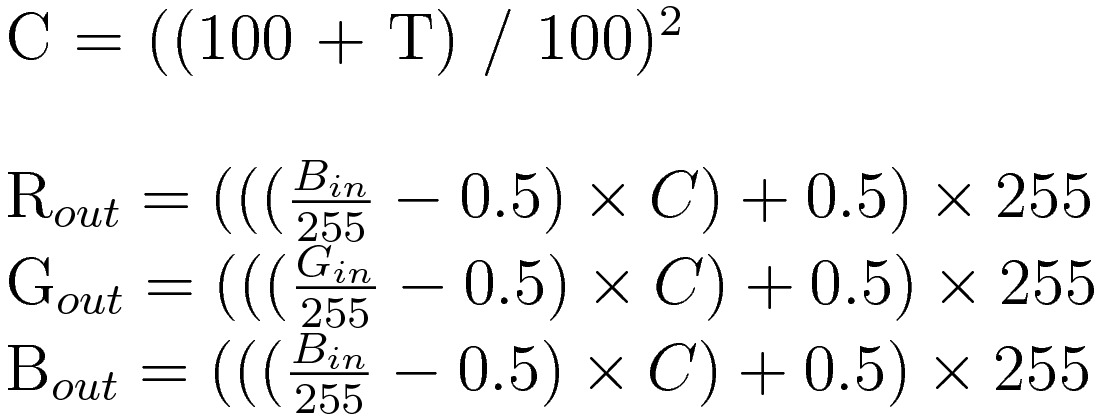


Figure .6 Contrast Formula (Esterhuizen, 2013)

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

A full listing of formulae used in the model can be found in the appendices.

The view’s tasks are to send events from the user interface to the model and to display the uploaded image and histogram. Events are fired by user interface in the event of the user uploading an image, performing a gesture on the rendering canvas, or adjusting the colour control sliders. In each case the event is caught by the view, relevant information extracted, and a second event fired, which is caught by the controller and passed onto the model so that it can update it’s state accordingly. The other task of the view is listen for events from the model, indicating it’s state has changed, and updating it’s representation of the model’s data.

The data flow diagram does not explicitly define tasks for the controller: this is because the controller’s role is to relay messages between the model and view. Implied tasks lie in between the view and model wherever data is passed between the two, and either receive events from the view and prompt the model to update it’s state or fetch data from the model and pass it to the view so that it can update it’s representation of the data.

## UI Design

The graphical user interface of the program will be composed of a central rendering of the histogram, surrounded by the image in the top right above an image upload button and three sliders for brightness, contrast, and saturation below, and other controls, chiefly what colour space the histogram is rendered in, to the left.

Uploading an image, adjusting the colour controls, or changing he colour space will fire an event that will be sent, by the view, to the model. Additionally, events are fired when the user drags or scrolls over the rendering, providing rotation, pan, and zoom controls on the rendering.

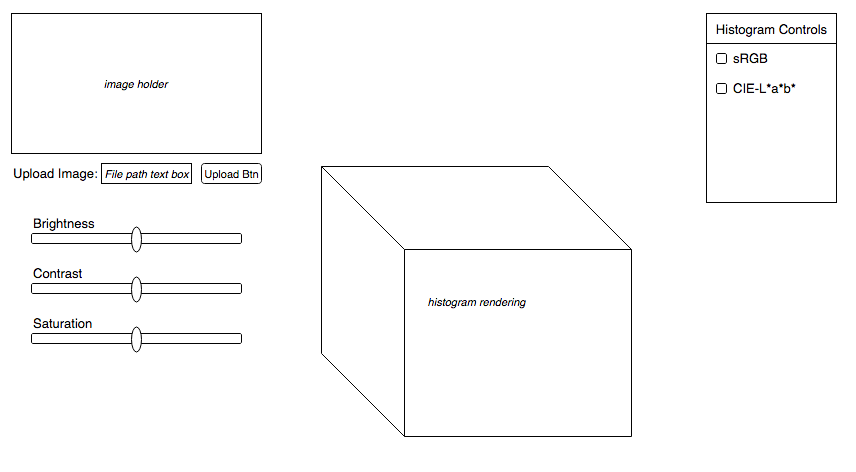


Figure : User Interface Sketch

## Testing and Evaluation

A suite of unit tests will be used to test the correctness of the program. Coupled with test driven development, the testing should ensure that code is divided into manageable pieces and result in better code. Tests should be written to cover a broad range of cases, focusing on marginal and extreme cases, which are the most likely to cause erroneous behaviour. Intentionally erroneous data should also be tested against to ensure that the program handles it correctly. Unfortunately it is not possible to formally test the rendering with unit tests: instead I will specify a number of well-defined manual tests, which will be conducted by a person. This is less than ideal but there is no other way to test the rendering.

In my design and specification I posed my idea of using 3D histograms as a visualation and tool in image editing as a hypothesis that I was attempting to prove or disprove. My hypothesis was to be tested by generating the histogram for a series of images, adjusting the colour controls, and observing the effects on the histogram. If a clear change was both seen in the histogram and could be predicted and replicated then I would consider my hypothesis to be proved, else it would be disproven. However this method of evaluation is subjective and getting enough results from unbiased people would be a problem. I have since come to the conclusion that I can not define what constitutes failure of my hypothesis well enough to pose it as a hypothesis. Instead I pose my original hypothesis as an idea that I am following through by developing my program. It will of course still be necessary to evaluate how effective the histogram was as a visualation and it’s potential as a tool.

