HARVEY MUDD
COLLEGE

Mathematics Clinic

Final Report for
Hewlett-Packard Laboratories

Identifying and Minimizing Non-smoothness Issues in ICC Profiles

April 29, 2007

Team Members

James Egan
Herbie Huff
Alex Izsak
William Warriner
Nate Chenette (Project Manager)

Advisor

Prof. Weiqing Gu

Liaisons

Dr. Ingeborg Tastl
Gary Dispoto

Abstract

Color management, or the translation of color data between electronic devices, entails transforming color data through a profile connection space (PCS). An ICC profile for a device consists of a set of color look-up tables that are interpolated to transform color data from a device-dependent representation to the PCS, and vice versa. Non-smoothness of an ICC profile is defined as banding and discontinuities that the profile introduces when transforming a smooth image. Currently, there are no automatic ways to identify or reduce non-smoothness—all evaluations are performed visually, and all corrections are be performed manually. Our project has two objectives: first, to create a metric that can quantitatively measure the non-smoothness of ICC profiles, and second, to develop a method that automatically adjusts ICC profiles to reduce non-smoothness. In this report, we cover the color science background for this project, then explain in detail the motivation and goals of the project. We then discuss our approach to modeling the problem, and detail several strategies we used to implement a solution. We also explain the architecture of our computer code, which will be delivered to color scientists at HP labs. Finally, we give a detailed overview of the year's accomplishments, and suggest directions for possible extensions of our work.

Contents

Abstract **iii**

Acknowledgments **ix**

1 Introduction **1**
1.1 Problem Statement 1
1.2 Objectives 2
1.3 Overview of Report 2

2 Color Science Background **5**
2.1 Color Spaces 5
2.2 ICC Profiles 9
2.3 Rendering Intents and Gamut Mapping 11

3 Mathematical Modeling of the Problem **15**
3.1 The Importance of Modeling in CIELab Color Space 15
3.2 Modeling the ICC Profile as a Discrete 3D Transformation . 17
3.3 Specification of a Non-Smoothness Metric 18
3.4 Using Spline Interpolants to Correct Non-smoothness 23

4 Constructing Trivariate Tensor Product B-Splines **25**
4.1 Introduction to Tensor Product Splines 25
4.2 Knot Placement and End Conditions 27

5 Methods for Identifying and Reducing Non-Smoothness Issues **29**
5.1 Introduction 29
5.2 Determining Metric Thresholds 30
5.3 Fall Semester Method: Using Bivariate Splines to Correct Non-Smoothness 31

5.4 Three Overall Methods for Identifying and Correcting Non-Smoothness . . . 33
5.5 The RGB →Lab approach . . . 43
5.6 Potential issue with our new approach . . . 43

6 Implementations **45**
6.1 Fall Semester . . . 45
6.2 Spring Semester . . . 54
6.3 Using Our Methods . . . 61

7 Year Summary **65**
7.1 Fall Semester . . . 65
7.2 Spring Semester . . . 65

8 Conclusion and Future Work **69**
8.1 Conclusion . . . 69
8.2 Future Work . . . 70

A Glossary **71**
A.1 Color Science . . . 71
A.2 Splines . . . 73
A.3 MATLAB Spline Toolbox Commands . . . 74

B MATLAB Code **77**
B.1 Summary of Functions . . . 77
B.2 Summary of Important MATLAB Built-in Functions . . . 79
B.3 Code . . . 80

C HowTo **81**
C.1 How To Print to the HP Photosmart 7960 . . . 81
C.2 How To Use X-Rite . . . 87
C.3 How To Use GamutViewer . . . 89
C.4 How To Use an ICC Profile . . . 94

Bibliography **101**

List of Figures

2.1 Visualization of the CMYK color space, with labeled endpoints. 6
2.2 Color management flowchart, with CIELab as the PCS. . . . 9

3.1 Visualization of a transformation including the device independent to device dependent perceptual intent tag of a profile. 16
3.2 A collection of slices of a profile and their corresponding spline approxiations. 18
3.3 The dihedral angle at an edge e_i. 20
3.4 The absolute mean curvature at a vertex v. 21
3.5 The adjusted definition of Absolute Mean Curvature 21
3.6 The Gaussian curvature at a vertex v. 23
3.7 A one dimensional representation of a spline approximation. 24

4.1 A bivariate tensor-product spline surface, with univariate curves shown running along the surface. 26

5.1 A triangular mesh is constructed by connecting points in the manner shown above. 30
5.2 Graph of some slices. 32
5.3 Graph of the approximated surfaces. 33

6.1 Irregularly spaced points in CIELab, resulting from starting with RGB grid points. 46
6.2 Diagram of the process of projecting a mesh onto its corresponding spline surface fit. 48

6.3 A series of gradient images beginning in sRGB. The first image is the input slice represented as a gradient. The second is the first slice passed through the profile via the CIELab to sRGB perceptual intent transformation. The third is the same as the second, but with one R value altered in the perceptual intent transformation. 51
6.4 Maximum dihedral angle at each point in the $L^\star$ slice with $L = 81$. 52
6.5 Visualization of a transformation including the device independent to device dependent perceptual intent tag of a profile. 55
6.6 The slice with input L held constant at 56.25 in profile `newest.icc` with the center point pulled by 50 in the a direction. 57
6.7 The slice with input L held constant at 31.25 in profile `newest.icc` with a matrix modelling real-world non-smoothness applied to it. 59

Acknowledgments

Dr. Ingeborg Tastl – for her commitment to our project, visionary guidance, cheerful personality, and desire to help us learn.

Gary Dispoto – for his friendliness and helpfulness, and all the time he has spent with us in spite of his busy schedule.

The team at HP Labs – for their hospitality and support, and for offering us advice on our project.

Garret Heckel – for helping us with questions we had regarding X-Rite ColorPort and Photoshop.

Barbara Schade – for her valuable administrative assistance and efficient communication.

Prof. Weiqing Gu – for her life analogies, words of wisdom, unique sense of humor, unequaled intelligence, and for ensuring we not only do our best work, but also value and enjoy life.

Chapter 1

Introduction

Hewlett-Packard Laboratories (HP Labs) performs cutting-edge research for Hewlett-Packard, one of the world's largest information technology corporations. Color printing technology has long been at the heart of HP Labs' concerns. For the third year in a row, a Harvey Mudd Math Clinic team worked with Hewlett-Packard Labs to solve a problem in color printing technology. The 2004-2005 HP clinic dealt with printer drift, and the 2005-2006 HP Labs Clinic team generated International Color Consortium (ICC) profiles. This year, our team designed a process to identify and minimize non-smoothness issues in ICC profiles.

The International Color Consortium dictates industry-wide standards for management and communication of color information. These standards are necessary because various color devices, such as scanners, cameras, printers, and display systems, all interpret and reproduce colors differently. An ICC profile describes a transformation between such device-dependent data and a device-independent color space, the profile connection space (PCS). It maps color data by interpolating discrete lookup tables that are built from actual color measurements. In our project, the PCS is the device-independent color space CIELab, which is designed to correspond closely with human vision.

1.1 Problem Statement

Under certain conditions, the transformation specified by an ICC profile may not be smooth; that is, though the input data changes smoothly, the output data does not. Potential causes of this problem include limitations on the achievable color range or *gamut* of the devices involved, degrees

of freedom in constructing the look-up table, sub-optimal algorithms applied when generating transformations from measurement data, and measurement errors. Such non-smoothness issues manifest themselves as color discontinuities in an end-product such as a photo or display. The non-smoothness we see may be due to a single problem, or a combination of several issues.

There is no analytical method for fixing such problems. Color scientists fix non-smoothness manually by printing test sheets, inspecting them for discontinuities, and making changes to the original ICC profile. Different algorithms can be applied and or lookup tables can be modified, but every adjustment is based on technical expertise, experience, and trial and error. Since there are no commonly agreed upon metrics to mathematically evaluate the smoothness of an ICC profile, there is no way to red-flag a potentially problematic profile. So, scientists must visually evaluate every new ICC profile. This is a time-consuming process that requires a trained observer; but it is also essential, because the ultimate quality of the printed product depends upon the visual impression that it makes.

1.2 Objectives

Our challenge was to detect and quantify non-smoothness in color transformations, and to automatically correct problems in ICC profiles accordingly. In particular, our goals were:

- To come up with an efficient, accurate metric to quantitatively measure non-smoothness of a color transformation.
- To develop a method that, based on output data of a transformation, automatically adjusts the ICC profile lookup table to minimize non-smoothness.

1.3 Overview of Report

In the forthcoming chapters we report on our approach to this problem, our methods for solving it, and the knowledge we have attained in the process. Chapter 2 gives a brief overview of necessary concepts and terminology from color science. In Chapter 3, we present our mathematical model of the problem and detail some necessary mathematical concepts. We devote Chapter 4 to explaining in detail how we construct multi-dimensional

splines. In Chapter **??** we delineate three distinct methods to solve this problem. We describe our non-smoothness metric, and then our multiple strategies to automatically correct non-smoothness issues. Next, in Chapter 6, we describe our current plans for implementation of these methods, and give details about algorithms, data structures, and the architecture of our code. Finally, in Chapter 7, we give an overview of our year's accomplishments, and in Chapter 8 we discuss possible directions in which our work could be extended. Glossary, Code, and How-To documentation concerning software and hardware are relegated to the appendices.

Chapter 2

Color Science Background

2.1 Color Spaces

Colorimetry is the measurement of color. Perception of color depends on the observer, light source, and object. The goal of colorimetry is to depict color through a standardized environment that reduces the specification of color to *tristimulus* values. Optimally, two stimuli sharing the same tristimulus values are perceived as the same color by an average viewer. *Color spaces* are three-dimensional mathematical spaces in which coordinates correspond either to the tristimulus values or to device values, and points indicate individual colors. Examples of color spaces include RGB, used in scanners, displays, and cameras; and CMY, used in printers.

In CMY space, white (W) is the origin, or $(0, 0, 0)$ on the ordered triple (C,M,Y). Cyan (C) or (1,0,0), magenta (M) or (0,1,0), and yellow (Y) or (0,0,1) are the three basis vectors of the space. By combining the vectors linearly we can obtain other colors: for example, C+M is blue (B) or $(1, 1, 0)$, C+Y is green (G) or $(1, 0, 1)$, M+Y is red (R) or $(0, 1, 1)$. Finally, black (K or key) is the sum of the three colors C+M+Y, or $(1, 1, 1)$. Since K = C+M+Y, the color space CMYK that is used extensively in printers is actually just CMY. (Black ink is used in printers to conserve the other inks and minimize imperfect mixing.) A representation of CMY/CMYK can be seen in Figure **??**.

The RGB space is similar, with the origin now black rather than white and with axes corresponding to red, green, and blue primaries. There are various types of RGB space—for instance, sRGB (*screen* RGB), which is a standardized encoding for colors to be displayed on an CRT monitor under specific viewing conditions, and pRGB (*printer* RGB), which encodes colors for printing. In sRGB, the origin corresponds to a blank (black) screen,

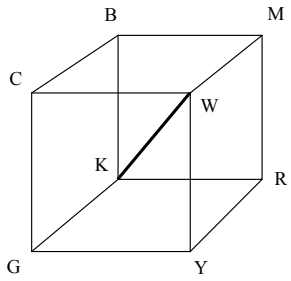


Figure 2.1: Visualization of the CMYK color space, with labeled endpoints.

just as the white origin in CMYK corresponds to the reference white of the printing medium, usually paper. As in CMY, we may obtain other colors by addition: here cyan is the sum of green and blue, yellow the sum of red and green, and magenta the sum of blue and red. The RGB colors are known as additive colors, because colors are obtained by combining light. The CMYK colors are known as subtractive colors, because colors are obtained by combining substances that reflect light.

Although additive and subtractive methods of specifying color are mathematically equivalent, this does not mean that they are equally adept at modeling physical color. Accurate color reproduction using subtractive means (such as ink) is much more difficult than reproduction using additive light sources. In monitors that use RGB, for example, the pixels emit light at an exact wavelength and intensity. Combining two pixels additively then produces exactly the expected third color. On the other hand, because inks are not perfect light absorbers, mixing two inks may fail to precisely produce the desired third color. Consequently, it is difficult to produce pleasing reds, blues and greens using cyan, magenta, and yellow inks as the primaries. It is also difficult to represent high levels of brightness, since inks passively reflect light incident upon them. In fact, since they absorb light, they actually reduce the brightness of the incident light.

The RGB and CMYK spaces work well to represent color for devices, since the coordinates axes are primaries the devices actually use to reproduce color. Most input devices, such as cameras and scanners, use the RGB space to represent information because the hardware records and combines three separate images, one for each of red, green and blue. Some output devices, like LCD and CRT monitors, emit red, green and blue light to display images. Other output devices, such as printers, use the CMY space, since

CMY coordinates directly correspond with amounts of cyan, magenta, yellow, and black inks used. However, in neither RGB nor CMYK do the axes correspond with the axes along which humans perceive color. That is, the RGB and CMYK spaces are *perceptually non-uniform*, meaning that distances in the color space do not necessarily correspond with color differences as perceived by a human viewer. In a *perceptually uniform* space, on the other hand, equal color distances in different regions of the color space do correspond with color differences determined by a human viewer.

To understand why RGB and CMYK are not perceptually uniform, one must understand the mechanism by which the human visual system processes stimuli. In the eye, two types of photoreceptors, rods and cones, detect incoming light. There are three types of cones, each most sensitive to a different frequency of light. One type is sensitive to light with frequencies in the red-orange range (S-type), another in the yellow-green range (M-type), and the final in the blue-violet range (L-type). After light hits the eye, neurons in the optic nerve encode the input from the cones into three signals. One signal, $(L + M + S)$, is the sum of inputs from all the cones, and denotes the overall cone responsivity; qualitatively, this corresponds to the overall lightness of the stimuli. The next signal,$(L - M + S)$, is the red-green signal, which describes where the signal is located on a red-green axis. Finally, $(L + M - S)$ is the yellow-blue signal. Because humans perceive colors along these axes, we never describe colors as a mixture of red and green, or of yellow and blue; however, we do describe colors as "yellowish-red" or "greenish-blue." (Fairchild [1998], 22-23)

A perceptually uniform color space must have axes arranged according to the three fundamental coordinates of human vision: the lightness coordinate L, the red-green coordinate a, and the yellow-blue coordinate b. Moreover, distances along the axes must be calibrated according to human visual sensitivity in different areas of the space, so that distance in the space corresponds very closely with perceived color differences. The CIELab (*Lab*) space meets these requirements. In addition, it is designed so that a Euclidean distance of 1 corresponds to a just noticeable difference between colors. Although this correspondence is not perfect, the CIELab space is still designated to be perceptually uniform because distances in the space very closely correspond with perceived color differences. Moreover, in a perceptually uniform space, three qualitative attributes of color—chroma, hue, and lightness—are arranged in a cylindrical coordinate system, and this is true is *Lab* . These qualitative terms are described in more detail in the glossary.

Aiming to reproduce color in terms of tristimulus values, color sci-

entists used RGB monochromatic primaries to create spectral power distributions of the wavelengths of visible light. That is, at a given wavelength, scientists empirically determined what mix of the three primaries best matched the given color. Unfortunately, some wavelengths of visible light were outside of the gamut of the three primaries. To match these stimuli, scientists first added some amount of one primary to the stimulus, and then matched the resulting stimulus as usual. This tactic is equivalent to adding a negative amount of the first primary. To deal with these negative values, scientists constructed theoretical monochromatic lights, which have everywhere-positive spectral power distributions. Effectively, these are imaginary primaries that require sources that can emit impossibly high amounts of light. These theoretical primaries are basis vectors of the CIEXYZ space (Fairchild [1998], 82), which was developed by the Commission Internationale de l'Éclairage in the 1920s.

Although CIEXYZ achieves the goal of specifying color as humans see it in terms of three coordinates, it is not a perceptually uniform space. It does not encode accurate information about color difference, nor does it account for non-linear visual responses and chromatic adaptation. Hence the development of the CIELAB space, which is a non-linear transformation of the XYZ tristimulus coordinates and the XYZ values of a reference white. The following set of equations define the CIELab space:

$$\begin{aligned} L^* &= 116 f\left(\frac{Y}{Y_N}\right) - 16 \\ a^* &= 500\left[f\left(\frac{X}{X_N}\right) - f\left(\frac{Y}{Y_N}\right)\right] \\ b^* &= 200\left[f\left(\frac{Y}{Y_N}\right) - f\left(\frac{Z}{Z_N}\right)\right] \\ f(x+) &= \begin{cases} x^{\frac{1}{3}}, & x > 0.008856 \\ 7.787x + \frac{16}{116}, & x \leq 0.008856 \end{cases} \end{aligned}$$

where X_N, Y_N, and Z_N are the tristimulus values of the reference white.Note that for sufficiently small values of x the function f is linear. The a transformation has more weight than the b transformation because experiments show that the human visual perception system gives a larger weight to blue and yellow than to red and green. It gives an even larger weight to luminance. Thus the values must be normalized to produce a perceptually uniform space. This space's perceptual uniformity is reinforced by the fact that

$$
\begin{aligned}
C^*_{ab} &= \sqrt{(a^*)^2 + (b^*)^2} \\
h^*_{ab} &= arctan\left(\frac{b^*}{a^*}\right) \\
\Delta E^*_{ab} &= \sqrt{(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2}
\end{aligned}
$$

where C^*_{ab} is the perceived chroma, constant on cylinders in CIELab space; h^*_{ab} is the perceived hue, constant on planes parallel with the luminance axis L^*; and ΔE^*_{ab} is the Euclidean distance between two sample colors in LAB, also called the color difference.

2.2 ICC Profiles

An ICC profile is a set of color lookup tables that are interpolated to transform color data from a device-dependent color space to a device-independent space, and vice versa. The device-independent space, called a *profile connection space*, is usually CIELab. For example, an ICC profile converts camera input data to CIELab, and then printer ICC profiles convert that *Lab* data to CMYK. The overall process is called *color management* and is depicted in Figure 2.2.

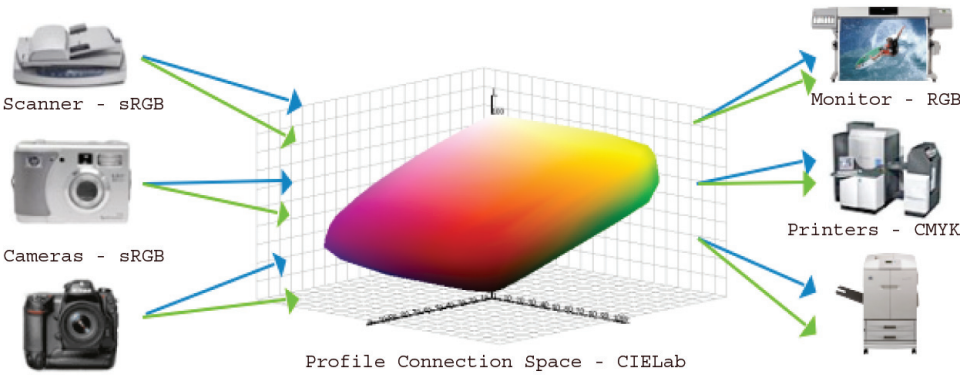


Figure 2.2: Color management flowchart, with CIELab as the PCS.

Each profile is highly sensitive to a large variety of factors. Obviously, each profile depends on what device is taking input or giving output; that

is, a printer requires a completely different profile from an LCD monitor. However, even two printer models produced by the same company using identical inks and media require two different profiles, because of mechanical inconsistencies between machines. Different profiles are also required for different print media, like glossy photo paper or plain paper, and for different kinds of inks. Because of the specificity they require and the speed with which new technology develops, new ICC profiles are constantly being generated. This means that it is especially important that ICC profiles can be efficiently created, tested, and refined.

2.2.1 Causes of Non-Smoothness

Gamut clipping occurs when the source gamut is larger than the destination gamut, and colors outside the gamut must be projected to within the boundary of the destination gamut. Gamut clipping can introduce loss of detail across gradients in images. Because gamut clipping cuts and shifts entire sections of the source gamut, certain regions of images lose vast amounts of detail. Consider an image of a forest taken from above; the image should be a vast array of greens. Some printers, however, are incapable of producing highly chromatic greens, especially in the shaded regions. Thus, if we were to use a gamut clipping algorithm to print this picture, we would likely clip a good number of those chromatic, darker greens to the boundary of the gamut. The nearest printable colors are likely to be nearby greens in the photo. The printed forest is now a smear of green across the page, with hardly any discernable shading between trees. The way in which the gamut is clipped depends upon the rendering intent of the profile; our method must respect this intent while correcting non-smoothness issues that arise.

One issue of non-smoothness is called banding. Banding occurs along a gradient in which bands of the wrong color are apparent. For example, a gradient from black to white in the background of a photo may contain bands of a variety of grays, rather than a smooth transition from black to white. This effect is most obvious when a gradient moves along the luminance axis in the CIELab space. The cause of this effect is from the interpolation of lookup table data. Remember that in order to transition from the source to the output through the PCS, we will need to use a CLUT at some point, and that this table is sparse. Interpolating the data on these points is linear, and if the underlying transformation from the PCS to our output space is non-linear, we will necessarily make a smooth curve non-smooth. In fact, wherever the slope of our interpolation of a gradient is greater than

the slope of the desired gradient itself, the color will be incorrect and we will have obvious banding.

