Adaptation of Colour Perception through Dynamic ICC Profile Modification

A thesis presented in partial
fulfilment of the requirements
for the degree of

Doctor of Philosophy
in Computer Science

at Massey University,
Albany (Auckland), New Zealand.

Guy Kristoffer Kloß

2010

Abstract

Digital colour cameras are dramatically falling in price, making them affordable for ubiquitous appliances in many applications. Change in colour perception with changing light conditions induce errors that may escape a user's awareness.

Colour constancy algorithms are based on inferring light properties (usually the white point) to correct colour. Other attempts using more data for colour correction – such as (ICC based) colour management – characterise a capturing device under given conditions through an input device profile. This profile can be applied to correct for deviating colour perception. But this profile is *only* valid for the specific conditions at the time of the characterisation, but fails with changes in light. This research presents a solution to the problem of long time observations with changes in the scene's illumination for common natural (overcast or clear, blue sky) and artificial sources (incandescent or fluorescent lamps).

Colour measurements for colour based reasoning need to be represented in a robustly defined way. One such suitable and well defined description is given by the CIE LAB colour space, a device-independent, visually linearised colour description. Colour transformations using ICC profile are also based on CIE colour descriptions. Therefore, also the corrective colour processing has been based on ICC based colour management. To verify the viability of CIE LAB based corrective colour processing colour constancy algorithms (White Patch Retinex and Grey World Assumption) have been modified to operate on $L^*a^*b^*$ colour tuples. Results were compared visually and numerically (using colour indexing) against those using the same algorithms operating on RGB colour tuples.

We can take advantage of the fact that we are dealing with image *streams* over time, adding another *dimension* usable for analysis. A solution to the problem of slowly changing light conditions in scenes with a static camera perspective is presented. It takes advantage of the small (frame-to-frame) changes in appearance of colour within the scene over time. Reoccurring objects or (background) areas of the scene are tracked to gather data points for an analysis. As a result, a suitable colour space distortion model has been devised through a first order Taylor approximation (affine transformation). By performing a multi-dimensional linear regression analysis on the tracked data points, parameterisations for the affine transformations were derived.

Finally, the device profile is updated by amalgamating the corrections from the model into the ICC profile for a single, comprehensive transformation. Following applications of the ICC based colour profiles are very fast and can be used in real-time with the camera's capturing frame rate (for current normal web cameras and low spec desktop computers). As light conditions usually change on a much slower time scale than the capturing rate of a camera, the computationally expensive profile adaptation generally showed to be usable for many frames.

The goal was to set out and find a solution for consistent colour capturing using digital cameras, which is capable of coping with changing light conditions. Theoretical backgrounds and strategies for such a system have been devised and implemented successfully.

Acknowledgements

My first and biggest thanks go to my lovely wife Friederike. Without her love and support the "Project Ph. D." would *not* have been possible. Especially the personal, mental and financial backing through the *W. I. F. E. scholarship scheme* were essential.

Of course, my parents and family have laid the corner stones by raising me (physically and mentally), and providing me with a good education. And my dad has dragged me into the printing industry. This gave me sound knowledge on what colour management is, how it is done and why it matters.

Ansgar: You are my best friend. You pointed out early, that there are life and opportunities also outside of Germany. During the last years, geographical distances were against seeing more of each other. But you were there for me early, you have "reoriented my corner stones" (to my parents' sadness), and I know you will always be there and be a good influence.

On a more academic side, I am thankful to Ken Hawick, for accepting me as a Ph. D. student without prior formal Computer Science education. And of course Napoleon Reyes, my supervisor, for giving me the "academic playground" at Massey that I have been exercising on, and for being a nag to guide me also through the more tedious essentials and red tape of the Ph. D. course.

I would also like to thank Andreas Schreiber, my former "bosslette" (little boss) and friend, who gave me the freedom to learn and "play" with Python on the job at the German Aerospace Centre (DLR) for productive purposes, that laid out the foundations for much of the programming conducted in this thesis. Andreas, I miss the boxing and collective departmental caffeine etiquette! Also, for getting me involved in the Grid Provenance Project, in which I met Omer Rana. Omer has pointed me towards New Zealand and Ken here at Massey University.

Brian Whitworth was (almost) always a very entertaining and inspiring discussional sparring partner in the corridor. Thanks for giving me reasons, to view my academic work beyond pure Computer Science.

Martí Maria and Graeme Gill, your feedback, information and insights into your respective colour management systems, that I have used extensively, were invaluable to the success of this research.

The education in my "original" academic degree in Chemical Engineering at the University of Dortmund has been hard, but fun. Lots of it ... and it has given me the highly valuable scientific background information that I am building on, which enabled me to think beyond business applications. It laid the foundations for the interest in Scientific Computing.

Finally, I would like to thank Pink Floyd for their fabulous music (and Radio Hauraki for playing them, and providing an online live radio feed). This music has provided some moments of sanity, while labouring on the research, to relieve me for a few minutes out of my working pace and to revitalise and give me new clarity.

Now, really ... finally! Many many thanks to all my friends, family, colleagues, reviewers, etc. that did not get a mention on this page. I want to keep this brief, so I am cutting off the list here. But you have helped to make it worth it!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital colour cameras are dramatically falling in price, making them affordable for ubiquitous appliances in many applications. An attempt to use colour information reveals a significant problem that usually escapes our awareness. Due to the adaptive nature of the human visual system we often do not recognise many changes in illumination characteristics. A camera however will measure scenes under changing illumination differently. Cameras often do provide means to compensate for these expected changes, but they do so from a point of view of user convenience rather than measurement accuracy. Often these compensations are biased by the goal to make the pictures "look pretty" rather than accurate.

Besides the artistic or documentary purpose of photography and filming, with increasing computing capabilities, digital cameras are increasingly used for analytical image capturing. Images are processed to extract information. This information may be based on shapes, bodily (3-D) objects (through multiple cameras or frames), colour, etc. In these cases the analyses of the artistic or aesthetic image content is of lesser interest, but rather precision is wanted.

The following scenario will provide an impression of what an image processing system may need to be able to cope with: A scene is observed over a day by a camera in a room with a window (no artificial light) and an overcast sky (quite "neutral" daylight illumination). The weather clears up to a spotless blue sky, and the light brightens up and changes to a more bluish shade. In the evening sun rays directly fall in through the window onto the scene to increase the light intensity further and "paint the scene" in yellow/orange shades while adding hard shadows. At dusk, as the sun sets, a person turns on fluorescent light in the room with yet another shade and light intensity; additionally the spectrum of the fluorescent light is composed differently.

Ideally the scene's colour composition is detected properly (after digital image processing), regardless of the camera images' brightness, dynamic range, and colour shifts due to the different lights' colour compositions. Our eyes together with the rest of the visual system tend to provide a "filtered" representation by adapting to the conditions quite well, the

raw camera images do not. We perceive a white piece of paper within the scene as white, regardless whether viewed in the direct sun or candle light.

This argumentation also may lead to the bold attempt of an insight, that models of human vision may not merely be derived by measurement only. One theory for demystifying the human vision system is to try to understand it by gaining an understanding of the *computational* problems to be solved to obtain equivalent results.

First Sect. 1.1 will introduce the problem domain for this work. The next Sect. 1.2 outlines the objectives to achieve, followed by a statement on the significance of this research in Sect. 1.3. Sect. 1.4 states the problem and describes the research attempt as it is inspired and differentiated from other related colour based research. In Sect. 1.5 the limitations in scope for the research are stated, and in Sect. 1.6 means of verification of validity and effectiveness are outlined. After a summary in Sect. 1.7, Sect. 1.8 will give an indication on how the chapters of this thesis are structured and how to approach reading it.

1.1 Problem Domain

Colour is an important channel of information in day-to-day life for humans. Colour coding is used in many cases for warning, outlining, marking, etc. A reason for this is, that in the human brain colour is perceived before form, which is perceived before motion [1]. After adaptation to an environment – which takes about one second – colour stimuli are the first a human can react to. The different perception stages can be registered in time quite precisely. Motion in contrast takes about 70–80 ms longer to process than colour. Therefore, also technical systems operating within daily situations are likely to have to identify the same colour coded markers, and therefore need to deal with colour efficiently and correctly.

The focus of this research are such systems. Namely those used for robotic research platforms (e. g. robot soccer), industrial identification (e. g. baggage identification), industrial classification (e. g. quality control). All these scenarios have got a few things in common: They demand online or near real-time video image processing, are used in indoor and/or outdoor environments with usually slowly changing conditions, and are built using relatively cheap commodity colour cameras (web camera type devices).

1.2 Objectives

This research attempts to implement a system to reveal colour information in a scene through images from a camera. The goal is to perceive this colour as precisely as possible. The goals is to characterise and eliminate the influences of the illuminant as well as the capturing device and therefore deduce the colourimetric reflectance of the objects over time in an instationarily illuminated environment. This should be solved by researching the following:

1. Derive output colour descriptors, that are independent of the device used, and are therefore comparable with those of arbitrary other devices. The colour descriptor has to be in a sensible colour space suitable for further processing.
   (Chapters 2, 4 and 6)

2. Develop practical solutions for modelling changes in the current colour perception. This implies a characterisation of the influences of the illuminant as well as the capturing device.
   (Chapters 3, 5, 6 and 9)

3. Develop and fit a model of the change in colour perception, and to be able to parameterise this model quantitatively.
   (Chapter 8)

4. Dynamically adapt to changes in conditions (light, scene, other influences), especially to changes in illumination are focus of the adaptation. Changes between common natural (overcast or clear, blue sky) and artificial sources (incandescent or fluorescent lamps) need to be considered.
   (Chapter 8)

5. Make use of the additional source of information available due to the fact that we are dealing with an image sequence, in which we can track changes in objects or regions.
   (Chapter 8)

1.3 Significance

The hybrid and integrating nature of this research makes it quite unique in its way. Currently, no openly available research that combines the use of device-independent colour description with an adaptation to changing light conditions over time could be identified. This thesis integrates means of "classic" colour constancy research with colour science aspects (on human vision, perceptional linearity, chromatic adaptation) and current best practises in colour management. Additionally, it tries to adapt to occurring changes in conditions while processing the image streams in (near) real-time.

The benefit of a dynamic chromatic adaptation promises to be valuable in many fields of colour based decision making. Colour measurements become more comparable. Therefore, all dependent processes should gain robustness. We are expecting improvements in our robot soccer implementation, eliminating tedious and error prone manual parameter tweaking by gaining a self-correcting system. Additionally, we will be able to improve the constancy for longer running robot soccer games, as well as enable robots to move between differently lit environments (in- or outdoors, robot rescue, etc.). Industrial partners will benefit from improved Colour Indexing quality for object identification, and others are aiming at better stability for colour based object sorting systems.

1.4 Statement of Problem and Research Attempt

Not in every general case, objects with *a priori* known colours are usable, because the same objects cannot be expected to often reoccur in a scene. Additionally, colour constancy algorithms requiring non-negligible computational time to process each image, may make them unsuitable for live image processing. Other attempts for colour correction – such as (ICC based) colour management – characterise a capturing device under given specific conditions through an input device profile. This profile can be applied to correct for deviating colour perception. But this stored profile is *only* valid for the given situation at the time of the characterisation, but fails when light conditions change.

We can take advantage of the fact that we are dealing with image *streams* over time, which adds another *dimension* usable for analysis. According to our delimitation of scope, scenes we are dealing with are generally changing slowly, and colour adaptations will only need to change slowly as well. Therefore, we are basing the research in this thesis on the following expectations:

- Areas of the scene – such as the background – are generally more stable than a commonly faster changing foreground.
- In live image capturing, changes between individual frames should usually quite small, thus a fairly robust statistical scene model can be derived.
- By observing changes in the scene, it could be possible to derive colour corrections.
- Also due to the slow changes, this correction should commonly have a certain validity for the duration of several to many frames captured by the camera.
- As a result it could be possible to update a corrective colour transformation (the device profile), which then can be applied to the captured colour image at the current time.

Even though this system is intended to be used on live camera image capturing, the computational overhead is not expected to be the limiting overall factor. As stated, illumination changes tend to happen slowly, and cases of quick changes are detectable. Therefore, the adaptive calibration system may run in a separate thread or process (alongside the live chromatic adaptation) to update the colour correction transformation at intervals. In case of strong sudden changes, the system will suffer a brief period to adapt to the changed situation. But this is an effect that also happens to humans in day to day situations. If the overall system gains robustness, this situation of momentary "blindness" can be tolerated.

For further analysis of the processed image stream – as well as for a robust, fast and credible way of adapting the colour of the incoming capturing device – it is desirable to make use of a standardised, linear and device-independent colour space, like CIE LAB. Transformation to it is usually performed using a colour management module with the help of the above mentioned device colour profiles. The implementation of the system should make use of this process.

1.5 Scope

Implementations of the algorithms are geared towards suitability for the use cases we are dealing with (e. g. robotics, object identification). The system has to be able to handle a specified video image stream in (near) real-time, although a certain delay due to processing can be tolerated. Light sources considered should be limited to commonly occurring natural and artificial sources without extreme colour tints. This includes limited indoor and outdoor capability, while encountering commonly slow illumination and/or the scene changes.

The adaptation of the colour correction transformation is based on tracking colour changes over time. Doing so, it takes advantage of the fact that certain changes generally occurring at a slow rate (e. g. changes in light conditions and in the background). This assumption can in certain circumstances not be met. Three cases can be distinguished: (A) sudden, strong illumination changes (e. g. someone operates a light switch); (B) sudden strong change of scene (e. g. the camera is turned or moved quickly); and (C) a combination of the two previous cases. For case (A) – due to the fact of commonly small changes in scene between frames – the error in background/foreground segmentation will be quite low if a scene segmentation from a previous frame will be re-used. In case (B) the illumination probably only changed minimally, therefore, the adaptation can be preserved, and a new scene segmentation model needs to be derived. Case (C) causes more of a problem, as no advantage of a slowly changing environment can be used to derive a solution. This "bootstrapping" problem could be solved by pre-pending a colour constancy algorithm [1, 2] to determine an estimation of a scene's illumination.

This research only deals with the cases upholding the mentioned assumption of slow changes. The above stated "bail out strategies" are mentioned to provide strategies to cope with the cases invalidating this assumption.

1.6 Verification

We are intending to use the CIE LAB colour space (for its before mentioned virtues) for processing as well as the application of colour correction. For testing the suitability of corrective colour operations on CIE LAB colours, we are attempting to improve the most well known candidates for colour constancy (White Patch Retinex and Grey World Assumption algorithms) to work with that colour space. The correction results should be compared to those obtained with the device-dependent rg-chromaticity. For a verification of the quality of the colour correction, the colour image sets under various illuminations by Barnard [2] should be used.

Colour Indexing provides a method for recognising objects based on their distribution of colours [3, 4]. Barnard already verified the improvements of his chromatic adaptation efforts using this method [2]. This verification has gained some traction for benchmarking colour

constancy algorithms, so we will also use Colour Indexing for further verification. By this, it becomes possible to judge the quality of algorithms statistically and eliminate human bias.

For further verification beyond colour constancy we shall try to use a canonical reference within the scene, that is to be determined for the initial known conditions for future comparison.

1.7 Summary

In this thesis we are trying to establish a sound overview of the physical and perceptional basics of colour capturing and perception. It distinguishes between the illuminant's influence, the surface reflection and the camera's perception properties. Various methods to quantise colour through tristimulus description (using three colour descriptors/channels) are discussed (colour spaces). In order to obtain sensible results in a quantitative analysis from tristimulus based colour spaces, various requirements towards the describing colour space used need to be considered. The colour space needs to be independent of the specific capturing device (for comparability), and it needs to be linearised across the perceptible colour gamut. We will outline a way to achieve this by transforming the captured device-dependent colour space (e. g. RGB) into the device-independent CIE LAB colour space by means of the application of colour profiles.

For colour handling in digital environments, the focus has been set on standardised and industry approved colour spaces and methods. On this basis, possibilities to obtain robust colour information from live image capturing will be discussed – either by modifying existing algorithms, or deriving new correction algorithms, that can be adapted to illumination changes, as they appear in real time during a live observation.

We will outline some promising approaches for possible solutions based on tracking the changes in appearance of colour samples. The proposed research has been limited to the common case of scenes with slow changes of distinct properties, but possible solutions for cases violating these constraints will be discussed.

It is important to understand that this research will not be able to establish a precise description of the scene's chromaticity. But it tries to compensate for dynamic changes that are challenging to account for. It shall try to provide an estimation of the true chromaticity which is not directly obtainable through direct measurement using common cameras, as cameras are always subject to device specific capturing characteristics [5].

1.8 Hitch Hiker's Guide to this Thesis

Although one may desire to read this thesis from cover to cover, this is not necessary. The chapters have mostly been written to be self contained, and refer to other chapters/sections in a way citations do.

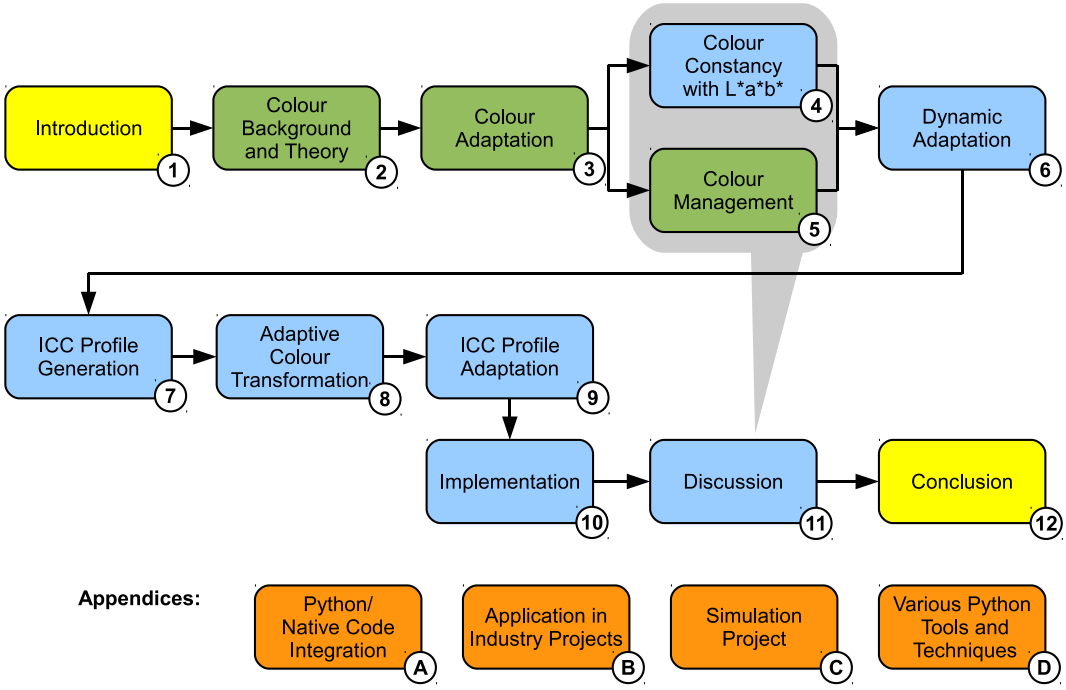

Figure 1.1: The Hitch Hiker's Guide to this Thesis.

> *"The introduction* [to the Guide] *begins like this:*
>
> *'Space,' it says, 'is big. Really big. You just won't believe how vastly hugely mindboggingly big it is. I mean you may think it's a long way down the road to the chemist, but that's just peanuts to space. Listen . . . ' and so on.*
>
> *(After a while the style settles down a bit and it begins to tell you things you really need to know, like . . . "*
>
> – Douglas Adams, "The Hitch Hiker's Guide to the Galaxy"

The "plot" of this thesis is not fully linear. As a guide outlining the structure to the different chapters Fig. 1.1 may be used.

All chapters highlighted in green introduce theoretical backgrounds, whereas the ones highlighted in blue contain our own contributions. Chapter numbers are given in the white circles at the bottom right of each chapter's box. As indicated in the figure, two different approaches for colour adaptation – "Colour Constancy with $L^*a^*b^*$" and "Colour Management" – are described as alternatives independently. Where the latter just introduces the theoretical background of colour management, the former already provides information on how we have extended it to work on the $L^*a^*b^*$ colour space. Ideas from these two branches are picked up in Chap. 6 to outline the hybrid approach that is taken in this thesis. The discussion in Chap. 11 then draws together the results from the past chapters while putting the results particularly in relationship to the information provided in Chap. 4 and 5.

Appendices are highlighted in orange. These contain supportive information on implementation issues encountered in the work for the thesis as well as some (partly unrelated) side line projects undertaken during the studies for this thesis. All colours in the figure not mentioned, yet, have been chosen to please the author.

Chapter 2

Colour Background and Theory

An image's colour perception depends on three basic factors: The physical content of the scene, the scene's illumination and the capturing characteristics of the camera. Attempts to deduce object colour from the images will need to cope with the influence of the illumination and the camera's characteristics. Furthermore, a large variety of colour spaces are available to describe colour. Differences between them and their fitness to quantify colour are discussed. Also the respective suitability for colourimetric computations are covered.

This chapter tries to establish a basic understanding of the intricacies behind the processes involved in capturing images and recognising colour – from light as a stimulus to the colour sensed values in cameras. The myriad of factors affecting colour perception by machine vision systems and the different technologies that evolved to address those issues are discussed.

The research ambitions are driven by the desire to operate on colour corrected imagery from digital capturing devices. First, in Sect. 2.1, we will introduce the intricacies of digital colour image capturing. Then, in Sect. 2.2, the basics for colour theory are established as they are needed for this thesis. Finally, in Sect. 2.3, these two are brought together into a description of colour encoding for the digital world.

2.1 Cameras as Measuring Devices

> *"What we observe is not nature itself,*
> *but nature exposed to our method of questioning."*
> – Werner Heisenberg, *Physics and Philosophy* (1958)

To understand problems in quantifiable colour capturing, we first have to achieve a better understanding of the image capturing process within digital cameras [5,6]. This process can be dissected into three distinct stages: The physical capturing of the image on the sensor, the rendering of the scene, and finally the encoding of the scene representation.

Capturing Digital colour cameras nowadays are equipped with a light sensitive CMOS or a CCD sensor. The exact function of these sensors is not so important for the understanding of this analysis. The sensor contains a matrix of single sensors that are (e. g. through filters) responsive to specific ranges of radiation in the visible light spectrum. During exposure each sensor collects an amount of discrete light quanta (photons). The longer the exposure, and the more intense the radiation is, the more light quanta are registered in each individual sensor. This leads to a limiting effect on the dynamic sensitivity range: Ideally a minimum of one photon can be detected on the lower boundary. For the upper boundary each sensor exposes a "maximum capacity." Beyond these extremes the sensors cannot distinguish between intensities anymore.

Rendering The firmware of the camera then reads the raw image captured from the sensor into software. Here the sensor response is rendered into the destination image. This stage compensates for improper illumination, applies white balancing [6], etc. Manufacturers intend to achieve with this rendering step a "visually pleasing representation" of the scene for the customer, partly by mocking human dynamic adaptability. Therefore, in most cases readings *will not* directly be passed through, but altered. Only some more expensive cameras offer the option of taking a pure raw picture, that can be read from the camera.

Rendering in the cameras is handled in a highly proprietary way, which is not transparent to the user. It is done in a way the camera manufacturer feels it will produce the most "pleasing image appearance" for the customers. The algorithms involved are trade secrets, and thus not cleanly reversible. Still, some process steps are known in principle [6]. These include individual white balancing of the image (or sequence of images) to compensate for the present illumination; scene re-lighting for a maximum dynamic image range and balance; boosted contrast and colour saturation (especially in the mid-tones) for vibrant colours; highlight compressions to compensate for glaring effects of specular reflection; and various other – potentially spatially inhomogeneous – scene rendering. These scene rendering algorithms are vendor, model and even version specific, thus they cannot be estimated with confidence.

Even film stocks in traditional photography are known to have different characteristics, which are selected by the photographer based on their preference. Standardised image formats and colour space encoding (see Sect. 2.3.1) may fool us into the impression that the representation of a scene from two different camera brands should match, but this is rarely the case.

Encoding Finally the rendered image representation is encoded into the standard compliant image format, usually in a standard compliant colour space as sRGB. Unfortunately, this standard compliant colour space encoding suggests a colour compliant representation of the imagery according to the defined standard. Due to the rendering process this is not true regarding the observed scene, but only towards the rendered representation of it.

2.2 Colour Theory

We need to establish an understanding of the physical background of colour creation in scenes, and how the human vision systems respond to it. Here we are focusing on the effects creating the colour of the scene, not the creation of the colours lighting the scene.

The origins of colour lay in the distribution of radiation intensities within the spectrum of visible light, the *visible spectrum*. The human eye can perceive light in the frequency range of 450–750 THz (10^{12} Hz). In air – or more precisely in vacuum[1] – this translates to wavelengths ranging from 400 to 700 nm, although some people may be able to perceive wavelengths from 380 to 780 nm. Isaac Newton first described colour appearance with respect to the visible spectrum [7] extensively, and thus can be considered the father of modern colour theories. He recognised that the colours comprising white light are "refracted" by different amounts, and he also understood that there is no "coloured" light, the colour being in the eye of the beholder. There is merely a range of energies associated with specific wavelengths.

Johann Wolfgang von Goethe strongly disagreed with Newton in the *Polemic Section* of his "Theory of Colours" [8] on the topic of spectral composition of colours, especially on the aspects that a proper mixture of all colours would compose the colour white. While Goethe's theory has never been accepted by physicists, his insight on the perception of colour have been influential. This human colour perception, together with Newton's insight that actual colours only exist in the eye of the beholder, will be further discussed with respect to Colour Spaces in Sect. 2.3.1.

The following section will focus first on the more technical aspects of image formation and modelling, whereas the next section after that will introduce human perception of these colours.

2.2.1 Image Formation and Capturing

The most general case encountered in scenes is the reflectance of light on the surface of bodies. Fig. 2.1 illustrates the simplest form of image formation due to reflection. The light emitted from a source (whether natural, artificial or due to reflectance from other objects) is incident on a surface. The light (or a fraction of it) is reflected towards the eye or an observant detector (e. g. camera). A lens focuses the incoming light onto a light sensitive surface.

In the human eye this surface is the retina containing light sensitive receptors. These receptors are the so called rod cells and cone cells. Rod cells are very light sensitive and used for night vision, they only detect lightness. Cone cells contain a colour pigment on their tip making them responsible for colour vision. These cells can be distinguished into three types, the *short (S), medium (M)* and *long* wavelength *(L)* sensitive cells.

[1] Due to a different refraction index this varies in other media, or even at different pressures or temperatures.

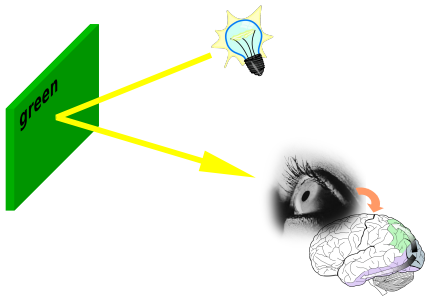

Figure 2.1: A simple model of image formation: light from a source is reflected from a (coloured) object's surface. A fraction of the (coloured) light is reflected to the eye or camera.

Similarly, digital (colour) cameras possess a light sensitive detector array – typically on a CCD or CMOS chip (see Sect. 2.1). These consist of differently colour sensitive detectors, usually for the three basic colours *red (R), green (G)* and *blue (B).*

The detectors in the sensitive surfaces are responsible for colour sensing or capturing capabilities (see Sect. 2.1). Rendering of the captured scene within the camera bears another analogy with human vision: The brain also post-processes the sensed image towards the perceived image.

Colour Image Formation

Our sensation of colour is a result of the processing of light energy first by the eye and later by the brain. This process – especially regarding the complex perception process in the brain – is not fully understood. But for the problem we are going to analyse a simplified – but sufficiently accurate – model is available [1, 9].

Light can be viewed essentially as a form of electromagnetic energy. A light source can be characterised by how much energy (or power) it emits at different wavelengths. This is called the source's emission spectrum. In Fig. 2.2 the intensity distribution of a typical day light is a continuous function of the wavelength within the visible range (solid line), this is the light's spectral power distribution (SPD). As one may expect looking at this figure, the SPD continues beyond the range of 400–700 nm both in the infra-red (above 700 nm) and the ultra-violet (below 400 nm) areas. The human eye generally does not possess the sensitivity to detect these.

Similarly to the incident light, the reflective surface of an object can be characterised by a reflectance function as shown in Fig. 2.3. This function determines the fractional reflectance of light relative to the various wavelengths. Again, only the visible range needs to be considered.

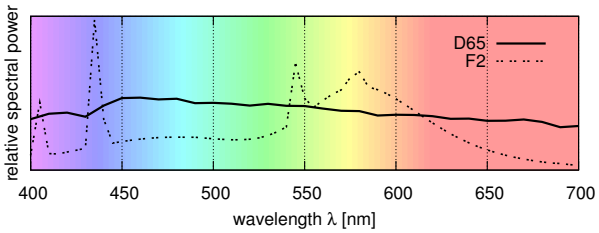

Figure 2.2: Spectral power distribution of CIE D_{65} daylight (6504 K) and F2 cool white fluorescent illuminant (4200 K), as specified by the CIE.

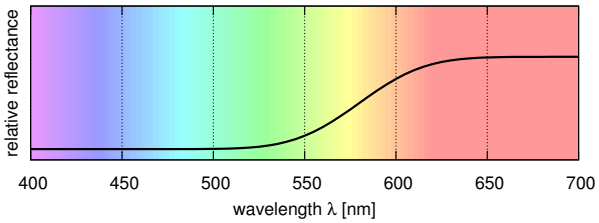

Figure 2.3: A bright red surface's spectral reflectance.

Taking the information from the power distribution function $E(\lambda)$ in Fig. 2.2 and the relative surface reflectance $S(\lambda)$ from Fig. 2.3 we can derive a colour signal function $C(\lambda)$:

$$C(\lambda) = E(\lambda)S(\lambda) \tag{2.1}$$

The colour signal function defines the SPD of light reflected from the object's surface. It is then focused to the sensitive plane of an eye (the retina) or a camera (film or sensor matrix). In Fig. 2.4 the relative sensitivities of the three cone types of the human eye for short, medium and long wavelengths are shown[2]. Note that these colours do not correspond to RGB colours (red, green, blue). In fact, the L and M cell's sensitivities are rather closely located together, with the M cell sensing in the bluish green, and the L cell in the yellowish green areas[3] (see right hand side of Fig. 2.4). Due to this fact, the eye itself *does not* directly sense any red colours. In contrast to the human cell response, digital cameras are constructed to exhibit distinct RGB responses suitable for technical purposes of image representation.

The total response of the eye's cone cells can therefore be stated through integrals over the wavelength range with a non-zero response ω. (2.2) yields the sensed triplet (s, m, l)

[2] $S(\lambda)$ in this case is *not* the relative surface reflectance function mentioned previously.

[3] The close location of the sensitivities of the L and M cells together with genetic variations are the reason for the common red-green colour blindness.

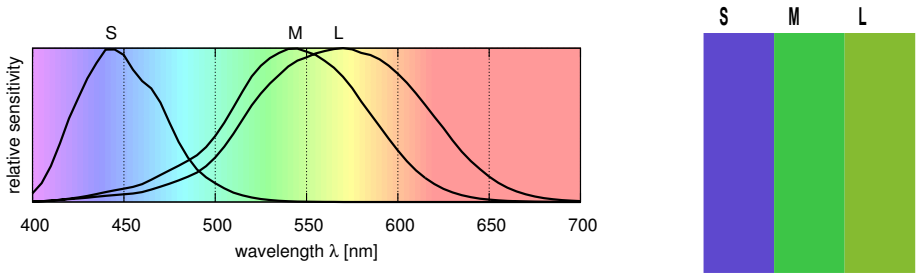

Figure 2.4: (left) Relative sensitivities of the cone cells within the human eye for the S, M and L cells ($S(\lambda)$, $M(\lambda)$, $L(\lambda)$). (right) Approximate representation of the colour the cone cell's are most responsive to.

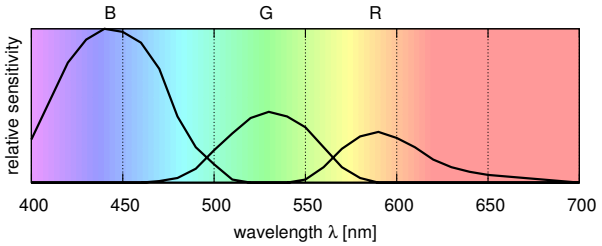

Figure 2.5: Relative sensitivities of a typical digital camera's R, G and B sensors ($R(\lambda)$, $G(\lambda)$, $B(\lambda)$).

that is subsequently perceived by the eye.

$$\begin{aligned} l &= \int_{\omega} C(\lambda)L(\lambda)d\lambda \\ m &= \int_{\omega} C(\lambda)M(\lambda)d\lambda \\ s &= \int_{\omega} C(\lambda)S(\lambda)d\lambda \end{aligned} \tag{2.2}$$

In technical environments the same colour signal is sensed within a camera, only using the camera's response functions for $R(\lambda)$, $G(\lambda)$, and $B(\lambda)$:

$$\begin{aligned} R &= \int_{\omega} C(\lambda)R(\lambda)d\lambda \\ G &= \int_{\omega} C(\lambda)G(\lambda)d\lambda \\ B &= \int_{\omega} C(\lambda)B(\lambda)d\lambda \end{aligned} \tag{2.3}$$

The sensory response (s, m, l) or (R, G, B) – called tristimulus – is usually processed, and not used directly. This processing either takes place within the brain leading to an imagery

perception of the scene, or through the scene rendering algorithms in the camera. This also means that the full spectral information is lost. Further processing the tristimulus values depends on implying some assumption of boundary conditions, usually of the illumination and colourimetric stability to changing illumination (where metamerism is not a significant concern, see Sect. 2.2.1).

Illumination

As discussed in Sect. 2.2.1, the illuminant's spectral power distribution (SPD) is the base information, from which a scene's colour spectrum is derived. Through the reflection and sensing process this illuminant's SPD is subtractively reduced. Light sources can be split into two distinct basic groups: Thermal radiation sources, and emission sources through electromagnetic radiation.

Thermal radiation sources are characterised by a smooth SPD over the wavelength range. These light sources are also called *black body* radiators or Planckian radiators. The theory behind states that each body – that does not reflect any foreign light (thus "black body") – emits a characteristic light spectrum that is *only* dependent on its temperature. According to Planck's law the spectral radiance of electromagnetic radiation from a black body is described at all wavelengths at a given temperature. With increasing temperature the curve's maximum moves up and from longer wavelengths (reddish light) towards shorter wavelength (bluer). Therefore, the characteristics of many common light sources can be approximated by associating it with a black body radiator's colour temperature [10]. Daylight for example is most commonly associated with either 5000 K (US standard) or 6500 K (European standard).

Emission sources in contrast emit photons during discrete events, when electrons in an excited media fall from one orbital back to another orbital. Every state of excitement is characteristic for a specific photon emission with a specific wavelength. Therefore, an emission spectrum is not continuous, but rather consists of distinct lines, or impulses.

Usually the SPDs of light sources are a mixture of these two extremes. The fluorescent light in Fig. 2.2 gives a good example of such hybrid light sources. Pure monochromatic light is also non-existent in nature, as currently the only light source for it are lasers. Additionally, light source's SPDs are often "filtered" by passing through various gasses or media having inverse effects of an emission source: Certain lines are "removed" as those spectral energies have been absorbed by exciting electrons' orbital state while passing through the media. Daylight for example originates mainly from the sun. The sun itself is a hot body emitting a black body spectrum. But additionally a wide variety of excitable gasses are to be found both around the sun as well as in our own atmosphere. Therefore, additional fractions will be added and others will be reduced due to filtering. Other diffuse secondary light sources are usually present, contributing minor "distortions" in the daylight's SPD as shown in Fig. 2.2.

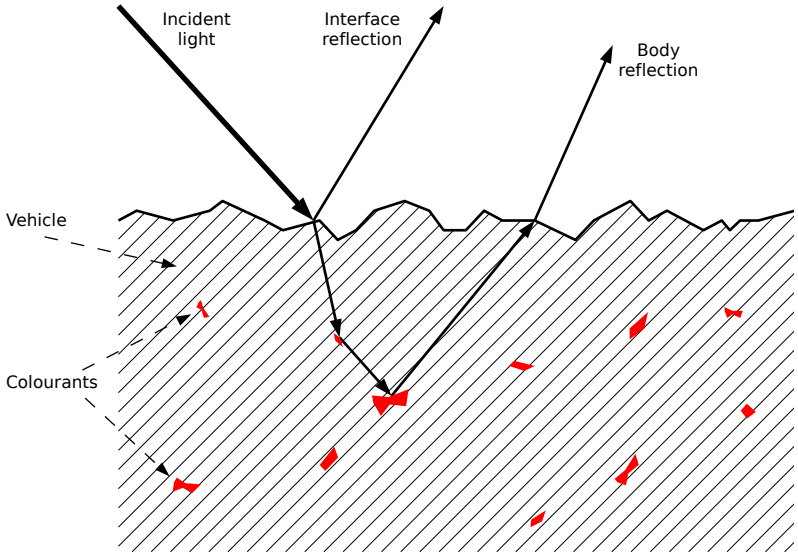

Figure 2.6: The dichromatic reflection model: Light is reflected either by *surface* reflection or *body* reflection.

Reflection Modelling

After reflection of light off the surfaces of objects, it appears impossible to infer properties of the illuminant from an observed scene. Generally this is true, but looking deeper into the physics of the reflection process with respect to the light's SPD gives us hints on potential solutions of this problem.

For this purpose we employ the dichromatic reflection model which was initially proposed by Shafer [11]. Fig. 2.6 shows the general geometry of the model to be used. Incident light $C(\lambda)$ is split into two fractions where it meets the surface of the reflecting body: One fraction is reflected off the *interface* of the surface, whereas the other is diffracted into the body. Some materials have got only interface reflection, like metals. These often expose wavelength-dependent reflectance behaviour, giving some metals (e. g. copper, gold) their characteristic colour. On materials exposing a body reflection, the light's SPD enters the substrate and is reflected back. The SPD changes while "travelling" through the *vehicle* and reflection on embedded colourants before leaving the body again. In (2.4) i and b indicate the interface and body reflection terms composing the overall surface reflection. c is an attenuation factor depending on geometry.

$$S(\lambda) = c_i S_i(\lambda) + c_b S_b(\lambda) \tag{2.4}$$

Given a generic sensory response ρ^k and a generic response function $R^k(\lambda)$ for the sensor of the k^{th} channel we can represent (2.2) or (2.3) as (2.5). Together with (2.1) we can expand it further to (2.6) and with (2.4) to (2.8). Resolving the integral and converting to

vector notation finally yields a response vector for the interface reflection $\boldsymbol{\rho}_i$ and the body reflection $\boldsymbol{\rho}_b$ in (2.10):

$$\rho^k = \int_\omega C(\lambda)R^k(\lambda)d\lambda \tag{2.5}$$

$$= \int_\omega E(\lambda)S(\lambda)R^k(\lambda)d\lambda \tag{2.6}$$

$$= \int_\omega \left[c_i S_i(\lambda) + c_b S_b(\lambda)\right] E(\lambda)R^k(\lambda)d\lambda \tag{2.7}$$

$$= \int_\omega \left[c_i S_i(\lambda)E(\lambda)R^k(\lambda) + c_b S_b(\lambda)E(\lambda)R^k(\lambda)\right] d\lambda \tag{2.8}$$

$$= c_i\rho_i^k + c_b\rho_b^k \tag{2.9}$$

$$\boldsymbol{\rho} = c_i\boldsymbol{\rho}_i + c_b\boldsymbol{\rho}_b \tag{2.10}$$

If a surface obeys the dichromatic model, $\boldsymbol{\rho}$ will fall into a plane through the origin defined by a linear combination of the two vectors for the interface and body components. And in the case of dielectrics (as most non-metals are), the interface reflection is not wavelength-dependent ($S_i(\lambda)$ is a constant), thus the colour due to interface reflection is the same as the illuminant ε. In the presence of two or more such surfaces with different body reflections, then the responses $\boldsymbol{\rho}$ will fall into different planes that will intersect in the illuminant direction ε [12]. A number of colour constancy algorithms have been based on this method.

The key to drawing conclusions on the illumination properties is to analyse the specularities, or more precisely analyse the sensor responses of the highlight reflection areas. In these areas the interface reflection becomes larger over the body reflection, and is therefore suitable for the analysis of the illuminant's properties. The difficulty of this process lies in segmenting the scene to uniform surfaces and finding these specularities, and determining whether they are suitable. As mentioned, the surface material needs to be dielectric, but other restrictions apply as well. Surfaces that are for example very rough, self luminous or made of fluorescent materials, are polarising, or reflect anisotropically and will render the dichromatic model useless.

Metamerism

> *"A colour has many faces, and one colour can be made to appear as two different colours."*
> – Josef Albers, *Interaction of Colour,* (1963)

When an object's reflected SPD $C(\lambda)$ is perceived by forming the colour's tristimulus (see Sect. 2.2.1) information is lost. Therefore, it is possible to create different colour samples with the same appearance (tristimulus response) under one illumination, but which will

create different responses under another illuminant. This effect is called *metamerism,* and describes the situation where two colour samples with different spectral power distributions appear to be the same colour when viewed side by side. It is the scientific description of the typical "department store effect," that lets objects (garments) look different under the fluorescent light in the department store than in natural light out on the street. Comparing natural light with fluorescent light as displayed in Fig. 2.2 shows that together with the more or less narrow banded receptors in the eyes or digital capturing equipment, colour perception can quite significantly shift for certain colours.

As the sensor responses for each human and for each camera are (potentially) different, it is impossible to state *one* tristimulus colour description that will be unique independently of the viewer. Metamerism shows that it can be particularly difficult to represent specific colours robustly using n-tuples (for human vision and digital cameras typically with $n = 3$). Theoretically an infinite number of different SPDs can match one colour represented as an n-tuple, and depending on the sensor response functions $R^k(\lambda)$ an infinite number of distinct n-tuple representations of one SPD can be found. Therefore colour measurements or perceptions are *always* sensor or device-dependent. This is especially true regarding the different characteristics of the response curves of camera vs. those of the human eye. Cameras do not satisfy the "Luther condition" (i. e. they do not produce the spectral responsitivity of the human eye's photo receptors) [13, 14].

If two objects with different spectral reflectivities have the same colour appearance (tristimulus values) under one light source, they are said to be metamerically matched. If they exhibit a marked difference under another light source (as with two pieces of cloth that look identical in a shop but different outdoors), they suffer from viewing illuminant sensitivity, frequently called metamerism. (This definition is misleading, but it has become quite common.) This phenomenon is most visible with neutral colours (greys).

2.2.2 Human Colour Vision

What is colour? Colour is a perception that depends on the response of the human visual system to light and the interaction of light with objects. We have described the processes of light sources and the interaction with objects in the sections above.

The retina – the eye's "sensing centre" – consists of two groups of sensors, the rods and the cones. In each eye about 100 million rods are responsible for luminance only in dimly lit situations. About 6 million cones provide the eye with the means to see colour in brighter situations. These cells are interconnected within the retina, and only about 1 million nerve fibres carry the visual information to the brain.

The colour impression is created by providing three different types of cone cells with differing spectral sensitivities (see S, M, and L cells in Fig. 2.5). As the SPD is condensed into a tuple (L, M, S), all spectral information is lost. The pigments in the cone cells responsible for different colour sensitivities may vary among individuals strongly. Especially

the L response (long wave, responds to yellowish green) is known for significant changes due to mutation. Therefore, we cannot expect the sensed tristimuli to be constant among people. Due to genetic mutation in some people the L and M response are located closer together, that under some conditions certain colours are difficult to distinguish from each other (e. g. the common red-green colour blindness). On the other hand mutations are known, that modifies one of the cones towards possessing a fourth pigment. It is not fully known whether this enables those to view colours in "four dimensions," or whether two of the cone responses are merged and lead to a wider ("blurred") colour response for certain colours.

> *"Wizards, even failed wizards, have in addition to rods and cones in their eyeballs the tiny octagons that enable them to see into the far octarine, the basic colour of which all other colours are merely pale shadows impinging on normal four-dimensional space. It is said to be a sort of fluorescent greenish-yellow purple."*
> – Terry Pratchett, *The Colour of Magic,* (1983)

Furthermore, it needs to be well understood, that the eye *does not* segment the perceived colour into a red, green, and blue component. The additive RGB primaries have been chosen at a time for technical reasons, as their distribution across the visible spectrum produces a wide-gamut colour representation, not because they match the eye's response in a particularly accurate way. In fact, this LMS representation is processed by the brain into an HSV like opponent colour space representation. The brain then discriminates colours on the one hand by their lightness, and on the other hand by their chromaticity (hue and saturation).

The spectral information is focused onto the retina by the optical system of the eye. Through optical refraction it is physically impossible to focus different wavelengths with only one lens system simultaneously. Therefore the eye is "optimised" to focus on the large overlap area of the L and M cones' (orange and green) sensitivities, and the optical system is not colour corrected. Fortunately, the human brain does extensive "post-processing" on the sensed information acquired from the visual nerves to compensate for this. Similarly it also compensates for the blind spot (where the optical nerves enter the retina). Without the brain, our vision system would therefore be vastly incomplete with respect to the visual impressions as we know them from day to day life.

2.2.3 Quantifying (Human) Colour Vision

In 1931 the CIE (Commission Internationale de L'Éclairage) standardised a quantified representation of human colour vision. It uses a set of imaginary *red, blue* and *green* primaries, that – when combined – cover the full gamut of human colour vision. Therefore, any combination of the three can match any monochromatic light source perceptible by the human eye.

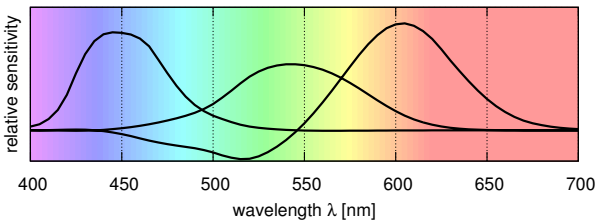

Figure 2.7: RGB colour matching functions with the primaries $\overline{r}$, $\overline{g}$ and $\overline{b}$ (right to left).

An experimental scheme was developed to match every monochromatic (spectral) light source with a combination of these three mentioned coloured light sources [15]. These primaries were chosen to be technically easy to realise by narrow band light sources. The experimental setup had already been invented by Hermann Graßmann about 1853 [16]. It resulted in the determination of three normalised weight factors, the CIE colour matching functions $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ and $\overline{b}(\lambda)$ (see Fig. 2.7). The standard CIE RGB space contains negative values. This may seem strange, but it was discovered that for certain pure monochromatic spectral colours it was impossible to find a match entirely using the three selected primaries. To match these, it was necessary to move one light source in the experiment (the red light) over to the side of the monochromatic light, resulting in a negative sign.

As described in Sect. 2.2.1 the visual impression of a colour is based on three factors: The properties of the light, surface reflectance and sensor. This experiment determined the perception properties of an "averaged" human observer as a baseline for comparative measurements; therefore it standardises the third mentioned factor.

To deduce the reflectance properties of a surface (its colour) quantitatively, also the light needs to be standardised. The CIE illuminant D_{50} (daylight at 5003 K "colour temperature") and the similar D_{65} illuminant (see spectrum in Fig. 2.2) were defined. D_{50} was declared as the default for the CIE colour spaces.

RGB colours for a spectrum $P(\lambda)$ are calculated by integrating over the visible range ω (with some scaling factor k):

$$\begin{aligned} R &= k \int_{\omega} C(\lambda)\overline{r}(\lambda)d\lambda \\ G &= k \int_{\omega} C(\lambda)\overline{g}(\lambda)d\lambda \\ B &= k \int_{\omega} C(\lambda)\overline{b}(\lambda)d\lambda \end{aligned} \tag{2.11}$$

To avoid negative values, the CIE introduced a coordinate transformation. Practically this was achieved through a linear transformation with three non-orthogonal base vectors into the new XYZ coordinate system. The base vectors were chosen so that the human visible gamut entirely fits into positive values of X, Y and Z.

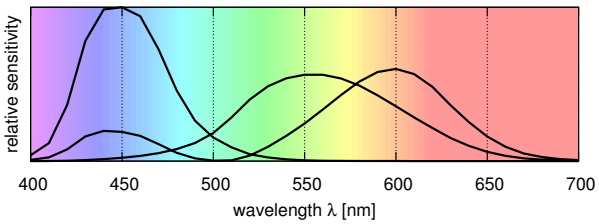

Figure 2.8: XYZ colour matching functions for the Standard Colourimetric Observer with the primaries $\overline{x}$, $\overline{y}$ and $\overline{z}$ (right to left).

Through a transformation matrix M the sensor response $\boldsymbol{\rho}_{XYZ}$ can be derived from a sensor response in CIE RGB space (2.12).

$$\boldsymbol{\rho}_{XYZ} = M\boldsymbol{\rho}_{RGB} \tag{2.12}$$

As an infinite number of such conversions exists, the X and Z base vectors were chosen in a way that they do not contribute to the luminance, thus the luminance is conveniently defined by Y only. (2.12) applied to the CIE RGB colour matching functions yields the corresponding XYZ colour matching functions for the CIE Standard Colourimetric Observer (see Fig. 2.8). As with (2.11) we can obtain XYZ values from the incoming SPD as well:

$$\begin{aligned} X &= k \int_{\omega} C(\lambda)\overline{x}(\lambda)d\lambda \\ Y &= k \int_{\omega} C(\lambda)\overline{y}(\lambda)d\lambda \\ Z &= k \int_{\omega} C(\lambda)\overline{z}(\lambda)d\lambda \end{aligned} \tag{2.13}$$

The humans' capability to see (in colour) is the fundamental reason for us to be interested in quantifying colour. Therefore, human perception has been the base for establishing the means to quantify colour. CIE XYZ was the first commonly accepted attempt, based on the common denominator of human vision in contrast to technical oriented solutions (as RGB).

2.3 Colours in the Digital World

The two common colour spaces for technical purposes (RGB) and visually oriented purposes (XYZ) are available to quantify the human vision system sensibly (Sect. 2.2.3). Neither one of these, however, is without its flaws: They are both neither perceptually uniform nor linear.

Additionally, to simply introduce more visual linearity in the RGB components when used on common cathode ray tube (CRT) displays, the simple concept of *gamma correction*

was introduced (the luminance of the display is not linear with the signal voltage applied to the tube). This exponentially distorts the linearity of the colour channels yielding an R'G'B' colour space. Although, this is highly ambiguous in commonly used terms today, as mostly no distinction between the two is made. It is usually unclear whether the channels in "RGB" are linear or gamma corrected. Furthermore, different systems are known to use different values for gamma to commonly lie between 1.0 and 4.0 for various display types.

Perceptional nonlinearity means, that colours of (just) noticeable differences are much further apart in some regions than in others. In colour related digital image processing, a stable determination of quantitative colour differences is very important.

In this section we first introduce sensible colour spaces for quantitative colour analyses in digital imagery. Then we will build a distinction between device-dependent and independent colour spaces, which will then be expressed in a perceptually linearised way. On the basis of these concepts we will introduce colour management. Finally, the method to adapt device-independent colour representations to different illuminants is outlined.

2.3.1 Colour Spaces

A *colour model* is an abstract mathematical model describing the way colours can be represented as tuples of numbers, typically through three or four values or colour components. When this model is associated with a precise description of how the components are to be interpreted (viewing conditions, etc.), the resulting set of colours is called a *colour space.*

Colours as read straight out of e. g. digital cameras, scanners, etc. are dependent on the calibration and characteristics of a specific device. Therefore, they are referred as *device-dependent colour spaces* (see Sect. 2.3.2). For quantitative reasoning on colour space values usage of an *absolute colour space* is essential. In colour science, there are two meanings of the term "absolute colour space:"

- A colour space in which colours are unambiguous, that is, where the interpretations of colours in the space are colourimetrically defined without reference to external factors.
- A colour space in which the perceptual difference between colours is directly related to distances between colours as represented by points in the colour space.

The first refers to colour management due to device abstraction into an independent, intermediate colour space (described in Sect. 2.3.3) by means of a colour management process as described in Sect. 2.3.5.

The latter is a fundamental requirement to be able to apply linear differential reasoning on the actual colour tuple values. Theoretically it is possible to perform a transformation like this on device-dependent colour spaces. However, without prior characterisation of the linearity of a device this is pointless. Therefore it is always considered together with device-independent colour spaces. Some perceptionally linear device-independent colour spaces are outlined in Sect. 2.3.4.

2.3.2 Device-Dependent Colour Spaces

Device-dependent colour spaces are those, that are only meaningful relative to a specific device, and potentially only under specific conditions. These can be distinguished in input colour spaces (cameras, scanners, etc.) and output colour spaces (CRT screens, flat panel screens, printers, etc.). It is obvious, that the colour representation of a camera for example will vary with changing light conditions, and potentially also between cameras of different vendors. Colour output will also largely differ on a variety of screens, and even more on printers using different printing media (paper, transparency, inks, etc.).

Therefore the RAW RGB (not gamma corrected) of a good digital camera will *not* match the RGB of a CRT screen (gamma correction needed), it will be different from the gamma corrected output as produced by the camera's firmware, and it will definitely be quite different from RGB (or even CMYK) output on printers. All these colour spaces are dependent on a specific device, and usually only accurate under certain fixed conditions.

Even within the family of RGB colour spaces there is large ambiguity, as well as there are errors associated with the conversion between them [17]. The manufacturer, specific model (even the production batch and individual device), the age, illumination situation, temperature, media and many more factors influence the colourimetric match of the devices.

2.3.3 Device-Independent Colour Spaces

Each device is different in its colour sensing characteristics. Cameras often render their images in RGB, YUV or other colour spaces, and screens for example use RGB for input. But the displaying properties (e. g. through the phosphorus) are usually different from those of the capturing device. Therefore it is important to distinguish between device-dependent (RGB, YUV, HLS, CMYK, etc.) and device-independent colour spaces that have been "normalised" and are therefore device-independent.

The already mentioned CIE XYZ and sRGB are examples of device-independent colour spaces, as opposed to a generic RGB colour space. But even among RGB colour spaces a great amount of ambiguity is large (see Fig. 2.9).

A popular way to convert colours from a device-dependent colour space as RGB into absolute colour is to determine an ICC profile (see Sect. 2.3.5), which contains the characteristics of the RGB device. This is not the only way to express an absolute colour, but it is the standard in many industries. Usually these ICC profiles contain the "rules" in the form of per-channel curves and additionally either matrices or *look-up tables* (LUTs) to convert the specific device's colour space – under certain given conditions – to or from an intermediate, device-independent colour space. This intermediate space is also called a *"profile connection space"* (PCS). This is either CIE XYZ or CIE LAB.

One absolute colour can generally be converted to another absolute colour, and back again; however, each colour space has its own gamut, and converting colours that lie outside

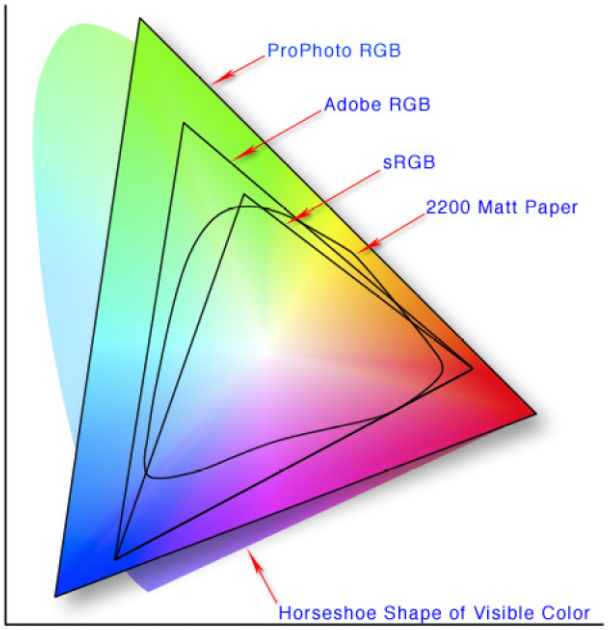

Figure 2.9: Colours representable by various device-independent RGB colour spaces in an xy-chromaticity diagram compared to coated paper.

that gamut will not produce correct results. Due to its limitation in size of representable colour gamut, sRGB therefore is usually not used as a PCS. There are also likely to be rounding errors, especially if the range consists of only 256 distinct values per component (for 8 bit encodings).

Also note that one part of the definition of an absolute colour are the viewing conditions. The same colour, viewed under different natural or artificial lighting conditions, will look differently. Those involved professionally with colour matching may use viewing rooms, lit by standardised lighting and conditions as are defined for the device-independent colour space used.

Occasionally, one may find explicit rules for the conversion between non-absolute colour spaces. For example HLS is defined as a mapping of RGB. Both are non-absolute, but a conversion between them should retain the same colour. However, in general, a direct conversion between two non-absolute colour spaces (for example, RGB to CMYK) or between absolute and non-absolute colour spaces (for example, RGB to $L^*a^*b^*$) is usually a meaningless concept, as they are lacking a common reference. Although formulae are available that give results that are with little accuracy only remotely comparable.

2.3.4 Perceptually Linerarised Colour Spaces

A great flaw in perceptional non-linearity of RGB and XYZ for comparative (linear) computations is this: Just noticeable colour differences are much greater in some regions (of the

RGB or XYZ colour space) than in others. A solution to this is a perceptionally linearised, device-independent colour system that relates Euclidean distances between colours (points in colour space) with perceived differences.

One of these has been established in 1912 by the American artist Albert H. Munsell [18]. He was the first to separate *hue, value* (or lightness) and *chroma* (roughly saturation) into perceptually uniform and independent dimensions. The Munsell system, and particularly the later renotations, is based on rigorous measurements of human subjects' visual responses to colour, putting it on a firm experimental scientific basis. Because of this basis in human visual perception, Munsell's system has outlasted its contemporary colour models, and though it has been superseded for many uses by models such as CIE LAB (see below) and CIE CAM02, it is still widely in use today.

The CIE specified in 1976 simultaneously two colour spaces as perceptually uniform versions of the XYZ colour space, which today have superseded most other models. These are namely the $L^*a^*b^*$ and the $L^*u^*v^*$ colour spaces [19].

CIE LUV has been a popular colour space for lighting, video and photographic applications. The linearisation of this space has been achieved pragmatically by means of various projections (onto an imaginary "curved" surface), until the numerical difference between the tristimulus values of two colours approximated the perceived difference between the colours. In contrast the values in the CIE LAB space are computed through four transformation steps, which are intuitively logical when examined separately, and have been derived through a combination of perceptionally sensible operations.

Both, CIE LAB and CIE LUV, are opponent colour spaces. In both of these the component L^* represents the lightness of the colour. The chromaticity of the colour is represented in the contributions of the a^* and b^* values for CIE LAB or the u^* and v^* values for CIE LUV respectively. In the CIE LAB space (which is more extensively used in this thesis), the a^* value defines the chromaticity raging from green to red (negative to positive), whereas b^* ranges from blue to yellow (negative to positive).

It is considered sensible to always use CIE LAB to represent surface colours, and not CIE LUV. Validation studies [20] of several colour appearance models have been performed. Overall, these studies showed that different models performed well in predicting some types of colour judgements but not in others. In every case, however, CIE LAB typically performed well against more complex models and occasionally was as good as the best; CIE LUV was often the worst.

