# 3D Colour Histogram

*Specification & Design Document*

# 1. Problem Introduction

## 1.1. Motivation

During my internship I worked on visualisations of colour space transformations. While researching I discovered 3D colour histograms, which show the distribution of colour in an image. A website I found that was of particular interest was Javier Villarroal's (2013), which produces the histograms of stock images, five of which are of the same scene but with colour alterations applied. I found the difference between these histograms fascinating and it opened my eyes to the potential of using histograms as a tool in image editing.

## 1.2. Context

**1.2.1. Colour Spaces and Models** A colour model is an abstract model used to describe colour in a coordinate system, independent of physical representation. The models used in my project are RGB and CIE-L*a*b*. RGB defines colours by their red, green, and blue components, while CIE-L*a*b* defines colours by a lightness value (L), a red/green component (a), and a blue/yellow component (b).

A colour space is a complete set of colours within a model that can be applied to a physical medium. sRGB and CIE-XYZ are spaces within the RGB model that are used in my project: sRGB, because it is widely used and easy to comprehend; and CIE-XYZ, because it can be used as an intermediate step in colour space transformations. While it is a model, CIE-L*a*b* can also be used as a space and will be in my project because it is larger and more perceptually uniform than sRGB.

**1.2.2. 2D Histograms** The 2D colour histogram of an image graphs light intensities, along the x-axis, against the number of pixels with a particular luminosity on the y-axis. Luminosity histograms plot the absolute brightness of for each pixel and colour histograms plot three graphs, one each for the red, green, and blue components. 2D histograms are used to check exposure in an image on digital cameras or in image editing software.

3D colour histograms use three axes to create a 3D colour space in which colours are plotted; the size of a plot is proportional to the number of pixels with that colour. Like 2D histograms, 3D histograms can also be used to check exposure but they are primarily used to view the distribution of colour in an image because unlike in their 2D counterpart, individual colours can be identified in a 3D histogram.

**1.2.3. Transformations** The histogram in my project may display colours in either the sRGB or CIE-L*a*b* space and it will therefore be necessary to transform between these spaces. Initially sRGB must be transformed to CIE-XYZ, which can then be transformed to CIE-L*a*b*. sRGB's luminosity is gamma corrected and exponential but CIE-XYZ has a linear scale luminance; a correction function is applied to sRGB to linearise it. To convert the sRGB coordinate vector to CIE-XYZ it is multiplied by a transformation matrix, which is derived from the coordinates of the sRGB primaries and white point in CIE-XYZ. Finally, colours in CIE-XYZ can be transformed into CIE-L*a*b* using a simple formula. Relevant formulae can be found in §3.1.4.

## 1.3. Definition and Requirements

The experience from my internship suggests that histograms could be used as a visualation or tool in image editing, leading me to develop the hypotheses: "Colour alterations have a

significant effect on an image's 3D colour histogram" and "A histogram can be used as a visualation or tool for use in image editing". My project will be a program that I will use to resolve my hypotheses by testing whether colour alterations have a significant and meaningful effect on an image's histogram. 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.

The basic requirements of my program are that it should allow the user to: upload an image; view its histogram in the sRGB or CIE-L*a*b* space; adjust the image's brightness, contrast, and saturation levels; and view the effects of these changes on the histogram. My program's requirements are covered more expensively in §3. At the end of my project, my program should be able to assist me in resolving my hypotheses.

## 2. System Architecture

My project architecture is designed with the Model View Controller design pattern: the model processes and stores data, as well as contains the business logic; the view handles presentation; and the controller contains an instance of the model and the view, handling all communication between the two by sending commands to update the model's state and update the view's presentation of the model. The UML diagram in Figure 2.1 shows the relationships between the MVC components. I chose the Model View Controller pattern because I am confident in using it from previous projects and it is suited for applications with a user interface.

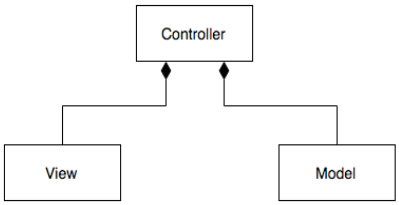


**Figure 2.1 Model View Controller UML Diagram**

## 3. Component Design

I have broken down my program into component tasks, each of which is defined by the job that it must perform. Figure 3.1 is a Data Flow Diagram that illustrates how data is passed between component tasks and other entities, such as the UI and data storage. Components are also grouped to reflect the MVC design pattern, described in §2.

