Color Management Experiments using Adaptive Neighborhoods for Local Regression

Erika Chin, University of Virginia, Charlottesville, VA
at University of Washington for DMP, Summer 2006

Abstract

We built 3-D and 1-D look up tables (LUTs) to transform a user's desired device-independent colors (CIELab) to the device-dependent color space (RGB). We considered experimental adaptive neighborhood and estimation methods for building the 3-D and 1-D LUTs. Methods of finding neighborhoods include: smallest enclosing neighborhood (SEN), smallest enclosing inclusive neighborhood (SENR), natural neighbors neighborhood (NN), natural neighbors inclusive neighborhood (NNR), and 15-nearest neighbors. The estimation techniques investigated were: local linear regression, ridge regression, and linear interpolation with maximum entropy (LIME) weighted regression.

Three printers were tested using combinations of the five neighborhood definitions (SEN, SENR, NN, NNR, and 15-nearest neighbors) and three regression techniques (local linear regression, ridge regression, and LIME weighted regression).

Index Terms

inverse color management, ICC profiles, adaptive neighborhoods, enclosing neighborhoods, smallest enclosing neighborhood, natural neighbors, convex hull, local linear regression, ridge regression, linear interpolation with maximum entropy (LIME) weighted regression

I. Introduction

INVERSE color management is the transformation from the user-desired colors (CIELab) to the device-dependent color space (RGB). It allows users to specify a color they desire and find what color input must be passed to the printer in order for the printer to print the desired color. Due to the varying nonlinear color response of every printer, inverse color management and printer characterization has always been a challenging problem. In this paper, we explored a lattice-based empirical method for inverse device characterization [1]. Color management in printers is controlled by a color management module (CMM) that uses a color management standard (created by the International Color Consortium) called ICC profiles. The main components of the ICC profiles are a 3-D LUT and three 1-D LUTs which divide the printer characterization process into two separate stages. The first stage, passing input CIELab values through the 3-D LUT, works toward characterizing the printer. The second stage, passing the values obtained from the 3-D LUT through the 1-D LUT, is the calibration stage which works toward linearizing the printer and ensuring that the neutrals remain in the neutral area. The 3-D LUT represents a grid that spans the CIELab space and maps each point to an RGB value. Given a CIELab value, the value can be looked up in the 3-D grid (or interpolated, if the point lies between grid points) and then calibrated with the 1-D LUT.

Instead of measuring all possible RGB values ($256 \times 256 \times 256$), the current commercial standard is to estimate the LUTs from a small set of training data. Bala showed that local linear regression works well with limited training data [1]. A small set of training data presents the problems of how to find local neighborhoods with which to estimate and once the neighborhoods are found, which estimation techniques to use. Estimation techniques greatly depend on the size of a neighborhood. Different results are yielded when the estimation neighborhood is too small, too large, or not close enough to the test point. It is also important to consider the effect of a test point that falls outside of the convex hull of all training points. In this paper, we explore different adaptive neighborhood definitions and estimation methods to build the 1-D and 3-D LUTs for ICC profiles.

II. Background and Process of Device Calibration and Characterization

To color manage a printer, one must first model the transformation from the device-dependent color space (RGB) to the device-independent color space (CIELab). This process can be separated into two main stages. The first stage, calibration, works toward linearizing the printer [1]. The second stage characterizes the calibrated printer. Using the calibrated RGB values and CIELab pairs, one can create a 3-D LUT that will translate CIELab to RGB. Although the calibration of the printer is performed after the characterization function is applied, to derive the calibration LUT, the 1-D table is built before characterization [3]. We will discuss the calibration method first.

A. Calibration: 1-D LUTs

The problem with uncalibrated printers is that when sending the printer a gray valued RGB (where $R = G = B = d$), one would expect the printer to print a gray value that is equivalent to CIELab (100(d/255), 0, 0). Instead, the printed value may not necessarily be neutral. Calibration fixes this problem.

We used the gray-balanced calibration method [1].

