Investigating the Development of Colour Interpolation Techniques for GIMP

Thesis

Submitted in partial fulfilment of the requirements for the Degree of Bachelor of Science (Honours) in Computer Science and Information Systems at Rhodes University.

by

Vera Kukic

RHODES UNIVERSITY

Department of Computer Science

10 November 2003
Abstract

In this study we investigate recent automated greyscale image colourisation techniques using a custom developed GNU Image Manipulation Package (GIMP) plug-in. This process involves the transfer of colour between a compositionally similar source colour image and destination greyscale image. Emerging colourisation techniques offer an automated solution to the colourisation problem.

We have implemented a GIMP plug-in that was used to critically evaluate the colourisation process. A complete analysis was performed on the images by varying parameters to find the best balance between the neighbourhood size, sampling technique, weighted ratios of pixel luminance, standard deviation and mean. The plug-in developed offers an integrated greyscale image colourisation capability to the GIMP suite, and brings the power of automated colourisation to the GIMP user community.

We discovered that a trivial scan line uniform sampling technique produced comparable if not slightly better results than more computationally expensive techniques based on pixel neighbourhood statistical analysis. We conclude that it is important for images to correlate in composition and size.

Computer Review Categories

I.3.3  [Computer Graphics] Picture/Image Generation – Bitmap and framebuffer operations
I.3.7  [Computer Graphics] Three-Dimensional Graphics and Realism– Color, shading, shadowing and texture
I.4.3  [Image Processing and Computer Vision] Enhancement – Grayscale manipulation
I.4.9  [Image Processing and Computer Vision] Applications
Acknowledgments

To Shaun Bangay, Adele Lobb and George Wells, my supervisors, without whom this would not have been possible. Thank you for all your support, patience and assistance.

To Austin Poulton, thank you for all your support and for always believing in me.

To my family, especially to my parents, thank you for making this possible.

To my friends, for their support and the many hours spent in the lab working and laughing.

Thank you
Table of Contents

Abstract........................................................................................................................................... ii

Acknowledgments.......................................................................................................................... iii

1 Introduction............................................................................................................................. 2
  1.1 Motivation......................................................................................................................... 2
  1.2 Research Goals .................................................................................................................. 3
  1.3 Document Overview .......................................................................................................... 3

2 Related Work........................................................................................................................... 4
  2.1 Colourization ..................................................................................................................... 4
    2.1.1 Pseudocolouring ......................................................................................................... 4
    2.1.2 Colour Transfer between Images ............................................................................... 5
    2.1.3 Transferring Colour to Greyscale Images ................................................................. 8
  2.2 Texture Synthesis ............................................................................................................ 11
    2.2.1 Markov Random Field Texture Model ..................................................................... 12
  2.3 Summary .......................................................................................................................... 15

3 Design..................................................................................................................................... 17
  3.1 Generic Colourisation Process ........................................................................................ 17
    3.1.1 Image Acquisition .................................................................................................... 18
    3.1.2 Pre Processing .......................................................................................................... 18
    3.1.3 Colour Transfer ........................................................................................................ 18
  3.2 User Interface .................................................................................................................. 19
  3.3 System Design and Structure ........................................................................................... 21
    3.3.1 Plug-in Procedure ..................................................................................................... 23
    3.3.2 Design with Swatches .............................................................................................. 25
  3.4 Summary .......................................................................................................................... 27

4 Implementation..................................................................................................................... 28
  4.1 Implementation Architecture ........................................................................................... 28
  4.2 Environment .................................................................................................................... 28
  4.3 Plug-in Architecture ......................................................................................................... 29
  4.4 Development Issues ......................................................................................................... 31
    4.4.1 GUI development using GTK .................................................................................. 31
List of Figures

2-1 RGB-cube mapping [Gonzalez et al 1977] ................................................................. 5
2-2 Iaß colour space [Vieira et al 2003] ............................................................................. 7
2-3 3x3 Pixel Neighbourhood ............................................................................................. 9
2-4 Example of luminance image histogram [Maintz] ...................................................... 10
2-5 Use of Swatches to Colourise [Welsh et al 2002] ....................................................... 11
2-7 Grid for multi-scale MRF Modelling [Paget et al 1998] .............................................. 15

3-1 Generic Colourisation Process .................................................................................. 17
3-2 GIMP Plug-in menu .................................................................................................... 20
3-3 User Interface ............................................................................................................. 20
3-4 UML Design ................................................................................................................ 21
3-5 Detail of Plug-in Procedure ......................................................................................... 24
3-6 Swatches System Procedural Flow ............................................................................. 26

4-1 Layer Implementation Architecture ........................................................................... 28
4-2 Defining PLUG_IN_INFO .............................................................................................. 30
4-3 query() procedure ...................................................................................................... 30
4-4 Registering the Plug-in ............................................................................................... 31
4-5 run() procedure ......................................................................................................... 31
4-6 Content of Config File ................................................................................................ 33
4-7 Process listing depicting the grey2colour plug-in PID .............................................. 35
4-8 Debugging session ........................................................................................................ 35

5-1 Source and Destination Images .................................................................................. 37
5-2 Source Luminance Histogram .................................................................................... 38
5-3 Destination Luminance Histogram ............................................................................ 38
5-4 Scaled Source Luminance Histogram ....................................................................... 38
5-5 Distribution of Sample Pixels Luminance Histogram ............................................... 39
5-6 Uniform Sampling Results ........................................................................................ 40
5-7 Luminance Matching Error Ratio ............................................................................. 41
5-8 Results of Varying the Ratios ..................................................................................... 42
5-9 Results of Varying the Neighbourhood Size ................................................................. 43
5-10 Number of Samples versus Error Ratio ........................................................................ 44
5-11 Source and Destination Images .................................................................................... 44
5-12 The Effects of Sample Size ........................................................................................... 44
5-13 Results of Varying the Ratios with Random Tile Sampling ........................................... 45
5-14 Results of Varying the Neighbourhood Size with Random Tile Sampling ................ 46
5-15 Effects of Sample Size on the Error Ratio ........................................................................ 47
5-16 Execution Time versus Error Ratio ................................................................................ 48

List of Tables

5-1 Colourisation Parameters ............................................................................................... 39

8-1 Error Ratio ...................................................................................................................... 56
Chapter 1

Introduction

In this chapter we set the stage for the research undertaken, state the problem under consideration, lists our research goals and gives an overview of this document.

1.1 Motivation

Recently there has been an increase of articles dealing with automated image colourisation techniques, one such area is adding colour to greyscale images. Colour is added to greyscale images to increase their visual appeal. The process of colourization can be applied to black and white photographs, classic movies and scientific images. Moreover, the integration of automated colourisation techniques with in a common graphics and image manipulation package would be great benefit to ordinary users. They are able to easily colourise images in an automated fashion, as opposed to traditional colour mapping methods. An integrated colourisation technique is more useful than a stand alone application since it allows the user to manipulate and alter the images with all the other available functionality within the package.

Hence, it was decided to explore and evaluate these emerging colourisation techniques in a widely used open source graphics package known as GIMP (GNU Image Manipulation Package). The benefits of this are thus, not only can we use the GIMP framework as an implementation and evaluation platform but also provide a useful colourisation plug-in for the wider GIMP user community.

Thus, the objective of this project is to explore colour interpolation in the GNU Image Manipulation Package (GIMP) framework. The aim is to develop colour interpolation plug-ins specifically for colourising greyscale images for GIMP.
This document describes the approach used to colour greyscale images using the information from a colour source image as suggested by Welsh et al [2002].

1.2 Research Goals

The research objectives of our study are articulated below:

1. Develop an image colourisation plug-in for GIMP.
2. Investigate and critically evaluate the process of colouring greyscale image using the developed plug-in.

We intend to implement both the general procedure of colouring greyscale image by using the information of an entire source colour image and by transferring colour between selected swatches with images. We perform these tasks using the plug-in framework provided by GIMP.

1.3 Document Overview

The remainder of the thesis comprises of the following chapters. Chapter 2 discusses the research relevant to image interpolation specifically for the area of colourizing images. In Chapter 3 we describe design model and specify the solution structure. Chapter 4 deals with general implementation issues and explanations of the procedures followed. The discussion of results is in Chapter 5. Finally, in Chapter 6 the application and possibilities for future work are described followed by the general conclusion of the whole project.
Chapter 2

Related Work

This chapter describes the related work relevant to this study. We examine the ideas of changing an image’s colour space, transfer of colour information between colour images, transfer of colour from colour images to greyscale images and texture synthesis. The chapter focuses on recent work by Welsh et al [2002] and their algorithm for transferring colour to greyscale images which forms the basis for our development.

2.1 Colourization

2.1.1 Pseudocolouring

Pseudocolouring is a technique used to extract information from an image by applying colours different from the natural colours in the image to highlight certain features. Pseudocolour is a colour mapping that is designed to enhance the visibility and detectability of objects within an image by a human observer. Pratt [1978] describes this method as “image enhancement”. Klein [2003] explains that pseudocolouring is the process by which different colours are assigned to various digital numbers (DN). Digital Numbers are values stored in pixels of an image that represent the amount of light reflected to the sensor. Gonzalez et al [1977] describe pseudocolouring as an automatic process of mapping luminance values to colour values. The objective is thus to assign a colour to each pixel based on its luminance. Pseudocolour mapping is a technique used for adding colour to greyscale images such as X-rays, Magnetic Resonance Imaging (MRI), scanning electron microscopy ultrasound and other images in which colour does not exist. This approach does not alter the information of the image, as no extra information is introduced.

Lehman et al [1997] describe the pseudocolour mapping mathematically by a transformation of a curve in a colour space. Figure 2-1 contains a representation of the RGB colour space.
with two pseudocolour mappings. Mapping A, the main diagonal in the RGB-cube, represents the achromatic path through all shades of grey. The path follows from black to white in ascending order of all the grey values. Each grey value is mapped to a specific colour in the RGB colour space. The colour is defined by the corresponding coordinates in the colour space. According to Lehman et al [1997] for colour and continuous-brightness to be realized the transformation curve should follow a spiral path through the colour space along the diagonal as described by Mapping B.