## Time Plan

The time plan can be seen in Figure 3:3. Tasks are defined and highlighted weeks denote which are devoted to completing them. The first part of this time plan breaks the development stage down in 13 distinct steps and the final three tasks are concerned with writing this report and preparing for the poster session and demonstration. Formal deadlines are explicitly marked, but informal deadlines are implied at the end of each task’s allotted time.

Dividing development into short steps, each no longer than 3 weeks, allowed an iterative development cycle methodology to be adopted. Each cycle spans the length of one task in the time plan and consists of implementing and testing a feature. Using short development cycles allows the process to be more flexible as frequent adjustments to the time plan or design can be made as necessary.



Figure : Time Plan Gantt Chart

# Development and Testing

The Specification and Design document loosely defines a development and testing methodology to accompany the time plan (see §3.6) as well as a testing plan (see §3.5). This section of the report describes the development and testing process that actually took place, provides illustrations, and summarises noteworthy points in development where a particular difficulty was faced or development changed unexpectedly.

## Methodology

The methodology defined in §3.6 was used in the development of the project, alongside the testing outlined in §3.5. These ideas were loosely defined so were built upon and refined as the project progressed. Each development cycle, coinciding with a task on the time plan, came to be composed of three steps: implementation of the feature; testing of the feature with test-driven development; and reflection. There is also some retrospective testing conducted for those features that could not be tested with unit tests.

*Example of developing feature in time scheduled.*

A short and iterative development cycle was kept throughout development and while a number of changes were made to the design, the time plan was left unchanged and task deadlines mostly kept to. Initially the plan of test-driven development was adhered to during several cycles tests were left until the end.

An example of how tests were written is the first feature developed with the test-driven development paradigm. Before start of the second term a small amount of code had been written to support the literature review and design documents. This code hadn’t been written with the aforementioned development methodology, resulting in sub-standard code. The first week of development was devoted to bringing this code up to an acceptable standard and testing it; after this the development methodology could begin to be used. The method to perform colour quantisation was the first to be written in this manner and it was a success.

*Include code snippet here.*

A diary containing a detailed record of progress was kept through development. This allowed reflection on the progress made at the end of each cycle. A copy of the diary can be found in the appendices.

The Specification and Design document also specifies that the project will be developed with the Model View Controller (MVC) design pattern, the details of which can be seen in §3.2. During development I would adjust the original design of my project, seen in §3.3, so that it would adhere to the MVC design pattern.

The first task was to restructure the code into an MVC design pattern. The MVC design pattern works on the principal that the model and view are separated and access between them is carefully controlled; this behaviour is achieved through objects and while JavaScript provides objects using a prototype model it is traditionally procedural and I had not used JavaScript objects in this context. Alex Netkachov’s web page (Model-View-Controller (MVC) with JavaScript, 2015) provided me with a template of an MVC design pattern in JavaScript and I based the structure of my project off of it. I later came to understand that I was using objects incorrectly and all of my fields and methods were public, but I would go on to rectify this.

While I thought that my original data flow diagram (see §3.3) was true to the MVC design pattern I soon realised that it was not. In particular I noticed that rendering the histogram should be part of the view and not the model because it is concerned with representing the data. Consequently the model should pass the view a list of colours, which it creates a 3D colour histogram for. This resulted in creating my own object to represent a colour because I was previously relying on the rendering framework’s colour object in the model, which would have meant the model was dependant on the view. This colour object neatly encapsulates a colour’s values in multiple colour spaces and would go on to contain the methods used to transform between colour spaces.

## Points of Note

To begin testing I had to configure a testing framework for the IDE that I was using, Jet Brain’s WebStorm. After researching unit test frameworks for JavaScript I decided on using the Karma framework to run QUnit tests.

I noticed that a lot of the code was difficult to test because it involved a lot of interaction with the DOM and user, which could not be simulated in the tests. This was helped slightly by refactoring into an MVC framework so that most methods within the model could be tested independently of the view, but methods in the view remained untestable. However I wrote some tests for the methods that were easily testable independent of any interaction with the DOM or user.

The next stage of development was to allow the user to upload an image, extract it’s colours, and render them in the sRGB histogram. I found that the HTML5 canvas element accepts most commonly used image file types (and even video files) and returns a context object that allows me to query the colour value of pixels. Once I had an array of colours I needed to divided them up into uniformly spaced bins for plotting on the histogram. Eric Liao’s colour quantisation pseudo-code (Uniform Color Quantization, 2013) converts colours from 24-bit colour to 8-bit colour. I decided that 16-bit colour (i.e. 16 bins along each axis of the histogram) would be appropriate for my application so adjusted Liao’s algorithm accordingly. His algorithm used a look-up table (LUT) that holds a mapping from one space to another, but this was unnecessary as I discovered that the same results could be achieved with arithmetic operations simple enough that a LUT overcomplicated the process.

At this stage in development I was able to plot the colours from an image on the histogram: however I noticed that a small number of dominant colours were dwarfing the remaining, less common, colours. By comparing my results to Villarroal’s histogram (H3Stogram - 3D Interactive Color Histogram, 2013) these observations were confirmed, as my histogram was very different to his. By adjusting the size of the plots to be proportional to the natural logarithm of the value they represent all the plots were brought into comparable sizes. While a linear scale showed the true distribution of colours within an image it was difficult to infer any information about any colours other than the few most dominant, so a logarithmic scale allows for easier comparison between all colours.

I noticed that most images had a significant number of colours with very few instances and these plots, while included in the histogram, were barely visible. I decided to apply a minimum size to plots and any plots smaller than this would be increased to the minimum value. I chose to still include the small plots rather than omit them because I thought it more helpful to keep as much information as possible; but this did raise the question of whether to allow the user to have control over choices such as this. Other possible options that the user could have include what shape to plot bins as on the histogram and how transparent to make them. Giving the user more control would allow them to tune the program to their preferences and perhaps help them to extract information. I chose to not follow this path at this stage in development because it was more important to finish the key features of the program, but this is an area that I would like to explore given some more time.