Steps to create the 1-D calibration LUT:

1) Print an RGB color chart.
2) Measure the printed colors in CIELab with a spectrophotometer to get (RGB, CIELab) pairs. The (RGB, CIELab) pairs make up the training set (for building the 1-D LUTs).
3) Find the range of the measured luminance values. This will become the range of L^* values for the test set. Divide the range into even increments of (maximum luminance - minimum luminance)/255. This list of (L^*, 0, 0) represents the incremental shades of gray in CIELab space (between the minimum and maximum producible luminance values measured in the training patch) that make up the test set.
4) Use this training and test data to estimate the R'G'B' values for each L^*: For each point in the test set, find its neighborhood of training points (using the same neighbor algorithm that will be used for the 3-D characterization). Then use local linear, ridge, or LIME weighted regression to get the R'G'B' value that is expected to be equivalent to (L^*, 0, 0).
5) The LUTs can then be built. Because we want $R = G = B = 255(L^*/100)$, $255(L^*/100)$ is the input to each 1-D LUT, and the output is the derived R' (or G' or B') value.

When you input (d, d, d) in RGB, the 1-D LUTs will convert it to R'G'B' values that correspond to (100(d/255), 0,0) in CIELab space. See Figure 1.

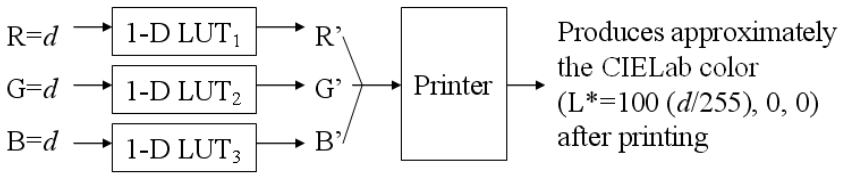


Fig. 1. Effect of the 1-D LUTs: Pass the LUTs $R = G = B = d$ and the LUTs will return the R'G'B' that produces a gray-tone when sent to the printer.

B. Characterization: 3-D LUT

Once the 1-D LUTs are built you can then proceed to use these LUTs to create the 3-D LUT. First take the inverse of the 1-D LUTs (using 1-D linear interpolation) and use these inverse tables to look up the RGB values in the training pairs to get new (adjusted RGB, CIELab). Then use these new training pairs and use the gridpoints as test points to built the 3-D LUT.

{Side note: Inverting the 1-D LUT works best when the values are monotonic and the range of the 1-D LUT is equal to the domain of the inverse 1-D LUT. If the 1-D LUTs' R'G'B' values do not span 0

to 255, when taking the inverse of the 1-D LUT, for any estimate below the minimum value in the 1-D LUT, set the inverse LUT value to the R (or G or B) that derives the minimum R' (or G' or B') and for any estimate above the maximum value, set it to the R (or G or B) that derives the maximum R' (or G' or B').}

The end result is a 3-D LUT and three 1-D LUTs that when used together will color manage a printer. See Figure 2.

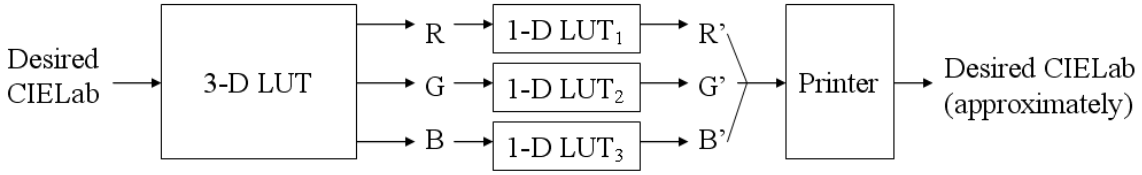


Fig. 2. Inverse color management: To determine what RGB value to pass to a printer to get the desired value, start with the desired value in the CIELab device-independent color space. Then use the 3-D LUT to get the RGB value. This is then passed through the 1-D LUTs to get the new R'G'B' value.

III. Definitions of Adaptive Neigborhoods

In order to estimate the RGB value of a grid point in the 3-D LUT (or 1-D LUT), we must define the local neighborhood for each test point. We consider five different methods for deriving the neighborhood of a point: smallest enclosing neighborhood (SEN), smallest enclosing inclusive neighborhood (SENR), natural neighbors (NN), natural neighbors inclusive neighborhood (NN), and 15-nearest neighbors (15).

SEN, SENR, NN, and NNR find neighborhoods that try to enclose the test point in the convex hull of the neighborhood. This can potentially improve an estimate because it tries to give training information from every side of the test point, and thus more balanced information, whereas with 15-nearest neighbors, one may get training data only on one side of the point.

A. Smallest Enclosing Neighborhood (SEN)

Smallest enclosing neighborhood is an adaptive neighborhood that chooses the next nearest neighbor only if it reduces the test point's distance to the convex hull created by the neighborhood with the goal of having the test point within the convex hull [4].