Figure 2-1 RGB-cube mapping [Gonzalez et al 1977]

The authors conclude that for medical imaging keeping the original brightness progression is very important and the process of pseudocolouring is suitable for this mapping. Thus pseudocolouring is used to emphasize the detailed structure of greyscale images. However, we require context sensitive transformation of colour between two images and this approach is not sufficient enough to produce reasonable results.

2.1.2 Colour Transfer between Images

Reinhard et al [2001] describe the process of altering an image’s colour. Their method is a form of colour correction that uses information from a source image to alter the colours in a destination image. The colour characteristics are transferred from the source to the target image but the luminance value of the target image remains the same. For this modification to
be possible the colour channels must be modified in tandem. The \( la\beta \) colour space that has decorrelated channels was developed by Ruderman et al [1998]. Both images are converted to the \( la\beta \) colour space to make the modification on the image’s data. For the target image to take on the source image’s look and feel the distribution of the data needs to be known. The mean and standard deviation are calculated for both images for each axis separately.

The results are then converted back to the RGB colour space from the \( la\beta \) space. The quality of the results depends on the composition of the two images. Images of similar composition should be chosen for this process of transferring colour. If the images vary greatly in composition, the source image has a lot of grass and the target image has a large sky area, the transfer of statistics will fail. Reinhard et al [2001] introduces the idea of swatches to overcome this problem. Swatches are used to sample different areas of an image and the statistics are calculated separately on each of the selected areas. Then the input image’s pixels are scaled and shifted according to the statistics of each of the swatches creating clusters. The next step is to calculate, for each pixel, the distance to the centre of each of the swatches and divide it with the standard deviation of that cluster. The weighted contributions of each cluster are calculated using the scaled inverse distance. The pixels are blended with the weights inversely proportional to the normalized distances and the final colour is yielded.

2.1.2.1 Decorrelated Colour Space

Ruderman et al [1998] developed the \( la\beta \) colour space to minimize the correlation between the three axes of the colour space. The idea for this colour space is based on the perception that the human visual system is best suited for image processing. The \( la\beta \) colour space is best suited for situations when the colour channels need to be manipulated separately. \( l \) is the achromatic luminance channel and \( a \) and \( \beta \) are the two achromatic channels corresponding to yellow-blue and red-green opponent channels. Change made in one channel should minimally affect the other channels. The \( la\beta \) colour space is logarithmic and thus the uniform changes in the channels are equally detectable. Figure 2-2 illustrates the \( la\beta \) colour space where the \( a \) and \( \beta \) values approximate to zero for neutral colours white, grey and black. Vieira et al [2003] define \( a \) as a measurement of red for a positive or green for a negative and \( \beta \) as a measurement of yellow for \( \beta \) positive or blue for \( \beta \) negative.
Figure 2-2: Colour space 

For colour transfer, the luminance value of the greyscale image is retained while the colour information is accepted from the source colour image. To be able to separate the luminance from the colour values, the $\ell a\beta$ colour space is used. As a result, both the source and colour image are converted to $\ell a\beta$ colour space. $\ell a\beta$ is a transform of the LMS cone space. A normal observer has trichromatic colour vision that is determined by cones of the LMS cone space. Every colour can be represented by three variables. LMS stands for long, medium and short wavelength, and the output of the cones determines the colour [Zhang, 2003]. Reinhard et al. [2001] explain the process of conversion from the RGB colour space to the $\ell a\beta$ colour space is done in three steps. This transformation is done using matrix manipulation. First, the RGB values are converted to XYZ tristimulus values, and then these values are converted to the LMS cone space. The XYZ tristimulus values do not correspond to real colours; X is a super-saturated purplish red, Y is a supersaturated form of the real spectral green, and Z is a supersaturated form of the real spectral blue [Bangay, 2003].
The data from the LMS space is then converted to the logarithmic space, Equation (2.3), to eliminate skew and finally it is transformed to the \( la\beta \) colour space by using the transform in Equation (2.4).

\[
\begin{align*}
L &= \log L \\
M &= \log M \\
S &= \log S \\
\end{align*}
\]  

(2.3)

\[
\begin{bmatrix}
L \\
M \\
S
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
\frac{1}{\sqrt{3}} & 0 & 0 \\
0 & \frac{1}{\sqrt{6}} & 0 \\
0 & 0 & \frac{1}{\sqrt{2}}
\end{bmatrix} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & -2 \\
1 & -1 & 0
\end{bmatrix} \begin{bmatrix}
L \\
M \\
S
\end{bmatrix}
\]  

(2.4)

After colour matching the image is converted back to its original RGB colour space. This is done in reverse order of the RGB to \( la\beta \) transformation. The first step is to multiply the \( la\beta \) values by the inverse matrix manipulation of the Equation (2.4). Then the pixel values are raised to the power of ten to go back to linear space. The last transformation can be done directly between LMS and RGB using the inverse of Equations (2.1) and (2.2) shown by Equation (2.5).

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} = \begin{bmatrix}
4.4679 & -3.5873 & 0.1193 \\
-1.2186 & 2.3809 & -0.1624 \\
0.0497 & -0.2439 & 1.2045
\end{bmatrix} \begin{bmatrix}
L \\
M \\
S
\end{bmatrix}
\]  

(2.5)

### 2.1.3 Transferring Colour to Greyscale Images

Previous work has been done by Reinhard et al [2001] on transferring colour between colour images. Welsh et al [2002] extend this idea to colouring greyscale images by transferring the colour information from a source colour image to a destination greyscale image. This approach attempts to minimize the amount of human labour required in colouring greyscale images. Only the chromatic values from the colour image are transferred so as to attain the original luminance values of the target image. This means that the three-dimensional RGB values of the colour image need to be assigned to an image which only varies along one dimension.
A general algorithm for colouring greyscale images is developed by Welsh et al [2002]. The first step of this algorithm is to convert both the colour and greyscale images into the \( \text{laß} \) colour space. The \( \text{laß} \) space allows for colour to be transferred from one image to the other without changing the luminance values in the target image. Welsh et al [2002] uses the colour space transformation procedure directly from Reinhard et al [2001]. In order for the colour information to be transferred from the colour image to the greyscale image, each pixel in the greyscale image must be matched to a pixel in the colour image. The matching procedure uses the luminance value and neighbourhood statistics of the pixel for comparison. The neighbourhood statistics are precomputed over the entire image and they consist of the standard deviation of the luminance in the pixel neighbourhood. In Figure 2-3 the lighter shaded pixels represent the pixel neighbourhood and the darker coloured pixel represents the pixel for which the statistics are being calculated.

![Figure 2-3 3x3 Pixel Neighbourhood](image)

The next step in the procedure is to create luminance histograms of the two images. The image luminance histogram is the frequency plot of the grey values occurring in an image. It represents the image’s grey shade diversity. Twan Maintz [2003] describes the luminance histogram diagrammatically as shown in Figure 2-4. This is the representation for twelve pixels only, histograms for complete images are denser and cover the complete range of all the luminance values \( \{0, \ldots, 255\} \).
Luminance remapping, proposed by Hertzmann et al [2001], describes the process of scaling the luminance histograms of the two images. The luminance of the target image remains unchanged and the luminance of the source image is shifted and scaled to fit the histogram of the target image. This transformation brings the histograms of the colour image and the greyscale image into correspondence. Hertzmann et al [2001] apply a linear map that matches the means and variances of the luminance distributions. Let $Y(p)$ be the luminance of a pixel in image A (the colour image), then the remapping procedure is defined as:

$$Y(p) \leftarrow \frac{\sigma_B}{\sigma_A} (Y(p) - \mu_A) + \mu_B$$

(2.4)

where $\mu_A$ and $\mu_B$ are mean luminances and $\sigma_A$ and $\sigma_B$ are the standard deviations of the luminances of the two images. The mean and standard deviation values are computed with respect to the luminance distributions of the two images A and B. The next step is to go through the greyscale image pixel by pixel in scan line order to find the best corresponding match in the colour image. The a and b values of the source image are transferred to the target image once the best matching pixel is found.

This process is not perfect and thus some images will not be coloured correctly as the same luminance values can represent different colours. If corresponding regions of the two images do not have similar luminance values the target image will not appear correct. This approach works well on images when corresponding regions between two images also correspond in luminance values. To improve the process of transferring colour between images swatches are introduced.
2.1.3.1 Swatches

Swatches are introduced to match similar areas between two images. A swatch is a rectangle that is used to sample the colours which are within its selection. They allow more user interaction in the colour transfer procedure and improve results. The user selectively chooses the regions to create swatches. The same colour transfer procedure is used as in the global matching procedure except only between the source and target swatches. The use of swatches reduces the number of comparisons needed to be made by the greyscale image to find the perfect match. The comparisons are thus only made between the relating swatches. The rest of the corresponding region is coloured in using texture synthesis [Efros et al 1999; Efros et al 2001]. Efros et al use the $L_2$ metric for calculating distance to find texture matches. $L_2$ is the metric used to define the distance between two points $(L_2)^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$. They define the error distance $E$ using the $L_2$ metric between neighbourhood $N_g$ in the greyscale image and neighbourhood $N_s$ in the colourised swatch as:

$$E(N_g, N_s) = \sum_{p \in N} (I(p) - S(p))^2$$

(2.5)

where $I$ is the greyscale image, $S$ is the luminance channel of the colourised swatch and $p$ represents the pixels in these neighbourhoods. The advantage of using swatches, as stated by Welsh et al [2002], is the selective colour transfer between the swatches which prevents the pixels with similar neighbourhood statistics but from the wrong part of the image from corrupting the target swatch colours. Swatches allow for interactive colouring and give the user the power to colour areas that do not correspond in composition. Figure 2-5 shows the application of swatches during the colour transfer procedure.

