Representation of Color Space Transformations for Effective Calibration and Control

Richard E. Groff Daniel E. Koditschek Pramod P. Khargonekar
EECS, University of Michigan, Ann Arbor, Michigan USA
Tracy E. Thieret
J.C. Wilson Center for Research and Technology, Xerox Corporation
Webster, New York USA

Abstract

We propose the "minvar" algorithm for computing continuous, continuously invertible, piecewise linear (PL) approximations of color space transformations that can serve as functional replacements wherever look-up tables are presently used. After motivating the importance of invertible approximants in color space management applications, we review the parameterization and computational implementation of PL functions as representing one useful instance of this notion. Finally, we describe the present version of the minvar algorithm and compare the approximations it yields with standard industrial practice — interpolation of look-up table data.

1. Introduction

In color image reproduction, a device must determine the recipe for mixing the available colorants to form each desired output color within its gamut. This recipe may be modeled as a transformation from the input (colorant) space to the output (measurement) space. Since this recipe function is affected by a variety of disturbances, we seek to perform real-time control on the system to stabilize color reproduction. For this purpose, it is useful to have access to an effectively computable approximation to both the transformation and its inverse. The term "effectively computable" means that the approximation should be both numerically simple and parsimoniously parameterized, which would ensure that the control action could be imposed in a timely manner using a small number of data patches. In this paper, we explore the class of piecewise linear (PL) functions as affording an effectively computable approximation of the color space transformation and its inverse.

All computable functions are defined by some finite dimensional parameter space. A continuous, continuously (and closed-form) invertible PL function on a multi-dimensional space is parameterized by a set of knot points and a connectivity. The knot points are defined by pairs of "vertices" in the domain (input) and codomain (output) spaces. The connectivity defines a triangulation (of the domain and codomain) with the knots serving as vertices — triangles for a two dimension space; tetrahedra for a 3d space; and, more generally, n-simplices for an n-dimensional space. The resulting PL approximant is invertible if the triangulation has no "tangles" — knots whose domain vertices have codomain images contained within the interior of codomain cells for which they do not serve as vertices. A PL function is computed by first determining in which domain cell the input lies, and then multiplying that input by a matrix constructed from the knots that govern that cell. When invertible, the inverse of a PL function may be computed in closed form by simply reversing the roles of the domain and codomain triangulations. Thus, a PL function is computationally quite simple.

The degree of parsimony of an approximant reduces to the question of how approximation errors decrease as the dimension of the defining parameter space is allowed to increase. In the case of a PL, the dimension of the parameter space is a linear function of the number of knot points. It is standard practice in the color systems management industry [6] to use PL interpolation on calibrated lookup tables (LUT) wherein the input-output data are treated as knots. Unfortunately, the resulting PL is not generally invertible. Moreover, since the grid density of an LUT is typically quite high (e.g., entailing thousands of color patches) the resulting PL cannot be considered parsimonious. Given an LUT data set, it seems natural to inquire whether some better means of locating the knots might yield an invertible and far more parsimonious PL representation.

In this paper, we introduce the "minvar" algorithm for computing piecewise linear (PL) approximations of color space transformations. For a given set of sampled (input-

*This work is supported in part by NSF Award ECS-96322801

output) data, an initial set of knots and a specified connectivity, these algorithms seek to improve the derived PL approximation of the sampled data by iteratively moving the knots. Thus, the minvar algorithm yields continuous PL approximations that can serve as functional replacements wherever look-up tables are presently used, while adding the guarantee of closed form invertibility. At the present early stage of inquiry, for the specific variants we have examined, the minvar procedure yields several factors of improvement relative to standard industrial practice. We believe that further refinement of the algorithm might well yield considerably greater improvement.