Colour transformations were the next key step in my project. The colours extracted from the HTML5 canvas context were in sRGB space but I now wanted to display those same colours in CIE-L\*a\*b\* as this would give a new perspective, and one that represents differences in colour more uniformly. From my research I had concluded that to get from sRGB to CIE-L\*a\*b\* I would go through CIE-XYZ: transforming from CIE-XYZ to CIE-L\*a\*b\* was easily done with formulae but sRGB to CIE-XYZ involved a gamma correction function and matrix transformation. Irotek’s web site (Color conversion math and formulas, 2014) contained the CIE-XYZ to CIE-L\*a\*b\* formulae but also similar formulae for sRGB to CIE-XYZ. Previously I had been sceptical of these seemingly simplified formulae, thinking that they would not be equivalent, but I contacted the owners of the web site and they assured me that all the stages of the sRGB to CIE-XYZ transformation were represented perfectly within these simple formulae. During development I took their word for it and continued, safe in the knowledge that if it was wrong I could easily swap it out for an alternative, correct transformation. I have since gone back to assure myself that their formulae are equivalent to a gamma correction function followed by a matrix transformation and have satisfied for myself that they are identical (see Appendices). I then implemented these formulae as methods on the colour object, allowing me to transform a colour between the two colour spaces.

Once the colours had been transformed in CIE-L\*a\*b\* space the next challenge was rendering them. sRGB colours had been easy to render because WebGL used the RGB space and if the colours geometry vertices was defined, colours in between were linearly interpolated. In order to render CIE-L\*a\*b\*, colours had to be converted back to sRGB: however because the distributions of colours in CIE-L\*a\*b\* is different to sRGB they could not be linearly interpolated, and instead I would have to specify the colour of each pixel. This is achieved with the use of fragment shaders, which are attached to a geometry’s material and run on the client’s GPU.

I had previously implemented an MVC design based upon Alex Netkachov’s template (Model-View-Controller (MVC) with JavaScript, 2015) but at this point in development I discovered that I had made a mistake: all of my fields and methods in objects had been public. In the interest of keeping my project object-oriented I changed my code so that fields and methods that I wanted to be private, where. While this didn’t not initially change the running of the code it did improve development as I could not accidentally access a variable that I wanted to be private. I could then add stricter get and set methods to objects such as the colour to regulate how it was used and to ensure erroneous values are not inputted.

Now that the histogram could be viewed in both sRGB and CIE-L\*a\*b\* space it was discussed whether viewing both renderings side-by-side would be useful. I did not implemented this feature at the time as I deemed it not to be a priority but it was another consideration to return to alongside other controls that I might provide for the user.

The last feature left to add to my program was colour adjustment controls. As outlined in my literature review I had chosen to provide brightness, contrast, and saturation controls in my project, as I believed them to be the most intuitive controls that I thought would have a useful affect on the histogram. I had not determined exactly how to implement these colour controls in my research, so that was my first job. Brightness turned out to be uncomplicated, and consisted of adding a constant to each colour channel in RGB. The most concise method for adjusting saturation that I found was to transform to HSL (Hue Saturation Lightness) space, change the S value, and transform back to sRGB. The most complex of the three was contrast, a formula for which could be found on Loch’s web page (Image Processing Algorithms Part 5: Contrast Adjustment, 2008). I apply all colour adjustments to sRGB colours for two reasons: it is convenient as the formula would be different for each colour space; and this is the colour space that the colours are received in and displayed in the image.

A consideration that must be made when applying colour alterations is that they are not strictly reversible because if one channel is clipped and the others are not then just reversing the calculation will not result in the original colour. Therefore it is important to apply each colour alteration to the original, unaltered colours. This does however prompt some consideration into how a colour channel should be clipped. Currently I am performing a naïve clip, that is if one channel is outside it’s valid range it is set to the appropriate limit and the other channels are not affected. The results I get from this approach are satisfactory but it might be worth some consideration if I can be subtler. For example if a channel exceeds it’s limits, each colour channel could be scaled proportionally to it’s distance from the centre of the colour space so that the most extreme channel is within it’s bounds. I did not have time to experiment with this during development but it is a consideration that I would like to explore in the future to see if it improves results of my colour adjustments.

At this point all key features of my program had been added. I was on schedule and had done enough debugging along the way that I did not need to use my contingency time, meaning that I could focus the last of my time of optimising the speed of my web page. I had noticed that images larger than 200 pixels by 200 pixels would take an unreasonable amount of time to process: in the order of 30 seconds to a minute. I used Google Chrome’s JavaScript profiler to identify which of my function calls where taking up the most time and the results were surprising: I was expecting the colour transformations to be taking the longest, but reading colour values of pixels from the HTML5 canvas’ context was taking the majority of the program’s time, 24 to 25 seconds for a 305x235pixel image. I tried to sample fewer pixels in the image, say one in four, but this resulted in a blocky image being reproduced, there was a notable loss of colours in the histogram, and the computation time was only reduced to about 6.5 seconds. I quickly noticed that instead of querying the DOM for each pixel I could collect all the pixels at once. The same idea was also applied to setting pixel values when colour adjustments were applied. This reduced the computation time to roughly 0.66 seconds.