Yet another issue to consider is measurement errors introduced in the creation of the CLUT. When creating the lookup table, a large number of measurements are taken with great care. Sometimes, however, erroneous measurements can be taken and introduced into the lookup table. Sometimes, errors are due simply to error of variation in the hardware itself. When this occurs, a gradient passing through the region around the erroneous data point may be non-smooth. The reason for this is that the point itself is non-smooth in relation to the points around it. To a human observer looking at a gradient passing near the error, it should be fairly obvious if the error is large enough. Graphically, it would appear as a bump or spike in a gradient passing near or through it. Once identified however, these issues can be readily corrected. In fact, these sorts of errors can be reduced by simply repeating measurements over and over again, and averaging the results, to account for random error.

The current process of testing whether or not an ICC profiles is smooth is highly subjective and rather time-consuming. Most techniques rely on the judgment of color scientists, who study images and take measurement data to ensure that a profile is smooth. Furthermore, when measurements show that there are non-smoothness issues, there are as yet no automatic ways to correct them. Adjustments must be made by hand, and then new measurement data must be produced. This process is repeated until a profile appears smooth.

2.3 Rendering Intents and Gamut Mapping

The *gamut* of a color reproduction system is the range of achievable colors that system can achieve. The system includes all elements involved in producing the stimulus perceived by your eye - lighting, ink type, media type, etc.

Each color transformation in an ICC profile must handle the problems that arise when the achievable gamuts of the devices involved are unequal—that is, when one device can achieve colors that the other cannot, or vice versa. For example, the gamuts of RGB devices include more vivid reds, greens, and blues than those attainable in CMYK devices, and so an RGB $\rightarrow$ CMYK profile must adjust its map to account for the colors that fall outside of the CMYK device's gamut. The way a transformation handles gamut issues is called its *rendering intent*. The ICC defines four different rendering

intents: absolute colorimetric, media relative colorimetric, perceptual, and saturation. Since the absolute colorimetric and media relative colorimetric intents differ only in their treatment of the white point, we will cover them as a unit.

- The **perceptual intent** aims to preserve the overall quality of an image. It is most often used for output device profiles, particularly in RGB → CMYK conversions for color printers, and also color space profiles and scanner profiles. The intent compresses and expands parts of the gamut of the image so that it fits within the gamut of the output device. Even though this technically means that the transformation inaccurately reproduces *all* the colors in the image, a viewer will not notice because all the colors have been adjusted proportionally.

- The **saturation intent** projects out-of-gamut colors into the gamut, then readjusts in-gamut colors toward the outside edge of the color space. This increases the saturation of the image. It is most effective for reproduction of cartoons, comics, or business graphics. The goal is to reproduce, say, yellow not in an accurate way, but with pure yellow ink.

- The **colorimetric intents**, unlike the saturation intent and the perceptual intent, aim first and foremost to accurately reproduce as many colors as possible. They map point-to-point within the gamut by simply converting the coordinates mathematically, then compress out-of-gamut colors without making corresponding adjustments to in-gamut colors. This differential treatment of colors can have noticeable effects, so the colorimetric intents work best when mapping gamuts of similar sizes. Mapping between widely differing gamuts using a colorimetric intent can introduce artifacts into an image, especially if the image spans a large section of the gamut. Some common uses for the colorimetric intents are fine art reproduction and image proofing.

Once a transformation with a given rendering intent has been created, evaluation methods for testing the profile then depend upon the rendering intent. For example, a profile that uses colorimetric intent will be evaluated primarly upon its ability to accurately reproduce individual colors. Colors used for testing the profile should be different from the colors used to generate the profile. When colors from outside the device gamut are tested, the

profile's accuracy will depend greatly upon what method it uses to map those colors into the new gamut.

A transformation that uses the perceptual intent, on the other hand, should be evaluated visually, since the perceptual intent's primary purpose is a pleasing, natural, and consistent reproduction of the image. To do this, color scientists print outrepresentative images and test charts and look for inconsistencies or unpleasing effects.

In our project, we dealt with the perceptual intent and the relative colorimetric intent. We solved non-smoothness issues generated by the perceptual intent of an ICC profile, and we used the profile's relative colorimetric intent as a mathematical transformation to translate between color spaces, as described in more detail in Chapter 5.

Our solution is current with ICC v4 (version 4) standards. The ICC is continually updating industry conventions for profiles. In ICC v2 profiles, the inverse of the perceptual tag was not a required component; the assumption was that only one direction of the transformation was necessary for most purposes. In ICC v4 profiles, however, the perceptual intent's inverse must be included in the set of look-up tables. So our solution must correct both perceptual CLUTs in the v4 profile.

2.3.1 Profile Tags

The color look-up tables in a given profile are known as that profile's *tags*. For reference purposes, here we explain the names of the tags that we work with, and the nature of their data structures.

The tags specify the images of regularly spaced points in a cube in the input space. The spacing of the cube depends upon the profile's dimension, which we refer to in short-hand and computer code as dim. Standard dimensions for profiles are 9, 17, and 33. So in the case of a profile with dimension 17, which is the most common, the spacing along the L-axis, which ranges from 0 to 100, is

```
[0, 6.25, 12.5, 18.75, 25, 31.25, 37.5, 43.75, 50, 56.25, 62.5, 68.75,
75, 81.25, 87.5, 93.75, 100]
```

and the spacing along the a and b axes, which range from -128 to 127, is

```
[-128, -112, -96, -80, -64, -48, -32, -16, 0, 16, 32, 48, 64, 80, 96,
112, 127].
```

In sRGB and pRGB, the cube is regularly spaced between 0 and 1 on each axis.

In tag nomenclature, the letter "A" always refers to the device-dependent space, and the letter "B" refers to the profile connection space. So for our

profiles, "A" refers to sRGB or pRGB, and "B" refers to *Lab* . The number at the end of the tag marks the intent: "0" indicates perceptual intents, and "1" indicates relative colorimetric intents.

- **BToA0:** This is the perceptual intent transformation. As described above, the color lookup table is a dim $\times$ dim $\times$ dim $\times 3$ array. The first three coordinates specify the index of a point in a regular cube in *Lab* space, and the final dimension contains the image of that point in RGB. So, for example, when dim $= 17$, the (8,8,8,3) coordinate in the LUT is the B coordinate of the *Lab* point $(43.75, -16, -16)$.

- **AToB0:** This is the inverse of the perceptual intent, which is needed for proofing purposes. It is also a dim $\times$ dim $\times$ dim $\times 3$ array, specifying the output in *Lab* of points in a regular cube in RGB.

- **AToB1:** This is the relative colorimetric intent transformation from RGB to *Lab* . It is produced by measuring the *Lab* values of colors produced by the device and with known values in that device's RGB space. Depending on the profile, it may consist of a few matrices which are then interpolated linearly, or it may contain a full look-up table.

- **BToA1:** This is the inverse of the AToB1 tag. It is less reliable than the AToB1 tag, because it is not constructed using measurements. Rather, it is designed to invert whatever is encoded in the AToB1 tag. This inversion is not always perfect, since its construction involves interpolation as well as assumptions about the behavior of the transformation in between known measured points.

Chapter 3

Mathematical Modeling of the Problem

3.1 The Importance of Modeling in CIELab Color Space

We modeled a color transformation as a vector-valued transformation $\vec{F}$: $Lab \rightarrow Lab$. There are two reasons we considered $Lab \rightarrow Lab$ transformations. First, as described in the previous section, Lab is a perceptually uniform space, and distances in Lab correspond with perceived color differences. In seeking a metric to quantify non-smoothness, we are seeking to measure a perceptual phenomenon having to do with color difference. This measurement is possible in Lab .

Secondly, we worked in Lab because that makes our solution broadly applicable. Our methods work on both perceptual transformations of an ICC profile: both the transformation from the PCS to device-dependent space, the BtoA1 tag, and the device-dependent to PCS transformation used for proofing purposes, the AtoB1 tag. Furthermore, our model is even applicable to color transformations encoded in frameworks that are different from the ICC framework. Because we approach the problem in this way, our solution applies to almost any device that reproduces color: not only printers, but also scanners, cameras, display screens, and other devices.

We obtain the $Lab \rightarrow Lab$ transformation from CLUTs in the profile. Because we need to interface with the structures required by the CLUT, the MATLAB spline generation toolbox, and our discrete curvature measurements, this process can be somewhat complicated, as we discuss in detail in chapter 5. For the purposes of describing our mathematical model, though, we discuss the process in a simplified manner.

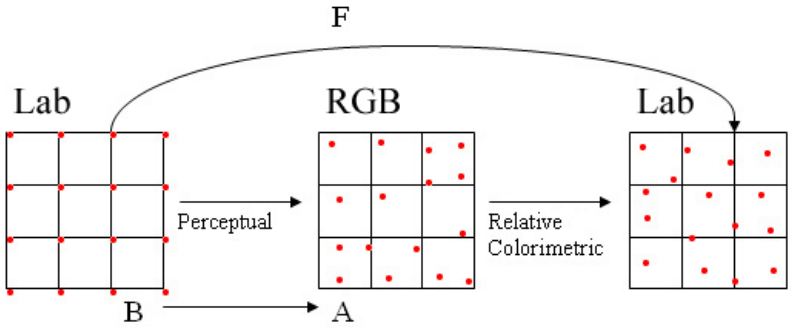


Figure 3.1: Visualization of a transformation including the device independent to device dependent perceptual intent tag of a profile.

If we are dealing with a transformation from an input space to CIELab, such as the sRGB $\rightarrow$ *Lab* profile discussed in Chapter 6, we proceed as follows:

1. Begin with a table of grid points in *Lab* .
2. Use the relative colorimetric intent to transform grid points to sRGB.
3. Apply the profile's perceptual transformation to these sRGB points.

This results in a discrete *Lab* $\rightarrow$ *Lab* transformation. For reasons that will be explained in Chapter 5, it is necessary that we begin with grid points in *Lab* .

If we are dealing with a transformation from *Lab* to an output space, such as CMY, we proceed as follows:

1. Begin with the input side of the CLUT, the *Lab* grid points.
2. Apply the profile to transform to CMY.
3. Use the relative colorimetric intent to translate back to *Lab* .

We obtain a discrete *Lab* $\rightarrow$ *Lab* transformation, as desired. This transformation is shown in Figure 3.1, with the composition of the perceptual and relative colorimetric intent transformations creating our desired F.

3.2 Modeling the ICC Profile as a Discrete 3D Transformation

For each profile, we now have a three dimensional lookup table that associates grid points in CIELab with output points also in CIELab. Denote the ith input point as (L_i, a_i, b_i), and denote the corresponding output point $(\hat{L}_i, \hat{a}_i, \hat{b}_i)$. Let us denote the overall discrete transformation as follows.

$$\vec{F}(L, a, b) = (F_{\hat{L}}(L, a, b), F_{\hat{a}}(L, a, b), F_{\hat{b}}(L, a, b)) = (\hat{L}, \hat{a}, \hat{b}),$$

where $\hat{L}$,$\hat{a}$, and $\hat{b}$ denote the output coordinates, marked with a hat for clarity. $F_{\hat{L}}$, $F_{\hat{a}}$, and $F_{\hat{b}}$ are component functions of $\vec{F}$.

Our goal is for the entire transformation $F : \mathbb{R}^3 \to \mathbb{R}^3$ to be smooth.

In order to facilitate visualization of our data, which is six-dimensional, we consider the component functions $F_{\hat{L}}$, $F_{\hat{a}}$, and $F_{\hat{b}}$, and we further divide the component functions into slices. Next we describe rigorously how these elements are related in terms of their smoothness.

In general, for a transformation

$$\vec{F} : \mathbb{R}^3 \to \mathbb{R}^3$$

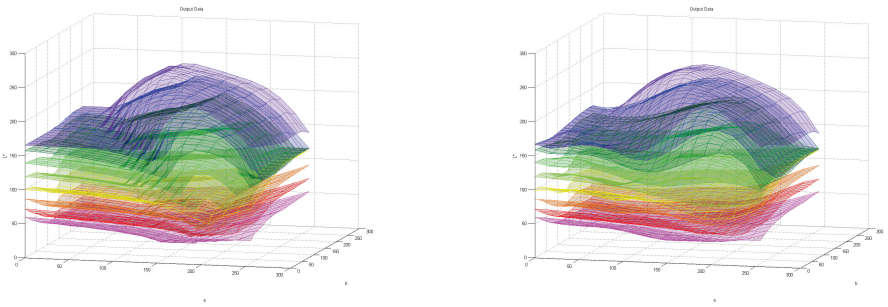
that maps

$$(L, a, b) \mapsto (F_{\hat{L}}(L, a, b), F_{\hat{a}}(L, a, b), F_{\hat{b}}(L, a, b)),$$

mathematically speaking, $\vec{F}$ is smooth **if and only if** each component function $F_{\hat{L}}$, $F_{\hat{a}}$, and $F_{\hat{b}}$ is smooth. Without loss of generality, if $F_{\hat{L}}$ is smooth, then each *slice* $F_{\hat{L}}(L_0, a, b)$, $F_{\hat{L}}(L, a_0, b)$, and $F_{\hat{L}}(L, a, b_0)$ is also smooth for discrete values of L_0, a_0, and b_0 in the domain of $\vec{F}$. This statement holds for $F_{\hat{a}}$ and $F_{\hat{b}}$ as well. However, the reverse direction of this statement is not necessarily true; that is, it is not necessarily the case that if a finite number of the slices of a component function are smooth, then the component function itself is smooth.

From a practical point of view, however, we assume that if we take enough slices and smooth them all to within a visual tolerance, that this will also smooth the component function to within a visual tolerance, and thus smooth the entire transformation. This is assumption is reasonable, since in the end our approach must correspond to human visual response, but we will subject it to testing.

Our approach is to look at these slices of data. In this way we reduce a six-dimensional data set to a collection of three-dimensional data sets. We hold one input constant, and consider the other two input variables as the



(a) A collection of slices of a profile, with $L^\star$ held constant at different values.

(b) The collection of spline approximations of each slice.

Figure 3.2: A collection of slices of a profile and their corresponding spline approxiations.

x and y axis in a three-dimensional plot. On the z-axis, we then plot one of the output variables. Graphing this produces a surface in three dimensions, on which non-smoothness issues manifest themselves as creases, bumps, or folds. Figure 3.2(a) shows a series of these slices. The advantage of the slice method is that we can visualize the transformation in this way, allowing us to intuitively and qualitatively evaluate non-smoothness for that slice. Then, when we smooth the data using the method of splines, which we describe later in this section, we can visualize the adjustments we are making, as well as the constraints we are imposing upon the spline. Figure 3.2(b) shows spline approximations corresponding to each of the previous slices. At the end of this chapter we discuss the spline method we used to correct non-smoothness.

3.3 Specification of a Non-Smoothness Metric

In quantifying a transformation's overall non-smoothness, we sought a metric that would

(1) red-flag a transformation with a non-smoothness issue, and

(2) identify the location and extent of the problems.

Our metric considers each slice of each component of the ICC profile transformation. The graph of each of these slices can be viewed as a mathematical *mesh*, and the problem of smoothing the slice is known in mathematics as *fairing* the mesh. To obtain a mathematical mesh from the 3D points in the slice, we must triangulate edges between the vertices. There are several ways to triangulate a mesh; we discuss our specific method for triangulating the mesh in Chapter 5. At each point in the mesh, our metric evaluates several measures of **discrete curvature**, including **dihedral angles**, **absolute mean curvature** at a vertex, and **Gaussian curvature** at a vertex.

3.3.1 Dihedral Angles

One method of measuring non-smoothness in a mesh is by looking at the dihedral angles of the mesh. A dihedral angle is defined on an edge in the following way:

- Let e_i, e_{i-1} and e_{i+1} be edges in the mesh
- Let t_i and t_{i+1} be the triangles with e_i as a common edge
- The normal of triangle t_i is given by

$$n_i = \frac{e_i \times e_{i-1}}{||e_i \times e_{i-1}||}.$$

- Similarly,

$$n_{i+1} = \frac{e_{i+1} \times e_i}{||e_{i+1} \times e_i||}.$$

Definition The **dihedral angle** at an edge e_i

$$\beta_i = \arccos \frac{n_i \cdot n_{i+1}}{||n_i||\ ||n_{i+1}||}.$$

The dihedral angle at an edge e_i is a measure of the smoothness of data at that edge. If β_i is small, then e_i is smooth with respect to the surrounding datapoints. A large dihedral angle implies a more sudden change in the data at e_i.

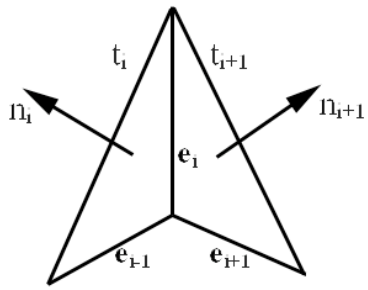


Figure 3.3: The dihedral angle at an edge e_i.

3.3.2 Absolute Mean Curvature: Standard Definition

However, looking at the dihedral angles alone is not sufficient to capture all causes of non-smoothness in a mesh. Consider a point at the top of a spike, such as the one shown in 3.4. Here, it may be possible that each of the dihedral angles at an edge e_i is small enough that it would not indicate non-smoothness in normal circumstances. However, the relatively large-looking jump in the data at vertex v may be causing a non-smoothness in the overall transformation. To identify issues such as this, we calculate the absolute mean curvature at each vertex in the mesh.

Definition The **absolute mean curvature** at a vertex v is

$$AMC\,(v) = \frac{1}{4}\sum_{i=1}^{n} ||e_i||\beta_i$$

Using this definition, even if the dihedral angles are relatively small, the length of the edges e_i results in a large absolute mean curvature.

3.3.3 New Definition of Absolute Mean Curvature

For our model, though, we need to adjust the standard definition of absolute mean curvature. This is because the spacings along the $x-$ and $y-$directions of the mesh are always regular. Because of this, when using the standard definition, point on a smooth mesh that rises steeply but uniformly have high absolute mean curvature values because the edge lengths

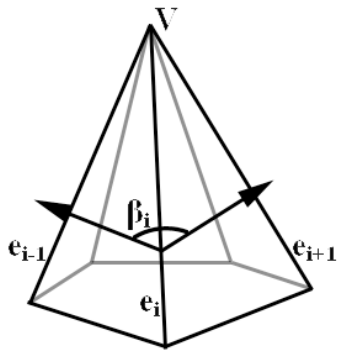


Figure 3.4: The absolute mean curvature at a vertex v.

along the mesh are long. Thus the standard definition would incorrectly identify these points as non-smooth.

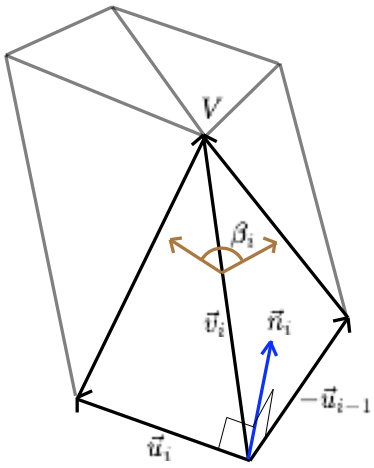


Figure 3.5: The adjusted definition of Absolute Mean Curvature

In this new definition, recommended by Prof. Gu, we take into account the "base vectors" u_i and u_{i-1} which connect the points surrounding the vertex v in question, as shown in figure 3.5. Their cross product is normalized to create $\vec{n}_i$. Then, the new edge length is defined to be the length of

the *projection* of the vector $\vec{v}_i$ onto this normalized vector $\vec{n}_i$. Formally,

$$\vec{n}_i = \frac{-\vec{u}_{i-1} \times \vec{u}_i}{|-\vec{u}_{i-1} \times \vec{u}_i|}$$

$$e_i = |proj_{\vec{n}_i}\vec{v}_i| = \left|\frac{\vec{v}_i \cdot \vec{n}_i}{|\vec{n}_i|^2}\vec{n}_i\right| = \vec{v}_i \cdot \vec{n}_i$$

where $|\vec{x}|$ denotes the Euclidean norm (length) of $\vec{x}$, defined by $|(x, y, z)| = \sqrt{x^2 + y^2 + z^2}$.

The absolute mean curvature is then computed using these new edge lengths e_i and the dihedral angles β_i according to the standard formula:

$$AMC(v) = \sum_i e_i \cdot \beta_i$$

3.3.4 Gaussian Curvature

A third measure of non-smoothness is the Gaussian curvature at a vertex.

Definition The **Gaussian curvature** at a vertex v is

$$\kappa(v) = 2\pi - \sum_{i=1}^{n} \alpha_i,$$

For vertices that are smooth with respect to the surrounding points, the area is approximately planar and the Gaussian curvature is close to 0. If the position of vertex v is a sudden change with respect to the surrounding data, then the angles are more acute and the Gaussian curvature will be closer to 2π. If the vertex implies a sudden change and the connecting edges are irregular (in a tutu pattern, for example), the Gaussian curvature is negative. Gaussian curvature looks at non-smoothness in a mesh on a different scale, and thus can help identify unique areas of non-smoothness in a mesh. Looking at the Gaussian curvature also gives us a better sense of how the mesh is changing over a larger area, giving information about trends in the data.

It is important to note that these metrics are obtained slice by slice. In order to evaluate the entire transformation, we compute these metrics for every slice in the transformation. This means we consider the fact that every point

$$(L_i, a_i, b_i) \mapsto (\hat{L}_i, \hat{a}_i, \hat{b}_i)$$

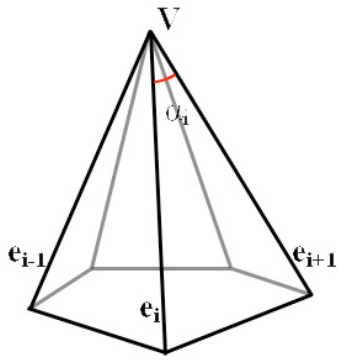


Figure 3.6: The Gaussian curvature at a vertex v.

in the CLUT is a member of nine meshes. The $L_i^\star$ coordinate, for example is a member of three meshes, the $(a, b, \hat{L})$ mesh with fixed L_i, the $(L, b, \hat{L})$ mesh with fixed a, and the $(L, a, \hat{L})$ mesh with fixed b. Similarly, $a_i^\star$ and $b_i^\star$ are also involved in three meshes each.