2. Rationale

Historically, xerographic process control has focused on stabilization of the process itself. It is common practice to calibrate the xerography during setup or warm-up and run largely open loop during printing. High quality, high speed color printing at high throughput rates requires active feedback and feedforward controls to maintain stable, predictable color performance. Xerox has practiced closed-loop control in copiers and printers for nearly 20 years. For monochrome and early color printing, stabilizing the process alone was sufficient. However, the number of process actuators available is very limited and the number of observables that require stabilization are many.

A 3 level control architecture was devised and patented in 1995 [8]. The first two levels of this structure stabilize the process by sensing internal variables (TC, Temp, RH, P/R voltages, etc.) and actuating (biases, corotron voltage, toner dispense, etc.) to remediate the process variations. Recent innovations have been productized first in the DocuColor 40 and more recently in the DocuColor 2060. These innovations involve color sensing at the output and feedback to the Tone Reproduction Curves (TRC) in the imaging system. Many short term variations in color reproduction are compensated for in this way. However, these three control processes only stabilize the single separation TRCs and do not address the disturbances that affect the color mixing recipes. Materials and process variability still require that these disturbances be addressed. The customer's frequent need for printer recalibration is testimony to this fact.

The next level to be addressed is the color mixing tables. These large LUTs are constructed by printer calibration and interpolated to yield the halftone densities of the individual primaries that are layered to construct the customer's desired colors. The data used to construct these tables is typically 1000 or more color patches each printed using a specified recipe (CMYK) and then colorimetrically read to determine the color value that resulted. Thus the forward table is constructed by interpolation of these data values [5,7].

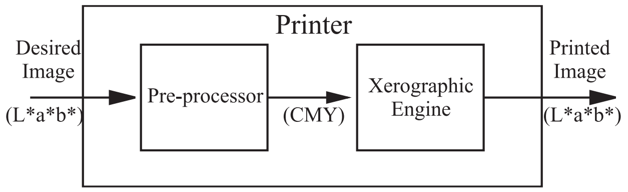

Figure 1: Decomposition of a xerographic printer into the print engine and pre-processor. The engine embodies the transformation T, while the pre-processor performs an approximation of the inverse $\tilde{T}^{-1}$, such that $T \circ \tilde{T}^{-1}$ is approximately the identity for within gamut colors.

There are two difficulties that attend migrating this process to real time in the machine. The first is the number of patches required and the allowable real estate needed to place them. Making paper prints reduces the productivity of the device and also requires that the patches be read. Input scanners would be useful for this purpose but there are large regions of color space where their sensitivity is inadequate. The second is that if the patches are made slowly and read by machine sensors, the number of patches required would consume a time comparable to the disturbances and thus are without value. This situation argues for a technique that will yield sufficient accuracy but require many fewer patches. If such a technique were realized, the goal of real-time control of color mixing might be accomplished.

The color space transformation problem may be abstracted to that of finding a parsimonious representation of multi-dimensional, non-linear transformations. The print engine embodies the transformation T from device dependent coordinates CMY to device independent coordinates L*a*b*, which is only observable through a set of color patch experiments as described above. Color science indicates that T is invertible when the domain is suitably restricted. An approximate inverse of the printer transformation $\tilde{T}^{-1}$ should be embedded in the printer's preprocessor, so that the composition $T \circ \tilde{T}^{-1}$ is approximately the identity map within gamut colors. In other words, the color requested of the printer is the color printed.

A LUT is one possible representation of such transformations, but LUTs are highly parameterized, and hence do not admit rapid update from sparse data, a requirement for xerographic process control. Experiments only provide knowledge of the forward transformation T, but to produce a uniform LUT some knowledge of the inverse transformation is necessary in order to select the appropriate color patches. An approximation technique which is closed form invertible would be beneficial, since control at the lower levels directly effects the forward transformation T, while T^{-1} is required by the preprocessor.