![Figure 2-5 Use of Swatches to Colourise [Welsh et al 2002]](image)

2.2 Texture Synthesis

Paget et al [1998] define texture as visual characteristic of an image segment that identifies it as belonging to a unique class. Each class is associated with a particular physical interpretation such as hair, grass, water or sand. Efros et al [1999] state that texture synthesis can be defined as a visual pattern on a 2-D plane which at some scale has a stationary distribution. Texture itself can only be defined with the greyscale component of the image,
the luminance [Chen et al 2001]. Texture is a local neighbourhood property which requires a finite neighbourhood and it cannot be defined from one pixel.

The aim of texture analysis is to mathematically model spatial pixel value interaction within an image [Paget et al 1998]. Textures can be segmented from the rest of the image if the spatial interactions for that particular texture can be modelled. The difficult part of texture analysis is trying to uniquely model the textures. Textures are classified as being regular or stochastic. Regular textures consist of repeating texels and stochastic textures have no explicit texels. Efros et al [1999] define texels as a scale of the texture elements. It is a general understanding that all real-world textures lie somewhere in between these two extremes.

Efros et al [2001] suggest that the two most important tasks in statistical texture synthesis are picking the right set of statistics to match and finding an algorithm to match them. According to Paget et al [1998] a texture model needs to capture all the relevant characteristics that uniquely define a texture. For our research we decided to use the texture synthesis algorithms suggested by Efros et al [1999] for texture segmentation. They implemented texture synthesis via a non-parametric Markov Random Field (MRF) model. Markov Random Field models are widely used in image restoration, region segmentation and texture synthesis. The MRF model is able to synthesise highly structured and stochastic types of textures.

2.2.1 Markov Random Field Texture Model

Paget et al [1998] define the property of MRF as: a variable $X_s$ at site $s$ on a lattice $S = \{s = (i, j) : 0 = i, j < M\}$, can be equal to any value $x_s$, but the probability $X_r = x_r$ depends upon the value $x_r$ at sites neighbouring $s$. The neighbouring sites are defined as those sites where $r \in N_s \subset S$, where $N_s$ represents the neighbourhood of $s$. The neighbourhood system is then defined as the set of all the neighbourhoods $N = \{N_s \subset s \in S\}$. Paget et al [1998] then go on to define the MRF by the local conditional probability density function (LCPDF) with respect to the neighbourhood system $N$:

$$P(X_s = x_s | X_r = x_r, r \in N_s) \quad s \in S$$ (2.4)

For an image to be modelled using the MRF each pixel in the image needs to be considered as a site on a lattice. The grey scale value of that pixel, its luminance, is the value of that site.
Thus $x_s$ is set to equal the value associated with that pixel. A condition needs to be satisfied where by the neighbourhood systems are symmetrical. The neighbourhood system $N^o_s = \{ N^o_s, s = (i, j) \in S \}$ is defined as:

$$N^o_s \{ r = (k, l) \in S : 0 < (k - i)^2 + (l - j)^2 \leq o \}$$ (2.5)

The diagram below depicts the neighbourhood systems of different orders $o$ as described by Paget et al [1998]. Figure 2-6 (a), (b), (c) shows a neighbourhood system of order $o = 1$, $2$ and $8$ respectively. The neighbourhood for the site $s = (i, j) = \square$ and $r = (k, l) \in N_s = \square$.

![Neighbourhood Systems](https://example.com/fig2-6a.png) ![Neighbourhood Systems](https://example.com/fig2-6b.png) ![Neighbourhood Systems](https://example.com/fig2-6c.png)

*Figure 2-6 Neighbourhood Systems [Paget et al 1998]*

Based on the MRF model Efros et al [1999] assume that each pixel is independent of its neighbourhood. This means that the probability distribution of brightness values for a pixel given the brightness values of its neighbourhood is independent of the rest of the image. The neighbourhood window is modelled as a square window around a pixel, as shown above. Efros et al [2001] allow for the size of the window to be a free parameter that is determined by the user based on the user’s perception of how stochastic the texture is. Thus the Markov Random Field model is formed by modelling the value at the centre pixel in terms of a conditional probability with respect to its neighbouring pixels values.

The next step in the MRF model according to Paget et al [1998] is to estimate the local conditional probability density function by building a multi-dimensional histogram of the image. The multi-dimensional histogram represents the nonparametric MRF model. The histogram is built from the frequency values of the occurrence of the different values of the luminance for a predefined neighbourhood. As the number of dimensions in the histogram increases, the data will become very sparsely distributed, irrespective of the number of grey levels represented. To make the histogram distribution smooth the Prazen density estimation via Kernel smoothing is used [Paget et al 1996].
To obtain the LCPDF, density estimation on the multi-dimensional histogram needs to be performed. The Prazen-Window density estimator is used for non-parametric density estimation. The Prazen-Window density estimator smooths each sample data point in the multi-dimensional histogram over a large area. The estimate density function $\hat{f}$, according to Paget et al [1998], is given by:

$$\hat{f}(y) = \frac{1}{nh^d} \sum_{k=1}^{n} K\left(\frac{1}{h}(y - Y_k)\right)$$  \hspace{1cm} (2.6)

where $n$ is the sample of real observations $Y_1, \ldots, Y_n$ in the $d$-dimensional histogram. $K$ is the kernel function which defines the shape of the smoothing. The window parameter $h$ defines the size of the kernel. The kernel function $K(y)$ is defined to be a standard multivariate normal density function:

$$K(y) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2} y^T y\right)$$  \hspace{1cm} (2.7)

The optimal window parameter $h_{opt}$ was derived under the assumption that $f$ is a standard multivariate normal density function and is defined by Silverman as:

$$h_{opt} = \sigma \left\{ \frac{4}{n(2d + 1)} \right\}^{1/(d+4)}$$  \hspace{1cm} (2.8)

Some texture characteristics are better resolved at some resolutions than others. The next step is to relax the image at various resolutions through multi-scale relaxation. Paget et al [1998] continue by synthesizing the texture using a method known as stochastic relaxation (SR). The texture synthesised using SR from a complete MRF model should be similar to that of the original texture. The process of synthesizing a texture using SR starts with any image at the top level and iteratively updates pixels in the image with respect to LCPDF sequentially working down. Paget et al [1998] suggest different algorithms, Metropolis algorithm, Gibbs sampler and Iterative Conditional Modes (ICM), for applying the stochastic relaxation. The SR is constrained with respect to the previous grid level. Figure 2-7 diagrammatically explains the process of multi-scale texture synthesis. A small portion of three levels of a 2:1 multi-grid hierarchy is shown. The grid at level $l=0$ represents the original image, where each intersection point is site $s \in S$. 

14
To eliminate the problem of estimating the model in a high-dimensional space Paget et al [1998] introduce the pixel temperature function to reduce the dimensionality of the histogram. This allows for the use of large neighbourhood systems for the representation of the texture. The pixel temperature function is used to determine the confidence that each pixel is of the correct value. The confidence associated with each pixel is expressed as a value $0 = t_s = 1$, where 1 represents complete confidence and 0 represents no confidence. Paget et al [1998] state that the pixel confidence helps constrain the SR process while implementing local annealing. This process is described by Paget et al [1998] in six steps. The first step is to initialize the pixel confidences: if the site has low confidence assign 0 and if it has high confidence assign 1. A pixel has high confidence if its value was obtained from a previous grid level. The next step in the process is to modify the kernel smoothing of the histogram so that the pixel that is being iterated is less dependent on the pixels with low confidence. The pixel confidence is increased, in step three, each time the pixel is iterated. The next step involves sampling from the distribution to choose the next pixel to be updated. If any pixel confidence is less than one then step two needs to be repeated. Once this level has been completed and an equilibrium state has been reached, the image is propagated down to the next level where it undergoes further relaxation using the same process.

2.3 Summary

A wide range of colourization techniques exist for various applications. Each application is different in respect of what category the image to be colourised falls into. Pseudocolouring is a technique typically used for colouring scientific images such as X-rays, Magnetic Resonance Imaging (MRI), scanning electron microscopy ultrasound and other images in

Figure 2-7 Grid for multi-scale MRF Modelling [Paget et al 1998]
which colour does not exist. This technique works on the basis of assigning different colours to different levels of grey within an image. It is not suitable for colouring black and white photographs as each greyscale value is assigned a specific colour, thus this process does not colourise with context. As the same luminance values can correspond to different colours a new method was proposed, transferring colour information of one image to another. An appropriate source image is chosen and its characteristics are applied to the destination image. The two images, source and destination, are converted to the decorrelated $l_a$ colour space to cater for easy colour transfer.

The colourization method was extended to cater for colouring greyscale images. Greyscale images are represented by a one dimensional distribution, hence only luminance channels can be matched between the two images. The statistics within the pixel’s neighbourhood are used to guide the matching process between two images because a single luminance value could represent different parts of an image. Swatches are used to improve and aid the matching process. The colour is transferred between the two swatches and then the image is coloured from the information in the target coloured swatch using the $L_2$ metric.
Chapter 3

Design

The following chapter describes the design of our colourisation solution. More specifically we present our generic colourisation process, user interface and an overview of the systems’ core functions. Our colourization plug-in is named grey2colour.

3.1 Generic Colourisation Process

We present a generic colourisation model that is based on the classical model of digital image processing [Gonzelz et al 1992].

![Generic Colourisation Process Diagram](image_url)

*Figure 3-1  Generic Colourisation Process*
The systems described in the literature on the colourization processes are standalone or ad hoc implementations and do not provide a flexible and extensible design for researchers to build upon. We attempt to provide a generic, flexible and extensible solution to image colourisation as depicted in Figure 3-1. The generic model consists of three major phases, image acquisition, pre-processing and colour transfer. The design conforms to the classical model of digital image processing and to the generic colourisation processes articulated in the literature. Almost all the colourisation algorithms described rely on colour space mapping, statistical analysis, sampling and colour transfer.

3.1.1 Image Acquisition
The input data are two images or two rectangular selections: a colour source image or selection and a destination greyscale image or selection. We constrain the source colour image to be a 24 bit RGB type and the destination image an 8-bit greyscale image. In our implementation of the grey2colour plug-in, the images are acquired from GIMP at run time. The colour information of the source image is to be transferred to the greyscale image without changing the original luminance of the destination image.