The smallest enclosing neighborhood finds the smallest (if possible) set of neighbors where the test point can be represented by the convex combination of the neighbors such that:

$$\sum_j w_j x_j = x \tag{1}$$

where x is the test point, x_j is the neighbor, w_j is the weight, and $\sum_j w_j = 1$.

Given a list of training points sorted by distance to the test point, consider each training point sequentially, and add the point to the neighborhood only if by adding this point, the nonconvexity of the neighborhood (the distance from the test point to the facet of the convex hull created by the neighborhood points) is reduced.

Steps to find the smallest enclosing neighborhood:

1) Sort the training points in order of distance to the test point.
2) Add the closest training point to the neighborhood. Call this x_1.
3) Draw a hyperplane normal to the unit vector from the test point v_1 to x_1 and consider only the points that are on the same side of the hyperplane as the test point. To do this:
   a) Find the unit vector v_1 from the test point to this neighbor x_1.
   b) Project all other training points (that are not included in the neighborhood) onto this vector by taking the dot product of the unit vector (v_1) with each training point not in the neighborhood.

For the projections that are less than the distance from the test point to x_1, those points are on the desired side of the hyperplane.

4) Define a value, $custeps$, to be the distance from the test point to the closest facet of the convex hull of the neighborhood. This value will determine if a test point is close enough to the convex hull for the algorithm to be satisfied and stop iterating. The distance from the test point to the closest facet of the convex hull of the neighborhood is also called the nonconvexity of the neighborhood.
5) Considering only the points on the close side of the hyperplane, add the new closest training point to the neighborhood.
6) Find the nonconvexity of the neighborhood. If the nonconvexity is less than $custeps$, stop iterating. This will be the neighborhood.
7) Use the hyperplane created to eliminate the training points on the far side from consideration.
8) Add the new closest training point to the neighborhood.
9) Repeat steps 6-9 until the test point is in the convex hull or cannot get any closer to the convex hull by adding any more neighbors. If the test point is outside of the hull of the training points, eventually the hyperplane will eliminate all other training points and there will be nothing to add to the neighborhood that will help the point get closer to the convex hull. Stop iterating and take this as the neighborhood.

Make sure that the number of neighbors is at least $d+1$ where d is the dimension of the data. If it is less than $d+1$, then add to the neighborhood the next closest training points until there are $d+1$ points in the neighborhood.

We used two variations of SEN and SENR. In one version, we required the neighborhood to have a minimum of four neighbors (the fewest number of points we can have and still perform a regression). If there were less than four neighbors, then the training point closest to the test point but not previously included in the neighborhood was added to the neighborhood. This was repeated until there were at least 4 neighbors. Tests using this algorithm were called SEN4 and SEN4R. The other version of SEN and SENR required the neighborhood to have a minimum of fifteen neighbors. These tests were called SEN15 and SEN15R.

B. Smallest Enclosing Inclusive Neighborhood (SENR)

Smallest enclosing inclusive neighborhood (SENR) is an adaptive neighborhood that chooses neighbors as defined by SEN and then includes all other training points that are within the radial sphere created by the farthest SEN point. In other words, given the set of smallest enclosing neighborhood indices $\mathcal{J}_n$, include training point x_j in the neighborhood if

$$\|g - x_j\|_2 \leq \max_{i \in \mathcal{J}_n} \|g - x_i\|_2. \tag{2}$$

where g is the test point.

C. Sibson's Natural Neighbors Neighborhood (NN)

Sibson's natural neighbors neighborhood (NN) is an adaptive neighborhood that generates the Voronoi tessellation (where every location in space is assigned to the closest point in the training set or the test point) and uses it to determine the boundary of each point's region, marked by the areas that are equally close to two or more training or test points. The training points whose regions are adjacent to the test point's region are the points that make up the neighborhood [5]. On average, if a point is within the convex hull of the training points, the neighborhood size is expected to be around 15. See Figure 3.