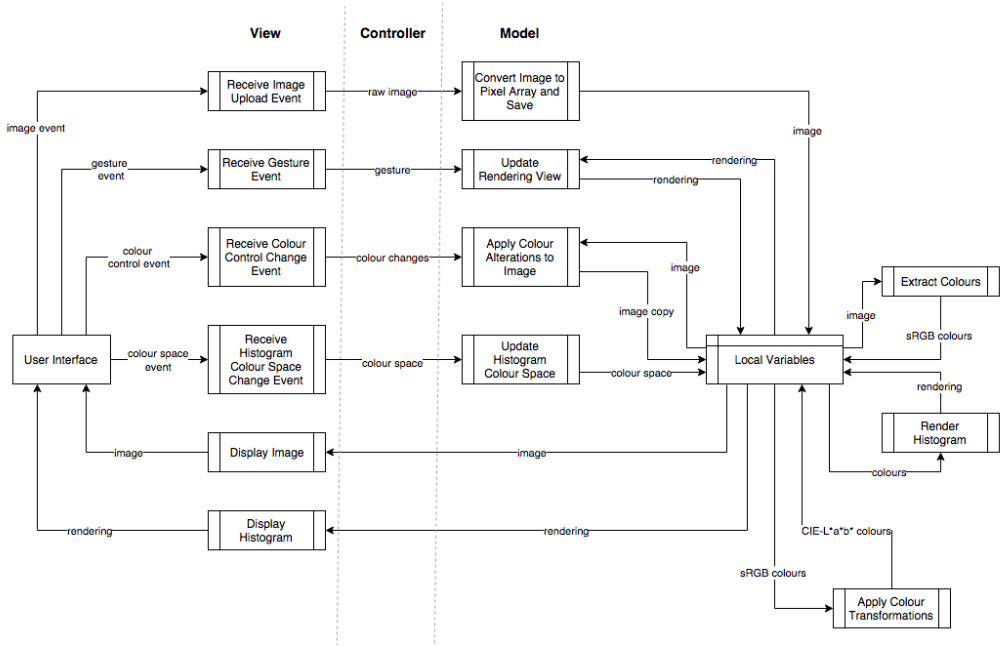


Figure 3.1 Data Flow Diagram

## 3.1. Model

The model processes and saves images and arrays of colours. Data needn't exist beyond the life of the program so databases are unnecessary and local variables are used instead. The model's components also contain the business logic of the program.

**3.1.1. Convert Image to Pixel Array and Save** An uploaded image should be converted from its raw format (.jpg, .png, etc.) to one that allows pixels to be iterated over, their colours to be queried, and uses the sRGB space. sRGB is chosen because it is one of the most widely used, minimising expensive transformations. Once converted, the image is stored in a local variable.

**3.1.2. Extract Colours** The pixels of an image will be iterated over and the RGB value of each pixel stored in an array, which will then be saved as a local variable.

**3.1.3. Update Histogram Colour Space** The user can change to view the histogram in the sRGB or CIE-L*a*b* space. This function updates the state of the model to reflect the user's current selection.

**3.1.4. Apply Colour Transformations** This function transforms an array of sRGB colours to the CIE-L*a*b* space. An overview of this transformation can be found in §1.2.3.

Figure 3.2 contains the formula to linearise sRGB's luminance to match CIE-XYZ's.

$$C_{linear} = \begin{cases} \frac{C_{srgb}}{12.92} & C_{srgb} \leq 0.04045 \\ \left(\frac{C_{srgb}+0.055}{1.055}\right)^{2.4} & C_{srgb} > 0.04045 \end{cases}$$

Figure 3.2 sRGB Linearisation Formula (Juckett, 2010)

Matrix multiplication is applied to the transformation matrix (Figure 3.3) and linearised sRGB vector to obtain the CIE-XYZ vector.

$$\begin{bmatrix} 0.412383 & 0.357585 & 0.18048 \\ 0.212635 & 0.71517 & 0.072192 \\ 0.01933 & 0.119195 & 0.950528 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

**Figure 3.3 sRGB to CIE-XYZ Transformation Matrix (Juckett, 2010)**

The CIE-XYZ values are inserted into the formula in Figure 3.4 and the CIE-L*a*b* values obtained.