After this the next largest process was instantiating colour objects. Currently, each time a colour object is created it is converted into the CIE-L\*a\*b\* and HSL colour spaces to save time later one: however it may be preferable to use lazy instantiation and only perform these transformations when necessary so to speed up the web page initially. Another consideration that I could make would be to only have one colour object for each unique colour, referenced by each instance of that colour in an image, as I am currently performing the same transformations many times for duplicate colours. An extreme solution would be to represent an image’s colours an a one dimensional array of colours and apply all transformations to that array. This approach would reduce the number of objects but duplicated colours would be transformed multiple times.

Dividing in the program into a model, view, and controller ensured that the model was testable because it was abstract from the view, which is hard to test due to its entanglement with the DOM and user interaction; the controller mostly just handles events, and as events are tested elsewhere, testing the controller is unnecessary. While most of the method in the model are easily tested, I was unable to test those that handle image files as I could not create dummy images: this was because image files were uploaded through the HTML5 File element, which could not be replicated in testing as it relied upon user interaction, and a file can not be uploaded from the local file system in JavaScript for security reasons. While this was an inconvenience, by structuring the code how I had, it had a minimal impact on my ability to test code.

One other element of my program that turned out to be impossible to test was the GLSL code: this is code defined within HTML script tags, loaded by WebGL and run on the client’s GPU. The GLSL code is not recognised by the testing framework and there are no development tools, so I can not test it. Currently the GLSL code determines how colours are interpreted and displayed in the rendering; this can be tested manually by inspecting the rendering and comparing it to what I know to be correct. While I am quite trusting of it’s ability to handle sRGB colours, displaying CIE-L\*a\*b\* colours requires a colour transformation and has more chance of going wrong. The formulae used in this transformation have been taken from my JavaScript code, which was tested, so the only errors in the code are typos that I would have made when moving them across. Unfortunately I can not think of a better way to test this as GLSL is required to correctly render CIE-L\*a\*b\* colours without severe performance issues. Moving more calculations to the client’s GPU would improve the speed of my program but it would make more complex code untestable, so it is currently a balance between testable code and quick code.

# Testing

# Description of the Final Product

## System Architecture

The product’s architecture implements the Model View Controller (MVC) design pattern: the model processes and stores data; the view handles the representation of data; and the controller acts as a liaison between the model and view. This structure separates the parts of the program responsible for processing data and representing data. The model is entirely independent of the view and has no knowledge of how the data is presented, and the view does not process data. The controller holds and instance of the model and the view, allowing it to accommodate communication between the two. When the user interacts with the view it fires events, which the controller listens for and passes relevant data onto the model, which updates it’s state accordingly. When the model’s state changes it also fires events to alert the view that it’s representation of the data needs to be updated. In my system, the view holds and instance of the model so that it can listen directly to event as well as access the model’s data.

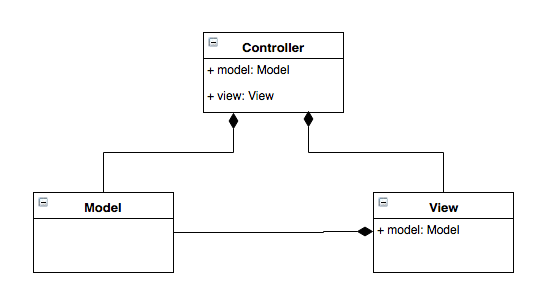


Figure : MVC UML Diagram

## Component Design

The program is divided into component tasks, each of which is defined by the job that it performs. The Data Flow Diagram (§Figure 6:2) depicts data being passed between tasks and other entities, such as the user interface or data storage. Data needn’t exist beyond the lifetime of the application and therefore databases or other persistent data stores are unnecessary; instead I have used private fields on the model to store data.

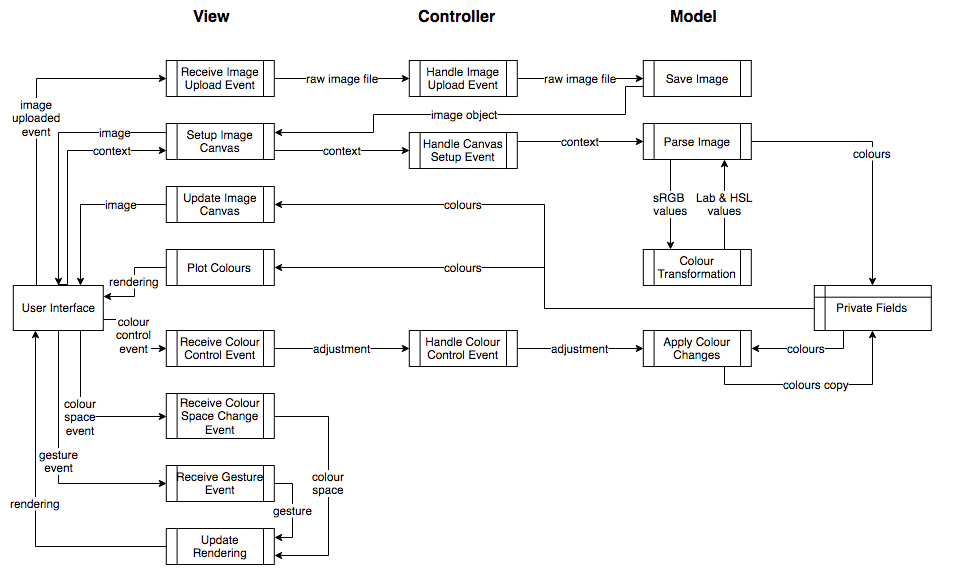


Figure : Data Flow Diagram

### View

Tasks in the view fall into two categories: handling events from the user interface and updating the representation of the data.