3.3.5 Spline Difference

A final way to detect non-smoothness is to look at how much the data deviates from the spline interpolant. As explained below, the spline fit captures overall trends in the data. So points that are a large distance from the spline are points that cause non-smoothness problems. This method only works if the spline is constructed so that it does not fit the data too tightly; again, we experimented with corresponding visual tests to determine the most effective spline interpolant for this method.

3.4 Using Spline Interpolants to Correct Non-smoothness

Our method for correcting non-smoothness also considers each output variable individually. We use *spline* interpolants to capture the overall trends in each mesh.

Definition A *spline* is a smooth, continuous approximant that is a linear combination of piecewise polynomial basis functions, usually of degree 3 or less.

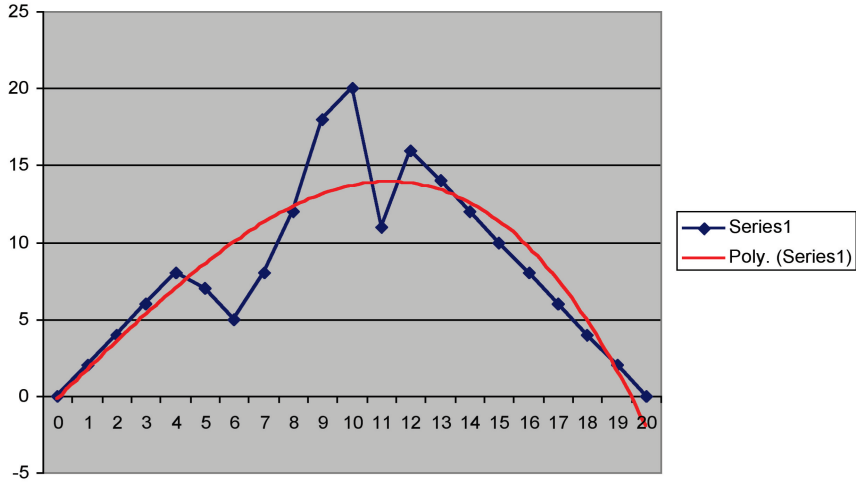


Figure 3.7: A one dimensional representation of a spline approximation.

The coefficients on the basis functions are chosen to best fit the data in a least squares sense. Splines can be curves, surfaces, or interpolants of higher dimensions. Splines are smooth in the sense that all directional derivatives match at the boundaries between the pieces of the interpolant. One has a number of degrees of freedom when constructing a spline: the position of the **knots**, which are the places where the polynomial pieces join, end conditions concerning the spline's derivatives, requirements that second derivatives also match at the knots, etc.

Splines are easy to construct in Matlab, and they are easy to adjust by changing the basis functions. We use piecewise cubic splines, because they respect local trends in the data, and they avoid the higher interpolation error possible when using polynomials of higher degree. A one-dimensional example of this is shown in figure 3.7.

Chapter 4

Constructing Trivariate Tensor Product B-Splines

4.1 Introduction to Tensor Product Splines

Tensor product splines interpolate data in several dimensions by first constructing sets of interpolating curves in each dimension and then taking the tensor product over all of the curves. Mathematically, this follows from the equation

$$f(x,y,z) = \sum_{u=1}^{U} \sum_{v=1}^{V} \sum_{w=1}^{W} B_{u,k}(x) B_{v,l}(y) B_{w,m}(z) a_{u,v,w}$$

where $f(x,y,z)$ is the trivariate spline, $B_{u,k}(x)$, $B_{u,k}(y)$, and $B_{u,k}(z)$ are the univariate curve sets, and $a_{u,v,w}$ are the minimizing coefficients. Each curve is constructed in the usual manner, by minimizing a distance function, usually the squared distance, and by taking into account a choice of knots and end conditions.

Theoretically, one could construct a single spline interpolant for the entire CLUT. This interpolant can be imagined as a vector field that approximates the vector field produced by plotting the output coordinates in the CLUT as vectors located at the grid input points in the CLUT. However, constructing a 3D $\rightarrow$ 3D spline is currently out of reach in MATLAB.

Originally, we were under the impression that we only had the capacity to produce 2D $\rightarrow$ 1D splines, known as bivariate splines or spline surfaces. So our original plan was to construct a spline surface for each mesh, and adjust points toward that mesh. However, since each output variable is a member of three sets of meshes, as mentioned above, this method would

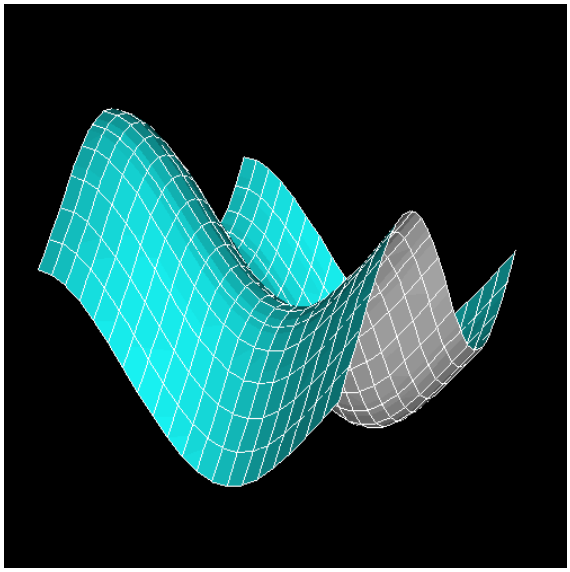


Figure 4.1: A bivariate tensor-product spline surface, with univariate curves shown running along the surface.

have required us to reconcile the adjustments recommended by each spline surface. The splines in figure 3.2(b) are bivariate splines, produced during an early implementation of our method.

However, during the spring semester we discovered that MATLAB does have the capacity to produce 3D $\rightarrow$ 1D splines, or trivariate splines. These interpolants can be imagined as scalar fields that interpolate the scalar field produced by plotting one of the output variable as a scalar value located at the corresponding grid input points of the CLUT. Using trivariate splines solves the reconciliation issue and requires fewer interpolants. So to correct non-smoothness issues, we adjusted points toward a tri-variate spline interpolant. For each color transformation, we constructed three spline interpolants, one for each output variable. In terms of the mathematical transformation $\vec{F}$ detailed earlier in this chapter, we have one interpolant for each of $F_{\hat{L}}$, $F_{\hat{a}}$, and $F_{\hat{b}}$.

Because such splines interpolate three dimensions of input data, and because they are constructed using the tensor product, they are known as *trivariate tensor product splines*. We experimented to find the proper knot configurations and end conditions for our splines; see chapter 6 for details.

In the case of the $\hat{L}$ coordinate, for example, such a spline is a fucntion S :Lab$\rightarrow$ $\hat{L}$. It is constructed by taking the tensor product of three sets of single-variate splines, one set for each input variable. Because the tensor-product spline is constructed in this way, the input data of our transformation must be on perpendicular axes; MATLAB cannot calculate the tensor product spline if the single-variate spline curves are not perpendicular to one another. The data sites need not be entirely regularly spaced in a grid, but on each axis they must be collinear at every point. We refer to these splines as B-splines because they are constructed by assigning coefficients to a set of polynomial basis functions.

4.2 Knot Placement and End Conditions

In order to calculate the trivariate tensor product spline of a regularly spaced scalar field such as the one induced by considering a single output coordinate plotted against three input coordinates, MATLAB first calculate the spline curves. These cubic spline curves are determined by the following choices.

- **knot placement** The *knots* determine the location of the ends of the piecewise polynomial basis functions. The knots are subject to the Schoenberg-Whitney conditions, which stipulate that the number of

knots must equal the number of data sites plus the order of the spline. In this condition, the knots also must not have multiplicity greater than one (one knot per location) and must lie strictly within the interval of the data sites (can not share the same value as the end data sites). Since we use cubic splines and in each dimension we have dim data points, our knot sets contain dim + 4 knots.

- **choice of end conditions** As described in the glossary, there are three types of *end conditions*: open end condition, closed end condition, or floating end condition. The end conditions determine how closely the spline will fit the data points at the ends of the curve.

- **fitting function** The coefficients on the polynomial basis functions are chosen to minimize a certain function that relates the spline to the data. Usually, this simply means that we find a least-squares fit to the data points. However, other fitting functions may be used. For example, one might want to weight some points so that the spline will fit them more closely.

The spline toolbox in MATLAB contains several functions that generate splines. Some require a knot set, some require specification of end conditions, some allow the user to specify a fitting function. After some experimentation, we have found the `spap2` function produces the smoothest fit to our color data. `spap2` uses a least-squares fit to all data points, generates automatic knot placement based on the data location, and produces a piecewise cubic spline (order 4).

Because knot placement in three dimensions is not necessarily intuitive, the advantage of this function is that it generates knots automatically.

Chapter 5

Methods for Identifying and Reducing Non-Smoothness Issues

5.1 Introduction

As mentioned in Chapter **??**, our methods for identifying and reducing non-smoothness rely on modeling a profile transformation as a mapping from *Lab* to *Lab* space. The way in which we achieve this mapping depends on whether we are modeling the AtoB0 tag or the BtoA0 tag of a profile, and in the case of the AtoB0 tag, we have two seperate methods to create the mapping.

After creating the *Lab* to *Lab* mapping, we can use it to measure non-smoothness via discrete curvature measurements on certain input and output components at a time, as described in Chapter **??**. We have come up with two ways to use this metric: the *thresholds test* and the *worst points test*.

5.2 Creating the $Lab \rightarrow Lab$ Mapping

5.2.1 BtoA0 case

5.2.2 AtoB0 case - Interpolation method

5.2.3 AtoB0 case - Profile method

5.3 Non-smoothness Metric Using Discrete Curvature

5.3.1 Measuring *Lab* output versus *Lab* input

5.3.2 Thresholds test

5.3.3 Worst Points test

5.4 Non-smoothness Correction Using Splines

5.5 Smoothing the output *Lab* data

5.6 Correction of the BtoA0 tag

5.7 Correction of the AtoB0 tag

In this chapter, we outline our methods for identifying and reducing non-smoothness. We have two methods for the BtoA0 tag and three methods for the AtoB0 tag. We need the extra method for the AtoB0 tag because it poses particular difficulties due to the fact that it is indexed by RGB. These methods differ according to the way they construct an Lab→Lab transformation, and according to how they detect non-smoothness.

Evaluating our discrete curvature measures requires we connect points in the tranformation with a mesh. There are many different ways to connect the vertices with edges. One might think that since we originally use a square CIELab data grid to align our data, we will want to use a square mesh for the slices. But, in measuring the non-smoothness of a mesh, we will be taking into account the normals of the faces of the mesh. In looking at a slice of data, it cannot be certain that any four points will define a plane. If we connect the datapoints as a square mesh, taking the normal of a face in two different ways may result in two different normals. It is important to have consistency when taking our measurements, so we will connect the data points as a triangular mesh.

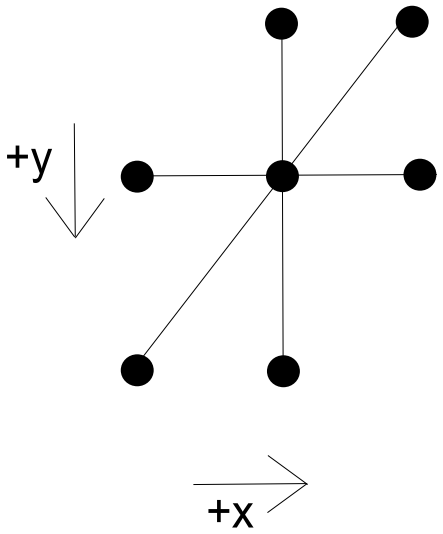


Figure 5.1: A triangular mesh is constructed by connecting points in the manner shown above.

We use the triangulation that is standard at HP, shown in figure 5.1.

As shown, each point is connected to six other points. Thus we calculate six normal vectors, six edge lengths, and $\binom{6}{2}$ dihedral angles. Using this information, we can calculate the absolute mean curvature and Gaussian curvature for the point.

5.8 Determining Metric Thresholds

All of our methods for identifying non-smoothness depend upon an established correspondence between the values of the discrete curvature measures and human perception of non-smoothness problems. We determine this relationship in two ways:

1. **Introduce artificial non-smoothness.** Starting with a profile that is known to be smooth, we change the values of output entries in the CLUT so that it becomes non-smooth. Then we look at images of those slices and evaluate the visual non-smoothness. Since we control the amount by which we change the CLUT, we can gradually reduce it until the non-smoothness we introduce is just barely visible. The discrete curvature values corresponding to such a change then correspond to a threshold level of non-smoothness.

2. **Experiment with thresholds.** Starting with a non-smooth profile, we search through all the slices and mark all points above some discrete curvature threshold, which we initially guess. Then we display images of the slices where above-threshold points are marked with a black dot. We adjust our threshold until only points corresponding to visual non-smoothness are marked.

For the first strategy, we introduced artificial non-smoothness in several patterns. Most simply, we altered a single point. We also introduced random error, oscillating patterns of error, and error patterns modeled from real profiles.

Upon examining the data these two strategies produced, we decided that we would allow color scientists using our non-smoothness identification program to examine the metric output alongside slice images from the profile, rather than having an absolute threshold.

5.9 Fall Semester Method: Using Bivariate Splines to Correct Non-Smoothness

Recall that in our transformation, each of the output values $L^\star$, $a^\star$, or $b^\star$ can be thought of as a function of the three input values. In order to construct our meshes, we first pick one of the input values to hold constant at a certain value. Then, we also pick an output value to graph as a function of the two remaining variable inputs. We can obtain a number of different slices by choosing a different output value, choosing a different input to hold constant, or by holding an input constant at a different value. In total, there are $3 \times 3 \times$ dim meshes to consider.

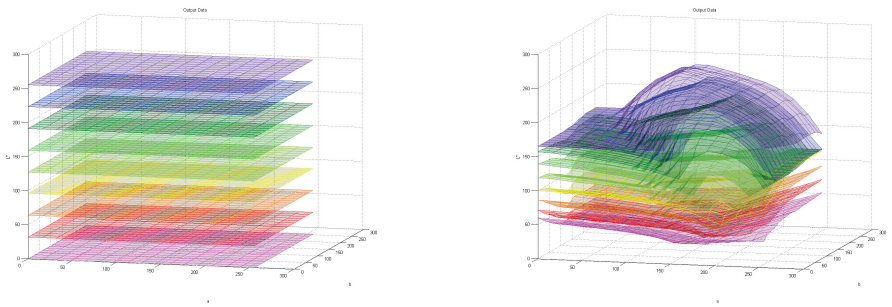
As discussed in chapter **??**, we then evaluate measures of discrete curvature on the triangulated mesh. These are the dihedral angles, absolute mean curvature, and Gaussian curvature.

The method of spline approximation offers us several advantages in correcting non-smoothness. When correcting non-smoothness, the visual impression of the transformation is more important than mathematical validity. Therefore, even after correcting non-smoothness to within a tolerance of the metric, we want our data to follow the same overall trend. The least-squares approximation generates a curve that does not necessarily go through all the data points. This offers us an advantage in terms of computing time. A spline approximation is generally of lower order than an exact interpolation. Also, as we saw in chapter 3, an exact interpolation some-

times gives us trends in the data where there are in fact none. In many cases, a spline approximation gives us a better approximation to the trend than does an exact interpolation. By using spline approximation, we now have a curve fit to the data that better matches the overall trend of the data and is of lower order.

The spline doubles as an indicator of non-smoothness. Because not all points fall on the approximated curve, it is possible to measure for each data point how far away it is from the corresponding approximated point on the spline. We then come up with a tolerance for visual perception of non-smoothness based on that numeric distance.

Another advantage to using the method of spline approximation is that we can minimize the number of errors created by our corrections. When correcting non-smoothness, we will move points on a mesh to within a tolerance of the metric. If we were to simply move the points without taking into consideration the surrounding points, we may be introducing additional issues of non-smoothness. Because splines are defined by the general trend of the data, moving a single point will slightly move the surrounding points to match the general trend. In this way, we hope to be able to preserve the pre-existing smoothness in the mesh while correcting the errors identified by our metric.

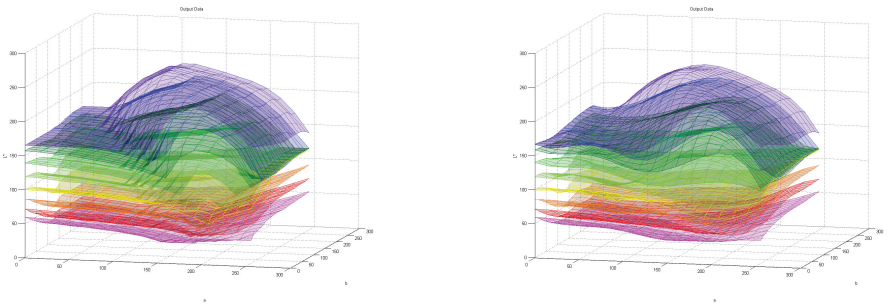


(a) Slices of L

(b) Graph of $\hat{L}$ vs. a and b for each slice

Figure 5.2: Graph of some slices.

Here is an example of the methods by which we correct non-smoothness. We take slices by holding L constant at a variety of values. The next graph includes the functions we attain by graphing $\hat{L}$ against a and b at each of the constant values of L. Finally, we see a graph of the spline approximation



(a) Graph of $\hat{L}$ vs. a and b for each slice

(b) Spline approximations for each slice

Figure 5.3: Graph of the approximated surfaces.

for the slices.

With these approximated surfaces, we will know in what direction to move the data points should we find a non-smooth area with our metric. A more detailed description of our fall semester implementation of splines can be found in chapter 6.

5.10 Three Overall Methods for Identifying and Correcting Non-Smoothness

As we tested our metric and began implementing our solutions, our initial plan developed into three methods. Method 1 and Method 2 differ only in their treatment of the AtoB0 tag; each uses a different procedure to obtain an Lab→Lab transformation. Method 3 differs from 1 and 2 in that it does not consider the discrete curvature metric. Rather, it considers the distance from the point to the spline interpolant.

Ultimately, the package we created allows a color scientist to choose which Method to employ.

5.10.1 Two Methods for Dealing with the AtoB0 Tag

The structure of the AtoB0 tag poses some difficulties to our methods. As mentioned in chapter 2, each entry in the AtoB0 tag is the Lab image of a

grid RGB point. Originally, we planned to obtain an Lab→Lab transformation from the AtoB0 tag by the following procedure:

1. Begin with a regular grid in Lab.
2. Use the BtoA1 tag to transform those points to RGB. These points are not on a regular grid in RGB, but the AtoB0 tag can still transform them using the interpolation functionality embedded in the tag.
3. Transform the RGB points to Lab using the AtoB0 tag.

However, there are several problems with this process. After applying the BtoA1 tag to the Lab grid, all the Lab points outside the RGB gamut get collapsed to the border of the RGB space. Because there is no way to differentiate between out-of-gamut points and points on the border of the gamut, since both get mapped to the border of RGB, all points on the border of RGB must be disregarded. Because we are disregarding so many points, we have far sparser data inside the RGB gamut, and so we have fewere points on which to eventually run our metrics.

Another problem with this process is that it employs the BtoA1 tag. Recall from chapter 2 that the BtoA1 tag is not always a reliable transformation. If the BtoA1 tag is not smooth, this process will require us to construct a new BtoA1 tag from the inverse of the AtoB1 tag.

A final drawback to this procedure is that once we correct the Lab output points using the spline, we do not know how to change the CLUT. This is because the Lab output in this process comes from non-grid RGB points, and in order to modify the CLUT we need to know the images of grid RGB points. We need to transform the grid RGB points to Lab$_i n$, and the evaluate the spline at those points.

Because this process has so many drawbacks, we came up with two alternatives. The first is described below in Method 1. The second alternative is to examine an RGB →Lab transformation, described at the end of this chapter.

5.10.2 Method 1: Use Interpolation to obtain an Lab→Lab Transformation

1. Start with regular gridpoints in RGB space:

$$RGB_{reg}$$

2. Send those RGB gridpoints to input CIELab through the AtoB1 tag:

$$LAB_{in} = \text{AtoB1}(RGB_{reg})$$

So each point $\mathbf{y} \in LAB_{in}$ is equal to AtoB1($\mathbf{x}$) for some $\mathbf{x} \in RGB_{reg}$.

3. Interpolate each of the CIELab gridpoints using the LAB_{in} points. That is, each CIELab gridpoint will have an associated tetrahedron of points in LAB_{in} along with the appropriate barycentric coordinates. That is, for every point $\mathbf{u} \in LAB_{reg}$, we have an interpolation mapping

$$f \;:\; \mathbf{u} \;\mapsto\; \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \mathbf{y}_3 & \mathbf{y}_4 \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix}$$

where each $\mathbf{y}_i \in LAB_{in}$ is a point of a tetrahedron and each b_i is the associated barycentric coordinate for that point. This interpolation actually involves some nuanced issues; see chapter 6 for details. We construct the tetrahedrons in RGB_{reg} so that they are regular, but then use the images of these points Lab_{out} to interpolate the images of the grid points in LAB_{reg}. Some Lab grid points will be out-of-gamut; those points will be recognized as not being inside a tetrahedral. So that we can still construct a spline, set these points equal to the nearest in-gamut point, and mark them as out-of-gamut for later reference.