These failings aside – from the same standing start, together with CIE LUV – CIE LAB has become the standard colourimetric space worldwide. CIE LAB remains the most practical and widely implemented colour model available. Indeed, several extensions of (or revisions to) the basic CIE LAB framework – including S-CIE LAB and RLAB – have been published to improve its performance and suitability for a wider range of colour modelling problems. Colours are not forced into an arbitrary geometric shape that distorts perceived

colour relationships. These extensions are possible because the CIE LAB geometry does not impose an arbitrary symmetry on chroma, hue or lightness. The calculation from XYZ values to $L^*a^*b^*$ values is simple and easily inverted (from $L^*a^*b^*$ values back into XYZ), and can be replaced by a variety of chromatic adaptation transforms.

2.3.5 Colour Management

In digital imaging systems, colour management is the controlled conversion between the colour representations of various devices, such as image scanners, digital cameras, monitors, TV screens, film printers, computer printers, offset presses, and corresponding media.

The primary goal of colour management is to obtain a good match across colour devices; for example, a video which should appear the same colour on a computer LCD monitor, a plasma TV screen and on a printed frame of video. Colour management helps to achieve the same appearance on all of these devices, provided the devices are capable of delivering the needed colour intensities [21].

Handling of colour on computers is a complicated business, given the wide variety and near infinite combinations of video cards, displays, printers, ink and cameras. Trying to join each input and output device individually would lead to an inflationary unmanageable numbers of conversion profiles. Colour management implemented by means of ICC profiles can bring some order to the chaos while preserving more of the richness of colour that human eyes can perceive. These profiles usually convert to and from an intermediate profile connection space (PCS). Therefore, per device and conditions only one profile is needed.

Colour management and the involved processes are much more extensively described in Chap. 5.

2.3.6 Illuminant Adaptation

Colour perception is strongly influenced by the illumination properties (see also Sect. 2.2.1). ICC transformations are standardised and only directly usable together with CIE D_{50} light. To be usable with alternative illuminants, additional adaptive transformations need to be performed on the colour tristimuli. This process is a computational replication of the human eye's psycho-physical perception phenomenon. It is called a *chromatic adaptation transformation* (CAT). Whenever one colour tristimulus needs to be adapted from one reference white (illuminant) to another one, a CAT needs to be performed. Unfortunately the CIE standard specifies for the common industry standard XYZ and $L^*a^*b^*$ colour spaces (see Sect. 2.3.4) the "wrong" *von Kries* transformation (linearly scaling the values in the CIE XYZ colour space). In the superior "true" von Kries transformation, the XYZ colour tuple is converted prior to scaling to the cone response domain LMS (see Sect. 2.2.1). Still better – and today the *de facto* "standard" and best practise for a CAT – is the Bradford transformation [22, 23]. The transformation has been determined empirically. It originally contains

a non-linearity, which is due to its almost negligible influence mostly omitted today. The Bradford CAT can simply be used through a matrix multiplication with the XYZ colour vector.

The general CAT in its linear form (2.14) is similar to the RGB to XYZ transformation (2.12). It is transforming the appearance from a source white s to the destination white d. These "whites" are representing the colours of the illuminants. The 3×3 transformation matrix $\boldsymbol{M}$ (2.15) is composed of a transformation from XYZ to the cone response domain of the eye $\boldsymbol{M}_A$, a multiplication with the diagonal matrix adapting each channel's white point linearly from source to destination white, and finally the inverse transformation back to XYZ colour space $\boldsymbol{M}_A^{-1}$. The adaptation parameters for the diagonal matrix are obtained through an equivalent transformation of the whites in XYZ colour to the cone response domains (2.16) and (2.17).

$$\boldsymbol{\rho}_d = \boldsymbol{M} \cdot \boldsymbol{\rho}_s \tag{2.14}$$

$$\boldsymbol{M} = \boldsymbol{M}_A \cdot \begin{bmatrix} l_d/l_s & 0 & 0 \\ 0 & m_d/m_s & 0 \\ 0 & 0 & s_d/s_s \end{bmatrix} \cdot \boldsymbol{M}_A^{-1} \tag{2.15}$$

$$(l_s, m_s, s_s)^T = (X_{ws}, Y_{ws}, Z_{ws})^T \cdot \boldsymbol{M}_A \tag{2.16}$$

$$(l_d, m_d, s_d)^T = (X_{wd}, Y_{wd}, Z_{wd})^T \cdot \boldsymbol{M}_A \tag{2.17}$$

The different CATs now differ only in the composition of the matrix $\boldsymbol{M}_A$. In the "wrong" von Kries transformation (XYZ scaling), this is the identity matrix (containing only values of 1.0 in the diagonal). Von Kries uses the "true" LMS cone responses, whereas the Bradford transformation uses the mentioned empirically determined transformation matrix.

All these computations try to compensate for shifts in the illuminant. Only the colour tristimulus is available as a base, whereas the full SPDs of the colour is lost. Grossly different spectral compositions of light *will* lead to a different colour impression. Therefore, colour tuples are only meaningful in conjunction with their respective illuminant's white point for similarly "behaved" illuminants. The CATs convert from one reference white to another. The reference white together with the black point define the tilt of the neutral axis, for which is compensated. So the approximation of the chromatic adaptation is only a good approximation for neutral colours (greys). It progressively becomes less accurate as colours are located further away from the neutral axis, and the more the illuminants differ from each other (or from the standardised CIE daylight SPDs) [22]. These CATs work very well for illuminants that are similar to daylight. For constancy between (strongly) differing illuminants the quality of the transformation result degrades.

Chapter 3

Colour Adaptation

On the road to the goal of an automatic, dynamic colour correction we are still missing algorithms to estimate the properties of the (current) illumination and methods to adapt the colour representation.

Colour constancy refers to the everyday perception that the colours of objects remain unchanged across significant changes in illumination colour and luminance level. Fairchild explains [20] colour constancy to be served by the mechanisms of chromatic adaptation and memory colour. It can easily be shown to be very poor when careful observations are made. The main reason for this is, because colour constancy is a result of subjective corrections in the human visual system. In everyday life we mostly perceive colours as not changing at all, as we often remember colours rather within their context than look at them closely. Studies of colour appearance and the derivation of colour appearance models are aiming to quantify and predict the failure of colour constancy. Therefore, research in this field can potentially lead to theories describing how the human visual system might strive for approximate colour constancy and the fundamental limitations preventing colour constancy in the real world. Such studies generally take place in the area of computational colour constancy with applications in machine vision.

Chromatic adaptation is a sub-process of colour constancy, and it is often used synonymously. In a way they are largely similar. Chromatic adaptation tries to model the *perception* of a scene under a given illuminant by transforming it towards a different illuminant, e. g. daylight. In contrast, colour constancy tries to identify the (nonexistent) *precise* perception of the same scene actually illuminated under the different illuminant. Which will often fail, usually due to metameric aberrations. As slim differences between chromatic adaptation and colour constancy are slim, and these terms are often used ambiguously, we will discuss these fields together.

Already Goethe conducted experiments on light, trying to accurately depict the complexities of human colour perception. The emphasis on brightness and boundaries correlates with modern research on the perception of colour [24]. Goethe was focused on exploring

how colour is perceived in a complex array of conditions. In contrast Newton's studies [7] were aimed more towards the development of a mathematical explanation for the behaviour of light.

First, in Sect. 3.1 we will show a general overview of classes for chromatic adaptation. In the following, different analytical methods for colour adaptations are presented: Spectral adaptation in Sect. 3.2, computational colour constancy in Sect. 3.3, histogram colour correlation in Sect. 3.4 as well as *a priori* knowledge based methods in Sect. 3.5. Finally, Sect. 3.6 presents a variety of ways to learn colour constancy. The chapter concludes with a comparison of the different outlined adaptation methods in Sect. 3.7.

3.1 Approaches for Chromatic Adaptation

To offer a possible solution to the problem for proper quantitative measuring, the retrieved image of the digital camera needs to be normalised. As we are currently only concerned with the colour information of the picture, the image will only be chromatically adapted. This chromatic adaptation is not to be confused with the *chromatic adaptation transformations* (CAT) as outlined in Sect. 2.3.6. For a lack of an alternative term that is equally descriptive, we are using the same term in a more broad way for all types of adaptation of colourimetric information. Of special interest are adaptations that go beyond the possibilities of the CATs of Sect. 2.3.6.

To compensate for the chromatic alterations in the capturing process, two principal possibilities are available: By measuring influencing contributions towards the capturing process externally (e. g. light intensity and spectral distribution, temperature), and from there inferring the colourimetric deviation. The other way is to use the camera response directly to deduce colour transformation rules. As the first way requires a fair amount of additional sensors, and it will most likely be incomplete in capturing all effects, and additionally it is very camera-dependent. Besides error estimations on the basis of the additional measurements, no evidence of researchers following this path could be found so far. Therefore, we will not further discuss it. For completeness it has been mentioned, but the direct camera based way is more robust, so we will focus on approaches analysing cameras' imagery.

For the chromatic adaptation we have got to consider three different cases: A static adaptation, a dynamic adaptation over time, and the case where we cannot find a (proper) adaptive solution.

3.1.1 Static Chromatic Adaptation

In cases facing a static illumination (not changing over time), a static chromatic adaptation can be performed. In this process, the capturing device (camera) is characterised within

the given environment. This is commonly achieved by using a transformation profile *(ICC profile)* as standardised by the International Color Consortium (ICC). These profiles describe a mapping of device colour to independent colour, and they are determined through a characterisation process capturing a (larger) number of precisely known colour patches.

One of the limitations for this proceeding is a statically illuminated scene (see Sect. 6.2). Additionally, the camera's scene perception has got to be rendered in a spatially homogeneous way, or the image has got to be retrieved in a raw format (RAW mode in some cameras).

This process of static chromatic adaptation closely relates to the common practise in colour management (see Chap. 5) as practiced within the graphic arts industry [25, 26]. Input and output devices are individually characterised within their given specific working environments by ICC profiles. All data "in between" is then handled in a device-independent colour space. This is usually the $L^*a^*b^*$ (or more precisely CIE $L^*a^*b^*$, commonly abbreviated as CIE LAB) colour space as specified by the International Commission on Illumination CIE. Further information on this and the involved colour spaces can be found in Sect. 2.3.5 and 2.3.1 as well as Chap. 5.

To use cameras as quantitative measuring devices for colour in this way, the camera's colour mapping characteristic (e. g. captured in the form of an ICC profile) and the current illumination conditions must be known and kept constant. The camera's image is first normalised (using its specific ICC profile) and then can be chromatically adapted from the current illumination to a light source (if not, the D_{50} standard is used). Colour information normalised this way can be analysed quantitatively without severe colour aberrations.

3.1.2 Dynamic Chromatic Adaptation

Under common conditions, the biggest problem is the fact that the light source in a scene is not known and/or not constant. As we know from our own experience, our eye can adapt quite well to perceive a sheet of paper to be white, and a ripe lemon always to be yellow. This is true regardless of commonly encountered light sources. The lemon does not look greenish under a blue sky or orange at the dusky reddish light, even though the light our eye senses is of that shade. We do not perceive the object alone, but the object within its context, and the "scene rendering" algorithm (see Sect. 2.1) in our brain performs the chromatic adaptation.

Therefore, other means of normalising the sensor response will need consideration. However, from an initial known condition, a system may be constructed to have the capability to dynamically adapt the transformation profile from its initial characterisation. For this purpose the initial characterisation could be handled by a properly established ICC profile (as described above). If the system is able to track changes in perception of colour in certain unchanged parts of the scene, it would be possible to deduce an ICC profile "deformation" over time (see Chap. 8 and 9).

Due to uncertainties, and a possible accumulation of errors, this process will be less precise than a strict static chromatic adaptation. But it should still give more precise information on the colours perceived as well as some valuable information on an error estimation.

The unchanged portions of the observed scene may consist of background information (segmented from the complete scene), or even some (few) chromatically known coloured objects or patches placed conveniently within the scene. Considering that most scene changes only occur at a low rate (compared with the frame rate of video cameras) these changes can become trackable.

This process also can only work in a stable way in the case of spatially homogeneous scene rendering within the camera or RAW image retrieval as above in the static case.

3.1.3 Failing Adaptation

As outlined above, proper chromatic adaptation depends on spatially homogeneous scene rendering, because the colourimetric adaptation transformation is applied to each pixel of the image independent of its location. To be precise, in any case where images are treated inhomogeneously, these adaptations will fail, as the camera's scene rendering is unknown. Also, any adaptation will fail, if the camera's own adaptation is lacking precision, the required speed of adaptation, or if the adaptation algorithm leads to harmonic interferences with e. g. the frequency of fluorescent illumination.

3.2 Spectral Adaptation

Fairchild discusses a very interesting *full* spectral adaptation [27]. It is based on the complete reflection model (see Sect. 2.2.1), and takes the SPD of a light source, the spectral reflectance, and the spectral sensitivities into account to derive the perceived colour tristimulus. This method is only possible if full spectral reflectance information for each pixel is available (e. g. images of flat objects), or where a specialised capturing device is sensing full SPDs for each pixel.

This approach is usually not possible, as the commonly available imaging devices only yield a colour tristimulus rather than a full spectral distribution. As spectral information is lost in sensing the tristimulus using common cameras, full SPD information of the scene cannot be recovered, which ideally would be necessary to perform full colour constancy transformations [28]. Having said that, some researchers are experimenting successfully with using colour imaging using multiple primaries beyond red, green and blue [13, 14]. However, the complexity in custom hardware for this is significant.

Additionally, this approach is only applicable if the surfaces in the scene are *all* segmented precisely, and their exact orientation and specific relative spectral reflectance is known. This is not possible for current common applications. However, it is rather an interesting theoretical problem.

3.3 Computational Colour Constancy

Barnard has established a comprehensive overview of various algorithms for illuminant estimation and colour constancy [2]. He extensively evaluates the algorithms' relative strengths and weaknesses, he contributes improvements to some of them, as well as aims to provide a good and fair comparisons of them.

Unfortunately, all his analyses were conducted in device-dependent colour spaces. Additionally, colour differences were quantified as Euclidean distances in a non-linearised RGB space (gamma corrected). As previously described, colour comparisons should be performed in a suitable colour space. For this usually a ΔE value derived from CIE LAB is considered to be current good practise [20]. This does not reduce the quality and ambitions of the analyses, but shows pointers for a direction of further research in this field.

A recent – and even more comprehensive overview – has been published by Ebner [1]. It gives a very detailed introduction to the topic of colour vision and colour constancy under physiological as well as computer science aspects. Various colour constancy algorithms are described and compared to each other with all their aspects of capability in uniform and non-uniform illumination environments, colour constancy learning, illuminant estimation, etc. Additionally, all algorithms are outlined and discussed in pseudo code with a detailed description of their parameters, etc.

Computational colour constancy examines every image individually. It tries to infer information from the image's content, and uses that to create an adaptive transformation that can be used for normalising its colour appearance.

3.4 Histogram Colour Correlation

Porikli has taken a completely different approach [29]. He stays entirely in the camera's own colour space, and matches the imaging characteristics directly between two distinct cameras or illumination conditions. For this, initially through histogram cross-correlation a direct mapping of the colour channels is performed. Therefore, it is possible to directly map between the sensor responses for known camera or lighting combinations. This approach circumvents the problem of accumulative mapping errors in a series of transformations. However, at the cost of the need for individual pair mappings of each setup, and no additional knowledge on conditions has been gained.

Currently only for the case of scenes with similar colour composition results have been published. Due to the fact that complex channel distortions are possible, this method may prove to be useful if applied in other colour matching attempts. Channels by themselves can be transformed with arbitrary mapping functions.

This approach is in a more standardised way also available through colour management by computing direct device link (ICC) profiles. Device link profiles are even more general

and may also cover the more complex colour space transformations. However, Porikli's way to deduce this information is quite unique in its approach for deriving these transformations.

3.5 A Priori Knowledge Based Methods

Many times, especially in robot vision, a few constant colour features in a given environment reoccur frequently. This may be e. g. in traffic (traffic signs), robotics (object and environment markings), etc. A common approach in these cases is often a colour based image segmentation, and successively applying *a priori* knowledge on the segmentation result to deduce information in a changed colour appearance.

The segmentation is usually done in multiple stages, feeding the first pass results into an area clustering for a continuous region extraction. These systems may be supported by various heuristics, threshold training through neural networks, probability maps, shape detection, etc. [30]. Others use colour labelling, region connection, and positional composition (colour coding of markers), etc. [31].

Other researchers take a different step in segmenting the coloured scene into distinctly known colours. Reyes uses a method called "Colour Contrast Fusion," in which for each known colour a set of image colour space transformations are performed to increase the detectability of the individual colours [32]. The pixels for that colour are then segmented through comparison with hue and saturation descriptors within an rg-chromaticity space. Current research in that direction aims at using genetic algorithms to determine a best set of descriptors and transformation parameters [33, 34].

Many other tools are usually combined for a pragmatic solution of the given problem. These systems need to "just work" for one specific use case (e. g. robot soccer), and the only information needed is the localisation of the given markers and colour encoded objects (ball, team colours, goal, other markings) within the images. Only a finite, limited number (about ten) colour classes are distinguished. Therefore, further image analysis is not undertaken to retrieve a scene in its corrected colours.

The environments faced are usually quite constant regarding illumination during operation of the setup, so an initial parametrisation beforehand is permissible. For these, purely technical approaches to colourimetry using device-dependent and non-linear colour spaces directly from the (robot's) camera system (RGB, YUV) often prove sufficient.

However, in cases that some objects of existing *a priori* knowledge should be present in the scene, these approaches may yield a valuable contribution to the overall scene correction. More precise knowledge about an identified object can be used as a reference for a more deterministic adaptation [34]. Additionally, the quality of *a priori* knowledge based approaches could benefit from being applied using more suitable (linearised, device-independent) colour spaces.

3.6 Learning Colour Constancy

Different researchers felt inspired by the robustness of biological systems in their ability to adapt their colour vision to ever changing conditions. This section will show a cross section of attempts to achieve colour constancy on digital systems by means of utilising methods of Artificial Intelligence (AI).

3.6.1 Biologically Based Approaches

Inspired by the human visual system, many scientists have tried to mock biological phenomena in colour constancy to yield colour corrected image perception. Courtney et al. have based an approach on psycho-physical data from a model of the human visual cortex *V4* [9]. Spacio-chromatic problems in colour perception were met successfully using a Neural Network simulation software aimed at a physiologically correct neural simulation. In fact, the results were in accordance with the V4 stage of the visual cortex model as referred to in the research. Some other researchers have followed the path of implementing V4 like behaviour in software.

Within the visual cortex of the brain, the primary visual cortex (V1), as well as the areas V2 and V4 process colour information. It is quite well understood today, that the area V4 contains cells that respond to the color of objects irrespective of the light that illuminates the object [1]. But at the same time, the comparison of the light intensity of the light from a patch with the intensity of the surrounding area is accomplished by some as yet unknown mechanism. Therefore, looking at the research of Courtney et al. purely on this basis is a mere academically interesting approach. However, it does not bear any real significance besides outlining yet another possible way inspired by biological research. But this approach in itself provides an interesting view point to make use of the information available. One of the benefits of the implementation is that it avoids a "wash-out effect," as observed by many other colour constancy algorithms that work on non-uniformly lit scenes. This effect leads to a colour saturation wash-out towards the centre of more extensive, solidly coloured areas.

3.6.2 Neural Network Based Approaches

Neural networks are – apart from being biologically inspired – also frequently used to solve problems of computational colour constancy (see Sect. 3.6.1). Their advantage is, that it is not necessary to analytically determine the illuminant's properties and compensate for them. The idea is to train the system on a multitude of known cases to deduce a system capable of performing colour constancy. Compensation using the neural network after training can be performed in a computationally cheap way, even though the initial training can be quite extensive and time consuming. Neural network architectures can be applied in very different ways. The most common distinctions for colour constancy applications are towards global illumination compensation and local pixel colour compensation.

Funt et al. for example have used a global image compensation [35]. The image scene colour space is discretised. If pixels of the scene are present within a discrete region of the colour space, the state of a neuron in the input layer of the network is set to 1, otherwise it is set to 0. The output layer of the network consists of only two neurons reflecting the image's white point in chromaticity coordinates. These can be used directly to colour correct the image. The global neural network approach as used by Funt et al. compares quite well against a few sampled others of the "classic" colour constancy algorithms (see Sect. 3.3). And it even performs very well for certain "difficult" scenes with only very few coloured areas, in which several others fail.

In contrast, Bascle et al. use neural networks to process the pixels of a scene individually [36]. The network is trained on data from *individual pixels* sampled randomly. The images providing these pixels for the training result from different commodity camera models (web cameras) under a set of only a few common illuminants (fluorescent lights, incandescent light bulbs, natural sunlight). The trained network is laid out as a modified multi-layer perceptron. It learns to ignore the luminance of the input and estimates two colour invariants (resembling the chromaticity characteristic) that are independent to the illumination. The trained network is used to disregard illumination and shadows within the scene for a colour based segmentation. The quality of the indifferent chromaticity estimation was as good or better than the "classic" colour constancy algorithms (see Sect. 3.3 again). As a look-up table could be compiled from the trained network result, the computation performance on the sample data was very fast as well. Bascle et al. went into great detail about comparing the implementation to a large variety of other algorithms with respect to their strengths, weaknesses and performances.

3.6.3 Probabilistic Learning

Rosenberg et al. [37] are also using chromaticity histograms, but using a different approach to Porikli's histogram colour correlation (see Sect. 3.4). Due to the logarithmic chromaticity space used, the illuminant's chromaticity will shift a colour linearly. In this approach one needs to establish (from a training set) the probabilities of a given chromaticity to appear in an image under a given illuminant's chromaticity. The individual probabilities are accumulated in a chromaticity histogram. The estimated illuminant is determined through a maximum likelihood approach through histogram similarity computations. Different methods for performing histogram similarity metrics are also compared.

Also this example of a statistical learning approach performs quite well in quality against common classic colour constancy approaches like the Grey World Assumption or the White Patch Retinex algorithm. Although, the quality comes at a price of being a rather computationally expensive algorithm.

Van de Weijer and Schmid are using an approach on the basis of the Bayes theorem [38]. Colours are histogrammed in a perceptionally linear colour space ($L^*a^*b^*$) and colour *names*

are assigned probabilistically [39]. This approach is not purely based on chromaticity values – as it incorporates achromatic colours – additional information is preserved (e. g. red → brown in the dark, → pink in the highlights). The identified colour name was joined in a "bag of words" scheme with other information (here: some shape information) to identify certain categories of objects in a scene.

This colour name approach was very successfully tested against a simple colour segmentation identifying soccer teams or flowers in scenes. For these purposes of classifying limited, colour-coded items in scenes it worked very robustly. But it does not allow to deduce information on the illumination, surface reflectance or object chromaticity.

3.6.4 Genetic Programming Based Approaches

Ebner has taken the approach of building a neural architecture. He arranged an array of processing elements for each pixel of the image. Each processing element has got access to the full pixel values of its own pixel and the four neighbours' luma values (similar to neural connections between spatially arranged colour processing cells in the visual cortex). A set of operation functions was given, and through genetic programming a suitable algorithm for computing a colour corrected image evolved [1, 40].

Due to the extreme long processing times for the genetic programming (10 days per run) – and due to the fact that six out of ten times the genetic algorithm was stuck in a local optimum – the approach does not seem very promising. However, it did show that genetic evolution based on biologically inspired boundary conditions can come up with suitable solutions for the given problem. Upon analysis of the evolved "best candidate" algorithm, it also showed (due to colour channel independence) processing similarities with the Retinex theory, which averages data from neighbouring elements.

3.7 Adaptation Method Comparison

The general problem of usability of Fairchild's spectral approach with general image capturing systems is obvious, as the spectral colour information is lost, and therefore it is only applicable for very specific and costly systems. Further limitations due to required precision of a scene segmentation apply as well.

The computational colour constancy methods – as described in the summary of Barnard's Ph. D. thesis [2] and the more general approaches in Ebner's book based on his habilitaton [1] – depend on a static analysis of the current scene to deduce illuminant information. It does not consider additionally information available in live image analysis.

Histogram based inter-camera calibration – as performed by Porikli [29] – demands single initial calibration of distinct device pairings. The number of possible distinct pairings explodes exponentially with the number of available devices. The approach only matches

information between the devices, but does not allow to deduce information on estimated chromaticities. The mappings for each channel are very flexible, but as these functions only operate on one channel at a time, no shearing, warping or rotation of the colour space is possible. Colour management based colour corrections allow for similar types of correction possibilities while still allowing for more complex colour space distortions.

Many methods are known to be used that are based on *a priori* knowledge of scene properties, like distinct colours of object markings (e. g. the ball colour, team markings, goal marks, etc. in robot soccer), colour names, etc. Use of these is definitely helpful, but their presence cannot be expected in a scene at all times under more general conditions.

Generally, the idea of learning colour constancy sounds promising towards a more dynamic and flexible solution. Either through statistical approaches or by means of Artificial Intelligence. Earlier attempts from the beginning of the 1990s were partially based on biologically-inspired implementations. Though these yielded good results, the theory on which they were based is more derived from guesswork of understanding the processes in the human visual cortex, and lacking true certainty. The approach taken by the implementation, however, is validly usable. Even though neural networks, as used in [9, 35, 36], are learning how to adapt colour, they do so only during a training phase. So they are still not able to adapt at run time to changing conditions.

Probabilistic methods [37, 39] as outlined are not used in an adaptive way. Their models may be updated statistically more easily, though, than a neural network can be re-trained.

Genetic programming promises to be the ultimate key to adaptive flexibility. But it is by far impracticable as its evolution consumes too much computational time to be useful even in a decoupled side-line process.

Common approaches based on colour management (see Sect. 3.1.1 and Chap. 5) yield comparable results as some of the above mentioned ones. Furthermore, they operate on a profile connection space of a colour encoding that is very suitable for further colour analysis. However, it demands an initial, individual and static characterisation of every device within a process. This step must be performed for every encountered condition, or upon every change into an unknown or new condition.

As all algorithms outlined in this review have been implemented with different use cases in mind, it is rather difficult to compare these directly. Researchers use a variety of different colour spaces for their computation. Reading the publications, it is obvious that many researchers are not even aware of certain problems with the used colour spaces. These can be for example a severe non-linearity (e. g. gamma corrected RGB), visual non-linearity, device dependence, etc. We can expect, that several of the approaches could perform better than stated in the publications, if a more suitable colour encoding would be used, and if adaptations to e. g. discretisations would be made accordingly. Especially as almost all algorithms make use of colour differencing for discretisation, thresholding or quality metrics, so potential improvements are possible.

Almost all the different methods of handling colour constancy outlined – or referenced here – provide items in the "researcher's toolbox" that may contribute valuable functions in a novel tool-chain towards solving the problem of Dynamic Chromatic Adaptation. Therefore, it has been chosen to explore a hybrid approach, that is based on the robustness of colour management with its device-independent, linearised profile connection colour space, which in corporates learning of current conditions based on input from the analytic methods described.

Chapter 4

Colour Constancy with $L^*a^*b^*$

Colour constancy algorithms aim at correcting colour towards a correct perception within scenes. To achieve this goal they estimate a white point (the illuminant's colour), and correct the scene for its influence. In contrast, colour management performs colour transformations on input images according to a pre-established input profile (ICC profile) for the given constellation of input device (camera) and conditions (illumination situation). The latter case presents a much more analytic approach (it is not based on an estimation), and is based on solid colour science and current industry best practises. Unfortunately, it is rather inflexible towards cases with altered conditions or capturing devices. The idea as outlined in this chapter is to take up colour correction on visually linearised and device-independent CIE colour spaces as used in colour management, and to try to apply them in the field of colour constancy. For this purpose, two of the most well known colour constancy algorithms – White Patch Retinex and Grey World Assumption – have been ported to operate on colours in the CIE LAB colour space. Barnard's popular set of imagery for testing colour constancy was used with the original implementations as a reference as well as the modified algorithms. The results appear to be promising, but they also revealed strengths and weaknesses.

4.1 Introduction

Colour constancy refers to the everyday perception, that the colours of objects remain unchanged across significant changes in illumination colour and luminance level [20]. Colour constancy is served by the mechanisms of chromatic adaptation and memory colour. The study of colour appearance and the derivation of colour appearance models are aiming to quantify and predict colour constant perception. Such studies generally take place in the arena of computational colour constancy.

In digital imaging systems, colour management is the controlled conversion between the colour representations of various devices, such as image scanners, digital cameras, monitors,

TV screens, film printers, computer printers, offset presses and corresponding media. The primary goal of colour management is to obtain a good match across colour devices; for example, a video which should appear the same colour on a computer LCD monitor, a plasma TV screen, and on a printed frame of video. Colour management helps to achieve the same appearance on all of these devices, provided the devices are capable of delivering the needed colour intensities [21].

RGB (and other) spaces model the output of physical devices rather than human visual perception. Due to their immediate availability, most colour constancy research is conducted using these device-dependent colour spaces. Colour constancy algorithms derive information based on the colour distribution in the RGB colour space, then shift and stretch colour along the colour primaries of this colour space. Differences in colour are quantified in terms of Euclidean distances in these non-linear spaces. Therefore, they may not be optimal for quantitative colour comparisons. Visually meaningful colour differences are usually expressed as a ΔE value derived from CIE LAB, and are considered to be current good practise [20] (based on the L^*, $a^*$and b^*coordinates of the CIE LAB colour space, see Sect. 2.3.4). CIE LAB aspires to perceptual uniformity, and its L^* component closely matches human perception of lightness. It can thus be used to make accurate colour balance corrections by modifying output curves in the a^* and b^* components, or to adjust the lightness contrast using the L^* component. It seems like a good idea to apply colour constancy algorithms on a device-independent and linearised colour space as CIE LAB for an exemplary evaluation.

In colour management, the adaptation from one illuminant to another is performed by *von Kries* type transformations. Commonly accepted now as being the best candidate for these so called *chromatic adaptation transformations (CATs)* is the Bradford transformation [22,23]. Additionally to the direct transformation on the $L^*a^*b^*$ colour coordinates, we will be evaluating a Bradford CAT.

The work described in this chapter is trying to evaluate the possibility of implementing variations of colour constancy algorithms in opponent colour spaces – particularly in the device-independent, visually linearised CIE LAB colour space. It is *not* the goal to find a competitor for the "Best Colour Constancy Algorithm Award."

The following section will introduce colour constancy algorithms in general, and specifically the ones used here for comparison. In Sect. 4.3 these algorithms are modified towards being useful in an opponent colour space (CIE LAB). Sect. 4.4 explains some details towards the implementation and evaluation, with finally Sect. 4.5 presenting comparative experimental results between the different implementations.

4.2 Colour Constancy Algorithms

The basic idea of colour constancy algorithms is two fold: Firstly, to determine an estimation of the *white point* in an image, which reflects the illuminant's properties (its colouration);

Secondly, to compensate for the colour cast introduced by the illuminant. As a result the image will be "normalised" towards a more neutral representation, in a way similar to our human visual apparatus, performing a (more robust) colour constant perception. Usually, the white point also influences the brightness and contrast in the process of the compensation, yielding a better dynamic range of the image.

Due to the focus towards a general comparison towards the viability, and an estimation of influences on the working colour space, only two "classic" members of the colour constancy family have been implemented: The *(White Patch) Retinex* algorithm and the *Grey World Assumption* as outlined by Marc Ebner [1]. These algorithms have got their limitations, but they have been well known for a long time and should therefore serve as a good example. To ease comparability of the algorithm modifications, we are focusing on uniformly illuminated scenes only.

These colour constancy algorithms are based on the estimation of a *white point* by inspecting image characteristics. The white point is used to transform the image pixels' colour tuples to a corrected representation. For the transformation we are assuming a linear relationship between the response of the sensor and the value of a pixel's colour channel. As most RGB encoded images are encoded in the $sRGB$ colour space (strictly speaking being $R'G'B'$), the colour tuples are encoded with a *gamma* value. This encoding is performed through a nonlinear operation used to code and decode the tristimulus values. In the simplest case this can be expressed as a power-law transformation with an exponent approximation of $\gamma \approx 2.2$. Before applying any algorithms, this transformation must be "un-done" to transform the $R'G'B'$ to RGB.

4.2.1 White Patch Retinex

The Retinex algorithm relies on having a bright patch somewhere in the image. The idea is, that this patch reflects the maximum intensity of light possible for each band. Assuming that we are dealing with approximate Lambertian reflection, this will be the colour of the illuminant (interface reflection in Fig. 2.6 of Sect. 2.2.1). Once a bright patch has been located, its colour value can be used for an estimate of the white point.

Practically, instead of looking for a bright patch (or white patch) one determines the maximum of each band over all pixels. Problems arise with some "noisy" pixels, (single) white pixels, or "clipped" pixels (the colour channel value of a pixel exceeds the encoding maximum, and is therefore restricted to the maximum, e. g. 255). To increase robustness a cutoff value is introduced, using the value at a certain cutoff percentage off the top. The colour bands are re-scaled using this white point estimation.

The listing in Fig. 4.1 outlines the basic implementation of the White Patch Retinex algorithm. First, the illuminant is estimated through determining a white point of the lightest values in each channel. In a second step, the pixel values of the image are adapted to this estimated white point. The implementation is simplified in the point, that it uses

```
1  number_of_pixels = pixels in image
2  estimated_illuminant = [0,0, 0.0, 0.0]
3
4  # Estimating the illuminant.
5  for each colour channel in [R, G, B]:
6      sorted_by_value = all colour tuples of channel sorted by value
7      cutoff_index = (1.0 - PERCENTAGE) * number_of_pixels
8      max_value = sorted_by_value[cutoff_index]
9      estimated_illuminant[channel] = max_value
10
11 # Transformation for estimated illuminant.
12 for each pixel in pixels of image:
13     for each colour channel in [R, G, B]:
14         pixel[channel] = pixel[channel] / estimated_illuminant[channel]
```

Figure 4.1: Basic implementation of the White Patch Retinex algorithm operating on RGB colours. `PERCENTAGE` = 0.04 for white point estimation as described by Ebner.

the estimation of a white point only, and disregards the potential computation of a black point. The reason for this is that it becomes more comparable with the implementation of the Grey World Assumption (see below), which also only works on the determination of *one* parameter only.

4.2.2 Grey World Assumption

According to the Grey World Assumption, most scenes are sufficiently complex regarding their colour distribution. Therefore, one can assume that, on average, the world is grey. An image, showing a sufficiently large number of different colours can be assumed as being grey as well.

The space average colour of the image therefore can be used to estimate the illuminant. The white point is determined by scaling the average colour with the maximum luminance. As with the White Patch Retinex implementation (Sect. 4.2.1), for the maximum determination also a cutoff percentage is used for robustness reasons. All colour channels (RGB) will be scaled with the corresponding channel value of the white point.

The listing in Fig. 4.2 outlines the basic implementation of the Grey World Assumption algorithm. First, the illuminant is estimated by the average colour of the image, and the image's pixel values are corrected by this average. In a second step, the pixel values are re-scaled by a value derived from the new maximum luminances. This is necessary as the intensities are boosted too much after correcting with the average channel values. The implementation described by Ebner [1] computes the luminance as an arithmetic mean of the three channels (R, G and B), giving particularly the red and blue channel too much weight. The weights in line 8 have been used to account for the channels' intensity contribution in the D_{65} illuminant used in sRGB colours. This computation of the pixel's luminance seems

```
number_of_pixels = pixels in image
average_channels = per channel mean value of all colour tuples

for each pixel in pixels of image:
    for each colour channel in [R, G, B]:
        pixel[channel] = pixel[channel] / average_channels[channel]

luminances = all colour tuples * [0.2125, 0.7154, 0.0721]
sorted_luminances = sort(luminances)
cutoff_index = (1.0 - PERCENTAGE) * number_of_pixels
max_luminance = sorted_luminances[cutoff_index]

for each pixel in pixels of image:
    for each colour channel in [R, G, B]:
        pixel[channel] = pixel[channel] / max_luminance
```

Figure 4.2: Basic implementation of the Grey World Assumption algorithm operating on RGB colours. `PERCENTAGE` = 0.02 for white point estimation as described by Ebner.

more chromatically correct, and gives better comparable results between the implementation modifications discussed in this chapter.

4.3 Algorithm Modifications

The CIE LAB colour space is visually linear, and independent of characteristics of the (capturing) device. An $L^*a^*b^*$ colour space is a colour-opponent space with dimension L^* for luminance and a^* and b^* for the colour-opponent dimensions, based on the non-linearly compressed CIE XYZ colour space coordinates. The luminance is a perceptual scale of the brightness as a nonlinear function of the relative luminance Y (of CIE XYZ).

a^* and b^* capture the chromaticity of the colour. Therefore, these are de-coupled from any intensity information. An estimation of a white point has to be dealt with differently in this colour space. Additionally, the adaptation to the estimated white point has to be performed differently. Scaling is applied to the L^* channel only, and scaling on a^* and b^* introduces shifts in chroma (saturation) of the colour tuples.

4.3.1 White Patch Retinex

As mentioned above, the only component in $L^*a^*b^*$ related to intensity is L^*. Therefore, it is not sensible to determine per channel maxima, so this step is to be limited to the luminance. The colour tuples of the whole image are sorted by their L^* component. The distribution of possible colour values in the $L^*a^*b^*$ space converges to a tip towards both extreme points in luminance, at the white point as well as at the black point. The values at

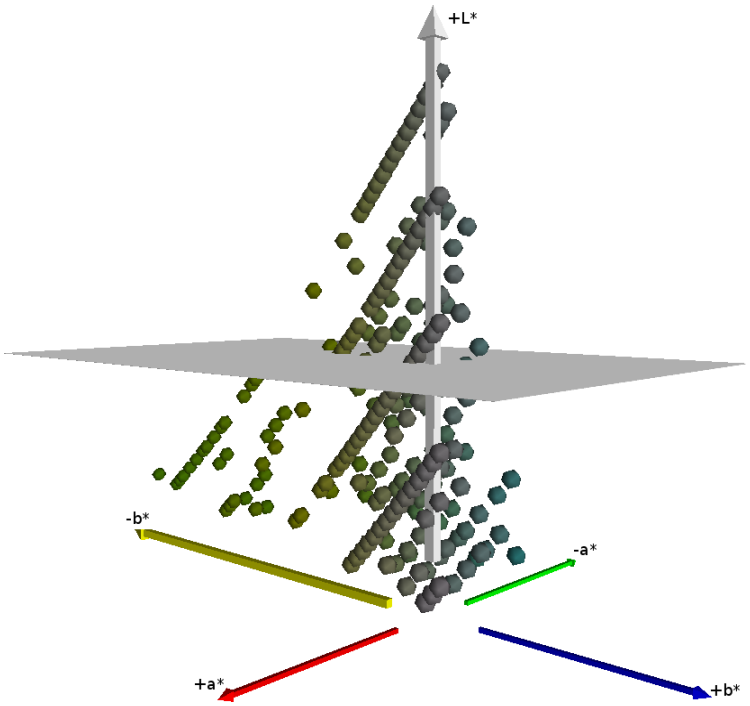

Figure 4.3: The locations in colour space of the pixels with the highest luminance are shown as spheres in $L^*a^*b^*$ space. The cutoff plane is indicated in grey.

```
number_of_pixels = pixels in image
sorted_by_luminance = all colour tuples sorted by luminance
cutoff_index = (1.0 - PERCENTAGE) * number_of_pixels
max_luminance_pixels = sorted_by_luminance with (index >= cutoff_index)
estimated_illuminant = per channel mean value of max_luminance_pixels
```

Figure 4.4: Basic steps for the illuminant estimation of the $L^*a^*b^*$ modified White Patch Retinex algorithm. `PERCENTAGE` = 0.04 for white point estimation as described by Ebner.

the defined cutoff point in the luminance channel (see Sect. 4.2.1) are co-located in a quite small space within the a^*b^* plane. However, they may introduce randomness of significant scale into the white point determination. Fig. 4.3 shows the cutoff plane, that bounds the lightest fraction of the image pixels. A pixel closest to that plane taken could by chance lie anywhere in the cross section of the plane within a hull surface of the pixel colours. To suppress this element of randomness, the $L^*a^*b^*$ values above the cutoff point are averaged for the determination of the white point.

The listing in Fig. 4.4 outlines in pseudo code the basic steps taken to estimate the illuminant from the image's colour tuples. For simplicity, it is limited to all pixels, including all clipped pixels. The resulting `estimated_illuminant` then is used in the chromatic adaptation (see Sect. 4.3.3).

```
1 number_of_pixels = pixels in image
2 average_channels = per channel mean value of all colour tuples
3 sorted_luminances = sorted L value of all colour tuples
4 cutoff_index = (1.0 - PERCENTAGE) * number_of_pixels
5 max_value_L = sorted_luminances[cutoff_index]
6 average_L = L value of average_channels
7 estimated_illuminant = average_channels * max_value_L / average_L
```

Figure 4.5: Basic steps for the illuminant estimation of the $L^*a^*b^*$ modified Grey World Assumption algorithm. `PERCENTAGE` = 0.02 for white point estimation as described by Ebner.

4.3.2 Grey World Assumption

The algorithm modifications for the Grey World Assumptions are much easier to accomplish. The average colour can be determined by averaging over all channels of $L^*a^*b^*$. The L^* component for the white point does not require averaging beyond the cutoff point, but can be used directly at the cutoff point's index (see pseudo code listing in Fig. 4.5).

Again, the resulting `estimated_illuminant` is used in the chromatic adaptation (see next Sect.).

4.3.3 Chromatic Adaptation

The adaptation to the estimated white point does not require scaling in all channels of $L^*a^*b^*$ space, as it is performed on RGB colours. This adaptation to channel scaling is outlined in pseudo code in the listing of Fig. 4.6. Scaling can only be performed on the L^* channel. Values in the a^*b^* plane are shifted (lines 11–13) by an amount linearly interpolated between the (ideal) black point and the white point (lines 6–9), depending on the value of L^*(line 7).

Experiments with the $L^*a^*b^*$ based implementation of the two above mentioned algorithms worked quite nicely in general. However, strong differences in the encountered illuminant exposed a very weak visual quality of the colour corrected images. Colours were often over compensated, and colours were often strongly under saturation.

A reason for this is assumed to be found in the fact that the colour adaptation has been performed on channels of perceived colour, which are not related to intensity (luminance), but to visual luminance and chromaticity. As we are dealing with colour defined on a visual perception basis, it is believed that a transformation within the cone response domain of the human eye yields best results. Colour scientists nowadays adapt to illuminants' colour influences using *von Kries* transformations [20]. Common best practise for CATs is the Bradford transformation [22, 23], as described in Sect. 2.3.6. It has also been implemented to adapt to the new estimated white point. This Bradford transformation replaces the shifting/scaling based on $L^*a^*b^*$ colours only (as described in this section), while maintaining the identical algorithms (from Sect. 4.3.1 and 4.3.2) for the white point estimation.

```
# Transformation value for L channel.
max_L = maximum L value of all image colour tuples
scale_L = max_L / estimated_illuminant.L

for tuple in all colour tuples of image:
    # Transformation values for a nd b channels.
    ratio = tuple.L / estimated_illuminant.L
    shift_a = ratio * estimated_illuminant.a
    shift_b = ratio * estimated_illuminant.b

    # Adjust shift.
    tuple.a -= shift_a
    tuple.b -= shift_b

    # Adjust scale.
    tuple.L *= scale_L
```

Figure 4.6: Channel scaling transformation for the estimated illuminant of the $L^*a^*b^*$ modified colour constancy algorithms.

4.4 Implementation and Evaluation

The implementation has been accomplished completely using the Python programming language. It uses various packages and extensions that directly tie in native C libraries to yield good performances of the application, while maintaining a very high speed of implementation due to its ease. Specifically, the Python Imaging Library (PIL) is used for image reading and writing and NumPy for fast and comfortable image and matrix processing. For colour management tasks (conversion between sRGB and CIE LAB, Bradford transformation) the new bindings PyLittleCMS [41] to the long standing and widely used Little CMS [42, 43] library have been used. PyLittleCMS was particularly developed and extended for this purpose [44].

Distortions are introduced due to the fact that real world images are often exposed in a way that pixels are "clipped." Clipping is, when one or more of the colour bands in the images are set to an extreme value (e. g. in RGB encoding usually 0 or 255). This happens mostly when the particular colour value exceeds the dynamic range of the encoding, as it often happens in very bright specular reflections. The restrictions in these colour tuples due to clipping therefore usually do not reflect the true shade of the illuminant's colour.

The results and quality of colour constancy algorithms can be quite hard to judge, as it is often not easy to obtain a true canonical image for a direct comparison. A common test for its effectiveness however is to use *colour indexing* [1, 2] as first described by Swain and Ballard [3]. The task in colour indexing is to identify an object in an image by its colour distribution. The colour distribution obtained through histograms of the colour tuples in the image are compared to the histograms of a *model database* through *histogram intersection* [3]. For the sake of best comparability, the implementation did not use any of

the (improved) later modifications, but followed the basic originally described scheme (with a minor modification as described in Sect. 4.4.2).

4.4.1 Implementation of Colour Constancy

As already mentioned in Sect. 4.2, the colour constancy algorithms have been implemented according to Ebner's pseudo code in [1]. For this purpose, the RGB colour space was converted to floating point values between 0.0 and 1.0, and the channels were linearised (removal of the gamma correction). The $L^*a^*b^*$ space was expressed in floating point values, with the luminance component L^* ranging between 0.0 and a maximum of 100.0, whereas the chromaticity components a^* and b^* are ranging between -128.0 and 128.0.

The colour constancy algorithm descriptions directly do not mention the aspect of handling of clipped pixels. The implementation of these algorithms therefore provides handling of all image pixels as well as selectively operating only on the un-clipped pixels to avoid any bias in determining of the white point. Applications discarding the clipped pixels should yield better and more reproducible results. For comparison, all computations in Sect. 4.6.2 have been performed using both methods: including and excluding clipped pixels. The visual evaluation in Sect. 4.6.1 features computations on un-clipped pixels only.

None of the parameters for the cutoff points have been adapted or "tuned" towards optimal colour constancy performance. For the sake of keeping the implementations as straight and comparable as possible, they have remained at the values as described by Ebner [1].

4.4.2 Implementation of Colour Indexing

The implementation of colour indexing is applied to the chromaticity plane of the colour tuples only, rather than to the full colour information as described by Swain and Ballard [3]. This is done for two reasons: The luminance of images is very variable, and largely influenced by shadows. Therefore, neglecting luminance should improve robustness over more extreme exposure differences. Secondly, Barnard's Ph. D. thesis [2] is a frequently acknowledged source of colour research that is also using colour indexing on chromaticity values only. It is frequently used for comparison, and Barnard's image set [2] is used in this research for qualitative and quantitative evaluation means as well.

For colour indexing on RGB based colour constancy algorithms, commonly rg chromaticity values are used. These are obtained by dividing the red and green channels by the sum of all three colour channels [2]. For $L^*a^*b^*$ based processing it is even easier, as just omitting the L^* component already results in a suitable chromaticity space, the a^*b^* plane. The two dimensional chromaticity plane of the colour space coverage is regularly sampled into 16×16 histogram bins. The lower and upper limits of the histogramming range have been

set to cover the full possible range of values, as provided by the image and the given library implementations (0.0–1.0 for RGB and -128.0–128.0 for $L^*a^*b^*$).

These histograms are directly used for the histogram intersection as in Swain and Ballard's original publication [3]. All images for colour indexing are of the same size, the individual objects appear roughly in the same size within the different images under the different illuminants, and they are taken in front of a black/neutral background. Therefore, for the evaluation runs no image alignment, scaling or other normalisation operation was required.

4.5 Experiments

For evaluation purposes we have run Barnard's complete image set [2] against the two colour constancy algorithms. Each algorithm was employed in four different variations: Each in an RGB and CIE LAB based algorithm, with the following chromatic adaptation based on the white point estimation performed in a channel scaling as well as a Bradford transformation. To prevent excessive pixel clipping, Barnard has strongly under-exposed the images, and encoded them in a 16 bit high dynamic range format. An alternative 8 bit format is also available, which we chose for our evaluation. This set is equally under-exposed, and pixel clipping does occur, although only in very small quantities. The image set has been obtained by illuminating the scenes purely with eleven different illuminants. Some of these illuminants were quite extreme due to a blue filter used together with some light sources.

Another set of images has undergone the same set of colour correction means. This set consists of "normal appearance" images under various conditions. These were all exposed to suit a human viewer for on-screen viewing. The images range from old scanned photographs, over some colourful sample scenes (properly exposed and white balanced by the camera), to images taken under water with a stronger blue-green cast.

For a quantitative evaluation through colour indexing, Barnard provides an image set of 20 objects [2] taken under the same mentioned eleven illuminants. As in a typical colour indexing sample set, for each illuminant the pictured object has been shifted slightly or rotated within the scene. Again, the before mentioned eight different methods for colour correction have been applied to each image before obtaining their histograms in the respective colour's chromaticity spaces. For reference, colour indexing has been applied to un-corrected images as well, and the pure 1-in-20 random chance to choose the correct image from the set of objects has been calculated.

4.6 Results

The results have been evaluated both visually and quantitatively according to the experiments conducted (see previous Sect.). Additionally to colour indexing, a chromaticity gamut evaluation has been performed for the quantitiative analysis.

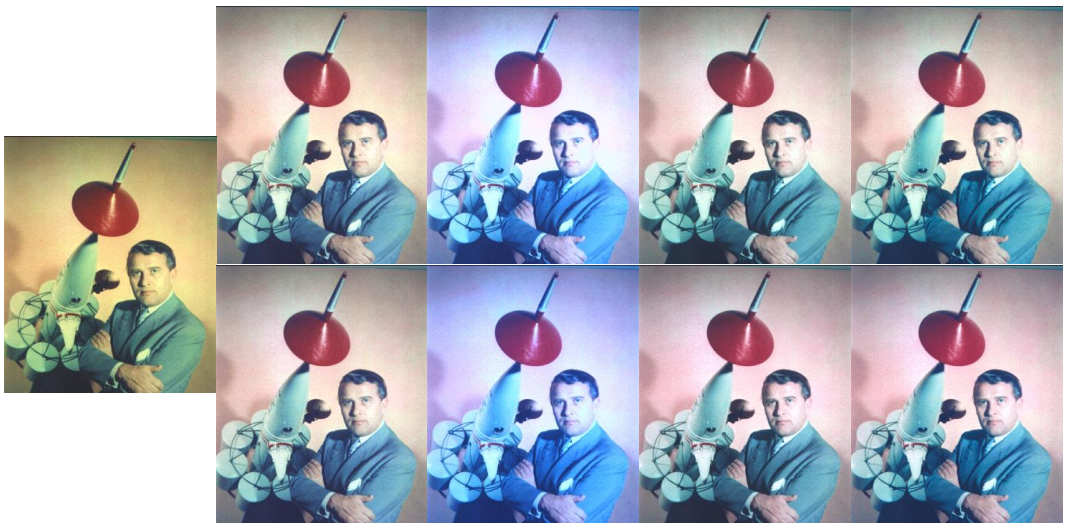

Figure 4.7: (left) Original 1950's picture of Wernher von Braun. (top row) Colour corrected according to Grey World Assumption. (bottom row) Colour corrected according to White Patch Retinex algorithm. Chromatic transformations (left to right): RGB channel scaling, RGB with Bradford transformation, $L^*a^*b^*$ with channel scaling, $L^*a^*b^*$ with Bradford transformation.

4.6.1 Visual Evaluation Results

We do not need to discuss weaknesses and strengths of colour constancy algorithms, that has been done multiple times in detail already [1,2]. We will focus particularly on differences between RGB and CIE LAB based processing on the one hand, as well as the chromatic adaptation strategy on the other hand.

First of all we have visually evaluated images of "natural appearance" that could be part of most people's photo collection. (Almost) already properly white balanced images did not show any significant changes beyond the characteristics of the particular colour constancy algorithm that are worth discussing. Therefore, results are not presented or discussed here any further. However, images with a visible and distracting colour cast do provide some potential for comparison. For this purpose we have used a 1950's photo of Wernher von Braun (Fig. 4.7) and an underwater picture (Fig. 4.8).

Both algorithms handle the removal of the yellowish "vintage cast" in Fig. 4.7 quite well. It is quite clearly visible, that a Bradford transformation applied with the RGB based estimation of the white point fails badly (second photo in the two rows). Both channel scaling as well as a Bradford transformation in $L^*a^*b^*$ work quite well and yield similar results.

Using an underwater scene (Fig. 4.8) with a strong blue-green cast from the dominant light underwater reveals some stronger differences. Colour constancy using the Grey World Assumption (top row) shows partially extreme colour shifts in the dominant colour of the

Figure 4.8: (left) Original under water photograph. (top row) Colour corrected according to Grey World Assumption. (bottom row) Colour corrected according to White Patch Retinex algorithm. Chromatic transformations (left to right): RGB channel scaling, RGB with Bradford transformation, $L^*a^*b^*$ with channel scaling, $L^*a^*b^*$ with Bradford transformation.

water when processed on RGB colours (left two images in top row). With both colour constancy algorithms on RGB colours (left two images in both rows) a strong unnatural over-saturation of skin colour can be observed, which is not obvious in the $L^*a^*b^*$ based processing. The human visual system has evolved to be able to perceive certain colour differences on a much more finely grained scale than others. We are able to determine the state of health of humans partly through noticing changes in skin colour. Additionally, subtle changes in less saturated colours are much more obvious to the human eye. The CIE LAB colour model has been designed to reflect the linearity of human colour perception. Therefore, it is much more capable of accommodating transformations suitable for subtle changes in "important colours," such as neutral greys and pale colours (as skin colour), which are important to humans' every day living conditions and survival.

Next, the colour constancy algorithms have been applied along with their modifications to images of the Barnard sets [2]. These scenes are purposefully under-exposed to prevent pixel clipping in the high intensity areas of the images. For the purpose of displaying, the luminance of the original image has been increased (on the very left of Fig. 4.9 and Fig. 4.10). The images after the application of the colour constancy algorithms have not been altered, their luminance has been boosted by the chromatic adaptation. Barnard has also used some quite extremely coloured illuminants to provide a challenge for the colour constancy implementations.

The ball in Fig. 4.9 has been exposed with an illuminant that is close to daylight, whereas the camera's white point is set towards accommodating an incandescent light source, result-

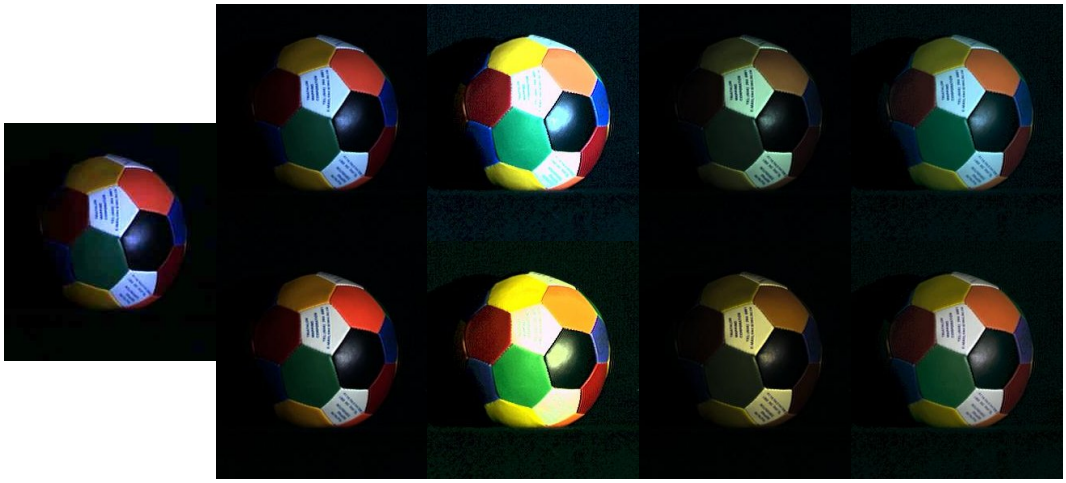

Figure 4.9: (left) Ball under Solux 4700 illuminant (lightness increased for displaying, cropped images). (top row) Colour corrected according to Grey World Assumption. (bottom row) Colour corrected according to White Patch Retinex algorithm. Chromatic transformations (left to right): RGB channel scaling, RGB with Bradford transformation, $L^*a^*b^*$ with channel scaling, $L^*a^*b^*$ with Bradford transformation.

ing in a slight bluish cast.

Both implementations (Grey World Assumption and White Point Retinex) yield reasonably good results when using channel scaling for RGB (very left of each row) as well as the Bradford transformation for $L^*a^*b^*$ (very right of each row). This is not very surprising, as *von Kries* type transformation are prescribed for white point adaptations in the CIE based colour system. Comparing the non-neutral colours of these two with each other, one can see that the colours in $L^*a^*b^*$ based transformations are not as saturated as in RGB based processing. The *von Kries* type transformations preserve colour information relative to each other without trying to introduce colour space distortions. They preserve the relative saturation of the under-exposed original image. Increasing the lightness of the original image for publication was performed in an image manipulation tool, which operates by scaling on RGB channels, and therefore increases colour saturation as well.

It is quite obvious, that the Bradford transformation applied to RGB based processing does not yield any reasonable results. As well as the simple channel scaling in $L^*a^*b^*$ colour space seems to over-compensate on neutral colours (such as white), and under-compensate on more saturated colours (red, blue).

The next sample (Fig. 4.10) shows the GretagMacbeth ColorChecker presenting the well known colour patches. It has been illuminated with an incandescent lamp equipped with a blue filter. This blue filter creates a rather large shift in the white point that needs to be compensated for. Again, the chromatic adaptations applied to their suitable colour spaces (channel scaling for RGB and Bradford transformation for $L^*a^*b^*$) perform well, and again the colours of the $L^*a^*b^*$ compensated space are slightly undersaturated. Some illuminants

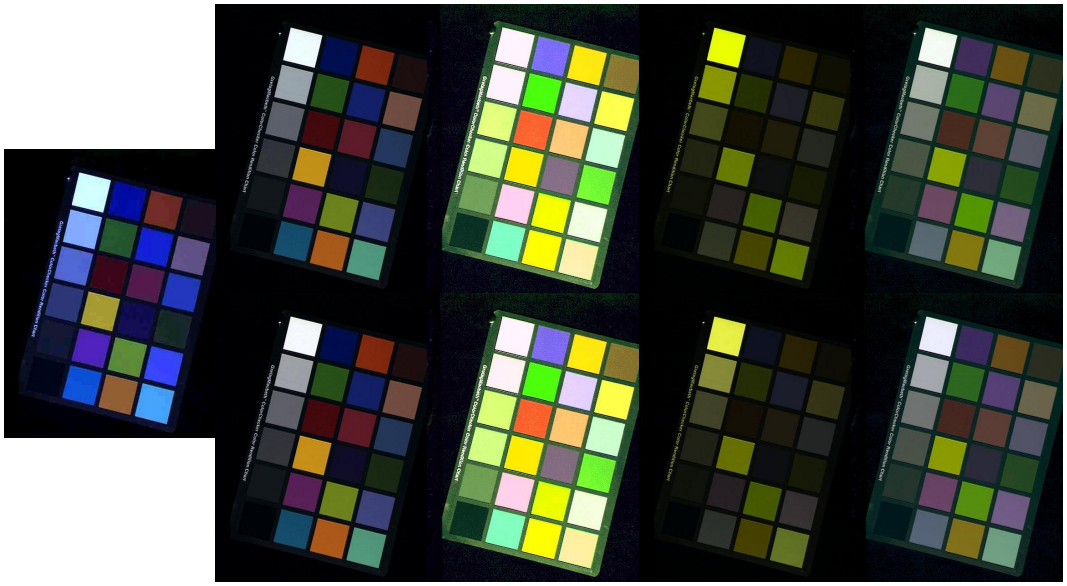

Figure 4.10: (left) GretagMacbeth ColorChecker under Sylvana 50MR16Q light with Roscolux 3202 blue filter (lightness increased for displaying). (top row) Colour corrected according to Grey World Assumption. (bottom row) Colour corrected according to White Patch Retinex algorithm. Chromatic transformations (left to right): RGB channel scaling, RGB with Bradford transformation, $L^*a^*b^*$ with channel scaling, $L^*a^*b^*$ with Bradford transformation.

in the Barnard image set are *very* far away from natural daylight or other common and more neutral illuminants. The results are all comparably weaker on these extremes. But *von Kries* type CATs are known to work best on small changes, and their quality degrades with the degree of difference. The other two compensations work noticeably worse and are not capable of compensating for the under-exposed image (Bradford transformation on RGB), and for the strongly shifted white point (channel scaling on $L^*a^*b^*$). It is obvious that this effect is more strongly visible in this image set than in the one from Fig. 4.9.