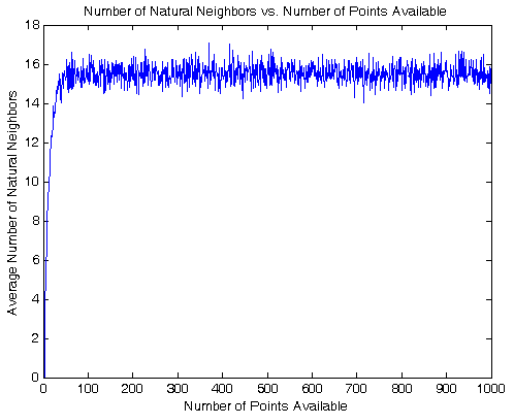


Fig. 3. Graph of the number of natural neighbors of a point (located at the origin) given a randomly generated set of points available. For each number of available points, 50 random sets were used and the number of natural neighbors was averaged over 50. As the number of available training points increased, the size of the natural neighbors neighborhood for a test point asymptotically converged to 15.

D. Sibson's Natural Neighbors Inclusive Neighborhood (NNR)

Sibson's natural neighbors inclusive neighborhood (NNR) is an enclosing, adaptive neighborhood that finds the natural neighbors and includes all other training points that are within the radial sphere created by the farthest natural neighbor points. In other words, given the set of natural neighbors neighborhood indices $\mathcal{J}_n$, include training point x_j in the neighborhood if

$$\|g - x_j\|_2 \leq \max_{i \in \mathcal{J}_n} \|g - x_i\|_2. \tag{3}$$

where g is the test point.

E. 15-Nearest Neighbors (15)

15-nearest neighbors (15) thats the 15 training points closest to the test point. Bala recommends 15 neighbors as a sufficient neighborhood size for estimation [3].

IV. Estimation Techniques

Three regression techniques were used to estimate the CIELab values at each gridpoint in the 3-D LUT: local linear regression, ridge regression, and LIME weighted regression. All regression techniques estimate R, G, and B independently.

A. Local Linear Regression

Local linear regression fits the least-squared error hyperplane to some local neighborhood of each gridpoint of the 3-D LUT. For a gridpoint $g \in Lab$, local linear regression finds the β_1, ε_1, β_2, ε_2, β_3, and ε_3 such that

$$\begin{aligned} \hat{R} &= {\beta_1}^T g + \varepsilon_1 \\ \hat{G} &= {\beta_2}^T g + \varepsilon_2 \\ \hat{B} &= {\beta_3}^T g + \varepsilon_3, \end{aligned} \tag{4}$$

where β_1, ε_1 are regression coefficients of a least-squares hyperplane fit to the neighborhood samples $\{x_j, R_j\}$, β_2, ε_2 are the vector of regression coefficients of a least-squares hyperplane fit to the neighborhood samples $\{x_j, G_j\}$, and β_3, ε_3 are the vector of regression coefficients of a least-squares hyperplane fit

to the neighborhood samples $\{x_j, B_j\}$. The algorithm minimizes the mean squared error (for each channel separately):

$$[\beta_1,\ \varepsilon_1] = \arg\min_{f_1(x)} \sum_j \left(f_1(x_j) - R_j\right)^2 \tag{5}$$

$$[\beta_2,\ \varepsilon_2] = \arg\min_{f_2(x)} \sum_j \left(f_2(x_j) - G_j\right)^2$$

$$[\beta_3,\ \varepsilon_3] = \arg\min_{f_3(x)} \sum_j \left(f_3(x_j) - B_j\right)^2 \tag{6}$$

where $x_j[1]$, $x_j[2]$, and $x_j[3]$ are the L^*, a^*, and b^* channels for neighbor x_j; $f_1(x_j)$ is the estimated value $\hat{R}$ and R_j is the measured value of R for neighbor j; $f_2(x_j)$ is the estimated value $\hat{G}$ and G_j is the measured value of G for neighbor j; $f_3(x_j)$ is the estimated value $\hat{B}$ and B_j is the measured value of B for neighbor j.

B. Ridge Regression

Ridge regression is similar to local linear regression, but it applies a penalty to solutions whose estimates are too steep. This effectively shrinks the coefficients of the regression so that the square of the linear regression coefficients is minimized:

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{7}$$

where a larger λ results in more stabilized coefficients [6].

C. LIME Weighted Regression

The goal of the LIME algorithm is to use linear interpolation and pick weights on the values to preserve the maximum entropy. Simple linear interpolation can yield multiple solutions that assign different weights to surrounding neighbors but still fall within the parameters given. The solution is not unique. With maximum entropy, one tries to choose the solution that will yield the most uniform weights. This is represented by

$$\arg\min_{w} \left\| \sum_{j=1}^{k} w_j x_j - x \right\|_2^2 + \lambda H(w) \tag{8}$$

where a large λ results in more evenly weighted neighbors and $H(w)$ represents the Shannon entropy of w [4].