$$\mathrm{x}_1 = \frac{X}{95.047}$$
$$\mathrm{y}_1 = \frac{Y}{100.000}$$
$$\mathrm{z}_1 = \frac{Z}{108.883}$$

$$\mathrm{x}_2 = \begin{cases} (x_1)^{\frac{1}{3}} & x_1 > 0.008856 \\ (7.787 \times x_1) + \frac{16}{116} & otherwise \end{cases}$$

$$\mathrm{y}_2 = \begin{cases} (y_1)^{\frac{1}{3}} & y_1 > 0.008856 \\ (7.787 \times y_1) + \frac{16}{116} & otherwise \end{cases}$$

$$\mathrm{z}_2 = \begin{cases} (z_1)^{\frac{1}{3}} & z_1 > 0.008856 \\ (7.787 \times z_1) + \frac{16}{116} & otherwise \end{cases}$$

$$\text{CIE-L*} = (116 \times y_2) - 16$$
$$\text{CIE-a*} = 500 \times (x_2 - y_2)$$
$$\text{CIE-b*} = 200 \times (y_2 - z_2)$$

**Figure 3.4 CIE-XYZ to CIE-L*a*b* formula (Irotek Group Ltd., 2014)**

**3.1.5. Render Histogram** A histogram of the colours extracted from the image, in the sRGB or CIE-L*a*b* space, will be rendered in this function. Once completed, the rendering will be saved in a local variable.

Colours extracted from the image will be in 24-bit colour. With a possible 16,777,216 colours not enough of the same colours will be present for any meaningful information to be drawn from the histogram: therefore colours should be reduced to 16-bit. Uniform colour quantisation achieves this by reassigning each 24-bit colour to their geometrically closest 16-bit colour.

The background of the rendering should a neutral grey with the frame of the chosen colour space in the centre. Pixels along the edges should be coloured by the colour at that point in the colour space and not interpolated. The colours will be plotted inside the space's frame 3D wireframe spheres, coloured by the colour they represent. The size of each plot will correspond to how many 24-bit colours mapped to it.

It is possible to display the full range of sRGB colours but CIE-L*a*b* is too large, which must be converted to sRGB for displaying. Preserving colour gradient is more important than accurately representing colour because CIE_L*a*b* was chosen for it's perceived uniformity: therefore colours will not be interpolated, but each colour transformed exactly.

#### 3.1.6. Update Rendering View

This function uses gestures from the UI to change the view in the rendering. All gestures move objects within the scene, rather than the camera. Single finger swipes or left mouse button drags will rotate the rendering around the centre point in the direction of movement. Two finger swipes or right mouse button drags will pan in the direction of movement. In/out pinching or down/up scrolling will zoom in/out on the centre point.

#### 3.1.7. Apply Colour Alterations to Image

This function changes the brightness, contrast, and saturation of an image. Alterations are applied to a copy of the sRGB image because changes are irreversible if values are clipped; the edited copy is saved to a separate local variable. Alterations are given as a number between -100 and +100.

Brightness is applied by adding or subtracting a uniform value, derived from the adjustment value, to the colour components. Figure 3.5 shows the formula to calculate a contrast value from the input (T) and apply it to a colour's components. Saturation is adjusted by transforming to the HSL space, changing the saturation value (S), and transforming back to sRGB: the transformation formulae can be found in Figures 3.6 and 3.7.

$$\mathrm{C} = ((100 + \mathrm{T}) \ / \ 100)^2$$

$$\mathrm{R}_{out} = (((\frac{B_{in}}{255} - 0.5) \times C) + 0.5) \times 255$$
$$\mathrm{G}_{out} = (((\frac{G_{in}}{255} - 0.5) \times C) + 0.5) \times 255$$
$$\mathrm{B}_{out} = (((\frac{B_{in}}{255} - 0.5) \times C) + 0.5) \times 255$$

**Figure 3.5 Contrast Formula (Esterhuizen, 2013)**

$$\mathrm{R}_1 = \frac{R}{255}$$
$$\mathrm{G}_1 = \frac{G}{255}$$
$$\mathrm{B}_1 = \frac{B}{255}$$
$$\mathrm{M} = \max(\mathrm{R}_1, G_1, B_1)$$
$$\mathrm{m} = \min(\mathrm{R}_1, G_1, B_1)$$
$$\Delta = M - m$$

$$\mathrm{H} = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times (\frac{G_1 - B_1}{\Delta} \mod 6) & M = R_1 \\ 60^\circ \times (\frac{B_1 - R_1}{\Delta} + 2) & M = G_1 \\ 60^\circ \times (\frac{R_1 - G_1}{\Delta} + 4) & M = B_1 \end{cases}$$

$$\mathrm{S} = \begin{cases} 0 & \Delta = 0 \\ \frac{\Delta}{1 - |2L - 1|} & otherwise \end{cases}$$