3.1.2 Pre Processing
The pre-processing stage for colourisation requires the colour space mapping and the computation of the statistics for the entire image. The image colour space mapping requires both images to be converted to the laß colour space. This computation is done so as to allow for luminance extraction from each pixel. Statistics are computed over the entire image or the selection. The statistics produced include the mean, standard deviation and luminance histograms.

3.1.3 Colour Transfer
The colour transfer stage of the colourization process calls for luminance scaling, sampling from the colour image, finding the best matches for each greyscale pixel and finally transferring the colour. The source and destination images or selections need to correspond in luminance distribution for the colour transfer process to be meaningful. Thus the luminance histogram of the source image is scaled. The source image luminance histogram is scaled and
shifted to fit the distribution of the destination image luminance histogram. The scaling of the luminance histograms brings the two images into correlation. As a result of this scaling better samples and matches are found between the two images. The next step is to sample pixels from the source colour image. Luminance values may be sampled from the source colour image using various sampling techniques. The different sampling techniques produce different results. Once the samples are gathered best matches need to be found in the greyscale image. The greyscale image is traversed in scan line order to find the best matches. Each pixel in the destination image is matched with the best approximating pixel in the source image. Once the best match is found for each pixel the colour information from the source sample that best matches the luminance of the destination pixel is transferred. The luminance value of the destination pixel remains the same and thus the destination image brightness remains the same. The only alteration made to the image or selection is the addition of colour channels.

3.2 User Interface

The plug-in should present a simple, intuitive interface that allows any user to colour greyscale images with no training on how to use the software. The user needs to open two images, a colour image as a source image and a greyscale image as a destination image. If numerous images are open, the interface allows the user to select the images that need to be manipulated from a drop down menu. The interface has an advantage of letting the user preview both images at the same time and it shows the updates of the colourization process. The interface also informs the user at what stage of the process the execution is using a status bar on the greyscale image. The plug-in is instantiated from the greyscale image under the colour section in the menu (Figure 3-2).
Since this was an experimental plug-in we decided to put all the functionality behind the
colourise button in the user interface to accommodate more efficient and faster
experimentation. Obviously, various colourisation parameters may be set using widgets on
the plug-in GUI in future implementations, but for our purposes colourisation parameters
were provided in a configuration file. The plug-in’s design is based on other plug-ins that are
available under GIMP and thus it adheres to the design standards of GIMP. The user interface
is shown in Figure 3-3.
3.3 System Design and Structure

Figure 3-4 UML Design

Figure 3-4 presents the UML design of our system. While our implementation is not object oriented, since the plug-in and supporting libraries were implemented in C, it was designed using a number of abstract data types that are equivalent to the classes described in the UML diagram above. The object oriented model provides a flexible and extensible design. There are a number of wrapper pixel data related classes that provide rich set of functions for image processing developers.

**Pixel** represents a single pixel within an image or selected region. It includes colour space attributes and x, y coordinates, and simple functions for converting between the RGB and laβ colour spaces. **PixelArea** is a matrix of Pixels and represents an image or rectangular selection. It provides high level functions for manipulating rectangular pixel areas. It is a convenient and intuitive wrapper to the GIMP API. Greyscale images have one byte of information per pixel, the luminance, whereas colour images have three or four bytes of data per pixel, the RGB values. As a result, image manipulation at the byte level is difficult in GIMP. The pixelArea structure provides a wrapper to byte level GIMP functions which operate on byte arrays instead of pixels. The PixelArea structure also has a number of functions which are useful for image processing. It provides function for the conversion...
between the two colour spaces RGB and \( l_a \), computation of statistics and functions to convert the byte array provided by GIMP into a two-dimensional array.

**PixelList** is a linear array of Pixel structures and is useful for sampling from pixel areas. The only functions within the PixelList is the sortByLuminance which is used to sort the samples in order of luminance intensity. The sorted pixel list speeds up the search for best pixel luminance matches between the two images.

The implementation of abstract data types needs specific create and delete functions. The createPixelArea() function uses the pixelArea structure and sets up a 2-dimensional empty array of type Pixel with the height and width dimensions acquired from the image. deletePixelArea() function frees the memory that was occupied by the pixelArea and deallocates all the pointers.

**PixelStatistics** structure includes attributes such as the luminance mean, luminance standard deviation and histogram. The **RunStatistics** structure holds attributes that are relevant to computational information such as the execution time and the number of source and destination pixels. In our design it is shown that the Statistics structure is a super class of the PixelStatistics and RunStatistics structures. As inheritance is not possible in C we implemented these structures to resemble the conceptual design as much as possible.

The **Grey2ColourPlug-in** class represents the plug-in and its user interface. It provides the functionality needed for running the plug-in. To get the required parameters for each run the plug-in class uses a configuration file. We have a **ConfigParams** structure which sets the parameters for execution and takes care of reading and writing of information from and to files.

The Colourisation Technique classes allows for various different colourisation processes. Hence, we have a base class **ColouriseTechnique** that consists of a source and target pixel area and a virtual method colourise that must be implemented in sub classes. The colourise methods will provide the implementation of specialised colourisation techniques. Hence, we have the Welsh Technique as a specialisation of the base class which provides an implementation of Welsh et al [2002] algorithm. Our design allows for additions of different
sub classes which implement the specialised colourisation technique, thus allowing for easy expansion.

3.3.1 Plug-in Procedure

The detail of the plug-in procedure, the functions that are called and their order of execution is described in Figure 3-5. For the process to be initialised GIMP needs to be started and the two images that are required for the colourizing procedure need to be opened from the image files. The Grey2Colour plug-in is instantiated from the greyscale image. The createPixelArea() function creates a manageable pixel area from the whole image or selection. setPixelAreafromPixelRegion() converts the image data, which is provided as a byte array from the GIMP run time, into a more manageable and intuitive matrix structure. pixelAreatoLAB() converts the pixel region that was set by the previous function into the decorrelated colour space \( l \alpha \beta \). The image statistics are computed using the computePixelAreaStats() function. computePixelAreaStats() only takes one parameter: the area for which the stats need to be computed for, and it calculates the mean and standard deviation for this area. The scaling and shifting of the source image luminance histogram to fit the distribution of the destination luminance histogram is done by the function mapLuminance(). The next step is to sample from the scaled and shifted source image. This can be done using the uniformTiledSampling() function which samples the centre pixel of each tile. The source image is divided into a number of rectangular regions known as tiles and the samples and their neighbourhood statistics are calculated for each tile. This function returns a PixelList of all the sampled pixels. The best matching pixel between the images is found by using findBestLuminocityMatch() function where by the best match for a given luminosity is found from a PixelList. For each pixel in the image in scan line order the best luminosity match is found. PixelList holds the sorted list of all the luminosity values of the source image. If the luminance value found is smaller than a specified error, it is accepted and the colour is transferred. Once the best matches are found for all the pixels in the greyscale image, it is converted back to the RGB colour space by pixelAreatoRGB(). setPixelRegionfromPixelArea() converts the information from pixelArea structure back into the GIMP byte array so that it can be displayed. The data that is gathered from this process is dumped to a file for analysis purposes by using the function dumpPixelAreatoFile(). Once the greyscale image is in RGB colour space the preview is updated using the built in GIMP function gimp_drawable_update().
Figure 3-5 Detail of Plug-in Procedure
3.3.2 Design with Swatches

The colour transfer process is the same as for the general procedure but the colour transfer only happens between selected swatches. The plug-in is also instantiated from the greyscale image and both images are converted to the laß colour space. From there the procedure changes from the original. The user selects a region in the colour image and a region in the greyscale image which is to be coloured with the information from the swatch in the colour image. The colour is transferred using the general matching procedure where the colour swatch luminance distribution histogram is transformed to fit the luminance distribution of the greyscale swatch. After a sampling technique is employed the luminance values are compared and the best matches are found. The colour is then transferred only to the selected swatch. These few steps are repeated until all the different regions to be coloured are selected. Once the swatch process is completed the source image is discarded as no more information is needed from it.

The rest of the colouring process is done using the colour information from the coloured parts of the greyscale image. The colour for the rest of the image is generated using texture synthesis. For each greyscale pixel a corresponding luminance value needs to be found within the coloured swatch. The pixel neighbourhood is included in the comparison of luminance values. If the pixel luminance and the neighbourhood statistics match an area in the swatch the pixel is coloured with the corresponding colour. The process is done for each pixel in the entire image. Figure 3-6 shows the procedural flow when swatches are used in the colourization process.
Figure 3-6 Swatches System Procedural Flow

Start

GIMF

Open Colour Image

Open Grayscale Image

Grey2Colour

Convert Images to .jpg Colour Space

Get Colour Swatch

Select Swatch on Grayscale Image

Compute Statistics

Map Luminance Histograms

Find Best Luminance Match

Transfer Colour

Are all searches selected?

YES

Discard Source Image

Generate Colour for Rest of File Image

Update Preview

END

Image Files
3.4 Summary

We propose a generic colourisation process design. The design is split up into three stages, image acquisition, pre processing and colour transfer. The colour transfer stage of the process design is generic and thus it allows for use of different sampling and matching techniques. As a result of our design these different sampling and matching techniques can be inherited from the colourization technique. The user interface is simple and intuitive and allows the user to view both images at the same time. The design considers two possibilities, the general procedure and the procedure using swatches. These are explained in detail using flow diagrams.

A number of core functions and data structures and the relationships between them are presented. These provide an overview of the complete solution’s design.
Chapter 4

Implementation

In this chapter we review system-related implementation issues. We discuss the choice of language and the use of open-source development, the plug-in architecture and the development issues such as the design of the GUI.

4.1 Implementation Architecture

<table>
<thead>
<tr>
<th>Grey2Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIMP</td>
</tr>
<tr>
<td>GTK</td>
</tr>
<tr>
<td>glib</td>
</tr>
<tr>
<td>OS - Linux/Windows</td>
</tr>
</tbody>
</table>

*Figure 4-1 Layer Implementation Architecture*

Figure 4-1 is the representation of the layered implementation architecture. The plug-in Grey2Colour lies on top since it uses the GIMP and GTK API functions which in turn rely on the glib, the gimp library. They are all layered on top of the operating system which in this case can be either Linux or Windows.