The weights for each neighbor are then used to weight the errors when fitting a hyperplane that has the minimum total weighted squared error. Then, the fitted hyperplanes (from ridge regression) are used to estimate the R, G, and B values.

V. EXPERIMENTS

Experiments were run on three printers: Epson Stylus Photo 2200 (ink jet), Epson Stylus Photo R300 (ink jet), and Ricoh Aficio 1232C (laser). The Epson Stylus Photo 2200 experiments were run using Epson ink on Epson Matte Heavyweight Paper. The Epson Stylus Photo R300 experiments were run using Premium Imaging Products ink (non-Epson) and on Epson Matte Heavyweight Paper. The Ricoh Aficio 1232C experiments were run on Office Depot (generic) laser copy paper.

For each experiment: The training data, (RGB, CIELab) pairs, were obtained from a Chromix color chart consisting of 918 color patches. The RGB value for the Chromix chart patches were read from

the image. Then the GretagMacbeth Spectrolino spectrophotometer was used to measure the patches and obtain the CIELab values. All CIELab reflectance measurements were taken by the spectrophotometer without a filter and at a 2° observer angle and with D50 illumination.

Then the test data set was built. A $17 \times 17 \times 17$ grid that spanned CIELab color space was used. Bala showed that a grid more finely sampled than $16 \times 16 \times 16$ does not afford any noticeable improvement in accuracy [1]. The $17 \times 17 \times 17$ grid in CIELab space where $L \in [0, 100]$, $a \in [-100, 100]$, and $b \in [-100, 100]$. For each test point, the training data was used to find a local neighborhood for the test point. Using this local neighborhood, we used regression to estimate the RGB value of the test point. For estimation, the ridge penalty was set to 0.1 and for LIME, the lambda variable was set to 100,000.

Once the LUTs were built, we measured the transformation performance. We chose ΔE^*_{94} error [1] as the metric for evaluating the device characterization:

$$\Delta E^*_{94} = \sqrt{\frac{\Delta L^*}{K_L S_L}^2 + \frac{\Delta C^*}{K_C S_C}^2 + \frac{\Delta H^*}{K_H S_H}^2} \tag{9}$$

$$\begin{aligned}
\Delta L^* &= L^*_1 - L^*_2 \\
\Delta C^* &= C^*_1 - C^*_2 \\
C^*_1 &= \sqrt{a^{*2}_1 + b^{*2}_1} \\
C^*_2 &= \sqrt{a^{*2}_2 + b^{*2}_2} \\
\Delta H^* &= \sqrt{\Delta a^{*2} + \Delta b^{*2} + \Delta C^{*2}} \\
\Delta a^* &= a^*_1 - a^*_2 \\
\Delta b^* &= b^*_1 - b^*_2 \\
S_L &= 1 \\
S_C &= 1 + K_1 C_1 \\
S_H &= 1 + K_2 C_1 \\
K_1 &= 0.045 \\
K_2 &= .015
\end{aligned} \tag{10}$$

We chose to measure error in terms of CIELab error because RGB is a device-dependent color space and a change in the R channel may not be equal to a change of the same size in the G channel and yet the RGB error of these different points could yield the same value. CIELab error gives a more perceptually meaningful value[2].

To find the error, we first generated new test data. We considered only colors that are obtainable by the printer (i.e within its gamut). Therefore, to ensure that all randomly generated CIELab test error points are within gamut, we generated the random points in RGB and then sent them to the printer and measured the printed patches in CIELab. Error test data consisted of 918 patches. Of the patches, 729 patches were randomly generated across the RGB color space ($R \in [0, 255]$, $G \in [0, 255]$, and $B \in [0, 255]$) and the remaining 189 patches were randomly generated across the near-neutral colors of the color space (color whose R, G, and B values were within 25 units of each other). Error of near-neutral values is important because the human eye is more sensitive to these areas. The error page was measured in CIELab and the 3-D and 1-D LUTs were used to transform the test CIELab values to RGB colors. The test patches were printed (with all software-accessible color management turned off), measured, and evaluated. 15 linear was considered the baseline for the experiments.