In light of these requirements we have chosen to pursue

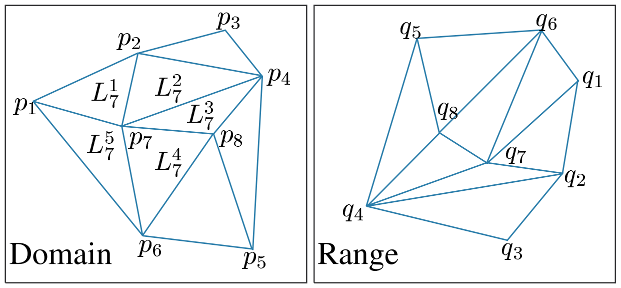

Figure 2: *An example of a two-dimensional piecewise linear function. Since the connectivity of the q_i's, inherited from the p_i's, is also a valid triangulation, the function is invertible. Moreover, it may be inverted simply by switching the labels of Domain and Range. The L_7^i are the least squares fits for cells surrounding p_7, computed in the first step of the minvar algorithm. Notice that L_7^3 and L_7^4 are also called L_8^j and L_8^k, respectively, since, as is generally the case, these cells are owned by more than one interior vertex.*

piecewise linear (PL) approximation of continuous invertible functions. This is a problem of general interest, arising when a relationship with some known invariant between inputs and outputs must be deduced from data. Applications exist in nearly every field of engineering.

3. Introduction to Multidimensional Piecewise Linear Functions

Since we are predominantly interested in approximating invertible functions, we assume the domain and codomain have the same dimension, $f : \mathbb{R}^n \to \mathbb{R}^n$. We will refer to PL functions of this type as n dimensional. A continuous piecewise linear function, f_P, admits a parsimonious representation. The domain of the function is partitioned into simplices. (A simplex is the convex hull of $n+1$ points: a line segment in one-dimension; a triangle in 2-D; a tetrahedron in 3-D; and so on.) Such a partition is called a triangulation. Call the vertices p_i. For each p_i assign a point in the codomain, q_i. This implicitly defines f_P such that $q_i = f_P(p_i)$. We refer to the pair (p_i, q_i) as a knot point of the piecewise linear function, and P is the set of all knot points. The triangulation of the domain implies a potential triangulation of the range. If $p_{i_1}, \dots, p_{i_{n+1}}$ are the vertices of a simplex in the domain, then in this simplex the PL function takes on values from the simplex in the codomain with vertices $q_{i_1}, \dots, q_{i_{n+1}}$. If the codomain simplices form a partition of the range, (i.e. they do not overlap or "tangle") then the PL function is invertible in closed form, and the inverse may be computed by simply switching the roles of the domain and range vertices. A PL function is parameterized by the location of its knot points and their "connectivity," the triangulation of the knots. (Generally for $n > 1$, there is more than one possible triangulation for a given set of vertices.) Figure 2 shows an example of a two dimensional PL function.

3.1. Evaluating the PL function

Suppose we have an n-dimensional piecewise linear function f_P and a query point x and we would like to find $f_P(x)$. First we must determine in which simplex x lies, say the i^{th} simplex with vertices $p_{i_1}, \dots, p_{i_{n+1}}$. We may verify that x does indeed lie in this simplex by checking that the $n+1$ barycentric coordinates of x with respect to the vertices are ≥ 0. This condition may be written as

$$W^{-1} \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0 \tag{1}$$

where W is defined below. Then $f_P(x)$ is simply computed as

$$\begin{bmatrix} f_P(x) \\ 1 \end{bmatrix} = QW^{-1} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2}$$

where

$$W = \begin{bmatrix} p_{i_1} & \dots & p_{i_{n+1}} \\ 1 & & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} q_{i_1} & \dots & q_{i_{n+1}} \\ 1 & & 1 \end{bmatrix} \tag{3}$$

For a fixed PL function, performance can be increased by precomputing the family of matrices QW^{-1}, which reduces the computational cost of evaluating the approximation to a matrix multiplication.