The question that arises now is this: Is it justifiable, that the colour constancy algorithms are boosting colour saturation? One may argue that the RGB colour corrected images' boost in saturation looks better (Figs. 4.10 and 4.9) compared with the source images. But the colour constancy algorithms introduce a target saturation which is not to be found in the source images. In contrast, the $L^*a^*b^*$ based colour constancy algorithms preserve better the subtle relationships of the colours, particularly in the less saturated colour ranges (see Fig. 4.8). The whole basic idea in chromatic adaptation transformations following the *von Kries* concept is to adapt to different white points without introducing any significant colour space distortions. This seems to work quite well when processing on CIE colour spaces such as CIE LAB. After all, the CIE prescribes the use of *von Kries* type CATs. Mostly accepted best practise is to use the Bradford transformation. Furthermore, it seems like single dominating colours (see Fig. 4.8) have much less of a strong negative effect on colour

constancy algorithms applied to images in the $L^*a^*b^*$ colour space.

We can simply summarise for the series of colour compensations computed, that one is best advised to use an algorithm for chromatic adaptation that was "made for" the underlying colour space. The (original) colour constancy algorithms have been developed using RGB colour spaces, and they rely on channel scaling for the compensation. CIE LAB is an opponent colour space, in which only one component (L^*) contains intensity information, whereas the other two purely contain chromaticity. *Von Kries* type CATs have long been proven to be reliable on the CIE based colour spaces, and therefore should be applied on these as well.

4.6.2 Quantitative Evaluation

To achieve a better quantifiable evaluation, different variations of colour constancy algorithms have been applied to the problem of colour indexing. The methodology follows closely the one performed by Barnard [2], also using his image set of objects for this purpose (20 objects under eleven different illuminants). Colour constancy algorithms were applied on this image set to form the model database (a set of all objects under one illuminant), and the same colour correction was used on an image under an arbitrary illuminant of the set. Colour indexing then tries to identify it within the model database.

Some initially unexpected results in the behaviour of $L^*a^*b^*$ based colour indexing were discovered. To analyse and explain this, chromaticity gamuts (Sect. 4.6.2) for the different colour spaces are used for comparison.

Colour Indexing

In Table 4.1 the full images (including all clipped pixels) have been used for colour indexing. First level scores (column "Score (1)") is the percentage of correct matches. In the next column "Score (1)" is increased by 1/2 of a similar score for second place ranked matches. Then "Score (3)" weighs the third rank match with an additional weight of 1/3 onto "Score (2)" (see [2]). Each score has been computed out of a statistical base of 2420 samples (20 images under 11 illuminants against a databases of 11 illuminants each). Due to this large sample size the statistical error is equal or less than 0.01. In one of 20 cases even by random a correct match would be found, as well as in every one case of 11 (when illuminant of the object and of the model database are the same) an ideal match can be found (due to identical colour constancy treatment).

The values in tables 4.1 and 4.2 are each the result of these 2420 computations undertaken for each of the eight varieties of colour constancy algorithms, two uncorrected colour indexing runs and the random selection. For comparison reasons the "1 in 20" random chance has been added to the table, along with no colour constancy algorithm to be applied ("None"). These are to indicate the boundaries and to judge thresholds for an improvement gained through effective colour corrections.

Table 4.1: Colour correction on un-clipped pixels only, colour indexing applied to all pixels. Best scores for each algorithm/colour space emphasised.

Algorithm	**Score (1)**	**Score (2)**	**Score (3)**
"1 in 20" random chance	0.05	0.08	0.09
None, RGB	0.39	0.45	0.48
None, $L^*a^*b^*$	0.25	0.31	0.33
White Patch Retinex, RGB, channel scaling	**0.62**	**0.68**	**0.70**
White Patch Retinex, RGB, Bradford transform	0.33	0.39	0.42
White Patch Retinex, $L^*a^*b^*$, channel scaling	0.24	0.30	0.33
White Patch Retinex, $L^*a^*b^*$, Bradford transform	**0.38**	**0.45**	**0.48**
Grey World, RGB, channel scaling	**0.60**	**0.66**	**0.68**
Grey World, RGB, Bradford transform	0.32	0.39	0.41
Grey World, $L^*a^*b^*$, channel scaling	0.27	0.32	0.35
Grey World, $L^*a^*b^*$, Bradford transform	**0.38**	**0.45**	**0.47**

The table's values indicate a sufficient quantitative indication that channel scaling is in comparison not suitable for $L^*a^*b^*$ based processing, as well as the Bradford transformation is not suitable for RGB based processing. It seems striking that $L^*a^*b^*$ based colour indexing scores noticeably worse than the one based on RGB colours (its scores are comparable to no colour correction at all). Particularly, as the visual evaluation of the processed image sets seemed quite promising. Reasons for this phenomenon will be given later in Sect. 4.6.2.

The scores in Table 4.2 have been computed with colour indexing applied to un-clipped pixels only, just as the colour constancy algorithms discards clipped pixels. In comparison with Table 4.1, it is obvious that pixel clipping causes a strong negative influence on image characterisation through colour indexing. Scores for both algorithm types – when used with a combination of the "proper" chromatic adaptation for the used colour space – are consistently much higher.

Chromaticity Gamuts

Colour indexing discriminates objects in images by their chromaticity distribution through the use of histograms. Good results can only be achieved through utilising a significant range of the available histogram bins. The histogram bin ranges were determined as to cover the whole "addressable space" of the used colour encoding. As the sRGB colour space is smaller and fits well inside the standard CIE LAB colour space, it seems logical, that the histogram bin utilisation in $L^*a^*b^*$ is worse than in RGB, resulting in lower scores in colour indexing (see Sect. 4.6.2).

To get an idea for the magnitude of this effect we have determined statistically the fraction of histogram bin usage. In average in rg-chromaticity 25 % (std. dev. 6 %) for all, and 13 % (std. dev. 5 %) for un-clipped pixels, of the available histogram bins were used.

Table 4.2: Colour correction and colour indexing applied to un-clipped pixels only. Best scores for each algorithm/colour space emphasised.

Algorithm	**Score (1)**	**Score (2)**	**Score (3)**
"1 in 20" random chance	0.05	0.08	0.09
None, RGB	0.35	0.39	0.41
None, $L^*a^*b^*$	0.29	0.34	0.37
White Patch Retinex, RGB, channel scaling	**0.80**	**0.85**	**0.86**
White Patch Retinex, RGB, Bradford transform	0.23	0.27	0.29
White Patch Retinex, $L^*a^*b^*$, channel scaling	0.23	0.27	0.29
White Patch Retinex, $L^*a^*b^*$, Bradford transform	**0.51**	**0.57**	**0.60**
Grey World, RGB, channel scaling	**0.80**	**0.83**	**0.85**
Grey World, RGB, Bradford transform	0.25	0.29	0.31
Grey World, $L^*a^*b^*$, channel scaling	0.26	0.30	0.32
Grey World, $L^*a^*b^*$, Bradford transform	**0.46**	**0.51**	**0.52**

In a^*b^*-chromaticity however, only 9 % (std. dev. 4%) and respectively 8 % (std. dev. 4%), were used. That means that only about 1/2 to 1/3 of all histogram bins contained data for CIE LAB based colour indexing, in comparison to the RGB based one. This seems to be the obvious reason for the decline in colour indexing scores when moving from rg-chromaticity to a^*b^*-chromaticity.

To analyse the *chromaticity gamut* more directly, we have computed average histograms over *all* colour constancy adapted images of the set. The chromaticity gamut can be understood as a "foot print" (or projection) of all chromaticities occurring in the scenes. The histograms in Fig. 4.11 are averaged over 880 image histograms (20 objects under 11 illuminants colour corrected with four variants of colour constancy algorithms).

Fig. 4.11 highlights these gamuts in black on the chromaticity plane, discretised by the histogram bins. In colour, above every one of these bins, a column indicating the relative frequency of occurrence is depicted. The sum over *all* images indicates only a small increase of the rg gamut (bottom row) over the a^*b^* gamut (top row). However, the colour distribution within the footprint is much more narrowly condensed over the neutral centre in comparison to the rg-chromaticities, yielding only small effective information content usable for discrimination in colour indexing. On a side note, due to its triangular shape the rg-chromaticity space suffers of only being capable of utilising maximally 1/2 of the available histogram space.

It is interesting to see the effect of pixel clipping in this figure. Most clipped pixels are close to neutral in their chromaticity, leading to a dramatic increase in the centre columns of a^*b^*-chromaticity. These suppress subtle differences in more saturated colours, which carry important information for colour indexing. The same effect can be identified for rg-chromaticities. But beyond that, many of the clipped pixels (with less than three clipped channels), form strongly distinguishable peaks on the usually only very little populated

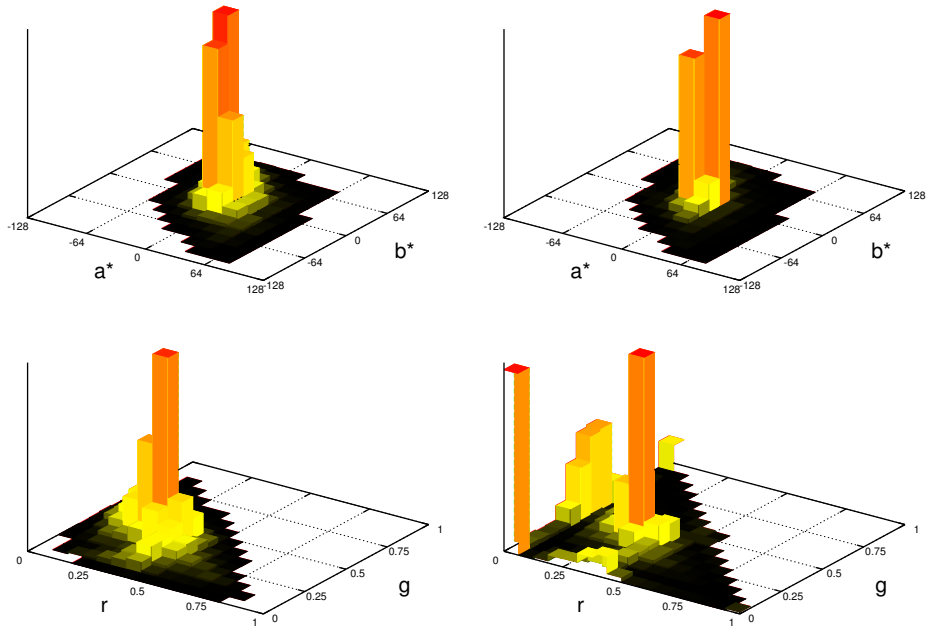

Figure 4.11: Chromaticity gamuts and colour distribution averaged over all images. The top row depicts histograms in a^*b^*-chromaticity, with the same in rg-chromaticity below. The diagrams on the left show histogramming applied to un-clipped pixels only, whereas the right includes all clipped pixel values.

perimeter of the rg-chromaticity space. Particularly these peaks in the extreme boundaries degrade colour indexing quality the most in practical applications.

4.7 Conclusions on Colour Constancy

The goal of this chapter was to provide insights into the possibility of using the CIE based $L^*a^*b^*$ colour space, as frequently used analytically and in industrial quality sensitive environments, for the domain of colour constancy. For this purpose, two well-known colour constancy algorithms – the White Patch Retinex and the Grey World Assumption – have been implemented to work on an $L^*a^*b^*$ colour space. The implementation attempted to build on a 1:1 preservation of the defining ideas of these algorithms without introducing any distorting "improvements" for a comparison. These algorithms are based on an implicit white point estimation, which is used to correct the colour space. We have analysed which methods of chromatic adaptation transformations work for the $L^*a^*b^*$ colour spaces. The CIE prescribed *von Kries* type transformations – specifically the Bradford transformation – consistently yielded good results with the $L^*a^*b^*$ colour space. We showed results for successfully applied $L^*a^*b^*$ based colour constancy on several image sets, as well as validated the potential for usability of $L^*a^*b^*$ based colour constancy for the purpose of object

recognition through colour indexing. Finally, we have discussed current weaknesses of the current implementation with indications on how to overcome them. Furthermore, details for an implementation regarding the selection of histogram ranges, as well as discarding clipped pixels have been highlighted.

Overall, the results have shown that the application of colour correction through the modified colour constancy algorithms on $L^*a^*b^*$ colour spaces does work. Results on whether the results are superior or inferior to the original RGB based implementations are still inconclusive. For a more quantitative comparison of colour constancy on $L^*a^*b^*$ vs. RGB colour spaces the colour indexing would have to be applied consistently to *one* colour space, regardless on what colour space the colour correction operates.

Future work in this area should include further comparative analyses with image sets that were not as under-exposed, or more "normally" exposed to confirm the theory, that the Bradford transformation preserves the saturation of under-exposed imagery. Furthermore, the behaviour of the colour constancy algorithms with respect to used parameter sets (for different colour spaces) needs to be analysed, to improve the effectiveness of the implementation. Lastly, it should be possible to significantly improve the colour indexing performance by selecting histogramming ranges to fit more tightly to the encountered chromaticity gamut.

Chapter 5

Colour Management

> *"If one says 'red' (the name of a colour) and there are 50 people listening, it can be expected that there will be 50 reds in their minds. And one can be sure that all these reds will be very different."*
> – Josef Albers, *Interaction of Colour,* (1963)

The purpose of colour management is to achieve most constant colour reproduction of images acquired from arbitrary input devices, onto other arbitrary output devices. In image reproduction workflows, this is not a simple source to destination mapping, but includes various steps of intermediate image manipulation (editing) as well as output simulation (proofing) on alternative output devices (e. g. printing a sample on a comparably cheap ink jet printer to simulate the appearance for the industrial production printing press). The industrial demand for these steps was quite large due to too many "rule of thumb" manipulations in the processing chain. The International Color Consortium (ICC) was funded in 1992 under the lead of FOGRA (German Graphic Technology Research Association). The ICC was to define universal standards, tools and applications for characterising different devices through *ICC profiles* and the transformations between the colour representations using these profiles through *Colour Management Modules (CMMs)* [45–48].

5.1 Introduction

A high similarity between the colours of images between input and output devices is called "colour fidelity." To achieve colour fidelity, Colour Management Systems (CMS) are used, however it is almost impossible to yield ideally correct results.

Colour management systems use device independent descriptions (device profiles) and device independent exchange colour spaces (profile connection spaces, PCS). The CMS uses the CMM to convert between these device dependent and device independent colour descriptions to enable colour fidelity between input and output devices. The numeric representation

through colour descriptors in the input/output device spaces (often an RGB or CMYK tuple) and the PCS (a CIE $L^*a^*b^*$ or XYZ tuple) is different, but should match up visually with good accuracy. The PCS colour representation can therefore be seen as a canonical representation of the colour itself, whereas the device dependent representations are *only* valid for a particular device under specific conditions. The reason for this is, that colour descriptions are device specific, and appearances can be very different. The CMS translates between the different interpretations.

In many cases, the colour workflow is not short and linear just between an input and an output device. For those simple cases one single "device link" transformation could be used. However, with a variety of input as well as output devices the number of potential links increases dramatically. If one is dealing with the common case of having to manipulate imagery interactively on a work station, which is to be printed later, already two different types of output devices (the screen as well as the printer) have to be handled and managed to express the same colour [45]. If the printing is finally going to happen on a printing press, often a preview (or more accurately a proof) is printed on a smaller commodity printer (e. g. ink-jet printer) to simulate the final product. In this case, the different printing characteristic of the press has to be simulated by the CMS for the smaller printer.

Several factors need to be considered in the ability to match colours between different devices. These all may induce shortcomings in accuracy of the colour translation, and will in many cases cause limitations in different regions of the colour space for different transformations.

- Concepts of colour image perception and generation: additive colour systems (for screens) vs. subtractive colour systems (for printing);
- Differently primary colours;
- Different media properties (whiteness of the paper or media, blackness of the CRT or flat screen, illuminant white point).

Ideally one would work only with a "true" spectral/physical representations of colour. This however, is in practice not possible, as it would involve too much effort, and for practical purpose current devices are using mostly three (or four for printing) primaries. Devices with higher spectral resolution tend to be very specialised, expensive and not much commonly available. Colour management tries to decouple the handling of colour from spectral distributions towards working on colour tuples only, as they are commonly used for input and output devices, but handle transitions between them in a scientifically sound way.

ICC based colour management has been designed with the whole process in mind, as it is required by the graphic arts industry. Therefore it is laid out to aid in minimising the progression of colour errors through an imaging workflow, which covers capturing, modification, analysis, output, proofing and potentially further steps.

The ICC describes Colour Management in their section "Terms and Definitions 4.8: Colour Management (digital imaging)" of the ICC specification [49] like this:

> "communication of the associated data required for unambiguous interpretation of colour content data, and application of colour data conversions, as required, to produce the intended reproductions
>
> NOTE 1: Colour content may consist of text, line art, graphics, and pictorial images, in raster or vector form, all of which may be colour managed.
>
> NOTE 2: Colour management considers the characteristics of input and output devices in determining colour data conversions for these devices."

In the following of this chapter, we are discussing different aspects of colour management. Sect. 5.2 starts off with ways to handle the characteristics of the hardware in the form of input and output devices. In Sect. 5.3, colour profiles, as derived from the hardware, are covered, with the process of the profile creation in Sect. 5.4. These profiles then can be used to transform colour representations as outlined in Sect. 5.5. The device independent colour is defined relative to the standard illuminant, and a conversion to an alternative illuminant is described in Sect. 5.6. Different freely available implementations for CMS' are listed in Sect. 5.7. Finally, in Sect. 5.8, we are concluding of how colour management and its properties are useful for the scope of this research.

5.2 Hardware

As every human perceives colour individually different, also devices do so. Particularly different classes of devices understand colour differently, as they might use different colour spaces to represent the colour, use different sensors, potentially with differences in the primaries, due to variations in production over time or age of the device, etc. This can be seen both from the side of perceiving as well as reproducing colour.

To describe the behaviour of the different devices and device types, their colour response has got to be measured and compared to a reference that is known in a standard colour space. This reference can be established for input devices by measuring the test target's colour (see Sect. 5.4) with a precise spectrophotometer prior to capturing it with the hardware (e. g. a scanner or camera). For output devices a test target pattern is transferred to the output device, and the output is measured afterwards using a spectrophotometer or colourimeter in a standard colour space.

To capture and control the behaviour of the hardware, two basic processes can be used: *Characterisation* and *calibration*. Most commonly one is dealing with characterisation, in which the device is used as it is, and the relationship of canonical colour to device colour is analysed. This describes the colour imaging characteristics of the device. Calibration

is often used synonymously to characterisation. But strictly speaking, a calibration may include adjustments to the device itself.

To circumvent the need of full characterisation or calibration many devices are produced to a certain common norm. One of these is for example the very common sRGB standard colour space created cooperatively by Hewlett-Packard and Microsoft in 1996 for use on monitors, printers and the Internet. It has been designed with the goal of being a common denominator to avoid colour mismatches in day-to-day image representation. As it is assumed to work on a wide variety of different classes of devices, it has been chosen to have a comparably small gamut of colours it can represent, which mostly work on a wide variety of imaging devices. But many devices' colour imaging capabilities go beyond the specification of sRGB, so it is not necessarily good to rely on it for all purposes. Furthermore, manufacturers are inherently interested in selling their hardware. Therefore, it is quite common that factory settings are aiming to improve brightness and brilliance to be as good as or better compared to competing products. Manufacturer settings therefore are tuned for marketing, not colour fidelity.

In imaging, therefore, one can gain significant improvements in colour accuracy by using colour management with characterised (and possibly calibrated) hardware. Even devices from a single manufacturer, or within the same product line, can vary quite significantly [47].

5.3 Colour Profiles

Colour profiles – in the form of ICC profiles – form the basis to convert colour representations through ICC standardised colour management. They describe a mapping of device colours (colour response or display colour) to canonical standard colours. All colour profiles map some specific colour space to a profile connection space (PCS). This PCS is either CIE XYZ or CIE LAB, which are different – but equivalent – representations of the same colour space. The specific colour spaces are usually either *device spaces* or *working spaces*.

Device spaces are specific to a particular device's colour representation. These could be for a scanner or digital camera often in RGB or YUV, and for a printer for example CMYK or RGB. The profiles are *only* valid for the specific conditions they were created for. These conditions on the one hand consist of device settings, and on the other hand further boundary conditions. For cameras, these are specific light conditions, for a printer the media (paper, film, ink, toner, etc.) have to be set as constant for a given profile. The working conditions of hardware may also need to be kept within certain boundaries (e. g. by ensuring a sufficient warm-up phase for devices).

Working spaces are colour spaces that are used for processing and editing of colour imagery. These can be spaces such as sRGB, Adobe RGB or ProPhoto. As these working spaces are completely fixed and defined, they will be equal in all cases, and predefined, constant profiles may be used. In working with colour imagery, one has to keep in mind that a

conversion to a working space may induce *clipping* (limiting intense colour values to minimum/maximum intensity), if the gamut of the working space is too small; or *posterisation* (banding, coarse colour sampling), if the working space gamut is too large.

Lastly, some other ICC profiles exist: Some are used to represent the canonical CIE colours for conversions to the PCS; Others may for example describe a direct link between two distinct devices.

Profiles can be represented separately to be used by the CMS, for example in the file system, a data base or directly in memory. Within certain image formats themselves they may also be embedded (such as TIFF, JPEG, PNG, EPS, PDF, and SVG). This may be particularly convenient for images to store colour information in their original format without degradation of quality, by embedding the capturing device's characterisation or the quantisation of the chosen working space [45, 48].

5.4 Profile Creation

The profile creation is based on a colour measurement. Commonly, spectrophotometers or colourimeters are used for this purposes. For output devices (e. g. monitor, printer), the known device colours will be measured in the output by the measuring device in the canonical colour space. Inversely, for input devices (e. g. camera, scanner), the known canonical colours will be captured by the input device in the device colour space. Using these known device and canonical colour pairings, a mapping or transformation between these colour spaces is derived. Also derived from this is the *gamut* – the totality of all representable colours of a device. Depending on the type of a device, these profiles are generated in different ways.

Many device colour spaces, especially those with 8 bit per channel encoding, use gamma correction. To achieve good quality profiles, as a first step a set of linearisations is performed between the device colour and the rest of the transformations to "undo" the gamma correction (using the "A" curves, see Fig. 7.1 in Chap. 7). The device colour data is now present in a more uniformly distributed form, which largely increases the quality of the achievable transformation.

The enclosing volume of the scattered measurement data is used to define the gamut of the device. From this data, a transformation between the linearised device colour and the device independent profile connection space data must be determined. This step is the core task of *profiling*. Profiling may be used to generate two basic types of profiles. These are on the one hand the mathematically very straight "shaper/matrix" profiles. They include a set of shaper curves ("M" curves), a linear transformation matrix to yield a best fit, and optionally post-linearisation curves ("B" curves). These profiles are only available for devices using exactly three colour channels. For a more detailed description see Chap. 7. On the other hand, a profile may contain a multi-dimensional colour lookup table (CLUT). CLUT based profiles are much more complicated to create, but their capabilities and matching accuracy

by far exceeds the shaper/matrix profiles' for real world devices. Computations to determine a suitable CLUT usually involve a complex mathematical process using multi-dimensional spline fitting, interpolation, smoothing, etc. Additionally, often testing, personal judgement and parameter tuning to obtain profiles performing as good as possible are also involved. Again, see Chap. 7 for a much more detailed description and implementation notes.

Due to "aging effects," hardware devices' colour accuracy may undergo a certain drift, so that profiling is not a one-off procedure for a specific device (for given media or illumination conditions). Regular re-profiling of these devices is usually necessary in these cases. Also, manufacturer provided profiles are only valid to a certain extent for a certain device series under conditions that are not too well documented by the manufacturer. It is usually advisable to perform individual characterisations to generate ICC profiles for specific given conditions.

The obtained ICC profile, as a result of the hardware characterisation, is either an input or an output profile. Input profiles commonly only demand translation capabilities from device colour space to the PCS. As a minimum output profiles must provide a PCS to device colour space transformation. In practice however, especially for the graphic arts industry, it is often necessary to also include a transformation in reverse. It is used to "compute backwards" the appearance of a print, and emulate its look on a different output device. This process is called "proofing": For screens, this is a "soft proof," and for other physical output media it is a "hard proof" [45, 47, 48].

5.5 Colour Translation

In most practical applications and workflows, a user is not interested in the colour information on the level of the PCS. ICC profiles only describe the characterisation of the devices. As stated above, a colour management module (CMM) is responsible for conducting computationally the transformations described by the ICC profiles on the colour information. Typical, simple colour workflows would therefore have a form as illustrated in Fig. 5.1. The intermediate PCS can be considered as a *"lingua franca"* one can translate every input device colour encoding into, and from which one can convert into every other output device colour encoding. Captured imagery from a profiled camera could be, for example, displayed (comparatively) true to original colour on a profiled display. If the image is to be used on various unknown output devices, for example to be distributed on the Internet, a generic sRGB output profile could be used, as most computers know how to handle sRGB encoded colour information at least reasonably well.

For this reason, typical workflows consist of two colour space transformations, which the CMS conducts through the CMM: A transformation to the PCS, and then from PCS to some output colour space. As this is a very common task, modern CMMs know how to smelter this multi step process into a single transformation operation to gain computational efficiency and reduce an accumulation of repeated rounding errors [42]. If one is interested in

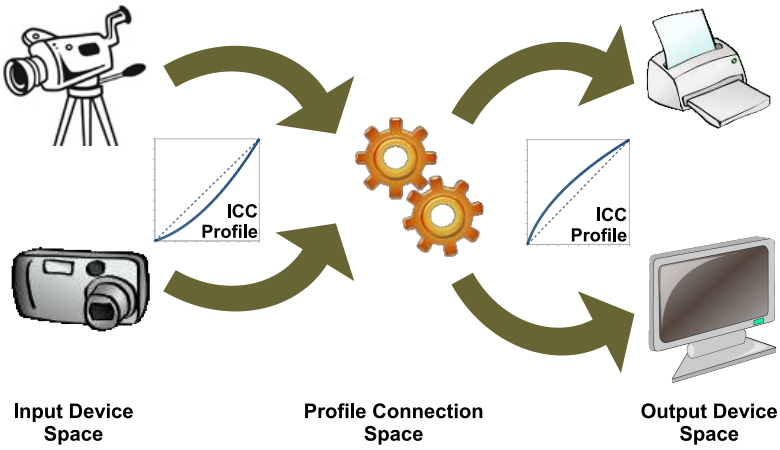

Figure 5.1: Common simple colour transformation workflows to retain colour fidelity on the path from an arbitrary input device towards another arbitrary output device.

a CIE space based colour representation, as it is also used for the PCS, a generic CIE LAB or CIE XYZ profile is used for one of the two ICC profiles in the transformation.

As mentioned previously, different devices have a different gamut. Colours representable on one device may be physically impossible to render on another device. The process of mapping representations of out-of-gamut colours into the gamut of a destination space is called *gamut mapping*. The CMM therefore needs to have information on how to treat these out-of-gamut colours in a translation. These may be very different for different purposes. Natural image representations are supposed to still look natural after the transformation, whereas in business diagrams, the intent might be to retain a maximum possible brilliance of the colours for the respective different output devices. These rules are specified through a so called *rendering intent*. The output profiles need to be able to support this particular desired rendering intent, by containing the necessary information that the CMM can use. The two *"colourimetric"* intents preserve the colourimetry of in-gamut colors at the expense of out-of-gamut colours. Mappings of out-of-gamut colours is vendor-specific. *"Perceptual"* and *"saturation"* rendering intents modify colourimetric values to account for different devices, media and viewing conditions [48].

Graeme Gill states on the Argyll CMS mailing list the process of the application of table lookup elements within an ICC V2 profile transformation[1] when used with Argyll. The process can be seen as principally identical for virtually all CMMs available:

> "First each channel goes through the per channel input tables, then through the multi-dimensional colour LUT, then each channel goes through the per channel output table. In the above example *[not given]*, the per channel curves have 512 entries, the multi-dimensional colour lookup has $9 \times 9 \times 9$ entries. For the A2B

[1] http://www.freelists.org/post/argyllcms/Printer-Profiling-Patch-Count-\and-Profiling-Generation-Quality,2

table, the input is RGB and output is PCS (usually $L^*a^*b^*$). For the B2A table the input is PCS and the output is RGB.

Typically, linear interpolation is used between entries, and Argyll profiles will all be using 16 bit entry values. For the multi-dimensional lookup there are many possible ways of interpolating within a cell. One is to use multi-linear interpolation (which ironically isn't actually linear for more than one dimension), or to break the cell up into smaller geometric pieces such as simplexes, which is often faster and uses linear interpolation.

How these pieces interact is somewhat up to the profile maker. The input tables have the effect of both transforming the space in which the CLUT interpolation is performed, and also changing the position of the entries. Juggled well, the per channel curves can minimise the errors introduced by the relatively low resolution of the CLUT and the interpolation within each cell."

5.6 Illuminant Adaptation

After translating a device dependent colour description into the device independent PCS, the colour is represented relative to the CIE D_{50} illuminant (standardised daylight equivalent to a black body radiator of 5003 K colour temperature). The colour representation can be converted to alternative white points by means of a chromatic adaptation transformation (CAT). Most commonly a linear Bradford transformation is used for this purpose. A full description of this process is outlined in Sect. 2.3.6.

5.7 Implementations

A wide variety of different CMMs has been implemented and is available. With the variety of vendors also a variety of usage scenarios comes along. Some are operating system services, some are implemented as libraries that can be used by applications or other services, and some are end user applications themselves.

Some of the better known commercial CMMs are for example ColorSync (by Apple for Mac OS) and Adobe CMM. The first is implemented as an operating system service to various applications requesting them. It has also been ported (in the past, but dropped again since) to MS Windows. Adobe CMM is a module (library) offering colour management transformation services to applications or other libraries.

On the non-commercial side, a few open source implementations of CMMs are available, too. Currently, these are namely Little CMS [42, 43], Argyll CMS [50] and SampleICC [51] (a reference implementation by the ICC). Little CMS (or LCMS) is a library which is in use by most open source applications as well as closed source applications (due to its liberal license and mature implementation). Argyll CMS in contrast is a suite of tools to facilitate

colour management through various individual command line applications. Some other applications using it are calling these tools in a separate process. Argyll's big advantage is the excellent profiler included. It is available under the more restrictive GPL license as well on request from the author in a commercial license. SampleICC in contrast does not have (to our knowledge) any active user base. It is a collection of libraries and tools, but its main purpose is more an aid by the ICC as a reference to other implementations. Its existence is more academic to study the internals of a CMM, and the code base is not very actively maintained.

As mentioned earlier in this chapter, the CMM is the functional core component of colour management. It can either be used directly by colour management aware applications, or it can be invoked by a colour management system (CMS). In many cases, an integrated colour management system is provisioned through the operating system (OS, e. g. in Apple Mac OS and Microsoft Windows). For Linux (and the open source world), such systems are currently under development. The CMS will take imagery and colour correct it for output through connected devices. The CMS knows (or is instructed through configurations) what output profiles to use and transforms the image to the designated output device's destination colour space before displaying or printing. The CMS also manages available devices and their profiles, so that it can perform the appropriate mapping.

On the application level, colour aware applications are ideally interfacing with the integrated CMS for output purposes (or input purposes in the case of cameras, scanners, etc.). Some colour aware application can manage the full source to destination chain themselves. In the case of the availability of additional OS integrated systems, one has to make sure that the colour chain is not compromised due to undesired multiple transformations.

5.8 Conclusions on Colour Management

Unless one is interested in the colour information on the level of the PCS directly, real world applications usually transform colours from a source colour space to a destination colour space. These are usually either device (camera, scanner, display, printer, etc.), working (e. g. Adobe RGB) or generic standard spaces (e. g. sRGB). For the scope of this research we are very interested in the PCS, namely CIE LAB. Therefore, we are dealing almost exclusively with ICC input profiles.

We are seeking a representation that is decoupled from a specific device, displays a much more linear behaviour (than e. g. RGB), and that is much better and more strictly defined. Choosing CIE LAB as a working space also means that we are dealing with an opponent colour space, which decouples lightness from chromaticity (hue and saturation). The reason for choosing to use ICC based colour management is that it is scientifically sound and has undergone extensive practice checks through years of application in the industry. Therefore, we do not need to prove the credibility of the approach. Other equally sound approaches

do exist, but they may require a verification, whether they form a solid basis for our colour research.

Due to the availability of the openly and freely available Little CMS library, we were easily able to implement all needed colour transformations. The profiler in Argyll CMS has shown to produce competitively good results in comparison with expensive commercial alternatives[2]. Therefore, it was chosen to be used both as a reference as well as a base for further successive colour based processing.

[2] http://article.gmane.org/gmane.comp.graphics.argyllcms/659

Chapter 6

Dynamic Adaptation

In this research we are trying to perform live colour measurements and reasoning through camera image perception. The colour perception as built into cameras by manufacturers is intended to "look good," but not to be accurate from a colourimetric perspective. In the previous chapters, we have discussed two popular techniques that can be used to increase chromatic correctness of perceived colours. First, we have discussed colour constancy in Chap. 4, particularly with a focus on being applicable on device-independent, standardised colour spaces. In Chap. 5, we have outlined aspects of standard compliant colour management, as it is used successfully in the graphic arts industry for more than a decade already.

Colour perception of cameras is influenced mainly by the current illumination of the captured scene. Other possible influences can be e. g. temperature, humidity, changed zoom factor of the lens, age, etc. These factors can change with varying rates over a measuring campaign with a camera. In order to increase the robustness of colour perception of a camera, we have a need for a system that is capable of adapting to occurring changes at run time.

Based on the discovered shortcomings of colour constancy in terms of adaptation capabilities and the static nature of ICC colour management (Sect. 6.1 and 6.2), in this chapter we will set up the foundation ideas of the research undertaken. A proposed hybrid approach is outlined in Sect. 6.3. This approach defines the work discussed in the following three chapters, namely the anatomy of an ICC profile and how to generate one from profiling data (Chap. 7), adaptive colour transformations to correct colour space distortions introduced through illumination changes (Chap. 8) and finally a way to augment existing ICC profiles with information from this corrective transformation for validity under currently given conditions (Chap. 9).

6.1 Limitations of Colour Constancy

Colour constancy works on a single image basis. It employs heuristics on the image to determine chromatic hints that can be used to correct colours. Then, the colours of the image are subjected to a corrective transformation. Mark Ebner discusses numerous algorithms to achieve this goal in his book "Color Constancy" [1]. Often, these hints determined are the "white point," the "black point" or an "average grey." The corrective transformation is built on the basis of these hints. This already reveals two significant draw backs of colour constancy: The low number of hints and ignoring multiple frames in time.

The low number of input parameters derived from the few hints determined are sufficient for colour corrections that involve to a certain degree a combination of shifting, tilting or simple scaling of the colour space volume. For more sophisticated transformations that involve for example rotation, bending, shearing or even anisotropic, non-uniform or local deformations cannot be determined through this low number of hints. To stay with linear transformations, an affine transformation for example requires already four data points acquired. To reduce unwanted side effects due to an insufficient accuracy of the hints, a larger number of data points is desirable, so that the transformation parameters can be acquired e. g. through a least squares error sum estimation.

Secondly, colour constancy does not take comparative information between frames acquired at different points of time into consideration. Scenes tend to exhibit certain constant features over time, which may reveal valuable additional information that can be used for better approximations of corrective transformations. This information may be derived from objects that are re-occurring, backgrounds that change only slightly, etc.

6.2 Limitations of Colour Management

When applying colour management on capturing devices, an input device ICC profile is needed. This profile is the result of a device characterisation conducted under specific (fixed) conditions. If these conditions change, or the device's capturing characteristics change, the profile will not represent a correct capturing characterisation anymore. In cases like this, a re-characterisation (potentially including a re-calibration involving changes to the device itself) will have to be performed (see Fig. 6.1). If the new conditions are known, potentially a pre-existing alternative profile matching the current conditions can be used. Of these two options, the first case is very disruptive to a continuous capturing campaign, and a full re-characterisation is a process with a duration in the order of magnitude of minutes (depending on how they are conducted).

For the reasons mentioned above, colour correction in dynamically changing environments using colour management with static profiles is practically not possible. The process of re-characterisation is too disruptive, and an archive of different profiles is too limiting in common natural environments.

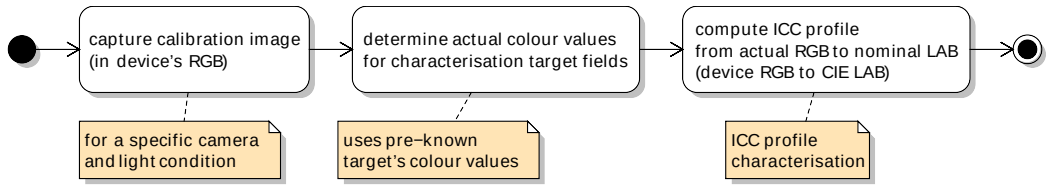

Figure 6.1: Static characterisation of camera setup to derive an ICC profile for a camera under given light conditions.

6.3 Dynamic Profile Adaptation – A Hybrid Approach

We have seen in Chap. 4, that colour constancy based colour image enhancement work on CIE colour spaces. Opponent colour spaces like CIE LAB are device-independent and visually very linear. For this reason, they form a solid basis on which to conduct colour analysis and adaptation. Colour constancy also gives us some good indicators as to where and how to find certain trackable distinct colours (like the white/black points or average colours) which can be used as part of the data set employed to determine corrective data transformations.

On the other hand, colour management provides the concepts and tools to perform corrective colour transformations on sound scientific foundations, which also have been industry proven for a long time. These colour transformations are conducted in a way that within the colour management module, all colour representations are converted to a profile connection space (PCS). This is either CIE XYZ or CIE LAB. Therefore, it is easily possible to use colour management to transform device-dependent input colours (RGB) to independent colours in $L^*a^*b^*$.

A full colour adaptation workflow for a proposed system could look like this:

0. *Initial full device characterisation:*
   Creating an ICC profile for the capturing device under initial conditions.

1. *Track multiple colour samples:*
   Acquire a sufficiently large number of data points through observation. Either by identification of distinct colours on trackable objects or by identification of static scene elements.

2. *Detect the shift in colour appearance:*
   Characterising the colour shift.

3. *Derive a compensative transformation:*
   Mapping the characterisation of the colour shift into a model, and determining the parameters for it numerically.

4. *Smelter this compensation into the existing ICC profile:*
   Fusing the newly acquired corrective transformation into the input profile, to have *one* total profile transformation that handles the current state of illumination.

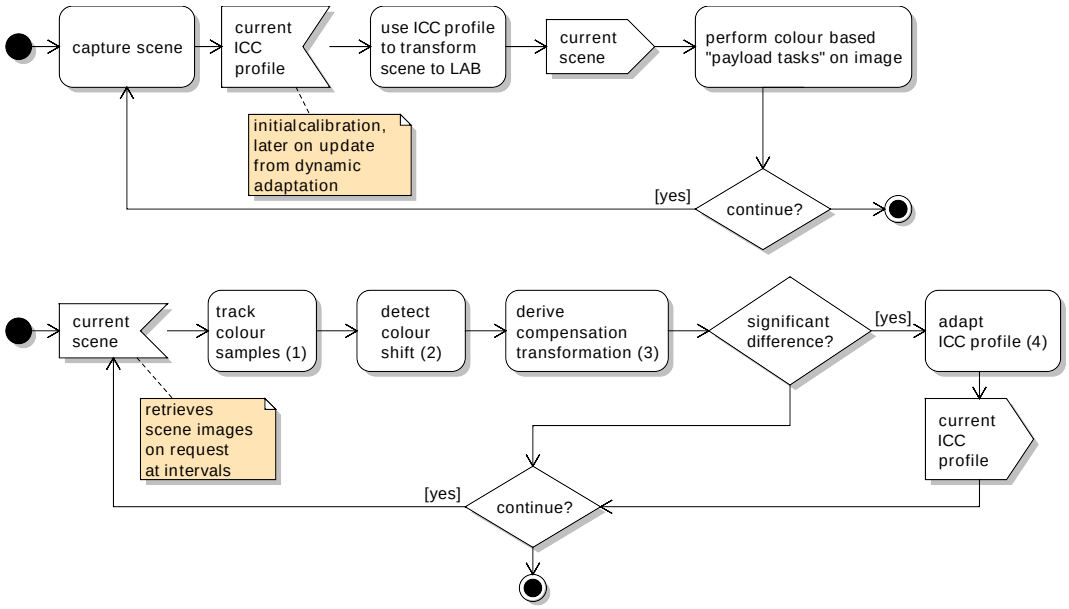

Figure 6.2: Image processing (top) and ICC profile adaption processes (bottom). Organised in a service oriented architecture, requesting images/ICC profiles on demand.

The steps 1–4 are repeated as necessary over the duration of a measuring campaign. It seems commonly practical to assume that illumination conditions do not change too rapidly (relative to the frame rate of video capturing). Therefore, we do not need to repeatedly perform these adaptation steps for every frame captured by the camera. A good measure for repeats may be the time scales of several seconds to minutes, or when some embedded heuristic indicates that a certain threshold of colour change in the tracked colour samples has occurred.

Fig. 6.2 outlines a sample scheme in a UML activity diagram, that implements the decoupled processes through communication using a service oriented architecture (SOA). The image acquisition and processing loop (top) may run with the full frame rate of the camera and apply a corrective ICC colour transformation (with the current ICC profile from step 0 or 4) in real time. The current scene content is sent to the independently running characterisation process. Upon receiving current scene content, this process (bottom) will proceed with the adaptation steps for the profile (1–4). In case the process decides that an updated profile is needed, the new profile is sent to the image acquisition and processing loop. This decoupling ensures a minimum disruption of the acquisition/processing loop, while still being able to react to changed conditions by an updated profile generated through an independent adaptation process.

In a similar approach, a system operating in this architecture has been implemented by Anzani et al. [52]. The major difference is, that Anzani et al. do not correct colour image perception, but perform colour segmentation (2-D Gaussian classifiers) for individual objects (presented for the domain of robot soccer). The adaptation of these parameters is

implemented to run on a separate computer which updates the segmentation parameters at a lower rate (5–8 Hz) than the image processing (15 Hz).

6.4 Conclusions on Dynamic Adaptation

Others have "synthetically" created corrective ICC profile for the purpose of colour correction rather than device characterisation [53,54]. In this research we are proposing a system that will alter a device's characterisation (the ICC profile), to fit it to the current conditions. Through doing this, we are able to build on solidly founded and tested ICC colour management processes as well as apply colour corrections in a more controlled and linear colour space than RGB, which may use at times unknown gamma functions. To achieve this goal it is essential to have a both a solid understanding of the process of ICC profiling as well as the internals of an ICC profile. Initial profiling is step zero in the previous section. The following Chap. 7 starts off at this point, and therefore provides a deep insight into the profiling process by outlining the construction of our own profiler.

The quality of the correction is on the one hand dependent on the accuracy of modelling the shift, as well as on the accuracy with which tracked samples' colours can be determined. The often rather complex characteristics of colour perception of cameras are largely already captured by the initially determined ICC input profile of the device. The adaptive terms smelted into the ICC profile only have a corrective nature, and should be generally much simpler in character than the full device characterisation. Therefore, it can be modelled with a lot less parameters, and the (computational) overhead over full characterisation is significantly lower. The steps one, two and three in the previous section describe approaches for colour sample tracking, characterisation of an introduced colour shift and the construction of a corrective colour transformation. Possible solutions to these three aspects are developed in Chap. 8.

An update of the ICC profile can be performed at run time without an interruption of the continuous observation. This is performed by a separate process, that is running largely independent of the colour correction process in the acquisition loop. The concurrently decoupled updating process produces adapted new profiles at regular intervals, or upon demand after sudden changes. A sample architecture for an implementation of this scheme through a service oriented architecture has been outlined. Employing this system as proposed will shift the colour correction approach from deductive or statistical inference in colour constancy to a measurement based derivation, while maintaining minimal disruption to image acquisition and processing. Updating an existing ICC profile with information from the corrective colour transformation requires access to the internals of the ICC profile. At this point the extra efforts undertaken by building our own profiler in Chap. 7 are giving us full access to all aspects of profile creation for this purpose. Chap. 9 picks up from this point and describes the exact process of fusing the colour correction into a standard conformant ICC profile.

Chapter 7

ICC Profile Generation

As mentioned in Chap. 5, ICC profiles are used to perform accurate and predictable colour conversion between different device colour spaces. This is accomplished by "bridging" the transformation via an intermediate, device-independent profile connection space (PCS), which can be either CIE LAB or CIE XYZ. For the purposes of this research, we are interested in using input transformations from the camera's device-dependent input colour space towards the independent PCS. Particularly, the properties of a PCS in $L^*a^*b^*$ are of interest.

This transformation is usually performed using a Colour Management Module (CMM) in conjunction with an ICC profile. ICC profiles have been standardised by the International Color Consortium (ICC) [49], which is also a confirmed international standard *ISO 15076-1:2005*.

As introduced, this chapter presents the steps taken to create ICC profiles for the research purpose: That is ICC input profiles for common cameras with three colour channels towards a CIE LAB PCS. The descriptions start with the general construction and elements of ICC profiles (Sect. 7.1), through the mathematical background of representing the transformation (Sect. 7.2) and its implementation (Sect. 7.3) towards a discussion of the results as presented in Sect. 7.4.

7.1 Blueprint of an ICC Profile

The precise construction of an ICC profile is described in the current version 4.4.0.0 specification [49]. The descriptions in this chapter target *device profiles* and areas relevant to this research only.

An ICC profile consists of exactly three sections:

- A profile header,

- a profile tag table,
- and the profile's tag data for all elements.

The *header* contains profile meta information like profile size, profile format version number, data colour space, PCS, time stamp, rendering intent, PCS white point (mandatory D_{50}), etc. This information is used by the CMM to determine what the profile can be used for, and it contains further meta data for managing the available ICC profiles on a system. The *tag table* serves management purposes for the specific profile only. It contains only the number of tags included, and for each tag the tag type, the byte offset within the profile, and the size of that specific tag. The *tag data* then actually contains the "pay load" of the profile used by the CMM to apply colour transformations. It contains data for each of the tags listed in the tag table.

7.1.1 ICC Tags

As mentioned, the tags contain the content used for performing actual colour space transformations. Some of these tags may contain additional meta data (profile description, copyright, manufacturer description, device model description, etc.) as well as tags containing colourimetric data (media white point, custom chromatic adaptation matrix, etc.). But the heart of the profile is stored in the transformation specific tags (transformation matrix, tone reproduction curves, "A to B"/"B to A", etc.). This last group of tags contains the transformation information, and these need to be crafted for each characterisation of a device to enable ICC based colour management.

For the most flexible and accurate characterisation one needs to resort to the Colour Lookup Table (CLUT) based profiles. The transformation information is stored in the "AToBx" tags for transformations to PCS, and "BToAx" tags for transformations from PCS. x denotes an indicator expressing which transformation to choose depending on the selected *rendering intent* (see Sect. 7.1.2).

"AToBx"/"BToAx" tags, as suitable here, consist of five processing elements (as shown in Fig. 7.1). *"A" curves* are the pre-linearisation tables per device colour channel (1-D), and *"B" curves* are the post-linearisation tables per PCS channel (1-D), respectively. The *"M" curves* are single channel transformation curves (1-D), permissible only together with a *linear transformation matrix*, which is only permissible for a PCS in CIE XYZ. The core of most non-trivial device profiles is the *multi dimensional CLUT*.

Elements not used in the tag can be either left out or have to be "disabled" by setting them to an identity transformation. For the case of the desired $L^*a^*b^*$ based device profiles, this is done for the matrix and "M" curves, as they are not applicable, as well as for the "B" curves often not used (as the CLUT interpolation then yields $L^*a^*b^*$ values directly).

These pre-linearisation curves, as well as the CLUT, must be computed in the device characterisation process (see Chap. 5) to populate the "AToBx"/"BToAx" tags. Within that process, a multi dimensional smoothing interpolation (see Sect. 7.2) must be performed.

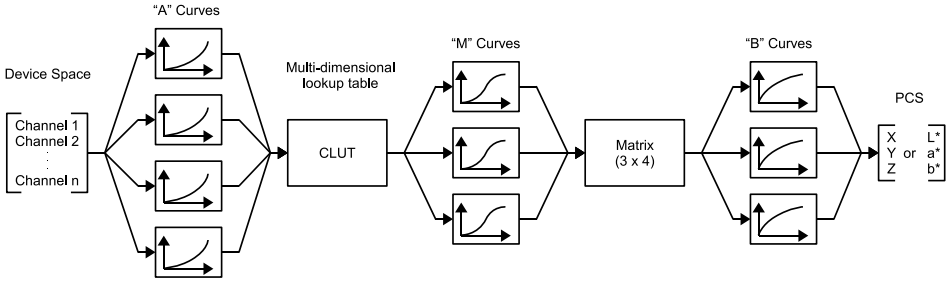

Figure 7.1: Chain of processing elements for a transformation from device to PCS with three or more components/colours.

Due to their enhanced capabilities and flexibility, we will use CLUT based profiles in this research only. A detailed description on how shaper/matrix based ICC profiles can be created can be found in [55].

7.1.2 Rendering Intents

When the gamut of the source colour space exceeds that of the destination, the saturated colours need to be treated in order to fit into the gamut. This process is performed by the CMM, and is handled according to one of four ways specified by the ICC, called *rendering intents*: (ICC) absolute colourimetric, (media-) relative colourimetric, perceptual and saturation [6,21]. The rendering intent is chosen by the user, and depends on the source image, source and destination colour spaces, and the "intent" of the application.

ICC Absolute – ICC Absolute colourimetry and media-relative colourimetry actually use the same table (from the "AToB1" tag), but differ in the adjustment for the media white point. This rendering intent is calculated by the CMM by transforming the coordinates to place the CIE colourimetry (relative to a perfect reflecting diffuser illuminated by a D_{50} illumination source) into the PCS. If the output device has for all colours a larger gamut than the source profile, i. e., all the colours in the source can be represented in the output, using the absolute colourimetry rendering intent would "ideally" (ignoring noise, precision, etc.) give an exact output of the specified CIE LAB values. Perceptually, the colours may appear incorrect, but instrument measurements of the resulting output would match the source. Colours outside of the system's possible colour are mapped to the boundary of the colour gamut.

Media-Relative – The goal in media-relative colourimetry (using the "AToB1" tag) is to be truthful to the specified colour, with only a correction for the media white point. Media differences are the only thing that need to be adjusted for. Gamut mapping (for out of gamut colours) usually is handled in a way that hue and lightness are maintained at the cost of reduced saturation. Relative colourimetric is the default rendering intent on most systems.

Perceptual and Saturation – For the perceptual ("AToB0" tag) and saturation ("AToB2" tag) intents, the results largely depend upon the profile maker. This is one way for competitors in this market to differentiate themselves. Perceptual intent profiles are commonly created to result in "pleasing images," while the saturation intent is often used for eye-catching business graphics. This is achieved through the use of different perceptual remaps of the data as well as different gamut mapping methods. Both of these are commonly recommended for colour separation for printing.

Profiles can be annotated with a supported rendering intent, and not support the full range of possible rendering intents. In these cases often only the "AToB0" and "BToA0" tags are written, but used for either media relative, perceptual or saturation intents as well, depending on which one the profile has been created for.

The question now is, which rendering intent to use for the purpose of capturing imagery from a digital camera to analyse colourimetric information? The perceptual and saturation intents handle colour in a proprietary and non-standardised way to create "pleasing images" by compressing or expanding the tone scale [49], particularly for colours close to the gamut boundary. We are interested in maintaining image colourimetry as exactly as possible. So these intents are not suitable. Additionally, the PCS can assumed to be large enough to accommodate any input device gamut. Technically, the PCS is limited only by the boundaries of the PCS colour encoding (16 or 8 bit), which is designed to be large enough for all realistic cases. Due to the lacking compensation for the illuminant's white point, absolute colourimetry is not ideal [56]. Therefore, media-relative colourimetry is the rendering intend most suitable for this research purpose.

7.2 Interpolation

In the characterisation process (described in Chap. 5) for an input device, a precisely manufactured test target is used. This test target contains a larger number (usually a few hundred) of coloured test patches. The colour of each of these patches has been measured spectrophotometrically (e. g. as CIE XYZ or CIE LAB colour tuples). An image of this test target is captured by the input device, in this case the digital colour camera. For each of the test patches, the colour appearance in the device's colour space (e. g. RGB tuples) is determined. A mapping of the absolute CIE colours to the respective device colours is performed to produce the 1-D pre-linearisation "A" curves as well as the multi-dimensional colour LUTs (see Sect. 7.1).

The 1-D curves (per channel) often consist of a table with a resolution of 1024 nodes. Most CMMs use an internal floating point or 16 bit integer representation of the colour channels to reduce rounding artefacts. Intermediate values not coinciding with nodes of the interpolation table are linearly interpolated. The multi-dimensional CLUTs are constructed in a similar fashion. These often have resolutions of 17 or 33 nodes per dimension, resulting in a size of 17^3 or 33^3 interpolation nodes for common tri-chromatic input devices. Previous

studies have shown, that for most devices a finer sampling than 17 nodes does not yield a noticeable improvement in accuracy [57]. Intermediate values (between the CLUT nodes) are (tri-) linearly interpolated.

These interpolations are performed within the CMM, and due to their simple nature they do not pose any significant challenge. In the process of the creation of these tables however, interpolation needs to be accomplished in a much more complex fashion. On the one hand, these transformation tables need to be created from a very sparse data sampling, as the number of measured data points to total data nodes is usually between $1:10$ and $1:100$. Furthermore, the data points are always inaccurate to a certain degree due to measurement. Lastly, the data points are not arranged in an orderly fashion, but have to be assumed to be semi randomly distributed. The interpolation has to deal with these issues to produce a smoothed (vector) interpolation field over the whole gamut of the device.

Many approaches with even more variations can be found for this type of problems. Kriging for example, is made for interpolation of a single realisation of a random field, however it does not handle data point inaccuracies well. More appropriate are multi-dimensional versions of regression analysis models, based on multiple observations of a multivariate data set. The encountered lookup tables are overall quite smooth (they contain no "kinks" or "folds"), and they can be locally very well approximated linearly within the required resolutions of tables. The creation of the two types of tables therefore can very well be managed through the fitting of a linear spline function.

7.2.1 Regularised Linear Spline Fitting

Smooth interpolations have been accomplished with various approaches. Many implementations are based on performing multi-dimensional local linear regression computations on neighbourhoods of measurement data surounding the LUT nodes [53,54,58,59]. Others have used back propagating neural networks [60] that were trained on the measurement data to populate the node points of the LUT. Also, discrete smooth interpolation [61] can be used on structured as well as unstructured grids. Quite popular for multi-dimensional problems on regular grids is the approach of using *regularised thin plate splines*. These are also frequently and successfully used to interpolate geographical topographies from an irregular series of sparsely sampled altitude measurements. This approach can easily be extended to the interpolation of vector fields in higher dimensionalities. Bone has described a very flexible approach for this ,based on a physical model of nodes interconnected through springs [62]. This model has also been used successfully for the creation of LUTs in device colour characterisations [63,64], and it forms (in a simplified variation) the numerical basis of the very popular and highly appreciated open source CMM and profiling tool Argyll CMS [50].

The basic idea behind the noise reducing fit, towards a regularised linear spline, is to represent the problem as a simple physical model as illustrated in Fig. 7.2. In this model, each node is represented as a mechanical joint. All these joint are connected through mechanical

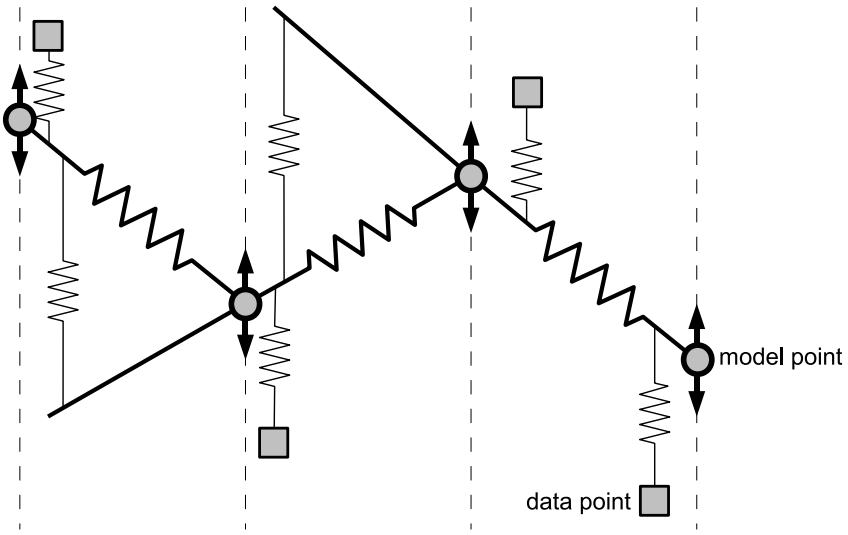

Figure 7.2: The mechanical analogue of the modelling process. Nodes, interconnected with elastic springs into an elastic model. Fixed data points are also attached to the elastic model through individual springs.

springs to their neighbour joints *(first difference springs)*, to form an elastic network spacing the joints equally. Data points are located in their respective positions, which do not have to line up with the joints' positions. The fixed data points are connected with further springs *(data springs)* to the spring links between the joints to attract them elastically. Lastly, each spring linking the joints extends beyond the joint stiffly, and is linked to the next connection spring with another spring *(second difference springs)*. This spring counteracts bending, and therefore smooths out the curvature of the curve through the model nodes' joints.

By balancing the spring coefficients relative to each other, one can control the fitting of the curve to the data points, which are imprecise due to measurement noise. "Stiffer" data springs give a closer fit to the data points. The spring elongation of the first difference springs is proportional to the first derivative, determined by the difference between the joints (slope between nodes). Increasing the first difference spring stiffness spaces data points more evenly. Finally, the spring elongation of the second difference springs is proportional to the second derivative, determined by the change in slope between the links, before and after a joint. Increasing this spring stiffness resists tight bends in the curvature.

In higher dimensional systems, each joint will not be directly connected (first difference springs) to two neighbours (1-D), but to four (2-D) or to six (3-D). Similarly, also the number of second difference springs increases with the number of neighbours. Furthermore, also "mixed" (diagonal, over two dimensions) second derivatives appear, which are assigned second difference springs, too.