$$\mathrm{L} = 0.5(\mathrm{M} + \mathrm{m})$$

**Figure 3.6 sRGB to HSL (RapidTables, 2015)**

$$\mathrm{C} = (1 - |2L - 1|) \times S$$
$$\mathrm{X} = \mathrm{C} \times (1 - |\frac{H}{60^\circ} \mod 2 - 1|)$$
$$\mathrm{m} = \mathrm{L} - \mathrm{C}/2$$

$$(\mathrm{R}_1, G_1, B_1) = \begin{cases} (C, X, 0) & 0^\circ \le H < 60^\circ \\ (X, C, 0) & 60^\circ \le H < 120^\circ \\ (0, C, X) & 120^\circ \le H < 180^\circ \\ (0, X, C) & 180^\circ \le H < 240^\circ \\ (X, 0, C) & 240^\circ \le H < 300^\circ \\ (C, 0, X) & 300^\circ \le H < 360^\circ \end{cases}$$

$$(\mathrm{R}, \mathrm{G}, \mathrm{B}) = (\mathrm{R}_1 + m, G_1 + m, B_1 + m)$$

**Figure 3.7 HSL to sRGB (RapidTables, 2015)**

## 3.2. View

The view presents information from the model and handles events from the UI.

#### 3.2.1. Receive Image Upload Event

When the user uploads an image an event will be fired that contains the raw image file, which is caught by this function and passed on to the model for processing. JPEG, PNG, and GIF files should be supported; other formats are desirable but not required.

#### 3.2.2. Receive Gesture Event

Touch or mouse gestures on the rendering fire an event containing the gesture. This function extracts relevant information from the gesture passes it to the model through the controller, where the rendering view will be updated.

#### 3.2.3. Receive Colour Control Change Event

Adjustments to one of the three horizontal sliders will cause an event to fire, conveying which slider was moved and how far it's new position is from 0. This information is passed to the model, via the controller, applying colour alterations to the image.

#### 3.2.4. Receive Histogram Colour Space Change Event

Changing the histogram colour space setting fires an event containing the newly selected space, which is passed to the model so the rendering can be updated.

#### 3.2.5. Display Image & Display Histogram

These functions controls how the user-uploaded image and histogram rendering is displayed to the user. The image or rendering, fetched from the model by the controller, is placed in its assigned space in the UI.

### 3.3. Controller

The controller updates the model's state and the view's presentation of the model. Any data passed between the model and the view in Figure 3.1 goes through the controller. Functions within the controller either receive events from the view and update the model's state, or fetch data from the model and pass it to the view.

## 4. User Interface Design

The rendering will fill the application window, with the histogram in the centre; other UI components will be placed over the top of the rendering. In the top left corner is a holder for the user-uploaded image. Below this box is a labelled text box and button, allowing the user to select an image from their local file system. The three sliders below the image control the image's brightness, saturation, and contrast. In the top right are the histogram colour space options. A simple sketch of the UI can be seen in Figure 4.1.

Uploading an image, moving the sliders, or changing the colour space will fire an event containing the details of the interaction. An event will also be fired on mouse or touch gestures anywhere else in the window, interpreted as an attempt to move the rendering view.

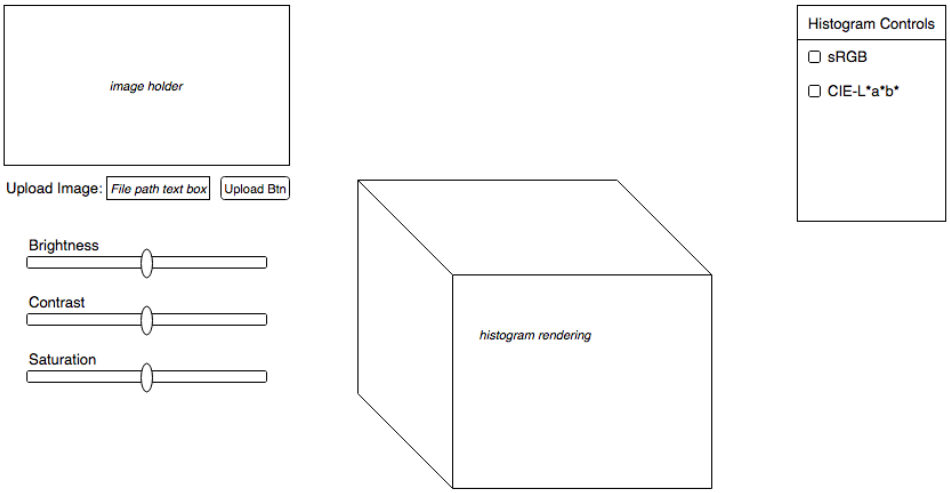


Figure 4.1 UI Sketch

# 5. Testing

A suite of unit tests will be used to test my program. Unit tests break features down into manageable pieces that can be tested independently, avoiding over complication and improving code coverage. In §3 my program is divided into component tasks, for each of which a unit test can be written. Firstly, the validity of input values should be tested. Tests should be written to cover a broad range of cases, in particular marginal and extreme cases, such as range limits. Intentionally erroneous data should also be inputted to test that it can be handled and that the appropriate error messages are displayed. Regarding presentation and user interaction, data sent to the display should be tested and events simulated. Testing the rendering is difficult to do programmatically, so a set of manual tests will be conducted; a list of clearly defined and unambiguous criteria will be defined to avoid oversights.