4. We can rewrite the function above as

$$f \;:\; \mathbf{u} \;\mapsto\; \begin{bmatrix} \text{AtoB1}(\mathbf{x}_1) & \text{AtoB1}(\mathbf{x}_2) & \text{AtoB1}(\mathbf{x}_3) & \text{AtoB1}(\mathbf{x}_4) \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix}$$

Let us define a new function based on these values for $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$:

$$g \;:\; \mathbf{u} \;\mapsto\; \begin{bmatrix} \text{AtoB0}(\mathbf{x}_1) & \text{AtoB0}(\mathbf{x}_2) & \text{AtoB0}(\mathbf{x}_3) & \text{AtoB0}(\mathbf{x}_4) \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix}$$

5. Now, we have a mapping from each gridpoint in LAB_{reg} to a point inside a tetrahedron in CIELab space that has been through the AtoB0 tag. We can use the interpolation (collapse the four points down to one using the barycentric coordinates) to find that single point. Let's define this as a new function, m_1:

$$\begin{aligned} m_1 \;:\; \mathbf{u} \;&\mapsto\; f^{-1}\left(\begin{bmatrix} \text{AtoB0}(\mathbf{x}_1) & \text{AtoB0}(\mathbf{x}_2) & \text{AtoB0}(\mathbf{x}_3) & \text{AtoB0}(\mathbf{x}_4) \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix}\right) \\ &=\; \mathbf{v} \in LAB_{out} \end{aligned}$$

6. This function, m_1 :Lab$_{reg}$ $\rightarrow$Lab$_{out}$, is the Lab$\rightarrow$Lab transformation that we use in our metric, and for constructing the spline.

7. **Metric - Worst points:** Our metric will find the points in LAB_{reg} with worst discrete curvature on certain CIELab slices. At this point, we can simply report the CIELab values of these worst points, and print out the problem CIELab slice onto the screen through the image transformation
$$LAB \overset{\text{BtoA1}}{\longrightarrow} RGB \overset{\text{AtoB0}}{\longrightarrow} LAB \overset{\text{lab2srgb}}{\longrightarrow} sRGB.$$
This will allow the color scientist to determine whether or not the profile tag has non-smoothness problems.

8. **Metric - Thresholds:** The color scientist also has the option to input discrete curvature thresholds. Our metric will find the points in LAB_{reg} with over-threshold discrete curvature on certain CIELab slices, and we can visualize these points and slices via the method above for worst points. However, here, we also want to report the problems in RGB space, so that we can associate non-smoothness problems with points in the AtoB0 CLUT. To do this, we take neighborhoods of each over-threshold point in Lab$_{reg}$, and if AtoB1($\mathbf{x}$) lies within that threshold, we flag the point $\mathbf{x} \in RGB_{reg}$.

9. **Spline Correction Method:** The above steps will give us a spline that is a smooth version of the transformation m_1. To correct the AtoB0 tag: transform all the RGB_{reg} values to LAB_{in} using the AtoB1 tag, and then evaluate the spline on each of those Lab points. This will give us a set of data points that comprises a "smooth CLUT".

10. We then iteratively adjust the profile toward the spline, and allow the color scientist to view output images at each step of the iterative process. The amount that we adjust at each iteration depends upon the color scientist's input concerning the extent of the non-smoothness problems: if there is a big problem, we adjust toward the spline more drastically, and if there is a small problem, we make small adjustments with each step. The color scientist then decides when the profile is acceptably smooth, and stops the iteration.

Remark: Given the discrete curvature values for all the meshes in the profile, it may be useful to red-flag those points that are causing non-smoothness. We want to red-flag RGB points in the AtoB0 CLUT, but our discrete curvatures measures are associated with Lab input points. In order to red-flag RGB points when using Method 1, do the following:

- Identify which Lab points are having trouble by identifying all points with discrete curvature measures over a certain threshold.
- For each Lab point identified above, find the surrounding non-grid Lab points that are the images of RGB grid points under the AtoB1 tag. These are the points AtoB1($\mathbf{x}$) that were calculated in step 2 of Method 1.
- Raise a red-flag at each of the RGB points $\mathbf{x}$.

Raising a red-flag in the BtoA0 tag, on the other hand, is straightforward because the CLUT entries are the same Lab points being examined by the metric.

5.10.3 Method 2: Use Profile Tags to obtain an $Lab \to Lab$ Transformation

1. Start with regular gridpoints in CIELab space:

$$LAB_{reg}$$

2. Use the BtoA1 tag to get corresponding RGB points:

$$RGB_{irr} = \text{BtoA1}(LAB_{reg})$$

3. Use the AtoB0 tag to get the image of these points in CIELab space:

$$LAB_{out} = \text{AtoB0}(RGB_{irr}) = \text{AtoB0}(\text{BtoA1}(LAB_{reg}))$$

   Define this as a function from a point $\mathbf{u} \in LAB_{reg}$ to a point $\mathbf{v} \in LAB_{out}$:

$$\begin{aligned} m_2 \quad : \quad \mathbf{u} \quad &\mapsto \quad \text{AtoB0}(\text{BtoA1}(\mathbf{u})) \\ &= \quad \mathbf{v} \in LAB_{out} \end{aligned}$$

4. This function, $m_2 : LAB_{reg} \to LAB_{out}$, is the CIELab $\to$ CIELab transformation that we use in our metric, and for constructing the spline. The final steps of Method 2 are then identical to the final steps of Method 1:

5. **Metric - Worst points:** Our metric will find the points in LAB_{reg} with worst discrete curvature on certain CIELab slices. At this point, we can simply report the CIELab values of these worst points, and print

out the problem CIELab slice onto the screen through the image transformation

$$LAB \overset{\text{BtoA1}}{\longrightarrow} RGB \overset{\text{AtoB0}}{\longrightarrow} LAB \overset{\text{lab2srgb}}{\longrightarrow} sRGB.$$

This will allow the color scientist to determine whether or not the profile tag has non-smoothness problems.

6. **Metric - Thresholds:** The color scientist also has the option to input discrete curvature thresholds. Our metric will find the points in LAB_{reg} with over-threshold discrete curvature on certain CIELab slices, and we can visualize these points and slices via the method above for worst points. However, here, we also want to report the problems in RGB space, so that we can associate non-smoothness problems with points in the AtoB0 CLUT. To do this, we take neighborhoods of each over-threshold point in LAB_{reg}, and if AtoB1($\mathbf{x}$) lies within that threshold, we flag the point $\mathbf{x} \in RGB_{reg}$.

7. **Spline Correction Method:** The above steps will give us a spline that is a smooth version of the transformation m_1. To correct the AtoB0 tag: transform all the RGB_{reg} values to LAB_{in} using the AtoB1 tag, and then evaluate the spline on each of those Lab points. This will give us a set of data points that we can think of as a "smooth CLUT".

8. We then iteratively adjust the profile toward the spline, and allow the color scientist to view output images at each step of the iterative process. The amount that we adjust at each iteration depends upon the color scientist's input concerning the extent of the non-smoothness problems: if there is a big problem, we adjust toward the spline more drastically, and if there is a small problem, we make small adjustments with each step. The color scientist then decides when the profile is acceptably smooth, and stops the iteration.

Remark: Again, given the discrete curvature values for all the meshes in the profile, it may be useful to red-flag those points that are causing non-smoothness. We want to red-flag RGB points in the AtoB0 CLUT, but our discrete curvatures measures are associated with Lab input points. In order to red-flag RGB points when using Method 2, do the following:

- Identify which Lab points are having trouble by identifying all points with discrete curvature measures over a certain threshold.

- For each Lab point identified above, use the BtoA1 tag to map it to RGB. This point will not be on the RGB grid.

- Find the RGB grid points closest to it (code written by last year's HP Labs team can do this).
- Raise a red-flag at each of the RGB grid points.

5.10.4 Using Splines for Both Identification and Correction

Since many of the difficulties associated with fixing the AtoB0 tag stem from the conflicting structure requirements of the splines, the CLUT, and the discrete curvature measurements, another way to solve the problem is to actually use splines to measure non-smoothness. This method can also be used for the BtoA0 tag.

Given a set of data and the three spline fits for each coordinate of the output data, we simply subtract one from the other to obtain the difference between the spline and the data. Points that are a large distance from the spline are points that deviate from the overall trends in the transformation, which means that they are points that are probably non-smooth.

The method proceeds as follows:

1. First follow either the first six steps of Method 1 or the first four steps of Method 2 to obtain an Lab→Lab transformation.
2. Construct a spline interpolant for each output variable.
3. For each output variable, calculate the distance from each point to the spline - that is, simply find the absolute value of the difference between the spline value and the value of the output variable in question.
4. We then iteratively adjust the profile toward the spline, and allow the color scientist to view output images at each step of the iterative process. The amount that we adjust at each iteration depends upon the color scientist's input concerning the extent of the non-smoothness problems: if there is a big problem, we adjust toward the spline more drastically, and if there is a small problem, we make small adjustments with each step. The color scientist then decides when the profile is acceptably smooth, and stops the iteration.

Remark: Given the spline differences at all points in the profile, it may be useful to red-flag those points that are causing non-smoothness. In the case of the AtoB0 tag, we want to red-flag RGB points in the CLUT, but our discrete curvatures measures are associated with Lab input points. In order to red-flag RGB points when using Method 3, do the following:

- Identify which Lab points are having trouble by identifying all points with spline differences larger than a certain threshold.
- If the process from Method 2 was used to construct an Lab→Lab transformation, then for each Lab point identified above, use the BtoA1 tag to map it to RGB. This point will not be on the RGB grid. If the process from Method 1 was used, find the surrounding non-grid Lab points that are the images of RGB grid points under the AtoB1 tag. These are the points AtoB1($\mathbf{x}$) that were calculated in step 2 of Method 1.
- In either case, find the RGB grid points closest to the point with a problem.
- Raise a red-flag at each of the RGB grid points.

5.10.5 Method for the BtoA0 Tag

Since the BtoA0 tag does not have the complications discussed above, the methods for identifying and correcting non-smoothness problems in the BtoA0 tag are more straightforward. They are as follows.

1. Beginning with a dim by dim by dim grid in Lab, transform the points to RGB using the BtoA0 tag, and then transform those points back to Lab using the AtoB1 tag. This is the Lab→Lab transformation we will consider.
2. By holding either L, a, or b constant, take slices of the transformation and evaluate the discrete curvature on these meshes.
3. Construct a spline interpolant for each of the three output variables. Evaluate the distance from the spline for each output.
4. Using spline difference and discrete curvature, the color scientist can then determine how much, if any, non-smoothness is in the color transformation. We then correct this non-smoothness using the iterative method of splines.
5. Once spline correction has produced new Lab_{out} values, convert these to RGB using the BtoA1 tag. Substitute these new RGB values back into the CLUT.

Remark: In some cases when the BtoA1 tag is not reliable, it may be necessary to use interpolation to convert the Lab_{out} values back to RGB, or to construct a new BtoA1 tag by interpolation from the AtoB1 tag.

5.10.6 Alternative Method: Examine an RGB →Lab Transformation

Another method for dealing with the AtoB0 tag that we developed is to examine an RGB →Lab transformation. In this section, we describe the motivation for this approach, how it works, and some of the benefits and drawbacks.

5.10.7 Indexing inconsistent with CLUT

This is not a problem with the metric part of our project, but it arises when we try to coordinate our metric with corrections to the CLUT to create a smooth profile. We want to change an entry in the CLUT only if it exceeds our non- smoothness thresholds as measured by our metric. The AtoB0 tag's CLUT contains Lab data arranged according to an implicit regular RGB grid. But with our old approach, we have regular Lab data rather than regular RGB data. So to identify the problem points in the CLUT would demand some sort of interpolation (which could be difficult because Lab and RGB gamuts are dissimilar in shape and size.)

5.10.8 Difficulty detecting out-of-gamut points

An Lab→Lab approach requires that we locate and label out-of-gamut points in the RGB space (that is, points from the input Lab grid that are gamut-clipped during the BtoA1 transformation.) This is necessary, because such points no longer correspond to regularly spaced data as they go through the AtoB0 transformation, which is the transformation we are interested in. The discrete curvature of these points are later disregarded by our metric.

In the Lab→Lab methods, we locate out-of-gamut points by looking at their RGB values after the BtoA1 transformation. Any point with one of R, G, or B equal to the extreme values of 0 or 1 is labeled out-of-gamut. However, there are also valid (in-gamut) points—usually on the boundary of the gamut—with R/G/B = 0 or 1, and we will lose these points by this method. Some trials we have run have resulted in strange arrangements of apparently "out-of-gamut" points—for instance, two areas of in-gamut

points within a slice that are separated by a barrier of our tagged-as-out-of-gamut points. This seems to indicate that this method of detecting out-of-gamut points is not totally accurate.

5.10.9 Unpredictable number of test points

The BtoA1 transformation on its own typically reduces the number of test points by a factor of about 4 through gamut clipping. The problem of having a reduced number of test points has an easy solution—we simply start with a denser regular input grid in Lab. However, this brings up the question: how much denser should the input grid be? We would want the number of in-gamut points in RGB to be about the same as the number of test points in the regular grid we use in testing the BtoA0 tag (where all points in our regular grid are in-gamut). But there is no easy way to compute the size of regular Lab grid that will bring us closest to this value, and there is surely no way to achieve this value exactly (or even closely?)

Essentially, this means that we will have an unpredictable number of test points, and they will have a variable density in the input grid. This means it's very hard to compare tests and thus establish thresholds. Different grid densities lead to widely different ranges of discrete curvature measures. A certain measure at a high density might correspond to visual non-smoothness while the same measure at a lower density might look smooth. And, since it's more likely that an over-threshold point will be found in a test grid that has more points, we may need to scale thresholds based on the number of points tested. It's possible that these issues are minor, but to make sure of this would take a lot of empirical testing. And if it turns out to be a big problem, appropriate solutions (scaling factors or other adjustments based on number of test points) would be hard to determine, and may not even exist.

5.10.10 Issues with arrangement of test points

Note that the BtoA1 transformation causes our test slice to take an irregular arrangement in terms of the in-gamut points. That is, one row of regular input Lab data no longer has a fixed number of test points. Since the discrete measures at a point take into account the neighboring points (to calculate edge lengths, faces, angles), the measures at every point next to an out-of-gamut point (call it a "border point") don't mean much. The irregular shape of the test slice compounds this problem, because the irregular shape creates more "border points."

We could treat the border points in the irregularly-shaped arrangement as we do in the regular box formation (that is, only run our measures on the neighbor points that are in-gamut.) However, the number of in-gamut neighbors is unpredictable and variable, which complicates the process. And in any case, theory and our empirical observations indicate that discrete curvature measurements on the points on the border of the test grid (even when the grid is regular) are less accurate than those on the inside of the grid.

5.10.11 BtoA1 transformation may not be smooth

One of the assumptions of this approach is that the BtoA1 transformation—bringing input Lab points into the space (pRGB or sRGB) that is the input for our tag to test (AtoB0)—is smooth. That is, any non-smoothness we see in the output data is almost all due to issues with the AtoB0 transformation rather than the BtoA1 transformation. We made this assumption because we know that relative colorimetric transformations are typically quite smooth, but we can't be totally sure this is always the case.

5.11 The RGB →Lab approach

The new idea is simply to start with a regular RGB grid instead, remove the first transformation from our test setup, and then measure Lab vs. input RGB data for our metric.

New transformation:

$$\text{Reg grid in pRGB/sRGB} \xrightarrow{\text{AtoB0}} \text{Output Lab}$$

This solves all of the problems mentioned above.

5.12 Potential issue with our new approach

The main issue this approach may introduce is that the perceptual non-uniformity of the RGB space may make our metric results inaccurate with respect to visual evaluation. The reason we originally decided on a Lab to Lab map was that Lab is a perceptually uniform space, meaning that distances in the space correspond well to human perception, and hence measurements in the space using our metric have perceptual meaning.

The perceptual uniformity of pRGB is determined by the ink *separation*, or correspondence between pRGB values and amounts of ink used by the device. This means that the perceptual uniformity of pRGB varies from printer to printer. For this reason, we would have different thresholds depending upon the printer and the linearity of the separation. This is a major drawback to this approach.

Chapter 6

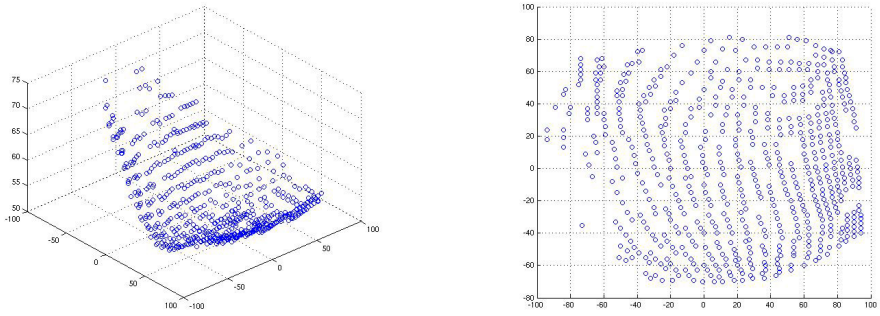
Implementations

6.1 Fall Semester

During the fall, we began to implement our methods in MATLAB. We wrote functions to take slices, calculate discrete curvatures, display a visual representation of the mesh, and construct spline interpolants. At that time we were still using bivariate spline approximations.

6.1.1 First Attempt at a Model

Modeling our problem requires that we examine an overall transformation from CIELab space to itself, as discussed in Chapter 3. To achieve this, our liaison sent us data beginning as a regularly spaced grid in sRGB space, along with two sets of transformed data in CIELab space. The first set of transformed data passes through the relative colorimetric intent transformation from sRGB to CIELab, and the second set of data passes through the perceptual intent transformation of the `sRGBv4_HP_FUJI_Prototype_June_06_b.icc` profile from sRGB to CIELab. Plotting the first set of transformed data, the domain of F, against the second set, the range of F, poses some interesting problems. By the nature of the transformations and the fact that the original data forms a regular grid in sRGB, neither of the transformed sets of data are regularly spaced in CIELab. As a result, creating a useful surface plot of the data using the `mesh` function is impossible. Using the `scatter3` function in Matlab allows for visualization of where a surface might lie for a given hyperplane, or slice, but there still exists the problem of how to generate a spline surface using this data.



(a) $Lab \overset{\text{RC}}{\leftarrow} sRGB \overset{\text{P}}{\rightarrow}$

(b) A projection of our first data on to the $a^\star b^\star$-plane.

Figure 6.1: Irregularly spaced points in CIELab, resulting from starting with RGB grid points.

Figure 6.1 contains a visualization of the data as described above. As seen in Figure 6.1(a), creating a regular mesh of these data is impossible. Because of the irregularity of data in the xy-plane, generating a spline surface becomes extremely difficult. In particular, generating a cubic spline on either the x- or y-axis where there exists only a single data point is impossible. Figure 6.1(b) shows that there are not nearly enough data points on any given x or y constant plane to form a reasonable approximation. Solving this problem requires data regularly spaced on the xy-plane; in other words, the first set of data must be regularly spaced. The most intuitive way to produce this requires that our pre-transformed data begin in CIELab instead of sRGB.

We begin with a set of regularly spaced CIELab data and pass it through the CIELab to sRGB relative colorimetric intent transformation, passing the result through the sRGB to CIELab perceptual intent transformation. This is identical to the set of transformations described in the previous attempt because the relative colorimetric intent transformation is invertible — in fact the CIELab to sRGB relative colorimetric transformation is its inverse — and the second transformation has not changed. However, this set of transformations provides the advantage of beginning with a regularly spaced grid of CIELab data as the domain of F. The transformation we want to evaluate is the perceptual transformation from CIELab to sRGB.

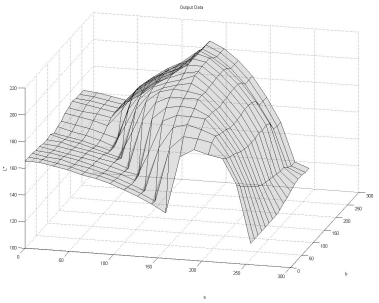
Because we would like the transformation to go from CIELab to itself, we can first apply the BtoA0 tag of the profile, which takes CIELab data to sRGB data using the perceptual intent, followed by an application of the AtoB1 tag, which takes sRGB data back to CIELab using the relative colorimetric intent.

With a regularly spaced domain of F, any of the slices described in Chapters 3 and 4 will necessarily have regularly spaced data in the xy-plane. Thus, it becomes possible to generate a spline surface using the tensor product of splines in both the x and y directions. These splines approximate the existing points in a least-squares sense, and use a simple knot set with knots of multiplicity four at each end and three knots placed between the end knots. This requires the function `spap2` performed on two sets of data, in the x and y directions, and two sets of knots, one for each direction; order is assumed to be the same for each direction, though both values must be passed to the function. The naive approach to solving the problem of non-smoothness issues using spline surfaces involves simply projecting every point on every hyperplane to its corresponding spline surface. Figure 6.2 shows the process by which the points representing a hyperplane of F is projected to a spline approximating that hyperplane in a least-squares sense via the naive approach. These images were generated using the function `first_test`, which incorporates the functions `AtoB0Tag` and `array2matrix` which turns an n^2 by 1 array into an n by n matrix, as well as the built in functions `spap2`, `fnval` and `mesh`. Unfortunately, issues arise using this simple method which must be considered.

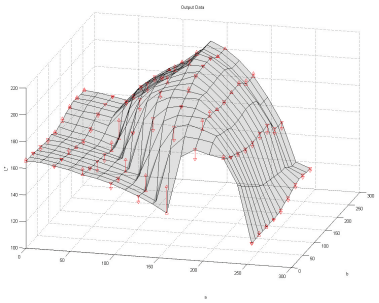
6.1.2 Resolving Issues In the Initial Model

As noted in the previous section, finitely many smooth hyperplanes of F does not imply smoothness of F. This means that in the strictest sense smoothing F with our method will not be perfect. However, to get around this, it should be assumed that smoothing all of the hyperplanes individually will smooth F to an acceptable degree. This assumption is safe because it is highly improbable any implementation of a solution will be unacceptably non-smooth. After making this assumption, there are still a number of questions arising from the initial implementation which, when answered, will refine it.