For the general application of the regularised linear smooth spline fitting, one has to account for kinks, sharp bends or folds. These will be potentially smoothed out too much, and the resulting topography will not follow closely a desired shape. A possible counter

measure against this over-smoothing effect can be to use a method of spring weakening after the convergence of the (numerical) solution of the general spline problem. Areas with high (second) derivatives are under the highest stress due to sharp bends introduced through the data points. Therefore, if springs can be weakened progressively by a function of their elongation (magnitude of the derivative), a more "snug" fit of the topography to the measured data points in and around areas of these high distortions can be achieved.

As the LUT matrix shapes tend to be rather smooth, progressive spring weakening can be neglected. The intent is to get a smoothed representation ("noisy" only to a certain degree) just from data. So, equal spacing of the LUT nodes is not an issue, either. The coupling of the nodes through the second difference springs is all that is needed to generate a good fit. Lastly, also due to the already "good enough" coupling and basically smooth nature, we can disregard mixed second difference springs (diagonal second difference springs). All these simplifications result in a largely reduced numerical overhead in computing the spline fit without a noticeable loss in quality.

Let us now discuss the mathematical side of this mechanical model. In the analogy, the system strives to minimise its energy, which is composed of the sum of the energies of all the springs. Therefore, the energy can be regarded as a penalty function to be minimised. We are going to start to formulate the problem for the simple 1-D case first in (7.1) and (7.2). The index e denotes data point spring terms. Indices x, y and z indicate the spacial direction of the spring terms. Second difference spring terms are indicated by a dual index, e. g. xx for the uni-directional, and xy for "mixed" second difference springs. (Note, that the description by Bone [62] contains errors, by not including the coefficient "2" in the second difference springs P_{xx} of (7.2).)

$$P = P_e + w_x P_x + w_{xx} P_{xx} \tag{7.1}$$

where

$$\begin{aligned} P_e &= \sum_n \left(d_n - \sum_i a_i^n u_i \right)^2 k_n^e \\ P_x &= \sum_i \left(u_{i+1} - u_i \right)^2 k_i^x \\ P_{xx} &= \sum_i \left(u_{i-1} - 2u_i + u_{i+1} \right)^2 k_i^{xx} \end{aligned} \tag{7.2}$$

The penalty P is the sum of the three penalties: for the elongations of the springs to the data points P_e, the first difference springs P_x and the second difference springs P_{xx} – with the corresponding weight factors w balancing the individual contributions. The spring system is defined using a linear spring force model (LSF) with constant spring coefficients k ("stiffness" of a spring). The data point penalty P_e is comprised of the energies associated

with the springs of all n data points at the positions d_n, respective to each of their influenced model nodes. Contributions of the model nodes for these data points are derived through their interpolation coefficients a_i^n. Note, that the superscripts of a_i as well as the spring constants k are labels, *not* powers. The values at the model nodes i are denoted as u_i.

For the equivalent problem in the 2-D case, with "mixed" second difference springs, analogously according to (7.3) and (7.4).

$$P = P_e + w_x P_x + w_y P_y + w_{xx} P_{xx} + w_{yy} P_{yy} + w_{xy} P_{xy} \tag{7.3}$$

where

$$\begin{aligned}
P_e &= \sum_n \left(\boldsymbol{d}_n - \sum_i a_{i,j}^n \boldsymbol{u}_{i,j} \right)^2 k_n^e \\
P_x &= \sum_{i,j} \left(\boldsymbol{u}_{i+1,j} - \boldsymbol{u}_{i,j} \right)^2 k_{i,j}^x \\
P_y &= \sum_{i,j} \left(\boldsymbol{u}_{i,j+1} - \boldsymbol{u}_{i,j} \right)^2 k_{i,j}^x \\
P_{xx} &= \sum_{i,j} \left(\boldsymbol{u}_{i-1,j} - 2\boldsymbol{u}_{i,j} + \boldsymbol{u}_{i+1,j} \right)^2 k_{i,j}^{xx} \\
P_{yy} &= \sum_{i,j} \left(\boldsymbol{u}_{i,j-1} - 2\boldsymbol{u}_{i,j} + \boldsymbol{u}_{i,j+1} \right)^2 k_{i,j}^{yy} \\
P_{xy} &= \sum_{i,j} \left(\boldsymbol{u}_{i+1,j+1} - \boldsymbol{u}_{i,j+1} - \boldsymbol{u}_{i+1,j} + \boldsymbol{u}_{i,j} \right)^2 k_{i,j}^{xy}
\end{aligned} \tag{7.4}$$

As the values u and data points d from the 1-D case were scalars, in the 2-D case they now became vectors $\boldsymbol{u}$ and $\boldsymbol{d}$ in an n-D to n-D spline interpolation.

Finally, the 3-D case is very similar to the 2-D case in its form, only containing a third dimension component. For brevity only the sum of the energies is listed in (7.5).

$$P = P_e + w_x P_x + w_y P_y + w_z P_z + w_{xx} P_{xx} + w_{yy} P_{yy} + w_{zz} P_{zz} + w_{xy} P_{xy} + w_{yz} P_{yz} + w_{xz} P_{xz} \tag{7.5}$$

In the LSF model, spring forces are linear with the elongation, as the spring constants k are constant for each spring. The energies will depend quadratically with on the extensions as stated above. A quadratic function will have one distinct extreme (obviously always a minimum in this case). At the minimum of the penalty function, the gradient with respect to any of the variables will be zero. This will give us a set of linear equations, one equation for each model point. In the 3-D case we have

$$\nabla P = \frac{\partial P}{\partial \boldsymbol{u}_{i,j,k}} = \sum_\tau w_\tau \frac{\partial P_\tau}{\partial \boldsymbol{u}_{i,j,k}} = 0 \tag{7.6}$$

7.2.2 Numerical Formulation and Solution

To solve the minimum energy/penalty problem, we need to differentiate the quadratic penalty terms. These are then collected for each model mesh node in turn. Below, we are deriving the differential terms to consider for each node "i, j, k", indicated more compactly for this dependent variable by the subscript "$*$".

First the data springs (7.7). Here "$\delta_{i,j,k}$" denotes all direct surrounding neighbour nodes of the current node (excluding the current node itself). These neighbours include the diagonal direct neighbours.

$$\nabla P_e = -2\, k_n^e a_*^n \left(\boldsymbol{d}_n - \sum_{\delta_{i,j,k}} a_{i,j,k}^n \boldsymbol{u}_{i,j,k} - a_*^n \boldsymbol{u}_* \right) \tag{7.7}$$

Each observed node participates in spring tensions with its direct neighbours. In the first difference springs, these are the forward and backward neighbours resulting in two contributions. For simplicity, only the differential in one dimension (x) is explicitly stated (7.8), with the other dimensional terms analogously.

$$\begin{aligned} \nabla P_x = &-2\ (-k_*^x \boldsymbol{u}_* + k_*^x \boldsymbol{u}_{i+1,j,k}) \\ &-2\ \left(-k_{i-1,j,k}^x \boldsymbol{u}_* + k_{i-1,j,k}^x \boldsymbol{u}_{i-1,j,k}\right) \\ = &-2\ \left[(-k_*^x - k_{i-1,j,k}^x)\boldsymbol{u}_* + k_*^x \boldsymbol{u}_{i+1,j,k} + k_{i-1,j,k}^x \boldsymbol{u}_{i-1,j,k}\right] \end{aligned} \tag{7.8}$$

The uni-directional second difference springs contain the coupling contributions from forward and backward nodes. Again, only one of the three equations is stated (7.9).

$$\nabla P_{xx} = -2\, k_*^{xx} \left(-4\boldsymbol{u}_* + 2\boldsymbol{u}_{i+1,j,k} + 2\boldsymbol{u}_{i-1,j,k}\right) \tag{7.9}$$

The second difference springs also contain diagonal components over two dimensions. For these, again, only one representative, in this case in the xy plane, is explicitly stated (7.10).

In this plane, each node has got four surrounding quadrants contributing penalties.

$$
\begin{aligned}
\nabla P_{xy} = &-2\ \left[-k_{*}^{xy}\ (\boldsymbol{u}_{i+1,j+1,k} - \boldsymbol{u}_{i+1,j,k} - \boldsymbol{u}_{i,j+1,k} + \boldsymbol{u}_{*})\right] \\
&-2\ \left[-k_{i-1,j,k}^{xy}\ (-\boldsymbol{u}_{i-1,j+1,k} + \boldsymbol{u}_{i-1,j,k} + \boldsymbol{u}_{i,j+1,k} - \boldsymbol{u}_{*})\right] \\
&-2\ \left[-k_{i-1,j-1,k}^{xy}\ (\boldsymbol{u}_{i-1,j-1,k} - \boldsymbol{u}_{i-1,j,k} - \boldsymbol{u}_{i,j-1,k} + \boldsymbol{u}_{*})\right] \\
&-2\ \left[-k_{i+1,j-1,k}^{xy}\ (-\boldsymbol{u}_{i+1,j-1,k} + \boldsymbol{u}_{i+1,j,k} + \boldsymbol{u}_{i,j-1,k} - \boldsymbol{u}_{*})\right] \\
= &-2\ \left[\left(-k_{*}^{xy} - k_{i-1,j,k}^{xy} + k_{i-1,j-1,k}^{xy} - k_{i,j-1,k}^{xy}\right)\boldsymbol{u}_{*}\right. \\
&\quad + k_{*}^{xy}\ \boldsymbol{u}_{i+1,j+1,k} \\
&\quad + k_{i-1,j,k}^{xy}\ \boldsymbol{u}_{i-1,j+1,k} \\
&\quad + k_{i-1,j-1,k}^{xy}\ \boldsymbol{u}_{i-1,j-1,k} \\
&\quad + k_{i,j-1,k}^{xy}\ \boldsymbol{u}_{i+1,j-1,k} \\
&\quad + (k_{i-1,j,k}^{xy} - k_{*}^{xy})\ \boldsymbol{u}_{i+1,j,k} \\
&\quad + (k_{i-1,j,k}^{xy} - k_{i-1,j-1,k}^{xy})\ \boldsymbol{u}_{i-1,j,k} \\
&\quad + (k_{i,j-1,k}^{xy} - k_{i-1,j-1,k}^{xy})\ \boldsymbol{u}_{i,j-1,k} \\
&\quad \left. + (k_{i,j-1,k}^{xy} - k_{*}^{xy})\ \boldsymbol{u}_{i+1,j,k}\right]
\end{aligned}
\tag{7.10}
$$

Setting up a linear equation system using (7.6) as the differential term of (7.5), with (7.7)–(7.10) (including the equations for dimensions not listed) for every node of the model mesh, creates a sparse linear equation system typically of the order of 1000–1,000,000 unknowns. Such large systems are commonly solved numerically, a very common approach is using Gauss-Seidel iterations. For each model point, the equations corresponding to that model point are solved for the new value at that point, while all other model points remain fixed. The new model value is used for all subsequent computations. In this way, in turn all model nodes are "unlocked," while all others remain "locked," to find the individual equilibrium points: re-locking, and moving to the next model point, sweeping the entire mesh repetitively. By this approach, the total energy/penalty of the system decreases. The iterations look like a wave propagating through the grid relaxing local spring tensions [62].

Another effect is, that high frequency distortions will decay much more quickly than low frequency distortions. For this purpose, so called "full multi grid" methods (FMG) [65] (as frequently used e. g. in computational fluid dynamics) may be used in their different variations, to take optimal advantage of different convergence properties at different grid resolutions. Due to the uniformly continuous nature with only low curvatures, it is sufficient to start on a rough grid, and approach the desired target resolution within a few steps after each resolution has converged sufficiently. The starting values of the next finer resolution are derived through (poly-) linear interpolation of the previously converged coarser resolution.

According to the full implementation of the described regularised spline fitting algorithm described by Bone [62], this multi grid iteration using Gauss-Seidel for a solution of the LSF problem is described as *stage 1.* To extend the LSF for high local gradients it needs to be

transformed to a non-linear spring force problem (NSF). For this, in *stage 2*, springs with high extensions are successively weakened, and each model is solved until converged before the springs with the highest extensions are weakened again. A system like that should be converged until it is very close to its energetic minimum. The small outstanding corrections then are performed in *stage 3*, in which after every relaxation iteration spring coefficients are adapted to relieve stronger outstanding local tensions.

7.2.3 Approach for Border Problems

The way the fitting of the spline is solved for all nodes, all spring loads associated to that node are added and minimised through relocating the node's position value. This works well almost on the entire model mesh. However, this model fails for the corner nodes of the mesh, where the possible spring forces cannot be counteracted by opposite neighbours. All terms that cannot be "satisfied" by the availability of neighbour nodes in the spacial directions are discarded. On the corners (eight corners in 3-D), at least one of the required neighbour nodes for each term is missing. The linear system is not defined, if no data point is present in the surrounding mesh cell.

These points are irrelevant for the future purpose of the application, as they usually lie outside of the device's colour gamut. So, a good precision for their model values is not needed. But they do have a strong influence on their neighbours and the overall shape of the smoothed/converged mesh topography. Following are the various approaches that have been evaluated for solving this problem:

- Fixed setting (to 0 (zero) or an extreme value),
- Replication of the closest data point,
- n-dimensional extrapolation from $n+1$ closest data points,
- Extrapolation from $n+1$ closest mesh nodes after every relaxation iteration,
- Smoothing extrapolation through poly-linear regression.

Applying a *fixed setting* to the extreme corners was the easiest solution. A new data point at the exact corner nodes' locations was created, and its value was set to a zero vector. Distortions to the converged mesh were by far too big, and the result was not usable. Alternatively, a heuristic was developed to set the corner values to extreme values that are theoretically plausible. Distortions to the topography were not as severe, as a generally sensible shape was generated this way. But the results were still too far off a satisfactory fit.

As a next approximation, the *closest data point* has been replicated at the corners in the fashion of a nearest neighbour approximation. This approximation already worked quite well in the way that it retained the general shape of the interpolation volume. However, the

data point sampling density usually is not satisfactorily sampled close to the extreme values of the input space, as these mostly represent out of gamut colour values. Therefore, the nearest neighbour interpolation is "cutting corners", and introduces a gamut compression in the interpolation volume of the destination space.

Due to the smooth, and locally rather uniform, nature of the destination splines, the idea was to use the nearest $n+1$ data points to compute an *n-dimensional extrapolation* towards the corner points. Due to the measurement error of the data points, this approach failed badly, and results were far worse than the closest data point approach described above. The combination of sometimes closely co-located data points with overlapping error margins resulted in very poor positioning of the extrapolated data points.

Another attempt was to *extrapolate from spline mesh nodes* towards the corner points. This process needs "boot strapping" using initially the nearest neighbour approach mentioned above, and then newly extrapolating after every convergence iteration using the $n+1$ nearest mesh points. Initially, the geometry of the mesh's volume was good, but it degraded with each iteration. The corners and bounding surfaces "sagged" in like a soufflé incautiously and prematurely leaving the oven.

To counter the measurement error of the data points a greater number of these data points needed to be taken into consideration, to smooth out the error distribution. As stated above, in a first approximation, the spline shape can locally be assumed to be linear. Computing a *poly-linear regression* using a higher number of data points closest to the missing corner finally resulted in good fits of the overall converged spline. For the regression, the common calculation of linear regressions using least square sum minimisation was extended to the multi-dimensional case.

Multi-Dimensional Regression

Due to the smooth and continuous nature of the vector fields encountered, vectors can be linearly approximated within local neighbourhoods. This linear approximation in $\mathbb{R}^n$ space is an affine transformation, and can be determined through a linear regression computation on a set of more than $n+1$ domain and corresponding range vectors of the vector field.

We can use this regression computation to determine a good fit of the hyper-cube's corner points for the smoothing interpolation from a range of nearest neighbour points. Furthermore, this regression is also very useful for computing corrective affine transformations as used in Chap. 8 (see Sect. 8.2.1).

As all computations are facilitated through Python scripts, the algorithm implementation is geared to work as efficiently as possible for Python using the NumPy [66, 67] extension library.

The algorithm is mathematically derived from a simple least squares based linear regression [68] towards higher dimensionalities. A review of existing descriptions resulted in an

algorithm as described by Helmuth Späth [69] for similar problems. The implementation as used here however follows directly the principles of the commonly known linear regression calculation, and its implementation is further described in [70].

We are assuming two vector sets P (representing in our case the set of vectors in the transformation's source colour space) and Q (representing the set of vectors in the destination colour space, for input profiles these are the device-independent $L^*a^*b^*$ vectors) – each of m vectors – describing the source and destination of the affine transformation, given by

$$\boldsymbol{p}_i = (p_{1i}, \ldots, p_{ni})^T, \qquad \boldsymbol{q}_i = (q_{1i}, \ldots, q_{ni})^T \qquad (i = 1, \ldots, m). \tag{7.11}$$

We now want to find some matrix $\boldsymbol{A}$ and translation vector $\boldsymbol{b}$

$$\begin{aligned} \boldsymbol{A} &= \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \\ \boldsymbol{b} &= (b_1, \ldots, b_n)^T \end{aligned} \tag{7.12}$$

such that

$$\boldsymbol{q}_i \approx \boldsymbol{p}_i^T \cdot \boldsymbol{A} + \boldsymbol{b} \qquad (i = 1, \ldots, m). \tag{7.13}$$

Dot product multiplication of matrices does not commute. However, transposing both matrices allows us to reverse the order of the terms in the dot multiplication. The matrix $\boldsymbol{A}$ is a square unknown, so it does not require any treatment, whereas the column vector $\boldsymbol{p_i}$ is transposed to a row vector $\boldsymbol{p}_i^T$ for this purpose. The reason for this rearrangement is that NumPy [66, 67] supports very fast and simple to perform dot product multiplication of vector arrays with matrices. This multiplication of the whole array of vectors can be executed very efficiently in a single function call.

The number of unknowns is $n^2 + n$, therefore we need at least $m > n^2 + n$ items in P and Q. For this linear regression computation, the Euclidean error $\boldsymbol{\epsilon}$ needs to be minimised globally.

$$\boldsymbol{\epsilon}_i = \boldsymbol{p}_i^T \cdot \boldsymbol{A} + \boldsymbol{b} - \boldsymbol{q}_i \tag{7.14}$$

A way to do this is to find the least sum of the squared errors E:

$$
\begin{aligned}
E(\boldsymbol{A},\boldsymbol{b}) &= \sum_{i=1}^{m} \|\boldsymbol{p}_i^T \cdot \boldsymbol{A} + \boldsymbol{b} - \boldsymbol{q}_i\|^2 \\
&= \sum_{i=1}^{m} \left[\begin{pmatrix} p_{1i} \\ \vdots \\ p_{ni} \end{pmatrix}^T \cdot \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} - \begin{pmatrix} q_{1i} \\ \vdots \\ q_{ni} \end{pmatrix} \right]^2 \\
&= \sum_{i=1}^{m} \left[\begin{pmatrix} p_{1i}a_{11} + p_{2i}a_{21} + & \cdots & + p_{ni}a_{n1} \\ p_{1i}a_{12} + p_{2i}a_{22} + & \cdots & + p_{ni}a_{n2} \\ \vdots & \ddots & \vdots \\ p_{1i}a_{1n} + p_{2i}a_{2n} + & \cdots & + p_{ni}a_{nn} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} - \begin{pmatrix} q_{1i} \\ q_{2i} \\ \vdots \\ q_{ni} \end{pmatrix} \right]^2
\end{aligned}
\tag{7.15}
$$

The minimum can be found by determining $\boldsymbol{A}$ and $\boldsymbol{b}$ where the partial derivatives of E become zero:

$$
\begin{aligned}
\frac{\partial E}{\partial a_{11}} &= 0 = 2\sum_{i=1}^{m} p_{1i}\left(p_{1i}a_{11} + p_{2i}a_{21} + \cdots + p_{ni}a_{n1} + b_1 - q_{1i}\right) \\
\frac{\partial E}{\partial a_{12}} &= 0 = 2\sum_{i=1}^{m} p_{1i}\left(p_{1i}a_{12} + p_{2i}a_{22} + \cdots + p_{ni}a_{n2} + b_2 - q_{2i}\right) \\
&\vdots \\
\frac{\partial E}{\partial a_{21}} &= 0 = 2\sum_{i=1}^{m} p_{2i}\left(p_{1i}a_{11} + p_{1i}a_{21} + \cdots + p_{ni}a_{n1} + b_1 - q_{1i}\right) \\
\frac{\partial E}{\partial a_{22}} &= 0 = 2\sum_{i=1}^{m} p_{2i}\left(p_{1i}a_{12} + p_{2i}a_{22} + \cdots + p_{ni}a_{n2} + b_2 - q_{2i}\right) \\
&\vdots \\
\frac{\partial E}{\partial a_{nn}} &= 0 = 2\sum_{i=1}^{m} p_{ni}\left(p_{1i}a_{1n} + p_{2i}a_{2n} + \cdots + p_{ni}a_{nn} + b_n - q_{ni}\right)
\end{aligned}
\tag{7.16}
$$

and

$$\begin{aligned}
\frac{\partial E}{\partial b_1} &= 0 = 2\sum_{i=1}^{m}(p_{1i}a_{11} + p_{2i}a_{21} + \cdots + p_{ni}a_{n1} + b_1 - q_{1i}) \\
\frac{\partial E}{\partial b_2} &= 0 = 2\sum_{i=1}^{m}(p_{1i}a_{12} + p_{2i}a_{22} + \cdots + p_{ni}a_{n2} + b_2 - q_{2i}) \\
&\vdots \\
\frac{\partial E}{\partial b_n} &= 0 = 2\sum_{i=1}^{m}(p_{1i}a_{1n} + p_{2i}a_{2n} + \cdots + p_{ni}a_{nn} + b_n - q_{ni})
\end{aligned} \tag{7.17}$$

From (7.16) and (7.17) we obtained a linear equation system, that can be described in the form

$$\boldsymbol{C} \cdot \boldsymbol{x} = \boldsymbol{d} \tag{7.18}$$

with the sparse matrix $\boldsymbol{C}$ to solve for the "solution vector" $\boldsymbol{x}$ containing the unknowns of the matrix $\boldsymbol{A}$ as well as the vector $\boldsymbol{b}$. The system has got $t = n^2 + n$ unknowns. The $t \times t$ size matrix is populated and can be solved easily using Gaussian elimination or any linear system solver. In this case, the readily available solver from the NumPy package is used. Alternatively, the system could also easily be described as a set of n equation systems, consisting of one equation for each system containing of a_{ij} with $i = 1, \ldots, n$ from (7.16), and one equation for b_i from (7.17). This approach would result in less memory consumption for the then non-sparse matrix and less memory consumption for the solver, but at the expense of a slightly more complicated implementation. The simpler implementation, however, proved to be extremely fast even for higher dimensions n and point set sizes m, that the additional implementation overhead was not justifiable.

After obtaining the solution vector $\boldsymbol{x}$, the first n^2 elements of it are used to construct the linear matrix $\boldsymbol{A}$, and the last n elements for the translation vector $\boldsymbol{b}$ used for the approximate transformation stated in (7.13) that is to be applied to the current colours of set P.

7.3 Implementation

To recap, the following outlines roughly the process for computing the CLUT for an ICC profile from data obtained through measuring a test chart is summarised:

0. Build a computational mesh with a coarse starting resolution.

1. Add "synthetic" data points for corner nodes
   by computing poly-linear regressions to treat the border problems.

2. Set up the equation systems for the mechanical analogy on each node.

3. Solve the equation system numerically using Gauss-Seidel iterations.

4. Refine the mesh to the next higher node density, and proceed with step 1 until the final resolution is reached.

5. When the final resolution is reached and converged,
   populate the CLUT with the mesh's result vector field.

In the profiler's core, the smoothing regularised linear spline fitting has been implemented, and applied to create input profiles from an RGB camera towards $L^*a^*b^*$ PCS for 16 bit precision CLUT based ICC profiles. The fitted spline was used to populate a LUT (in the experiments with 17^3 nodes) in the "AToB1Tag" for the media-relative colourimetric rendering intent. To distinguish this profiler from others in later comparisons, it has been named "Mārama," for the Māori word for light:

> mārama
>
> 1. (stative) be clear, light (not dark), easy to understand, lucid, bright, transparent. *[...]*
> 2. (noun) brightness, clearness.
>
> (Source: `http://maoridictionary.co.nz/?dictionaryKeywords=marama`)

Argyll CMS [50] was used in all experimental cases to extract measurement data from the image of the captured test target (Fig. 7.6, top left). The tool `scanin` identifies the colour patches in the target's image using a target description file (`.cht` file), and pairs the identified patch source colour data with the canonical CIE colour data obtained from the target's vendor (`.cie` file). The results are written out to a data file (`.ti3` file), which the Mārama and Argyll ICC colour profilers are capable of reading. The profiler from the Argyll CMS tool suite is `colprof`. It is well known in the field of colour researchers for its quality, and therefore used here as a reference. Our own profiling system Mārama has been similarly equipped with a parser for this particular file format to read the `.ti3` files. The comparison of the obtained ICC profiles can be reduced to the pure profile generation, and is independent of the way the measurement data is retrieved from the image of the test target.

This research's sole focus is on the numerical mapping and the colourimetric tracking of colour changes. Technical details are largely delegated to specific existing libraries: Reading of images is handled by the Python Imaging Library (PIL) [71], measuring of the test target's colour patches by Argyll CMS' `scanin` [50], and handling of ICC profiles (creating, manipulating and applying) by Little CMS [42].

7.4 Results

Besides the experimentation with the corner problem (Sect. 7.2.3) that was necessary to make the algorithm work in the first place, further parameters and variations needed to be examined for suitability. These were primarily the different spring constants and their behaviour (LSF vs. NSF), as well as the suitability of various simplifications.

Sect. 7.2.1 introduced a method to derive fitted lookup table data for interpolation that can be derived from a sparse collection of measurements. Results of the validation for the LUT fitting process are given below in Sect. 7.4.1. In real world environment measurements are always superimposed by noise. Sect. 7.4.2 gives the results of an evaluation of the ICC colour profiling process using Mārama in the presence of measurement noise in comparison to reference profiles computed using Argyll CMS. Sect. 7.2.1 and Sect. 7.4.1 together shall prove the suitability of using the chosen spline fitting approach for the purpose of building colour lookup tables for the purpose of ICC profiling.

7.4.1 n-D Spline Fitting

For constructing ICC profiles for this research, primarily 3-D to 3-D spline fitting was the essential hurdle to overcome for building the CLUTs. Additionally, 1-D to 1-D fitting (Fig. 7.3 left) is useful for building the pre-linearisation tables per input channel, as cameras usually produce gamma corrected output. Therefore, we have tested the suitability for these given cases. For ease of analysing multi-dimensional cases, we have also conducted experiments with 2-D to 2-D fitting experiments (Fig. 7.3 right), as it is much easier to perform a visual evaluation for these.

In the 1-D case on the left of Fig. 7.3, a linear spline curve is fitted to a set of five data points (red diamonds in the middle graph). The data points have been computed from a quadratic function with added random noise. The fitting is performed in a multi grid fashion, starting out with a five node mesh ranging from 0.0 to 255.0. After five iterations, the spline reached its convergence criteria of a maximum iteration displacement for a node of 0.5 units per iteration step. The convergence criteria is plotted in the top graph, with the solid line representing the maximum node displacement of the iteration, and the dashed line representing the average displacement. The mesh was refined to a finer resolution (from 5 to 9 nodes) in the same range, by splitting each interval in half. This finer resolution converged after four iterations (9^{th} total iteration). Each fitted resolution spline is displayed in the centre plot. Further refinements were conducted through 17 to 33 nodes for the final destination spline shape in the bottom plot. The convergence plot shows a typical shape in the plot with a logarithmic ordinate scale, as it can also be observed for higher dimensional problems.

The 2-D sample on the right hand side of Fig. 7.3 was created in a similar way. 200 randomly distributed data points were computed, and random noise was added. The data

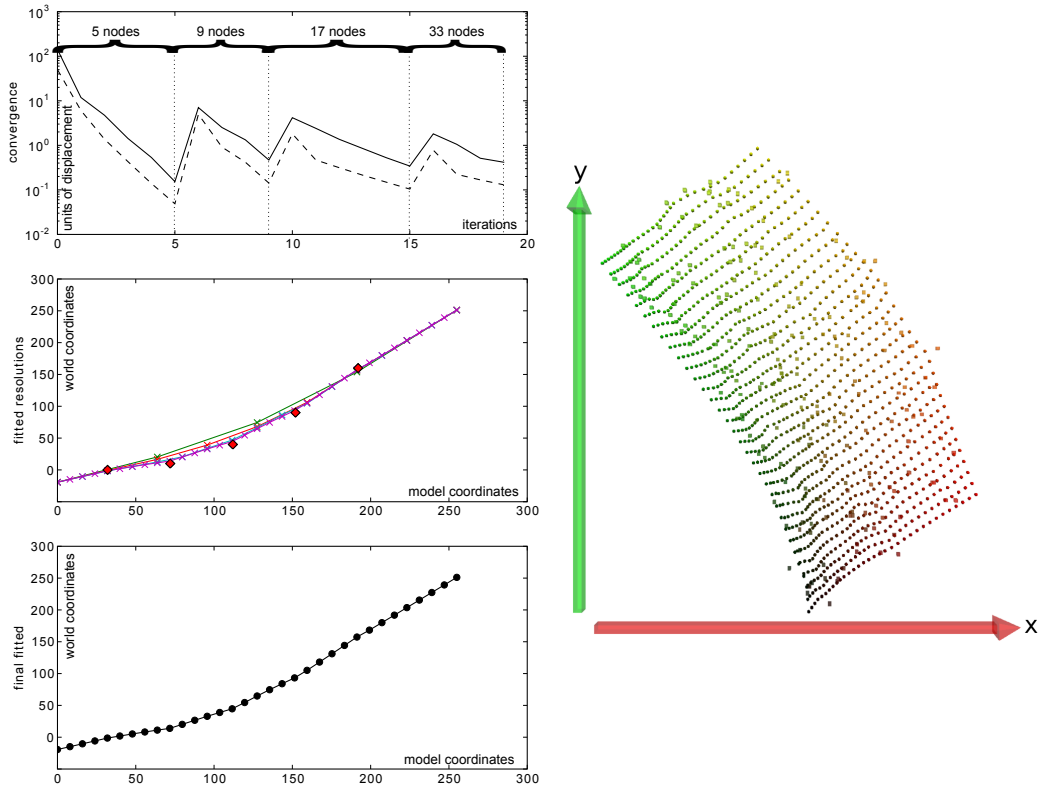

Figure 7.3: Spline interpolations in 1-D and 2-D. (left top) Maximum and average node displacement per iteration; (left middle) Converged fitted 1-D spline curves for different resolutions; (left bottom) Converged 1-D spline curve at final highest resolution. (right) Converged 2-D linear spline approximating a test function with common difficult features: rotated and shifted rectangular and uniform source data space; outer shape contains concave and convex curvatures; non-uniform node spacing.

points are rendered as little cubes, whereas the mesh nodes are rendered as little spheres. The function for computing the data points was chosen to test certain features: The rectangular and uniform source data space is rotated and shifted; the overall outer shape contains concave as well as convex curvature; and it features a non-uniform node spacing. These are all "features" that can potentially be found in ICC profile CLUTs. One can clearly see the reconstructed topography of the original function transformation. The major deviations from ideal fits are clearly due to the added (synthetic) noise of the data points, as well as to areas more sparsely sampled with data points. Balancing the relationship between the coefficients of the data springs and the second difference springs leads to a trade-off between a close fit to the (noisy) data points and a smooth shape of the spline.

Fig. 7.4 illustrates the 3-D to 3-D spline fitting for a CLUT. The data points are actual measurements from a captured test target, they are rendered in the less orderly distributed colourful cubes in the two $L^*a^*b^*$ diagrams. The two diagrams represent different stages in the multi grid fitting sequence. To better show the distribution of the data points and the

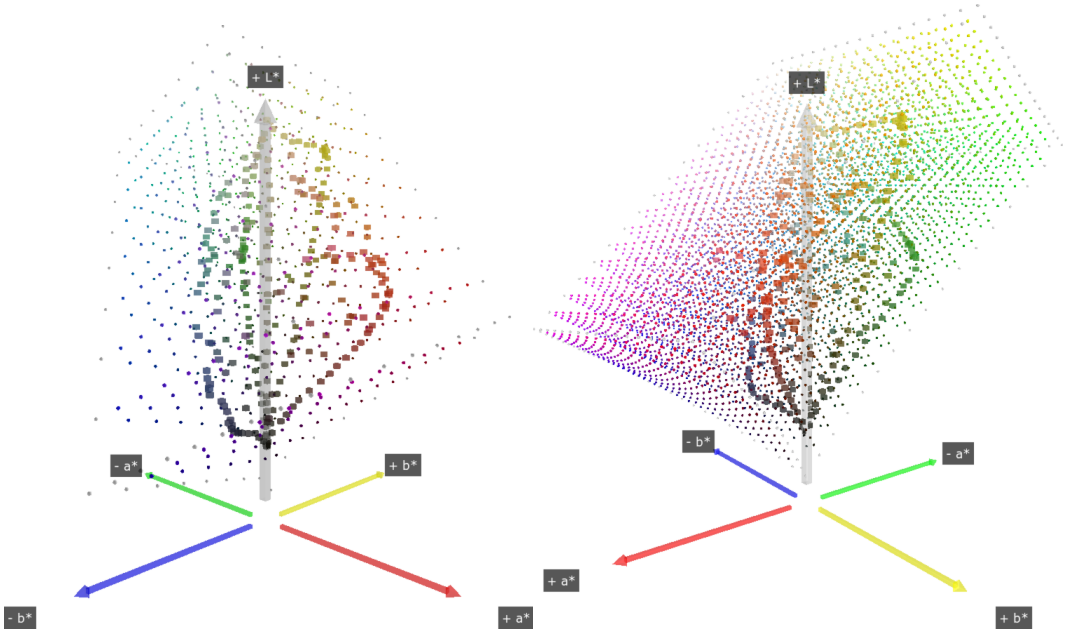

Figure 7.4: 3-D rendering of the interpolation volumes for an ICC profile. The 285 coloured blocks represent the locations of the test chart's *(Christophe Métairie Photographie DigitaL TargeT 003)* colour patches in $L^*a^*b^*$ space, the coloured spheres are the LUT node positions. (left) With 9^3 interpolation nodes and (right) at a later refined mesh with 17^3 nodes at a different angle.

shape of the distorted interpolation cubes, the two diagrams have been rendered at different angles in $L^*a^*b^*$ space. As it can be seen, the test target manufacturers use a distribution of colours to cover the neutral/grey axis as well as "chains" of colours of similar hue but with differing intensities and saturations from dark to light. The intention is to sample a volume in colour space, that is as big as possible, using only a relatively small number of colour patches.

In principle, we have discovered that the LUTs resulting from the spline fitting algorithm, as discussed in this chapter, are sufficiently good. Actually, the algorithm can even be simplified by "cutting some corners" without a degradation in quality. The first difference spring coefficients for example can be reduced to zero (0.0), effectively disabling them, and saving computational time. Furthermore, the problem can be satisfactorily solved as an LSF by discarding further iterations in the stages 2 and 3 for the NSF. Finally, the coupling was strong enough using the homogeneous second difference spring terms only (P_{xx}, P_{yy} and P_{zz}), and disregarding the diagonal second difference terms (P_{xy}, P_{yz} and P_{xz}), again speeding up calculations.

However, we could see that the tendency for divergence of the solver, as well as the convergence rate were improved by retaining weak first difference springs (k^x, k^y and k^z), in comparison to the data springs and second difference springs (which were both approximately four times as high).

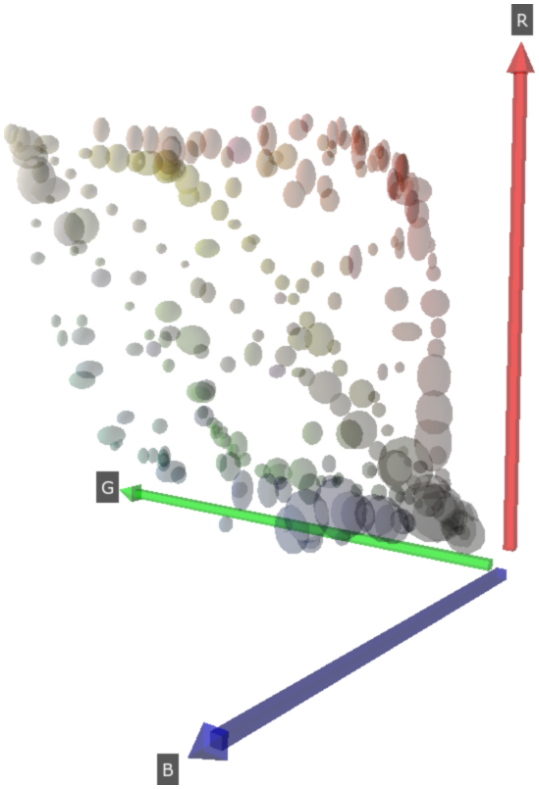

Figure 7.5: Rendering of standard deviation of data point measurements for a test chart in RGB colour space. Size of the spheroids proportional to measured standard deviations.

As information propagates much more quickly "downstream" with the iteration than "upstream," experiments with different sweeping sequences have been conducted. Unidirectional sweeps, random node sweeps as well as alternating direction (forward/backward) sweeps were tried. As already suggested by Bone [62], the alternating direction showed to be most efficient in the cases present.

7.4.2 Colour Profiling in the Presence of Measurement Noise

We felt it was sensible to adjust the data spring strength inversely proportional to the standard deviation of the data points' measurement error (standard deviation). Fig. 7.5 illustrates the distribution of measured test patches in RGB space (position in space), along with the standard deviation for each data point (size of spheroid). The measurement errors were usually highest towards the extremes of the neutral axis, in the region of the very dark and very light colours. We also see that at these points the density of measured colour patches is highest, so that a relative weakening of these high tolerance data measurements does not lead to a decreased quality of the data fit.

Fitting for this data set is already illustrated in Fig. 7.4. Fig. 7.5 displays the same data point distribution in the RGB colour space. The data set was used to compute a

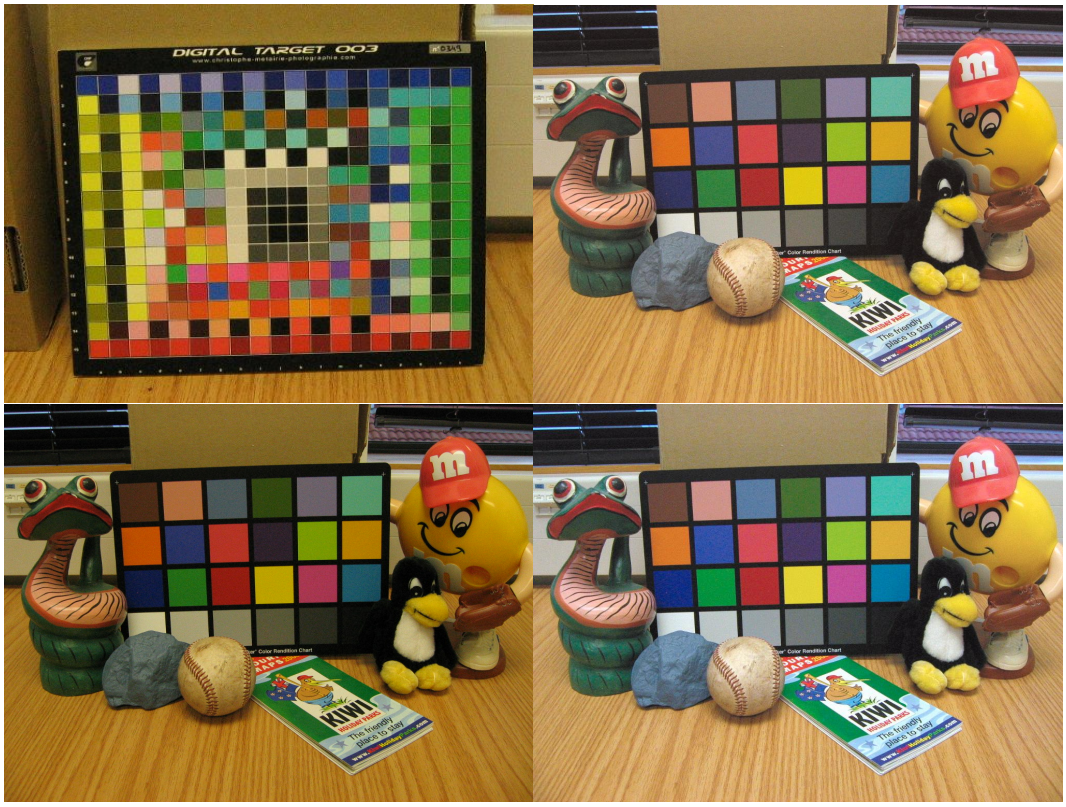

Figure 7.6: (top left) Uncorrected image captured of test chart used for computing the ICC profiles. (bottom left) Uncorrected RGB image as taken by the camera. (top right) Colour corrected image using the ICC profile by our profiler Mārama. (bottom right) Colour corrected image using an ICC profile created by Argyll CMS.

CLUT for an "AToB1" table for ICC media-relative colourimetric transformations. So far, no pre-linearisation tables on the device colour channels have been computed. The input data and results of the computed ICC profile are shown in Fig. 7.6. Measurements for the data points were extracted from the captured test chart on the top left. Our own profiling system (Mārama) was used to create an ICC profile, as well as Argyll CMS [50] to analogously compute a reference profile. A colourful sample scene was captured under identical conditions (bottom left), and our own (top right) and the reference profile (bottom right) were applied to correct for the lighting conditions and the camera's characteristics.

Ideally the two images on the right in Fig. 7.5 should be identical. However, slight differences are observable. Firstly, the image using our profile appears to expose a slight yellowish cast. This is most probably an effect caused by the poly-linear regression as explained in Sect. 7.2.3 for handling the corner point problem of the spline fitting. The corners of the colour cube that represent yellow (maximum red and green) and white (all channels at maximum) are very close together on the $L^*a^*b^*$ lightness scale L^*. When selecting data points for the regression computation these points influence each other, and some of the

Table 7.1: Average differences in colour appearance (and standard deviations of averages) to measured data points. Averages are computed over the 285 colour target patches of 13 ICC profiles.

Profiler	Mārama	Argyll reference #1 (no pre-linearisation)	Argyll reference #2 (default behaviour)
ΔE^*_{ab}	7.9 (2.4)	5.8 (1.4)	5.5 (1.2)
ΔE^*_{CMC}	6.1 (1.7)	4.3 (1.1)	4.1 (0.9)

yellow chromaticity "bleeds" into the white. The next difference is, that our profile yields slightly lower colour saturation and contrast in the mid tones. This is a common effect, that can be observed when comparing the results of (differential) image processing operating on gamma corrected vs. linear colour spaces. As our profile does not employ pre-linearisation tables, the CLUT has to "go the whole way" itself, which slightly increases inaccuracies. Introspection of the Argyll CMS created reference profile reveals that the "AToBx" tags all include pre-linearisation tables. Martí Maria, the author of the LCMS colour management system [42], has determined, that for a good precision ICC profile without pre-linearisation, a number of 48 node points per dimension in the CLUT is necessary. An equal quality can be achieved by creating a pre-linearisation table of six points with a CLUT resolution of just six nodes per dimension[1].

To get a feeling on how well the created CLUT, and therefore the ICC profile, fits the purpose of mapping the device colour space to the canonical CIE LAB colour space (the PCS), we have computed ΔE values for fitted profiles. ΔE describes the "difference in appearance" of a colour to another colour. The "E" stands for the German word "Empfindung" (sensation or perception). CIE LAB was created to be perceptually uniform, with the unit steps to be of a magnitude to be just visually noticeable. In 1976 the CIE specified the ΔE^*_{ab} as the Euclidean distance in $L^*a^*b^*$ space, as a measure of colour appearance difference. And as a "rule of thumb" a $\Delta E^*_{ab} = 1$ being just visually differentiable. But the CIE LAB space is not *exactly* visually uniform, therefore several other ΔE definitions were derived over time. One being ΔE^*_{CMC} (1:1), it is a method that has become an ISO standard *(ISO 105-J03)*. This implementation uses a lightness weight of 1.0 and a chroma weight of 1.0 for use with perceptibility data. It is more useful to determine how visually perceptilble a colour difference is [72].

ΔE^*_{ab} is useful for an estimate of a numerical fit of the fitted spline curve, contributing the largest part to the total ICC profile based transformation. For an estimate of the visual fit, ΔE^*_{CMC} can be used. We have analysed the ΔE values of all 285 colour patches (of the *Christophe Métairie Photographie DigitaL TargeT 003)* as shown in Fig. 7.6 (top left). The colour differences resulting from different ways of generating the profiles are compared in Table 7.1 against the canonical $L^*a^*b^*$ measurement values (as provided by the test target's vendor). The profiles have been generated on images captured by a *Logitech*

[1] http://www.mail-archive.com/lcms-user@lists.sourceforge.net/msg00129.html

QuickCam Pro 9000 in the illumination cases from Table 8.2 (see Chap. 8): "`27000000`", "`27270000`", "`65000000`", "`65270000`", "`65650000`", "`6565blue`", "`6565gree`", "`6565oran`", "`6565red_`", "`6565yell`", "`fluoresc`", "`Inca____`" and "`SiccDim_`". As a reference, we have generated profiles using the before mentioned Argyll CMS. Reference "1" contains a CLUT only, whereas reference "2" adheres to Argyll's default behaviour by additionally using per channel "A" and "B" curves to yield a better fit for the transformation (see Fig. 7.1). Their difference in this comparison is rather small, but it becomes much more pronounced in cases with more extreme illuminations (e. g. under exposure), as it aims to distribute the input values evenly across the LUT input indexes. In the analysed samples, this positive effect is indicated by an increase in average accuracy of $\Delta E^*_{ab} > 0.5$ for the more under exposed illumination cases, due to the effect of the "A" curves. A lack of the linearisation curves gives a better comparison to the quality of the Mārama generated profiles, as these do not employ any use of linearisation curves, either.

Due to the measurement noise (see 7.5), it is unwise to fit the spline towards minimising the ΔE too far, as that way the spline function loses its smoothness. We can see that our own profile fitting performs a bit worse than the reference profiles generated through Argyll CMS. On individual samples with very noisy measurements, it often happens that the standard deviation of the input data used for compiling these profiles is *worse* than the error of the fitted spline. So the smoothing effect of the spline computation levels out individual measurement noise to improve the overall fitted shape.

7.5 Conclusions and Future Work on Profile Generation

As a necessary precursor to understanding the whole ICC profiling technology, the significant efforts of implementing a self built profiler was undertaken. For the further portions of this research full access to the ICC profile internals was mandatory. The results of this implementation leading to the profiler "Mārama" satisfied these needs adequately.

We were able to solve the problem of computing colour device characterisations in the form of standard conformant ICC profiles. This is a non-trivial task. The ICC was formed in 1993 to standardise the ambitions of FOGRA's (German Research Organisation of the Graphic Arts) research [49] for an international audience. Since then, various applications and tool chains have been built for profiling and colour management, most of them by commercial vendors. A few implementations are available for free in open source. Most noticeable Little CMS [42], Argyll CMS [50] and SampleICC [51]. Each of these implements a subset of these features: reading, writing, manipulating and applying ICC profiles. Only one (Argyll CMS) provides the capability of a sensible profiler, which is also highly regarded by the community for producing high quality results. Argyll CMS is now in its tenth year of development. Although ICC profiles generated with our own Mārama profiler are not perfect in comparison with the Argyll CMS reference, they still are quite usable already. This is supported by a qualitative and quantitative comparison, in which the Argyll reference profiles were superior, but our own implementation of a profiling algorithm shows

competitive ambitions already. Quantitatively, the quality as determined from our current experiment is comparable to that of the measurement accuracy for individual colour patches of the test chart used.

Distinct improvements can be expected by calculating pre-linearisation shaper curves ("A" curves). These will compensate for input channels' non-linearity due to gamma corrections, and result in a better fidelity of contrast and colour saturation in the mid tones, as described in Sect. 7.4.2

Mārama is still lacking many capabilities, as well as it has some catching up to do in the field of accuracy. Further improvement can be expected from refining the spline fitting code to consider a larger number of involved spring functions when solving the equations iteratively for each node. For the second difference springs each node is involved in the terms of three nodes for each dimension. The corner point nodes are influenced by the neighbour's second difference springs. To fully include the terms of these second difference springs of the direct neighbours the scope must be extended to build up these terms looking at more than the direct neighbours, but also include certain second but next neighbours. So far, this extension has not been considered for the simplicity of the computations. However, as discussions with Graeme Gill [73] (the author of Argyll CMS [50]) have revealed, this approach would result in a solution that is independent of the need for a special treatment for the corner nodes (see Sect. 7.2.3).

It may also be interesting to compare the quality of the resulting ICC profiles towards the difference between using fixed data springs and the approach of adjusting the data spring coefficients. Currently, these spring coefficients are inversely proportional to the accuracy of the data point's measurement.

Finally, also other approaches to derive the CLUT entries may be evaluated. One of these approaches that seems worth exploring are regressions over (adaptive) enclosing neighbourhoods, as outlined by the team around Gupta [53, 58].

Chapter 8

Adaptive Colour Transformation

An ICC profile generated as described in Chap. 7 can compensate the colour perception statically for specific conditions. If these conditions change, the colour perception is subject to a systematical change. The source of this change is in the most common cases mainly a change in illumination conditions, but may also be caused to a much lesser extent e. g. by temperature changes of the measuring device (the camera). This chapter focuses on determining light-induced colour shifts from a small number of trackable colour samples (in the tens or less), and methods to compensate for them.

To get a feeling for the type of colour space distortions encountered in different light conditions, a series of experiments with different light sources were conducted as outlined in Sect. 8.1. From these insights, colour transformation compensations are derived as described with a quantitative comparison in Sect. 8.2. Information on obtaining the characteristics is discussed in Sect. 8.3, which is subsequently applied to cases involving *a priori* knowledge in Sect. 8.4, as well as cases lacking *a priori* knowledge that are taking advantage of a static perspective in Sect. 8.5.

8.1 Analysing Light-Induced Colour Shifts

In order to be able to compensate for changes in colour perception, we first had to determine how different light conditions change the appearance of colour. If we can characterise the changes sufficiently easy for a range of different illuminant types, we may be able to reduce the effort of determining an ICC profile to a hand full of parameters that need to be established, when starting with a "decent" basic characterisation of the capturing system.

In an initial setup, we have observed the shifts in 3-dimensional colour histograms in *RGB* and $L^*a^*b^*$ colour spaces, while the illumination was varied (Fig. 8.1). Under observation were the colour distribution in a 3-D histogram (Fig. 8.1 left), as well as the "colour

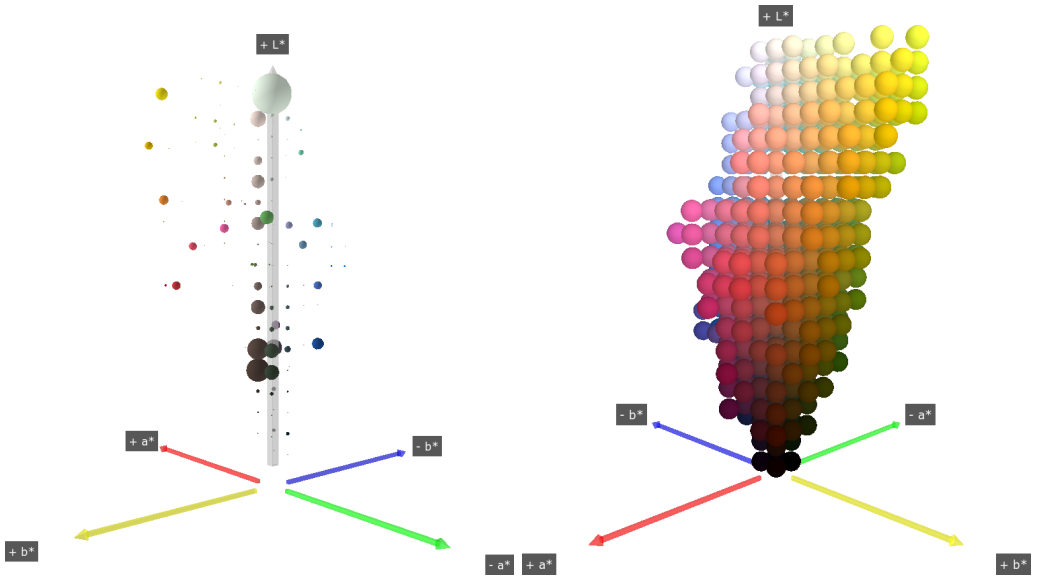

Figure 8.1: Visual analysis of colour shifts with changing light conditions by observing an indoor office environment. (left) Live 3-D histogramming in $L^*a^*b^*$ colour space. (right) "Colour footprint" of the current scene.

footprint" of a scene (every histogram bin that was not empty was marked in the 3-D diagram, Fig. 8.1 right). It was expected – as well as observed – that the distribution of colours in the histogram underwent systematic changes depending on the illuminant variations. The results, as shown in the figure, however were practically not easily useful in obtaining significant data that would be helpful in identifying the shape or magnitude of a colour space distortion.

To obtain a good understanding of the colour shifts encountered, we conducted a series of tests. In these tests, a test chart (as used for ICC profiling, see Chap. 5) was captured by a camera under varying light conditions. For this experiment, we used three different cameras (for settings see Tab. 8.1):

- *Canon PowerShot S50* digital still camera (contains a 5.0 M-pixel 1/1.8 inch CCD,
- *Logitech QuickCam Pro 9000* USB web cam (contains 2.0 M-pixel colour CMOS sensor with a 3.7 mm focal length lens),
- *UniBrain Fire-i BCL 1.2* IEEE 1394 colour web cam with the standard lens (contains 1/4 inch colour Sony ICX-098BQ CCD sensor and 4.3 mm focal length standard lens without IR coating);

as well as three different test targets:

- *Wolf Faust camera calibration Target C1* (IT 8.7 layout),

- *Christophe Métairie Photographie DigitaL TargeT 003* (the "old version"),
- *X-Rite GretagMacbeth 24 patch ColorChecker* (20 × 29 cm size version)

in 18 different illumination cases (see Tab. 8.2).

The illuminants mentioned in Tab. 8.2 consist of combinations of an assortment of various common commercially available sources and coloured clear transparencies (for filters), which were available free of charge in a local "junk room." Except for the white LEDs, all illuminants are powered by 240 V, 50 Hz AC power straight from the room's power net.

- Two different fluorescent energy saver light bulbs, namely the 24 W *Philips Tornado Energy saver Cool daylight* and 24 W *Philips Tornado Energy saver Warm white* (nominally 6500 K and 2700 K colour temperature respectively).
- Incandescent 100 W *Philips Floodline* frosted bulb with 80° reflector.
- Incandescent 250 W *Osram Siccatherm IR1* frosted heating lamp.
- A set of two *Optek Lednium Series 828-OVTL09LG3WD Daylight white* arrays of nine white LEDs operating at reduced power (nominally 5800 ± 600 K colour temperature).
- The room's standard light fixing, consisting of ceiling mounted *Philips TL-D 58W/840 "New Generation" Cool White* fluorescent tubes (nominally 4000 K colour temperature).
- Ambient "office daylight" from open windows, with some minor content of fluorescent tube light (that could not be shut off).
- A set of five clear transparency covers over one light bulb only, in the colours: red, orange, yellow, green and blue.

Fig. 8.2 shows the geometric arrangement used for the sampled image series of the different illumination cases. The two web cams (UniBrain Fire-i BCL and Logitech QuickCam Pro 9000, Fig. 8.2 right) have been fixed in position on a locked swing arm that allowed a reproducible positioning of the cameras through the whole series. The Canon PowerShot S50 still camera was mounted on a tripod in a precisely marked position (not shown in figure). A fixture was used to position the colour test charts in the same position at the same angle for every captured frame. Lastly, the illuminants were mounted at an angle and distance that allowed a homogeneous illumination of the scene without causing any specularities or glare within the captured scene (all windows in the room were completely darkened). Only the two cases using office illumination (ceiling mounted tubes) and daylight (two windows in the room) occurred at a different angle, however still homogeneous and diffuse without specularities or glare to not influence measurement results negatively. In the image on the left of Fig. 8.2 four energy saver light sources (two of 6500 K and 2700 K each) are mounted (illumination case "`65652727`"). Before each light was used for sampling, sufficient time was

Table 8.1: Camera settings for colour shift samples series.

Camera	**Setting**	**Value**
Canon PowerShot S50	White Balance	Daylight
	ISO Speed	100
	Effect	off
	BKT mode	off
	Flash	off
	Shutter speed	1/125 s
	Aperture	F 4.0
	Macro mode	on
	Resolution	1600 × 1200
Logitech QuickCam Pro 9000	Brightness	121
	Gain	37
	Contrast	25
	White Balance Temperature, Auto	0
	Power Line Frequency	1
	Sharpness	224
	Backlight Compensation	1
	Exposure, Auto	0
	Exposure, Absolute	266
	Exposure, Auto Priority	0
	Frame Rate	5.0
	Saturation	32
	White Balance, Temperature	4003
	Resolution	960 × 720
UniBrain Fire-i BCL	Brightness	False, 337
	Auto Exposure	False, 422
	Shutter	3
	Gain	37
	Timeout	1
	Sharpness	80
	Gamma	1
	Frame Rate	3.75
	White Balance Mode	False
	White Balance U	91
	Wite Balance V	69
	Saturation	90
	Resolution	640 × 480

given for warm up of the lamps, to ensure constant conditions for all different cameras and

Table 8.2: Description of illuminant/filter combinations for illumination cases.

Illuminant Code	Light Source Combination
`27000000`	one 2700 K energy saver lamp
`27270000`	two 2700 K energy saver lamps
`65000000`	one 6500 K energy saver lamp
`65270000`	one 6500 K + one 2700 K energy saver lamp
`65650000`	two 6500 K energy saver lamps
`65652727`	two 6500 K + two 2700 K energy saver lamps
`6565blue`	one 6500 K + one 6500 K energy saver lamp w/ blue filter
`6565gree`	one 6500 K + one 6500 K energy saver lamp w/ green filter
`6565oran`	one 6500 K + one 6500 K energy saver lamp w/ orange filter
`6565red_`	one 6500 K + one 6500 K energy saver lamp w/ red filter
`6565yell`	one 6500 K + one 6500 K energy saver lamp w/ yellow filter
`fluoresc`	fluorescent light tube
`Inca____`	standard incandescent light bulb
`IncaSicc`	standard incandescent + thermal incandescent light bulb
`LED_____`	two white LEDs at reduced power
`SiccBrit`	thermal incandescent light bulb (close)
`SiccDim_`	thermal incandescent light bulb (distant)
`daylight`	ambient office daylight with some minor fluorescent tube content

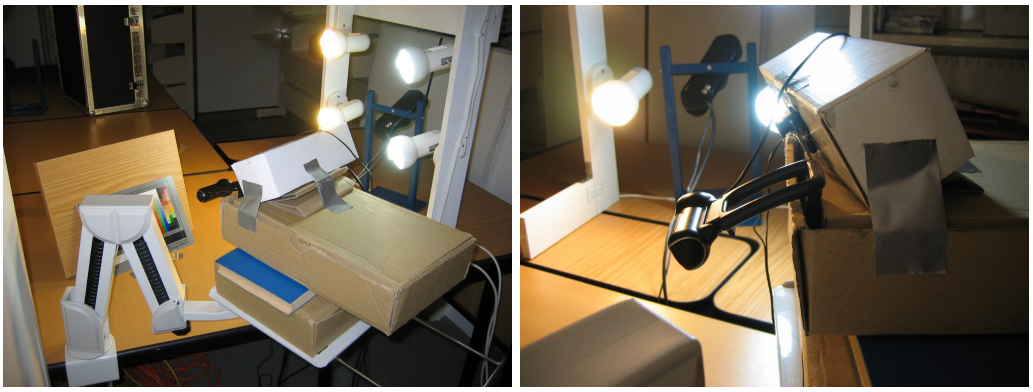

Figure 8.2: Setup of sampling assembly for the colour shift sample analysis series. (left) Geometric arrangement of the colour test chart, light fixture (for the case "`65652727`") and cameras. (right) The cameras are mounted into a fixed position to ensure equal an capturing arrangement for all illumination cases.

test charts.