4. The minvar algorithm

Let $\mathcal{Z} = (x_i, y_i)_{i=1}^d$ be a data set of d input-output pairs. In function approximation, we search through a parameterized family of functions in order to minimize some error criterion, such as mean squared error (MSE) or maximum error. The piecewise linear functions are parameterized by the location of the knots and their connectivity. For the present we will treat knot location as the only changeable parameter, reserving the discussion of changing connectivity till the end of the section. Moreover, we generally fix any p_i which is an extreme point of the domain, that is, on a "corner" of the domain. This prevents the algorithm from unwittingly shrinking the domain of the approximation. Often, approximation schemes use a gradient scheme to minimize the mean squared error. The mean square error is

$$MSE = \frac{1}{d} \sum_{i=1}^{d} \| y_i - f_P(x_i) \|^2 \tag{4}$$

In the case of a piecewise linear function, $f_P(x)$ is as defined in §3.1. While gradient descent algorithms guarantee a reduction in the MSE from the point in the parameter space at which they start, the descent can often get stuck in a local minimum of the MSE, which could be much worse than the global minimum. Also, gradient descent tends to be computationally expensive and slow.

The minvar algorithm is a non-gradient method for computing a good n-dimensional piecewise linear approximation to a set of data. It is a generalization of the graph intersection (GI) algorithm, for one-dimensional approximations [3,4]. Numerical evidence shows that the GI algorithm is less computationally costly than a gradient descent algorithm for computing piecewise linear approximations. Moreover, piecewise linear approximations from the GI algorithm have a similar error per parameter rate as compared to other approximation techniques such as neural networks and Taylor series polynomials. Moreover, there is a local convergence proof for the GI algorithm. The success of the one-dimensional algorithm motivated a generalization to higher dimensions.

Each iteration of the minvar algorithm, like the GI algorithm, consists of two basic steps. In the first step, for each simplex in the domain triangulation the least squares linear fit is computed for the data in that cell. These are locally optimal, but in general will be discontinuous at the simplex boundaries. The least squares linear fit can be computed in closed form. For example, if $\mathcal{Z}_i \subset \mathcal{Z}$ is the subset of data which lies in cell i, (That is, the domain points of the data lie in the simplex given by $p_{i_1}, \ldots, p_{i_{n+1}}$.) then the least squares map, $L(x) = Ax + b$, is given by

$$\begin{bmatrix} A^{\mathrm{T}} \\ b^{\mathrm{T}} \end{bmatrix} = U^{-1} V \tag{5}$$

$$U = \frac{1}{|\mathcal{Z}_i|} \sum_{x_j \in \mathcal{Z}_i} \begin{bmatrix} x \\ 1 \end{bmatrix} \begin{bmatrix} x^{\mathrm{T}} & 1 \end{bmatrix} \tag{6}$$

$$V = \frac{1}{|\mathcal{Z}_i|} \sum_{x_j \in \mathcal{Z}_i} \begin{bmatrix} x \\ 1 \end{bmatrix} y^{\mathrm{T}} \tag{7}$$

The second step in the minvar algorithm moves the knots of the PL function to enforce continuity while trying to match the least squares fits. In the GI algorithm, this step moves the knot point to the intersection of the least squares fits from knot's two neighboring cells. The n-dimensional case is more difficult since a vertex is generally shared by more than two cells, so there is no unique intersection point. For example, in Figure 2, the vertex p_7 is a member of 5 different cells, $L_7^1, \ldots, L_7^5$. Since there will not in general be a unique intersection point, we choose to move the domain vertex to the point where the surrounding least squares fits have the most tightly clustered outputs. Mathematically we express this as a minimization of the variance of the least squares fits over the domain. Explicitly,

$$p_i(k+1) = \underset{x \in \mathbb{R}^n}{\arg\min} \ \ \mathrm{var} L_i^j(x) + \lambda \left\| x - p_i(k) \right\|^2 \tag{8}$$

where

$$\mathrm{var} L_i^j(x) = \sum_{j=1}^{N_i} \left(L_i^j(x) - \frac{1}{N_i} \sum_{l=1}^{N_i} L_i^l(x) \right)^2 . \tag{9}$$