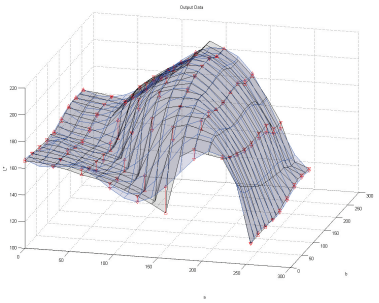
In the initial implementation, the spline surfaces were generated using least-squares approximating splines of order four with naive knot placement. While this method creates a spline which appears smooth, it may be the case that another approximation method will produce a more desirable



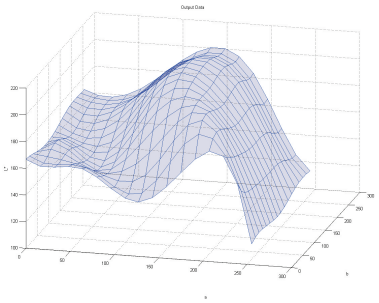
(a) A mesh representing a hyperplane, $L^\star$ output, $L^\star = 81$



(b) Projecting the data to a least-squares approximating spline surface.



(c) The spline surface after projecting.



(d) The final result.

Figure 6.2: Diagram of the process of projecting a mesh onto its corresponding spline surface fit.

result. For now the assumption is that a least-squares approximation will provide useable results, though other approximations need to be tested. Methods for improving knot placement will also be tested, including having one knot at each data point, with end knots having multiplicity equal to the order of the spline. Placing knots at each data point makes sense because then the control mesh of the spline surface will lie directly over the mesh representing the hyperplane, and it will be easier to control what happens to an individual point on the spline surface corresponding to an existing data point; simply adjust the relevant control point.

6.1.3 Implementing and Adapting a Metric

Ultimately, non-smoothness as measured by any metric must correspond with non-smoothness as measured by a human observer. In other words, if visible non-smoothness occurs in a profile, the metric must show this, and if there is no such occurence, then the metric must not show non-smoothness exists. Thus, the data returned by the metric measuring discrete curvature should correspond to human perception. In a mathematical sense, the metric might return a set of data for each point, but we must still connect this data with human perception. Thus, thresholds need to be set up for each of the metric measurements. Doing this only requires that we use an existing ICC profile, and artificially create the desired non-smoothness errors, then establish a correlation between the metric measurements, which are produced through a variety of discrete cuvature measurements, and the visibility of the artificial non-smoothness.

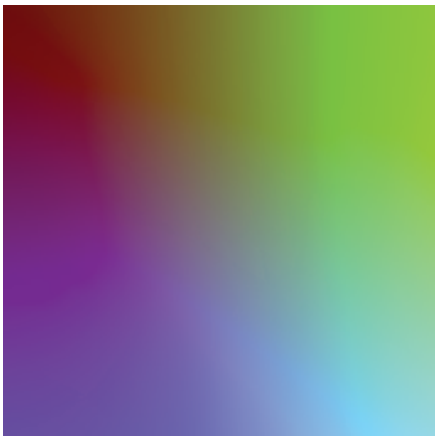
To introduce errors and measure them, first a basis for comparison is required, and then an alteration of the profile's perceptual tags are required. In the first attempt, only the BtoA0 tag (CIELab to sRGB) of the profile will be altered. This produces a gradient image representing a plane of the RGB cube with one of the three values — red, green or blue — held constant, as shown in Figure 6.3(a). This plane is sent first through the relative colorimetric intent transformation to CIELab and then the original BtoA0 tag (CIELab to sRGB) is applied; the image produced is a reference, or basis for comparison, as shown in Figure 6.3(b). After altering the perceptual intent transformation of the original profile by introducing an error, say increasing the R channel of one point in the LUT by 10, a new gradient image is produced displaying the error. Performing this alteration requires loading the profile into Matlab using the `iccread` function, and then altering data contained in `profile.BtoA0.CLUT`. The data in the profile is written as unsigned, 16-bit integers. In this case, the error is obvious when observed by

a human; it appears as a blue patch in the center of the gradient image, as seen in Figure 6.3(c). From this it is possible to reduce the error until it is no longer visible to a human observer.

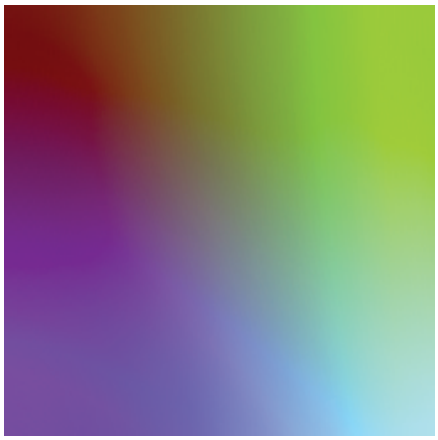
In the actual implementation of this idea, we want to look at modifications in $L^\star, a^\star$and$b^\star$. To do this, we will use gradient images with constant $L^\star$, for example. Instead of generating gradient images in sRGB, we will generate them in CIELab. The exact process for this will be discussed in chapter 5.4. The purpose of this is to establish thresholds for visibility of non-smoothness errors in profile LUTs. It is related to the measured color differences between individual colors, but our specific interest lies with the effect a particular color difference has when included into a LUT, which is used for interpolation. Effectively, we want to know whether certain color differences in parts of the color space create visible non-smoothness.

For points where we are going to make corrections, the thresholds can be used as follows. Projecting to a spline now only requires we get within the threshold of the point's corresponding value on the spline, rather than exactly on the spline, allowing the original purpose of the profile to remain relatively intact. This process can be extended to the $a^\star$ and $b^\star$ axes as well, and can be extended to include more complicated non-smoothness for comparison with dihedral angles and other discrete curvature measurements contained in our metric.

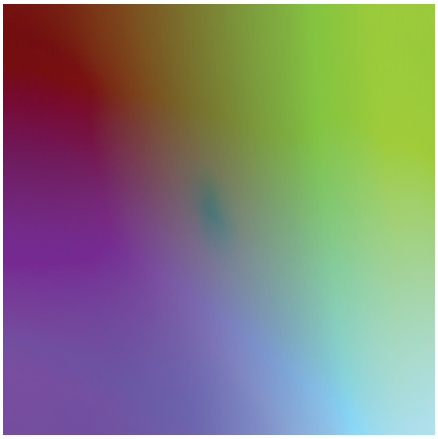
Obtaining dihedral angles from a mesh, while intuitive mathematically, requires a fair bit of coding. Since our data forms regular squares on the xy-plane, the naive approach to generating the angles would be to find the normals for each square about a point, then compare each combination across that point, and attribute those angles to the point. However, the data is not quite so nicely behaved, and the data, while forming squares on the xy-plane, are in general not coplanar. That is, a normal can not be ascribed to each face of the mesh, because they are not in fact faces. Solving this problem requires triangulating the mesh. In the interests of saving computation time, the triangulation only connects the points (x, y) and $(x + 1, y - 1)$ for all (x, y) in the mesh such that $(x + 1, y - 1)$ is also in the mesh. Since three points define a plane, every triangle must be coplanar, and so for any three points there is a unique normal for the face described by those points. Thus calculating the normals of the mesh faces is possible using the function `find_normals`. Then, using the function `find_dihedral_angles` the dihedral angles of every combination of normals across a point is ascribed to that point. Since the triangulation creates six triangles around every point not on an edge, there are $\binom{6}{2}$ dihedral angles ascribed to every point not on an edge. From these 15 dihedral angles it is possible to



(a) An sRGB gradient image before transformation.



(b) An sRGB gradient image after transformation with an unaltered profile LUT.



(c) An sRGB gradient image after transformation with one LUT R channel value changed.

Figure 6.3: A series of gradient images beginning in sRGB. The first image is the input slice represented as a gradient. The second is the first slice passed through the profile via the CIELab to sRGB perceptual intent transformation. The third is the same as the second, but with one R value altered in the perceptual intent transformation.

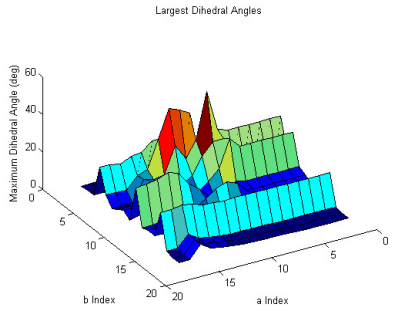


Figure 6.4: Maximum dihedral angle at each point in the $L^\star$ slice with $L = 81$.

get an idea of nonsmoothness at a point. Also possible is generating the absolute mean curvature of the point, using the 6 proper dihedral angles from the 15. Figure 6.4 shows the maximum dihedral angle at each point in the hyperplane shown in Figure 6.2(a). This data was generated using the function `main_test`, which incorporates and automates the previous two functions.

6.1.4 Solutions to Problems In the Metric Implementation

As a result of the initial attempts of establishing thresholds, problems arose. The initial attempt only included measurement of changes in the BtoA0 tag of the profile. Also included should be measurements of changes in the AtoB0 tag of the profile, because we wnat to evaluate both directions of an ICC profile. Another, more immediate issue is that the initial attempt at changing the BtoA0 tag began with RGB data and ended with RGB data, for simplicity of coding. However, ultimately, the data should begin and end with CIELab data, to correspond with our overall model of the problem. Coding this is more difficult than our initial attempt, but still possible.

First, note that the BtoA0 tag outputs data in sRGB space, as mentioned previously, yet any alterations need to be made in CIELab space. Beginning with data in CIELab space we can use the unmodified BtoA0 tag to go from CIELab to sRGB. From here, we can use a modified version of the relative colorimetric intent transformation to get from sRGB to CIELab. This modified version of the transformation will be the original relative colorimetric intent transformation, except for a few changes to the LUT, as before. The difference now is that we can change say the L channel by 10, instead of the R channel by 10. Now we have data that has so far passed through an

unmodified BtoA0 tag, and the relative colorimetric intent transformation from sRGB to CIELab which has been modified to include non-smoothness errors. Using the relative colorimetric intent transformation from CIELab to sRGB, the overall process essentially describes a new perceptual intent transformation, or a new BtoA0 tag which has been modified to include non-smoothness errors which modify data in CIELab, rather than in sRGB. Now we can apply this new transformation to a regular grid of CIELab data, and then apply the original sRGB to Lab relative colorimetric intent transformation to generate gradient images exactly as before, except in CIELab space, instead of sRGB space.

While we have so far produced an implementation for correlating mathematical non-smoothness in CIELab space with visible non-smoothness, we must be careful. Matlab interprets color data as sRGB when producing images. To display the new CIELab gradient images as they should be viewed, we must first save the gradient images as sRGB TIFF files, and then load them into Photoshop. Using this program, we can change the R channel to $L^\star$, G to $a^\star$ and B to $b^\star$. With the image in CIELab space we can proceed normally, correlating mathematical non-smoothness with visible non-smoothness.

To include measurement of changes in the AtoB0 tag, from CIELab to CIELab, is simpler than for the BtoA0 tag. Since the data resulting from the transformation is in CIELab space, changes can be made directly to the AtoB0 tag to produce the gradient images. So, by beginning with data in CIELab space, and passing it through first the CIELab to sRGB relative colorimetric tag, and then the AtoB0 tag (sRGB to CIELAB) produces the desired results. Unfortunately, care must be taken for this direction through the profile, as colors outside of the sRGB gamut will be clipped to the boundary of the gamut. Marking the out of gamut points before passing them through the tags, and then marking the boundary of the gamut in the gradient image will suffice to solve the problem. In this direction, we will not be evaluating experiments for non-smoothness thresholds; it suffices to measure these in one direction. However, this process is still useful for evaluating non-smoothness in this direction, to ensure that our metric is producing the correct results once we begin to automate the process.

Now that we can evaluate non-smoothness in either direction through the profile, and will be able to obtain thresholds for non-smoothness, it will be possible to automate the non-smoothness correction process. We begin by reporting non-smoothness to a user, once thresholds have been determined for non-smoothness. We will simply check every point using some metric defined by discrete curvature measurements, as discussed above.

For each point, return whether that point is smooth or not. If a given point is non-smooth, visualize for the program's user the non-smoothness using gradient images, and then attempt to automatically correct the non-smoothness. Show the same gradient after the correction is made, for comparison. Using this method it will be possible to determine the effectiveness of our model at reducing non-smoothness.

6.2 Spring Semester

During the spring semester we examined the visual correspondence between the discrete curvature values and visual non-smoothness. Toward this end, we wrote the functions `worst_points`, `change_matrix`, and `pull_point` described below. In examining the relationship between metric values and visual non-smoothness, we concluded that we did not have the expert eye or testing capacity to determine absolute thresholds, so we adjusted our approach to include the threshold as user input.

We also discovered that we had the capacity to construct trivariate spline interpolants, and we implemented these, replacing the bivariate splines in our previous implementation.

As we were coding during the spring semester, we discovered the data structure issues in the AtoB0 tag that required us to develop the three methods described in chapter 5. To solve these issues, we implemented a tetrahedral interpolation method, using some code from last year's HP Labs math clinic. We also implemented an iterative method for correcting non-smoothness using splines.

6.2.1 Finding Thresholds

In order to determine thresholds for our metric, our we introduced a variety of artificial non-smoothness problems to an existing smooth ICC profile. The simplest of these is the perturbation of a single point. We make modifications to the BtoA0 tag of a smooth profile in order to avoid out-of-gamut issues, among other things. We change one coordinate of one output point in the transformation schematized in figure 6.7.

Lab_{in} is a table of regularly spaced Lab values. Next we use the BtoA0 tag of the profile to transform these grid points to RGB, and then apply the relative colorimetric intent to transform back to Lab. Next we alter one of the coordinates of a single point in Lab_{out}. This gives us LAB_{new}, which only differs by that single change. We now have an Lab to Lab transforma-

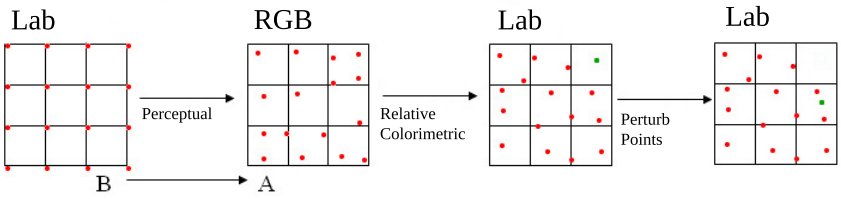


Figure 6.5: Visualization of a transformation including the device independent to device dependent perceptual intent tag of a profile.

tion from Lab_{in} to Lab_{out} on which we can take slices and calculate discrete curvatures.

In order to view the corresponding visual ramifications of this change, we need to substitute back into the Lab to RGB CLUT of the BtoA0 tag, so we can apply that tag to a gradient image. So we convert Lab_{out} back to RGB using the relative colorimetric tag. The resulting RGB values are the new output values for the CLUT.

Now, more detail about this process and the structure of each of these objects: Lab_{in} is a two-dimensional table with 17^3 rows and 3 columns. The first column contains L values, the second column contains a values, and the third contains b values. Moving down the table, values proceed through the Lab space in a regular order: b changes first, then a, then L, as shown below.

L	a	b
0	-128	-128
0	-128	-112
0	-128	-96
$\vdots$	$\vdots$	$\vdots$
0	-128	127
0	-112	-128
0	-112	-112
$\vdots$	$\vdots$	$\vdots$
0	127	127
6.25	-128	-128
6.25	-128	-112
$\vdots$	$\vdots$	$\vdots$
100	127	127

Similarly, as we apply transformations to this input table, at each step we have a $17^3 \times 3$ table with values (in RGB or LAB respectively) corresponding to the LAB_{in} entry at that row. We use the index i to refer to the ith row of these tables. Because LAB_{in} has the regular arrangement shown above, we always know what the input value is for a given row index i.

In addition, we need to translate between the $17^3 \times 3$ data structure of Lab_{in} and the $17 \times 17 \times 17$ data structure of the color look-up table. Each entry in the CLUT is an RGB value indexed by the LAB point that maps to it. So the (a, b, c) entry of the CLUT is a triple in RGB. Since the CLUT is also regularly spaced, there exists a one-to-one correspondence between the row index i and the BtoA0 CLUT indices (a, b, c). This is shown below.

$$\begin{aligned} c = &\ \lfloor (i-1)/(17^2) \rfloor + 1 \\ b = &\ \lfloor ((i-1) - (c-1)17^2)/17 \rfloor + 1 \\ a = &\ (i-1) - (c-1)17^2 - (b-1)17 + 1, \end{aligned}$$

where we need to add 1 because MATLAB indexing starts at 1.

We also specify a 17×17 matrix to superimpose upon an entire slice, changing the z-values of the output mesh. We constructed a variety of matrices that mimic real life non-smoothness, which we store in the function `change_matrix`.

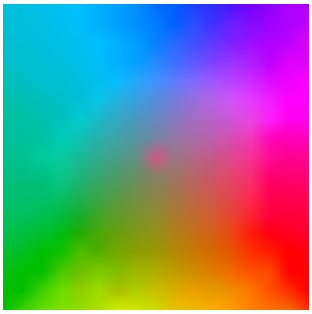


Figure 6.6: The slice with input L held constant at 56.25 in profile `newest.icc` with the center point pulled by 50 in the a direction.

6.2.2 Mimicking Real World Non-Smoothness Problems

Ingeborg gave us glossy printouts of output slices from gradient RGB inputs. Each of these inputs is a face of the RGB cube, except for three diagonal slices through the RGB cube. In these printouts one can observe real world non-smoothness problems: dark bands, sudden color changes, etc. We want to model these real world non-smoothness problems using our slices. We want to create a matrix to input into pull-point that mimics the patterns we see in the printouts. We overlay transparency paper and estimate the delta-z values for a slice. This produces a matrix which we can then use as input to `pull_point.m`. Many of the values in this matrix may be zero.

6.2.3 Threshold Values

First threshold estimates were established by pulling a single point from a smooth profile in either positive or negative L, a or b directions until non-smoothness was just barely visible. The location of this point, amount pulled, and discrete curvatures at that point were then recorded. This was repeated over different points in the thresholds. The resulting data told us what metric measurements correspond to barely visible non-smoothness, which we then used as threshold values. The average and minimum values of the curvature measures at the points pulled are as follows. The columns in this table correspond to the three directions that the outpute could be in.

	L	a	b
Average Maximum Dihedral Angle About the Points:	.3642	0.8516	1.15
Average Absolute Mean Curvature:	4.633	21.4949	48.4025
Average Gaussian Curvature:	0.1075	0.8683	1.548
Min of Maximum Dihedral Angel About the Points:	0.2785	0.61	0.94
Min Absolute Mean Curvature:	3.19	10.9	16.15
Min Positive Gaussian Curvature:	0.05	0.75	0.5263

Note that in almost no cases was Gaussian curvature negative, which would indicate a saddle point. When Gaussian curvature was negative, the negative values were minuscule (usually greater than $-.06$ except in one case.) To be exact still, the negative values in the L direction were on

Figure 6.7: The slice with input L held constant at 31.25 in profile `newest.icc` with a matrix modelling real-world non-smoothness applied to it.

average $-.036$, in the a direction $-.11$ and no negative Gaussian curvatures were seen in the b direction.

We also created models of real world non-smoothness issues, allowing us to pull a set of points in a slice so it would have non-smoothness corresponding to that seen in profiles sent to us by our liaison. By testing these we saw that the thresholds established above would catch points around the generated non-smoothness. We also began finding significant negative Gaussian curvatures in points about the introduced non-smoothness. From these tests it appeared that points with Gaussian curvature less than -.04 should be red-flagged.

Next we set different thresholds and checked which points in a slice were then red-flagged. This allowed us to see if certain threshold values were too sensitive or insensitive to non-smoothness. After all this, it seemed reasonable to red-flag points if absolute mean curvature was greater than 4 in the when L is the output, greater than 10 when a is the output, or greater than 15 when b was the output. For any output we also would red-flag point with Gaussian curvature less than -0.05 or greater than 1, or if the maximum dihedral angle about that point is greater than 0.8.

6.2.4 AtoB0 Tag Difficulties

As detailed in Chapter 5, the Lab→Lab structure required by the metric conflicts with the RGB →Lab structure required for substituting into the CLUT.

We could have used an RGB →Lab transformation, but pRGB is not perceptually uniform. We discussed the perceptual uniformity with our liaison, who told us it would depend on the ink separation for each printer. Color scientists attempt to make the separation linear, but have different levels of success depending upon the printer. So the problem with an RGB →Lab method is the range of acceptable metric values would vary from device to device.

After some discussion, we implemented the following three methods. Here we talk in more detail about how our implementations work, we show some output we produced, and talk about how we solved the issues that arose along the way.

6.2.5 Revising the Interpolation Method

In our implementation of Method 1, we encountered some complications in the interpolation. Our code follows these steps:

1. Make a regular RGB grid, tetrahedralize it using `delaunay3`, a built-in Matlab function that creates a Delaunay tetrahedralization.
2. Run AtoB1 on the RGB grid to obtain Lab_{in}.
3. Using Lab_{in} as the endpoints of the tetrahedrons, we interpolate the grid Lab points contained in those tetrahedrons. The Matlab function `tsearchn` determines which tetrahedron the grid point is contained in, and then calculates the barycentric coordinates of the grid point. If the grid point is out-of-gamut, `tsearchn` cannot find a tetrahedron, and so returns `NaN`.