4.2 Environment

We were limited in choosing a programming language since GIMP only caters for C and Perl. Our choice was based on prior knowledge, available resources and the functionality that the language offers. Perl is a scripting language and is not widely used. In terms of learning there is not much written in Perl. C is a low level, powerful language with a large and diverse user base. GIMP was originally developed using C and thus we decided to go with this
option. But, C may have memory leaks and thus memory has to be dynamically allocated and de-allocated with care.

GIMP is an open source image manipulation package. Most of the functionality present in other image manipulation packages is present in GIMP as well. It supports a large number of image file types such as BMP, JPEG, PNG, GIF, TIFF. GIMP is expandable and extensible and allows for improvement and augmentation through the use of plug-ins. These plug-ins add functionality to GIMP. All the functionality and libraries are freely available and can be downloaded from the GIMP website. GIMP itself can be run on Windows as well as on Linux. In the beginning stages of the project we used Linux for development. As time progressed we managed to successfully install GIMP on Windows 2000 and build the developed files. As a result the implementation is cross platform.

We implement our colourizing process as a plug-in for GIMP using C. GIMP operates with great processing speed on a Pentium IV. The development machine runs RedHat 9 Linux and Windows XP at 1.8GHZ with 512MB RAM. Tests were done to compare processing speeds and these indicate that GIMP can successfully be run with fewer requirements.

4.3 Plug-in Architecture

The definition for a plug-in from the Free On-line Dictionary of Computing is “A file containing data used to alter, enhance or extend the operation of a parent application program” [Unknown, 2003].

In order for the main GIMP program to invoke the developed plug-in, the plug-ins need to implement a specified interface. Each plug-in will have a unique GUI depending on its functionality. GIMP interfaces with plug-ins by searching for files of the appropriate format and incorporating them into its menu structure. The search is only done across the plug-in’s sub directories.

There are some things that a plug-in must have. The plug-in has to register itself in the procedure database (PDB) when it is queried and it has to have a function to run when GIMP calls for the procedure. Every C program must have a main() function and it is the first
function that is called when the program is executed. This function in GIMP is simply defined as MAIN(). MAIN is defined in the libgim/gimp.h file, and it calls the gimp_main function. The gimp_main function sets up the signal handling and the communication between the plug-in and GIMP. GIMP plug-ins are queried on start-up and this is when they register themselves in the PDB. The PLUG_IN_INFO global variable is used by the query procedure. PLUG_IN_INFO consists of pointers to four functions. The functions that are important are the query() and run() functions, the first two functions are not normally used and thus are set to NULL.

```c
/* Setting PLUG_IN_INFO */
GimpPlugInInfor PLUG_IN_INFO =
{
    NULL, /* init_proc */
    NULL, /* quit_proc */
    query, /* query_proc */
    run,  /* run_proc */
};
```

**Figure 4-2 Defining PLUG_IN_INFO**

The query() function, see Figure 4-3 makes the call to the gimp_install_procedure which registers the plug-in in the PDB. The first few parameters in the gimp_install_procedure are self-explanatory. The menu_path parameter is a string that describes where in the menu the procedure should be installed. GIMP uses this information to place the plug-in in the menu and to decide what type of procedure it is. The parameters which are necessary for the procedure are determined by the type of the procedure. Our plug-in is in the <Image> menu location and thus requires three parameters, an integer, an image and a drawable. Other parameters can be added to the ones required but the required parameters must come first and in order. The *param points to an array of parameter definitions and nparams is the number of parameters as an integer which is equal to the length of the array.

```c
static void query (void)
{
    static GimpParamDef args[] =
    {
        { GIMP_PDB_INT32, "run_mode", "Interactive, non-interactive" },
        { GIMP_PDB_IMAGE, "image_id", "Input image (unused)" },
        { GIMP_PDB_DRAWABLE, "drawable_id", "The drawable to be colorized" }
    };

gimp_install_procedure (char *name, char *blurb, char *help, char *author, char
    *copyright, char *date, char *menu_path, char
    *image_types, GimpPDBProcType type, int nparams, int
    *return_vals, GimpParamDef *params, GimpParamDef
    *return_vals);
```

**Figure 4-3 query() procedure**
The position where the plug-in is incorporated into the GIMP structure is defined within the plug-in in the query() procedure. An example of the code used to do this is shown in Figure 4.4.

```
//register plug-in in the PDB

/*plug in procedure (grey2colour, */
/*"Pick up colour distribution from the colour image, */
/*and applies it to the greyscale image."), */
/*"This plug-in colorizes the contents of the specified */
/*greyscale layer", */
/*"Vera Kukic", */
/*"g99k0588@campus.ru.ac.za", */
/*"2003", */
/*N("<image>/Filters/Colors/Grey2Colour"), */
/*"RGB*, GRAY*", */
/*GIMP_PLUGIN, */
nargs,nreturn_params, args, return_params);
```

**Figure 4-4 Registering the Plug-in**

The run() procedure sets the status code return value. This function is where the action begins and it is called to run the procedure of the plug-in that is installed in the procedural database. The run() procedure makes sure that the plug-in is called correctly and that the return values are set correctly.

```
static void run (gchar *name,
    gint nparams,
    GimpParam *param,
    gint *nreturn_vals,
    GimpParam **return_vals);
```

**Figure 4-5 run() procedure**

### 4.4 Development Issues

#### 4.4.1 GUI development using GTK

GNU provides a user interface design tool that unfortunately is not easy to use. Even thought this tool aids in rapid development and removes the burden from the programmer we decided to develop the interface directly. The GUI was designed using the GTK library, which is the GIMP interface API [Turner, 2000]. The interface was designed with the help of the GTK tutorials that are available on-line [Gale *et al* 2000: Gale *et al* 2003]. GTK uses a hierarchical approach to interface design. This means that the layout of the GUI was designed using a table and then attaching different objects at the right cells in the table.
We found that drawing rectangular selections, swatches on the preview window more difficult than originally anticipated. There is not an example or plug-in where such a method has been employed and GTK signal handling is not well documented.

4.4.2 Experiences with GIMP plug-ins

A basic understanding of the GIMP plug-in structure is required before a plug-in can be written. The package is open source and all the code is available for analysis. We were able to run plug-ins and see their functionality and then compare it to the code used. GIMP needs to be recompiled for each new addition to take effect. This is slightly inconvenient as it requires fGIMP to be closed and re-opened after each compilation.

GIMP allows for low level access of one dimensional character arrays which contain an entire area of an image or the drawable. GIMP has gimp_pixmap_region_get and gimp_pixmap_region_set functions which are used for pixel manipulation. These are slow in terms of processing and hinder the speed of compilation. It is advised in the documentation not to use these functions if doing a large amount of processing on images since these functions degrade performance considerably. To combat this problem we added a layer of abstraction to the GIMP API for easier manipulation of images or areas of images.

4.4.3 Catering for extensions

Several features of GIMP and C make it possible to include provisions for extensions to the colourization plug-in in the future. Currently the plug-in is specific to colour transfer between a source colour image and a destination greyscale image with similar compositions but this can be extended to cater for two colour images or images of different compositions.

We have designed a generic preview procedure which can be implemented in other plug-ins. The code can simply be extracted and reused in cases where two previews are required. By creating separate files that calculate the neighbourhood statistics and perform calculations on pixels within an image we made it possible for other plug-ins to use the same functions without rewriting the code.
4.5 Configuration and Logging

To make the process of using different parameters in each run easier we created a configuration file which we aptly called config. We used the configuration file as opposed to doing manual configuration via the plug-in interface, since this was easier and more efficient for experimentation. The config file holds the parameters that can be changed by the user to view different results. Figure 4-6 is an example of the values the config file can have. The first parameter represents the weighting that will be used in calculating the proportions of luminance, standard deviation and mean associated with each pixel. Sample Method allows the user to choose which sampling technique to use, from the ones available, during the sampling process. The number of row and column tile parameters are used to subdivide the image. The user specifies the granularity which the image is to be subdivided into by giving the number of tiles across and down. The neighbourhood size represents the size of the area around the pixel that is to be included in the calculations of neighbourhood statistics.

```
LUM RATIOS: 0.95 0.05 0.01
SAMPLE METHOD: 1
NUM COL TILES: 16
NUM ROW TILES: 20
NEIGHBOURHOOD SIZE: 5
```

Figure 4-6 Content of Config File

The statistics and timing data from each colourization run is written to a uniquely named log file, whose data is then used for analysis. The source image luminance statistics data is captured to the log file before and after the mapping process. The original data of the greyscale image is also dumped to the log file. This allows us to graph the difference and confirm that the transformation of the source luminance values has taken place. The log file also contains run statistics such as the time to execute, the number of pixels in the image, number of samples taken and the luminosity error ratio.

4.6 Debugging GIMP Plug-Ins

GIMP Plug-ins are not standalone applications since they cannot be launched directly from the operating system. Plug-in binaries have a special header block which only can be interpreted by the GIMP runtime; hence they may only be launched from within GIMP. Although GIMP plug-ins, once launched, run in their own process space, attempting to start them directly from the operating system, will be unsuccessful, since the operating system is
unable to provide the necessary GIMP runtime parameters, which must be passed to every GIMP plug-in.