The white points of all used illuminants are listed in Tab. 8.3. These relative white points have been determined by analysing the appearance of the white patch of the test charts after applying the ICC profile transformation determined for the reference case ("`65270000`").

Table 8.3: Relative white points of used illuminants in $L^*a^*b^*$, determined by analysing the appearance of the test chart's white sample. The chart's colours have first been colour corrected with the ICC profile of the reference ("`65270000`").

Illuminant Code	White Point
`27000000`	$L^*a^*b^*$ = (83.6, 9.6, 38.2)
`27270000`	$L^*a^*b^*$ = (100.2, -4.6, 33.5)
`65000000`	$L^*a^*b^*$ = (82.5, -16.1, -19.1)
`65270000`	$L^*a^*b^*$ = (100.0, 0.5, 0.6)
`65650000`	$L^*a^*b^*$ = (99.9, -10.1, -8.1)
`65652727`	$L^*a^*b^*$ = (100.4, -5.3, -0.4)
`6565blue`	$L^*a^*b^*$ = (85.1, -13.2, -33.9)
`6565gree`	$L^*a^*b^*$ = (84.0, -18.0, -19.1)
`6565oran`	$L^*a^*b^*$ = (92.4, -8.6, -9.6)
`6565red_`	$L^*a^*b^*$ = (86.8, -6.0, -18.9)
`6565yell`	$L^*a^*b^*$ = (93.2, -19.4, 2.3)
`fluoresc`	$L^*a^*b^*$ = (96.2, -1.7, 7.5)
`Inca____`	$L^*a^*b^*$ = (98.2, 2.9, 23.3)
`SiccDim_`	$L^*a^*b^*$ = (99.5, -1.4, 21.2)
`LED_____`	$L^*a^*b^*$ = (52.5, -3.7, -32.8)

Therefore, shifts in the white point can be determined by comparison with the white point of the reference case. These white points are used throughout this chapter for all white point adaptations using a Bradford transformation for a chromatic adaptation transformation (CAT). The illuminants "`IncaSicc`", "`SiccBrit`" and "`daylight`" were captured too bright for the given (constant) camera settings. Therefore image pixel values in the bright (white) regions were clipped and the white point could not be determined with satisfactory accuracy for further analyses. For this reason analysis results from these have been omitted from discussions of results following in this chapter.

In these experiments, the systematic shift in the $L^*a^*b^*$ colour space was quite clearly visible. We used the illumination case "`65270000`" as a reference, as it featured a somewhat medium light intensity within the series of samples, and it exposed an intermediate white point within the field of illuminants. Some typical and meaningful samples of this analysis are shown in Figs. 8.3 and 8.4. To get a better feeling for the spatiality of the plot, each case has been depicted from two different perspectives: A perspective side view, as well as a top-down view onto the a^*b^* plane. It is clearly visible, that the colour shifts indicated through coloured vector arrows follow the general directions of the anticipated light changes. The single 6500 K energy saver lamp would be darker and more "bluish" than the combination of the 6500 K with the 2700 K energy saver lamps; the two 6500 K lamps would be similar in intensity, but more "bluish" than the reference; and finally, the fluorescent room light with nominally 4000 K colour temperature would have a somewhat comparable colour

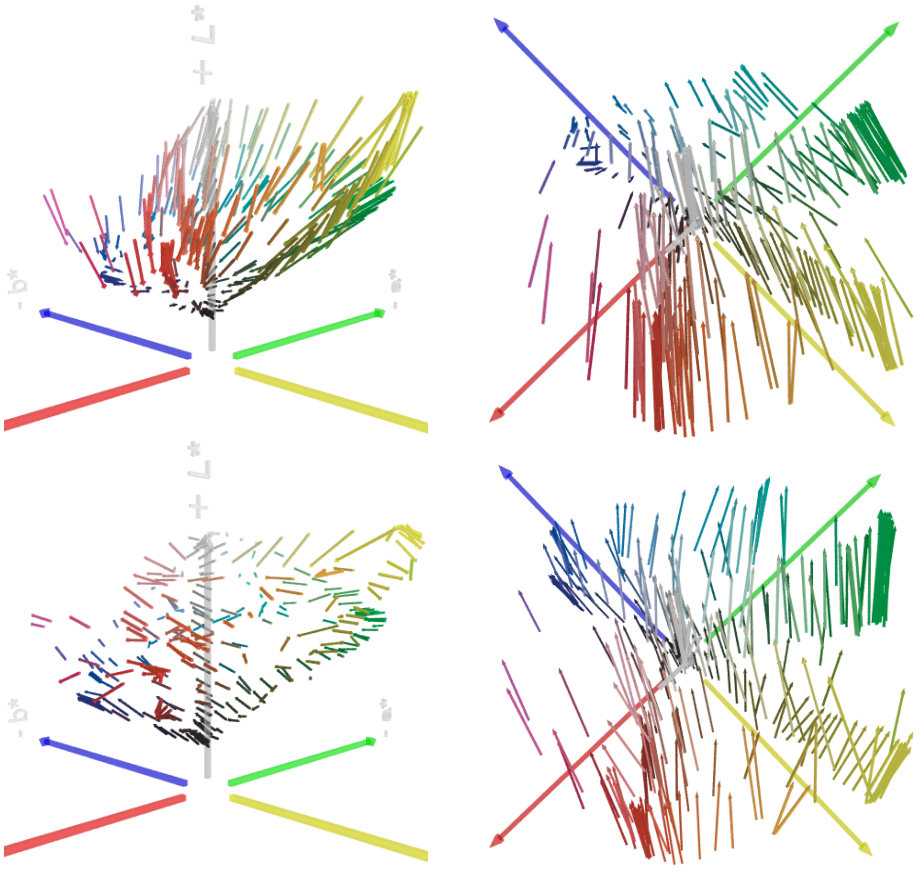

Figure 8.3: Shift vector field of the 285 colour patches relative to ICC profiled reference conditions ("65270000") in $L^*a^*b^*$ colour space. (left column) Perspective view on $L^*a^*b^*$ colour space. (right column) View along the L^* axis onto the a^*b^* plane of colour space. Illumination cases darker and more "bluish" ("65000000", top) and similar intensity and more "bluish" ("65650000", bottom). Note: Some of the axis labels may be clipped in the figures.

temperature as the reference consisting of 6500 K and 2700 K lamps.

These general shifts are one phenomenon that can be observed. Beyond that, also scaling effects like a "focus point" for the shifts (Fig. 8.3 top in the "blues" and Fig. 8.4 bottom in the "reds") as well as rotation (Fig. 8.4 bottom right) could be observed. Due to the nature of very different spectral characteristics of the different illuminants, further colour space distortions are not just possible, but expected. But these are visually not easily detectable.

8.2 Compensation For Colour Shifts

Many forms of compensation for colour shifts can be imagined up and implemented. But just as it is the fact in obtaining an ICC profile in the first place (see Chap. 7), it is important

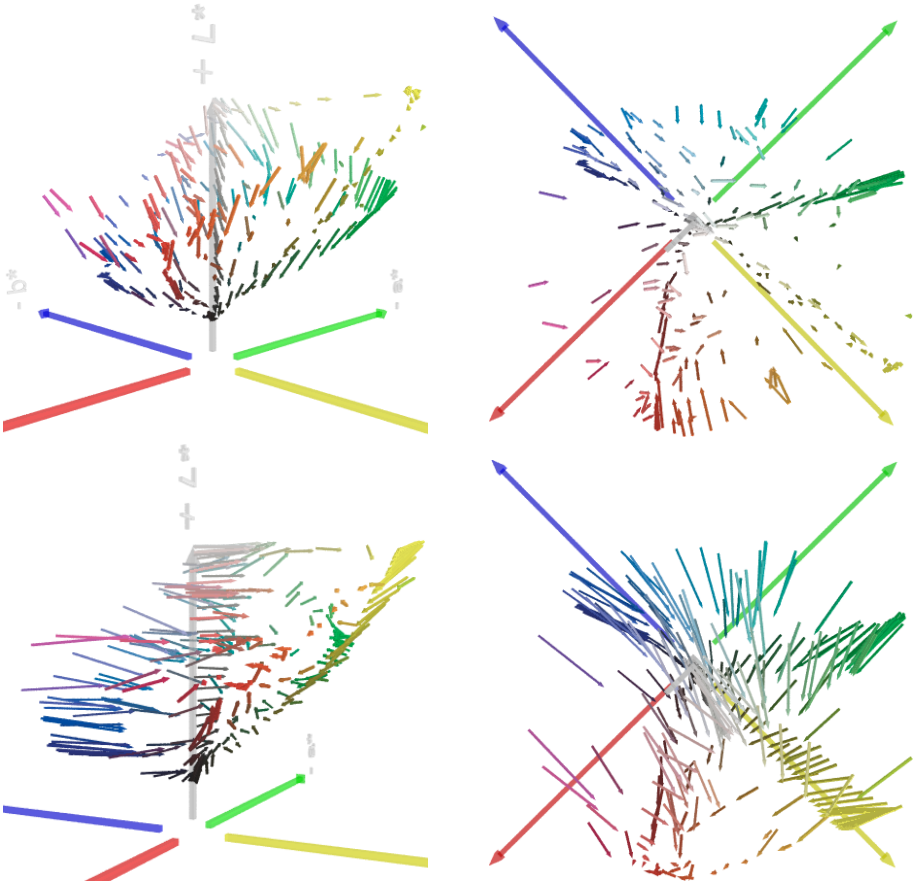

Figure 8.4: Same reference ("`65270000`") and layout as in Fig. 8.3 slightly darker light source with similar colour ("`fluoresc`", top) and similar intense, but incandescent rather than fluorescent illuminant ("`Inca____`", bottom).

to compensate for the systematic changes only, and not for individual "outliers." This keeps the transformation "smooth."

If one can find a good estimate of the white point, a Chromatic Adaptation Transformation (CAT), as outlined in Chap. 4 (e. g. through a Bradford transformation), can be applied. This adaptation works well in cases where just the white point and intensity shifts. In cases with significant influences of more complex types of distortions, this approach will fail. In engineering, a common approach is to tackle such problems by approximation through a Taylor series. The series is terminated upon reaching a sufficiently good approximation. In many cases, the series is terminated after the first (linear) term, yielding a 1^{st} order Taylor approximation [74]. This often accounts already for the largest individual contribution. The analysis of the cases in the previous Sect. 8.1 shows that many of the colour shifts seem to obey, at least loosely, the properties of affine transformations. White point compensation through a CAT is in fact a special case of an affine transformation. In a similar way as ICC profile colour lookup tables (CLUTs, see Chap. 7) are created, also compensating transfor-

mations of almost arbitrary complexity can be created. This approach is discussed in [53,54] for the purpose of learning colour enhancements from a (small) set of colour pairs, and encode them into an ICC profile. In this section, we are mainly analysing and discussing the effectiveness of (combinations of) affine transformations, applied to different colour spaces, in order to compensate for the light induced colour shifts.

As we have discovered in Chap. 4, it makes quite a difference in which colour space an adaptation transformation is applied. We have already seen that a *Bradford transformation* is best suited for a white point adaptation, so we are not deviating from it. But for the affine transformation, we have got also the choice of various colour spaces to determine and apply colour corrections on. Beyond the $L^*a^*b^*$ colour space, we can first transform the colours to a *cone response domain* to apply the transformations. There we have got the choice of performing transformations in the style of *von Kries*, *XYZ scaling* ("wrong" von Kries transformation) as well as a *Bradford like* transformation. The latter three only differ in their particulars of the conversion from XYZ colours to the cone response domain (using a transformation matrix with different coefficients). The "Bradford like" cone response domain is obtained using the transformation matrix used for the Bradford transformation to convert to/from the XYZ colour space. For CATs today, the Bradford transformation is considered to be the best solution, and XYZ scaling is prescribed in the official standard for ICC profile based colour transformations. Therefore, we have reduced the comparison to affine transformations based on $L^*a^*b^*$ colours, a Bradford like colour space, as well as XYZ scaling. All these three have been applied with and without a prior white point adaptation.

Next, we are describing how one obtains a suitable affine transformation through a regression computation. In Sect. 8.2.2, we are first going to compare quantitatively the mean errors of colour shifts after the transformation, and then the improvement in the vector field of the colour shifts after applying the most promising affine transformation in Sect. 8.2.3. Finally in Sect. 8.2.4, we will have a look at the visual improvement by applying the affine transformations to the images that were used in this series.

8.2.1 Obtaining the Affine Transformation

We are using a set of reference colours and a set of current colours. We are seeking the best affine transformation that will transform the current colours to match the reference colours with the least total error. Each colour can be (depending on colour space and colour system) described as a vector, which has got (in the cases of $L^*a^*b^*$ or XYZ colours) three components. The problem can be solved through a multi-dimensional linear regression computation suitable for $\mathbb{R}^3$ space. This is a general problem, and the solution has been abstracted to work (theoretically) in any $\mathbb{R}^n$ space, it is already described in Sect. 7.2.3. In this regression computation the vector sets P represents in the set of current colours, and Q represents the set of reference colours. All further computations are performed exactly in the same way to obtain a solution.

8.2.2 Quantitative Evaluation

Tab. 8.4 uses all 285 colour patches of the Christophe Métairie Photographie DigitaL TargeT 003 test target to find the best match of a colour transformation. All the images under comparison in the table have been captured using the Logitech QuickCam Pro 9000 (USB web cam) camera. Again, as a reference the case "`65270000`" (as in Sect. 8.1) is used. The table states the average ΔE^*_{ab} values and their standard deviation to the expected reference colours after applying the affine colour transformation in the designated colour space. The affine transformation has been determined by computation of a least error square sums based linear regression calculation as described in Sect. 7.2.3 and [70].

The following types of corrective colour transformations were applied:

- Optionally a chromatic adaptation transformation (a Bradford CAT) up front.
- An affine corrective colour transformation applied to one of these three colour spaces:
  - "Bradford like" colour space – The colour tuples have been transformed into a cone response domain as used for the Bradford CAT.
  - CIE LAB – The affine transformation was applied directly to $L^*a^*b^*$ colour tuples.
  - CIE XYZ – The affine transformation was applied to XYZ colour tuples as used for an "XYZ scaling" CAT.

The initial idea was, that an adaptation of the white point (through a Bradford CAT) would compensate for a large amount of the encountered colour shifts already. A following affine transformation then could compensate for further colour distortions. As in CATs a variety of different cone response domains is used for the adaptation, we have decided to sample a variety of colour spaces for the application of an affine transformation as well. The candidates were to use straight without further conversion the $L^*a^*b^*$ space (as it was a successful approach in Chap. 4), as well as a "Bradford like" cone response space as the current best practice for CATs as well as the XYZ space (as prescribed by the ICC for CATs).

The values in Tab. 8.4 were obtained by using different combinations of these operations. Looking more closely at the values charted, some things become obvious:

- The set of ΔE^*_{ab} colour differences for each transformation space is (almost) the same, no matter whether a prior white balancing transformation was applied or not.
- The average ΔE^*_{ab} values of conversations performed in a cone response like domain are (almost) all the same (XYZ scaling or transformation on Bradford like space).
- The results with an affine transformation are (almost) all better than a CAT alone, and the transformation in $L^*a^*b^*$ colour space are all better than the transformations applied in a cone response like domain (XYZ scaling or transformation on Bradford like space).

Table 8.4: Adaptation of colour tuples under 14 different illuminants against one reference illuminant ("65270000"). The adaptations were performed using a best fit of an affine transformation with and without a CAT (using the Bradford transformation).

affine transf. using		no CAT			CAT	with CAT		
		Bradf. like	$L^*a^*b^*$	XYZ	only	Bradf. like	$L^*a^*b^*$	XYZ
27000000	avg. ΔE^*_{ab}	12.3	10.6	12.3	16.9	12.0	10.5	12.0
	std. dev.	5.7	6.2	5.7	10.5	5.9	6.1	5.9
27270000	avg. ΔE^*_{ab}	11.7	9.9	11.7	15.2	11.7	9.9	11.7
	std. dev.	6.4	6.2	6.4	7.5	6.4	6.2	6.4
65000000	avg. ΔE^*_{ab}	9.0	7.6	9.0	9.9	8.9	7.5	8.9
	std. dev.	4.4	3.7	4.4	4.7	4.4	3.7	4.4
65270000		— *reference* —						
65650000	avg. ΔE^*_{ab}	10.3	6.7	10.3	14.9	10.2	6.7	10.2
	std. dev.	7.8	3.5	7.8	6.1	7.8	3.5	7.8
65652727	avg. ΔE^*_{ab}	10.6	6.5	10.6	12.3	10.6	6.5	10.6
	std. dev.	7.5	3.6	7.5	4.9	7.4	3.7	7.4
6565blue	avg. ΔE^*_{ab}	11.3	7.5	11.3	13.3	11.3	7.4	11.3
	std. dev.	6.8	3.2	6.8	6.4	6.8	3.1	6.8
6565gree	avg. ΔE^*_{ab}	9.1	7.5	9.1	9.9	9.0	7.5	9.0
	std. dev.	4.7	3.9	4.7	5.2	4.7	3.9	4.7
6565oran	avg. ΔE^*_{ab}	6.1	5.5	6.1	7.2	6.0	5.4	6.0
	std. dev.	3.0	3.0	3.0	3.6	3.0	3.0	3.0
6565red_	avg. ΔE^*_{ab}	7.5	6.6	7.5	8.5	7.3	6.5	7.3
	std. dev.	3.6	3.4	3.6	3.8	3.5	3.4	3.5
6565yell	avg. ΔE^*_{ab}	5.9	5.2	5.9	7.3	5.9	5.1	5.9
	std. dev.	2.9	2.8	2.9	3.8	2.9	2.8	2.9
fluoresc	avg. ΔE^*_{ab}	7.8	4.9	7.8	6.0	7.6	4.8	7.6
	std. dev.	4.6	2.4	4.6	3.9	4.4	2.4	4.4
Inca____	avg. ΔE^*_{ab}	8.5	6.0	8.5	12.9	8.4	6.0	8.4
	std. dev.	4.4	2.6	4.4	6.5	4.4	2.6	4.4
SiccDim_	avg. ΔE^*_{ab}	9.8	7.0	9.8	14.1	9.7	7.0	9.7
	std. dev.	6.2	3.1	6.2	6.0	6.1	3.1	6.1
LED_____	avg. ΔE^*_{ab}	14.4	11.8	14.4	19.6	13.9	11.6	13.9
	std. dev.	6.6	5.3	6.6	8.1	6.2	5.1	6.2

To substantiate this impression, average values over the tabulated illumination cases have been computed (Tab. 8.5). As the LED illuminant was comparably too dark, and introduces clipping in the low lightness pixel values, the case "LED_____" has been eliminated from the averaging. Looking at the best fitting affine transformations (applied to $L^*a^*b^*$ colour space), we can see that in average the colour shift error is only about 60 % of the magnitude compared to white point compensation only cases. In many cases, the overall average deviation decreases even to values smaller than half of that error.

We can only detect differences of some significance between white balancing only, an affine transformation performed in a cone response like domain (XYZ scaling and Bradford like transformation), and an affine transformation applied to colours in the $L^*a^*b^*$ space. The similarity of a the cases with and without white balancing can be explained by the fact

Table 8.5: Averages and standard deviations of transformation results for all data points and all illuminants (excl. "LED_____") from Tab. 8.4.

Transformation	**avg.**	ΔE^*_{ab}
CAT only	11.4	(5.6)
Bradford like and XYZ scaling (with CAT)	9.1	(5.2)
Bradford like and XYZ scaling (without CAT)	9.2	(5.2)
$L^*a^*b^*$ scaling (with CAT)	7.0	(3.6)
$L^*a^*b^*$ scaling (without CAT)	7.0	(3.7)

that a white point adaptation is also an affine transformation, so it can be "encapsulated" within the determined affine transformation.

As to the second observation, a white point adaptation performed on XYZ is in practise very similar to one performed using a Bradford transformation. They are not ideally identical, but the shift in white points commonly only results in a commonly minor difference in chromaticity. After all, the ICC *specifies* XYZ scaling as the official method for white point adaptation. In our present cases, with otherwise high errors and standard deviations, this minor error does not result in a significant magnitude.

The third observed point is rather simple to explain. White balancing only compensates the chromaticity towards a corrected white point. Any subsequent affine transformation is expected to improve the adaptation accuracy. As we are using ΔE^*_{ab} as a quality measure (within the $L^*a^*b^*$ colour space), a minimisation of the least square sum error in that space will obviously lead to a better fit of the affine transformation in $L^*a^*b^*$ colour space as well.

For practical purposes, it is unrealistic to have hundreds of measurement points available that can be used to determine an affine transformation. It is commonly quite easy to determine a white and a black point (see Chap. 4), as done when using colour constancy algorithms. If one may be able to detect at least two more distinctly coloured objects/areas (for the case of an $\mathbb{R}^3$ space) within a scene, one can derive an affine transformation as needed for this purpose. Due to involved measurement error terms this minimum however leads to rather unstable transformation, resulting in colour shifting artefacts, especially as one cannot assume that these additional colour points are well distributed over the colour gamut. To get a feeling of feasibility, we have conducted a similar computation as above (which is using all patches), but only using six specified patches. The six patches were the aforementioned white and black, as well as fairly saturated colours along the red, green, blue and yellow axes of the $L^*a^*b^*$ colour space. Tab. 8.6 lists (equivalently to Tab. 8.4) the determined colour difference evaluations.

The analysis reveals no obvious difference to the previous one using all colour patches, except for increased errors in colour adaptation quality. Again, omitting the LED illuminant, we have determined mean colour deviations for the different cases (Tab. 8.7). The strongly over-determined regression computation used to find the affine transformation of Tab. 8.4

Table 8.6: Adaptation of colour tuples under 14 different illuminants against one reference illuminant ("65270000"). The adaptations were performed using a best fit using six colour patches (white, black, red, green, blue and yellow) of an affine transformation with and without a CAT using the Bradford transformation.

affine transf. using		no CAT			CAT	with CAT		
		Bradf. like	$L^*a^*b^*$	XYZ	only	Bradf. like	$L^*a^*b^*$	XYZ
27000000	avg. ΔE^*_{ab}	15.7	11.5	15.7	16.9	15.7	11.4	15.7
	std. dev.	8.3	7.3	8.3	10.5	8.3	7.3	8.3
27270000	avg. ΔE^*_{ab}	15.0	12.0	15.0	15.2	15.0	12.0	15.0
	std. dev.	9.8	8.8	9.8	7.5	9.7	8.8	9.7
65000000	avg. ΔE^*_{ab}	10.4	8.2	10.4	9.9	10.4	8.2	10.4
	std. dev.	5.9	3.9	5.9	4.7	5.9	3.9	5.9
65270000		— *reference* —						
65650000	avg. ΔE^*_{ab}	15.0	7.8	15.0	14.9	15.0	7.8	15.0
	std. dev.	11.0	4.0	11.0	6.1	11.0	4.0	11.0
65652727	avg. ΔE^*_{ab}	14.7	7.6	14.7	12.3	14.6	7.7	14.6
	std. dev.	9.0	3.8	9.0	4.9	9.0	3.8	9.0
6565blue	avg. ΔE^*_{ab}	13.9	8.0	13.9	13.3	13.8	8.0	13.8
	std. dev.	9.5	3.3	9.5	6.4	9.4	3.3	9.4
6565gree	avg. ΔE^*_{ab}	11.0	8.0	11.0	9.9	10.9	7.9	10.9
	std. dev.	6.9	3.9	6.9	5.2	6.8	3.8	6.8
6565oran	avg. ΔE^*_{ab}	6.9	6.0	6.9	7.2	6.8	6.0	6.8
	std. dev.	3.0	3.2	3.0	3.6	3.0	3.2	3.0
6565red_	avg. ΔE^*_{ab}	7.9	7.4	7.9	8.5	7.9	7.4	7.9
	std. dev.	4.0	3.7	4.0	3.8	4.0	3.7	4.0
6565yell	avg. ΔE^*_{ab}	6.9	5.8	6.9	7.3	6.9	5.8	6.9
	std. dev.	4.0	3.1	4.0	3.8	3.9	3.1	3.9
fluoresc	avg. ΔE^*_{ab}	10.3	5.3	10.3	6.0	10.2	5.3	10.2
	std. dev.	6.3	2.9	6.3	3.9	6.2	2.9	6.2
Inca____	avg. ΔE^*_{ab}	12.7	8.0	12.7	12.9	12.7	8.0	12.7
	std. dev.	6.8	3.3	6.8	6.5	6.8	3.3	6.8
SiccDim_	avg. ΔE^*_{ab}	13.2	8.5	13.2	14.1	13.1	8.5	13.1
	std. dev.	7.8	4.8	7.8	6.0	7.7	4.8	7.7
LED_____	avg. ΔE^*_{ab}	18.8	12.9	18.8	19.6	18.7	12.7	18.7
	std. dev.	13.4	6.3	13.4	8.1	13.4	6.2	13.4

and 8.5 is in average between one and three ΔE^*_{ab} units better than the one computed using only six patches from Tab. 8.6 and 8.7. As a measure of reference, one unit of ΔE^*_{ab} is said to be roughly the quantity a human eye can determine as a minimum colour difference between two colours.

8.2.3 Evaluation of the Shift Vector Field

The illustrated cases of the shift vector field of the 285 colour patches from Figs. 8.3 and 8.4 have been "re-visited" after applying the affine transformations as described in Tab. 8.4. For brevity only one of the "best match candidates" – affine transformation applied in $L^*a^*b^*$ colour space with no white point adaptation – is illustrated in Figs. 8.5 and 8.6. It is quite

Table 8.7: Averages and standard deviations of transformation results for all illuminants (excl. "LED_____") from Tab. 8.6, but using only six data points.

Transformation	**avg.**	ΔE^*_{ab}
CAT only	11.4	(5.6)
Bradford like and XYZ scaling (with CAT)	11.8	(7.0)
Bradford like and XYZ scaling (without CAT)	11.8	(7.1)
$L^*a^*b^*$ scaling (with CAT)	8.0	(4.3)
$L^*a^*b^*$ scaling (without CAT)	8.0	(4.3)

clear to see that also some of the very strong colour shifts have been met by the correction. The overall magnitude of the colour shifts has decreased significantly due to the corrective transformation. Certain areas however still show distinctly large systematic shifts. After all, the affine transformation can only meet the needs of a first approximation to the highly non-linear deformation through linear corrections. These could be met by higher order corrective terms that need to be determined.

A Bradford transformation for adapting to a different white point is designed to keep neutral colours "in balance" without introducing any undesired shift. The same is not true for the affine transformations as used in this chapter. The transformations have been determined by minimising the overall error. Depending on the non-linearity of a colour space distortion introduced through the illumination shift a minimisation of the total error may lead to neutral colours moving away from the L^* axis in $L^*a^*b^*$. This effect was observable for some samples of the series. Fig. 8.7 illustrates the case "27270000" that shows a shift of neutral colours towards green/yellow quite well in the shift vector field diagram.

Undesirable colour distortions of this kind can be met e. g. by putting a higher relative weight onto colour samples with a low saturation (that are located closer to the L^* axis) than those with a higher saturation. Another possibility is to choose a higher number of colour samples with a low chroma to increase their relative weight.

8.2.4 Visual Image Evaluation

In this evaluation we have again used the illumination case "65270000" as a reference. After a corrective transformation, the image appearance should more closely resemble the one of the reference as illustrated in Fig. 8.8. We have applied all corrective colour transformation as listed in Tab. 8.4 to the images of all illumination cases.

For brevity reasons, only the four cases used in the other evaluations from the previous sections are illustrated. We have also reduced the illustrated samples to those undergoing the two most promising transformations (the ones that apply an affine transformation on the $L^*a^*b^*$ colour space, with and without prior white point adaptation). In the quantitative analysis of Sect. 8.2.2, these came out as equally well suited by our means of analysis used in that section.

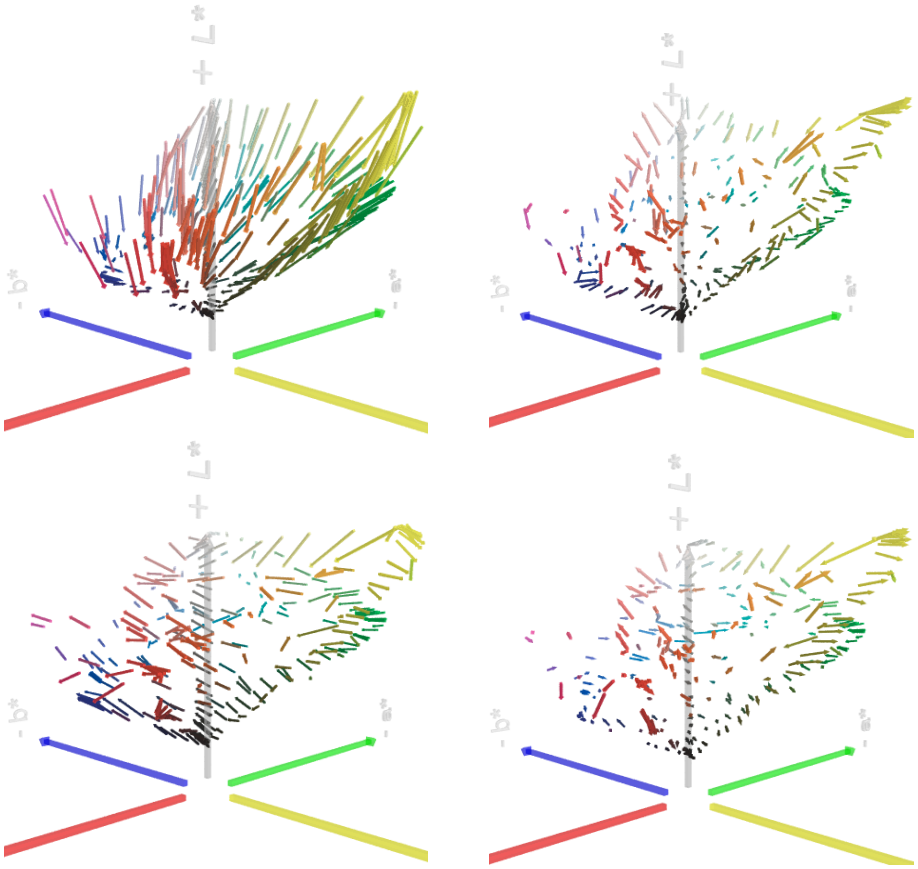

Figure 8.5: Shift vector field of the 285 colour patches relative to ICC profiled reference conditions ("65270000") in $L^*a^*b^*$ colour space. (left column) Uncorrected colour appearance. (right column) Colour appearance after applying an affine transformation in $L^*a^*b^*$ colour space derived from all patches with no prior white point adaptation. The illumination cases are the same as in Fig. 8.3: "65000000" (top) and "65650000" (bottom).

Fig. 8.9 illustrates the uncorrected images of the illumination cases (left column) to the corrected versions to the right next to them. We are comparing these corrected images to the reference image from Fig. 8.8. Mathematically, we have seen that the two corrections with and without a prior white point adaptation perform equally well, but visually we can see that the images still retain some of the colour cast of the illuminant. It seems like the affine transformation operating on a different colour space, compared to the white point adaptation (performed using a Bradford transformation), does not perform well. We can see, that the blue cast of the illuminants "65000000" and "65650000" (top two rows) is to a slighter extent still present in the middle column, whereas the transformation without the white point adaptation on the very right much more closely resembles the appearance of the reference "65270000" of Fig. 8.8. Similarly, the yellow cast of sample "Inca____" in the last row is mostly removed in the corrected sample on the right.

Of the series of compared colour corrections in the image series of Fig. 8.9, it seems

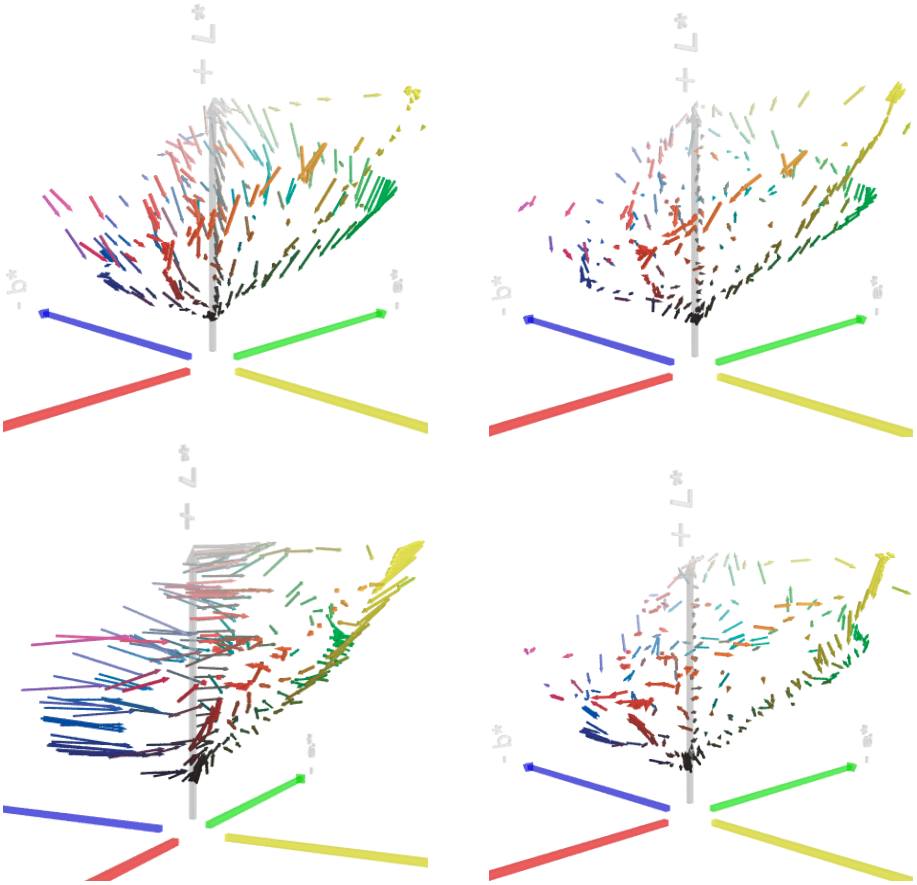

Figure 8.6: Same reference ("`65270000`") and layout as in Fig. 8.5. The illumination cases are the same as in Fig. 8.4: "`fluoresc`" (top) and "`Inca____`" (bottom).

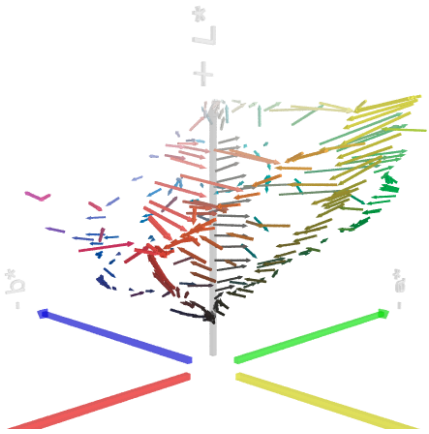

Figure 8.7: Colour correction against the same reference ("`65270000`") from Fig. 8.5, but for the illumination case "`27270000`".

like the correction of the sample "`Inca____`" performs worst. In this comparison series of

Figure 8.8: Reference image "`65270000`".

illuminants, the reference as well as the three other comparison cases all used fluorescent light sources (different combination of fluorescent energy saver lamps and fluorescent tubes), which expose similarities in the characteristics of their spectral power distributions (SPD). The case "`Inca____`" in contrast results from an incandescent light source, which features a very smooth, well behaved curve in the SPD. These differences result in non-linear shifts of colour appearance, for which the affine transformation cannot compensate, well resulting in the residual colour shift that can be observed. This shows the limitations of a correction using affine transformations: They work well for cases in which the illumination shifts, but when the illumination characteristics in the SPD vary, their perform will suffer.

8.3 Determining Colour Shifts

Previously in this chapter, we have been dealing with images containing structured data that could be processed automatically using hundreds of trackable colour patches, extracting their colour, and operating on them. In the real world, one would hardly ever be able to face conditions like these. To determine suitable affine transformations, one must identify a minimum of four "colour points," which can be mapped from a reference scene to a current scene. One suitable point that is easy to find in most scenes is through white point estimation, as it is used in colour constancy (see Chap. 4), and in a similar fashion a black point can be found. This leaves a minimum of two further colour points to determine or track. Obviously, these points must be linear independent, so that the linear equation system for determining the affine transformation is solvable. To prevent undesirably bad results caused by measurement errors, these should be quite "far apart" from each other in the colour space.

Beyond this necessity of a minimum number of colour points, we have seen in Sect. 8.2.2 that over-determined systems result in a more accurate and robust fit. Therefore, it is desirable to find as many usable colour points as possible within a scene as a base for computing an affine mapping transformation.

A way to find these colour points is to analyse the scene and extract various types of colour hints. The method for finding these hints can be highly domain specific. In certain

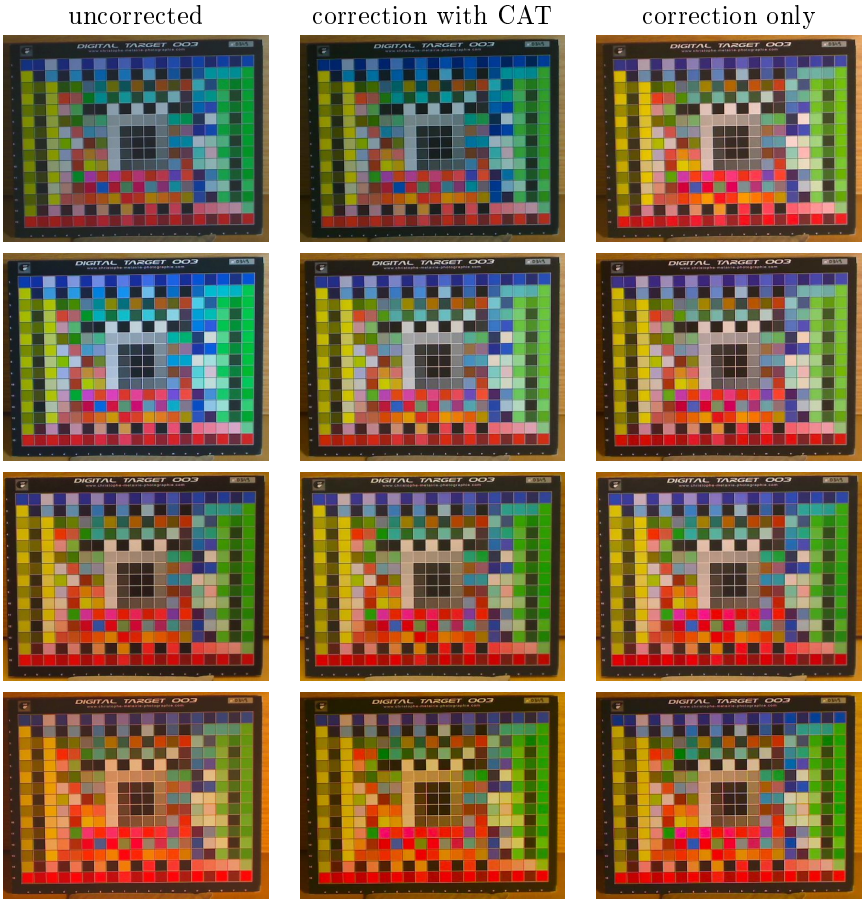

Figure 8.9: Appearance of the 285 colour patches with various compensation transformations towards normalising the image against the reference ("`65270000`"). (left column) Uncorrected colour appearance. (centre column) Colour appearance after applying an affine transformation in $L^*a^*b^*$ colour space derived from all patches with prior white point adaptation. (right column) Similar to centre column, but without a prior white point adaptation. Colour correction works clearly better without the prior CAT. Illumination cases are the same as in Fig. 8.3 for the top two rows ("`65000000`" and "`65650000`"), and the same as in Fig. 8.4 for the bottom two rows ("`fluoresc`" and "`Inca____`").

cases, one can *expect* to be able to identify reoccurring objects or markings in a scene. Cases like this can be found for example in robot soccer [32, 75] in game field markings, the distinctly coloured ball, the colour markings of participating robots, as well as other obvious hints from the environment. In other domains these could be signs or markings, as they can be found in traffic or factory environments. Of course, colour hints can also be placed with intent within the field of view of a camera within the scene to be analysed.

In the absence of distinct hints involving *a priori* knowledge, the problem becomes more difficult. The following lists a few methods that may each, or in combination, be helpful in

determining additional hints.

- Finding (and potentially tracking) colour point with "extreme" $L^*C^*h^*_{ab}$ values, those of maximum saturations C^* for different hues and maximum/minimum lightness L^*.
- Segmenting the survey area, and tracking colours for the individual segments.
- Tracking only colour changes of segments for distinct hue segments (e. g. six different segments in h^*_{ab}).
- In case of changing scene composition, segment the scene into an (unchanged) background and a (changing) foreground. Focus only on changes in the static background.

In the following two sections, the focus is on simple approaches to segment colour samples within the scene. The aim of this segmentation of the survey area is to track colours within distinct image areas (e. g. of objects). Following in Sect. 8.4, we are assuming feedback from a scene segmentation in the robot soccer domain. The colour appearance of indicator patches on top of the robots are already available through tracking the objects, and this data can be used directly for a mapping towards a reference appearance. Sect. 8.5 in contrast takes a different approach. It is based to some extent on the colour constancy approach employed in the *Grey World Assumption* (GWA). According the GWA any scene containing a sufficiently complex colour composition will in average tend towards a neutral grey. But we are applying it not to the complete scene, but to small (rectangular) segments of it only. Therefore, we can track the changes of a characteristic "tint" of a small area rather than the neutral grey of the whole scene.

8.4 *A priori* Knowledge Based Correction

In this analysis, we will focus on employing *a priori* knowledge for corrective colour adaptations. The sample used is taken from the domain of robot soccer. In some of the leagues, the soccer robots do not act autonomously, but are radio controlled units, guided by a remote computer system. This system acquires its input from overhead cameras, viewing down onto the playing field, and identifying the individual robots and objects on the field by distinct colour markings. Therefore, the capability of differentiating between the objects in the field is partly dependent on the robustness of the colour detection.

It was not easy to facilitate the capability of adjusting reproducibly a large range of light intensities in our robotics lab. Therefore, we used a setting in the camera's API to vary the shutter speed, to simulate a wide range of light intensity change from strongly under to strongly over-exposed capturing. An exposure time of 20 ms resulted in good results. Further images were acquired with settings of 3, 5, and 10 ms for under-exposed images, and 30, 40, 50, 60, 70 and 80 ms for over-exposed images. In the example data set, we have used the existent diffuse fluorescent office illumination without any "daylight influences."

For the experiments, the same card stock as used for marking the robots is present in the scenes. The samples in Fig. 8.10 are arranged repeatedly in nine zones of the playing field. Colour information of these samples is averaged in the indicated areas (red frames). The redundant arrangement has been chosen for reasons of the slightly inhomogeneously illuminated scene, as it can be seen particularly well in the blue stock samples, as illustrated in the right image of the figure (red frames).

8.4.1 Initial Camera Characterisation/ICC Profiling

If image pixels reach either end of the dynamic range of the camera in its current settings, they are "clipped," and assume a value of either 255 or zero. If pixel values for channels are clipped, their chromaticity is beyond the detectable range of the camera, and the use of such pixels leads to distorted results. The reference setting therefore needs to be chosen in a way to contain a minimum of clipped pixels to be of adequate use. In this case, a shutter speed of 20 ms resulted in such good intermediate results. This setting has been used as a referenc,e and was used to determine the ICC profile from the information given through the test target placed in the scene.

Fig. 8.10 illustrates the scene under analysis. The 24 patch *GretagMacbeth ColorChecker* has been placed in the scene (yellow frame in left image). Its colour values are used for computing the initial characterisation to an ICC profile. The also seen *DigitalTargeT 003* has not been used for this purpose, as the colour readings at the resolution of the camera were not adequate for the purpose. The profile is computed using the open source profiler *Argyll CMS* [50]. This *input profile* is applied to every frame captured by the camera using the open source colour management system *Little CMS* [42, 44] to normalise the representation.

8.4.2 Colour Adaptation

The initial colour characterisation describes a static colour correction process. Due to the course of a running session however the illumination conditions may change. Bystanders may shadow more or less secondary light sources, indirect daylight through open windows may change its quality due to clouds, etc.

To derive the adaptive colour transformation for different light conditions we now have to determine pairs of colour tuples, mapping a colour for the reference frame (20 ms shutter speed) to a corresponding colour in a frame captured with different light/exposure setting. Commonly found objects are tracked, and a unique identification of objects can be performed at all times to detect their current colour appearance. For this experiment we kept the scene static and read the colour of the colour markers from their respective constant locations. The five colour markers, in addition to a white and black point that usually can be estimated quite easily in scenes (through the approaches from colour constancy [1, 2]), are therefore used for the determination of the corrective affine colour transformation in $L^*a^*b^*$ colour

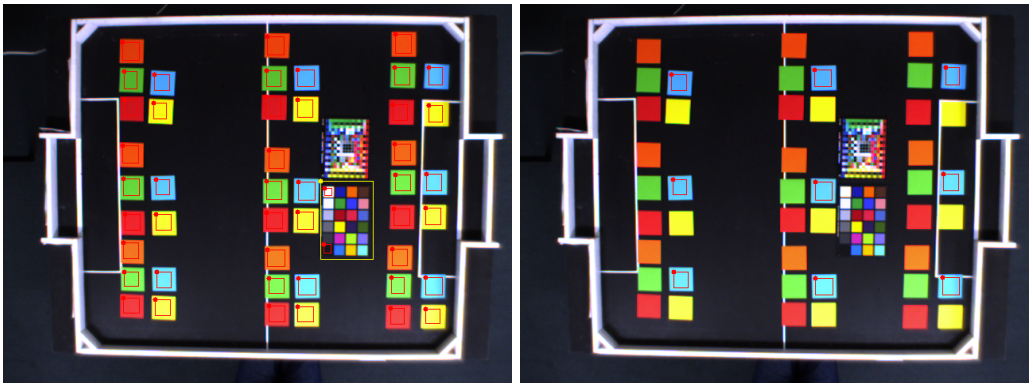

Figure 8.10: The reference frame at a shutter speed of 20 ms after ICC based correction. The ICC profile was computed through characterisation using the 24 patch GretagMacbeth ColorChecker present in the image. The five different colour samples occurring repeatedly are of the same coloured card stock as used for manufacturing our colour markers of the soccer robots. (left) Indication of all seven colour areas used to compute the corrective affine transformation. (right) Only the areas of the blue colours for averaging to determine an affine transformation.

space. For better reproducibility, we chose to use the black and white colour samples of the GretagMacbeth ColorChecker instead of using an alternative algorithm for determining the white and black points analytically. Fig. 8.10 shows the areas of all colour markers by red boxes in the left hand image, and a selection just of the blue markers in the right image. Area averaged colour value are used for all indicated colours.

As described in Sect. 8.2, we must have $n \geq 4$ colour pairs to determine an affine transformation in $\mathbb{R}^3$. Given the five colour samples plus the white and black point yields a total of seven colour samples. So this approach should be still robust enough in case of a failure in detecting a few samples.

All images are first statically colour corrected against the reference profile obtained with the 20 ms exposure case to obtain an $L^*a^*b^*$ image. Then, the affine colour transformation is computed by using the seven identified colour values (red, orange, yellow, blue and green of the card stock, as well as white and black on the test chart of Fig. 8.10) against their reference values. This affine colour correction is applied to the image. Colour difference metrics are extracted, and a corrected colour image as well as the images' colour data in $L^*C^*h^*_{ab}$ colour space ($L^*a^*b^*$ colour expressed in polar coordinates) is passed on to the colour segmentation algorithm used in the robot soccer application (see Sect. 8.4.4).

For comparison, an equivalent process has been conducted *without* applying the affine transformation. Yellow exposes a very high lightness, and therefore tends to be subject to pixel clipping in its channels earlier than others. For that reason, we have also conducted an analogue test run including the affine transformation above, but using all colours except for yellow.

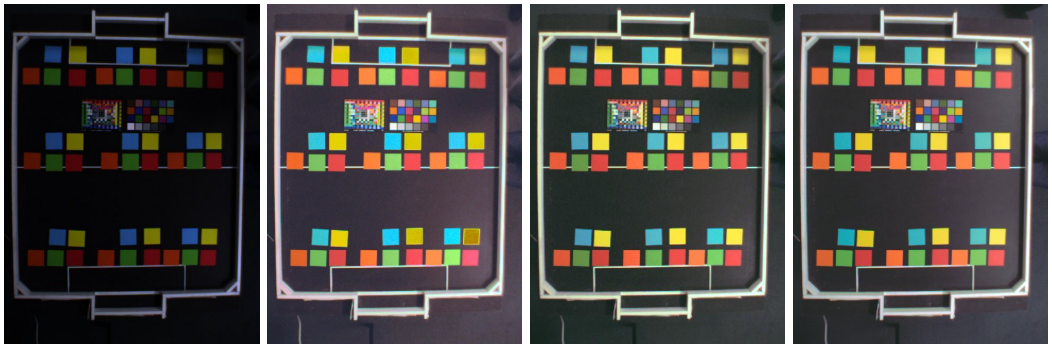

Figure 8.11: The individual steps of the colour correction. (left) Frame as captured from the camera. (second) After application of the reference ICC profile. (third) After application of the corrective affine transformation. (right) The reference image (also colour corrected).

8.4.3 Colour Adaptation Results

First of all, we are going to have a visual inspection of one of the samples through the various steps of colour correction (Fig. 8.11) for the under-exposed sample with an exposure time of 10 ms. Quantitative colour information is very important, but can be at times deceptive towards the true accuracy of the correction (compare numerical results from Sect. 8.2.2 with visual results in Sect. 8.2.4).

As one can see comparing the right two images in the figure, the dark base board of the field has got a slightly different colour tint, but the colour patches fit quite well visually. The mean ΔE^*_{ab} value to the reference over the seven colour patches has dropped from 24 (second image) to 6.5 (third image) after the corrective transformation. A ΔE^*_{ab} value of one is generally said to be visually *just* distinguishable, if the coloured areas are directly adjacent.

To get a feeling for the effectiveness of the different corrective transformations, we have plotted the obtained ΔE^*_{ab} values for the different colour corrections against the camera's exposure time (Fig. 8.12). An exposure of 20 ms was used as a reference, the colour difference there is by definition equal to zero. The solid lines illustrate the change in the mean colour difference for all seven colour patches under scrutiny, whereas the dotted lines indicate the maximum and minimum respective values of the set.

Some observations become obvious when studying the diagram:

- If no colour correction is applied, the mean and maximum errors are lower for over-exposed images. In contrary for any of the applied colour corrections, the error becomes lower – or effectiveness of the correction better – for under-exposed images.
- Starting from shutter speeds of around 40–50 ms, significant pixel clipping starts to happen within the areas of our colour samples. The chromaticity/saturation of the yellow patches starts to converge towards the white point, which limits the maximum error, and the dotted curves take a bend and "flatten out" towards an upper bound.

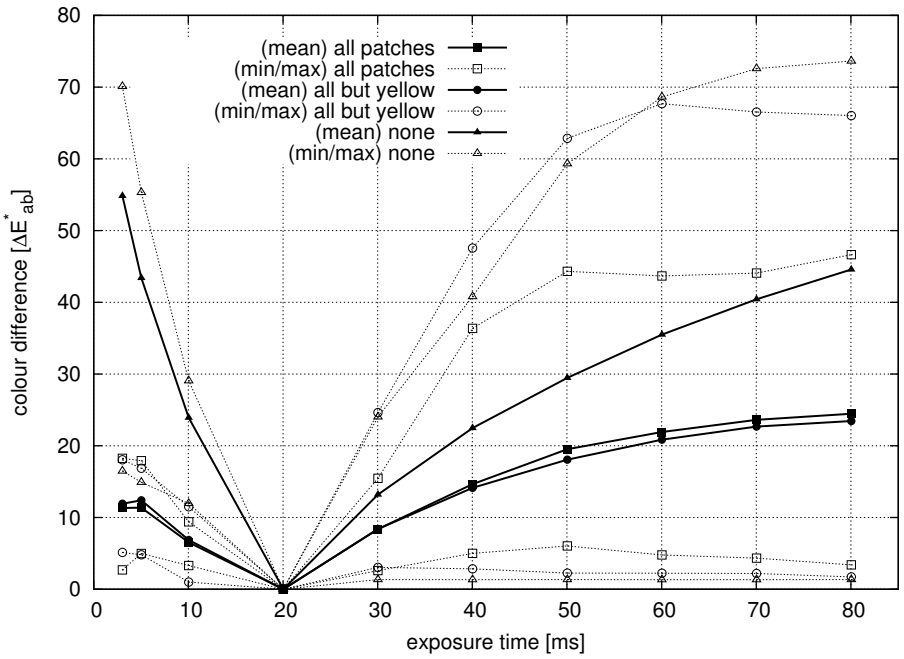

Figure 8.12: Plot of the linear colour difference measure ΔE^*_{ab} of the colour patches used for defining the transformation. The mean (solid lines) is plotted against the camera's shutter speeds. Minimum and maximum colour differences (dotted lines) are shown to indicate the range of spread. The sets of three curves are determined by using different numbers of colour patches in obtaining an affine transformation: all seven colours, all colours except for yellow, and no corrective transformation.

- The influence between using the patch set with and without yellow on the average is minute. Of course the accuracy of yellow colours will be lower when omitting that patch, but the other colours with low ΔE^*_{ab} values will gain between 2–5 in accuracy.

If an application demands a certain colour detection quality, the workable range of the camera exposure and tolerable light conditions can be significantly extended by using an adaptive colour correction. If the demand for robust colour separation can for example be given to a threshold of $\Delta E^*_{ab} < 10$, we can estimate the acceptable exposure range. Using no dynamic adaptive corrections, the range (intersection with the $\Delta E^*_{ab} = 10$ line) is approximately between 16–28 ms (12 ms wide). Using dynamic correction, that range increases to approximately 7–32 ms (25 ms wide). Acceptable exposure tolerances therefore could be more than doubled.

Finally, we might gain slight further improvements in colour correction quality by discarding colour patches as they start to show a certain level of pixel clipping. This would result in hybrid results, choosing the better approach towards the selection of patches to use for computing the affine transformation.

8.4.4 Colour Segmentation

The goal of applying corrective colour adaptation to camera captured images is to enhance the robustness of colour segmentation, as it is used in the robot soccer domain. Therefore, a series of colour segmentation tests were performed to compare the effectiveness. Traditionally, these analyses are performed on rg-chromaticities, computed directly from the (linearised) RGB colour tuples. The $L^*a^*b^*$ colour space is visually linearised already, and it contains the chromaticity already in its a^*b^* plane by disregarding the L^* component. To test our prognosis of increased robustness of the overall system, we have conducted four colour segmentation experiments, for colour corrected and uncorrected images, each operating on rg- and a^*b^*-chromaticity respectively.

The colour segmentation is based on a colour classification, discriminating boosted colour channels in a hue/saturation segmented chromaticity plane. Chromaticities are represented in hue/saturation tuples. As the colours to identify are fixed and known, an initial training phase is conducted, in which classifier parameters are determined. These parameters consist of two things: Firstly the lower and upper boundaries in the two chromaticity channels (hue and saturation) for each colour. Secondly, the parameters for "colour boosting" operations. These boosting operations have the effect of forcing chromaticity values of desired colour samples more closely into the designated segmentation areas, while at the same time dispersing negative samples out of these areas. This "Colour Contrast Fusion" is a concept developed and successfully applied by Reyes and Shin [75, 76]. The colour segmentation can therefore be strictly applied to the colour tuples only, without consideration of spacial characteristics, thus yielding a very fast pixel classification.

In the previous colour adaptation, we have computed an image representation also in the $L^*C^*h^*_{ab}$ colour space (Sect. 8.4.2). $L^*C^*h^*_{ab}$ is a polar representation of $L^*a^*b^*$, so we can use C^* and h^*_{ab} directly in place of hue and saturation from the RGB colour space processing. Boosting and classification parameters are determined using a genetic algorithm. The process can therefore be separated into two distinct steps: training of the parameters and matching of the colour samples. For the training, an empirically determined fitness function (8.1) is used. Its goal is to increase the number of true positive pixels by penalising false positives strongly. This fitness function is a combination of sigmoid functions to balance rewards for detected true positive detections n_t, while penalising strongly false positives n_f. The input values f_t and f_f are normalised with the total number of pixels processed n_total. This fitness function is illustrated in the plot of Fig. 8.13. The result of this training is a set of optimally adjusted parameters for the colour boosting and optimal boundaries for the

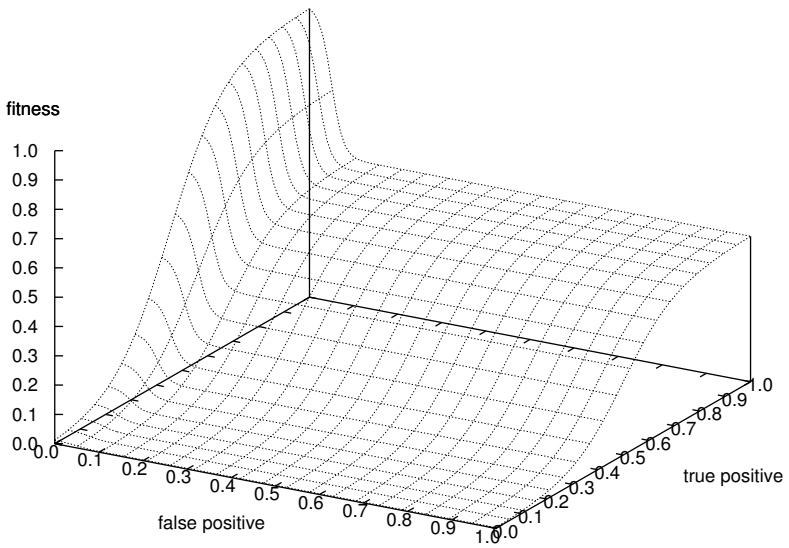

Figure 8.13: Fitness score function used for determining the colour classification parameters in the genetic algorithm. This function is also used for the evaluation of the colour correction in Figs. 8.14 and 8.15.

colour pixel classification without the need for manual intervention.

$$
\begin{aligned}
f_\mathrm{t} &= \frac{n_\mathrm{t}}{n_\mathrm{total}} \\
f_\mathrm{f} &= \frac{n_\mathrm{f}}{n_\mathrm{total}} \\
f(f_\mathrm{t}, f_\mathrm{f}) &= \frac{1}{2}\left[\left(e^{-10\,(f_\mathrm{f}-0.5)}\right)^{-1} + \left(1-\left(1+e^{-75\,(f_\mathrm{t}-0.05)}\right)^{-1}\right)\left(1+e^{-10\,(f_\mathrm{f}-0.4)}\right)^{-1}\right]
\end{aligned}
\tag{8.1}
$$

Fig. 8.14 illustrates the quality of the results after the training using a genetic algorithm. The results assume that an ICC profile for each exposure time has been determined, and colours are adapted as good as possible. The training is performed for each of these exposure cases individually to its best. The purpose behind this is to determine the overall quality and maximum capability towards extracting information from these images.