Test driven development (TDD) will also be practiced in the development of the program. TDD involves the repetition of short development cycles: a test defining desired behaviour is written and will initially fail; the code to pass the test is then written afterwards. TDD forces one to write code that is always correct and to constantly consider and revaluate the behaviour of the program, leading to better code.

# 6. Evaluation

I will judge the success of my project by whether I can resolve my hypotheses using the program. Using my program I will view the histogram of various images and experiment with adjusting the brightness, contrast, and saturation levels. The two things I will be looking for in the histograms are significant and obvious changes when colour adjustments are applied, and whether these changes are meaningful, determined by whether patterns can be spotted when applying similar adjustments. If I am able to consistently and accurately predict the effects of various colour adjustments to the histograms of unseen images, I will consider my hypotheses to be shown true: otherwise my hypotheses will be disproven for the moment, but there may still be changes to the histogram or colour controls that would be more successful. In either case, my project will be considered a success as it allows me to answer my hypotheses.

## 7. Time Plan

| Task | Term 2 | | | | | | | | | | | | Easter | | | | Term 3 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Basic UI outline | | | | | | | | | | | | | | | | | | | | | | | |
| Render 3D, interactive cube | | | | | | | | | | | | | | | | | | | | | | | |
| Plot spheres in cube | | | | | | | | | | | | | | | | | | DL = Deadline | | | | | |
| Upload image & display (updating UI) | | | | | | | | | | | | | | | | | | | | | | | |
| Extract colours from image | | | | | | | | | | | | | | | | | | | | | | | |
| Plot extracted sRGB colours on histogram | | | | | | | | | | | | | | | | | | | | | | | |
| Transform sRGB to CIE-L*a*b* | | | | | | | | | | | | | | | | | | | | | | | |
| Plot CIE-L*a*b* colours on histogram | | | | | | | | | | | | | | | | | | | | | | | |
| Update UI w/ colour controls | | | | | | | | | | | | | | | | | | | | | | | |
| Implement colour adjustment algorithms | | | | | | | | | | | | | | | | | | | | | | | |
| Update histogram w/ editted image | | | | | | | | | | | | | | | | | | | | | | | |
| Optimise rendering speed | | | | | | | | | | | | | | | | | | | | | | | |
| Debugging/contingency time | | | | | | | | | | | | | | | | | | | | | | | |
| Evaluation and hyptheses testing | | | | | | | | | | | | | | | | | | | | | | | |
| Write final report | | | | | | | | | | | | | | | | | DL | | | | | | |
| Prepare poster and demonstration | | | | | | | | | | | | | | | | | | | | DL | | | |

## 8. Prototype Progress

I have been developing a prototype of my program alongside my preliminary research as a proof of concept. I chose to implement it as a webpage written using JavaScript because it increases accessibility and allows me to use WebGL, a 3D rendering library that I have experience with using.

So far, the prototype has focused on the rendering and I have successfully rendered a histogram in the sRGB space. Starting with an interactive wireframe cube, I coloured the edges according to their position in the sRGB space, and then rendered a number of coloured wireframe spheres inside the cube. Colours were calculated using fragment shaders and a small amount of OpenGLSL code: this is run on the GPU and allows the colour of individual pixels to be set based upon their coordinates within the colour space. This approach ensures that colours are exact and not interpolated incorrectly by WebGL. Plots on the histogram are also grouped into a single geometry, increasing the speed of the rendering. I have also used the HTML5 file input to allow the user to upload an image, from which an array of sRGB colours is extracted. These colours can then be plotted on the histogram: however colours are plotted in 24-bit and not accumulated, so the histogram is too crowded and does not convey how many pixels there are of each colour.

From my prototype I have concluded that the tasks of extracting colours and rendering a histogram are a lot slower than I expected. I will look to optimise these operations, in particular colour quantisation and colour transformations, which I anticipate as being the most time and resource consuming functions of my program.

## Bibliography


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/

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

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

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

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