VI. Results and Comments

A. Neighborhood Size

On average the SEN4 neighborhood sizes range from 4 to 30. The SEN4R neighborhood sizes range from 4 to 60. SEN15 has 15 to 40 neighbors, while SEN15R has 15 to 150 neighbors. Natural neighbors can have around 7 to 160 and NNR can have 8 to 918 neighbors. 15 nearest neighbors will always have a neighborhood size of 15. See Figure 4.

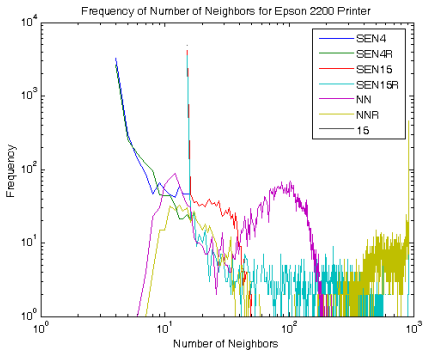


Fig. 4. Frequency of neighborhood sizes for the different neighborhood methods for the Epson Photo Stylus 2200

B. Error

We looked at average error in the 189 neutral patches, average error in the 729 nonneutral patches, maximum error in the neutral patches, maximum error in the nonneutral patches, and overall average error of all patches. See Tables I, II, and III.

TABLE I
Errors from the Epson Photo Stylus 2200

Neighborhood method	Estimation	Neutral Avg. Error	Nonneutral Avg. Error	Neutral Max. Error	Nonneutral Max. Error	Overall Avg. Error
SEN4	Linear	2.10	3.10	5.27	27.63	2.89
	Ridge	2.03	2.28	4.00	12.91	2.23
SEN4R	Linear	1.84	2.33	4.29	25.77	2.23
	Ridge	1.78	2.23	4.19	9.19	2.14
SEN15	Linear	2.00	2.38	4.52	10.09	2.30
	Ridge	1.84	2.16	3.74	8.00	2.10
SEN15R	Linear	1.93	2.34	4.06	7.84	2.26
	Ridge	1.91	2.32	4.02	7.75	2.23
NN	Linear	1.79	2.41	3.90	9.82	2.29
	Ridge	1.60	2.20	3.72	9.25	2.07
NNR	Linear	1.61	2.46	3.80	8.02	2.28
	Ridge	1.42	2.46	3.23	8.21	2.25
15	Linear	1.56	2.43	3.76	14.60	2.25
	Ridge	1.65	2.43	3.81	13.76	2.27
	LIME	1.37	2.40	3.31	14.22	2.19

We also categorized errors into level of perceivable differences (subjective) where errors from 0-4=Unperceivable, 4.01-8=Slightly Perceivable, 8.01-13=A Shade Different, 13+=Different Color. See Figure 5.

TABLE II
ERRORS FROM THE EPSON PHOTO STYLUS R300

Neighborhood method	Estimation	Neutral Avg. Error	Nonneutral Avg. Error	Neutral Max. Error	Nonneutral Max. Error	Overall Avg. Error
SEN4	Linear	1.73	2.38	12.28	22.25	2.25
	Ridge	1.18	1.79	2.85	5.72	1.67
SEN4R	Linear	1.02	1.67	2.71	20.50	1.54
	Ridge	0.92	1.55	2.59	5.98	1.42
SEN15	Linear	1.42	1.71	4.78	5.77	1.65
	Ridge	0.99	1.64	2.51	4.94	1.51
SEN15R	Linear	0.95	1.51	2.46	4.19	1.40
	Ridge	0.89	1.52	2.60	4.72	1.39
NN	Linear	1.01	1.71	2.70	5.28	1.56
	Ridge	1.01	1.54	2.61	4.26	1.43
NNR	Linear	1.15	1.77	2.74	5.65	1.65
	Ridge	1.12	1.79	2.93	5.65	1.65
15	Linear	1.02	1.55	2.46	4.93	1.44
	Ridge	1.00	1.55	2.45	4.98	1.43
	LIME	0.97	1.53	2.51	4.92	1.42