At first inspection this appears to pose a problem for debugging a non-standalone executable. The implication is that developers may have to compile and debug GIMP from source as a whole, in order to identify errors in a plug-in. Fortunately this is not the case, since a GIMP plug-in runs as an independent process, hence we can avail ourselves of the little used “attach” command in the GNU debugger. Typically, in fact almost all, plug-ins have some form of GUI interface that is displayed and allows the user to interact with, setting parameters, before executing the transformation the plug-in is supposed to apply to the image. The trick is to intercept control of the plug-in with the GNU debugger (gdb) between the time the plug-in interface is displayed and before the user interaction takes place. The debugging process we adapted is elucidated below:

1. Firstly, we compile the plug-in with debugging information in the binary. This usually swells the plug-in binary size by 3 or 4 times since additional meta information is compiled into the binary, including the original variable name symbol table, and explicit references from machine code to the original source code. This is done using the \textit{-g} option of gcc.
2. Start GIMP and open an image or images to manipulate and launch the plug-in that is to be debugged. GIMP will load and launch the plug-in as a separate process.
3. Start a gdb session loading the plug-in’s executable file. In our case: \texttt{gdb grey2colour}
4. Obtain a process listing of running processes from the operating system. On UNIX and Linux platforms simply execute \texttt{ps -aux} and find the process identifier for the plug-in. On windows platforms run the task manager to find the relevant PID.
5. In the gdb session attach to the running plug-in process using the `attach <PID>` command, in this case `attach 1572`. The GNU debugger effectively takes over execution control of the plug-in. It is then possible to set break points, watch variables, step execution of the plug-in and so on. Figure 4-8 depicts a typical debugging session.
4.7 Summary

We developed our Grey2Colour plug-in based on a layered implementation architecture. We chose to develop the colour transfer procedure as a plug-in for GIMP using C. Every plug-in needs to have: #include libgimp/gimp.h, PLUG_IN_INFO, MAIN, a query function with a call to gimp_install_procedure and a run function which sets the status code return value. The GUI was developed directly without the use of a GUI designer and this did not hinder the resulting interface. We make provisions for extensions by developing generic files and interfaces. We created a configuration file to speed up the process of passing parameters to the plug-in. To accommodate for extraction of data we developed log files which store all the information about the images, prior and post to processing. The building of plug-ins is explained in detail since plug-ins are not stand alone applications and thus debugging of a non-standalone executable is difficult.
Chapter 5

Results

The primary goal of this chapter is to critically assess the accuracy of image colourisation both perceptually and quantitatively. In this chapter we describe our quantitative experiments of the colourisation process and evaluate the effect of various parameters including neighbourhood size, sampling technique and number of samples. Finally we discuss the results based on the empirical data and the quality of the sample colourised images presented.

5.1 Luminance Remapping

Luminance remapping is the process of shifting and scaling the source luminance histogram to fit the distribution of the destination luminance histogram. Figure 5-2 and Figure 5-3 show the luminance histograms of the source and destination image respectively. The histogram for the transformed source image is shown in Figure 5-4.

![Source Image](image1.png) ![Destination Image](image2.png)

*Figure 5-1 Source and Destination Images*
From the above graphs it can be seen that the source image is scaled to fit the destination image distribution. The scaling process aids in luminance matching by allowing better matches between the sample luminance values from the source image and the luminance values from the greyscale image. The samples gathered from the scaled source image match the scaled source distribution as shown in Figure 5-5.
5.2 Colourisation Parameters and Statistics

There are a number of experimental parameters that affect the final results of the colourisation process. Table 5-1 shows the sampling methods and the parameters that are used by them.

<table>
<thead>
<tr>
<th>Colourisation Parameters</th>
<th>Sampling Method</th>
<th>Uniform</th>
<th>Uniform Tiled</th>
<th>Random Tiled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Number of Tiles: nCols X nRows</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Samples per Tile</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Sampling Interval</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Pixel Neighbourhood</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Neighbourhood Size</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Luminosity Ratios</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

*Table 5-1 Colourisation Parameters*

Three different sampling techniques were evaluated and are described in more detail in the following sections.

**Sample Size:** The sample size refers to the number of sample pixels used to colour the target greyscale image of area. The samples are drawn from the source colour image and populate a list which is used to transfer colour to the target image.

**Pixel Neighbourhood:** These parameters influence the pixel neighbourhood statistical analysis (see Section 2.1.3). The luminosity ratios are the weighted averages of the pixel luminance, the pixel neighbourhood mean and the pixel standard deviation.

A number of statistics are generated from the colourisation process. The most important quantitative metric is the luminosity matching error ratio. It is the sum of the matching error over the total number of pixels in the destination image. The luminosity matching error is the difference between the luminosity of the best matched sampled pixel and the luminosity of a
given target pixel. Sampling ratio is the sample number of pixels over the total number of pixels in the source image. Timing data is also generated for each experiment. For each of the factors tested we provide an overview of the experiment, present and assess the results and evaluate their effect on the overall colourisation process.

5.2.1 Sampling Technique

We developed three sampling techniques to evaluate the colour transfer process. The sampling technique is the methods by which pixels are sampled from source colour image and used to populate the colour transfer list. The first technique relies solely on uniformly sampling across the entire image. The other two sampling techniques both make use of tiles and neighbourhood statistical analysis. The first of the two methods uses uniform sampling and the second uses random sampling within the tiles. The algorithms used for each sampling method are presented as detailed pseudo code in Appendix D.

5.2.2 Uniform Sampling

Uniform sampling is the scan line sampling of the entire source image. The sampling interval is set in the configuration file. The more samples there are the longer it takes for the matching process to take place. By varying the interval we can see the difference in results as shown in Figure 5-6. It can be seen as long as the interval is small and the number of samples is large the image has no artefacts. From Figure 5-6 it is evident that the colour range across the image decreases as the number of samples decrease.

![Figure 5-6 Uniform Sampling Results](image-url)
5.2.3 Uniform Tiled Sampling

Uniform tiled sampling requires that the image be segmented into rectangular areas known as tiles. The samples for this method are the centre pixels of each tile, thus to produce better results the number of tiles in the image should be large. This method also makes use of the weighted average ratios of the pixel luminance, the pixel neighbourhood luminance mean and the pixel neighbourhood standard deviation.

5.2.3.1 Weighted Ratios

The best matching colour sample is selected based on the weighted average of the luminance, the standard deviation and the mean. The ratios affect the colour transfer process thus we attempt to find the best balance between these ratios. The following section looks at the varied ratios of these weights and determines which combination produces the best results. The rest of the variables are kept constant to be able to account only for changes in weighted ratios. The image is divided into tiles, 20 by 20, with the neighbourhood size set at 20.

![Error Ratio](image)

*Figure 5-7 Luminance Matching Error Ratio*

Table 8-1 in Appendix A, shows the error ratio dependent on the different weightings between the three values. Run number 60, with the following ratio 0 for pixel luminance, 0 for mean of luminance values and 1 for standard deviation value, shows the worst result. This is because the standard deviation is only a fraction of the actual pixel luminance value. Run
number 12 shows the best error ratio for this method of luminance matching, with 0.6 for pixel luminance and 0.4 for the mean value of the luminance neighbourhood. The results in Figure 5-7 show the difference between the error ratios. It can be seen that the differences are numerically small and almost not noticeable but the difference is visible in the final results of the colourised images (Figure 5-8).

![Figure 5-8 Results of Varying the Ratios](image)

From the results we can deduce that the best ratio is obtained when the pixel luminance value and pixel neighbourhood mean are used. The standard deviation of the pixel neighbourhood incorrectly affects the luminance matching process. Thus, the best match is obtained when 60 percent of the pixel luminance value and 40 percent of the neighbourhood mean value is used.

5.2.3.2 Neighbourhood Size

Neighbourhood size affects the value of the pixel mean and the pixel standard deviation of the luminance thus it is important to pick the size of the neighbourhood carefully. If a large neighbourhood size is selected, information that is not relevant to the particular pixel will be included in the calculations of statistics and it will affect the luminance matching process. The smaller the neighbourhood size the better the results. We must also keep in mind that for high resolution images the neighbourhood size can be bigger. Using the best combinations of
the weighted ratios we tested the impact of the neighbourhood size. Figure 5-9 shows the results that were obtained with different neighbourhood sizes and weightings for the matching procedure.

![Figure 5-9 Results of Varying the Neighbourhood Size](image)

The results in Figure 5-9 show that as the neighbourhood size gets bigger the artefacts in the image increase. Thus from our results we can deduce that the best neighbourhood size is 3.

### 5.2.3.3 Number of Tiles

In this sampling technique we introduced tiles as a means of dividing an image into sections. The number of tiles that the image is segmented into affects the number of samples generated for the luminance matching procedure. Therefore, the number of samples is predetermined by the number of tiles in the image. Figure 5-10 shows the ratio between the number of tiles, which is the same as sample size, to the error ratio.
The source and destination images used for this testing are shown in Figure 5-11. These images were chosen because of their size and colour span.

**Figure 5-10 Number of Samples versus Error Ratio**

**Figure 5-11 Source and Destination Images**

**Figure 5-12 The Effects of Sample Size**
We conclude that as the number of samples per image increase the error ratio decreases and the colourisation results produced are better as can be seen in Figure 5-12.

5.2.4 Random Tiled Sampling

Random tiled sampling relies on the tiles within an image and the number of samples per tile. Unlike the uniform tiled sampling, this method randomly selects the same number of samples in each tile. The number of samples per tile and the number of tiles is again set in the configuration file.

First we test out the weighted averages suggested by the uniform tiled sampling technique. Our results confirm that the best weighted average is the same as for the previous method, which can be seen in Figure 5-13. The next step is to confirm that the best neighbourhood size should be 3 as suggested by the results from the uniform tiled sampling technique. Figure 5-14 shows the comparison between two neighbourhood sizes and from the images it is clear that better results are produced using the neighbourhood of size 3.

![Figure 5-13 Results of Varying the Ratios with Random Tile Sampling](image)

**Figure 5-13 Results of Varying the Ratios with Random Tile Sampling**
Figure 5-14 Results of Varying the Neighbourhood Size with Random Tile Sampling

5.2.4.1 Sampling Ratio

The sampling ratio is the ratio between the number of samples taken from the image and the total number of pixels in the image. The number of tiles and samples per tile affect the results obtained. The number of samples per tile determines the number of random samples taken from a single tile. The more tiles there are, the smaller the artefacts appear in the colourised images. We conclude that the more samples there are, whether they are obtained via a large number of tile or a large number of samples within a tile, the better the results obtained are.