The λ term is a regularization that restrains the vertex from moving long distances. This is necessary to prevent a vertex from moving through one of its opposing faces, "tangling" the triangulation. Notice that (8) is quadratic and can be minimized in closed form,

$$p_i(k+1) = -H^{-1} h \tag{10}$$

where

$$H = \frac{1}{N_i} \sum_{j=1}^{N_i} \tilde{A}_j^{\mathrm{T}} \tilde{A}_j + \lambda I \qquad h = \frac{1}{N_i} \sum_{j=1}^{N_i} \tilde{A}_j^{\mathrm{T}} b_j - \lambda p_i(k)$$

$$\tilde{A}_j = A_j - \bar{A} \qquad \tilde{b}_j = b_j - \bar{b}$$

$$\bar{A} = \sum_{j=1}^{N_i} A_j \qquad \bar{b} = \sum_{j=1}^{N_i} b_j$$

where N_i is the number of simplices that share the vertex p_i. The range value q_i is then computed as a weighted average of the least squares fits.

$$q_i(k+1) = \frac{1}{\sum_{k=1}^{N_i} \alpha_k} \sum_{j=1}^{N_i} \alpha_j L_i^j(p_i(k+1)) \tag{11}$$

There are a number of reasonable alternatives for these weights, the easiest being uniform weights, $\alpha_j = 1$. Alternatively, using $\alpha_j = |\mathcal{Z}_j|$ (i.e. the cardinality of the population in $\mathcal{Z}_j$) gives a lower mean squared error by giving a weightier vote to cells with more data points. Another alternative would be weighting based on cell volume or cell location.

In our problem, the domain of the approximation is a cube, and it is necessary to use vertices on the cube's faces and edges in addition to the corners in order to yield a good approximation. Rather than rigidly fixing these boundary points which are not extreme, it is possible to derive a constrained version of the minimization above, which will restrict movement of the vertex to an affine subspace, such as a plane or edge.

One recurring problem with the current implementation of the algorithm is mesh tangles. The minvar algorithm often adjusts the knots in such a way that one knot moves toward and through one of its opposing faces, giving an invalid triangulation of the domain. There are several ways to deal with this problem. One is to add weights to the cost function that would penalize making a cell too thin. Alternatively, we may view tangling as an indication from the algorithm that the underlying function should be approximated using a different triangulation of the knot points, and change the triangulation appropriately. This is an appealing solution to the tangling problem, but progress has been slowed by a scarcity of retriangulation algorithms for dimensions higher than three, though some recent work looks promising [2].

5. Comparison to Industry Practice

Xerox has provided a set of data from CMY to L*a*b* generated from a color model [1] of a commercial printer. This data forms a 21x21x21 uniform grid in CMY space. Three uniform LUTs, with sizes 3x3x3, 6x6x6, and 11x11x11, were subsampled from the data, in order to compare the minvar algorithm against a typical industry LUT. Note that these LUTs go from CMY to L*a*b*, whereas a printer requires the inverse LUT, which is more difficult to construct. However, these results give a flavor of the performance of PL against a LUT for an interesting nonlinear function. A validation set, one tenth of the total amount of data, was selected from the points remaining after the subsampling. All data except the validation data was used for building the piecewise linear approximation using the minvar algorithm.

Tetrahedral interpolation is used on the lookup table. This interpolation scheme breaks up each cube of data in the LUT into six tetrahedra and does piecewise linear interpolation on these tetrahedra. Thus, in this case the LUT is itself a PL function, with very rigidly located knots. The potential benefit of the PL comes from the flexibility of moving the knots, locating more approximation effort in areas that need it.