Only the scores for the corrected image in a^*b^* space exposes significant improvements over its counterpart for uncorrected colour. We can see that the variation ranges in a^*b^*-chromaticity are higher than for rg-chromaticity. For this, two explanations can be found: Firstly, the classification is based on a genetic algorithm, which is somewhat subject to a random distribution, which might cause values sometimes to behave more erroneously than

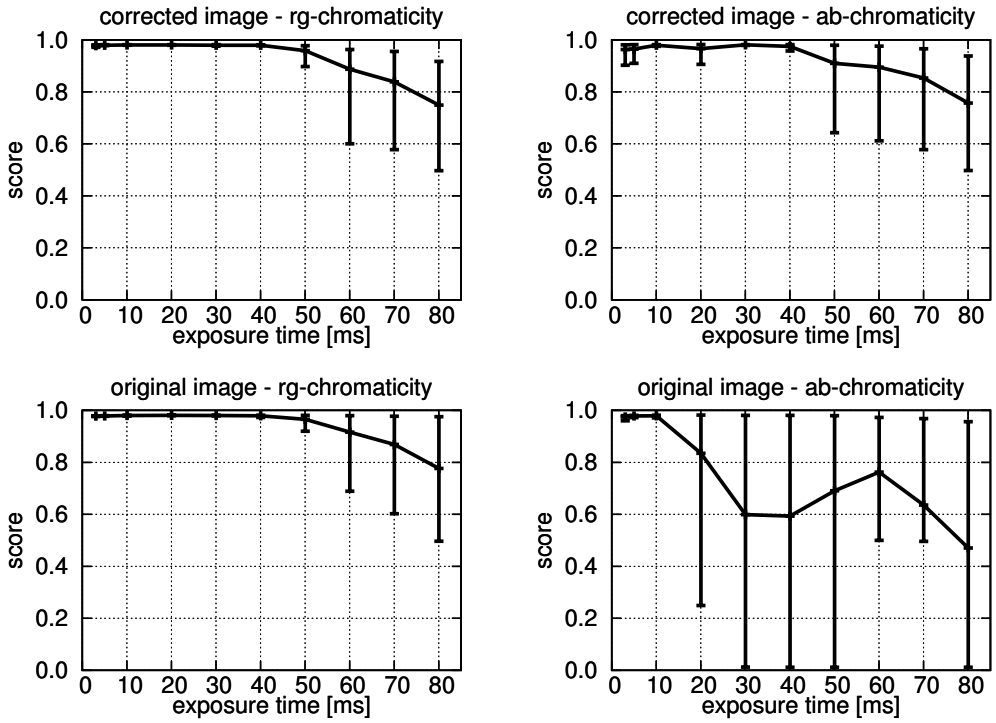

Figure 8.14: Fitness scores of the trained classifiers plotted against the camera's shutter speeds. The solid curves show the mean value over the five colour samples placed within the image, the vertical bars indicate the range from minimal to maximal score for the samples. (top row) Results using colour corrected input images. (bottom row) For comparison results using the raw camera images. (left column) Training performed on rg-chromaticity values. (right column) Training performed on a^*b^*chromaticity values.

at other times. But the main reason is to expect is, that the rg-chromaticity gamut is only a fraction of the size of the a^*b^* chromaticity plane. In fact the sRGB colour space is only 821,000 ΔE^3 units in volume, whereas the $L^*a^*b^*$ profile connection space is 4,110,000 ΔE^3 units in size [77]. The colour classification algorithm however uses the same relative "granularity" over the whole area, which resolves the "footprint" of actually used colours much more coarsely. It is expected that some tuning to the algorithm towards the used colour space would largely resolve these issues as already discussed in Sect. 4.6.2.

The results from Fig. 8.14 therefore provide an idea of the theoretically best achievable boundary for the matching of colours in the subsequent analysis. Fig. 8.15 shows the resulting scores of colour matching using the classifier determined for the case of a 20 ms shutter speed. This classifier is applied for all exposure settings alike.

In this case, the matching scores around the reference point are very high, and they drop off the greater the distance of the exposure time from the reference point is. This is expected and obvious for all four conducted trial runs. Much more significant is the fact, that the curves are supported much wider at a higher level in the neighbourhood of the reference

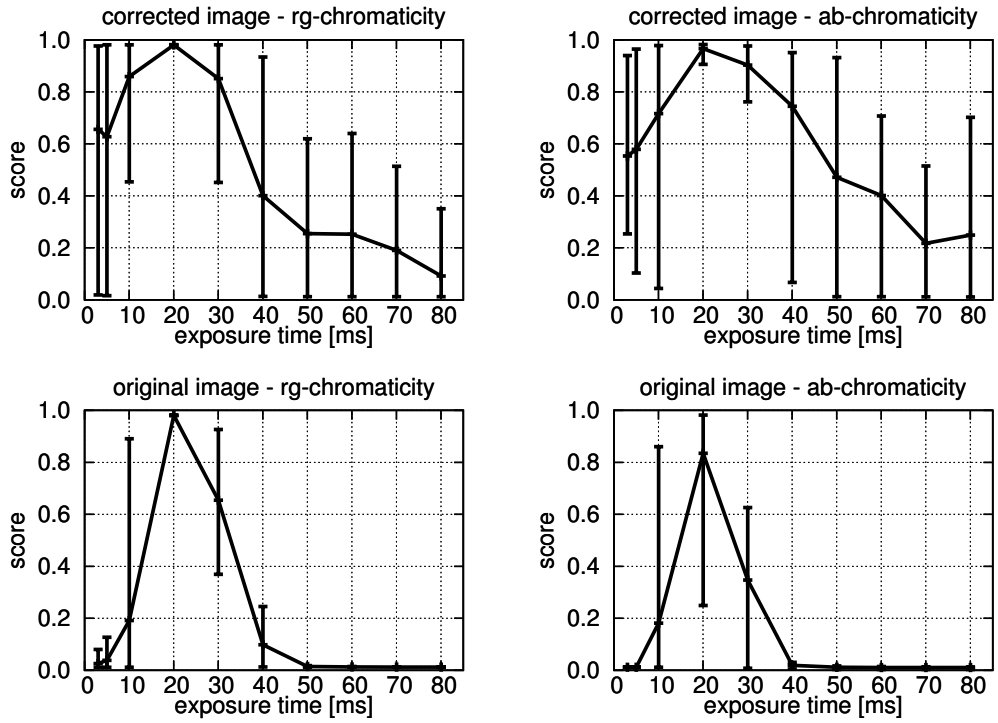

Figure 8.15: Fitness scores of the matching performance using the reference classifier for exposure time 20 ms. Computation cases for the four images as in Fig. 8.14.

point for colour corrected images in comparison with the raw, uncorrected images. Also the average scores never drop fully towards zero for the colour corrected image series, compared to the uncorrected cases. Furthermore, the colour corrected images in over-exposed cases for a^*b^*chromaticity seem to behave in a sightly more robust way towards higher scores than for rg-chromaticity. Considering the fact of relatively large spreading bands, and that we encountered higher tolerances for the training of a^*b^*chromaticity based data leaves this last observation statistically unsupported.

8.5 Static Perspective Based, Continuous Correction

In the absence of *a priori* knowledge, one must resort to different means of detecting colour pairs to include into the determination of the affine transformation. We have undertaken capturing of long term camera measurement series with a static perspective. The goal on these recordings is to stabilise the colour appearance in an environment with changing light conditions.

For this purpose, we have recorded an image series over a day (approximately 9 hours), capturing one frame every minute. We have configured the camera settings in a way, that all settings except for the shutter speed have been fixed manually. This way, we were able

Figure 8.16: Outdoor scene (left) and indoor scene (right), as used for the long term static perspective observation with colour correction.

to prevent any white balancing, but still regain the safety that the images were not severely under or over-exposed, and the colour range stayed better within the camera's dynamic range.

The computation of the affine adaptation transformation was performed on a regular subdivision of the frame into square regions. The average $L^*a^*b^*$ colour was computed for each of these regions. The first frame was designated as a reference, and subsequent frames were mapped towards this one using a best fit transformation from the current to the reference set of colours. Many changes occurred in the scenes over time, for example by people moving into and out of the field of view. Regions subject to such changes exposed a high degree of error in the regression computation. Such regions were dismissed from the computation of the final affine transformation after an initial regression computation. The distribution shape of the errors for the individual regions showed very high probabilities for low ΔE^*_{ab} values, falling very sharply afterwards and only leaving very low probabilities for higher errors. This was to be expected, as the scene changes were mostly very local only. Therefore, we could assume that almost all non-changed section of the scene could be found in the interval up to slightly more than the standard deviation. We chose a cut-off point of $1.25\,\sigma$ for the limit of regions to include into the regression computation.

The scenes under scrutiny were on the one hand an outdoor scene (Fig. 8.16 left), with mostly shaded objects in the foreground, and a background showing vehicle traffic as well as strong light fluctuations due to alternating sunny and cloudy weather conditions. It was one day captured with a good quality camera, that was quite capable of adapting to changing light intensities ("Logitech QuickCam Pro 9000", see Sect. 8.1), and a poorer camera that had only very limited capability to change the exposure settings automatically ("Creative Labs Webcam 5" CMOS sensor, model "PD-1000"). The second scene was an office scene with people traffic (Fig. 8.16 right), a mixture of artificial light and natural day light, as well as light being switched on and off during the time of measurement. This series was also performed using the better of the two cameras.

Each image series was analysed in its original, uncorrected form as well as with correction

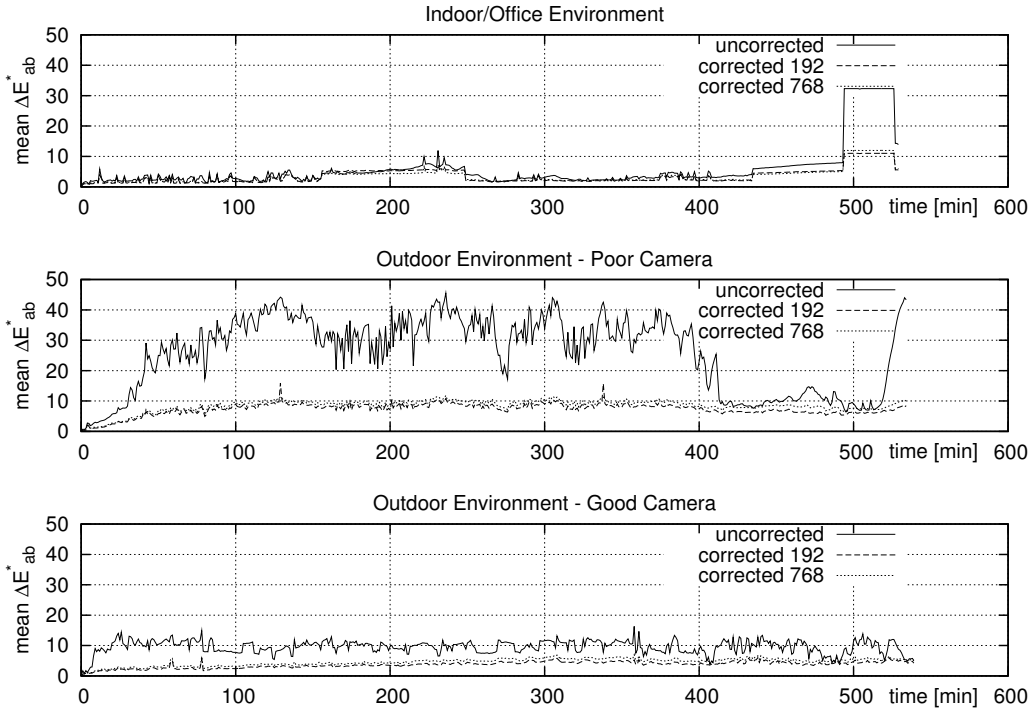

Figure 8.17: Results of the colour accuracy estimation of the long term observation for the different scenarios. The solid line shows the colour deviation ΔE^*_{ab} for uncorrected images, the broken lines indicate those for corrected colours. The corrections were conducted using 192 and 768 square regions respectively.

using subdivisions of $16 \times 12 = 192$ and $32 \times 24 = 768$ square regions for the colour averaging. The regression problem therefore was highly over-defined to give enough room to allow for the deletion of changed regions as well as for the measurement error of this type of colour correction.

8.5.1 Continuous Long Term Correction Results

Fig. 8.17 illustrates an estimation of the average colour differences ΔE^*_{ab} between the (initial) reference frame and the current frame. The solid line indicates the uncorrected case, whereas the two broken lines indicate the colour corrected cases. These observations and deductions can be derived from the graphs:

- The granularity of subdivisions in the scene for the purpose of colour correction do not seem to have a significant influence.
- Room for colour correction in the very stable indoor sample was very limited to few occurrences, in which significant changes in the scene or light were encountered (e. g.

minutes 160–240 as well as minutes 410 and later). This indicates, that we do not seem to be introducing additional colour space distortions into the scenario.

- In both outdoor scenes, the reference image was captured in the morning's dawn light. The day's light intensity change creates potential for improvement through colour correction.
- The night's darkness (in one of the outdoor scenes past minute 520) introduces a significant change towards under-exposure, that can be handled equally well as the over-exposure during intense daylight.
- The bigger the introduced change of light in the camera's perception, the better the relative effect of correction is.
- But, the better the used camera's initial adaptation for exposure (as seen in the two outdoor scenes), the better the absolute result of colour changes is.

A visual inspection of the recorded and corrected footage revealed the following:

- The office scene was kept constant with respect to the background, as the moving objects/people were the source of disturbance.
- In the outdoor scenes the foreground was the common robust scene portion, with traffic and weather/daylight influencing mainly the background. Both cameras' automatic exposure adjustments very strongly reacted to changes in the background, whereas the colour correction stabilised the colour appearance of the foreground objects.
- The outdoor scene captured with the poorer camera does not handle the adaptation to strong light changes very well. The dawn and the dusk conditions show very little influence on the captured image. The correction matches very closely that of the original capture. Bright daylight, as well as the darkness of the late evening, introduce strong changes the camera does not cope with well. But even in these the foreground resembles quite closely that of the reference, with the background looking *very* different for both of these cases.

8.5.2 Summary of Continuous Correction Performance

This approach taken for a colour correction is inherently less accurate, as it tries to infer colour changes through observations of the whole scene. But as shown, it does provide a viable way for handling the colour correction task without precise input through *a priori* knowledge. This approach taken still bears a large margin of improvement. As mentioned in Sect. 8.3, better quality data can be obtained by an improved scene segmentation. Instead of subdividing it regularly into rectangular regions, the scene can be segmented by foreground/background to exclude regions of change. A segmentation by objects may result in higher saturation data points to better sample the size of the colour volume.

A certain advantage of the used segmentation approach is the higher number of data samples with a low saturation. This helps to stabilise the neutral axis, and prevents an overall colour cast most noticeable in the neutral colours. It provides an effective measure to counter this effect visible in Fig. 8.7 of Sect. 8.2.3.

8.6 Conclusions on Adaptive Colour Transformations

In this chapter colour changes induced due to changes of light conditions were analysed. Compensating for them is a matter of finding a suitable transformation, that ideally reveals the canonical representation of the colours encountered. Of course this is impossible, and we can only achieve a more or less well suited approximation of this ideal. Already an ideal white point adaptation on an sRGB encoded image introduces an estimated error of roughly $\Delta E^*_{ab} = 2$ [17]. We are dealing in this case with non-ideal colour shifts, which we are trying to compensate for. Lots of computations on colour spaces are involved, so our overall error can be expected to lie beyond this threshold, no matter how accurate the determined transformation is.

For many of the colours sampled, we have been able to achieve quite good results already, just by using an affine transformation. This affine transformation was determined by means of using a 3-dimensional vector field regression computation. This transformation requires a minimum of four source/destination pairs of colours to match the current to the reference colours. An over-determination results in a more accurate determination of the affine transformation, while in the presence of measurement noise or other inaccuracies. Higher order polynomial transformation may lead to further improvements of the transformation quality, but they also demand a much higher number of input parameters. This happens at the expense of a decreased over-determination of the problem to solve. For cases with many accurately determinable colour pairings, this approach however could lead to an improved quality.

We were able to improve the range of tolerance for different machine vision applications regarding colour detection robustness. The robustness is still dependent both on the degree of similarity of the illuminant type (a different spectral power distribution), as well as the magnitude of change in illumination intensity. Further limitations are given by the dynamic range of the capturing device (camera). The usable range could be extended in all cases over the non-corrected imaging cases. Even in cases beyond adequate correction for certain colours, some results could still be made usable.

This study showed, that results tended to be best when the affine transformation was determined and applied directly on an $L^*a^*b^*$ based colour space. A prior application of a white point adaptation (using a Bradford transformation) proved to be rather hindering than supporting. This was revealed by a visual evaluation, even though the quantitative analysis did not expose this effect.

Overall, an affine transformation on top of an ICC based colour compensation (as determined for the reference) proved to be a good compromise towards a practical adaptation of colour appearance. A high over-determination of the regression problem, as well as accurate data and not too extreme illumination shifts (of quality as well as intensity), can be handled quite well by the methods discussed.

In future enhancements, some improvements to the long term correction of static perspectives can be expected by using alternative scene segmentations. The current strict segmentation by a regular, orthogonal grid into rectangles often yields local colour averages close to the neutral axis. More distinct object segmentation approaches would provide more saturated data samples. This would promise to improve the accuracy of the affine transformation, as the samples used for the regression computation appear less clustered.

Chapter 9

ICC Profile Adaptation

In Chap. 7 we have discussed creating an ICC profile from characterisation data. In Chap. 8 we have discussed a way of correcting for colour shifts, that have occurred in the duration of a measuring sequence after the initial characterisation. In this chapter, the focus will be on a way to update an initially generated ICC profile, so that it incorporates the determined colour correction determined in Chap. 8. This way, the colour adaptation can be completely decoupled from a simple and fast colour correction, that only has to rely on an ICC profile. For further explanation see Fig. 6.1 in Chap. 6, outlining a workflow including a "worker process" dealing with the ICC colour correction, and a decoupled "adaptation process" that deals with the creation of adapted ICC profiles suitable for the current time. The goal is to fuse the whole applicable colour transformation chain into a single ICC profile, so the worker process only deals with a single, comprehensive ICC profile transformation.

In the following, Sect. 9.1 will first introduce what needs to be adapted within the structure of ICC profiles. Sect. 9.2 outlines the transformations to be undertaken in this process to update the information embedded in the profile. The implementation of these steps are described in Sect. 9.3. Finally, Sect. 9.4 sums up results gathered with adapted ICC profiles.

9.1 Correcting ICC Profiles

To modify an ICC profile, we will first have to go back to have a look into the anatomy of an ICC profile. For an input profile (correcting the colour input for example for a camera), the "active ingredient" of an ICC colour transformation is the "AToBx" tag. For a Colour Look-Up Table (CLUT) based profile – as we are dealing with here – the "AToBx" tags may only contain these three used transformation steps (see Fig. 7.1): A pre-linearisation using the "A" tables, a CLUT table, and a post-linearisation using the "B" tables. All non-participating transformation steps – either because they are not allowed, or because they are not used – are set to identity transformations and can be ignored (see Sect. 7.1.1).

We are assuming that all these three allowed transformations (the two sets of linearisation tables and the CLUT) are used. If not, their identity transformation will result in a neutral outcome and can therefore be ignored in the following discussion of this chapter, as it will not influence the outcome in the processing.

In order to fuse our specific colour adaptation into the input process chain, we need to manipulate the content of one or more of these three transformations in the "AToBx" tag for an input profile. To enable the full adaptation possible through an affine transformation derived in Chap. 8, the CLUT has to be adapted. The linearisation tables are sometimes not used, and will be left as they are, but they do need to be considered in the process of updating the "AToBx" tag's content.

In the case where the linearisation tables are used, theoretically "re-balancing" of the totality of the three transformations towards optimising the overall behaviour of the profile could be considered. However, an optimisation attempt could, if not done properly, introduce further rounding errors into the total transformation. A significant change in the quality of the updated ICC profile cannot be expected for the small changes in the quality and quantity of the illuminant. For larger changes, the error introduced through measurement and the adaptation process (from Chap. 8) is expected to be dominant, so that re-balancing the transformations is not an effective means of improving the quality.

9.2 Profile Adaptation Process

In the previous section, we have determined that the CLUT in the "AToBx" needs to be updated. The corrective colour transformation is applied on the current colour output in CIE LAB colour space. This is, in the presence of "B" curves (post-linearisation curves), available at the end of the colour correction chain of an ICC profile transformation. A CLUT value is transformed through the "B" curves to an $L^*a^*b^*$ colour value and corrected. This corrected colour value now needs to be transformed back to the CLUT colour space using inverse "B" curves, and then can finally replace the original CLUT entry.

The "B" curves are one-dimensional look-up tables per channel. Theoretically, these can assume any shape. Technically though, they only make sense in colour transformation applications if they are monotonously increasing, as they would otherwise lead to meaningless artifacts in the colour space mapping. Due to this fact, these three curves can easily be inverted for our purposes.

The correction of the CLUT entries in the "AToBx" is outlined in pseudo code in Fig. 9.1. In preparation, first the three "B" curves need to be inverted for the L^*, a^* and b^* channels (lines 12–14). After that, this five-step-process is applied for each node to update the whole CLUT:

1. Take every destination (colour) tuple from the CLUT (line 19)

```
1  # We have got these transformations available:
2  #     transform_B_L(value)
3  #     transform_B_a(value)
4  #     transform_B_b(value)
5  # each returning a transformed value for the given channel.

7  # Additionally of course the corrective colour transformation:
8  #     correction(colour_vector_Lab)
9  # returning a corrected Lab colour vector.

11 # Preparation: Invert "B" curve transformations.
12 tansform_B_L_inv = invert(transform_B_L)
13 tansform_B_a_inv = invert(transform_B_a)
14 tansform_B_b_inv = invert(transform_B_b)

16 # Iterate over all CLUT entries.
17 for node in CLUT_nodes:
18     # Step 1: Get original CLUT destination value.
19     original_value = get_CLUT_value(node)

21     # Step 2: Apply "B" curve transformation.
22     original_Lab.L = transform_B_L(original_value.L)
23     original_Lab.a = transform_B_a(original_value.a)
24     original_Lab.b = transform_B_b(original_value.b)

26     # Step 3: Apply corrective (affine) transformation.
27     corrected_Lab = correction(original_Lab)

29     # Step 4: Apply inverted "B" curve transformation.
30     corrected_value.L = tansform_B_L_inv(corrected_Lab.L)
31     corrected_value.a = tansform_B_a_inv(corrected_Lab.a)
32     corrected_value.b = tansform_B_b_inv(corrected_Lab.b)

34     # Step 5: Update CLUT for this node.
35     set_CLUT_value(node, corrected_value)
```

Figure 9.1: Pseudo code of the algorithm updating the CLUT values in an ICC profile.

2. Apply “B” curve transformation to that tuple (lines 22–24)

3. Apply corrective transformation to resulting colour tuple (line 27)

4. Apply inverse “B” curve transformation to adapted colour tuple (lines 30–32)

5. Replace the original (colour) tuple in the CLUT with the newly derived one (line 35)

The ICC profile with the corrected CLUT can now be written back to a file, or used directly if it is resident in memory.

9.3 Implementation

In order to gain access to the content of the information stored within profiles, LCMS [42] is used, as already in Chap. 7 to create and write profiles. Unfortunately, LCMS only provides limited access to profile data in the current line of 1.x versions. The author of LCMS is currently working on the beta of version 2.0, which is to include full (and more convenient) access to all enclosed profile data. However, the information needed to update the CLUT in a profile is already accessible through some tricks using the API of the current version 1.19.

For the purpose of this research, the Python bindings library PyLittleCMS [41,44] needed extending to make use of these features, to conveniently access the profile's CLUT(s). Very conveniently, LCMS also provides a fast function for linearisation curve inversion, we are using for the inversion of the "B" post-linearisation curve.

9.4 Results

All colour transformations that have been determined in Sect. 8.2 were used for a comparison. In this comparison the "base profile" was determined for the reference case "`65270000`", and applied to every image. A corrective affine transformation was then determined to correct the illumination cases against this reference. From here, we have once corrected the colour images using the affine transformation directly, as well as applied the same correction to the "base profile" (as described above in Sect. 9.2), which then was used to correct the colour appearance of the original image in one step.

In almost all cases, the appearance of the corrected images was visually indistinguishable between the pairs of colour correction processes. The cases where correction artifacts were clearly visible, was in areas of extreme over- or under-exposure. These areas could be seen specifically for some illumination cases. The CMM (Little CMS) performing the corrective transformation used internal "sanity checks," to prevent numeric roll-overs beyond the boundaries of the number space (16-bit unsigned integers used internally, within the profile's encoding as well as the CMM). Applying the affine transformation directly to the 16-bit $L^*a^*b^*$ encoded colours led to these roll-overs, which were not treated explicitly to yield a more realistic output.

Beyond the roll-over artifacts, pixel value differences were not visibly detectable. They mainly resulted from rounding differences, due to the different process chains when applying a theoretically quantitative equivalent correction.

The base profile used was created with Argyll CMS [50], which has been determined as a good reference in past chapters already. These profiles are well optimised and make use of "A" and "B" tables by default. Therefore, the full profile adaptation chain had to be put to use as described above, including the post-linearisations ("B" tables) together with their inversion.

9.5 Conclusions and Future Work for Profile Adaptation

ICC profiles describe corrective colour transformations, which are applicable using a colour matching module (CMM). These profiles are only valid for very specific conditions (e. g. device, illumination, media type). In Chap. 8, we have made the attempt to determine corrective colour transformations to compensate for changes encountered since the device characterisation (determination of the ICC profile), to get a more accurate colour perception of input devices.

In this chapter, we have investigated and implemented a mechanism to modify an existing ICC profile, by incorporating the change of an additional corrective transformation into it. Using such a modified – or updated – profile eliminates the need of chaining corrective colour transformations, and enables one to use a *single* standard compliant ICC colour transformation for the whole process. This has various advantages: Less computation overhead (faster); usability of virtually any (commercially or freely) available CMM; the transformation can take advantage of additional sanity checks in the CMM; and no dependence on further needed colour correction code.

The results obtained through this ICC compatible single transformation using a modified input profile were in comparison of equal quality with the previously used multiple individual steps. Slight differences between the results were caused by rounding differences in the different processing chains and were not visibly detectable. Some larger differences were encountered due to the fact that the application of the corrective transformation in the multiple step process could exceed the valid numerical ranges of values (0–65535 for unsigned 16-bit values) for areas of extreme illumination in the images. These cases were handled much more "gracefully" by the CMM together with the ICC profile to avoid disruptive artifacts. This treatment of extreme cases however does not make the colour correction more accurate for those pixels, it just makes the visual appearance less disruptive to a viewer's eye.

As outlined in Chap. 7, pre- and post-linearisation has not been implemented in our own profiling code. Once that has been done, experiment with potential improvements by re-balancing the linearisation tables in conjunction with the CLUT could be undertaken to derive optimised adaptation results.

Chapter 10

Implementation

The implementation for the code used in this research is based on a variety of different libraries and tools. An evaluation of colour management systems (CMS) is given in Sect. 10.1. In the core of this research, codes for colour adaptation processes were developed. The approaches for these different implementations are outlined in Sect. 10.2. Sect. 10.3 lists the exact versions used for the different libraries and tools. Finally, in Sect. 10.4, we are presenting the outcome towards the chosen path way of the implementation towards suitability.

10.1 CMS Tool Evaluation

Particularly for the purpose of colour management systems (CMS), only a small number of implementations in the open source are available. The most common three ones are evaluated for this research in this section. These are also virtually the only choices. Namely, we have evaluated Argyll CMS [50], SampleICC [51] and Little CMS (LCMS) [42, 43].

Since version 3.5 of the Mozilla Firefox web browser, another implementation is available in the open source. It is tightly integrated into the Mozilla code base, aiming at a very fast and lean implementation. It has come into existence past the decision of an adoption of a CMS tool, and was therefore not fully evaluated. Due to its tight integration, significant work would have to be undertaken to extract it as a library for general use, and no guarantees are given towards the stability of the API. As far as it is known, the implementation can apply ICC profiles for colour correction only, and it currently provides limited functionality for certain common profile classes only.

10.1.1 Argyll CMS

Argyll CMS [50] is (among other virtues) known in the community for the high quality of its profiler and a complete set of command line tools. All these tools available are

implemented platform independently to work on Linux/UNIX operating systems, Mac OS as well as a wide variety of Windows versions. Due to the quality and platform independence Argyll CMS enjoys a wide user base. The tools are available for the command line only, with no GUI tool support or externally usable libraries provided. However, many people have implemented applications with graphical user interfaces to ease the use of many of the tools, particularly for the profiler. Further benefits for users are the exhaustive and up-to-date documentation and the active community with a very helpful mailing list. Graeme Gill, the author of Argyll, is also *very* active on this mailing list, and always eager to resolve problems, documentation ambiguities, potential bugs, general problems of understanding, take up pointers for improvements, etc.

As mentioned above, the quality of the profiler is one of the highlights of Argyll. The profiling code is highly sophisticated, and very complex, as it has been repeatedly improved for more than a decade, now. An attempt to decipher the used profiling algorithm has been made. The C code is extremely lengthy and, due to the evolutionary nature, hard to scrutinise. For the purposes of an "own" profiler (see Sect. 10.2) this turned out to be too complex.

The Argyll tools have been integrated into various GUI tools, mainly for profiling. A slightly modified version of the profiler has been extracted to be used in LPROF, the profiling spin-off tool originally developed for Little CMS (see Sect. 10.1.3). The intent of this however is not as much a general purpose profiling library, but a tool that is intended for certain simple profiling purposes, and not much for general use.

Argyll CMS is licensed under the GNU General Public License (GPL) in version 3. This restrictive open source license intends to ensure future freedom of the code, but prohibits any use of it within applications or libraries under an incompatible license (e. g. for commercial, closed source applications). For this research, this license does not mean any restrictions, and due to the very good and easy communication with the author, and the active community, Argyll CMS is a good contender for being useful for the intended purposes of this research. The biggest general restriction is, that no libraries with a usable API are available to make use of the only internally available features that we require (e. g. accessing the content of profiles, altering and saving them again).

10.1.2 SampleICC

SampleICC [51] is an open source, cross platform C++ library for reading, writing, manipulating and applying ICC profiles. It is considered to be the reference implementation for colour management by the ICC, with the library closely following the structure of the specification. It is licensed under the "ICC Software License, version 0.2," which is a minor variation of the BSD license (a very liberal open source license).

For these given reasons, as well as the fact that it also includes an implementation of a profiling algorithm, it seems like it could have been the ideal candidate for use in this

project, and as a base for further research. The code base is implemented in C++ in an object oriented way. Specific build and IDE configurations for MacOS (Xcode) and Windows (Visual C++) are included, and a combination of `autoconfig`/`automake` that should work for Linux/UNIX are available. Unfortunately, the code base is not suitable to be built robustly in a cross-platform way on some sampled Linux systems using currently common build tool chains. The author of SampleICC was also not very responsive towards questions, and a community around the library does not really exist. After many failed attempts of fixes and patches we had to give up trying to build SampleICC for our working platform (Linux).

We also found in the case of SampleICC, that the profiling algorithm is hard to scrutinise. Additionally, the quality of it was not verifiable as no community feedback or own experiments could be conducted using it. SampleICC failed to be a successful alternative for this research, due to the reasons of lacking verification and the problems to build the libraries successfully.

10.1.3 Little CMS

The last contestant in the evaluation of a suitable CMS was Little CMS (or LCMS) [42, 43]. It is available primarily as a cross platform C library for reading and applying ICC profiles, and it is available in the open source under the MIT Public License (a very liberal open source license). An extremely positive aspect about it is, that it is very widely used, and can be considered to be the *de facto* standard for a CMS library used in nearly all open source projects needing an implementation of a CMM. Therefore, it features a very wide, active and helpful community. All major problems of embedding it into existing applications, building on different platforms for different purposes, etc. are likely to have been addressed, discussed and resolved. Furthermore, the author provides an extensive tutorial and library API documentation, along with an extensive set of example code in C and Python. The documentation does feature a few minor inaccuracies in less often used functions. These, however, were mostly easily resolved through the examples or communication with the author.

As our research code was intended to be implemented in Python, it was positive to see that it also includes a variety of language bindings, including one for Python. Unfortunately, these bindings are incomplete, not very easy to use and lacking an idiomatic Python API. The API basically mirrors the C API in another language. To be fully usable, new Python bindings for this library had to be created (see description in Sect. 10.2).

These new bindings were much more complete, and were also able to handle colour transformations on whole arrays – rather than for single tuples at a time – to significantly improve the performance of profile applications. Furthermore, now the Python API offers features for accessing profile contents (for reading and modifying) on a low level, and writing profiles. This opened the door to create and modify profiles in a convenient way.

Little CMS does not offer any support for profiling. LPROF used to be an implementation of a simple profiler based on it, but it was discontinued by the author. The community

has picked up the code and revived it. Nowadays, LPROF is a simple profiler GUI front end which interfaces Argyll CMS (see Sect. 10.1.1) for profile creation. It can also be used to manage certain aspects of profile handling for the operating system (e. g. monitor profiles).

10.2 Approach

It was clear up front that any development would be driven to make use of hybrid programming language environments. That means, that not any one language would be dictated to be the sole one used for the implementation, but different ones would be chosen depending on their strength for a certain task.

For reasons of speed of an implementation and clarity, Python was chosen as a base language. It shines through a very large multitude of available modules in the core library, and it has attracted numerous developments contributing further functionality in libraries, extensions and frameworks. Python is designed in its base to be very useful as a "glue language," meaning that it is very easy to integrate different languages or libraries with it. Much of the core functionality is provided through long standing and well known native libraries, which are interfaced with an easy to use Python API. Many other projects have picked up that approach to provide extra functionality in this way, that translates to an ease of programming where possible, but still maintaining a speed of execution where necessary. An example where this has been exercised successfully are the mathematical tool boxes NumPy [66,67] (Numerical Python) and SciPy [78] (Scientific Python). Both of these for example are making heavy use of well known libraries as BLAS/LAPACK or ATLAS (Automatically Tuned Linear Algebra Software), UMFPACK, ARPACK, as well as other optimised libraries for fast Fourier transformations, optimisation, interpolation, etc.

On a similar approach, any operation in pure Python that shows a performance bottleneck can easily be analysed, and in case of a benefit, ported to native code that can be called through a function. Many helper libraries and tools are available to make use of such native codes as easy as possible. Even cross-paradigm code mingling within a single source file is possible (e. g. through Cython [79] and Pyrex [80]), with the run time delegating on the fly compilation, execution and parameter type conversion automatically.

All of the evaluated CMS tools have their individual merits and draw backs. Under the bottom line Little CMS "won" the evaluation, as it is provided as a library under a very non-restrictive open source license, it is mature and it features a good community with support. Below, we are going to describe the motivations and result for the creation of new Python bindings to facilitate colour management from Python.

We have implemented the mathematical challenging aspects of the ICC profile generation process. These form the input, used to write the new profiles to the file system with the help of the bindings to LCMS.

Python Bindings to Little CMS: Due to the fact that LCMS is implemented in pure C, with a well outlined and simple API, it was comparably simple to interface to from Python. A still rather crude Python API was created using code generation with the help of *ctypeslib*, a collection of supporting tools for Python's native library integration tool Ctypes [81]. This generated API now was nearly a one-to-one mapping of the complete C API. A much more "Pythonic" object oriented API then was "plugged on top" of this one for much more intuitive usage [44, 82]. A detailed description on the process of the implementation is discussed in Appendix A.

These new bindings now were complete, *much* easier to use and *much* more intuitive for the designated language Python. Additionally, the new bindings – called PyLittleCMS [41] – now provide automatic management of memory segments that were allocated, so that objects were freed upon disuse (with garbage collection). Several core users of LCMS from Python have by now also switched to these bindings in favour of the default provided Python bindings in Little CMS. They are already used commercially by an Australian company, and interest for including them officially into Debian and Ubuntu Linux has been expressed.

PyLittleCMS has been developed fully through test driven development. It is almost entirely supported by test coverage with unit tests using Python's *nose* testing framework.

Profiler Code "Mārama": For certain tasks, the Argyll CMS profiler was used, as it also serves as a high quality reference. But for aspects of this research a deeper insight and access to profiling data was necessary. The profiling algorithms of Argyll and SampleICC were not easily decipherable, so a "fresh" new self made implementation had to be targeted, the profiler "Mārama."

Through communication and feedback with the author of Argyll [73] (Graeme Gill), good references for starting points of an implementation have been received. The core of the profiling algorithm – the population of the multi-dimensional CLUT – has been implemented accordingly through a multi-dimensional spline fitting modelled after Bone (see Sect. 7.2). The spline fitting and interpolation algorithms have been implemented in pure Python with extensive use of the NumPy and SciPy libraries. The computation time for a fully fitted CLUT is several minutes. In comparison, the profiler in Argyll CMS uses between 5 and 20 seconds for this. Our code is not very fast, but acceptable for a prototypical implementation aimed at offering adaptability of the code with maximum ease.

The fitted CLUT data is then used for writing an ICC profile using Little CMS. For this purpose, further enhancements to the PyLittleCMS bindings (see above) were necessary to support the less commonly used features of LCMS for creating profile tags and writing profiles. This was a bit of a challenge, as these features are much less well documented and explored through examples, as most users are just *applying* ICC profile colour corrections, rather than creating profiles.

The computational performance of the spline fitting and profile generation was not under scrutiny in this research. Although it is always insightful to be able to gain an understanding

of the problem's complexity. Some information on the implementation and performance: The whole system has been implemented in Python (version 2.5 and/or 2.6), running in a single thread with the Python specialising compiler "Psyco" enabled. The system still provides plenty of optimisation potential. Computation times on an Intel Pentium D CPU (two cores) with 3.20 GHz and 2 GB of RAM ranged between 2 and 20 minutes depending on the problem, the parametrisation and applied simplifications (see Chap. 7). For comparison, the command line tool `colprof` from the Argyll colour management tool suite [50] implemented in pure C runs for roughly ten seconds.

The profiler's code base is also developed through test driven development. It is also using nose, but the test coverage is – due to difficulties of testing numerically converging codes – not as high as the one of PyLittleCMS.

Colour Constancy: The colour constancy algorithms (see Chap. 4) also have been implemented in Python. Reading and writing of images has been facilitated through the Python Imaging Library (PIL) [71], and all subsequent matrix operations on the image data have been conducted using the well optimised NumPy library. Due to the very limited programming logic that was executed in pure Python, as much processing was executed internally within the native library, these implementations performed very well.

Colour Adaptation: All variations of the colour adaptation algorithms have in principle been implemented in a similar fashion to colour constancy from Sect. 10.2. Also, the extraction of trackable colour sample data was handled by regional "slicing" of image array data with the help of NumPy.

For the computation of linear regressions and optimisations, the available routines in SciPy were used. The algorithms' implementations were chosen from this library upon usability, and did not have to be reimplemented (see Chap. 8).

Profile Adaptation: Profile adaptation is based on the results of colour adaptation computations (see colour adaptation above). Therefore, the approaches used are extended for it. Furthermore, it ends in writing a new, updated profile to the file system, so portions of the profiling code (see "Mārama" above) are used as well. However, it requires access to current profile information.

Access to the profile information (CLUT data and per channel shaper curves) needs to be gained through some "back doors" using LCMS. Certain LCMS functions provide access to profile data structures. These structures are mainly considered for library internal use only, but with knowledge of the various fields, they can be used to access the profile data. PyLittleCMS needed further extensions to access this information, and to represent the data from this structure in easily accessible ways for the Python code by mapping it into a NumPy array object. Also, the per channel shaper curves needed facilities to invert them.

Table 10.1: Exact versions of codes, tools and libraries used or discussed for this research.

Name	Version
Little CMS	1.18
Argyll CMS	1.0.4
SampleICC	2008-11-24 (1.3.8)
PyLittleCMS	0.7 (at time of writing)
Python	2.6.2
Python nose	0.10.4
GCC	4.3.3
glibc	2.9
GCCXML	0.9.0
PyGCCXML	1.0.0
ctypeslib	0.9
NumPy	1.2.1
SciPy	0.7.0
PIL	1.1.6
OpenCV	1.0.0
ctypes-opencv	0.7.3
matplotlib	0.98.5.2
GNUplot	4.2.4
Visual Python	5.11

These additions to PyLittleCMS opened up pathways to alter existing ICC profiles. The basis for these alterations was given through the colour adaptation results that were fused into the profiles (see Chap. 9).

10.3 Versions Used

The core of all development has been undertaken on Ubuntu Linux 9.04 (Jaunty Jackalope). Specific versions of the different tools, libraries, run times, etc. are listed in Table 10.1. PyLittleCMS [41] – as developed during the course of this thesis – has reached the version given in the table by the end of the thesis.

Several tools are mentioned, that have played minor parts in the development of the tools. These were mainly for accessing live imagery from digital cameras (OpenCV and ctypes-opencv) as well as for plotting and rendering of (live) data [83] (matplotlib, GNUplot, Visual Python).

10.4 Outcome

All attempted implementations worked. The development and evaluation went ahead with rapid succession of iteratively modified versions and varieties for testing alternative approaches. Even after months, the code base was still very readable and easily modifiable.

The implementation process was mostly supported by test driven development. This allowed for a rapid evolution of the code base involving heavy refactoring. Any errors due to oversights were quickly found and eliminated, and it was assured at all times that changes on the code base did not have any negative influences on the correctness of execution of other (dependent) portions. Regular regression testing has more than doubled the size of the code base (primary code + tests). Still, the immediate detection of problems and constant automated testing have improved the code quality (in software architecture) as well as the speed of the implementation.

Python is an interpreted language, which is in pure form (not involving any external, native libraries) in most cases slower than Java. However, it makes extensive use of native libraries within the language's core packages, as well as in many available extension packages. The execution speed within these is extremely high, especially as many of the mathematical/scientific extensions make heavy use of highly optimised implementations. Frequent use of these largely increases the overall execution speed, to often match that of pure C implementations. This is due to the fact that it is often impossible for a researcher to optimise all algorithms and loops as thoroughly as those that are used in the special scientific modules. As the overhead of using external natively compiled libraries is quite low, also the use of the well optimised and mature Little CMS C library has been quite straight forward and yields good and fast results. Only the profiling algorithm using a rather complex relaxation algorithm over many (nested) loops showed to be quite slow. The speed of execution along with the frequency of need for it however did not justify a largely optimised reimplementation for the purposes of this research.

10.5 Conclusions on Implementation

All research code has been implemented in Python, supported by highly optimised libraries implemented in natively compiled shared libraries. Unavailable tools for colour management purposes have been implemented by wrapping the Little CMS C library to a Python package with an object oriented API. This was also used as a base for building a profiler to write CLUT and per channel shaper curve data to ICC compliant profiles. The data for these structures was derived through an implementation of a multi-dimensional linear spline fitting algorithm.

The colour constancy and colour adaptation approaches were easily derived and implemented in Python using common imaging and numeric libraries. Finally, the processing of

large experimentation sets and studies could be easily accomplished by using a "glue script" to automate the processing of large series of computations over night or several days.

We are quite happy with the choice of using Python as a base for this work. The new Python bindings for LCMS – PyLittleCMS – enjoy a good stability and usability, which is reflected by lots of interest from users in the community for them.

Regarding the CMS used, it shows the importance of an evaluation of a variety of available systems for colour management tools/libraries to determine the right choice. The "obvious" initial instinct of using the open source reference implementation proved to be much more difficult and cumbersome to use. Also, the officially provided Python bindings for LCMS proved to be insufficient. But the effort to provide more suitable bindings, and to implement our own profiler with full and precise access to profiling information, showed to be a path worth taking.

Overall, the goal of the implementations was a proof of concepts, with the ability to test many competing codes and implementations against each other. This had to be done on a code base that was easy to maintain, so that one has a good assurance of the validity of the code without extensive manual testing even after minor changes. The aim of the research is the result, not the "performance crown." Certainly, the implementations can be optimised for speed if implemented in compiled, native code (e. g. in C/C++). Depending on the field of use this might be a necessity in future use cases anyway. But the more clean and much more brief Python implementations can be used as a proof of concept, and prototype in pseudo code, which can be much more easily translated into lower level languages with faster execution speed.

Chapter 11

Discussion

This chapter summarises the outcome of the research undertaken. First in Sect. 11.1, we stating the contributions achieved. In Sect. 11.2 a summary of findings is given. Sect. 11.3 discusses the outcome of the research with an outlook on applicability and future research potential. Finally, Sect. 11.4 gives an outlook of the future research potential that can be derived from this thesis.

11.1 Contributions

This research consists in its base of four different fields: The modification of colour constancy algorithms to be applied to the $L^*a^*b^*$ colour space rather than the device colour space; Then, the evaluation of a variety of colour management systems (CMS) to select a suitable one for our purposes, and the implementation of an ICC profiling system on top of it as the basis for the following research; On the basis of the profile connection space (PCS), as derived from colour transformations with the CMS, the development of concepts for adaptive transformations for colour corrections; Lastly, these corrective transformations were fused into an existing ICC profile to update it to a current illumination situation, using the information and knowledge gained from the profiler and using the CMS library. Additionally, some results from practical experience with the actual implementations undertaken in this research.

11.1.1 Colour Constancy with $L^*a^*b^*$

Reviewing the current state of colour research in the literature, it became clear that the most robust scientific base for colour processing is given on CIE colour spaces. A colour representation in these is usually derived through an ICC specified colour transformation, involving the application of device profiles. Therefore, these CIE colour spaces are not

individually device specific anymore, but normalised to a standardised observer. Specifically, the linearised $L^*a^*b^*$ CIE space seems to present desirable qualities for the application of quantitative colour operations. Another quality for certain types of reasoning in the $L^*a^*b^*$ space is that it is an opponent colour space. This means that the three components are describing linearly independent aspects that are complimentary to each other. The L^* channel contains the colour's lightness, whereas the a^* and b^* channels encode its colour information (chromaticity).

Many colour scientists are applying colour corrective operations straight to the device colour spaces. These are often the RGB spaces available from the device, as they can be passed straight to output devices (like computer displays). Some scientists are first applying a channel linearisation, based on an assumed gamma correction value (e. g. $\gamma \approx 2.2$ for sRGB), whereas others are not aware of the distortions of this power function, and are computing directly on the RGB tuples.

In our opinion, a lot of ambiguity and guesswork of colour processing could be eliminated if one would work only on device independent and linear colour representations as CIE LAB. To undertake corrective colour transformations, that are to be fused back into an ICC profile, it is convenient if corrective colour transformations are derived and applied to $L^*a^*b^*$ values in the PCS directly.

For this purpose, we have modified the two most well known colour constancy algorithms – White Patch Retinex and Grey World Assumption – to be applicable to the $L^*a^*b^*$ colour space. For these two, and many other colour constancy algorithms, the overall process is comprised of two distinct steps: In the first one, the illuminant's colour is estimated, and in the second, this information is used to transform the colour space. The latter step usually involves (linear) scaling of the space's colour channels.

If the modification of the algorithm is performed in an analogous fashion, then as the RGB channels are scaled for the device colour implementation, we must scale the $L^*a^*b^*$ channels for the CIE LAB implementation. However, we are dealing here with an adaptation from an arbitrary illuminant to a normalised illuminant. In colour science, this is handled by so called chromatic adaptation transformations (CAT). This process is usually handled by *von Kries* type transformations, which apply the adaptation in a "cone response domain" (named after the response of the light sensitive cone cells in the eye responsible for colour vision). Three different methods for the CAT were compared to the straight $L^*a^*b^*$ channel scaling: the "true" *von Kries* transformation applied to the normalised LMS cone responses, the "wrong" *von Kries* transformation using XYZ channel scaling, and the Bradford transformation, currently accepted to be the best compromise.

Most researchers are quantifying the quality of colour constancy algorithms by using the scores of colour indexing as a benchmark. In colour indexing, objects are identified by their distribution of colour occurrences in a specific colour space. As the colour appearance changes with different illuminants, also the identification probability decreases. Colour correction systems, such as colour constancy algorithms, are then used to correct the colour

representation before the identification, and the scores for different implementations are compared.

It showed, that in colour indexing for comparison of approaches the $L^*a^*b^*$ based implementations of the colour constancy algorithms performed comparatively well with respect to the "classic" RGB based approaches. And particularly the Bradford transformation was most suitable for an applied CAT.

Furthermore, we found that the colour indexing applied to the RGB space in comparison to the one on $L^*a^*b^*$ performed better. An analysis revealed, that both colour spaces were sampled with the same granularity (number of bins) in the histogramming based process. But the sRGB colour space is a smaller sub set of the full $L^*a^*b^*$ volume, therefore the histogramming in $L^*a^*b^*$ was comparably more coarse, and the matching quality was decreased. It is expected, that the $L^*a^*b^*$ based colour indexing would perform equally well or better, if the parameters for histogram boundaries and resolutions are adapted.

11.1.2 Colour Management and ICC Profiling

In an evaluation of openly available colour management systems (CMS), Little CMS (LCMS) was found to be the most suitable candidate for our research. Most of the implementation of the research was to be implemented in Python, but the Python bindings of LCMS, as shipped with the library, were insufficient. For this reason, we have implemented a new Python wrapper library for LCMS, PyLittleCMS.

The outset is to base colour corrections on an $L^*a^*b^*$ based PCS, derived through ICC profile transformations. Due to this, special value falls towards the understanding and inner processes in the determination of such ICC profiles. Specifically, full access to internal information, as the embedded colour look-up table (CLUT) and per channel shaper curves are essential. No openly available implementation could be found, that would reveal a good understanding of the profiling process, as well as grant access to the required data. These reasons motivated us to implement our own profiler. The implementation required algorithms for multi-dimensional spline fitting, interpolation and linear regression to derive CLUTs, as well as perform curve fitting in the profiling process.

The implementation of these numerical algorithms, together with the LCMS bindings were used to build the profiler. The implementation of a profiler itself is a non-trivial task, but not as much new and innovative research. However, as mentioned above, this was necessary to gain insight and access to the profile information required as ground work for this research.

11.1.3 Adaptive Colour Transformation

We have outlined a concept of applying corrective transformations based on device independent, linearised CIE colour spaces, as stated in Sect. 11.1.1. Just applying such corrections

on an alternative colour space however was not the goal. Our intention was to derive an alternative concept for determining a colour adaptation.

Colour constancy is based on the analysis of information from an individual frame. The approach of general ICC profile application is based on the determination of a single, initial, static characterisation of the capturing device. Our anticipated approach however, was to derive a system that attempts to observe or track objects or distinct colour appearances in a scene, and track the changes in the appearance of these samples over time. More specifically, we are tracking the changes in the *location* of these samples in the colour space. From this, we attempted to derive a corrective transformation to compensate for the encountered changes. To gain a baseline for the process, and to determine a suitable $L^*a^*b^*$ representation of colours, we performed an initial device characterisation under starting conditions, so we could apply this resulting ICC profile on the device image representation, to receive an $L^*a^*b^*$ image representation for colour correction.

To evaluate the viability of this concept, we have observed the shift of locations of a multitude of samples in colour space during illumination changes. What we needed to derive from this observation data was a model or characterisation of the colour space distortion. This model then could be parametrised, to define a reverse transformation for this distortion. We have chosen to start out with an engineering approach using a Taylor series, to theoretically model changes of arbitrary complexity. Tests revealed quickly, that a 1^{st} order Taylor approximation (up to the linear term) is sufficiently good for most of our purposes. A 1^{st} order Taylor approximation is (in this multi-dimensional case) an affine transformation.

The big advantage of an affine transformation is the small amount of input data needed to define it. Only a minimum of four data points are necessary to specify it. In a simple chromatic adaptation transformation (or some of the simple colour constancy algorithms), as described in this research, only one data point, the white point of the illuminant, is used. So, in comparison, an affine transformation already carries four times as much information, and is therefore much more capable in adapting the colour appearance from one illuminant to another.

Having said that, most of the tracked data points encounter a measurement or detection error, superimposed on the colour information. Therefore, it is better to acquire a higher number of data points, and determine the affine transformation parameters through a multi-dimensional regression computation, to compensate for these errors by averaging. Naturally, to compensate for non-linearities and systematic errors, it is desirable if the tracked data points are not co-located with close proximity, but well distributed over the available colour space.

Results from these colour compensations revealed a much improved colour perception, even for strongly different illuminants in their intensity or characteristic. The suitability of this approach was mostly limited by the limitations of the dynamic range of the camera. If channel values were limited by the maximum (usually channel value of 255) or minimum (usually channel value of 0), the colour representation suffered a "kink" in the continuously

smooth distortion. If tracked colours were suffering from this channel clipping, the derived affine transformation was not representative for the "true" affine component of the distortion.

11.1.4 ICC Profile Adaptation

An ICC profile characterises the relationship of device colour to colour represented in the PCS. As a PCS both CIE LAB and CIE XYZ are permissible, and together with $L^*C^*h^*_{ab}$ they are interchangeable representations of the same colour space. In our case, CIE LAB was used, and our corrective colour transformations are derived to operate directly on $L^*a^*b^*$ colour. It seemed obvious to us, to fuse the adaptive colour transformation into the existing ICC profiles, for the ease of a single step, standardised corrective application of these profiles.

These corrective transformations – in our case affine transformations – cannot be expressed through per-channel corrections. They need to be applied through corrections within the CLUT of a profile. In the normal process of applying an input profile, the device colour is first transformed through per-channel (e. g. RGB) pre-linearisation curves, these results are processed through of the CLUT data interpolation, and finally the results are converted through the per-channel post-linearisation curves yielding $L^*a^*b^*$ colours. To update the CLUT with our derived corrective transformation, each CLUT value had to be post-linearised (with the "B" curves), corrected (through our correction), and transformed back to the CLUT domain using an inverted per channel "B" curve transformation.

We have been able to use this process successfully. It requires the ability to read a profile, and access its internal data structures, correct them, and create again a standard compliant ICC profile. The updated profiles were fully usable by the tested CMS, and produced visually undistinguishable colour images in comparison to those produced in the multi-step process. Cases in the multi-step process, in which the colour encoding after a correction would "roll over" (to exceed the numeric range of the encoding), were not addressed. This effect created at times very unpleasant artifacts in areas of more extreme light colours. The colour management systems employed for the corrections using the updated colour profiles in contrast do know how to handle these cases "gracefully" to avoid deteriorating artifacts. This does not improve the correctness of the colour representation in these areas, but it prevents visually awkward artifacts.

The process of using updated profiles yields a strong simplification and speedup of the adaptation process. Assuming that updates to the ICC profile are only necessary at intervals that are very long compared to the frame rate of live image capturing, this can mean significant savings in processing time under live capturing conditions.

11.1.5 Implementation

As stated before, our implementation is largely based on the Python programming language, accessing a variety of natively compiled libraries in the background for processing speed.

One of these libraries is Little CMS or LCMS, which is very mature and extensively used in the community for many applications. Due to the shortcomings of the Python bindings provided with the library, we had to implement our own bindings to it (PyLittleCMS). These bindings are very easy to use through an object oriented API, which handles the allocation and deallocation of native data structures transparently, without the need of intervention by the user. Deallocation is handled automatically upon garbage collection through the Python run time. The bindings are much more complete, as they feature access to the *full* API of the C library through automatic code generation of the binding code. Almost all code provided through the object oriented API is covered by unit tests to ensure its quality. The new Python bindings already enjoy a good acceptance within the community as well as with Martí Maria, the developer and maintainer of Little CMS.

The profiler created in this research is also implemented in Python. It is slow compared to the ones found in other CMS', but the speed is sufficiently fast for these research purposes. In this research, is serves more as a "proof of concept" and learning exercise to gain a more substantial understanding of the inner details of the profiling process as well as the ICC profile data structures. The accessibility of information and data has enabled us to modify and update profiles as needed.

It was only natural, that all other implementations of colour constancy algorithms as well as the adaptive colour corrections, were also implemented in Python. This enabled a very fast prototyping and implementation of the code. Due to the heavy use of highly optimised native implementations, also the execution performance was quite good. It can be expected, that the optimised linear algebra implementations of these libraries yields a comparative performance to a native implementation in C, that does not resort back to a stack of optimised libraries.

It was very pleasant to develop the code base on the application side in Python, as we could easily automate and modify the existing implementation in place for testing and computation of parameter series. In testing for example, it was possible to employ Python "decorators" to wrap function or method calls without modification of the main code base. This way, it was for example very easy to implement graphical output and plotting to be performed on every update of internal data, to observe directly the dynamics of the spline fitting process. Also, extensive data series processing could be seamlessly scripted in the same language as the main implementation, with a minimum overhead to compute, evaluate, visualise and summarise the outcomes of long running extensive test series over night or several days on a weekend.

11.2 Summary of Findings

Overall, we can distinguish our findings into three categories: The work on colour spaces, implementing means to dynamically adapt a corrective transformation to a current illumi-

nation situation, and lastly the integration of such colour corrections into ICC based colour management work flows.

11.2.1 Colour Spaces

We found, that for many problems it is possible to change the working space from a non linear device space (often RGB) to a device independent linear colour space as CIE LAB. The algorithms then often do need some modification to support this. Due to this, also the implementation of algorithms can be simplified in many cases, as colour lightness (L^* channel) and chromaticity (plane from the a^*b^* components) are directly encoded.

Using $L^*a^*b^*$ as a working colour space works in many cases similarly good and can improve the outcome of the application of an algorithm. This is due to the linear nature (in contrast to most gamma corrected device colour spaces) mostly true for algorithms that are based on differential colour analysis. Further improvements result from higher uniformity in the linear visual characteristics of the colour space. Colour pairs, that show to have visually the same difference, exhibit varying numerical differences in different areas of many device colour spaces.

Lastly, due to the strong theoretical foundation and practical experience with the conversion to CIE LAB as well as its properties, we do not need to question the credibility of this colour space.

11.2.2 Dynamic Colour Adaptation

To correct for colour changes during an observation, one needs to gather information that can be used to build a compensation strategy. For our purpose, the idea of tracking changes in the colour appearance of objects worked quite well. The ability to use this information for the correction largely depends on the ability and accuracy of tracking such objects, and the ability to determine their colour accurately.

An approach to compensate for theoretically arbitrary colour space distortions is to use a Taylor series. In our analyses we found that a 1^{st} order Taylor approximation (affine transformation) usually compensates for the largest fraction of the distortion. Due to the need of only having to track a minimum of four data samples, an affine transformation also proves to be a goal that can be achieved in many environments. With more than four samples, the derived transformation becomes more robust, as it can be computed through a multi-dimensional linear regression.

We found cases that do not feature distinct trackable objects much more difficult to deal with. We have been able to implement a system that tracks a multitude of regional average colours, and uses those to compute an affine transformation. This worked, as these small regions were often dominated by individual colours, producing data points usable in a linear regression computation. Heuristics were used to decide which samples to use, and which

to discard for the regression computation. Due to inaccuracies in this heuristic, together with a higher inaccuracy in the determined averages, colour corrections were not nearly as good as those using distinctly trackable objects. This field of research definitely contains much potential for improvement. An alternative scene segmentation (currently regular and rectangular) could improve these results significantly.

11.2.3 Integration with Colour Management

Integrating the colour correction transformation into the regular colour management process bears a series of advantages. First of all, it can be used as a drop in solution for existing ICC profile based correction, we just need to provide adequate profiles for the current point of time. The processing for colour correction can be fused with ICC compliant colour management. Due to the decoupling of the adaptation's implementation, video processing code development can be based on any suitable code base that integrates any ICC compliant CMM.