The problem, then, is that some points (54 or so out of the 4913) are considered by the interpolation to be out-of-gamut, even if independent calculations show them to be right in the middle of the gamut. This problem is caused by the fact that we tetrahedralized the data in RGB, and then transformed it to Lab, and some tetrahedrons may have collapsed. If the tetrahedron has a very small volume, it causes a numerical error: Matlab must divide by zero to calculate the barycentric coordinates, and the result is `Nan`. To solve this, we perturb those points by a gradually increasing 'epsilon' until we can find a couple of tetrahedrons that are close to the grid Lab point. Using those vertices we make a good tetrahedron for the *original* point, and find the corresponding barycentric coordinates.

After this whole process, we have appended new tetrahedrons (maybe up to 54 of them) onto the original set of tetrahedrons, and every single grid Lab point will have interpolation information. Now rather than using the imprecise BtoA1 tag, we interpolate our grid Lab points.

6.3 Using Our Methods

The main functions with which to access our methods are

```
thresholds_test
worst_points_test
correct_profile
```

The following sections outline various uses of the code. See the comments of each function for specific syntax and explanation of output.

6.3.1 `thresholds_test`

This function is a semi-complete version of our proposed **non- smoothness metric**. As proposed, the non-smoothness metric would use discrete curvature measures, and thresholds on those measures, to identify problem points in a profile transformation. It is only "semi"-completed because we have not established actual numerical thresholds. We realized that, without the requisite experience, any numbers we come up with are probably inaccurate. In order to turn this function into the completed metric, one would have to find appropriate thresholds (perhaps depending on grid size and other factors), plug these into the function, and adjust the output of the function for the desired result.

The `thresholds_test` function is, however, currently useful as a **proof-of-concept** of our discrete curvature measures. If a user wishes to see how well our methods are working, he or she can input some liberal (low) thresholds to the `thresholds_test` function, and see whether the marked over-threshold points correspond well with visual non-smoothness in the output slices displayed.

6.3.2 `worst_points_test`

As a replacement for a working metric, we have the `worst_points _test` function, which is intended for use in **speeding up manual non-smoothness identification**. The function sends slices from across the entire gamut through the given profile, and outputs visualizations of those slices that contain the points of most severe discrete curvature. This gives the user an idea of where the worst non-smoothness exists in the gamut. Also, since several "worst point" slices are provided, if the user sees no visible non- smoothness in these worst slices, then it's likely that the entire profile is smooth. In this way, the `worst_points_test` function provides a faster, less subjective, more complete way of manually identifying non-smoothness than current methods.

6.3.3 `correct_profile`

The `correct_profile` method is the implementation of our scheme to automatically correct ICC profile non-smoothness using splines. Originally, we proposed that this method would take into account our metric's measurements of the profile, but in its final incarnation, this is not the case, for several reasons: (1) our metric is unfinished; (2) our former idea of

only correcting points that were over-threshold can actually *worsen* non-smoothness; (3) it's time-consuming to implement and to execute, and is almost certainly not worth the effort. Instead, the `correct_profile` method works all on its own. The user inputs the profile/tag and a "smooth factor"—a number from 0 to 1 that determines how far the CLUT points will be adjusted towards the smooth, splined approximation. Here, the more we adjust towards the spline, the less our new CLUT will respect the profile's intent, so the variable smoothing factor allows the user to find a balance between the two extremes. This function can also be used iteratively: adjust by a small amount, then re-spline, adjust again, etc., until the desired smoothness/intent balance is reached.

Chapter 7

Year Summary

7.1 Fall Semester

Building our Background. We spent quite some time getting to know our problem, learning about color science and color management, and understanding various mathematical topics, including splines and elements of discrete curvature. We familiarized ourselves with relevant software, including Adobe Photoshop, HP's GamutViewer, X-Rite ColorPort, and MATLAB (including the Spline and Color Management toolboxes).

Modeling our Problem and Resolving Issues. The next step was to model our problem. We decided how best to compare and manipulate data, how to work with ICC profile structures, and how to implement our techniques using MATLAB. This process, and the issues we resolved along the way, are explained in the Chapter 6.

Starting Work on Code. By the end of the fall, we finished a significant portion of the initial implementation of our non-smoothness metric. Specifically, we wrote MATLAB code that determines dihedral angles at every grid point of output slices as described by our problem model. The functions are listed and described in more detail in the Appendix.

7.2 Spring Semester

Implementation of the Metric. At the beginning of the spring, we added to the dihedral angles functionality by writing functions to determine ab-

solute mean curvature and Gaussian curvature. We implemented a tagging system to account for out-of-gamut points, so that we would not concern ourselves with metric values for those points. We also wrote code to display images of output slices on the monitor, which would allow us to compare metric values to visual appearance.

Establishing Thresholds and Fine-Tuning the Metric. Next we split up into subteams in order to examine the correspondence between metric values and visual non-smoothness. One subteam wrote code to introduce artificial non-smoothness to an already smooth profile. The other subteam wrote code to search through the entire profile and find the worst points, as well as code to mark all points in an image of a slice that are above a user-determined curvature threshold. When we combined the efforts of the two subteams, we had some data on what curvature values correspond to barely visible non-smoothness. However, looking at the data, we decided that it would be impossible to fix an absolute threshold and that we would shift our implementations of the non-smoothness metric so that the discrete curvature threshold is a user input, rather than an absolute.

Implementation of Splines for Non-Smoothness Correction During the spring, we discovered that we had the capacity to construct trivariate spline interpolants. We implemented a function that creates a trivariate spline for each output variable.

Confronting and Solving Data Structure Issues While working intimately with the data structures in the profile tags, we realized that it was difficult to translate between our metric data structures and the profile data structure, and we revised our code accordingly. In the case of the AtoB0 tag, we encountered difficulties obtaining an Lab→Lab transformation beginning with a regular Lab grid. We implemented three ways to solve this problem: 1) interpolation, 2) use profile tags, and 3) use an RGB → Lab transformation.

Presentations and Final Report Towards the end of the spring, we gave a presentation at Harvey Mudd to the other students, faculty, and liaisons. We also documented our work in our Final Report.

Finalizing Implementations and Packaging the Deliverables At the end of the semester, we finalized an iterative spline correction method and a

function to create new color look-up tables. We also finalized the interpolation method for obtaining an Lab→Lab transformation. Last, we organized our code architecture, commented it thoroughly for outside users, and packaged it all together.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

As our final product, we have produced a package of Matlab programs that

- calculates the absolute mean curvature and Gaussian curvature for all slices in the profile,
- finds and displays the slices with the worst discrete curvature measurements, and marks the worst points in the image,
- corrects non-smoothness problems by adjusting all points toward a trivariate spline interpolant, allowing the color scientist to determine how drastically to adjust the points.

Our programs accomplish the above tasks for both tags of the profile, and for CLUTs of varying sizes. The tests we have conducted on our metric show that discrete curvature does correspond with non-smoothness, but because we do not have the expertise to recognize barely visible non-smoothness, nor do we have the time to run extensive tests, we cannot determine absolute metric thresholds.

Similarly, the tests we have conducted on our spline correction method show that adjusting toward a spline does create smoother ICC profiles. Again, we do not have the expertise to recognize whether the changes the program makes respect the profile's perceptual intent to an acceptable degree, but our non-expert evaluation is that the method works well.

Our programs allow for the color scientist to customize how drastically the spline correction adjusts CLUT entries, and they allow the color scientist to view the metric outputs in the context of their corresponding visual

transformation. By allowing for this interface with the expertise of the color scientist, we have created the tools to identify and correct non-smoothness problems in ICC profiles.

8.2 Future Work

Because we were non-experts, we were not able to produce a fully automatic non-smoothness identification and correction system. For a fully automatic identification system, we would need to establish absolute thresholds for the metric. This would require extensive testing and an expert eye. In addition, these thresholds may have to be direction-dependent, since the perceptual uniformity of the Lab space is not perfect. The thresholds may even vary across different regions of the space.

Future work with the splines could include experimentation with knot configurations. Finally, in future work, one might seek to interpolate the entire 3D $\rightarrow$ 3D CLUT at once. This is theoretically possible, although it is out of reach in the Matlab toolbox.

Appendix A

Glossary

A.1 Color Science

Achromcatic Color Perceived color devoid of hue.

Brightness Amount of light perceived to be emitted by a color.

CIELAB Space A perceptually uniform, device independent color space. The space itself is based on a mostly non-linear transformation of the CIEXYZ values of colors. The three bases of the space are lightness (L^*), a red-green axis (a^*),and a blue-yellow axis (b^*). The space can also be thought of as a cylindrical space, with the line on the lightness axis representing achromatic colors. In this case, cylinders about the gray line are of constant chroma and half-planes radiating from the gray line are constant hue. It should be noted the space is not entirely perceptually uniform, as the hue-constant planes are not equally spaced around the gray line, and chroma-constant cylinders are not a constant distance from the gray line. See section 2.1 of this document for further explanation.

CIEXYZ Space A perceptually non-uniform, device independent color space useful for converting values from device dependent spaces to values in CIELAB space. The space itself is based on tristimulus value color matching functions dependent on three bases, X, Y and Z, wherein color combinations of two colors are found on the line which lies between the wavelength representation of those colors. Furthermore, a color's opposite lies on the opposite side of a line passing through the source white and the original color. See section 2.1 of this document for further explanation.

CMY/CYMK Space A perceptually non-uniform, device dependent color space useful for printing devices. The three bases of the space are cyan (C), magenta (M) and yellow (Y), with black (K) added to improve the gamut of the space. Black is a linear combination of the other three colors in this space, hence it is not four-dimensional. The space is subtractive, meaning that colors are expressed in terms of light absorbed, rather than light emitted. For example, if a color absorbs a high amount of red light, it will appear cyan, and thus be emitting only significant quantities of blue and green light. Contrast this with RGB space, which is additive.

Chroma Relative measure of the colorfulness of a color compared with the brightness of a source white. Can be expressed as $Chroma = \frac{Colorfulness}{Brightness(white)}$.

Chromatic Color Perceived color possessing a hue.

Color Space A mathematical space modeling color using a set of linearly independent bases. Perceived colors can always be expressed as a combination of three bases, so color spaces are always three dimensional. Examples include: RGB, CMY, CIELAB and CIEXYZ.

Colorfulness Absolute amount of chromaticity of a color. Colors with higher colorfulness appear richer than other colors.

Gamut A subset of a color space representing those colors which are reproducible or detectable by a given device.

Hue Absolute measure of whether a color is red, yellow, green or blue, or some combination of red and yellow, yellow and green, green and blue, or blue and red. Combinations of red and green, or blue and yellow, form gray colors.

Lightness Relative perception of the brightness of a color compared with the brightness of a source white. Can be expressed as $Lightness = \frac{Brightness}{Brightness(white)}$.

Luminance Measured amount of light energy emitted by a radiating or reflecting body.

Perceptual Uniformity A color space is said to be perceptually uniform if a color change in the space corresponds with a color change in human perception.

Profile Connection Space A device independent color space which can be used to transition color data between two ICC profiles.

RGB Space A perceptually non-uniform, device dependent color space useful for light emitting or light capturing devices, such as cameras and monitors. The three bases of this space are red (R), green (G) and blue (B). The space is additive, meaning that colors are expressed as combinations of emitted light. Contrast this with CMY space, which is subtractive.

Rendering Intent Method for mapping colors from one gamut to another. Several exist, including colorimetric intents, which are designed to maintain color accuracy, and perceptual intent, which is designed to maintain contrast ratios between colors.

Saturation Relative measurement of the colorfulness of a color in relation to its brightness. Alternately, the chroma of a color in relation to its lightness. Can be expressed as
$Saturation = \frac{Colorfulness}{Brightness} = \frac{Colorfulness}{Brightness(white)} \cdot \frac{Brightness(white)}{Brightness} = \frac{Chroma}{Lightness}$.

Tristimulus Values The weights given to each of the fixed primary stimuli (basis vectors of the color space) to specify a certain color.

A.2 Splines

Affine Invariance Applying an affine transformation to a control polygon and then rendering the curve has the same effect as rendering the curve and then applying the affine transformation; that is to say, the two processes commute.[1]

Affine Transformation Any transformation which is a composition of rotation, dilation, translation and shear.[2]

Convex Hull For a set of vertices, the convex hull of those vertices is the space contained between any two or more of the vertices, including the boundary. [2]

Convex Hull Property A spline curve segment is entirely contained within the closed convex hull of the control points affecting that segment.[1]

[1]Source: *Knotty: A B-spline Visualization*
[2]Source: `http://mathworld.wolfram.com/`

Vertex Locality Changing the position of a control point of a B-spline curve changes only changes the nearest k intervals, where k is the order of the curve.[3]

Knot Locality Changing a knot of a B-spline curve changes only the $k-1$ intervals to the left and $k-1$ intervals to the right, where k is the order of the curve.[3]

Variation Diminishing Property A straight line will not pass through a B-spline curve more times than it passes through the lines segments sequentially connecting the control points of the curve.[3]

Closed End Condition The $k-1$ control points at each end of an order k B-spline curve overlap, and the k segments formed by the $k+1$ knots at each end overlap. When these conditions are met the curve is closed.

Floating End Condition No condition is enforced on the endpoints of a B-spline curve, and thus the endpoints of the curve "float" without necessarily meeting the first and last control points, or each other.

Open End Condition The control points at each end of an order k B-spline curve have knots with multiplicity k. When this condition is met the endpoints of the curve are the same as the first and last control points, respectively.[4]

Kernel Algorithms of B-spline Curves Any algorithm which allows for simplifying or refining a B-spline curve. For example, turning a floating end condition curve into an open end condition curve, adding control points, or adding knots to an existing B-spline curve for additional control, without modifying the shape of the curve.

A.3 MATLAB Spline Toolbox Commands

`cs = csapi(x,y)` Takes as input an array x of x-axis data and an array y of y-axis data (this can be a function of the x-axis data) and outputs a cubic spline interpolation of the input data.

`csprime = fnder(cs)` Takes as input a function cs and outputs its derivative.

[3]Source: *Knotty: A B-spline Visualization*

[4]Source: `http://www.math.hmc.edu/~gu/math142/mellon/Application_to_CAGD/B-Splines_and_Spline_curves/other_Conditions.html#floating`

`csbig = fnint(cs)` Takes as input a function cs and outputs its integral.

`v = fnval(f, x)` This functions takes as input a function f and an array x of x-axis values, and replaces each data point with the function f at that point.

`csp = csape(x, y, conds)` Takes as input an array x of x-axis data and an array y of y-axis data (this can be a function of the x-axis data) and outputs a cubic spline interpolation of the input data with end conditions conds.

`spline = spapi(k, x, y)` Takes as input an array x of x-axis data and an array y of y-axis data, and outputs a spline of order k.

`spline = spaps(x, y, tol)` Takes as input an array x of x-axis data and an array y of y-axis data and a number tol, and outputs a B-spline minimizing a smoothness condition which is a transform of the square of the m-th derivative (default m=2) of the spline.

`spline = spap2(l, k, x, y)` Takes as input an array x of x-axis data, an array y of y-axis data and outputs a B-spline of order k with l polynomial pieces.

`knots = optknt(x, k)` Takes as input an array x of x-axis data and an order k, and outputs an optimal knot sequence for the given data.

`knots = newknt(sp)` Takes as input a spline sp and outputs a new series of knots which are better distributed.

(Note that all of the above functions can be extended to bivariate tensor product spline surfaces. Also, by forcing a spline to pass through particular sets of points, one can create spline approximations to closed curves and surfaces.)

`curve = cscvn(points)` Takes as input a set of points and outputs a curve which is a natural cubic spline curve passing through the points. If the first and last points coincide, with no repetitions, the curve is forced periodic. If there are other reptitions, then corners may appear in the curve. This function may be extended to $\mathbb{R}^3$.

Appendix B

MATLAB Code

B.1 Summary of Functions

`main_test(hold_var,hold_val_index,out_var)` A function that combines all the other functions to compute dihedral angles and output the data in an easily visualizable fashion. The relevant data slice is based on the three input variables; `hold_var` is the input to hold constant, either `'L'`, `'a'`, or `'b'`, `hold_val_index` is the index of the held value, and `out_var` is the variable to look at as output.

Sample use of `main_test`:

```
max_dihedral_angles_of_slice = main_test(input_const_param, ...
input_const_val_index, output_param);
```

Ex: `X = main_test('L', 5, 'L');` would keep input L constant at the 5th value in the original input range of L values (among 17 possible values) and look at L output. `X` will be a 17x17 grid of the max dihedral angle at each mesh point.

`array2matrix(A)` Converts a linear array into a square (2-dimensional) one. This requires that the data is in two variables.

`AtoB0Tag(gridInput, profileName)` Converts the color data contained in gridInput first through the relative colorimetric B to A tag of the profile given by profileName, and then through the perceptual A to B tag of the same profile. The starting data must be in Lab, and the output will also be in Lab.

`BtoA0Tag(gridInput, profileName)` Converts the color data contained in gridInput first through the perceptual B to A tag of the profile given by profileName, and then through the relative colorimetric A to B tag of the same profile. The starting data must be in Lab, and the output will also be in Lab.

`ColorArray(nrSamples, r, g, b)` Given a number of samples nrSamples, usually 17 or 33, the number of data points on each axis of the grid used, and red, green and blue color data, r, g and b respectively, this code will create a 3-dimensional array containing all entries with a color vector [r g b] which can then be used to create constant-color meshes.

`create_lab_grid_range(nr_samples)` Returns a data structure that contains a regular grid in LAB space, along with arrays representing the range of each variable L, a and b.

`find_dihedral_angles(normals)` Given a set of normals in a grid (as outputted from `find_normals`), computes every dihedral angle at each point in the grid. Right now, the function returns only an array of the maximum dihedral angle found at each point, but can be modified to output an array of angles showing each.

`find_normals(slice_datapoints, x_axis, y_axis, dim)` Given a slice of color data, along with the range of each original grid input, finds the normal vector corresponding to each triangular face in the mesh.

`first_test(const_param, out_param)` Given a parameter to hold constant, 1 for L, 2 for a, 3 for b, and a parameter to output, the same, this function will generate a variety of images depending on which parts of the code are commented out. By default all entries are commented out, but the code has the capability of generating slices as they appear before and after the AToB0Tag code is implemented, as well as subsequent slices. It can even generate an image demonstrating the change between a slice and its spline approximation.

`slice(A,B,hold_col,hold_val)` A function that gives us a "slice": it takes input and output data, plus a parameter and a value to hold constant, and returns the input/output data with those constant values of that parameter.

`site_visit_demo()` A script which produces a series of three images displaying an artifical non-smoothness in a profile tag. The images are displayed in RGB, but the code can be modified to output in Lab.

B.2 Summary of Important MATLAB Built-in Functions

`uint8(x)` Converts any number `x` to an 8-bit unsigned integer. Applicable to entire arrays or matrices. Useful for the built-in application of `srgb2lab` and vice-versa in Matlab's Image Processing Toolbox.

`iccread(filename)` Returns a structure that allows Matlab to access the ICC profile specified by the string `filename`. Important subfields of the structure contain relevant data, such as the CLUT for each of the transformations and intents. Specific tags of interest are the `AtoB0` and `BtoA0` tags, which transform into and out of device-independent space using the perceptual intent, and `AtoB1` and `BtoA1`, which do the same using the relative colorimetric intent.

`makecform(type)` Creates a color transformation structure using a computable transformation specified by `type`. Example: `makecform('lab2srgb')` indicates a direct Lab-to-sRGB transformation. Another transformation of interest is `'srgb2lab'`, which performs the reverse transformation.

`makecform('clut', profile_name, lut_type)` Creates a color transformation structure based on the input parameters. 'clut' signifies that makecform should use a single CLUT contained in one profile tag to create the transformation. `profile_name` must be a string which is the filename of the profile, and `lut_type` is the name of the tag indicating direction and intent. Useful `lut_type` choices include those `AtoB0`, `BtoA0`, `AtoB1` and `BtoA1`.

`applycform(input, cform)` Transforms the data in the array `input` (for our purposes, a $M \times 3$ array) using `cform` and outputs the results, which are the same size as the input, but with the individual values altered based on the transformation cform.

`augknt(knots, k)` Takes the vector `knots` of knot values and outputs the same vector, but in nondecreasing order and with the minimum and

maximum knots repeated `k` times. Useful for creating an ordered series of knots for use with spline generation functions.

`fnval(f,x)` Applies the function `f` to each of the values in `x`. For us, `x` will be a 2-element cell array, with each element a vector. We will use `fnval` to evaluate a spline at given points.

`spap2(knots, order, domain, range)` Generates a function that represents a spline. Specifically, it returns the B-form of the least-squares spline approximation `f` of order `order` and knot sequence `knots` such that `f` maps the data in `domain` to `range`. Since we are constructing 3D splines, `domain` is a 2-element cell array representing an evenly spaced grid, and `range` is one of the output variables of the ICC profile transformation.

`mesh(X, Y, Z)` Creates a 3D mesh plot, with `Z` representing the function values to be plotted; thus, `Z` is a square matrix. `X` and `Y` can be either square matrices or arrays. If they're arrays, then `Z(i,j)` corresponds to the value at `X(j)` and `Y(i)`.