Events are generated by the user interface when a user does one of the following: uploads an image, makes a gesture on the rendering, changes the colour space of the histogram, or makes an change to the colour control sliders. Each event has it’s own tasks in the view which receives an event from the DOM, extracts relevant data, and either passes it into the controller or handles it internally within the view. The Receive Image Upload Event and Receive Colour Control Event tasks extract the raw image file and colour adjustments from the neutral position on the slider respectively and pass it onto the controller. The Receive Colour Space Change Event and Receive Gesture Event tasks handle the event within the view because they do not affect the state of the model. Instead the new colour space or the gesture is used to update the histogram.

The view contains an instance of the model and listens to events that it fires. When an event is fired this indicates that the model’s state has changed and the view fetches the relevant data from the model are redraws the histogram. The main example of this process is the Update Image Canvas and Plot Colours tasks. These tasks are notified when the model’s state changes and fetch the list of updated list of colours. The first task, Update Image Canvas, takes the colours and paints them to the image canvas, thus allowing an edited version of the image to be viewed by the user. The second tasks, Plot Colours, takes the colours and plots them on the histogram in the relevant colour space.

The task that behaves slightly differently is Setup Image Canvas. This task is different to Update Image Canvas as it is called only when an image is first uploaded to the program and rather than fetching data from the model itself, it receives data in the event that triggers it. The task receives an Image object from the model, which was obtained from the raw image file sent by the Receive Image Upload Event task; this object is placed on a canvas in the DOM, allowing the image is be visible to the user. The context for this canvas is then sent back to the model: it is this canvas that can be queried for the colour values at each pixel and is used to extract colours from the image.

Using the HTML5 canvas to hold the image allows most common image file types to be interpreted as an array of pixels: however the draw-back to this approach is that the view processes some data, in that it transforms the image into a form that is useful to the model. This does not strictly follow the MVC design, but performing this conversion myself in the model would have been too much work to achieve and my results would not have been as good as those obtained using the HTML canvas.

### Model

The model’s responsibility is to process and store data and it is home to the program’s business logic. The controller contains a reference to the model and when it sees an event fired from the view it calls the appropriate method in the model to respond accordingly, so all tasks in the model are indirectly prompted by events in the view.

The first task run in the model is Save Image, which takes a raw image file and puts it into an Image object that can be interpreted by the view and displayed on the web page. Once this is done the model fires an event containing this image, for which the view listens for. When the image has been placed in a canvas a context of that canvas is sent back to the model: this context can be inspected to get the colour value at each pixel and the Parse Image task uses it to extract a list of colours in the image. For each colour in the image, an object is created with represents that colour. On instantiation this object converts the sRGB colour into the CIE-L\*a\*b\* and HSL space for use later on. The list of colour objects is then stored as a private field of the model; this field can be accessed through a getter method that is called by the view on it’s instance of the model to access the latest list of colours.

The colour object represents a single colour. The object contains a number of fields which holds the colour’s values in the sRGB, CIE-L\*a\*b\*, and HSL spaces. These fields are populated on instantiation by methods on the object: these methods contain the formulae for converting between spaces. Each formulae is a programmatic representation of operations: for example the conversion from sRGB to CIE-XYZ (which is used as an intermediate space when transforming to CIE-L\*a\*b\*) consists of linearising the sRGB values before applying a transformation matrix to the colour vector. The details of these transformation formulae and the equivalent mathematical operations can be seen in the appendix.

The other operation that the model carries out is applying colour alterations to the colours extracted from an image, carried out in the Apply Colour Changes task. The colour alteration to apply is sent to the model from the controller. It is important that the altered colours do not overwrite the original colours because colour adjustments are not reversible if colour channels are clipped. This means that the value used to determine the degree of the alteration is given in relation to the neutral position on the sliders and colour adjustments are applied to a copy of the list of colours, which is saved as to not overwrite the original. The altered list is accessed by the view. Once the colour alterations have been applied an event is fired by the model, prompting the view to update it’s histogram and image with the altered colours.

Each colour control, brightness, saturation, and contrast, has it’s own method of application. Brightness is the simplest to apply as it is done by adding or subtracting a constant to the red, green, and blue colour channels. The formula for contrast calculates a contrast correction factor, which is then applied to each colour channel (red, green, and blue) independently. The formula was sourced from Francis Loch’s web page (Image Processing Algorithms Part 5: Contrast Adjustment, 2008) and can be found in the appendix. Formulae for applying saturation are not as simple as those used for contrast and brightness, but an easy method that I used was to convert the sRGB colour into HSL space, adjust the S value, and convert back to sRGB. As mentioned previously this transformation is included in the colour object and it done using formulae found in the appendix. For any of these colour alterations, if the value of a channel falls outside the valid range of values it is clipped to the limit that it passed. This naïve approach to clipping works well enough for the first iteration of development, but it is an area that I want to look into to see if there is a more intelligent and accurate method of clipping that doesn’t cause unexpected colours to emerge.

### Controller

The controller contains an instance of the model and the view. It listen for events fired by the view and then calls the appropriate methods on the model to update it’s state accordingly. This can be seen in the data flow diagram by the tasks that intercept data travelling from the view to the model. The controller does not intercept data between the model and the view because the view listens directly to events fired by the model. The controller is the part of the MVC design that enables the view and model to be independent of each other and it provides a layer of isolation between the two.

## User Interface

The user interface of my program is a web page. The focus of the web page is the rendering of the histogram, which is placed in the centre of the window, and to the top left of the rendering is a copy of the image uploaded by the user. Running along the top of the page is a section that contains controls: a button allowing the user to upload an image from their local file system; a radio button giving the user the choice of viewing the histogram in the sRGB or CIE-L\*a\*b\* spaces; and a series of three sliders, representing the brightness, contrast, and saturation controls.