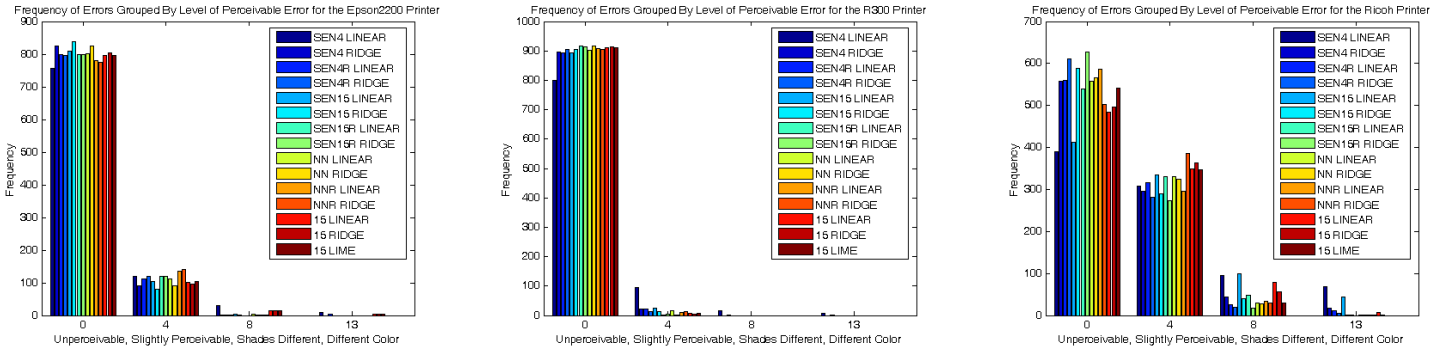
TABLE III
ERRORS FROM THE RICOH AFICIO 1232C

Neighborhood method	Estimation	Neutral Avg. Error	Nonneutral Avg. Error	Neutral Max. Error	Nonneutral Max. Error	Overall Avg. Error
SEN4	Linear	3.82	8.98	22.01	55.25	7.92
	Ridge	3.24	4.33	16.50	31.94	4.10
SEN4R	Linear	3.44	4.27	16.13	53.83	4.10
	Ridge	3.04	3.66	17.32	20.63	3.53
SEN15	Linear	3.94	6.93	16.61	45.13	6.32
	Ridge	3.31	3.84	14.73	11.31	3.73
SEN15R	Linear	3.70	4.03	13.53	12.43	3.96
	Ridge	3.25	3.45	12.33	10.78	3.41
NN	Linear	3.83	3.74	14.83	9.96	3.76
	Ridge	3.78	3.69	15.48	10.23	3.71
NNR	Linear	3.28	3.74	16.04	12.96	3.65
	Ridge	3.97	4.03	14.41	12.11	4.02
15	Linear	4.40	4.41	14.83	17.66	4.41
	Ridge	4.21	4.16	15.16	13.00	4.17
	LIME	3.95	3.75	12.74	11.02	3.79

Overall, the ink jet printers were characterized more accurately than the laser printer. For all methods, the best results were yielded on the Epson R300. The characterization of the Ricoh had more variability. This indicates that the Ricoh printer was more nonlinear than the Epson printers.

For the Epson 2200, SEN15 ridge, SEN4 ridge, NN ridge, and SEN15 linear performed well. 15 nearest neighbors with linear, ridge, and LIME consistently had multiple color differences.

For the Epson R300, SEN15R linear performed the best with 917 unperceivable differences and one slightly perceivable difference. SEN4 and SEN4R linear were the only methods that had color or shade differences. Methods on the R300 are ranked from best performance to worst as follows: SEN15R linear,



(a) Epson Color Stylus 2200 (b) Epson Color Stylus R300 (c) Ricoh Aficio 1232C

Fig. 5. This graph show the frequency of errors grouped by the amount of error for the Epson 2200 (left), Epson R300 (middle), and Ricoh Aficio (right) printers. For errors 0-4=Unperceivable, 4.01-8=Slighly Perceivable, 8.01-13=A Shade Different, 13+=Different Color. Better performance is shown by high frequencies in the unperceivable differences, lower frequencies in the slightly perceivable differences, and no color differences.

NN ridge, 15 ridge, SEN15R ridge, 15 linear, 15 LIME, NNR linear, SEN4R ridge, SEN15 ridge, NNR ridge, NN linear, SEN4 ridge, SEN15 Linear, SEN4R linear, SEN4 linear.

For the Ricoh printer, SEN15 ridge performed the best. It had the most unperceivable color differences, the least perceivable and shade differences and no different colors. All SEN4 and SEN4R neighborhoods performed around or below the baseline in terms of number of totally different colors. (SEN4R ridge performed reasonably well in terms of unperceivable difference, but has a few different colors.) SEN15 ridge and NNR linear performed well above the baseline. NN ridge, NN linear, and 15 LIME are average.