5.2.5 Processing Time

As the number of samples increases so does the processing time. The processing time is affected by the number of samples or the number of tiles and number of samples per tile. The more samples, or tiles and samples per tile there are the more comparisons need to be made in the luminance matching process thus increasing the processing time.
5.2.6 Analysis

For analysis purposes we computed several experiments to see the effect sample size has on error ratio using the different sampling techniques. We compare the performance of our sampling techniques in Figure 5-15. From our graph it can be seen that the random tiled sampling method produces the worst results as the error ratio is very high in comparison to the other two methods. The best performing sampling method, the uniform sampling method, shows the lowest error ratio in relation to sample size.

The execution time of the colourisation process improves as the error ratio gets bigger. The bigger the error ratio the fewer comparisons were made during the sample luminance matching process and thus the computation time is shorter. Figure 5-16 shows the three way relationship between the sampling method, error ratio and time. We can see that the method that produces the smallest error range and timing range is the uniform sampling technique.
5.3 Summary

From our experiments it can be seen that the best results are achieved with a mix of all the variables. The best weighted ratios in conjunction with the small neighbourhood size give best results for the tiled sampling techniques. From our results we deduce that the best weighted ratio is a combination of pixel luminance and pixel neighbourhood mean in ratio of 6:4 respectively. Based on experimental results, the best performing technique, with the smallest error ratios is the uniform sampling technique.
Chapter 6

Conclusion

In this study our objective was to develop a plug-in for image interpolation which colourises greyscale images based on input from a colour image. We implemented a colourisation algorithm which colours greyscale images and uses a suitable data structure that is both flexible and allows provision for future work.

6.1 Research Contributions

The objectives set forth in Section 1.2 have been achieved.

1. We developed a flexible and extensible design for a generic colourisation model that allows for different implementations of luminance remapping, sampling techniques, and matching methods. Based on the hierarchy of the design different colourisation techniques can be implemented. Future researchers can adopt our design for experimentation and development purposes. This design has been implemented as a GIMP plug-in.

2. The plug-in integrated into the GIMP toolset which allows users to colourise images for the applications listed in Section 6.2. The developed plug-in allows novices users to easily colourise images.

3. We used the GIMP framework to evaluate the colourisation process. Using experimental analysis we were able to determine the best combination of factors that affect the colourisation process. We evaluated three sampling techniques: uniform sampling, uniform tile sampling and random tile sampling. The best colourisation technique is the uniform sampling technique even though it is simple and computationally inexpensive. We conclude that this colourisation implementation can be successfully used to colour
greyscale images using the correct number of samples, neighbourhood statistics and weighted ratios where necessary.

6.2 Application

Colourisation of greyscale images spans a large number of applications. The basic application of our work is the colourisation of black and white photographs and pictures. Our colourisation implementation can be extended to colour black and white film. Key frames on the film are colourised and the frames in between are colourised using the originally coloured frames.

Our implementation can also be applied to scientific images such as X-rays, Magnetic Resonance Imaging, Scanning Electron Microscopy and any other greyscale images that need additional information in terms of colour. We can choose a specific source colour image depending on what results we want to obtain from colouring scientific images.

6.3 Future Work

The colourisation process can be extended to use swatches to allow for more precision in colouring of greyscale images. Welsh et al [2002] suggest $L_2$ as a distance metric used to find texture matches during the colourisation process. This method does not deal well with faces as faces have a lot of information packed closely together. An extension to our work could be an implementation of a technique that deals well with small surface areas that have attributes that are similar in luminance.

In our implementation the source image is selected manually, this process can be automated via a database. The source image can be searched for based on the luminance histograms of images, image signatures and textures. This implementation would make the entire process automated and would only require user intervention in the form of selecting the greyscale image to be colourised.

While we explored a number of sampling techniques, new and faster techniques that use different benchmarks can be implemented. Our research bases colour transfer only on
luminance information, further research can be done into the area of colour transfer combining both texture and luminance.
References


## Appendix A

### Error Ratio Table

Table 8-1 shows the error ratios obtained from different weighted averages using the uniform tile sampling method. The pixel luminance, pixel neighbourhood mean and pixel neighbourhood standard deviation columns represent the proportion of each values used in the calculation of the final luminance value.

<table>
<thead>
<tr>
<th>Run</th>
<th>Weights</th>
<th>Pixel Luminance</th>
<th>Pixel Neighbourhood Mean</th>
<th>Pixel Neighbourhood Standard Deviation</th>
<th>Error Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00407761</td>
</tr>
<tr>
<td>2</td>
<td>0.95</td>
<td>0.05</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00407761</td>
</tr>
<tr>
<td>3</td>
<td>0.90</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00404386</td>
</tr>
<tr>
<td>4</td>
<td>0.90</td>
<td>0.0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.11183</td>
</tr>
<tr>
<td>5</td>
<td>0.80</td>
<td>0.2</td>
<td>0.0</td>
<td>0.1</td>
<td>0.00459041</td>
</tr>
<tr>
<td>6</td>
<td>0.80</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.118311</td>
</tr>
<tr>
<td>7</td>
<td>0.80</td>
<td>0.0</td>
<td>0.2</td>
<td>0.2</td>
<td>0.301344</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00492877</td>
</tr>
<tr>
<td>9</td>
<td>0.70</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.12191</td>
</tr>
<tr>
<td>10</td>
<td>0.70</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.31184</td>
</tr>
<tr>
<td>11</td>
<td>0.70</td>
<td>0.0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.587157</td>
</tr>
<tr>
<td>12</td>
<td>0.60</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00377964</td>
</tr>
<tr>
<td>13</td>
<td>0.60</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td>0.123733</td>
</tr>
<tr>
<td>14</td>
<td>0.60</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.322518</td>
</tr>
<tr>
<td>15</td>
<td>0.60</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0.601072</td>
</tr>
<tr>
<td>16</td>
<td>0.60</td>
<td>0.0</td>
<td>0.4</td>
<td>0.4</td>
<td>0.928606</td>
</tr>
<tr>
<td>17</td>
<td>0.50</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00438065</td>
</tr>
<tr>
<td>18</td>
<td>0.50</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.125938</td>
</tr>
<tr>
<td>19</td>
<td>0.50</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
<td>0.333454</td>
</tr>
<tr>
<td>20</td>
<td>0.50</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.465913</td>
</tr>
<tr>
<td>21</td>
<td>0.50</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.614964</td>
</tr>
<tr>
<td>22</td>
<td>0.50</td>
<td>0.1</td>
<td>0.4</td>
<td>0.4</td>
<td>0.943986</td>
</tr>
<tr>
<td>23</td>
<td>0.50</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>1.28683</td>
</tr>
<tr>
<td>24</td>
<td>0.40</td>
<td>0.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00420861</td>
</tr>
<tr>
<td>25</td>
<td>0.40</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.128042</td>
</tr>
<tr>
<td>26</td>
<td>0.40</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.342632</td>
</tr>
<tr>
<td>27</td>
<td>0.40</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.629077</td>
</tr>
<tr>
<td>28</td>
<td>0.40</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.959421</td>
</tr>
<tr>
<td>29</td>
<td>0.40</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
<td>1.30609</td>
</tr>
<tr>
<td>30</td>
<td>0.40</td>
<td>0.0</td>
<td>0.6</td>
<td>0.6</td>
<td>1.62564</td>
</tr>
</tbody>
</table>
## Table 8-1 Error Ratio

<table>
<thead>
<tr>
<th>Run</th>
<th>Weights</th>
<th>Pixel Luminance</th>
<th>Pixel Neighbourhood Mean</th>
<th>Pixel Neighbourhood Standard Deviation</th>
<th>Error Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0.00446238</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0.3</td>
<td>0.6</td>
<td>0.1</td>
<td>0.129121</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>0.3</td>
<td>0.5</td>
<td>0.2</td>
<td>0.345919</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>0.3</td>
<td>0.4</td>
<td>0.3</td>
<td>0.643168</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.771475</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0.3</td>
<td>0.3</td>
<td>0.4</td>
<td>0.974879</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td>1.32243</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0.3</td>
<td>0.1</td>
<td>0.6</td>
<td>1.64494</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>0.3</td>
<td>0</td>
<td>0.7</td>
<td>1.96475</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0.2</td>
<td>0.8</td>
<td>0</td>
<td>0.0046655</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>0.2</td>
<td>0.7</td>
<td>0.1</td>
<td>0.129924</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0.2</td>
<td>0.6</td>
<td>0.2</td>
<td>0.349266</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>0.2</td>
<td>0.5</td>
<td>0.3</td>
<td>0.657386</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.990356</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0.2</td>
<td>0.3</td>
<td>0.5</td>
<td>1.33817</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>1.66423</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>0.2</td>
<td>0.1</td>
<td>0.7</td>
<td>1.98405</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0.2</td>
<td>0</td>
<td>0.8</td>
<td>2.24997</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>0.1</td>
<td>0.9</td>
<td>0</td>
<td>0.00454454</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.1</td>
<td>0.8</td>
<td>0.1</td>
<td>0.130233</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>0.1</td>
<td>0.7</td>
<td>0.2</td>
<td>0.352808</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>0.1</td>
<td>0.6</td>
<td>0.3</td>
<td>0.67174</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>1.00588</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>0.1</td>
<td>0.4</td>
<td>0.5</td>
<td>1.35391</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>0.1</td>
<td>0.3</td>
<td>0.6</td>
<td>1.68353</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>0.1</td>
<td>0.2</td>
<td>0.7</td>
<td>2.00335</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>0.1</td>
<td>0.1</td>
<td>0.8</td>
<td>2.29871</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>0.1</td>
<td>0</td>
<td>0.9</td>
<td>2.47084</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00460506</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.50422</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Sample Colourised Images
Appendix C

Example Log File

This is an example of a log file generated by the plug-in for evaluation and experimentation purposes.