Interacting the user interface fires events that are handled by the view. The button below the image canvas prompts a window to appear which allows the user to navigate their local file system and select an image file; an event is fired when they confirm their selection containing a copy of the image file. As explained previously, the image will then appear in the canvas above the button. Changing the colour space radio button selection will also fire an event, this one causing the rendering to change to the selected colour space. Changing the position of one of the three sliders triggers an event containing the new position of the slider. Shortly after the image and the histogram will be updated with the newly altered colours. The final user interaction with the UI to cause an event is gestures on the rendering: dragging with the left or right mouse button held and scrolling with allow the user to rotate, pan, or zoom the rendering. Each gesture triggers and event containing the buttons pressed and location of the cursor on the screen, which is interpreted by the view to move the rendering accordingly.

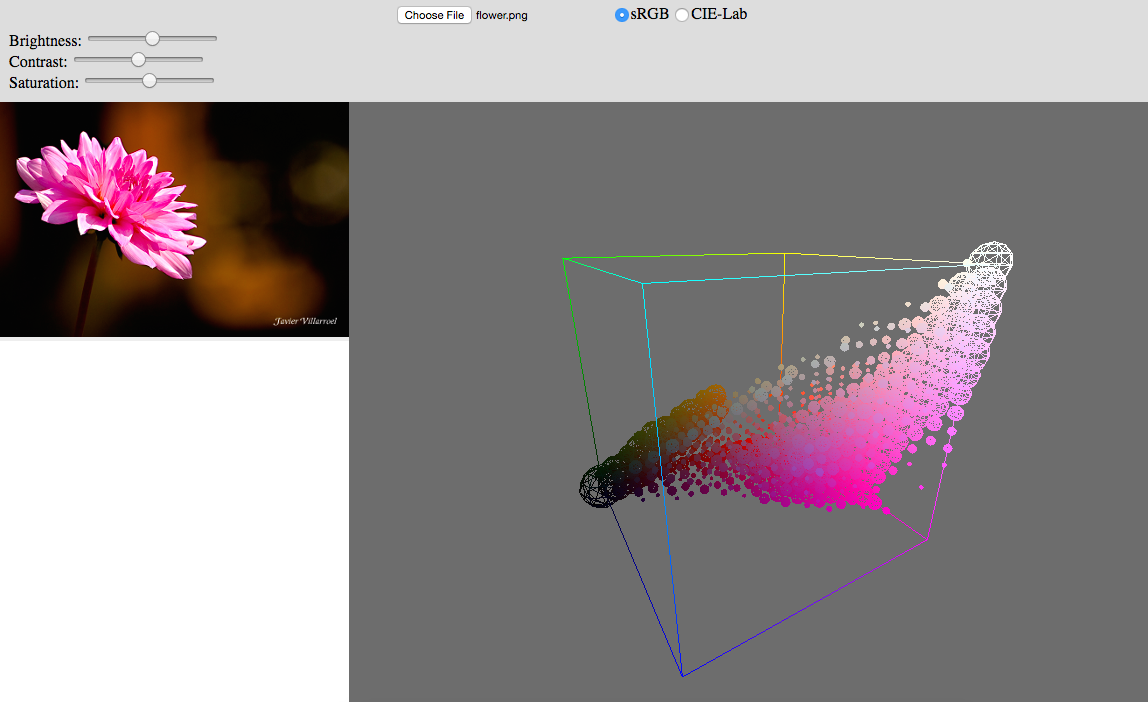


Figure : User Interface

## Product Behaviour and Performance

The original design specifies the product’s basic requirements are to allow the user to: upload an image from their local storage; view the image’s 3D colour histogram in the sRGB or CIE-L\*a\*b\* space as in interactive rendering; change an image’s brightness, saturation, and contrast levels; and view the effects of these colour adjustments on both the image and histogram. The final product provides all of these features.

References will be made to the product’s user interface, which can be seen in Figure 6:3. The user uploads an image by clicking the button labelled “Choose File” at the top of the window; this prompts a second window to open, allowing the user to navigate their file system and select an image file. Once confirmed, the uploaded image will be processed and will appear in the canvas on the left while the histogram of the image appears in the canvas on the right; this process will take a few seconds.

Initially presented in the sRGB space, the histogram can also be viewed in the CIE-L\*a\*b\* space through the use of the radio button at the top of the window. This will redraw the histogram in the new colour space. This should not take more than a second because the colours are transformed into the CIE-L\*a\*b\* space when the image is uploaded.

The rendering of the histogram can be manipulated by the user through gestures made on the rendering canvas. A left-click and drag will rotate the rendering around it’s centre; a right-click and drag pans the rendering’s centre; and scrolling will zoom in and out on the rendering centre.

Finally, the brightness, contrast, and saturation levels of the uploaded image can be adjusted through the use of the three sliders in the top left of the window. Initially in the centre position, moving them will cause the image and it’s histogram to be redrawn with the appropriate colour alterations applied. This process will also take a few seconds while the colours are manipulated.

While it provides all of the features defined in the design, it does not perform these operations in real-time as hoped.

# Evaluation of the Final Product

## Introduction

My project can be evaluated by comparing it to the specification and design document that I wrote before starting development: if my final product meets the requirements and matches the design in this document it will be considered successful. There were points in development where the design had to be modified in light of new information but as long as these changes can be justified and still meet the initial requirements they will be deemed acceptable. It can also be evaluated through user testing and experimentation to see if it allows me to answer the questions that I set out to answer: can 3D histograms be used as a visualisation tool in image editing in such a way that they provide a different perspective to common tools provided in image editing software.