B.3 Code

Appendix C

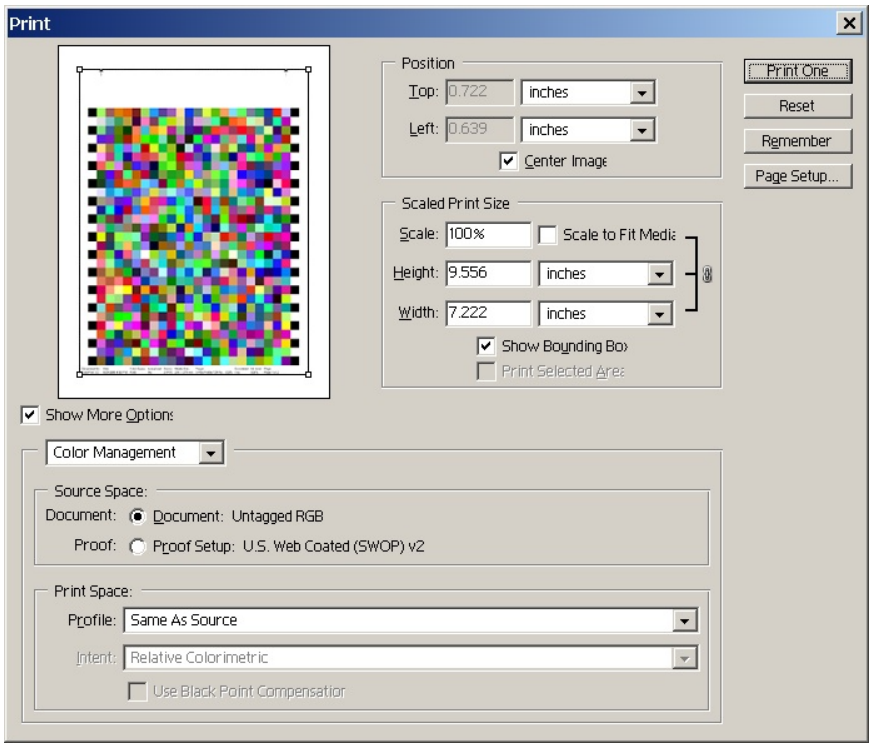
HowTo

C.1 How To Print to the HP Photosmart 7960

How to print to the HP Photosmart 7960 printer with desired settings:

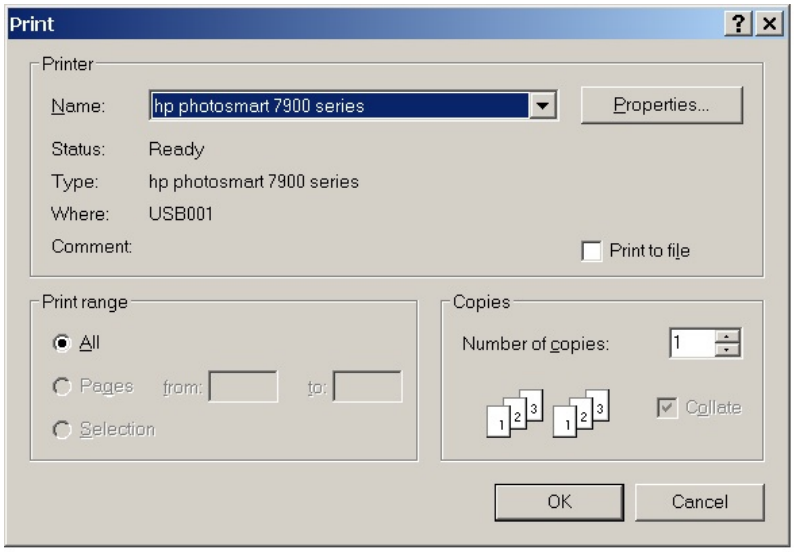
1. First of all, there may be instances when we are not interested in color management. If you want to use the default printer profile and default settings, go to **File**, then **Print**. Pick the HP 7960 from the drop-down arrow, and click Ok to print.

2. If, however, you wish to perform more color management while printing in Photoshop, go to **File**, then **Print with Preview**.

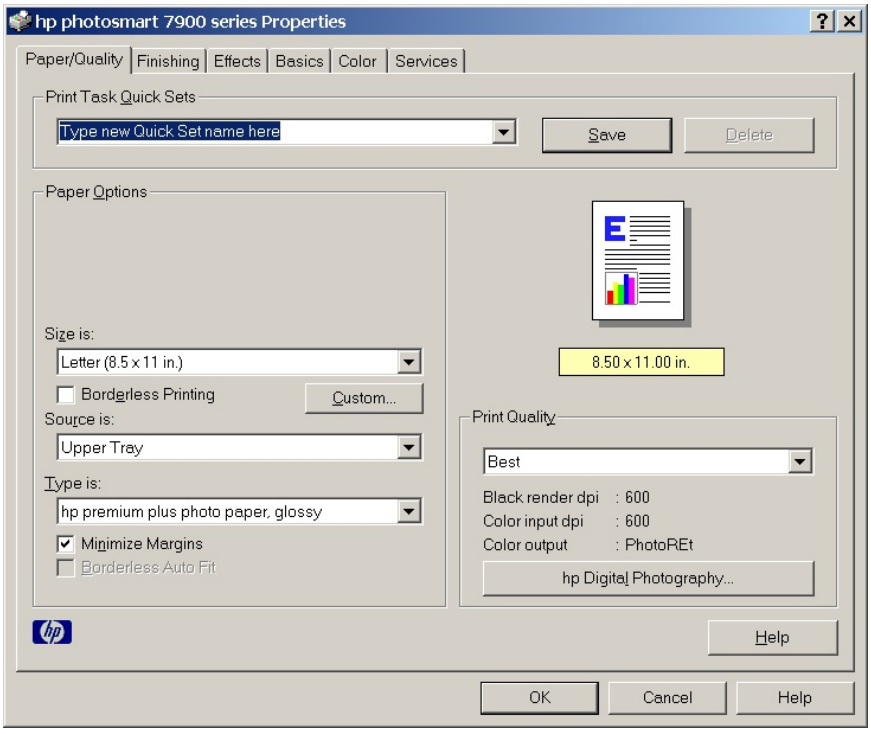


3. In the bottom of the window there is a section labeled **Print Space**. Make sure that in the **Profile** drop-down tab, **Same As Source** is selected.

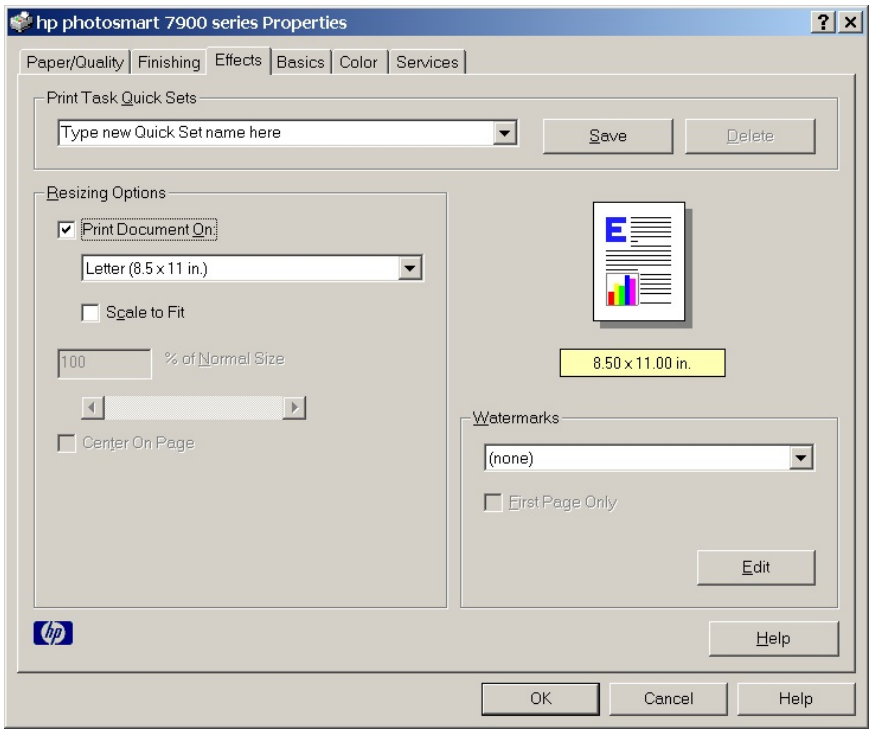
4. Click on the **Print One** button in the upper-left corner.



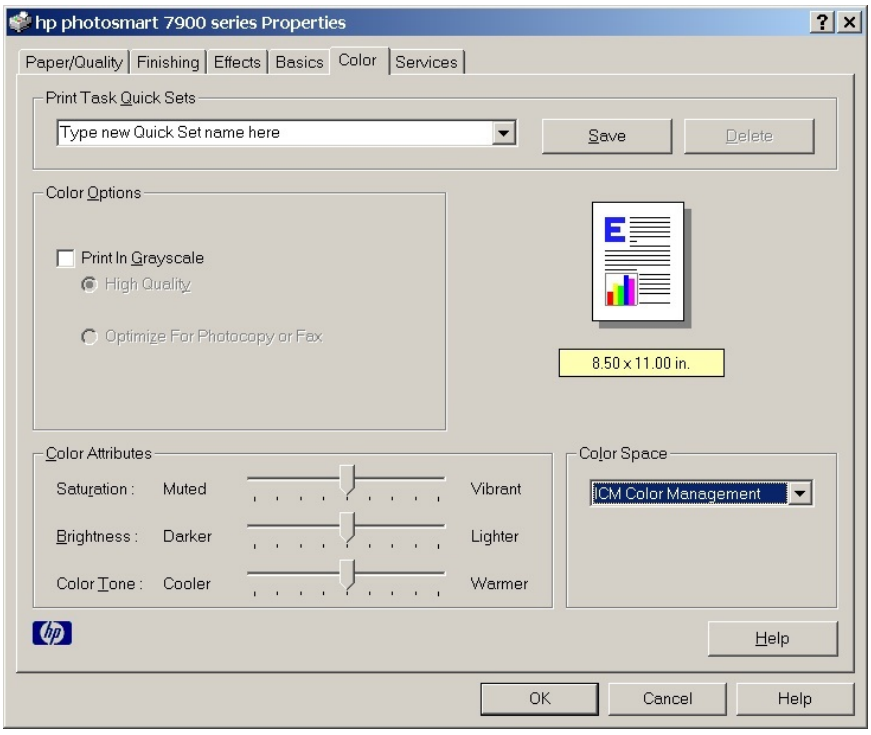
5. In the Print window, under **Name**, select **hp photosmart 7900 series**.
6. Click on the **Properties** button, located to the right of **Name**.



7. In the Properties window, go to the **Type is:** drop-down tab in the lower-left hand corner. Select **hp premium plus photo paper, glossy**. Notice that on the right hand side of the screen, the **Print Quality** drop-down tab has been automatically changed to **Best**.



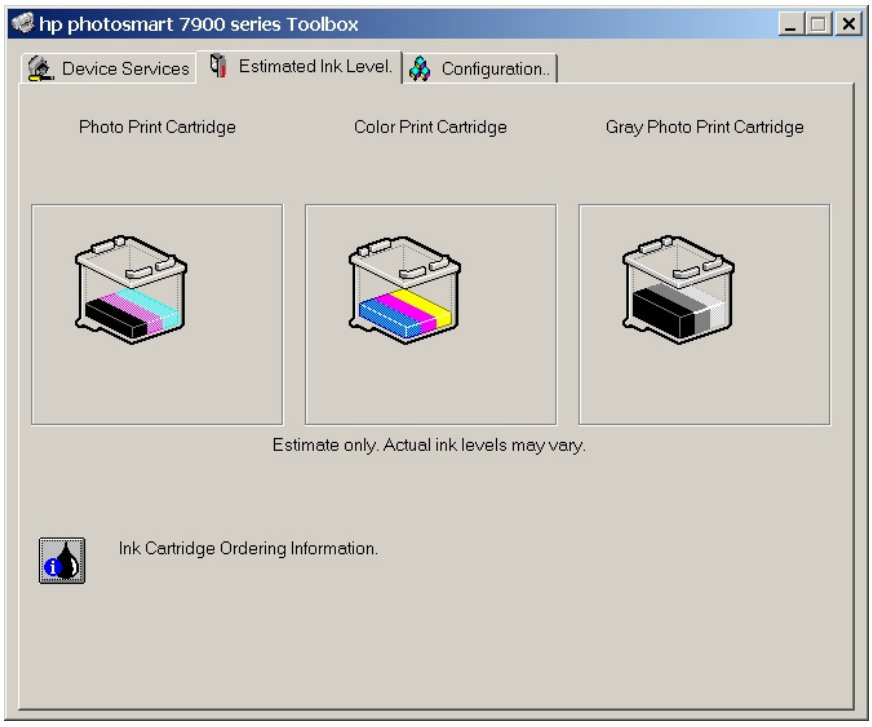
8. On the top of the window, press the **Effects** tab. Under **Resizing Options**, on the left hand side of the window, click on the **Print Document On** box. See that the default **Letter (8.5 x 11 in.)** is selected in the drop-down tab.



9. Go back to the top of the window and press the **Color** tab. In the bottom right corner, under **Color Space**, choose **ICM Color Management** in the drop-down tab.

10. If color management is instead set to **sRGB** and **Same As Source** is checked in the previous window, the printer will use the default printer transformation. Note: This is not desirable for printing targets.

11. Click on the **Settings** tab on the top of the window, and click on **Service This Device**.



12. Click on the **Estimated Ink Level** tab on the top of this window to see the printer's relative ink levels. Be sure to replace the ink when levels get too low.

13. At this point, you should save these settings so you won't have to redo these steps whenever you want to print. In any of the tabs in the Properties window, go to the **Print Task Quick Sets** section on the top of the window. Type a name for your settings state, and click **Save**. Now, every time you want to print, you can select the settings state in the drop-down tab.

14. Click on **Okay**, then **Okay** again in the Print window.

C.2 How To Use X-Rite

Before we can take a measurement of a target or even print it, we must produce the target. To do this,

1. Open the program **X-Rite ColorPort** by clicking Start, mousing over **Programs**, then **X-Rite** and finally **ColorPort 1.0.1**.

2. Change the **Measurement Device** drop-down box to **X-Rite DTP70** on the left side of the window.

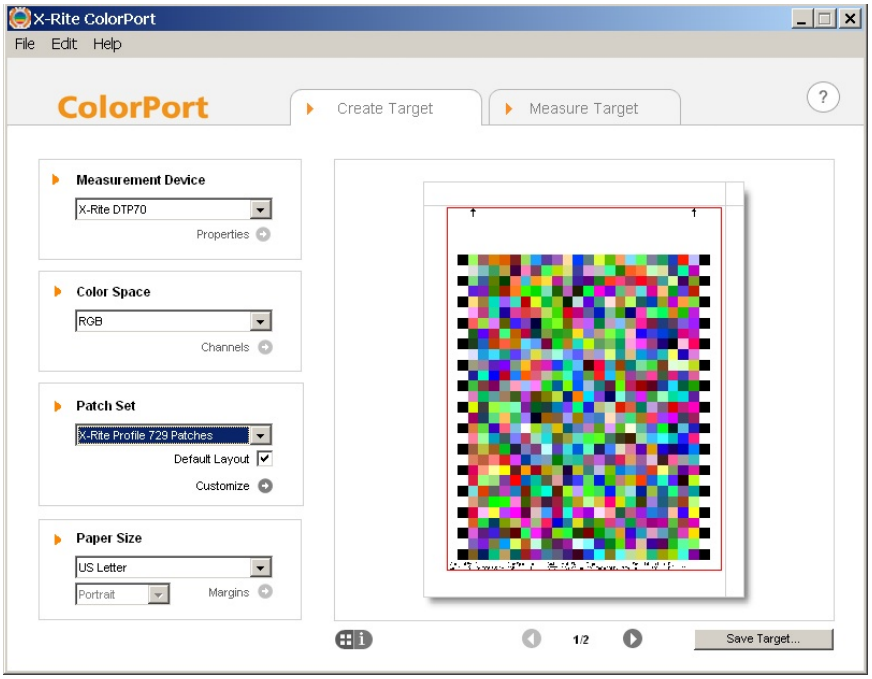
3. Change the **Color Space** drop-down box to **RGB** on the left side of the window.

4. Change the **Patch Set** drop-down box to **X-Rite Profile 729 Patches** on the left side of the window.

5. Make sure the **Default Layout** check-box is checked.

6. Make sure the paper size is **US Letter**, as this is the size of paper the patches will ultimately be printed on.

   Since the program updates the patch preview in real-time, on the right side of the window should be an updated version of the patch set.



7. Click **Save Target As...** in the lower right hand corner of the window.

8. Save to the desired location and proceed to the next section of this document.

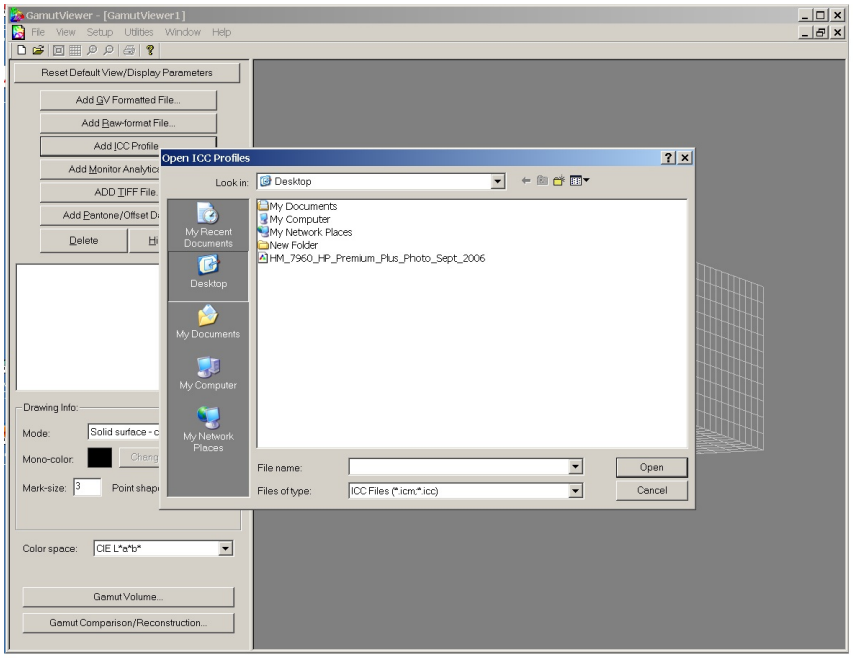
C.3 How To Use GamutViewer

GamutViewer is a useful tool for visualizing color gamuts in a 3D environment. It can be used to compare the gamuts of two devices, or the gamuts of an image and a printer. To use it:

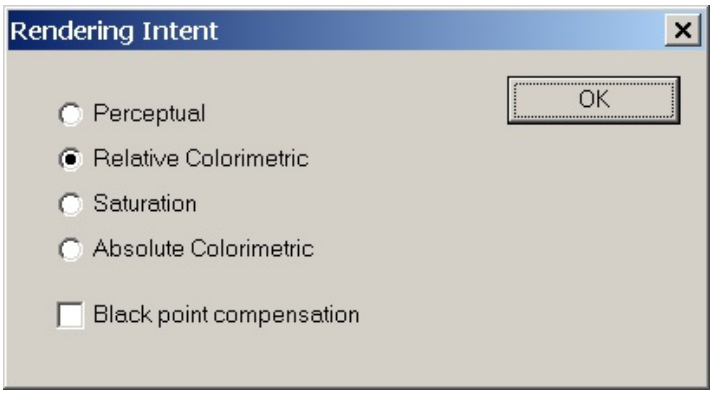
1. Open the program **GamutViewer** by clicking Start, mousing over "Programs", then **GamutViewer** and finally **GamutViewer**.

   Once in the program, to add an ICC profile gamut,

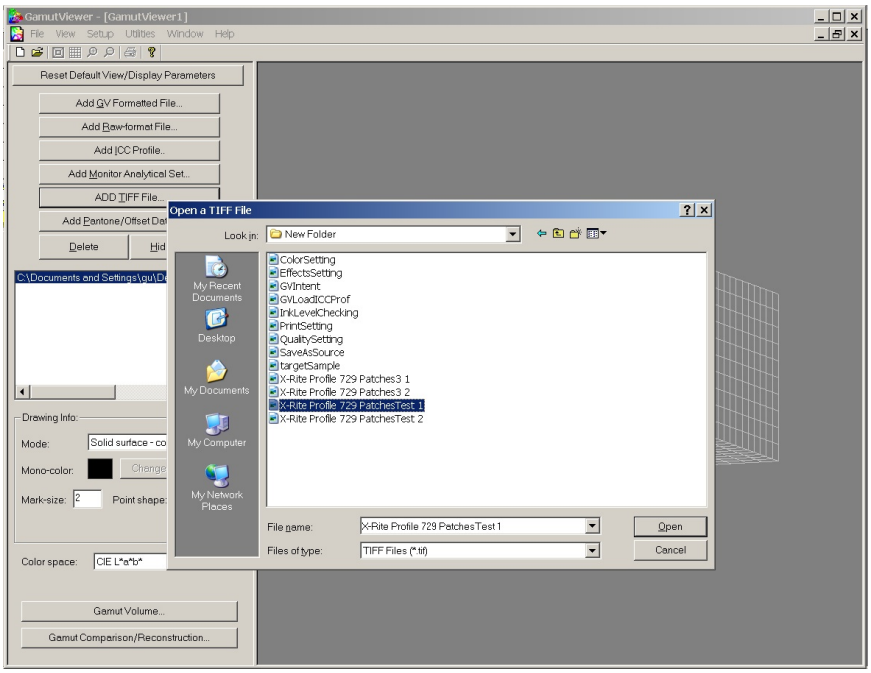
2. Click **Add ICC Profile...**



3. From the dialog box, go to the folder your *.icc or *.icm file is located, and either double click it, or click it and click **Open** in the lower-right.