Overall, SEN4 linear and SEN15 linear consistently performed poorly. In all printers, SEN4R linear had a few color differences. In overall errors, NN linear, 15 linear, and 15 ridge performed poorly. 15 LIME, SEN4R ridge, SEN15R ridge and NN ridge performed well. SEN15R linear and SEN4 ridge had a high performance variability. NNR linear and NNR ridge performs below the 15 linear baseline.

In general, medium small neighborhoods ($\tilde{15}$) tend to make closer estimates, while very small neighborhoods or very large neighborhoods tend to result in bad estimates (unless tempered by ridge regression). Mostly, ridge regression performs better than its linear regression counterpart. Error on small neighborhoods (SEN4) can be improved by taking its radial neighborhood (SEN4R) instead.

VII. Related Issues

LIME weighted regression with 15 nearest neighbors seems to consistently do reasonably well. As the LIME algorithm is specifically targeted to do well with enclosed neighborhoods, it would be interesting to see more about how LIME does with the other neighborhoods, like SEN15 or SEN4R. Currently LIME's delta value for balancing the weights is at 316.2 ($= \sqrt{100,000}$). It would be interesting to see whether lowering the delta (and making the weights less even) would result in a better estimate.

Due to the concave curvature of the border of the printer gamut, it is possible for SEN to keep on taking training points along the border, no matter how far the points are from the test point, if the test point is outside of the convex hull of the training points. To avoid these large neighborhoods, it might be good to experiment with a version of SEN that takes the change of the nonconvexity into account when adding a point into the neighborhood. If the nonconvexity only decreases by a small amount, it may not be worth looking for more points in the neighborhood.

VIII. Conclusion and Future Work

In general, all enclosing neighborhood methods (SEN4, SEN4R, SEN15, SEN15R, NN, and NNR) result in close estimates when the test point is well within the gamut of the printer. It is when a test point

is outside of the convex hull of the neighbors that interesting estimates can be found. For future work, one could consider the in- and out-of-gamut issues separately and find which combination of estimation techniques (neighborhood method and regression) leads to a closer estimate when the test point is fully within the convex hull of its training points and find which combination yields a better estimate when the test point is outside of the convex hull (and probably near the boundary of the gamut). Then, by considering whether a method can find a neighborhood whose convex hull encases the test point, one can choose which method to use for estimation. Alternatively, one can simply find which situation, in- or out-of-gamut, affects the overall error more, and choose the method that caters towards fixing that gamut issue. I suspect that one of the enclosing neighborhoods and LIME weighted regression would estimate points within the gamut accurately. 15 nearest neighbors and ridge regression may work for points around the gamut boundary.

We explored different methods of finding neighborhoods and estimating LUTs for inverse color management. Of the different methods, we found 15 LIME, SEN4R ridge, SEN15R ridge and NN ridge to perform well on average. In general, ridge regression on neighborhoods of around 15 tend to produce reasonable transformations.

ACKNOWLEDGMENTS

I would like to thank my mentor, Maya Gupta at the University of Washington for continued advice and support on my research. I would also like to thank the Computer Research Association's Committee on the Status of Women in Computing Research (CRA-W), the Distributed Mentor Project (DMP), and University of Washington for the opportunity to come out to Seattle for research. Thanks also to Chromix and Xerox for their advice.

REFERENCES

[1] G. Sharma, *Digital Color Imaging*. Boca Raton, FL, United States of America: CRC Press LLC, 2003, ch. 5.
[2] M. R. Gupta, "Inverting color transforms."
[3] M. R. Gupta and R. Bala, "Personal communication with Raja Bala," 2006.
[4] M. R. Gupta, E. Garcia, and A. Stroilov, "Learning ICC profiles for custom color enhancements with adaptive neighborhoods," *IEEE Transactions on Image Processing*.
[5] A. Okabe, B. Boots, K. Sugihara, and S. Chiu, *Spatial Tessellations*. Chichester, England: John Wiley and Sons, Ltd., 2000, ch. 6, p. 418.
[6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY: Springer, 2001, ch. 2.
[7] M. R. Gupta, R. Gray, and R. Olshen, "Nonparametric supervised learning by linear interpolation with maximum entropy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 5, May 2006.

Erika Chin is a fourth year undergraduate at the University of Virginia, Charlottesville, VA. She will be graduating from the UVA School of Engineering and Applied Sciences in 2007 and is expecting a BS in Computer Science and a minor in Engineering Business.