## Requirements

The program that I have developed meets all of the basic requirements defined in design. It allows the user to: upload an image; view it’s histogram in the sRGB and CIE-L\*a\*b\* spaces, adjust an image’s brightness, contrast, and saturation levels; and view the effects of these controls on the image and it’s histogram. I therefore conclude that my project has successfully met the requirements set in the original design. One requirement that I failed to meet was that the program should run in as close to real-time as possible. As it stands, displaying an image’s histogram and apply colour alterations takes a few seconds of computation. I have previously discussed in §4 Development how I have successfully improved the speed of my program, but it is still a way away from real-time, but there are further improvements that could be made, which are discussed later. I also mentioned that if I had time left at the end of development I would begin work on transforming the histogram from a visualisation to a tool: I did not start development on this but it is discussed later in this document.

## System Architecture

The design specified that the system architecture should be structured using the Model View Controller design pattern. As explained in §6 Description of the Final Product, I used the MVC design in my product. As I developed by program I learnt more about how MVC should be applied in the context of a web site and so the UML diagram that I provided in my original design changed. The controller still contained an instance of the view and model, but the design changed slightly, giving the view an instance of the model so that it could access data to be represented on the web page. This difference between my design and the final product’s structure was due to an error in the design, and I during development I have been reassured that my current implementation of the MVC design is correct.

## Component Design

The final product contains all of the features outlined in the original design’s data flow diagram, but some of the features have been moved and the flow of data changed. The most notable change is that tasks responsible for creating the rendering of the histogram have been moved from the model to the view: I made this decision early during development after coming to the conclusion that, while central to the core concept of my program, the histogram was entirely a representation of the data in the model and did not carry out any data processing itself. Two additional effects of the rendering being moved to the view is that colours must be fetched from the model by the view and that the current colour space of the histogram is stored as state in the view, not the model. As these changes does not affect how the program behaves and it is more in keeping with the MVC design than my original design, I call this change an improvement.

Another alteration to the structure of the program occurred when I discovered that the HTML5 canvas object could be used to convert most common image file formats into an array of pixels. My original data flow diagram had a task in the model that took the raw image file from the view and converted it into an array of pixels that could be queried and saved it to the model, before extracting the colours form that image. With the introduction of the HTML5 canvas element the first tasks was split into two and incorporated the second. The model would first receive the raw image file, convert it into an Image file, which did not allow pixels to be queried but was passed back to the view where it was placed in the canvas, and the context sent back to the model. The context is then used to extract the colours from the image. The structured defined in the design is correct and this change only came about when a new and easier method of converting an image file was discovered. The structure was updated accordingly and therefore is an acceptable change.

As well as the newly added tasks to handle the rendering, the view contains four tasks which receive events from the DOM. Events fired when an image is uploaded or a colour control changed follow the design’s specification and are passed to the controller, but the two tasks that receive colour space change and gesture events no longer pass the event on to the controller because now that rendering is done in the view, they do not affect the state of the model, and are handled within the view.

Another change made within the model is when colour transformations are applied. The task that transforms a colour from one colour space to another has been moved to work with the task that extracts colours. This was done because during development I decided to perform all colour transformations when a colour object is created. I made this decision because I thought it would be better to sacrifice some time at the start of the program so that it would run quicker with minimised loadings later on.

## User Interface

The user interface of the final product contains all of the features outlined in the design and it is a web page. My intent was to have the rendering taking up the entire web page with the image and controls overlaid on top inside containing boxes. The final product’s user interface is more basic: the rendering and image are in two adjacent divisions, with a third above them containing all necessary controls. Additionally the divisions do not resize to fill the web page, but the image’s division does resize to fit the image. The reason for my user interface not meeting the requirements set in the design was that during time constraints I prioritised the program rather than it’s user interface. I do not regret this decision as I was able to produce a program that contained all the necessary features and met requirements, but improving the interface would be one of the first things I would do, given more time.

## Product Behaviour and Performance

As shown in §6.4, the final product contains all the features defined in the original design and it has therefore met it’s requirements. While the product does provide all the necessary features to the user there are some nuances that have to be considered.

The optimistic goal of plotting an image’s colours on a histogram and performing colour alterations in real time was not reached. I prioritised creating a working product before optimising it and I deem this to be a wise decision. I made efforts to improve the performance of the product at the end of development and in the case of extracting colours from the image, was able to reduce the process down from 25 seconds to 0.6 seconds with a minor change. I have had more ideas as to how the program can be further optimised; unfortunately I did not have time to explore these options during development but they are discussed §9 along with other future work for the project such as the development of the project as a tool.

The other aspect of my program’s performance that must be considered is whether it was successful in allowing me to answer my initial questions, set out in §3.1: whether colour alterations have a significant effect on an image’s 3D colour histogram and whether this gives a unique point of view, unobtainable with conventional visualisations and whether this could be applied as a visualisation or tool in image editing software.

## User Testing

# Critical Assessment of the Project as a Whole

## Pre-Development

During the first term, my focus was on the literature review and the specification and design document. I had a good work ethic, which was helped by the time plan in the project statement and plan. I was satisfied with both these pieces of work and they would help me later, during development. During this time I also started writing code, the aim of which was twofold: it would support my literature review and design, allowing me to come to more informed conclusions; and secondly it would give me a head start in the development stage of my project. Because code that I wrote was largely based on that which I had already written over the previous summer and there was little original or new material, 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 fragile, containing a number of bugs. The poor structure also meant that the code was difficult to build upon. Intervention had to be taken at the beginning of development to get the code to an acceptable standard; luckily, due to a realistic and well thought out development plan I had ample time to fix the code.