Besides the aforementioned, we can see a variety of further advantages. First of all, the speed of execution is one of them. In case ICC colour transformations are applied anyway, we are moving from a multi-step to a one-step transformation process. This one process is applied within the CMM of the CMS, which is usually implemented in a way to yield a very good performance, both in terms of speed of the actual application of profiles as well as in the quality of the results. The speed argument is especially true for full source-destination transforms, as CMMs often smelter the transformation through input and output profile into a single transformation step. Another benefit is that a one-step process is less likely to accumulate rounding errors.

Lastly, this integration approach eases the possibility to decouple the adaptation and the correction processes (see Fig. 6.2 in Sect. 6.3). Loosely coupled software systems tend to be much easier to maintain over tightly coupled systems.

11.3 Context

> *"Life is pretty simple: You do some stuff. Most fails. Some works. You do more of what works. If it works big, others quickly copy it. Then you do something else. The trick is the doing something else."*
> – Thomas J. "Tom" Peters (often misattributed to Leonardo da Vinci)

11.3.1 Limitations (and Potential Solutions)

For the creation of a corrective transformation, we depend on the tracking of colour samples within the observed scene over time. In order to make the approaches of this research

work, we need to find something "trackable" within them, like reoccurring objects or stable areas (in the background) of the scene. In the absence of such features, the approaches described cannot be applied. Cases like that could be found in a camera that is moving, with changing scene content that cannot be expected to reoccur. Although, in many cases certain (occasionally) reoccurring objects can be found (e. g. road signs with distinct colours of sufficient stability in appearance in a vehicle mounted camera setup). An approach like this has not been implemented, yet.

Also, by breaking certain assumptions, the correction process can be "fooled," just like the human eye can be fooled for example in optical illusions. Some of these disturbing effects are metamerism or very strong illumination inhomogeneities. Also extreme (fast) changes can deter a proper detection. Currently no "bail out" solutions for this have been implemented, yet. These could be achieved by resorting for example to reuse a background/foreground segmentation of a previous frame, while assuming that the scene's geometric composition has changed only by an incrementally small amount. Another fall back solution could be to resort to using a colour constancy algorithm

As stated, in the approaches taken for this research, it must be possible to compensate the major contribution of the distortion of colour space by an affine transformation. If higher order terms outweigh the linear contribution, the given mechanism for a correction will fail. An alternative distortion model has to be found, which will either require a larger number of data points to be tracked, or it is likely restricted to very specific cases only.

11.3.2 Implications

Basing quantitative, and particularly differential, colour science on linearised colour spaces can resolve many issues in colour science. For example, operating on gamma corrected RGB colours yields too small differences in dark colours, and too high differences in bright colours. Additionally, the move of colour science away from "tinkering" with colour values, towards a more scientific/knowledge based handling of colour tuples for colour correction and analysis, leads to a better understanding of the underlying processes, and should lead to better and well founded results. Unfortunately, many people who want to make use of colour information from image capturing are lacking a general awareness of colour fidelity. If reasoning is applied to unsuitably non-linear colour spaces, unexpected and potentially faulty results may be derived.

11.3.3 Applications

Many of the principles and implementations of this research can be used as drop-in corrections for other processes. A sample for this can be seen from the results in Sect. 8.4. There, corrective colour transformations have been conducted in cooperative research with others to enhance the robustness of the identification of colour coded robots. The results of this

research can be very useful in many disciplines. Some of these may be in the domains of colour based object identification, quality control, colour based diagnostics, etc.

To ease the applicability of this research, with more work it could be "productised" into an appliance like library, that could be used in a "plug'n'play" like fashion. The user could configure the application to feed the image stream from a camera "through" it, and configure the way this colour correcting filter should work. For certain other cases, one could think up a stand alone appliance, that could be physically wired in between a camera and further recording/processing devices. This could be used flexibly for many types of colour correction applications, and the user does not have to worry about an implementation, but he could focus just on configuring and using it. However, this type of speculative usage scenario does require solutions to lots of corner cases, and provide robust fall-back solutions for the times when assumptions cannot be met.

11.4 Future Research Potential

The state of implementations is currently in a typical research state, far from usability in a real product. But it offers a good perspective on future extensions and research. Many improvements are already highlighted in the chapters of this thesis. In the following, we are summarising some of the key areas with an outlook of potentially more high level improvements.

An obvious one is, to make the implementation robust to changes beyond the current limitations, mainly for example in cases of to strong, sudden light changes or changes in the camera's perspective. For scenarios, in which we cannot build on *a priori* knowledge or distinct reoccurring objects that can be tracked, the method for the scene segmentation becomes very important. We can expect strongly improved accuracy of tracked colour data, if the scene is segmented with sufficient precision. The segmentation should result in a separation of distinct objects or contiguous surfaces of a mostly homogeneous colour appearance.

The implemented profiler in this work still offers vast potential for improvement. The points for this are distinctly in the quality of the spline and curve fitting, as well as in a generation of pre- and post-linearisation curves. Also an optimiser can be useful, that balances the fitting of the per-channel curves with the spline fitting of the CLUT.

We are currently only using an affine transformation as a corrective colour transformation. Other models can potentially be more suitable in general, or for particular cases. Different models and means of parameterising them can be derived and evaluated. Ideally, such a model could evolve, rather than being rigidly specified. For example neural nets or genetic algorithms could be used to determine a distortion model or adaptation parameters for a transformation.

A hybridisation of for example the White Patch Retinex algorithm with time based colour sample tracking seems like a promising modification. Bright specularities and black

points can often be found easily within scenes. In the presence of these, they can be used in two ways. On the one hand, to provide further data points. But on the other hand, these distinguished pieces carry more information as we know that they define the neutral axis, and as such, they are very useful to derive better colour distortion model parameters.

Finally, as mentioned in this thesis, the processes of deriving a colour correction transformation and of colour correcting are only very loosely coupled. Therefore, the implementation can easily be decoupled and potentially distributed, for example by using a service oriented architecture (SOA).

Chapter 12

Conclusions

We have built a system that perceives colour – and does that as precisely as possible – while adapting to light changes, when encountering common natural and artificial light sources. It obtains device independent and visually linear colour descriptors (CIE LAB colour space). The motivation for using this colour space is driven by its beneficial properties for analysing and comparing colours. Therefore, we have also based the time adaptive colour correction process to operate on this colour space.

For these reasons, the first step to this work was a thorough introduction to the process of colour capturing and perception, along with the encoding of different colour spaces. The different involved colour spaces were described in their origin, and compared by their individual properties, to foot the undertaken research on a solid base in colour science.

As a verification that colour corrective operations are sensible on CIE LAB colours, we have modified the well known White Patch Retinex and the Grey World Assumption colour constancy algorithms. Results obtained from these alterations were successfully validated using Colour Indexing. In this process, the effectiveness of the colour constancy algorithms operating on device RGB and CIE LAB colour spaces were subject to a comparative analysis using identification rates for various objects.

The approach for adaptive colour correction taken considers the additional factor "time," which could be used to extract further input data. This is in opposition to colour constancy, which only bases corrections on one image at a time. Information gathered by including time as a dimension was two fold: Primarily, a sequence of images in time enables us to study the shift of coloured elements in the scene; But also the fact that certain scene elements only change slowly can be taken into consideration, to ease the burden of having to adjust a colour correction for every frame individually. We can sum up, that the first enabled us to adapt colour corrections for the given moment; and the second made it possible to only update the correction in intervals. Thus, real time processing of corrected video feeds becomes possible.

The corrective process to generate an updated ICC profile is outlined. Due to the above mentioned common slowness of colour changes due to illumination, we can run this profile adaptation decoupled from a capturing process. The capturing process itself can operate mostly independently, and just receive occasional updates for the current ICC profile, to apply it to the incoming image stream from the camera (using a colour management system), enabling it to operate in (near) real time conditions.

12.1 Contributions

We have been able to show colour constancy can – with minor modifications – also be based on derived opponent colour spaces, as opposed to raw device colour spaces. In our research, we have employed processing on the device independent, linear CIE LAB colour space for this. This choice was fueled by the attempt to fuse practices of credible colour processing with various common approaches of colour correction. Besides the already mentioned colour constancy, these were inspired by different basic directions of correction attempts.

Firstly, the application of ICC profiles through a CMS can yield a good quality colour correction for fixed conditions, if the profile corresponds to the given conditions. The application of ICC profiles forms a proven, standardised, much applied base for additional corrections.

Further ideas were taken from biological and probabilistic approaches. Objects of interest are viewed in the context of a scene, which can be used to derive additional information for a correction. This information can be correlated with corresponding information acquired from previous or reference frames, to gain an understanding of the encountered shift in colour appearance through a temporal element beyond a single frame. A best fit matching of parameters for a colour space distortion model (through an approximation by an affine transformation) is performed, to derive an approximate correction for increased colourimetric stability.

Finally, known references within the scene's objects or context were harvested for the purpose of gathering colourimetric information, that can be used to derive the correction. This was achieved by tracking (reoccurring) objects or other scene elements. Their colouring then could be matched to the appearance from previous or reference frames. *A priori* knowledge can play a major role in this, where an application can build on distinct knowledge of appearance or composition of objects and scenes.

A key step in achieving this goal, was to move from single frame corrections (as used in colour constancy) to a time enhanced colour correction method (deriving information from a consecutive series of frames). These multiple corrective contributions were then fused into one standardised ICC profile, for a simple corrective ICC transformation, which is valid for the current illumination case only. This final, comprehensive input device profile closes the loop towards easy and well defined corrective colour acquisition with capturing devices.

12.2 Future Work

In the initial assumptions, we have envisioned to use a background/foreground segmentation to identify segments that are static in the scene composition. For the domain orf robot soccer, we have experienced that it is more suitable to track foreground objects, as these can be ideally identified and mapped to known colours of the initial conditions. The long term observations first used an implementation of a foreground/background segmentation. The implementation adapted on time scales that proved to be inappropriate for observations over hours. Instead, a probabilistic approach was taken, discarding outlier measurements from rectangular scene sections that hinder a good regression fitting. Therefore, the concept of a scene segmentation has been preserved. However, in some scenarios the approach of a background/foreground scene segmentation can still be very suitable. Future work should include this type of scene segmentation.

Another very promising concept to evaluate is a segmentation of the scene based on distinct object surfaces (of the background). This would reveal trackable colour samples with much higher chroma (saturation), and therefore lead to less clustered samples. That should strongly improve the quality of corrective transformations.

Also, fall back strategies in the case of the failure of assumptions have been mentioned. An implementation of heuristics to detect the failure of these assumptions, along with the fallback strategies, would likely improve the robustness of the overall colour correction system.

Many colour correction systems utilise Artificial Intelligence (AI). Some of these techniques are briefly mentioned in Chap. 3. AI as a means for deriving models of colour space distortion, distortion parameters, etc. could potentially open further possibilities. Particularly the "learning" of responses for quicker adaptation could be helpful, if certain illumination changes are predominant and reoccurring. The system could react to such changes much more quickly by reusing "pre-tinned" responses.

Lastly, the adaptation of the ICC profile and the application of the profile are already independent from each other. In Chap. 6 we have outlined an envisioned system, in which these two are decoupled into distinct processes. It should be rather trivial to implement a system, in which these two processes are communicating by means of a service oriented architecture. The profile adaptation – which is not time critical – could then be easily deployed on a dedicated machine, to reserve the computation cycles for the "payload" of running a dedicated application operating on the colour corrected imagery.

This is, of course, an incomplete list of future enhancements. Other potential improvements have been mentioned in the different chapters. This list, however, can be seen as a summary of the most obvious and pressing topics for future research and implementations based on the research presented.

Glossary

3D	three dimensional
CAT	chromatic adaptation transformation, the conversion of a colour tristimulus for a different illuminant
CCD	charge-coupled device, sensor type used in digital cameras
CIE	International Commission on Illumination (Commission Internationale de L'Éclairage)
CIE CAM02	CIE Colour Appearance Model (2002), correlates for the six technically-defined dimensions of colour appearance: brightness (luminance), lightness, colourfulness, chroma, saturation, and hue
CIE LAB	see CIE $L^*a^*b^*$
CIE $L^*a^*b^*$	CIE 1976 $L^*a^*b^*$ is a device-independent colour space, shortly also called CIE LAB; based directly on non-linearly compressed CIE 1931 XYZ colour space coordinates
CIE LCH	see CIE $L^*C^*h^*_{ab}$
CIE $L^*C^*h^*_{ab}$	Polar description of the CIE $L^*a^*b^*$ colour space. L^* is the luminance (lightness), C^* the chroma (saturation) and h^*_{ab} the hue.
CIE LUV	see CIE $L^*u^*v^*$
CIE $L^*u^*v^*$	CIE 1976 $L^*u^*v^*$ is a device-independent colour space, shortly also called CIE LUV; based directly on non-linearly compressed CIE 1931 XYZ colour space coordinates

clipping	limiting colour values to the minimum or maximum intensity for a channel if a colour falls outside the representable interval of a colour encoding
CLUT	Colour Lookup Table (multidimensional)
CMM	Colour Management Modules, an implementation for ICC compatible colour transformation computations
CMOS	complementary metal-oxide-semiconductor, sensor type used in digital cameras
CMS	Colour Management Systems
CMYK	CMYK (cyan, magenta, yellow, black) colour space; used in subtractive primary colour systems e. g. for printing
CRT	cathode ray tube, a displaying concept used for computer screens and TVs
D_{50}	CIE Standard Illuminant, corresponds to a colour temperature of 5003 K
D_{65}	CIE Standard Illuminant, corresponds to a colour temperature of 6504 K
gamma correction	A nonlinear operation used to code and decode colour channel values in colour space encoding systems. A gamma correction is, in the simplest cases, defined by a power-law expression
gamut	The range of colours that a monitor, scanner, printer, camera, etc. can detect and/or reproduce.
HLS	Hue, Lightness, Saturation colour space
HSV	hue, saturation, value based colour space, an opponent colour space
ICC	International Color Consortium
ICC profile	a profile that describes the colour attributes of a particular device or viewing requirement by defining a mapping between the source or target colour space and a profile connection space; specified by the International Color Consortium

$L^*a^*b^*$	see CIE $L^*a^*b^*$
LMS	long, medium, short value based colour representation in the characteristic of the three types of light sensitive cone cells of the human eye
LUT	Lookup Table
$L^*u^*v^*$	see CIE $L^*u^*v^*$
PCS	profile connection colour space, usually CIE XYZ or CIE LAB
posterisation	color quantisation artifact, also known as banding; may occur when the color depth, sometimes called bit depth, is insufficient to accurately sample a continuous gradation of colours
RAW	image format for not rendered, raw sensor reading of digital camera images
rg-chromaticity	chromaticity description using the red and green channel, normalised through division by the sum or all (red, green, blue) colour channels
RGB	RGB (Red, Green, Blue) colour space
SPD	spectral power distribution
sRGB	standardised RGB (Red, Green, Blue) colour space
UML	Unified Modeling Language, a standardised general-purpose modeling language in the field of software engineering
YUV	colour space consisting of luma (Y) and two chrominance components (U, V), mostly used in TV and video standards

Appendix A

Python/Native Code Integration

Throughout this thesis we are mentioning primarily the use of the Python programming language for specific implementations. These often make use of native code libraries, compiled to binaries that are directly executable by the CPU of a computer. There are various reasons for using such libraries: Maybe because an existing library is to be reused without reimplementation; The execution speed could be critical and a higher performance than pure Python is required; Or the run time needs access to low level interfaces for which access is not provided by Python directly.

One of the grand fundamentals in software engineering is to use the tools that are best suited for a job, and not to prematurely decide on an implementation. That is often easier said than done, in the light of some complimentary requirements (e. g. rapid/easy implementation vs. needed speed of execution or vs. low level access to hardware). The "traditional" way [84] of binding native code to Python through *extending* or *embedding* is quite tedious and requires lots of manual coding in C.

For binding native code to Python by now a larger variety of tools and technologies are available. A categorisation of some of these approaches is given by Bagnall [85]. But we are also providing here a brief overview of a variety of binding tools. The field has been limited to tools that are widely in use today within the community, and that are promising to be future proof as well as not overly complicated to use. These are the contestants with (very brief) notes for use cases that suit their particular strengths:

- Use *Ctypes* [81], if you want to wrap pure C code very easily.
- Use *Boost.Python* [86, 87], if you want to create a more complete API for C++ that also reflects the object oriented nature of your native code, including inheritance into Python, etc.
- Use *cython* [79], if you want to easily speed up and migrate code from Python to speedier native code (Mixing is possible!).

- Use *SWIG* [88], if you want to wrap your code against several dynamic languages.

Kloss has given several talks on the wrapping of codes with Boost.Python[1]. That talk covered integrating the native code sensibly into Python by inheriting from C++ classes, overriding functionality and automatically generating the bindings using the *Py++* code generator and the software build tool *SCons*.

In this given research we are dealing with the example of wrapping the Little CMS (LCMS) library [42, 43]. LCMS is implemented in pure C, so for the ease of wrapping it, and being able to use the bindings for multiple platforms, we have decided to use *Ctypes* for this project.

A.1 Wrapping Little CMS with Ctypes

This section focuses on wrapping shared C libraries using Python's default *Ctypes* package [81], which is by default part of Python since version 2.5. Particularly tools to ease the process (by using code generation) and some best practises will be stressed. We will try to tell a "step-by-step" story of the wrapping and development process, that should be transferable to similar problems.

As an example the creation of a wrapper for the Little CMS colour management library [42, 43] is outlined. The library offers excellent features, and ships with "official" Python bindings (using *SWIG* [88]), but unfortunately with several shortcomings (incompleteness, un-Pythonic API, complex to use, etc.). So out of need and frustration the initial steps towards alternative Python bindings were undertaken. An alternative would be to fix or improve the bindings using *SWIG,* though for its simplicity and practicality we have chosen to use *Ctypes* for the wrapper to the plain C library.

Of course, wrapper code can be written manually, in this case directly using *Ctypes*. We do not provide a tutorial on how to use *Ctypes*. But the familiarity with this package is necessary when attempting to undertake serious library wrapping. The *Ctypes tutorial* and *Ctypes reference* on the project web site [81] are an excellent starting point for this. For extensive libraries and robustness towards an evolving API, code generation proved to be a good approach over manual editing. Code generators exist for *Boost.Python* as well as for *Ctypes* to ease the process of wrapping: *Py++* [89] (for *Boost.Python*) and *CtypesLib's* [81] `h2xml.py` and `xml2py.py`.

Three main reasons have influenced the decision to approach this project using *ctypes:*

- Ubiquity of the binding approach, as *Ctypes* is part of the default distribution.

[1] Talk slides available online under http://www.slideshare.net/XEmacs/thinking-hybrid-pythonc-integration, http://www.slideshare.net/XEmacs/thinking-hybrid-pythonc-integration-247208 and http://www.slideshare.net/XEmacs/thinking-hybrid-pythonc-integration-368593

- No compilation of native code to libraries is necessary. Additionally, this relieves one from installing a number of development tools, and the library wrapper can be approached in a platform independent way.
- The availability of a code generator to automate large portions of the wrapper implementation process for ease and robustness against changes.

The next section will first introduce a simple C example. This example is later migrated to Python code through the various "incarnations" of the Python wrapper. Sect. A.3 introduces how to facilitate the C library code from Python, in this case through code generation. Sect. A.4 explains how to refine the generated code to meet the desired functionality of the wrapper. The library is anything but "Pythonic," so Sect. A.5 explains an object oriented Façade API for the library that features "qualities we love."

We are only outlining some interesting fundamentals of the wrapper building process. Please refer to the source code for more precise details [82].

A.2 The Example

The sample code (listing in Fig. A.1) aims to convert image data from device dependent colour information to a standardised colour space. The input profile results from a device specific characterisation of a Hewlett Packard ScanJet (in the ICC profile `HPSJTW.ICM`). The output is in the standard conformant sRGB output colour space as it is used for the majority of displays on computers. For this a built-in profile from *LittleCMS* is used.

Input and output are characterised through so called "ICC profiles." For the input profile the characterisation is read from a file (line 8), and a built in output profile is used (line 9). The transformation object is set up using the profiles (lines 11–13), specifying the colour encoding in the in- and output as well as some further parameters not worth discussing here. In the **`for`** loop (lines 15–21) the image data is transformed line by line, operating on the number of pixels used per line (necessary as array rows are often padded).

The goal is to provide a suitable and easy to use API to perform the same task in Python.

A.3 Code Generation

Wrapping C data types, functions, constants, etc. with *Ctypes* is not particularly difficult. The tutorial, project web site and documentation on the wiki introduce this concept quite well. But in the presence of an existing larger library, manual wrapping can be tedious and error prone, as well as hard to keep consistent with the library in case of changes. This is especially true when the library is maintained by someone else. Therefore, it is advisable to generate the wrapper code.

```c
#include "lcms.h"

int correctColour(void) {
    cmsHPROFILE inProfile, outProfile;
    cmsHTRANSFORM myTransform;
    int i;

    inProfile  = cmsOpenProfileFromFile("HPSJTW.ICM", "r");
    outProfile = cmsCreate_sRGBProfile();

    myTransform = cmsCreateTransform(inProfile, TYPE_RGB_8,
                                     outProfile, TYPE_RGB_8,
                                     INTENT_PERCEPTUAL, 0);

    for (i = 0; i < scanLines; i++) {
        /* Skipped pointer handling of buffers.  */
        cmsDoTransform(myTransform,
                       pointerToYourInBuffer,
                       pointerToYourOutBuffer,
                       numberOfPixelsPerScanLine);
    }

    cmsDeleteTransform(myTransform);
    cmsCloseProfile(inProfile);
    cmsCloseProfile(outProfile);

    return 0;
}
```

Figure A.1: Example in C using the *LittleCMS* library directly.

Thomas Heller, the author of *Ctypes* has implemented a corresponding project *CtypesLib* that includes tools for code generation. The tool chain consists of two parts, the parser (for header files) and the code generator.

A.3.1 Parsing the Header File

The C header files are parsed by the tool `h2xml`. In the background it uses GCCXML, a GCC compiler that parses the code and generates an XML tree representation. Therefore, usually the same compiler that builds the binary of the library can be used to analyse the sources for the code generation. Alternative parsers often have problems determining a 100 % proper interpretation of the code. This is particularly true in the case of C code containing pre-processor macros, which can "commit" massively complex things.

A.3.2 Generating the Wrapper

In the next stage the parser tree in XML format is taken to generate the binding code in Python using *Ctypes*. This task is performed by the `xml2py` tool. The generator can be configured in its actions by means of switches passed to it. Of particular interest here are the `-k` and the `-r` switches. The former defines the kind of types to include in the output. In this case the **`#define`**s, functions, structure and union definitions are of interest, yielding `-kdfs`. Note: Dependencies are resolved automatically. The `-r` switch takes a regular expression the generator uses to identify symbols to generate code for. The full argument list is shown in the listing in Fig. A.2 (lines 11–15). The generated code is written to a Python module, in this case `_lcms`. It is made private by convention (leading underscore) to indicate that it is *not* to be used or modified directly.

A.3.3 Automating the Generator

Both `h2xml` and `xml2py` are Python scrips. Therefore, the generation process can be automated in a simple generator script. This makes all steps reproducible, documents the used settings, and makes the process robust towards evolutionary (smaller) changes in the C API. A largely simplified version is in the listing of Fig. A.2.

```
1  # Skipped declaration of paths.
2  HEADER_FILE = 'lcms.h'
3  header_basename = os.path.splitext(HEADER_FILE)[0]

5  h2xml.main(['h2xml.py', header_path,
6              '-c',
7              '-o',
8              '%s.xml' % header_basename])

10 SYMBOLS = ['cms.*', 'TYPE_.*', 'PT_.*', 'ic.*', 'LPcms.*', ...]
11 xml2py.main(['xml2py.py', '-kdfs',
12              '-l%s' % library_path,
13              '-o', module_path,
14              '-r%s' % '|'.join(SYMBOLS),
15              '%s.xml' % header_basename]
```

Figure A.2: Essential parts of the code generator script.

Generated code should *never* be edited manually. As some modification will be necessary to achieve the desired functionality (see Sect. A.4), automation becomes essential to yield reproducible results. Due to some shortcomings (see Sect. A.4) of the generated code however, some editing was necessary. This modification has also been integrated into the generator script to fully remove the need of manual editing.

A.4 Refining the C API

In the current version of *Ctypes* in Python 2.5 it is not possible to add e. g. `__repr__()` or `__str__()` methods to data types. Also, code for loading the shared library in a platform independent way needs to be "patched" into the generated code. A function in the code generator reads the whole generated module `_lcms` and writes it back to the file system, and in the course replacing three lines from the beginning of the file with the code snippet from the listing in Fig. A.3.

```
1  from _setup import *
2  import _setup

4  _libraries = {}
5  _libraries['/usr/lib/liblcms.so.1'] = _setup._init()
```

Figure A.3: Lines to be patched into the generated module `_lcms`.

`_setup` (listing in Fig. A.4) "monkey patches"[2] the class `ctypes.Structure` to include a `__repr__()` method (lines 4–10) for ease of use when representing wrapped objects for output. Furthermore, the loading of the shared library (DLL in Windows lingo) is abstracted to work in a platform independent way using the system's default search mechanism (lines 12–13).

```
1  import ctypes
2  from ctypes.util import find_library

4  class Structure(ctypes.Structure):
5      def __repr__(self):
6          """Print fields of the object."""
7          res = []
8          for field in self._fields_:
9              res.append('%s=%s' % (field[0], repr(getattr(self, field[0]))))
10         return '%s(%s)' % (self.__class__.__name__, ', '.join(res))

12 def _init():
13     return ctypes.cdll.LoadLibrary(find_library('lcms'))
```

Figure A.4: Extract from module `_setup.py`.

A.4.1 Creating the Basic Wrapper

Further modifications are less invasive. For this, the C API is refined into a module `c_lcms`. This module imports *everything* from the `generated._lcms` and overrides or adds certain

[2] A monkey patch is a way to extend or modify the runtime code of dynamic languages without altering the original source code: http://en.wikipedia.org/wiki/Monkey_patch

functionality individually (again through "monkey patching"). These are intended to make the C API a little bit easier to use through some helper functions, but mainly to make the new bindings more compatible with and similar to the official *SWIG* bindings (packaged together with *LittleCMS*). The wrapped C API can be used from Python (see Sect. A.4.2). Although, it still requires closing, freeing or deleting from the code after use, and `c_lcms` objects/structures do not feature methods for operations. This shortcoming will be solved later.

A.4.2 `c_lcms` Example

The wrapped raw C API in Python behaves in exactly the same way, it is just implemented in Python syntax (listing in Fig. A.5).

```
1  from c_lcms import *
2
3  def correctColour():
4      inProfile  = cmsOpenProfileFromFile('HPSJTW.ICM', 'r')
5      outProfile = cmsCreate_sRGBProfile()
6
7      myTransform = cmsCreateTransform(inProfile, TYPE_RGB_8,
8                                       outProfile, TYPE_RGB_8,
9                                       INTENT_PERCEPTUAL, 0)
10
11     for line in scanLines:
12         # Skipped handling of buffers.
13         cmsDoTransform(myTransform,
14                        yourInBuffer,
15                        yourOutBuffer,
16                        numberOfPixelsPerScanLine)
17
18     cmsDeleteTransform(myTransform)
19     cmsCloseProfile(inProfile)
20     cmsCloseProfile(outProfile)
```

Figure A.5: Example using the basic API of the `c_lcms` module.

A.5 A Pythonic API

To create the usual pleasant "batteries included" feeling when working with code in Python, another module – `littlecms` – was manually created, implementing the *Façade Design Pattern*. From here on we are moving away from the original C-like API. This high level object oriented Façade takes care of the internal handling of tedious and error prone operations. It also performs sanity checking and automatic detection for certain crucial parameters passed to the C API. This has drastically reduced problems with the low level nature of the underlying C library.

A.5.1 `littlecms` Example

Using `littlecms` the API is now object oriented (listing in Fig. A.6) with a `doTransform()` method on the `myTransform` object. But there are a few more interesting benefits of this API:

- Automatic disposing of C API instances hidden inside the `Profile` and `Transform` classes.
- Largely reduced code size with an easily comprehensible structure.
- Redundant passing of information (e. g. the in- and output colour spaces) is determined within the `Transform` constructor from information available in the `Profile` objects.
- Uses *NumPy* [66] arrays for convenience in the buffers, rather than introducing further custom types. On these data array types and shapes can be automatically matched up.
- The number of pixels for each scan line placed in `yourInBuffer` can usually be detected automatically.
- Compatible with the often used *PIL* [71] library.
- Several sanity checks prevent clashes of erroneously passed buffer sizes, shapes, types, etc. that would otherwise result in a crashed or "hanging" process.

```
1  from littlecms import Profile, PT_RGB, Transform
2
3  def correctColour():
4      inProfile = Profile('HPSJTW.ICM')
5      outProfile = Profile(colourSpace=PT_RGB)
6      myTransform = Transform(inProfile, outProfile)
7
8      for line in scanLines:
9          # Skipped handling of buffers.
10         myTransform.doTransform(yourNumpyInBuffer, yourNumpyOutBuffer)
```

Figure A.6: Example using the object oriented API of the `littlecms` module.

A.6 Conclusions on Python/Native Code Integration

Binding pure C libraries to Python is not very difficult, and the skills can be mastered in a rather short time frame. If done right, these bindings can be quite robust even towards certain changes in the evolving C API without the need of very time consuming manual tracking of all changes. As with many projects for this, it is vital to be able to automate the "mechanical" processes: Beyond the outlined code generation above, an important role

comes to automated code integrity testing (here: using *PyUnit* [90]) as well as an API documentation (here: using *Epydoc* [91]).

Unfortunately, as *CtypesLib* is still work in progress, the whole process did not go as smoothly as described here. It was particularly important to match up working versions properly between GCCXML (which in itself is still in development) and *CtypesLib*. In this case a current GCCXML in version 0.9.0 (as available in Ubuntu Intrepid Ibex, 8.10) required a branch of *CtypesLib* that needed to be checked out through the developer's Subversion repository. Furthermore, it was necessary to develop a fix for the code generator as it failed to generate code for `#define`d floating point constants. The patch has been reported to the author and is now in the source code repository. Also patching into the generated source code for overriding some features and manipulating the library loading code can be considered as being less than elegant.

Library wrapping as described in here was performed on version 1.16 of the *LittleCMS* library. While writing this, the author of LCMS has moved to the next stable version 1.17. Adapting the Python wrapper to this code base was a matter of about 15 minutes of work. The main task was fixing some unit tests due to rounding differences resulting from an improved numerical model within the library. The author of *LittleCMS* made a first preview of the upcoming version 2.0 (an almost complete rewrite) available recently. Adapting to that version took only about a good day of modifications, even though some substantial changes were made to the API. But even for this case only very little amounts of new code had to be written.

Overall, it is foreseeable that this type of library wrapping in the Python world will become more and more ubiquitous, as the tools for it mature. But already at the present time one does not have to fear the process. The time spent initially setting up the environment will be easily saved over all projects phases and iterations. Unfortunately we will probably not see *Ctypes* evolve to a point where it is able to interface to C++ libraries as well. The developers of *Ctypes* and *Py++* (Thomas Heller and Roman Yakovenko) have evaluated potential extensions, but the complex intricacies of C++ name mangling, and the differences within the different compiler implementations, have proven to be too difficult for this. For now, we will have to accept the fact that it will not be possible to interface C++ libraries without an intermediate step of generating "glue code" that will be compiled to a binding library.

Appendix B

Applications in Industry Projects

During the research phase of the thesis, work to several industry projects was contributed. This appendix outlines briefly work done for these projects. These were a baggage identification for airport baggage management systems (Sect. B.1), a service oriented architecture based farm management system (Sect. B.2) and a quality control system for electronic control consoles involving LCD displays (Sect. B.3).

B.1 Baggage Identification

Baggage handling systems at airports are working during the "rush hours" at airports close to the limitations of their capacity. Certain errors in baggage identification do happen over the course a bag takes on the extensive conveyor belt systems. Each misidentified item has to recirculate the system. Due to the nearly complete utilisation of the system already, each of these faults therefore put an additional load onto the system, which is (due to tail backs) over-proportional to the extra time for the item's handling.

The idea of this work was to improve the accuracy of the baggage identification to reduce the ratio of identification failures. Currently, most baggage handling systems perform a complete tag based identification at the time the baggage item enters the main conveyor belt system using automatic bar code readers. At that point of time the bag is "timed" using a photo eye. Due to knowledge of belt speeds and distances the bag is identified at further decision points through timing using further photo eyes and knowledge of the bag order on the belt. Upon mis-measurement of only about 30 mm, a baggage item is flagged as not correctly identifiable and has to undergo a complete tag reading and security scan (X-ray) again. Such differences easily occur if a bag's handle "flips" over or the bag shifts slightly in a turn.

Our approach was to build comparatively cheap "smart" cameras as appliances, that were to be distributed to the decision points, to be mounted above the conveyor belt. The

"smart" cameras contain an embedded computer capable of processing the live image stream and adapting to the current illumination condition. These cameras would observe the belt underneath and compute a set of specific metrics (digital fingerprints) from the images (see Sect. B.1.2). The metrics for each bag were to be sent off to a central matching server comparing the fingerprint to those stored in its database to identify a baggage item (see Sect. B.1.3). To compensate for differences in light quality at different places of the belt as well as changing light conditions, the image appearance is first (colour) corrected and normalised (see Sect. B.1.1). Colour correction for this case is performed in a similar fashion as outlined in this thesis: A foreground/background segmentation is performed, so that static background information can be extracted. Additionally, information from the colour histogramming is fed back to provide information on the change in colour appearance from a "before" state to a "now" state.

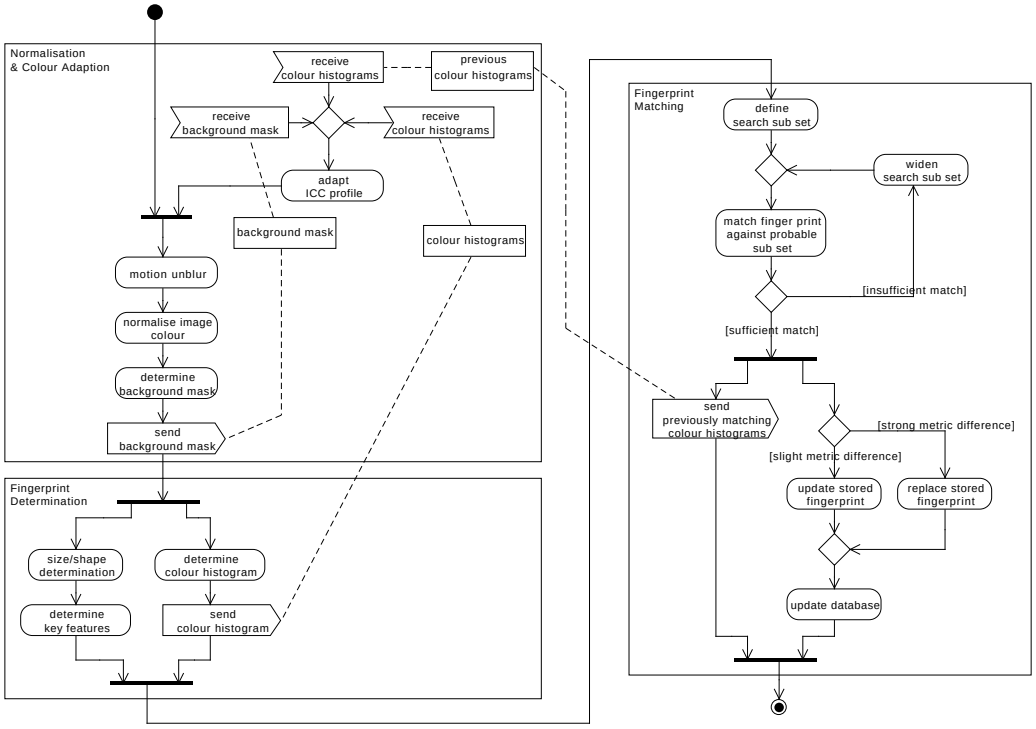

Figure B.1: UML activity diagram of the combined normalisation, visual fingerprinting and matching algorithm (including adaptive colour perception calibration and fingerprint updating) for the proposed system.

Fig. B.1 outlines the processing flow within the proposed system. The two steps on the left hand side (normalisation/colour adaptation and fingerprint determination) operate on the "smart" camera. The fingerprint matching step is to be performed on a central system which communicates with the distributed camera systems. It receives the digital fingerprints and matches them to the ones stored in its database. It also feeds back colour histogram

information of properly identified bags to the cameras to aid them in the adaptation process of the ICC profile (see Chap. 8 and 9).

A prototype of the system has been developed as a distributed system of software components for desktop computers. These components are described in the following sections. Beyond the three functions from Fig. B.1 also a process control is present.

B.1.1 Image Normalization

Images acquired by cameras in distributed locations within the system are deteriorated due to various influences. These influences are:

- motion (resulting in motion blur),
- background (different visual characteristics of the baggage belt and surrounding environment in different locations),
- varying illumination conditions (natural, artificial, changed due to the time of the day and/or external weather conditions, etc.) and
- differences in image capturing devices' perception (differences in the cameras due to model variation, batch production variation and ageing effects).

The background mask can also be used for the ICC profile adaption, as it separates the non-changeable fragments of the image from the changing fractions (bag).

In the prototype, the main application contains of a Python script, which calls into an image processing function in a native C/C++ library for the benefit of speed with the limited resources in an embedded device. This function performs motion un-blurring, an ICC profile colour correction and a statistical background/foreground segmentation. An adaptation of the ICC profiles is currently still missing from the prototype. Communication between the distributed components is implemented through a service oriented architecture using Web Services.

B.1.2 Determination of Visual Fingerprint

On the basis of the normalised image the visual metrics composing the visual fingerprint can be determined. A set of easily determinable and robust properties useful for object identification are determined. Usable for fingerprints are for example these:

- Colour Histograms (hue histogram and saturation/chroma histogram)
- basic size and shape (bounding box)

- pattern structure (extremes as solid colour or fine detailed regular pattern, can be determined analysing the frequency distribution after a Discrete Fourier Transformation)

The fingerprinting code operates on the foreground areas of the normalised image representation as provided by the processing step outlined in Sect. B.1.1. The "fingerprint" contains the metrics relevant for object identification. This step also decides on a heuristic which of the captured frames of an individual item is ideal for matching. A fingerprint derived from that frame then is passed on to the matching server.

Currently, the fingerprint in the prototype contains a logarithmically scaled a^*b^* chromaticity histogram and a bounding box (size, aspect ratio and angle). It is returned from the native function called upon processing a suitable baggage image frame. The fingerprint data is communicated to the matching server via a Web Service call.

B.1.3 Matching Algorithm

The bags are normally moved on the belt in an undisturbed order. Thus, the corresponding fingerprint in the sequence of recorded fingerprints within the system is known. For matching the fingerprint of a given bag against the recorded fingerprints for identification, only a sub set of records around the approximated location will need to be performed. The match will be positive for a match with a high certainty in the proximity of significantly lower correlation certainties.

In the case of an insufficiently distinct positive correlation, the algorithm extends the search domain to locate the fingerprint within a wider range, if the bags' order got disturbed on the conveyor belt. The matching algorithm must provide sufficient capabilities to withstand these errors. The reason is on the one hand the nature of inevitable margins of error of measurements extracted from the image analysis, and on the other hand the fact that details of bags may look different in different locations (e. g. due to a moved bag handle).

To further reduce the potential margin of error in the fingerprint matching, the fingerprint associated with a piece of baggage can be updated. In case of slight changes in the fingerprint, metrics should be updated through averaging of the metrics to decrease the statistical variation. In case of significant changes in metrics (e. g. due to a moved bag handle) the fingerprint would be replaced by the fingerprint of the current perception of the bag.

The information from a previous perception of an item at another location can be used to derive hints for a corrective adaptation of the ICC profile for the respective currently used camera.

In the prototype, the server matches the fingerprint against the database of stored fingerprints, and the result is written to a log file. For the fingerprint matching, currently only the chromaticity histogram is used by performing *colour indexing* as described by Swain

and Ballard [3] and already used in Sec. !4.6.2 of this thesis. The matching server is also implemented in pure Python as the histogram intersection is computationally cheap. Furthermore, the fingerprints in the database of the matching server are currently not updated or replaced.

B.1.4 Process Control

To "orchestrate" the collection of several distributed components a process control instance has been implemented. This application starts the local (and optionally remote) components. During run time, it can request status information through Web Services from all involved software components. Through the same mechanism, also the availability of the components (online/offline) can be controlled, and the system can be stopped and shut down "gracefully" as well.

B.1.5 Industry Partners

This work has been conducted in cooperation with the companies Baggage Sortation Management Ltd. and ecomgroup Ltd. We are thankful for their support and cooperation. They gave the opportunity to develop and test certain algorithms and software for live systems, and to validate its practical robustness through a real world validation.

B.2 Mobile Farm Management

Some farmers have experienced the desire to manage the live stock of their farms more efficiently. But the approach taken for managing the farm should impose as little management overhead as possible, while still providing an efficient digital interface to the key farming data.

The industry partner company has discovered, that farmers do not like to handle excessive paper or have to use specialised data acquisition equipment. But farmers ubiquitously *do* carry mobile phones. An idea was born to employ the many beneficial features of smart phones for this purpose. A mobile application on the smart phone could be used to enter data or to interface various farm appliances wirelessly. Entered or acquired data then could be sent (directly or after buffering) to a server (through web services) for further processing. Additionally, the farmer can use the smart phone as a client device to retrieve and update certain key information required for example for live stock treatments.

In this project, a prototype for a mobile remote client application was implemented. This application consisted of two software components, the mobile client and the server side service implementation.

The phone client was implemented as a Java mobile application (J2ME[1]) featuring a variety of interactive menus. Through the menus and interfaces, a variety of different client-server interactions were triggered. The intention of these was to provide a template for use of the different types of service request message that are possible. Further development could then use these templates to simply design and implement interfaces for the different interactions needed in a productive system.

On the server side, a Python instance was used, building on the Zolera SOAP Infrastructure (ZSI[2]). For communication, it provides the Web Services through the document/literal (wrapped) encoding, using a sub set of the official data types to suite the limitations of the J2ME web service stack (JSR-172[3]). For server side data querying only a database mock-up in local data structures was used.

The prototypical implementation proved to be quite helpful as a template for further implementations. Additionally, it could easily be used to test strategies of requesting individual vs. aggregated data, to test out the effects of network latency to derive access strategies suitable for an interactive application. For our own purposes, it served well as a test bench for evaluating Web Service communication in very restricted (embedded) environments.

B.2.1 Industry Partner

This work has been conducted in cooperation with the company Onto-it Ltd. We are thankful for their support and cooperation. They gave the opportunity to develop and test the viability of mobile and embedded applications for a distributed component communication using Web Services.

B.3 Manufacturing Quality Control

In a electronics manufacturing plant, control panels for home heating systems are manufactured. These panels are equipped with a series of a few control buttons and display elements like LEDs and a monochrome LCD panel. During the assembly of these units, it is possible that one or more of the components are mechanically or electronically faulty, as well as their internal programming could be faulty. In quality control, the units are plugged into a testing rig, which emulates the hardware and sensors of the air conditioning system. A testing operator then keys through a sequence of tests on the unit, testing it for functionality and the output elements for proper responses. This test procedure is not complicated or extremely long lasting. But due to fatigue and high production numbers it was desirable to improve the efficiency and robustness of this test to support the testing operator with partly automated tests.

[1] http://java.sun.com/javame/
[2] http://pywebsvcs.sourceforge.net/
[3] http://java.sun.com/javame/reference/apis/jsr172/

The test automation was to include in the first step a camera based check of the different output channels. For this purpose, a camera was mounted in a fixed position above the control console. The capturing loop detects changes in state on the displays through observing a series of defined regions of interest within the image. The regions of interest then are binarised (converted from colour images to black/white images). The quality assurance model then chose for each item to check a test image acquired at training of a working unit. A registration of the test segment over the current frame is performed to compensate for lateral shifts. Then an image correlation metric was determined to check for the full functionality of all LCD segments and LEDs under scrutiny. Through thresholding, a pass or fail result for the unit under test for that certain step was determined.

At the current time, the project is not finished, yet. But a prototype of the test definition through training as well as checks for functionality of LEDs and arbitrarily shaped LCD elements within the display have been developed. These checks are all working, although the work still requires further improvement. Some faults are still a challenge that is to be solved: small segments as elements of larger LCD structures, the discrimination of LCD segments with very small gaps between them (scale of roughly two image pixels), as well as display blemishes (shadows, stains, blots) that cannot be easily captured by a binarised image.

All image processing has been conducted through Python bindings to the OpenCV [92] library for image processing and machine vision. Some more computationally expensive routines for analysis have been implemented in a compiled C library, which was bound to and called from the Python application. Prototyping as well as the execution speed of the application was far more than satisfactory using this approach.

In the future, it is anticipated to extend this ongoing development. The input controls of the units are to be mechanically actuated through some automated control. Also many devices are to be tested simultaneously, so that an operator can load and unload a whole batch of units into a larger testing rig, freeing up human capacity for handling the produced goods.

B.3.1 Industry Partner

This work has been conducted in cooperation with the company Quick Circuit Electronic Manufacturing & Design Ltd. We are thankful for their support and cooperation. They gave the opportunity to develop and test for binary discretisation of image areas for object segmentation. Additionally insights into methods of image correlation could be gathered.

Appendix C

Simulation Project

A colleague at the institute, Paul Cowpertwait, is strongly involved in statistical rain fall modeling and simulation. He has developed point process models of rain fall [93, 94], which have been tested and fitted to historical meteorological data. These have recently been extended from a temporal model at a distinct point, to include a spacial extent over a whole catchment.

The nature of this simulation project requiring mathematical processing that was easy to implement and modify, as well as fast in execution. Due to this, we have cooperated on the implementation with the statistics discipline for the theoretical foundation. Our strong experience in multi-dimensional data visualisation proved to be a very welcome bonus for the cooperation in this project.

The screen shot in Fig. C.1 shows the simulation output at a given simulation time. The left plot contains the cumulative rain fall for the catchment according to the colour key on the very left. The right plot shows the current rain fall contribution from the last simulation time step. Single frames like this one are aggregated over the whole simulation to a movie file, which can easily be analysed visually in almost arbitrary time resolutions.

For experimentation, validation and visualisation we have implemented a simulation of this model in Python. First, in Sect. C.1 below, we are discussing the architecture of the implementation to cope with the task. Afterwards, in Sect. C.2 we will cover the tools used, and give some reasoning on the choices made.

C.1 Software Architecture

Statistical parameters are all stored in an easy to manage configuration file, so that the simulation code itself does not need altering. This way, it is easy to automatically compute experiment series for parameter studies or to tune the parameters to fit realistic data.

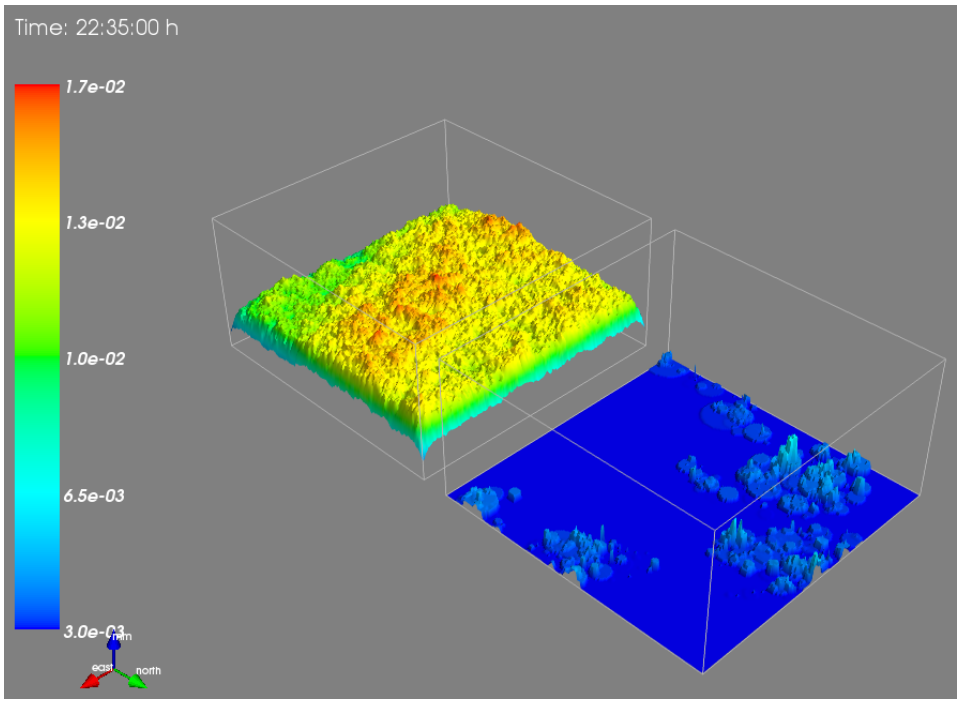

Figure C.1: Screen shot of simulation results at a point of time.

In the following, we are using some UML (Unified Modelling Language) diagrams to outline the architecture of the implementation. First, we provide a class diagram outlining the object oriented architecture of the code. Secondly, a sequence diagram outlines the typical actions taken within a simulation time step.

The simulation code itself is structured into four distinct parts. These parts are illustrated in the class diagram Fig. C.2, their grouping is highlighted using different colours:

- The simulation manager (red),
- a simulation catchment (grey),
- rain events (green),
- and a visualiser (yellow).

The *simulation manager* (`RainfallSimulation`) starts the process, acts as a container for the other three parts, and controls the macroscopic simulation flow. With the beginning of a simulation, it instantiates a *simulation catchment* (`Catchment`) defines the geographical extent, its granularity for the simulation, as well as some administrative information needed for the simulation. Next, an instance of the *visualiser* (`VisualiseResult`) is created, which will later update a 3-D plot of the catchment's total rain as well as the incremental contribution after every time step of the simulation.

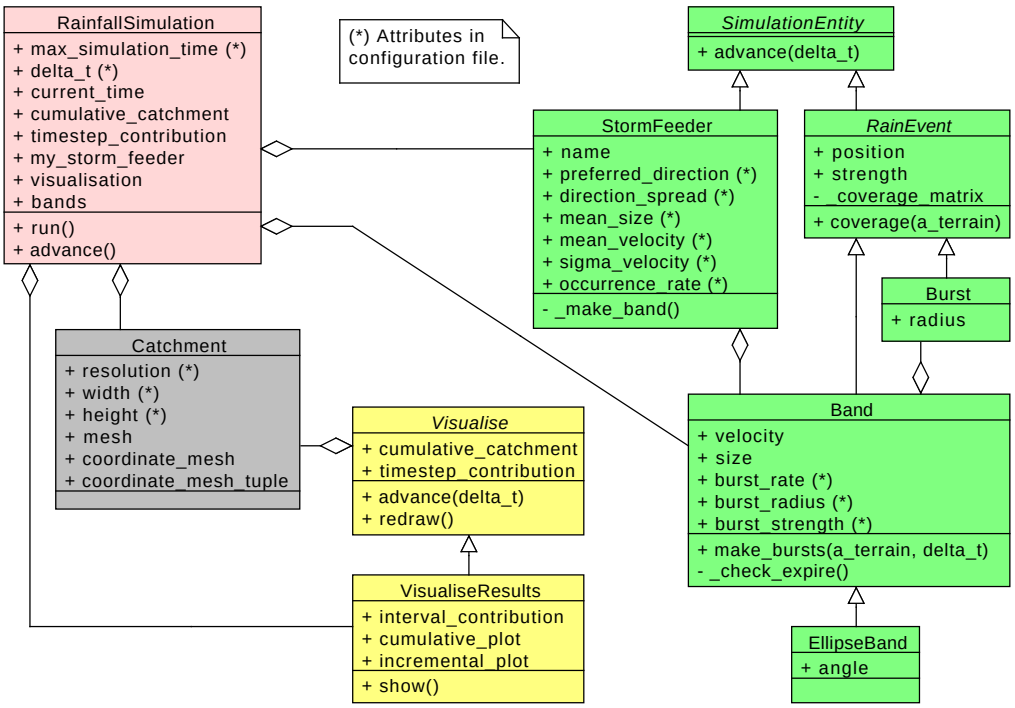

Figure C.2: Class diagram outlining the object oriented composition of the simulation implementation.

In the core of the simulation, the actual rain modelling is composed of a hierarchy of `SimulationEntity` classes. These represent *storms* (`StormFeeder`), as well as a hierarchy of the different types of (nested) rain events: *rain bands* (`Band` and `EllipseBand`) and *rain bursts* (`Burst`). A storm, while active, creates bands by the configured random parameters. These can be either – in the most simple case – linear bands of a certain width (and infinite extent in length) that move in a specific direction. For a more refined simulation, these can be modelled more realistically as ellipses, which can also pass only partly through the catchment. A rain band will create rain bursts during it's time over the catchment. These rain bursts lie within the rain band's coverage area, and they are "atomic" rain events that rain off instantaneously over a circular area.

Each of the actual rain event classes inherits all the functionality of the parent class. `EllipseBand` contains all the functionality of `Band`, `RainEvent` and `SimulationEntity`. It only overrides and extends behaviour as needed, avoiding implementation of redundant common code fragments. These are mainly basic "book keeping" facilities needed for the rain events, such as position and strength, as well as means to advance their state to the next time step, as well as to create access to determine their current coverage of the catchment.

Another advantage is, that it is much easier to implement new rain event types (e. g. with different geometries or behaviours) by inheriting from a suitable parent type. Only

functionality changes and further attributes are added. The `EllipseBand` for example uses an additional attribute `angle`, `size` is a vector rather than a scalar (major and minor semi-axes), and the `coverage()` method is modified to cater for the computation of the elliptical coverage matrix.

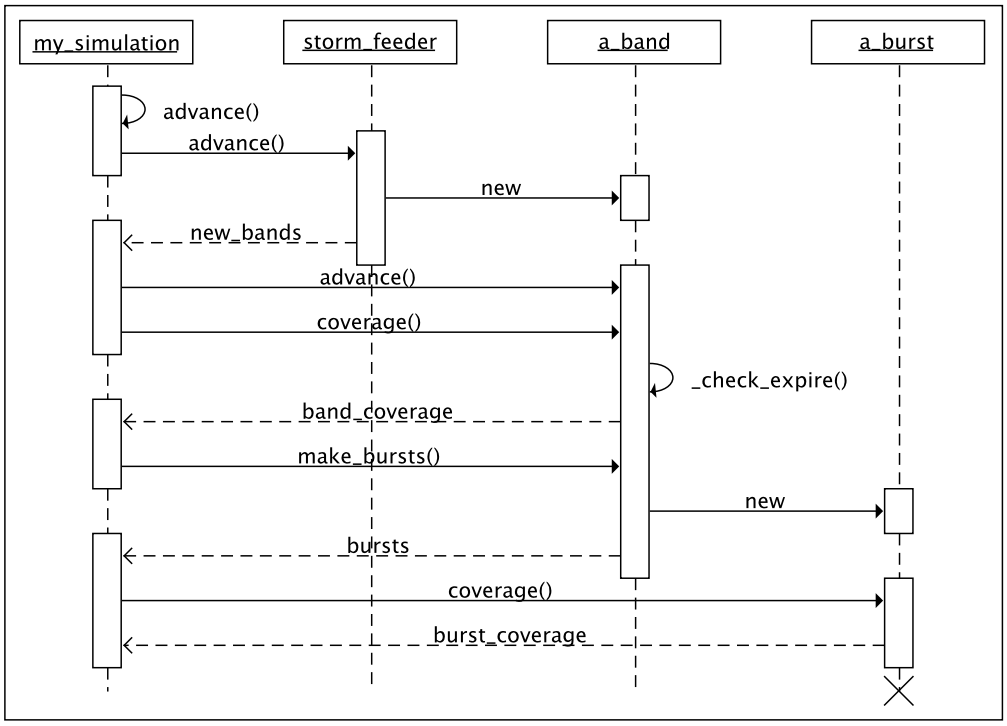

Figure C.3: Sequence diagram of actions during a simulation time step that creates a new rain band.

In the sequence diagram of Fig. C.3, the process of how the actual rain fall simulation is conducted internally is outlined. It all starts out with the call to `advance()` in the simulation itself. This function increases the simulation time by the time step Δt and triggers the following chain of actions in the various rain events: First of all, the storm feeder is "asked" whether new rain bands are due for creation. In the case of new bands the storm feeder will create them according to the desired band type using configured statistical distributions for direction, size, strength, etc. This new band is returned to the simulation, to be managed for the future until the band has passed through the simulation's catchment.

Next, the simulation requests each band in turn to advance the same time step Δt and to determine their coverage of the catchment.

This coverage matrix is used in all rain event types. It consists of a binary array containing a value of 1.0 only for the elements of the catchment matrix that are covered by the current rain event, and 0.0 otherwise. With the help of this matrix, it is very easy to

compute a time step's contribution by using the strength of the matrix as a scalar multiplier, or to identify cell elements that can be used for determining valid centres of rain bursts from a band. To reduce the need for a repeated computation, this matrix is cached for cases it has not changed.

At the computation of the coverage matrix, each band checks whether it has "expired" (completely passed through the catchment), so that it can be potentially removed from the list of managed bands. With the band's coverage, each band is requested to create a series of bursts, which are returned to the simulation. Again, the simulation asks now each of these new bursts for their coverage, and it applies the burst strength with the covered area onto the catchment. The bursts rain off instantaneously, so they expire immediately and are deleted automatically.

This nested cycle is repeated for every time step, in which each band is advanced, for which each burst is created and processed. Repeat ... until done ...

C.2 Implementation

As mentioned initially, the simulation has been implemented in the dynamically typed and interpreted programming language Python. As described in Sect. C.1, the task has been accomplished in an object oriented way.

Our choice for using Python was easily made, as it is a high productivity language that enables the programmer to accomplish programming tasks with only very little overhead. Furthermore, a huge variety of libraries and tools is available that are extremely productive as well as fast.

The core of the simulation involves large amounts of mathematical computations, vector and matrix operations, as well as various statistical random distributions. For this purpose we are using NumPy [67] from the SciPy Project[1] (Scientific Tools for Python), bringing Python up to par with the commercial Matlab™. NumPy is implemented in a natively compiled library. All major mathematical operations are performed using NumPy, which circumvents the need to conduct all calculations in the comparably slow interpreted pure Python. Besides the fact that NumPy offers a huge variety of mathematical implementations (e. g. vector algebra, matrix manipulations, statistical distributions, simple solvers, etc.), our biggest gain in this project is the ability to perform operations on a whole array, all conducted within the optimised bowels of the native C library. A matrix dot-product operation for example can be applied to a whole array of vectors at once.

The only other non-standard Python package used is Mayavi [95]. We use it for fast, hardware accelerated 3-D visualisation of the water levels in the rain catchment. Mayavi also provides functionality to export a scene as an image to the file system. For post simulation analysis, these frames are collectively taken and transformed to a movie file, which allows a

[1] http://scipy.org/

visual evaluation of the results in a time accelerated fashion lasting a minute, rather than an hour needed for computation time.

The choice of using Python together with NumPy and Mayavi has resulted in a fast and clean implementation, with good execution performance, towards a simulation package that uses open source tools and libraries only. No commercial or proprietary creatures were harmed in the process of this development.

C.3 Conclusions for Simulation Project

The implementation of the simulation code base has been accomplished in a very short time, it took only a few days. Due to the powerful properties of the employed numerical and visualisation libraries and packages, the code performs quite well. We have now also been able to parallelise certain code sections to take advantage of multi-core compute architectures.