Grey2Colour run log - Time ID: 1068413141

RUN CONFIG PARAMETERS

Luminosity Ratios:
- Pixel Luminance: 0.600000
- Neighbourhood mean luminance: 0.400000
Neighbourhood std. dev. luminance: 0.000000
Sample Method: 3
Samples Per Tile: 5
Grid: Columns = 20 Rows = 20
Neighbourhood Size: 3
Actual Neighbourhood Size: 0

PIXEL AREA: SOURCE PIXEL AREA BEFORE LUMINANCE SCALING

internal id: 0 from image: D:\gimp-dev\sample_images\colourtrees.jpg
offset in image: x = 0 y = 0
width = 236 height = 158 number of pixel = 37288 colourspace = RGB
Statistics
Luminosity: mean = 3.54916 Std Deviation = 0.440003
Luminosity Histogram
Value   Freq
0.000000   000000
0.100000   000000
0.200000   000000
0.300000   000000
. . . . . . . . . .

PIXEL AREA: DESTINATION PIXEL AREA

internal id: 1 from image: D:\gimp-dev\sample_images\greytrees.jpg
offset in image: x = 0 y = 0
width = 245 height = 157 number of pixel = 38465 colourspace = RGB
Statistics
Luminosity: mean = 3.49874 Std Deviation = 0.440003
Luminosity Histogram
Value   Freq
0.000000   000000
0.100000   000000
0.200000   000000
0.300000   000000
. . . . . . . . . .
PIXEL AREA: SOURCE PIXEL AREA AFTER LUMINANCE SCALING

internal id: 0 from image: D:\gimp-dev\sample_images\colourtrees.jpg
offset in image: x = 0 y = 0
width = 236 height = 158 number of pixel = 37288 colourspace = RGB
Statistics
Luminosity: mean = 3.49874 Std Deviation = 0.443559
Luminosity Histogram
Value   Freq
0.000000   000000
0.100000   000000
0.200000   000001
0.300000   000001

PIXEL LIST: SAMPLED LIST

List Size=2000
Actual NeighbourhoodSize if used = 3
Statistics
Luminosity: mean = 3.53153 Std Deviation = 0.422792
Luminosity Histogram
Value   Freq
. . . . . . . . . .
2.000000   000005
2.100000   000015
2.200000   000005
2.300000   000010
. . . . . . . . . .

RUN STATISTICS

Source pixels: 37288
Destination pixels: 38465
Sample pixels: 2000
Sample/Source pixel ratio: 0.053637
Total Luminosity matching error: 145.384
Luminosity matching error ratio 0.00377964
TIMINGS
Total time: 1.625000 sec

59
Appendix D

Pseudo-code Extracts

The full system source code may be found on the accompanying CD-ROM in the source code directory.

D.1 Colourisation Process

Pseudo code for the colourisation driver process within the grey2colour plug-in.

```plaintext
getColourisationConfig(conf)
srcDraw = gimp drawable_get()
gimp_pixel_rgn_init(srcPixelrgn, srcDraw, xLeft, yTop, xRight-xTop, yBottom-yTop)
srcPixelArea = createPixelArea()
setPixelAreafromPixelRgn(srcPixelrgn, srcPixelArea);
pi xel Ar eat oLAB(srcPixel xel Area);
computepi xel AreaStats(srcPixel xel Area);

    if (gimp_convert_rgb(destImgID))
    {
        destDraw = gimp drawable_get()
gimp_pixel_rgn_init(destPixelrgn, destDraw, xLeft, yTop, xRight-xTop, yBottom-yTop)
        destPixelArea = createPixelArea()
        setPixelAreafromPixelRgn(destPixelrgn, destPixelArea)
        pixelAr eoLAB(destPixel xel Area)
        computepi xel AreaStats(destPixel xel Area)
        scaleLuminance(srcPixelArea, destPixelArea)
        switch(samplingMethod)
        {
            1 :   list = uniformSampling(srcPixelArea, conf, stats)
            2 :   list = uniformTiledSampling(srcPixelArea, conf, stats)
            3 :   list = randomisedTiledSampling(srcPixelArea, conf, stats)
        }
        computePixelListStatistics(list);
        for (j=0 j<destPixelArea.height j++)
        {
            for(i=0 i<destPixelArea.width i++)
            {
                findBestLuminosityMatch(destPixelArea[j][i], list, a, b, stats)
destPixelArea[j][i].alpha = a
destPixelArea[j][i].beta = b
            }
        }
        pixelAr eat oRGB(destPixel xel Area)
gimp drawable_flush(destDraw)
deletePixelList(list)
deletePixelArea(destPixelArea)
}
deletePixelArea(srcPixelArea)
```
D.2 Sampling Methods

The sampling methods are responsible for populating a pixel list which is used for transferring colour to the target greyscale pixel area. The samples are drawn from the source colour pixel area based on sampling parameters specified in the configuration structure. Hence, all sampling methods are a function of the source pixel area.

\[ \text{samplePixelList} \rightarrow f(\text{srcPixelArea}) \]

The sampling methods have the following signature:

\[ \text{PixelList}^* \text{ samplingMethodName} (\text{PixelArea}^* \text{srcPixelArea}, \text{Config}^* \text{conf}, \text{Statistic}^* \text{stat}) \]

Pseudo code algorithms for each of the sampling methods used in the study are given below.

D.2.1 Uniform

```
PixelArea
Width = 10
Height = 6

Config Parameters
sampleInterval = 4

uniformSampling(pixelArea, conf, stats)
{
    stats.samplePixels = pixelArea.width*pixelArea.height/conf.sampleInterval
    pixelList = createPixelList(stats.samplePixels)
    listIndex = 0
    for (j=0 j<pixelArea.height j++)
    {
        for (i=0 i<pixelArea.width i++)
        {
            if ((j*pixelArea.width+i) % conf.sampleInterval == 0)
            {
                pixelList[list_index] = pixelArea[j][i]
                list_index++
            }
        }
    }
    return pixelList
}
```
D.2.2 Uniform Tiled

```plaintext
uniformTiledSampling(pixelArea, conf, stats) {
    neighbourhoodSize = conf.neighbourhoodSize;
    tileWidth = pixelArea.width / conf.nColTiles
    tileHeight = pixelArea.height / conf.nRowTiles
    stats.samplePixels = conf.nColTiles * conf.nRowTiles
    pixelList = createPixelList(stats.samplePixels)
    listIndex = 0;
    for (rowIndex=0; rowIndex<conf.nRowTiles; rowIndex++) {
        for (colIndex=0; colIndex<conf.nColTiles; colIndex++) {
            smplPixely = rowIndex*tileHeight + tileHeight/2
            smplPixelx = colIndex*tileWidth + tileWidth/2
            i = smplPixelx - neighbourhoodSize/2
            j = smplPixely - neighbourhoodSize/2
            neighbourhoodSum = 0.0
            neighbourhoodMean = 0.0
            neighbourhoodSD = 0.0
            n = 0
            for (k = 0; k < neighbourhoodSize*neighbourhoodSize; k++) {
                j = j + (k==0 ? 0 : k/neighbourhoodSize)
                if (k % neighbourhoodSize == 0)
                    i = smplPixelx - neighbourhoodSize/2
                else
                    i = i + k%neighbourhoodSize
                if (isIndiceInPixelArea(j, i, 0, pixelArea.height, 0, pixelArea.width) == 1) {
                    neighbourhood[n] = pixelArea.area[j][i].lum
                    neighbourhoodSum += neighbourhood[n]
                    n++
                }
            }
            neighbourhoodMean = neighbourhoodSum / n
            neighbourhoodSD = getStdDev(neighbourhood, n, neighbourhoodMean)
            pixelList[listIndex] = conf.samplePixelWeight * pixelArea[smplPixely][smplPixelx] +
                conf.neighbourhoodMeanWeight * neighbourhoodMean +
                conf.neighbourhoodSDWeight * neighbourhoodSD
            listIndex++
        }
    }
    return pixelList
}
```
Random Tiled Pixel List

```plaintext
D.2.3 Random Tiled Pixel List

randomTiledSampling(pixelArea, conf, stats)
{
    neighbourhoodSize = conf.neighbourhoodSize;
    tileWidth = pixelArea.width / conf.nColTiles;
    tileHeight = pixelArea.height / conf.nRowTiles;
    stats.samplePixels = conf.nColTiles * conf.nRowTiles * conf.samplesPerTile;
    pixelList = createPixelList(stats.samplePixels);
    listIndex = 0;
    for (rowIndex=0; rowIndex<conf.nRowTiles; rowIndex++)
    {
        for (colIndex=0; colIndex<conf.nColTiles; colIndex++)
        {
            for (smplCounter = 0; smplCounter < conf.samplesPerTile; smplCounter++)
            {
                smplPixely = rowIndex*tileHeight + getRandom()*tileHeight;
                smplPixelx = colIndex*tileWidth + getRandom()*tileWidth;

                j = smplPixely - neighbourhoodSize/2;
                i = smplPixelx - neighbourhoodSize/2;
                neighbourhoodSum = 0.0;
                neighbourhoodMean = 0.0;
                neighbourhoodSD = 0.0;
                n = 0;
                for (k = 0; k < neighbourhoodSize*neighbourhoodSize; k++)
                {
                    j = j + (k==0?0: k/neighbourhoodSize);
                    if (k % neighbourhoodSize==0)
                        i = smplPixelx - neighbourhoodSize/2;
                    else
                        i = i + k%neighbourhoodSize;
                    if (isIndiceInPixelArea(j, i, 0, pixelArea.height, 0, pixelArea.width)==1)
                    {
                        neighbourhood[n] = pixelArea.area[j][i].lum;
                        neighbourhoodSum = neighbourhoodSum + neighbourhood[n];
                        n++;
                    }
                }
                neighbourhoodMean = neighbourhoodSum / n;
                neighbourhoodSD = getStdDev(neighbourhood, n, neighbourhoodMean);
                pixelList[listIndex] = conf.samplePixelWeight * pixelArea[smplPixely][smplPixelx] +
                                      conf.neighbourhoodMeanWeight * neighbourhoodMean +
                                      conf.neighbourhoodSDWeight * neighbourhoodSD;
                listIndex++;
            }
        }
    }
    return pixelList;
}
```