4. Choose the intent depending on the purpose of comparison.
5. To add a *.tiff gamut, click **ADD TIFF File...**



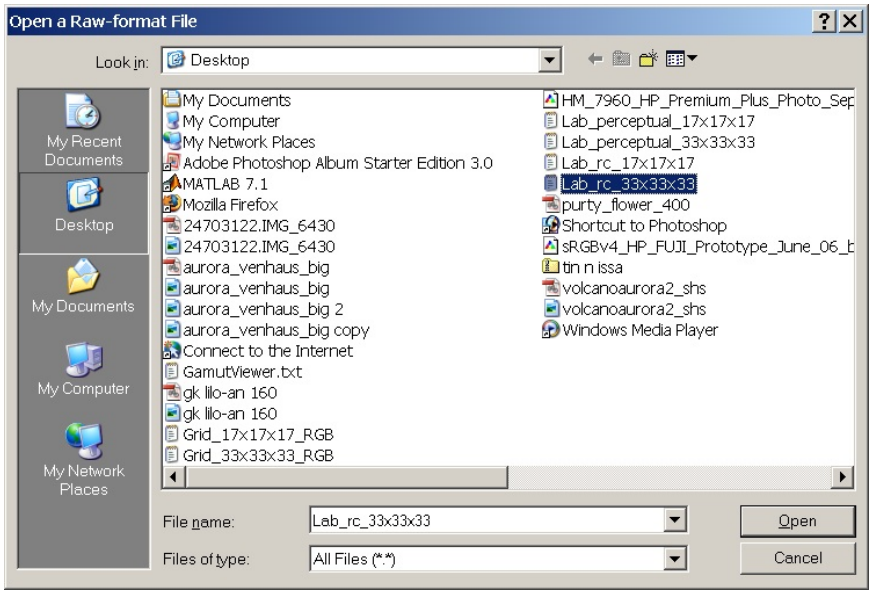
   (a) From the dialog box, go to the folder your *.tif or *.tiff file is located, and either double click it, or click it and click **Open** in the lower-right.
   (b) If no embedded profile is found, click **Ok** to apply the default profile.

(c) Clicking **Cancel** will allow another profile to be applied, instead of the default.

(d) Choose the intent depending on the purpose of comparison.

6. To add raw data from a *.txt file, click **Add Raw-format File...**

(a) From the dialog box, go to the folder your *.txt file is located, and either double click it, or click it and click **Open** in the lower-right.



(b) From the next dialog box, click the appropriate radio button under **File Format**, in this case, **Raw data without total patches information**.

(c) Make sure **Read all patches** is selected, and **This is an ordered data set** is check under **Swatch Info**.

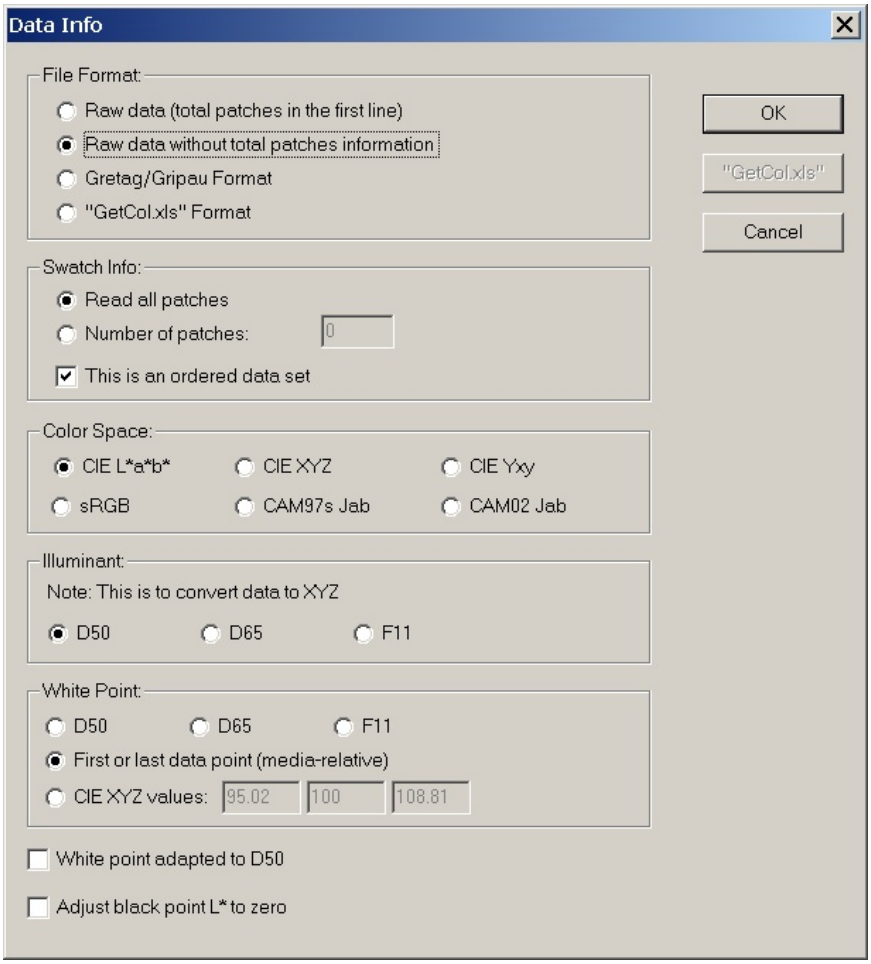
(d) Select the appropriate radio button under **Color Space**, in this case **CIE L*a*b***.

(e) Select the appropriate radio button under **Illuminant**, in this case the default, **D50**.

(f) Select the appropriate radio button under **White Point**, in this case **First or last data point (media-relative)**. In general, one

should choose D50, however, as the other selection requires the first or last data point to be a white measurement. Be careful here!

(g) Check the **Adjust black point L* to zero** as necessary, in this case we will leave it unchecked.



7. To change the drawing info for a gamut to more easily visualize multiple gamuts:

   (a) Make sure the correct gamut is selected in the selection box in the center of the right portion of the screen.

   (b) Click the **Mode** drop-down box and select the desired view mode.

(c) If a mono mode is selected, click **Change mono-color...** and select a color.

8. To compare two gamuts:

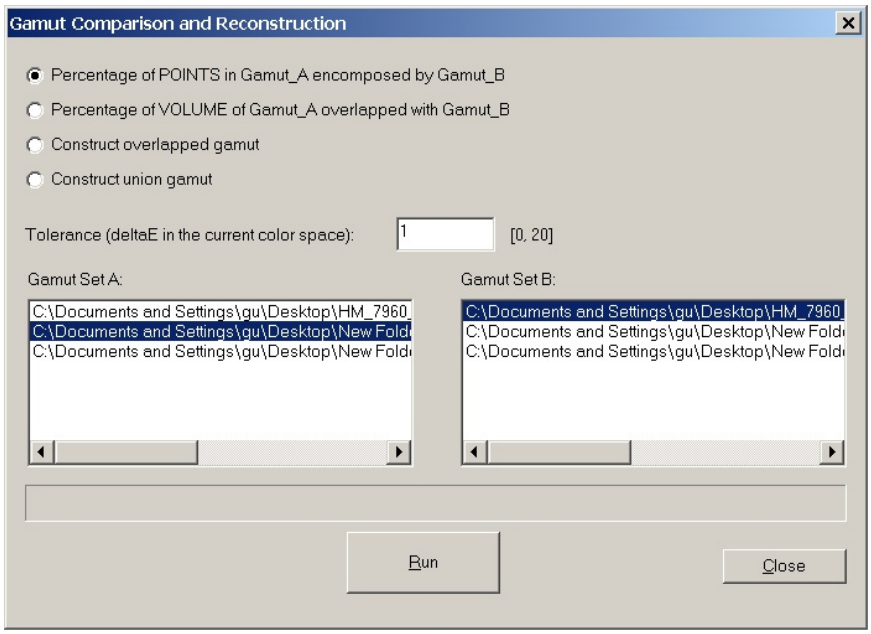
(a) Click **Gamut Comparison/Reconstruction** in the lower-left corner of the window.

(b) Click the **Percentage of POINTS in Gamut_A encomposed by Gamut_B** radio button to generate a percentage of points in Gamut_A that are in Gamut_B.

(c) Click the **Construct overlapped gamut** to generate a visualization of the intersection of Gamut_A and Gamut_B.

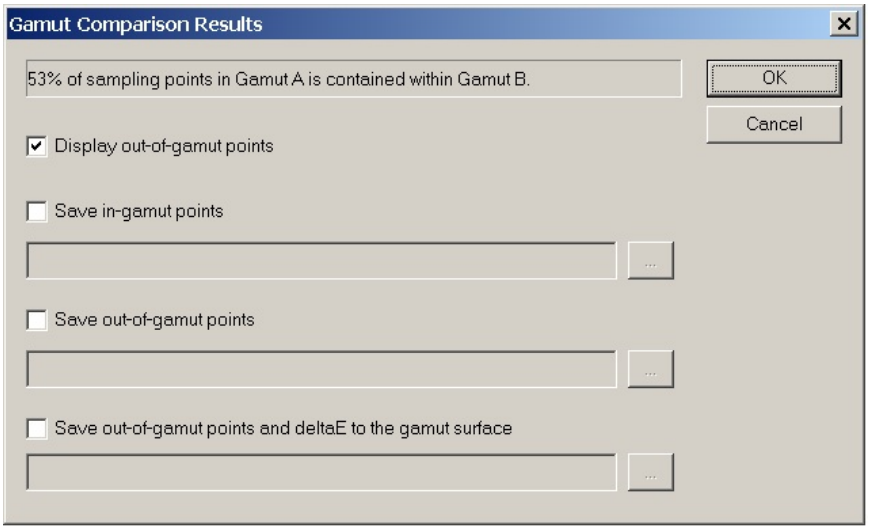
(d) Enter a tolerance in deltaE units of the current space in the **Tolerance** text box.

(e) Select Gamut_A and Gamut_B from their respective selection boxes.



(f) Click **Run**.

(g) When the process is finished for the Percentage of Points, make sure the **Display out-of-gamut points** check box is checked.



(h) Click **OK** to generate a new gamut which is the set of points in Gamut_A that are outside of Gamut_B.

This is useful for finding out what part of an image gamut is outside of a print gamut, for example. The process Construct overlapped gamut is useful for generating the gamut which is the intersection of the two gamuts.

C.4 How To Use an ICC Profile

C.4.1 Installing the Profile

The ICC profile should have the file extension **.icc**. In Windows, you need only to right-click the file and choose **Install Profile**. If this does not work, you can always manually place the ICC profile in the profile folder, located at

`C:\WINDOWS\system32\spool\drivers\color`.

ICC profiles are also located in the Adobe folder, which is located at

`C:\Program Files\Common Files\Adobe\Color\Profiles\Recommended`.

C.4.2 Using the ICC profiles in Photoshop

1. Open up Photoshop and load in an image. Once the image is loaded and visible on the screen, you are able to change the ICC profile with which the picture is rendered.

2. First, let's discuss the difference between assigning a profile to an image and converting an image to a profile. Don't worry about the specifics, we'll go into them immediately afterwards:

3. Each image you load into Photoshop comes with its own RGB values. Photoshop uses the image's embedded profile to change those values to CIE LAB space. Then, Photoshop uses a relative colorimetric transformation to convert those values to sRGB monitor space.

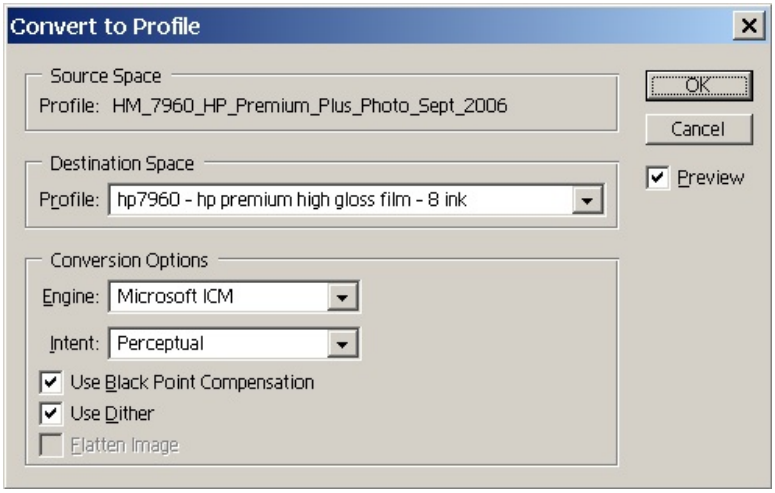
4. By using **Image...Mode...Assign Profile**, you choose the profile to use in the first transformation. Rather than using the embedded profile, Photoshop will then use the assigned profile to make the initial RGB→LAB transformation.

5. If you choose **Image...Mode...Convert To Profile**, the LAB values are actually passed through the chosen profile with whichever intent you have chosen (Perceptual, Relative Colorimetric, etc..).

6. Because the LAB→sRGB transformations are different from the LAB→CMYK transformations, the image on the monitor will not necessarily look like the eventual printed image.

C.4.3 Convert To Profile

7. From the top of the screen, go to **Image...Mode...Convert To Profile**. This will convert the RGB values of the image with the target profile so as to try and keep the intent of the original image.



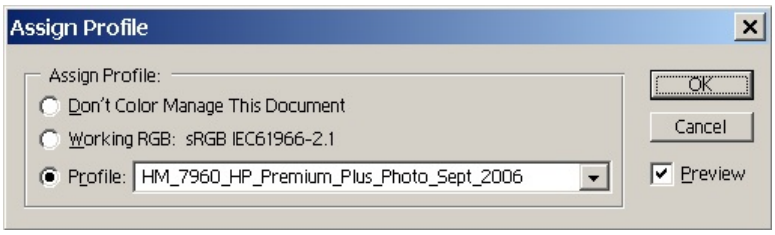
8. Here, you can change the colors of the image according to a profile. Pick from the Profile drop-down arrow the profile you wish to convert to. In the Conversion Options portion of the window, you can choose the intent with which you want to render the image. Make sure the **Preview** box on the right side of the window is checked so you can see the changes you have made. Because the conversion tries to keep the intent of the original image, the visual change may be subtle. To see areas that are actually being changed, it may be necessary to toggle the **Preview** box.

9. Another option in the Conversion Options portion of the window is **Use Black Point Compensation**. What this does is ensure that the colors are scaled correctly. When the black points of the destination and source are different, colors with low luminance will be clipped during the transformation. Black Point Compensation ensures that dark colors and shadows are transformed correctly. If Black Point Compensation is used, there is an extra processing step in the transformation to ensure that the black point of the source profile is correctly mapped to the black point of the target profile. Only then is the overall conversion carried out.

C.4.4 Assign Profile

10. From the top of the screen, go to **Image...Mode...Assign Profile**. This interprets the RGB values associated with the image using a cho-

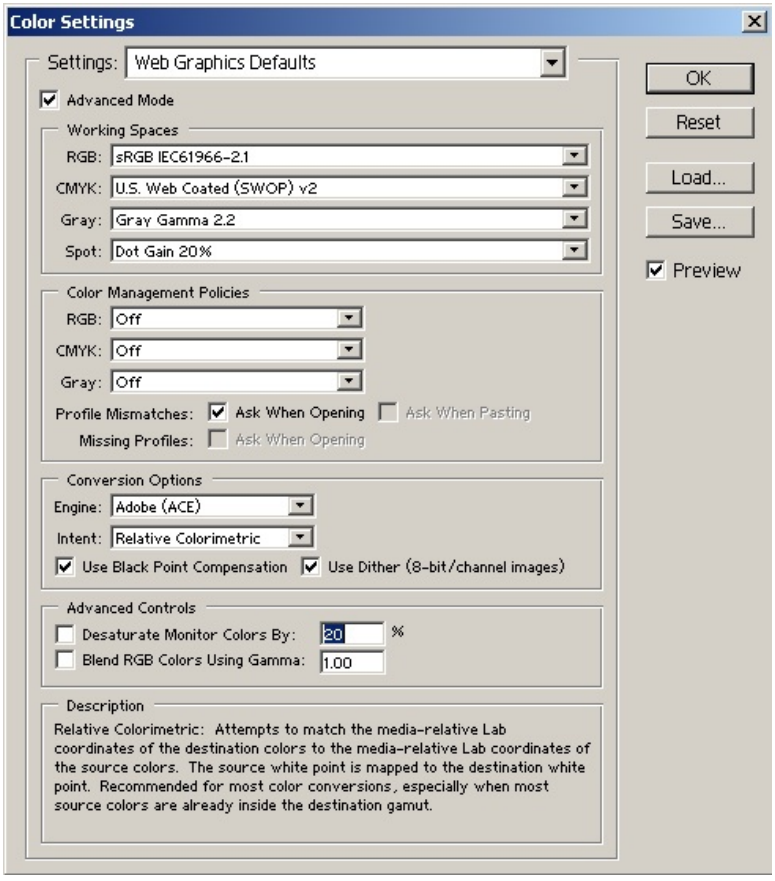
sen profile.



11. In the Assign Profile window, it is possible to check **Don't Color Manage This Document** to see what the image looks like with the default profile. In this case, the original RGB values will be sent directly to the monitor. When dealing with the test charts, this is the option we want to use.

12. Check the **Profile** box. In the drop-down list, you can choose a profile. If the **Preview** box is checked, click Okay and you will see the resulting change in the image immediately.

13. There is another way to Assign a Profile to your image. By going to **Edit...Color Settings**, you can do all the same softproofing as with Assign Profile, but now there are more options to choose from.



14. For instance, in **Image…Mode…Convert To Profile**, you can choose to work in the RGB, CMYK, or other color spaces. In the Color Settings window, you can now change the assigned profile for whichever color space you are using.

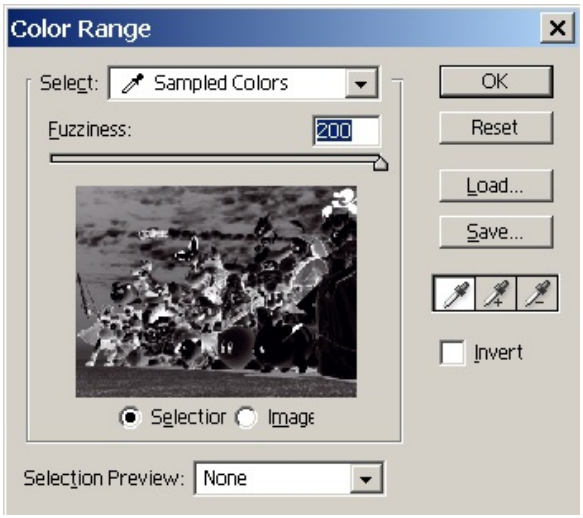
15. In the **Color Management Policies** section of the Color Settings window, you can tell Photoshop what to do when loading an image that already has an embedded profile. Using the drop-down arrow corresponding to whichever color space you wish to work in, you can choose to ignore all color management, keep the original embedded profile, or convert upon opening to the default profile (Note: In most cases, it is best to keep the embedded profile).

16. Under Advanced Controls, you can choose to desaturate or blend the colors on the monitor. This will help ensure that when viewing the

assigned profile on the monitor, there is no clipping that occurs in the conversion to the monitor profile. Keep in mind that this only gives you a good idea of the relative changes in color, and the colors will not be the same as in the printed image.

C.4.5 Color Range

17. Now, you can see more specifically how the profile has changed the image. Right-click on the image and choose **Color Range**.



18. In the Color Range window, the picture in the middle is the image. If the **Selection** circle below the image is checked, the image will be represented in grayscale based on what color is being sampled. Colors in the picture where the sampled color is represented will show up in the image as gray or white, based on how much of the sampled color is present.

19. There are two ways to change the selected color. First, there is the **Select** drop-down arrow in the Color Range window. From there, you can select **Red**, **Yellow**, etc. The image will change to reflect the selected color. The other way to pick the color to be represented is to manually sample the color. Use the eyedropper tool in the image itself to sample a color. The secondary image in the Color Range window will change to reflect the new color.

20. Another helpful feature in the Color Range window is the Out-of-Gamut preview. In **Select**, down at the bottom is **Out of Range**. Choose it to see which colors of the image will not be faithfully reproduced in printing.

Bibliography

B. Azose, G. Heckel, E. Johnson, J. Levin, and I. Win. Work statement: An implementation of new and effective methods of generating ICC profiles. Technical report, Harvey Mudd College, 2005.

Peter G. J. Barten. *Contrast Sensitivity of the Human Eye and its Effects on Image Quality*. SPIE Press, 1999.

International Color Consortium. Frequently asked questions about colour management. `http://www.color.org/faqs.html`, 2006a.

International Color Consortium. Glossary of terms. `http://www.color.org/ICC_white_paper5glossary.pdf`, 2006b.

Mark D. Fairchild. *Color Appearance Models*. Addison Wesley Longman, 1998.

Henry R. Kang. *Color Technology for Electronic Imaging Devices*. Spie Press, Bellingham, Washington, 1997.

Shin-ichiro Kitoh and Po-chieh Hung. A new method to quantitatively evaluate gradation smoothness of output media. *System Solution Technology*, 2005.

Y. Murakami, N. Hatano, J. Takiue, M. Yamaguchi, and N. Ohyama. Evaluation of smooth tonal change reproduction on multi-primary display: Comparison of color conversion algorithms. *Proceedings of SPIE-IS&T Electronic Imaging*, 5289, 2004.

Thor Olson. Smooth ramps: Walking the straight and narrow path through color space. *The Seventh Color Imaging Conference: Color Science, Systems, and Applications*, 1999.

Gaurav Sharma. *Digital Color Imaging Handbook*. CRC Press LCC, 2003.

Jonathan Yen. *Knotty: A B-Spline Visualization Program, Parts I and II*. Morgan Kaufmann, San Francisco, 1993.