Due to the ease and flexibility of the implementation language Python, the simulation has become a research tool for post graduate research students in statistics with no significant programming literacy. New simulations and parameterisations are easily configured. Automatic runs can be scripted with ease to run long lasting parameter studies, for example over night or weekends. It is also possible to take influence on the implementation or extend existing models, as it has been demonstrated to simulate elliptical bands over the previously existing linear band geometry.

As a result, the sequence of images generated by the simulation is assembled to a video file. For a visual analysis, this video file can be played in virtually any speed, as well as paused at any point. A qualitative check of the results is therefore very easy. A quantitative check can be performed with the distribution of rain within the catchment along with the simulation output for the different rain events written to a log file during the process of the simulation.

Appendix D

Various Python Tools and Techniques

Due to the strong involvement in coding up tools using the Python programming language, a variety of specific Python topics was examined during the research undertaken. One of these is already described at length in Appendix A. Some other topics in this field are briefly outlined in this appendix.

Firstly, parallel and distributed programming in Python using today's supportive tools are outlined in Sect. D.1. This section summarises the content of previously held presentations[1].

Then, Sect. D.2 tries to dive into certain aspects of graphical visualisation of data. Specifically focusing on the plotting of (multi-dimensional) data using 2-D and 3-D tools, which can update plots at run-time of an application, producing or acquiring new or updated data during its run time. Other visualisation tools for example for graph visualisation, post computation rendering and interactive visual data exploration are intentionally left out. This content also has been presented[2] previously at a conference [83].

D.1 Parallel and Distributed Programming

Threading and shared memory parallel programming paradigms are common (thanks to Windows and Java), but in many cases do not work as expected on Python due to the

[1] Talk given at the June 2008 and October 2009 meetings of the New Zealand Python User Group in Auckland and at the German Aerospace Centre's *TechTalk* series in Cologne in August 2008: http://www.slideshare.net/XEmacs/beating-the-sh-out-of-the-gil-multithreading-vs-multiprocessing and http://nzpug.org/MeetingsAuckland/October2009.

[2] Talk given at Kiwi PyCon in November 2009 in Christchurch: http://www.slideshare.net/XEmacs/python-data-plotting-and-visualisation-extravaganza

Global Interpreter Lock (GIL). This problem is inherent for the reference (C) Python implementation. The GIL was introduced to ease the programming of modules and extensions, and to speed up the execution of single threaded code.

It may be considered a fault of Python to not support the scaling of multiple threads to available CPUs or CPU cores. After all, many developers understand the concept of parallel threads. However, shared memory access in parallel threads within one process often leads to the most severe programming errors. Lock management can quickly overwhelm the mental capacity, or at least distract from the goals of the main development.

The Python run time serialises the execution of all code from different application threads to interleave within the process. The GIL can be released by native code (C) extensions, which is commonly done for input/output (I/O) intensive operations, to free execution time for other threads. This approach keeps the Python run time simple, resulting in a much simpler and more maintainable code base as well as speed improvements. Tests have shown, that the lock management necessary for free threading induces an execution speed penalty to only 60–65 % of single-threaded execution [96].

In the Python community the consensus has been reached (after fierce discussions) to rather employ multi-processing approaches. These are constructed after the paradigm to "share data not memory." The data is communicated between the processes by abstraction layers that mostly just work. In the case of better knowledge of the infrastructure, some of the parameters can be altered to yield a better performance. Due to the use of individual processes, fewer inherent dead lock situations are created, and the scenario can also often scale to multiple hosts and not just multiple CPUs (cores). For these reasons, the attempts to remove the GIL towards free threading have been abandoned by now.

D.1.1 Parallelisation Theory

In most cases, bottle necks in code can be distinguished by either CPU bound limitations or I/O bound limitations. In Python, threading is a good solution for coping with many I/O constrained problems. This overview is aiming more at solutions to CPU constrains needing extra attention.

Threads live within a process on one host. Processes in contrast are running independently from each other on the operating system level. They involve a higher memory overhead, but therefore have their own name space and memory, thus introducing less problems through competing access of resources and their management. But on UNIX like operating systems, the process overhead is *very* low, and (C) Python is currently rather inefficient in handling a multitude of threads (*Stackless Python* [97] is a well maintained fork that resolves single-threaded efficiency shortcomings of the standard C Python run time).

Parallel computing models can be distinguished by their level of abstraction [98] (see Table D.1). The lowest level demands programming close to the hardware. It is necessary to specify explicitly how parallelisation, communication and synchronisation are to be

Table D.1: Abstraction of the level of explicitness in the programming style for parallel computing models. *Explicit:* The programmer specifies it in the parallel program. *Implicit:* A compiler/run time system derives it from available information.

level	Parallelism	Communication	Synchronisation
4	implicit		
3	explicit	implicit	
2	explicit		implicit
1	explicit		

performed. This approach is best for performance tuning, but may imply premature optimisation. In contrast, the highest level offers the highest independence towards an executing machine. The computing model handles most or all tasks dealing with parallelisation, and may integrate even automatic parallelisation approaches.

The first mentioned model is not very popular due to its low abstraction, whereas the last is not very successful due to extreme difficulties in the effectiveness of automatically parallelised code. Level 3 is at times successful for specific purposes, while although tedious, level 1 is still popular for general programming, especially in the scientific community. With Python it is possible to consistently develop parallel code on level 2. For compiled languages (like C), often tools and libraries like the Message Passing Interface (MPI) for distributed memory applications or OpenMP for shared memory multi-threading are employed, which lift the implementation approach somewhere between the levels 1 and 2.

D.1.2 Process Based Parallelisation

Approaches used in today's Python can mostly be categorised into "smart multi-processing" and "smart task farming." The former is basically an extension to Python that uses distinct processes for parallelisation. But it masks the complexity of creating and managing these as well as supports communication between the tasks. The latter facilitate parallel operations on tasks submitted to a queue which are executed automatically by worker nodes. The scheduling, distribution of jobs and retrieval of results is handled automatically.

Smart Multi-Processing

Smart multi-processing in Python is provided for example by the `multiprocessing` package by Richard Oudkerk. Its API is designed to be used as a drop-in replacement for the threading API. New processes are created automatically, and code and data structures are automatically serialised and communicated to the forked worker processes for execution. Analogously, results are serialised and communicated back to the calling process. The user does not need to bother with process creation, inter-process communication or synchronisation.

`multiprocessing` is implemented in C, and it is very fast according to tests (e. g. from PEP-371[3]). It allows execution on multiple cores, multiple hosts or clusters. It provides an easy upgrade path as it mimics the `threading` API, including queues, pipes and locks. Objects from the main process are not accessed through shared state, but are synchronised through (various possible) managers. `multiprocessing` hides most details of communication and is directly usable with the default settings. To improve performance, or meet other requirements, the communication and synchronisation can be configured to suite the needs. Since Python in version 2.6, `multiprocessing` is part of the core Python distribution.

Smart Task Farming

Task farming is a technique that is applicable to problems that are usually not depending on inter-process communication to the main process during the execution of tasks executed in parallel. These problems are commonly termed as being "embarrassingly parallel." In many cases, certain types of problems can be decomposed in a way to yield such tasks, that can be computed independently from each other in parallel. These tasks are submitted by the main process to a processing pool server object, which dispatches them to a series of worker nodes. The number of worker nodes determines the number of tasks that can be processed concurrently. Upon completion of a task the server retrieves the results and schedules the next task for execution.

The `multiprocessing` package (see Sect. D.1.2) contains an implementation of this concept in a producer/consumer style. The worker nodes pull jobs from the server upon availability. Code, dependencies and data structures are serialised automatically for the distribution of jobs and the retrieval of results. Also the transfer is handled transparently to the user. The scheme to manage the farmed tasks can be managed in a variety of ways. Most simple is to just submit all the tasks individually, using the `apply_async()` method and returning a "job" object. Upon completion, the results can be retrieved individually from the job objects using the `get()` method. If it matters to retrieve the results immediately upon completion, a callback function can be supplied with the task submission, which is called with the results of the tasks as arguments. A further method of making use of the worker pool with `multiprocessing` is to use a *map-reduce* scheme of invocation. For this, the `map()` (blocking) or `map_async()` (non-blocking) methods on the pool server are used to submit a number of homogeneous jobs with a set of prepared parameters in an iterable object. Again, the asynchronous (non-blocking) version can be used with an optional argument to supply a callback function.

Another implementation of a task farming approach is *Parallel Python*[4] (in the package `pp`) by Vitalii Vanovschi. `pp` is implemented in pure Python, and therefore very platform independent, it also makes use of the default `threading` package functionality internally. It employs the task farming approach much more purely, and it may require some potential

[3] http://www.python.org/dev/peps/pep-0371/
[4] http://parallelpython.com

rethinking or restructuring of parallelisation. Code and data are also serialised and deployed to (remote) worker processes for execution, to avoid manual deployment of code. For security reasons, it can make use of encrypted inter-node communication. Dependencies in the code need to be identified manually by the developer. But an additional feature to ease usage is the automatic detection of available worker nodes – both on a local machine by detecting the number of CPUs/cores, as well as by joining and leaving of remote nodes for example on a cluster. Parallel Python enjoys active development, good documentation and an active community.

Other Parallelisation Tools

A variety of other strategies and tools for parallelisation in Python are available. One of the most classic strategies is to implement parallel codes using individual, distributed processes. Traditionally, communication is facilitated through the *Message Passing Interface* (MPI) specification (an API that allows many computers to communicate with each other). Python features a variety of MPI implementations, such as *pyMPI, Pypar, MPI for Python* or *pypvm*. For MPI type processing, it is required to determine the number of available processors for the entire duration of the computation. MPI standardises the message passing, but it still requires explicitly coded communication for all interaction, thus leaving the problem to solve for the developer to avoid deadlock and race conditions.

In contrast to this, *Bulk Synchronous Parallel* (BSP) processing is a model for designing parallel algorithms by structuring the problem into a sequence of super steps [98, 99]. It is a simplified alternative to the more mainstream MPI (although MPI may be used inside the BSP framework hidden away from the developer). BSP includes means for parallelised (mass) communication and barrier synchronisation for the super steps. The programming model allows a more robust implementation by avoiding deadlocks and easier debugging of the parallel code.

This following list is not complete, but intended to give a brief overview of parallelisation tools more or less commonly encountered.

- *IPython* for parallel computing [100]
  IPython is a comprehensive environment for interactive and exploratory computing. It includes an architecture for interactive parallel computing.

- *ScientificPython*
  ScientificPython (not to be confused with SciPy) is an open source library of scientific tools for common tasks in scientific computing. The tools implemented in this module include support for *Bulk Synchronous Parallel* processing.
  http://dirac.cnrs-orleans.fr/plone/software/scientificpython

- *Reactor based architectures, through Twisted* [101]
  Twisted is normally a networking library. But once a task is divided into sub-processes,

the next problem is to communicate with these sub-processes. Twisted can be used for this to implement concurrency in a truly cross platform universal solution. It operates by implementing the reactor pattern. That means, that everything works asynchronously (no blocking operations), and the remote component will "call back" upon completion (using the Hollywood scheme: "Don't call us, we call you").

- *pprocess* package by P. Boddie,
  `pprocess` provides only elementary support for parallel programming in Python, using a fork-based process creation model in conjunction with a channel-based communications model.
  http://www.boddie.org.uk/python/pprocess.html

- *Pyro*
  Pyro provides a distributed/remote object system, in the fashion of a remote procedure call (RPC) library for the Python programming language.
  http://pyro.sourceforge.net/

- *PyLinda*
  An implementation of the *Linda* coordination language. PyLinda supports several newly proposed extensions to the Linda distributed computing environment, like garbage collection, multiple tuple spaces, bulk tuple operations, etc.
  http://code.google.com/p/pylinda/

D.1.3 Inter-Process Communication

In cases of cooperatively concurrent processing, the tasks have a need to communicate. This communication can be between the individual sub-task and the main process or between the sub-tasks. If certain before mentioned tools for parallelisation are used, it is most likely that their provided infrastructure means for inter-process communication (IPC) are used (e. g. synchronisation managers in `multiprocessing`).

But not all IPC scenarios in concurrent processing environments can be addressed that way. In many cases a distributed component architecture is used, which spans sub-networks that may be partially shielded from each other. Another scenario may be one in which components implemented in different languages or on different platforms may need to interact with each other. In these cases, it is more suitable to rely on more standardised remote object communication means, such as CORBA (Common Object Request Broker Architecture), SOAP based Web Services, etc.

For their flexibility and maturity in overcoming common problems in network based communication, particularly two protocols for IPC have been examined and used: SOAP based Web Services and XMPP based messaging. A big advantage for these protocols is that many problems for them have been addressed already, such as firewall bypassing, authentication, etc. These are issues currently not considered in IPC scenarios conducted within

a (protected) cluster environment, but one may likely face in task based decomposition for more widely distributed environments.

Web Services

Web Services provide the possibility to invoke remote services by using a standardised networking protocol, usually HTTP (Hyper Text Transfer Protocol). Such services are provided by a server, whereas a suitable client is "consuming" them. The service requests and responses are commonly encoded using the SOAP protocol (Simple Object Access Protocol). Encoding of data structures for the request and response in the SOAP "envelope" can be performed in a variety of specified ways. Unfortunately, not all clients and servers do support the same types of encodings, and a variety of "dialects" or sub-sets of these are commonly found as well. For a maximum interoperability, it has been found that the "document/literal (wrapped)" encoding provides the best general interoperability.

The exact definition of the Web Service and its interactions are provided by a WSDL (Web Services Description Language) file in an XML grammar. Although, it is advertised that a WSDL can be generated automatically from an interface definition in a specific language, it is advisable for maximum interoperability to hand craft this file. With the WSDL file the bindings for both the server and the client can be built. This is commonly done in a static process, where a skeleton of "stubs" is generated and afterwards implemented for example through inheritance in an object oriented language. Especially for the client it is also possible to create bindings dynamically at run time.

For Python, several Web Services implementations are available. Particularly the Zolera SOAP Infrastructure (ZSI) [102] is a very versatile and standard compliant implementation which we have used.

For certain scenarios, SOAP Web Services worked quite well. But the biggest issues in general use cases are that Web Service calls are synchronous and they are client-server based. The first implies, that a longer running execution will block the service from returning in a timely manner, blocking the caller and leading to potential time outs in the communication link. Due to the latter, it is tedious to implement bi-directional communication. Every instance in need of being contacted has to implement a server additionally to the client to establish remote connections. This inflates the implementation of the communication layer significantly. Additionally, each node has to be enabled to match a service call end point with an invoker origination, which is not provided by default in the architecture and requires additional overhead.

XMPP Based Messaging

XMPP (Extensible Messaging and Presence Protocol) originates from the Jabber instant messaging protocol. This type of messaging implies a peer-to-peer communication, which

is facilitated through one or more servers. Any available server instance – whether publicly running or self provided – can be used, and servers pass messages from one end point through to the server of another end point. Each message sent includes a sender and receiver identifications through JIDs (Jabber IDs). As only client side implementations are necessary, the communication layer can be strongly simplified.

HTTP and Web Services can be used for IPC. In a similar fashion XMPP can be used for IPC. And just as HTTP is used as a transport medium for SOAP based Web Services, XMPP can be used for service invocation as well by implementing the specification draft for *IO Data* according to the XMPP Extension Protocol (XEP) 244[5]. Exemplary, this has been done for Java based services already [103], but not yet for Python based implementations.

Another big advantage of XMPP based communication and service invocation is, that it is inherently asynchronous. A service call receiver can identify the requester and contact it upon completion of the (potentially longer lasting) execution. The requester does not need to either keep a long lasting synchronous connection open, or receive a job ID, with which it can repeatedly request updates for completion of the job by polling its status.

D.1.4 Parallel and Distributed Programming Conclusions

The inherent implementation of the (C) Python reference using the GIL is making parallelisation efforts to make use of multiple CPU cores difficult, when approached in the traditional way through threads. Resolving the GIL has been shown not to be the only or best solution. Single threaded applications suffer tremendously over the more complicated lock management, and the danger of deadlocks and race conditions through shared memory access increase. Various approaches to "beat the GIL" have been conjured up by the Python community.

So, which approach for parallelisation or task distribution is now the best? This question cannot be answered directly. This section provided an overview of several suitable alternatives for solving the problem. But many of these are complimentary in their ways, and a user needs to evaluate which one to use for what purpose.

The same applies for inter-process communication. Many of the tools mentioned include easy to use IPC features, but in other scenarios – especially in more distributed environments over sub-net boundaries – a separate communication infrastructure may make sense. A developer has to get an overview of which approach is best suited for the current problem at hand.

In summary however, all of the discussed tools save a lot of time. The burdens of parallelisation are moved from an explicit implementation through the developer more towards an implicit management provided by the tool set. Particularly for the sub-tasks of handling the synchronisation and communication, tools provide a tremendous ease to the

[5] XEP-0244: http://xmpp.org/extensions/xep-0244.html

developer. Additionally, many of these approaches scale beyond the resources of just the local machine/memory system to clusters, which cannot be achieved using threading alone.

D.2 Data Plotting and Visualisation

Many applications produce *data*. Data by itself is often not too helpful. To generate *knowledge* out of data, a user usually has to digest the information contained within the data. Many people can extract patterns from information much more easily when the data is visualised. So, data that can be visualised in some way can be much more accessible for the purpose of understanding.

This part of this appendix focuses on the aspect of *data plotting* for these purposes. Data stored in some more or less structured form can be analysed in multiple ways. One aspect of this is post-analysis, which can sometimes conveniently be conducted in an interactive exploration fashion. One may for example import the data into a spreadsheet, or otherwise suitable software tool, allowing the presentation of the data in various ways. A user may "play" with different views, plots, scales, etc. to do so. Another way of accessing information from data visually is to plot it at the time of computation at a much earlier stage of the process. An engineer may for example be interested in the change of a parameter during the long numerical solution of a problem. This way, the process can be interrupted early to avoid an unnecessary waste of time and computational resources in case the solver diverges rather than converges.

The aspects discussed here are centering on displaying and updating data in plots at runtime of a computation. Whether it is a single scalar value plotted over time, or multi-dimensional vector fields are displayed.

Simple, one-dimensional data is boring, as it may just be represented by a single bar or "gauge" like display (e. g. a "speedometer"). Therefore, we are going to look straight at two- (see Sect. D.2.1) and three-dimensional plotting tools (see Sect. D.2.2), used for rendering numerical content through a GUI plot pane. In the minimum, these could be scalar values changing over time/iterations, through two- and three-dimensional data sets, up to snapshots of higher-dimensional data sets.

Some interesting tools that were under scrutiny for use could not be installed due to dependency problems of the currently used development platform (Ubuntu Linux releases 9.04 and 9.10). Nonetheless, these are still very well worth of a brief discussion here.

D.2.1 Two Dimensional Tools

The world of two dimensional plotting is (much more than that of three dimensional tool sets) generously populated by numerous tools. Many of them unmaintained, with no community, or otherwise only suitable for a specific niche. The tools discussed here do not claim to be

the "ideal" tools, but they have proven to have a good persistence over time and feature a larger community of users. The first one is the "godfather" of plotting tools – *Gnuplot* – which is still largely in use, especially in the academic community. In common usage, it is in the Python world nowadays often superseded by the more modern *matplotlib*.

Gnuplot.py

Gnuplot [104, 105] is most undoubtedly regarded by many as the original and most widely used plotting tool in the field. It has a usage history well exceeding a decade, and it is still maintained and enhanced. It is possible to use Gnuplot also for live plotting purposes from Python, usually through Michael Haggerty's Python bindings [106]. These bindings have come a bit of age, but are still very well and easily usable. One just has to avoid minor pitfalls due to the introduction of the `with` key word in Python, but that can be very easily and elegantly circumvented by using the renamed calling parameter `with_`, or by using dictionaries for plotting attributes. Fig. D.1 shows a simple example on how to engage in an updatable data plotting session.

Another point to mention is, that it uses temporary files for storage of the plotting data, rather than passing the data straight into Gnuplot. However, as Gnuplot is very fast, this does not introduce any noticeable delays in plotting compared to "matplotlib" (see next section).

matplotlib

matplotlib [107, 108] is a more recent plotting package that has gained significant traction in the Python community. It provides a very powerful and feature rich environment for plotting. The resulting plot panels, as well as the integrated GUI elements, look much more modern. The plotting panel can also be integrated into various other GUI applications, rather than "living" in a separate window (as compared to Gnuplot). Various generic GUI tool kits (wxPython, Qt or GTK) can be used for the GUI.

One of the features of matplotlib is, that it offers an additional procedural interface (the `pylab` interface) next to the object-oriented API. `pylab` is closely modelled to resemble the interface of MATLAB. The combination of matplotlib with NumPy [67] therefore can be used as a free, modern and easy to learn replacement for the commercial MATLAB package(s).

In Fig. D.2, the Gnuplot example is picked up and modified for using matplotlib. Matplotlib – in contrast to Gnuplot – runs within the same process. As it comes along with its own GUI event loop, a little more boiler plate code is needed to keep the main Python script "alive" to feed in further data.

```
import Gnuplot
import math
import time

SIZE = 100
INCREMENT = 10.0 * math.pi / SIZE

class MyPlotter(object):
    def __init__(self):
        self.values = []
        self.my_plot = Gnuplot.Gnuplot()
        self.current_x = 0.0

    def update(self):
        self.current_x += INCREMENT
        self.values.append(math.sin(self.current_x) / self.current_x)
        self.my_plot.plot(self.values)

    def run(self):
        for i in range(SIZE):
            self.update()
            time.sleep(0.1)

if __name__ == '__main__':
    plotter = MyPlotter()
    plotter.run()
    raw_input('Please press return to finish ...\n')
```

Figure D.1: Gnuplot example, adding values on every call to `update()` to a list for plotting.

Honorary Mentions

As mentioned in the introduction to this section, the user is facing a multitude of available plotting packages for two dimensions, which would clearly exceed the scope of this summary. Some of these, however, deserve a mention without too much further description.

RPy are Python bindings to *R,* a programming language and software environment for statistical computing and graphics [109, 110]. Through *RPy* [111] it is possible to use the features of R almost seamlessly, including graph plotting. In terms of plotting, R is comparable to Gnuplot, each having advantages and disadvantages to each other in different places.

Chaco is an open source Python plotting application toolkit for all types of 2D plotting [112]. It features a huge set of capabilities – both for live plotting, as well as for an interactive data visualization and exploration. Its capabilities go well beyond those of matplotlib. It builds on the very powerful Enthought Tool Suite and Traits by the company

```
import math
import matplotlib
matplotlib.use('GTKAgg') # do this before importing pylab or pyplot
from matplotlib import pyplot
import gobject

SIZE = 100
INCREMENT = 10.0 * math.pi / SIZE

class MyPlotter(object):
    def __init__(self):
        self.current_x = 0.0
        self.x_values = []
        self.y_values = []
        self.my_figure = pyplot.figure()
        self.my_plot = self.my_figure.add_subplot(111)
        self.my_curve = self.my_plot.plot(self.x_values, self.y_values)

    def update(self):
        self.current_x += INCREMENT
        self.x_values.append(self.current_x)
        self.y_values.append(math.sin(self.current_x) / self.current_x)
        self.my_curve[0].set_xdata(self.x_values)
        self.my_curve[0].set_ydata(self.y_values)
        self.my_plot.relim()
        self.my_plot.autoscale_view()
        self.my_figure.canvas.draw()

    def run(self):
        for i in range(SIZE):
            self.update()

if __name__ == '__main__':
    plotter = MyPlotter()
    gobject.idle_add(plotter.run)
    pyplot.show()
```

Figure D.2: Example for matplotlib, similar to the previous Gnuplot example.

Enthought [113], who are also sponsoring and supporting the open source SciPy tools [78]. Unfortunately, at the time of writing, it failed to install and/or run properly both under Ubuntu 9.04 as well as 9.10. Although an excellent tool, it can be rather difficult to match all dependencies and get everything right, even for the distributor's packages.

GracePlot A Python interface [114] to another grandfather of plotting tools, *xmgrace*. Both, GracePlot as well as xmgrace, did not experience much developer attention in the recent past. But this year GracePlot development has been picked up (again), with a release of a 2.0 version of the libraries. GracePlot owns its popularity for being very easy to

use, but it does not nearly offer the features of the previously mentioned tools.

D.2.2 Three Dimensional Tools

Three- and higher-dimensional plotting challenges the plotting tool in a different way. These tools usually all *have to* create the impression of three-dimensionality on a 2-D screen. Usually this is supported by the use of hardware accelerated OpenGL based libraries. Perspective and shading as well as transparency are tricks used to give the impression of spatiality.

This works still quite well for plotting 3-D data sets like surface plots in a 2-D domain. Unfortunately this is in many cases not enough. For example, when a scalar field within a 3-D space is to be visualised, or a 2-D vector field in a 2-D plane. Developers of plotting tools have put much effort into enabling the visualisation of these. Listed below are some of the tricks used to achieve this:

- *Quiver Plots*
  little "vector arrows" at (regular) intervals within the plot (see Fig. D.6)
- *ISO Surfaces*
  regions of equal values in a 3-D space are represented as (partly translucent) 2-D membranes
- *Colour Shading*
  using colour encoding on a surface to indicate a value (see Fig. D.3)
- *Cutting Planes*
  using (movable) planes cross-secting a volume to render the "inside" of a volume on it
- *Stream Lines*
  showing virtual paths of a number of particles as if they were moved through space by the forces of a field

Mayavi

Mayavi [115] is today – just like Chaco – a highly sophisticated plotting tool built on top of the Enthought Tool Suite and Traits, and it is supported by Enthought. For 3-D rendering purposes it uses TVTK, a Pythonic API to the Visualization Toolkit (VTK), which can be considered as being one of the de-facto tools for 3-D data rendering.

Mayavi features all the techniques mentioned above and is particularly well suited for these tricks. For multi-dimensional plotting in scientific applications, Mayavi is a tool definitely worth evaluating, and will probably score among the first places in any evaluation for Python based tools. A simple sample plot is shown in Fig. D.3 with the corresponding script in Fig. D.4.

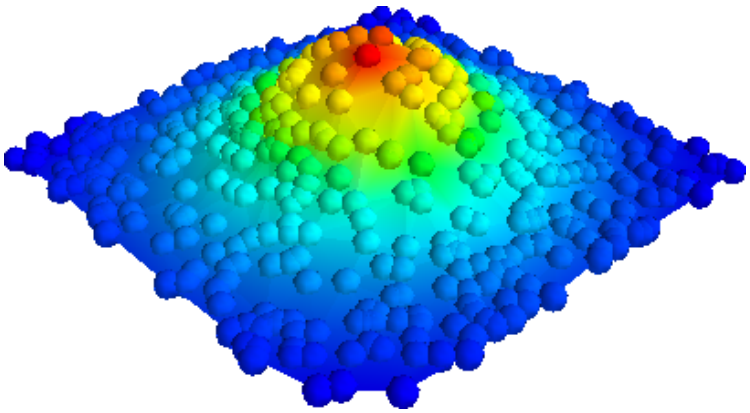

Figure D.3: Surface plot from irregularly sampled data created with Mayavi from script in Fig. D.4.

```
from enthought.mayavi import mlab
import numpy

# Create data with x and y random in the [-2, 2] segment, and z a
# Gaussian function of x and y.
numpy.random.seed(12345)
x = 4 * (numpy.random.random(500) - 0.5)
y = 4 * (numpy.random.random(500) - 0.5)

def f(x, y):
    return numpy.exp(-(x ** 2 + y ** 2))

z = f(x, y)

# Create a figure.
mlab.figure(1, fgcolor=(0, 0, 0), bgcolor=(1, 1, 1))

# Visualize the points.
points = mlab.points3d(x, y, z, z, scale_mode='none', scale_factor=0.2)

# Create and visualize the mesh.
mesh = mlab.pipeline.delaunay2d(points)
surface = mlab.pipeline.surface(mesh)

# Set viewing direction/distance and show.
mlab.view(47, 57, 8.2, (0.1, 0.15, 0.14))
mlab.show()
```

Figure D.4: Example of a surface plot from irregularly sampled data using Mayavi.

Visual Python

Visual Python [116] – or short VPython – is actually *not* a plotting tool. VPython is a tool that aims at making it "easy to create navigable 3-D displays and animations, even for those

```
import visual
import numpy

SIZE = 4

class PointCloud(object):
    def __init__(self):
        self.iterator = None
        self.balls = numpy.zeros([SIZE] * 3, dtype=object)
        for x in range(SIZE):
            for y in range(SIZE):
                for z in range(SIZE):
                    coords = numpy.array([x, y, z], dtype=float)
                    new_sphere = visual.sphere(pos=coords,
                                               radius=0.25,
                                               color=tuple(coords / (SIZE - 1)))
                    self.balls[x, y, z] =  new_sphere

    def update_balls(self):
        for x in range(SIZE):
            for y in range(SIZE):
                for z in range(SIZE):
                    offset = numpy.random.normal(loc=0.0,
                                                 scale=0.01,
                                                 size=3)
                    pos = self.balls[x, y, z].pos
                    self.balls[x, y, z].pos = pos + offset

    def run(self):
        for i in range(10000):
            self.update_balls()

if __name__ == '__main__':
    foo = PointCloud()
    foo.run()
```

Figure D.5: Example simulating a Brownian point cloud using VPython.

with limited programming experience." It is originally developed by David Scherer and now for several years maintained by Bruce Sherwood. It is mostly used for educational purposes (at universities) for teaching without the need to get deeper into programming.

And it was exactly this fact of ease of use, that has helped us many times to easily "knock up" visualisation tools to gain better insight into multi-dimensional data sets. Particularly for two reasons:

1. It is extremely easy to come up with a fast, hardware accelerated visualisation.
2. As it is not "plotting" in the general sense, one has got more influence on the exact representations.

Many of the data sets faced in this thesis were dealing with colour spaces, often encoded with three channels (e. g. RGB). This way, objects for colour transformations could be assigned *directly* the appropriate colour (3-D) to points in 3-D space, yielding a 6-D space that was to explore and analyse (see some samples in Fig. D.6). The figures 4.3, 7.3, 7.4, 7.5, 8.1, 8.3, 8.4, 8.5 and 8.6 were created this way.

VPython renders fast ... very fast! With the help of Python bindings to OpenCV [92] and NumPy, it was possible to capture and analyse frames off a web cam every 1-2 seconds, and display three dimensional colour space distributions (histograms) live (Fig. 8.1). The scene can be rotated, zoomed and panned with no visible performance impact during this live process.

During simulations for example, it is possible to update the `pos` attribute of rendered objects, and without further ado VPython would reposition and update the scene with minimal time impact on the computation. This even works very well with scenes containing more than 5000 nodes rendered as spheres in a cubic mesh (Figs 7.3 and 7.4).

Another very nice feature is, that a POVray export module is available for VPython. With this one, 3-D scenes can be rendered in much superior quality over the raw OpenGL window, by ray tracing output for highest publication quality rendering. The POVray output is not 100 % feature complete to VPython, but it does handle the basic object geometries very reliably and well (Figs. 8.3, 8.4, 8.5 and 8.6).

Mayavi "visual" Module

VTK through TVTK features similar possibilities to render 3-D objects, as VPython does. A first attempt of a compatibility module `enthought.tvtk.tools.visual` has been made by Raashid Baig, a student of Prabhu Ramachandran, the original author of Mayavi. Some aspects of VPython are covered by this `visual` module due to the attempt to retain the API.

This `visual` module therefore can partly be used as a replacement for VPython. But of course one must be aware of the differences:

- Once rendered, it is just as fast as VPython in the direct mouse interaction with the scene, but rendering can be a lot slower, particularly when using the default *wxWindows* GUI back end. This can be partly improved by switching to the *Qt4* back end (by setting the environment variable `ETS_TOOLKIT` to the value of `qt4`).
- The compatibility API is not yet complete, and some things are handled differently.
- The implementation was developed towards compatibility with VPython 3.x (current is version 5.x).
- Range checking through the "traits model" (a "trait" is a type definition that can be used for normal Python object attributes, giving the attributes some additional characteristics) based approach imposes some not so sensibles limitations to object sizing.

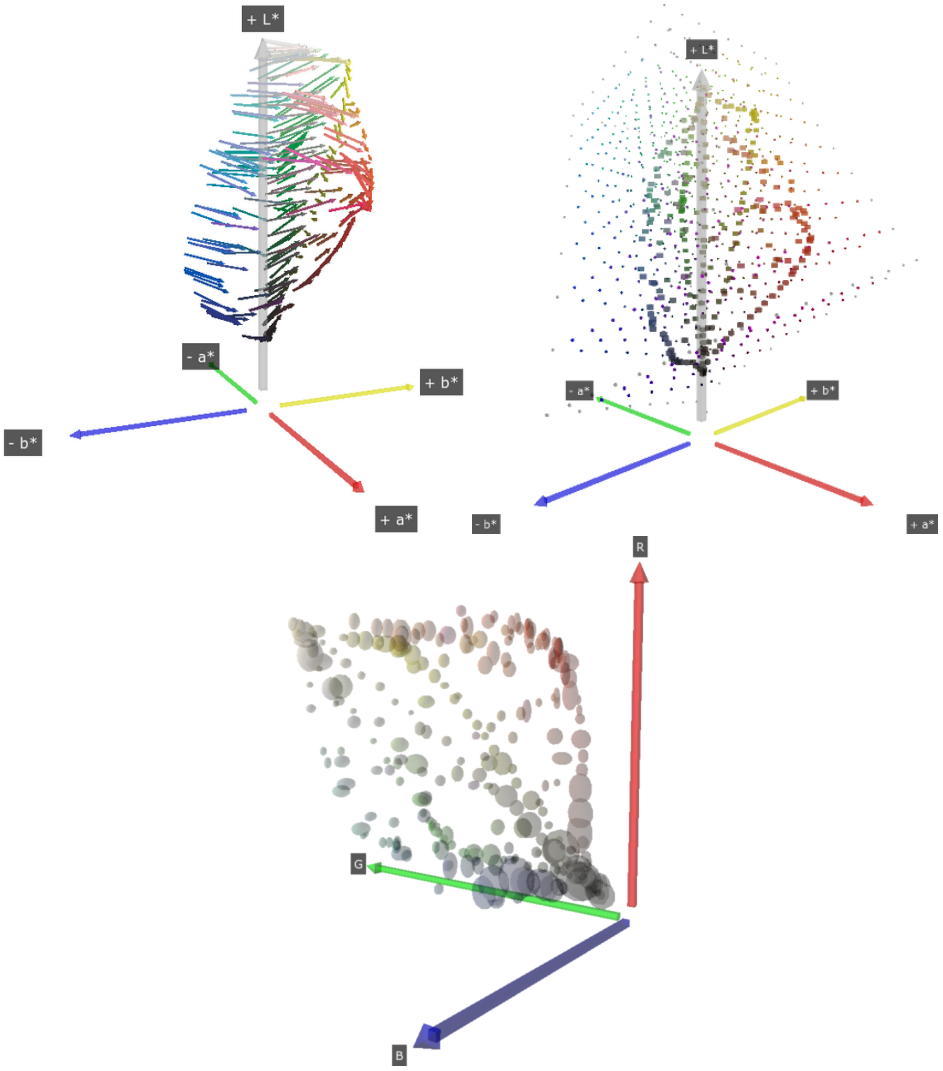

Figure D.6: Some sample plots using VPython. (top left) Colour shifts through arrows (quiver plot). (top right) Cubic point clouds of interpolation volume and data points. (bottom) Colour measurement values in RGB space with measurement errors indicated through ellipsis sizes.

- One can use the very convenient "Traits" based GUI dialogues to alter object properties.
- The implementation of the module in `visual.py` is quite easy to understand and extend.
- The Mayavi `visual` module does currently *not* perform all kinds of wild card imports internally (Which is good! VPython's name space of the `visual` module is "polluted" with wild card imports of `numpy` and `math`.).

```
from enthought.tvtk.tools import visual

# Rest of class definition clipped ...

    def run(self):
        self.iterator = visual.iterate(30, self.update_balls)
        visual.show()
```

Figure D.7: Adaptations to imports and the `run()` method from listing in Fig. D.5 for the Mayavi `visual` module.

Honorary Mentions

"Grandfather" Gnuplot is also capable of plotting in 3-D to some extent. But that is mainly reduced to some rather simple surface plots or curves and scattered points in 3-D. It is therefore not further discussed here. The access to Gnuplot's 3-D features works the same way by using the Gnuplot.py bindings as used for 2-D, just the usual 3-D features (e. g. "splot") are used.

D.2.3 Data Plotting and Visualisation Conclusions

This appendix gives by no means an exhaustive overview of plotting tools. But it shows that data plotting and visualisation – particularly for live plotting – is possible for two and more dimensions. The tools presented give good examples for people who want to engage in scientific data plotting and visualisation, and the features discussed just barely scratch the surface of what is possible. Everybody has to evaluate and study each tool for personal needs to come up with a good solution for the current problem at hand. However, all of the discussed tools are worth keeping in the personal "tool chest" for the time when they are needed.

The newer and more fully featured plotting tools (matplotlib, Mayavi, Chaco) commonly run within the same process, and they can be embedded within other (GUI) applications. Unfortunately, these features come at a cost by introducing further steps in order to make them "play nice" with the general core application. This is also mainly one of the reasons why the classics (like Gnuplot) still have a very strong stand in the scientific community, where the goal is functionality over "pretty applications."

All the above discussed plotting and visualisation tools (as well as Chaco) provide the feature to export rendered frames to image files. These can be used for documentation purposes, or to produce more advanced output compilations in the form of movie sequences compiled from them.

Bibliography

[1] M. Ebner, *Color Constancy*, ser. Imaging Science and Technology, M. A. Kriss, Ed. West Sussex, England: Wiley-IS&T, 2007.

[2] K. Barnard, "Practical Colour Constancy," Ph.D. dissertation, Simon Fraser University, School of Computing, 1999. [Online]. Available: http://kobus.ca/research/publications/PHD-99/

[3] M. J. Swain and D. H. Ballard, "Colour Indexing," *International Journal of Computer Vision*, vol. 7, pp. 11–32, 1991.

[4] G. Pass and R. Zabih, "Comparing Images Using Joint Histograms," *Journal of Multimedia Systems*, vol. 7, no. 3, pp. 234–24, May 1999.

[5] P. Green, "Digital photography color management basics," http://www.color.org/ICC_white_paper_20_Digital_photography_color_management_basics.pdf, International Color Consortium, Whitepaper 20, April 2005. [Online]. Available: http://www.color.org/whitepapers.html

[6] ——, "Using ICC profiles with digital camera images," http://www.color.org/ICC_white_paper_17_ICC_profiles_with_camera_images.pdf, International Color Consortium, Whitepaper 17, April 2005. [Online]. Available: http://www.color.org/whitepapers.html

[7] I. Newton, *Opticks*. Royal Society, London, 1704.

[8] J. W. von Goethe, *Zur Farbenlehre (Theory of Colours)*. Stuttgart, Tübingen: Cotta, 1810. [Online]. Available: http://www.textlog.de/goethe_farben.html

[9] S. M. Courtney, G. Buchsbaum, and L. H. Finkel, "Biologically-Based Neural Network Model of Color Constancy and Color Contrast," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 4, Baltimore, MD, USA, June 1992, pp. 55–60.

[10] G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2nd ed. New York: John Wiley & Sons, Inc., 1982.

[11] S. A. Shafer, "Using Color to Separate Reflection Components," *Color Research and Application*, vol. 10, no. 4, pp. 210–218, December 1985.

[12] H.-C. Lee, "Method for computing the scene-illuminant chromaticity from specular highlights," *Journal of the Optical Society of America. A, Optics and image science*, vol. 3, no. 10, pp. 1694–1699, October 1986.

[13] M. Yamaguchi, T. Teraji, K. Ohsawa, T. Uchiyama, H. Motomura, Y. Murakami, and N. Ohyama, "Color image reproduction based on the multispectral and multiprimary imaging: experimental evaluation," in *Proceedings of SPIE – The International Society for Optical Engineering*, vol. 4663, San Jose, CA, USA, 2002, pp. 15–26.

[14] M. Yamaguchi, H. Haneishi, and N. Ohyama, "Beyond Red-Green-Blue (RGB): Spectrum-Based Color Imaging Technology," *Journal of Imaging Science and Technology*, vol. 52, no. 1, pp. 010 201–(15), January 2008.

[15] E. F. Glynn, "Chromaticity Diagrams Lab Report," efg's Computer Lab and Reference Library, Tech. Rep., 1999. [Online]. Available: http://www.efg2.com/Lab/Graphics/Colors/Chromaticity.htm

[16] G. Hoffmann, "CIE Color Space," Fachhochschule Emden, Tech. Rep., 2000.

[17] D. Pascale, "A review of RGB color spaces," The BabelColor Company, Montreal, Canada, Tech. Rep., October 2003.

[18] E. R. Landa and M. D. Fairchild, "Charting Color from the Eye of the Beholder," *American Scientist*, vol. 93, pp. 436–438, 2005.

[19] B. MacEvoy, "Modern Color Models," http://www.handprint.com/HP/WCL/color7.html, 2005. [Online]. Available: http://www.handprint.com/HP/WCL/color7.html

[20] M. D. Fairchild, *Color Appearance Models*, 2nd ed. Chichester, UK: Wiley-IS&T, 2005.

[21] Wikipedia, "Colour Management," http://en.wikipedia.org/wiki/Colour_management, last accessed March 2010.

[22] B. J. Lindbloom, "Chromatic Adaptation Evaluation," http://www.brucelindbloom.com, Bruce J. Lindbloom, Tech. Rep., 2007, last accessed March 2010. [Online]. Available: http://www.brucelindbloom.com

[23] S. Süsstrunk, J. Holm, and G. D. Finlayson, "Chromatic adaptation performance of different RGB sensors," *IS&T SPIE Electronic Imaging 2001: Color Imaging*, vol. 4300, pp. 172–183, January 2001.

[24] N. Ribe and F. Steinle, "Exploratory Experimentation: Goethe, Land, and Color Theory," *Physics Today*, vol. 55, no. 7, pp. 43–49, July 2002. [Online]. Available: http://www.aip.org/pt/vol-55/iss-7/p43.html

[25] M. Has and T. Newman, "Color Management: Current Practice and The Adoption of a New Standard," http://www.color.org/wpaper1.html, International Color Consortium, Whitepaper 1, 1995. [Online]. Available: http://www.color.org/wpaper1.html

[26] J.-P. Homann, *Digitales Color-Management*. Springer-Verlag, Berlin, Heidelberg, New York, 1998.

[27] M. D. Fairchild, "Spectral Adaptation," *Color Research & Application*, vol. 32, no. 2, pp. 100–112, 2007.

[28] G. West and M. H. Brill, "Necessary and Sufficient Conditions for Von Kries Chromatic Adaptation to Give Color Constancy," *Journal of Mathematical Biology*, vol. 15, no. 2, pp. 249–258, October 1982.

[29] F. Porikli, "Inter-Camera Color Calibration using Cross-Correlation Model Function," in *In Proceedings of IEEE International Conference on Image Processing*, Barcelona, September 2003.

[30] C. Gönner, M. Rous, and K.-F. Kraiss, "Real-Time Adaptive Colour Segmentation for the RoboCup Middle Size League," in *Proceedings of RoboCup 2004: Robot Soccer World Cup VIII*, vol. 3276, Lisbon, Portugal, June 2005, pp. 402–409.

[31] B. Browning and M. Veloso, "Real-time, Adaptive Color-based Robot Vision," in *Proceedings of IROS'05*, Edmonton, Alberta, Canada, August 2005, pp. 3871–3876.

[32] N. H. Reyes, "Colour-Based Object Recognition Analysis and Application," Ph.D. dissertation, De La Salle University, 2004.

[33] H. Shin, A. Husselmann, and N. H. Reyes, "On the Robustness of Fuzzy-Genetic Colour Contrast Fusion with Variable Colour Depth," in *Proceedings of the 16th International Conference on Neural Information Processing (ICPNIP 2009)*, Bangkok, Thailand, December 2009, pp. 667–674.

[34] G. K. Kloss, H. Shin, and N. H. Reyes, "Dynamic Colour Adaptation for Colour Object Tracking," in *Proceedings of the 24th International Image and Vision Computing New Zealand 2009 (IVCNZ 2009)*, Wellington, New Zealand, November 2009, pp. 340–345.

[35] B. Funt, V. Cardei, and K. Barnard, "Learning Color Constancy," in *Proceedings of the 4th Color Imaging Conference: Color Science, Systems and Applications*, Scottsdale, AZ, November 1996, pp. 58–60.

[36] B. Bascle, O. Bernier, and V. Lemaire, "A Statistical Approach for Learning Invariants: Application to Image Color Correction and Learning Invariants to Illumination," in *Proceedings of the 13th International Conference on Neural Information Processing (ICONIP 2006)*, ser. Lecture Notes in Computer Science, vol. 4233. Hong Kong, China: Springer, October 2006, pp. 294–303.

[37] C. Rosenberg, M. Hebert, and S. Thrun, "Color Constancy Using KL-divergence," in *Proceedings of 8th IEEE International Conference on Computer Vision*, vol. 1, Vancouver, BC, July 2001, pp. 239–246.

[38] Wikipedia, "Bayes theorem," http://en.wikipedia.org/wiki/Bayes_theorem, last accessed March 2010.

[39] J. van de Weijer and C. Schmid, "Applying Color Names to Image Description," in *Proceedings of International Conference on Image Processing (ICIP 2007)*, vol. 3, 2007, pp. 493–496.

[40] M. Ebner, "Evolving Color Constancy," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1220–1229, August 2006.

[41] G. K. Kloss, "PyLittleCMS project," http://launchpad.net/pylittlecms, last accessed March 2010.

[42] M. Maria, "LittleCMS: A free color management engine in 100K," Large format printer division of Hewlett-Packard, Barcelona, Spain, Tech. Rep., December 2006, last accessed March 2010. [Online]. Available: http://www.littlecms.com/about.html

[43] ——, "LittleCMS project," http://littlecms.com/, last accessed March 2010.

[44] G. K. Kloss, "Automatic C Library Wrapping – Ctypes from the Trenches," *The Python Papers*, vol. 3, no. 3, pp. –, December 2008, [Online available] http://ojs.pythonpapers.org/index.php/tpp/issue/view/10.

[45] T. Johnson, "The role of ICC profiles in a colour reproduction system," http://www.color.org/whitepapers.xalter, International Color Consortium, Whitepaper 7, December 2004. [Online]. Available: http://color.org/ICC_white_paper_7_role_of_ICC_profiles.pdf

[46] J. Holm, "Color Management – Conceptual Overview, Evolution, Structure & Color Rendering Options," http://www.color.org/whitepapers.xalter, International Color Consortium, Whitepaper 4, December 2004. [Online]. Available: http://color.org/ICC_white_paper4color_management.pdf

[47] T. Hunecke, "ICCView – Internetseite zur Visualisierung von ICC-Profilen mit Hilfe von 3D-Farbraummodellen," ICCView, Tech. Rep., 2002. [Online]. Available: http://www.iccview.de/images/stories/iccview/ICCView-Colormanagement.pdf

[48] S. Süsstrunk, "Color Management – Introduction, ICC architecture, ICC profile, Profile Connection Space (PCS), Module (CMM)," Ecole Polytechnique Fédérale de Lausanne, Tech. Rep., current. [Online]. Available: http://encyclopedia.jrank.org/articles/pages/1074/Color-Management.html

[49] *Specification ICC.1:2004-10 (Profile version 4.2.0.0) [ISO 15076-1:2005]*, International Color Consortium Std., Rev. 1:2003-09, with errata incorporated, 5/22/2006. [Online]. Available: http://color.org/icc_specs2.xalter

[50] G. W. Gill, "ArgyllCMS project," http://argyllcms.com/, last accessed March 2010.

[51] M. Derhak, "Profile Compliance Testing – SampleICC Implementation Notes," http://www.color.org/ICC_white_paper_21-SampleICCProfileCompliance.pdf, International Color Consortium, Whitepaper 21, February 2006. [Online]. Available: http://www.color.org/whitepapers.html

[52] F. Anzani, D. Bosisio, M. Matteucci, and D. G. Sorrenti, "On-Line Color Calibration in Non-stationary Environments," in *Proceedings of RoboCup 2005: Robot Soccer World Cup IX*, ser. Lecture Notes in Artificial Intelligence, vol. 4020/2006, Osaka, Japan, July 2005, pp. 396–407.

[53] M. R. Gupta, E. K. Garcia, and A. Stroilov, "Learning Custom Color Transformations with Adaptive Neighborhoods," *Journal of Electronic Imaging*, vol. 17, no. 3, p. 033005, July 2008. [Online]. Available: http://link.aip.org/link/?JEI/17/033005/1

[54] M. R. Gupta, "Custom Color Enhancements by Statistical Learning," in *Proceedings of the IEEE International Conference on Image Processing (ICIP 2005)*, vol. 3, September 2005, pp. III–968–71.

[55] S.-H. Lee, T.-Y. Kim, and J.-S. Choi, "A Color Correction System using a Color Compensation Chart," in *Proceedings of the International Conference on Hybrid Information Technology (ICHIT 2006)*, vol. 1, November 2006, pp. 409–416.

[56] P. Green, "Common Color Management Workflows & Rendering Intent Usage," http://www.color.org/whitepapers.xalter, International Color Consortium, Whitepaper 9, March 2005. [Online]. Available: http://www.color.org/wpaper1.html

[57] R. Bala, *Digital Color Imaging Handbook*. CRC Press, December 2002, ch. 5 – Device Characterization, pp. 269–384.

[58] M. R. Gupta, E. K. Garcia, and E. Chin, "Adaptive Local Linear Regression With Application to Printer Color Management," *IEEE Transactions on Image Processing*, vol. 17, no. 6, pp. 936–945, 2008.

[59] E. M. Chin, E. K. Garcia, and M. R. Gupta, "Color Management of Printers by Regression over Enclosing Neighborhoods," in *Proceedings of the IEEE International Conference on Image Processing (ICIP 2007)*, vol. 2, September 2007, pp. II–161–II–164.

[60] C. Jing, E. Zhang, and Y. Chen, "The Establishment of Color Printer Profile File Based on ICC Standard," in *Proceedings of the 2nd International Congress on Image and Signal Processing (CISP 2009)*, October 2009, pp. 1–5.

[61] J.-L. Mallet, "Discrete Smooth Interpolation," *ACM Transactions on Graphics (TOG)*, vol. 8, no. 2, pp. 121–144, April 1989.

[62] D. J. Bone, "Adaptive Multi-dimensional Interpolation using Regularized Linear Splines," in *Proceedings of SPIE*, E. R. Dougherty, J. T. Astola, and H. G. Longbotham, Eds., vol. 1902, no. 1. San Jose, CA, USA: SPIE, 1993, pp. 243–253. [Online]. Available: http://link.aip.org/link/?PSI/1902/243/1

[63] ——, "Adaptive Color-Printer Modeling using Regularized Linear Splines," in *Proceedings of SPIE*, R. J. Motta and H. A. Berberian, Eds., vol. 1909, no. 1. San Jose, CA, USA: SPIE, 1993, pp. 104–115. [Online]. Available: http://link.aip.org/link/?PSI/1909/104/1

[64] D. J. Bone and D. Stevenson, "Modelling of Colour Hard Copy Devices using Regularised Linear Splines," in *Proceedings of the APRS workshop on Colour Imaging and Applications*, Canberra, AU, 1994, pp. 21–24.

[65] G. K. Kloss, "Optimierung eines Numerischen Verfahrens zur Simulation von Mehrphasenströmungen," Master's thesis, Universität Dortmund, Dortmund, Germany, September 1998.

[66] T. E. Oliphant, "NumPy Project," http://numpy.scipy.org/, last accessed July 2010.

[67] ——, *Guide to NumPy*, T. E. Oliphant, Ed. Trelgol Publishing, 2006. [Online]. Available: http://www.tramy.us/guidetoscipy.html

[68] Wikipedia, "Simple linear regression," http://en.wikipedia.org/wiki/Simple_linear_regression, last accessed July 2010.

[69] H. Späth, "Fitting affine and orthogonal transformations between two sets of points," *Mathematical Communications*, vol. 9, no. 1, pp. 27–34, June 2004.

[70] G. K. Kloss and T. F. Kloss, "n–Dimensional Linear Vector Field Regression with NumPy," *The Python Papers Source Codes*, vol. 2, pp. –, July 2010, [Online available] http://ojs.pythonpapers.org/index.php/tppsc/issue/view/20.

[71] F. Lundh, "Python Imaging Library (PIL) Project," http://www.pythonware.com/products/pil/, last accessed March 2010.

[72] B. J. Lindbloom, "Color Difference Calculator," http://www.brucelindbloom.com/ColorDifferenceCalc.html, April 2007, last accessed March 2010.

[73] G. Gill, "interpolation for LUT generation," Personal communication, May 2009.

[74] I. N. Bronstein and K. A. Semendjajew, *Taschenbuch der Mathematik*, 7th ed., V. Ziegler, Ed. Zürich and Frankfurt/M.: Verlag Harri Deutsch, 1967.

[75] N. H. Reyes and C. H. Messom, "Identifying Colour Objects with Fuzzy Colour Contrast Fusion," in *Proceedings of the 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems, and FIRA RoboWorld Congress (CIRA 2005)*, 2005.

[76] H. Shin, "Finding Near Optimum Colour Classifiers," Master's thesis, Massey University, 2009.

[77] K. Spaulding and G. Braun, "CIE TC8-05 Color Encoding Criteria, Draft 0.6," CIE division 8 technical committee 05: Communication of Colour Working Group, Tech. Rep., October 2001.

[78] "SciPy – Scientific Python Project," http://www.scipy.org/.

[79] S. Behnel, R. Bradshaw, and G. Ewing, "Cython Project," http://cython.org/, last accessed March 2010.

[80] G. Ewing, "Pyrex Project," http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/, last accessed March 2010.

[81] T. Heller, "Python Ctypes Project," http://starship.python.net/crew/theller/ctypes/, last accessed March 2010.

[82] G. K. Kloss, "Source Code: Automatic C Library Wrapping – Ctypes from the Trenches," *The Python Papers Source Codes*, vol. 1, pp. –, January 2009, [Online available] http://ojs.pythonpapers.org/index.php/tppsc/issue/view/13.

[83] ——, "Python Data Plotting and Visualisation Extravaganza," in *Proceedings of the First Kiwi PyCon*, ser. The Python Papers Monograph, G. K. Kloss, Ed., vol. 1, New Zealand Python User Group. Christchurch, New Zealand: The Python Papers, November 2009. [Online]. Available: http://ojs.pythonpapers.org/index.php/tppm/article/view/121

[84] *Official Python Documentation: Extending and Embedding the Python Interpreter*, Python Software Foundation.

[85] D. Bagnall, "Pyserf. A shortcut for writing C extensions in C," in *Proceedings of the First Kiwi PyCon*, ser. The Python Papers Monograph, G. K. Kloss, Ed., vol. 1, New Zealand Python User Group. Christchurch, New Zealand: The Python Papers, November 2009. [Online]. Available: http://ojs.pythonpapers.org/index.php/tppm/article/view/118

[86] D. Abrahams and R. W. Grosse-Kunstleve, "Building Hybrid Systems with Boost.Python," http://www.boostpro.com/writing/bpl.html, March 2003, last accessed March 2010.

[87] D. Abrahams, "Boost.Python Project," http://www.boost.org/libs/python/, last accessed March 2010.

[88] D. M. Beazley and W. S. Fulton, "SWIG Project," http://www.swig.org/, last accessed March 2010.

[89] R. Yakovenko, "Py++ Project," http://www.language-binding.net/pyplusplus/pyplusplus.html, last accessed March 2010.

[90] S. Purcell, "PyUnit Project," http://pyunit.sourceforge.net/, last accessed March 2010.

[91] E. Loper, "Epydoc Project," http://epydoc.sourceforge.net/, last accessed March 2010.

[92] "OpenCV Project," http://opencv.willowgarage.com/, last accessed March 2010.

[93] P. S. P. Cowpertwait, V. Isham, and C. Onof, "Point process models of rainfall: Developments for fine-scale structure," in *Proceedings of the Royal Society of London*, ser. A, vol. 463, no. 2086, October 2007, pp. 2569–2587.

[94] P. S. P. Cowpertwait, "A spatial-temporal point process model of rainfall for the Thames catchment, UK," *Journal of Hydrology*, vol. 330, no. 3-4, pp. 586–595, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B6V6C-4K7WTYF-2/2/6f2985bee27f2a35911240e70baa9f84

[95] G. Varoquaux and P. Ramachandran, "Mayavi: Making 3D Data Visualization Reusable," in *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, Pasadena, CA, USA, August 2008, pp. 51–56.

[96] A. Olsen, "Python 3000 with Free Threading project," http://code.google.com/p/python-safethread/, last accessed March 2010.

[97] C. Tismer, "Continuations and Stackless Python," in *Proceedings of the 8th International Python Conference*, Arlington, VA, January 2000. [Online]. Available: http://www.stackless.com/spcpaper.htm

[98] K. Hinsen, "Parallel Scripting with Python," *Computing in Science and Engineering*, vol. 9, no. 6, pp. 82–89, November 2007.

[99] K. Hinsen, H. P. Langtangen, O. Skavhaug, and Å. Ødegård, "Using BSP and Python to Simplify Parallel Programming," *Future Generation Computer Systems*, vol. 22, no. 1-2, pp. 123–157, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-4BT1BGT-1/2/b748fc00d0e185cc2ac9172b2773f410

[100] F. Pérez and B. E. Granger, "IPython: a System for Interactive Scientific Computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, May-June 2007. [Online]. Available: http://ipython.scipy.org

[101] B. Eckel, "Concurrency with Python, Twisted, and Flex," http://www.artima.com/weblogs/viewpost.jsp?thread=230001, May 2008, last accessed March 2010.

[102] Python Web Services Project, "Web Site," http://pywebsvcs.sourceforge.net/, last accessed March 2010. [Online]. Available: http://pywebsvcs.sourceforge.net/

[103] J. Wagener, O. Spjuth, E. L. Willighagen, and J. E. S. Wikberg, "XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services," *BioMed Central Bioinformatics*, vol. 10, no. 1, p. 279, May 2009. [Online]. Available: http://www.biomedcentral.com/1471-2105/10/279

[104] J. K. Philipp, *Gnuplot in Action: Understanding Data with Graphs*. Manning Publications, 2009.

[105] "Gnuplot Homepage," http://www.gnuplot.info/.

[106] M. Haggerty, "Gnuplot.py Web Site," http://gnuplot-py.sourceforge.net/.

[107] J. Hunter, "matplotlib Project Web Site," http://matplotlib.sourceforge.net/.

[108] "SciPy's matplotlib Cookbook," http://www.scipy.org/Cookbook/Matplotlib/.

[109] R. Ihaka and R. Gentleman, "R: A language for data analysis and graphics," *Journal of Computational and Graphical Statistics*, vol. 5, pp. 299–314, 1996.

[110] "R Main Project Web Page," http://www.r-project.org/.

[111] "RPy Project Web Site," http://rpy.sourceforge.net/.

[112] "Chaco Project," http://code.enthought.com/projects/chaco/.

[113] T. Vaught and E. Jones, "Enthought Scientific Computing Solutions, Inc." http://www.enthought.com/.

[114] "GracePlot Web Site," http://graceplot.sourceforge.net/.

[115] P. Ramachandran and G. Varoquaux, "Mayavi Project," http://code.enthought.com/projects/mayavi/.

[116] B. Sherwood and D. Scherer, "Visual Python Project," http://vpython.org/.