## Development Cycle

The start of the second term also marked the start of my development stage. My specification and design document included a time plan that spanned the final stages of my project, from development to the final poster session and demonstration. In particular it focused on the development of the project, which was broken into well-contained tasks that would take no longer than three weeks to complete. This allowed me to adopt a short and iterating development cycle. In each cycle I would complete the task, testing along the way, followed by a brief review to ensure the project still met the design and that I was on schedule. The short development cycles made progress easy to manage, preventing it from getting stuck down a tangent, and allowed me to be confident that the code was correct at each milestone. In hindsight I did not put enough emphasis on the review at the end of each cycle. Should I do it again, I would evaluate the code written in the past cycle and how it fits into the project as a whole. An example of where this would have helped is in my data flow diagram. As discussed previously, the structure of my program was modified at several points; this itself is not a problem as long as the project still conforms to the design, but at each change I did not give enough thought to the impact on the rest of the project. Consequently, at the end of development there were aspects of the project’s structure that were not in accordance with the MVC design.

While there was no formal evaluation at the end of each development cycle, I was able to reflect briefly in my diary. I kept a diary of my progress, which while initially intended for retrospective analysis during the writing of this report, allowed me to pause and assess what I had achieved at the end of each cycle. It was not completely intentional but this diary benefitted my project by allowing me time to reflect and gather my thoughts, and it is a piece of initiative that I am proud of.

## Time Plan

I set myself a goal to complete development by Easter so that the remaining time could be allotted to this report, the poster, and preparation for the demonstration. I knew from experience that estimating the time needed to complete software development tasks was difficult so I was generous with my timings and prepared to alter the schedule if required, as long as development was still forecast to finish by the start of Easter. However it was not necessary to change the development schedule because I met each deadline, often with time to spare. While this success was party due to a good work ethic, in hindsight the time allocated for certain tasks, such as colour adjustment algorithms and colour space transformations, was overly generous would have been better spent improving the efficiency of my program or exploring future development as a tool at the end of development.

## Testing

In the design, I stated that unit tests would be used to test my code and that I would practise the test-driven development programming paradigm. I think that this was a good decision because it meant there was not a large amount of testing to do at the end of development and I was confident that the program was correct throughout development, and while I kept with it initially I often got carried away and forget to test until I was finished implementing a feature. I was still able to test as I went along and but if I had kept to test-driven development more strictly I would have produced better code.

I also wish that I had done more research into testing frameworks before deciding upon QUnit. QUnit was useful but it was lacking functionality, leaving holes in my tests: for example there was not feature to give a threshold to acceptable values. Some parts of my program were inevitably going to be hard to test, such as the rendering, but testing may have been possible for some would if I had used a different testing framework. In summary, QUnit turned out to be a limited testing framework that only offered me the basics in testing ability and using it for my application was a poor choice.

## Conclusion

In conclusion, I am mostly satisfied by the manner with which I completed my project: my time plan was realistic and I kept to it; I kept to a short and iterating development cycle, preventing code from running away from me; and I kept a detailed diary of my progress. My largest downfall was the lack of discipline in testing. Not enough research was done into appropriate testing frameworks and while beneficial when it was used, test-driven development was not strictly kept to and some parts of the program remain untested because of this.

# Future Work

While I have successfully implemented all the features I set out to in the design there are still improvements that can be made. The project also has potential beyond it’s current form because, as stated at the start of this document, I see the potential for 3D colour histograms to be used as a tool of image editing as well as a visualisation.

In it’s current state, I see two key areas of the project that could be improved: the user interface, and the program’s performance. Due to it being a necessity of my project and not the focus, the user interface is poorly designed and implemented. If my project was to become something that I could make available online, the user interface would need a nearly complete overhaul to get it to the necessary standards. If the program was to be released it’s performance would also need to be improved, otherwise the computation times render it almost unusable. I have already begun to optimise the program but there is a lot more that could be done, and must be done if I wanted to reach my goal of making it real-time.

Looking beyond my current project, I see the possibility for 3D colour histograms to be used as tools, not just visualisations. My project was a prototype that aimed to answer the question of whether histograms offered a unique perspective on colours, unavailable with conventional and current tools. The results of my project are reassuring and suggest that histograms are useful as visualisations, but could this be carried on to develop them into tools as well?

* How these tools would work?

# Conclusion

# 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

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

Javi Fontan, D. G. (2015, April 10). *GLSL Sandbox*. Retrieved October 2, 2015 from GLSL Sandbox: http://glslsandbox.com/

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

Loch, F. G. (2008). *Image Processing Algorithms Part 5: Contrast Adjustment*. Retrieved March 14, 2016 from Dreamland Fantasy Studios: http://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

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

Rockwell, K. (2006). *How To Use Color Histograms*. Retrieved October 3, 2015 from Ken Rockwell: http://www.kenrockwell.com/tech/yrgb.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/

Tarlier, F. (2009, November 5). *lift/gamma/gain VS. ASC CDL*. Retrieved October 2, 2015 from Francois Tarlier's blog: http://www.francois-tarlier.com/blog/liftgammagain-vs-asc-cdl/

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

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

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

Wikipedia. (2015, January 8). *ASC CDL.* Retrieved October 2, 2015 from Wikipedia: https://en.wikipedia.org/wiki/ASC\_CDL

Wikipedia. (2015, September 26). *CIE 1931 color space*. Retrieved October 2, 2015 from Wikipedia: https://en.wikipedia.org/wiki/CIE\_1931\_color\_space#Color\_matching\_functions

Wikipedia. (2015, September 29). *Color histogram*. Retrieved October 2, 2015 from Wikipedia: https://en.wikipedia.org/wiki/Color\_histogram

# Appendices