Three different error measures are given for comparison. The first is the root mean squared error (RMSE), which is square root of MSE as defined in (4), so that the units are sensible. The second error measure is called $\overline{\Delta E}$ (or average ΔE). This is an average Euclidean distance in L*a*b* space, given by

$$\overline{\Delta E} = \frac{1}{d}\sum_{i=1}^{d} ||y_i - f_P(x_i)|| \qquad (12)$$

This quantity will be less than or equal to the RMSE. A ΔE of 1 corresponds to a just noticeable color difference. The third error measure is the L_∞ norm, or "max" norm,

Table 1: *Comparison of various error measures for tetrahedral interpolation LUTs and PL approximations computed by the minvar algorithm. PL(i, j, k, l) is a PL function with i knots fixed in the domain, j knots constrained to a 1D affine subspace in the domain, k knots constrained to a 2D affine subspace in the domain, and l freely movable knots.*

	Parameters	RMSE	ΔE	L_∞
PL (8,0,0,1)	30	5.01	4.19	16.98
PL (26,0,0,8)	126	2.95	2.46	10.59
PL (28,0,0,8)	132	2.72	2.31	8.42
PL (28,0,0,9)	138	2.73	2.32	8.87
PL (23,4,2,8)	143	2.36	2.07	6.89
LUT 3x3x3	81	6.94	6.17	16.92
LUT 6x6x6	648	1.31	1.10	3.66
LUT 11x11x11	3993	0.62	0.40	2.35

given by

$$L_\infty = \max_{i=1,\dots d} ||y_i - f_P(x_i)|| \qquad (13)$$

This error measure is important, because sometimes, as in the case of trademark colors, it is necessary to print specific colors very accurately. Thus, it necessary to bound the maximum color error over the entire space.

It is instructive to compare the errors of these approximations in relation to the number of parameters each approximation has at its disposal. For a uniform lookup table, the domain points are rigidly fixed. The only parameters are the range values of the table. Since the range is three dimensional, there are three parameters per point. Thus there are $3k^3$ parameters in a $k \times k \times k$ LUT. In the minvar algorithm, the domain values of knots on the exterior of the cube are fixed. Like the lookup table, these knots have 3 parameters each, because only the range value can change. For knots on the interior, both the domain and range can change, and thus these knots have 6 parameters each. If k is the total number of knots, and l is the number interior knots, then the total number of parameters is $3(k - l) + 6l$.

To show the potential benefit of moving knots we compared the 3x3x3 lookup table against a PL function with 9 knot points. The PL function had eight fixed vertices on the corners of the domain, and one movable interior vertex, giving a total 30 parameters. Though the PL has less than half the number of parameters as the LUT, it performs slightly better in terms of MSE and similarly in terms of L_∞ error.

Increasing the number of parameters in the PL to around 130 drops the MSE to under 3. In this case, it often appears that the initial triangulation is as important in forming a good approximation as the number of knots. This can be seen between PL(28,0,0,8) and PL(28,0,0,9). The latter, even though it has an additional knot point, has a "worse"

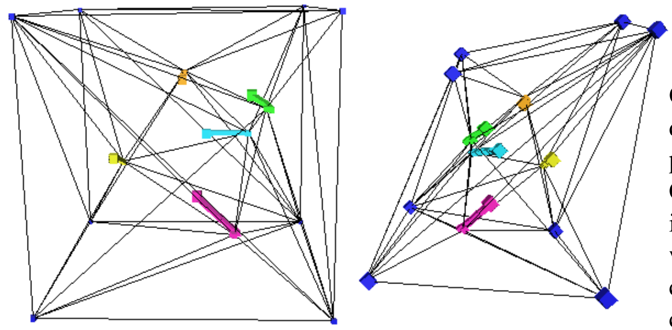

Figure 3: *A visualization of the domain (left) and range (right) triangulations of a PL approximation to color data. The function has 8 knots which have fixed domain values, the corners of the cube, and 5 knots which are freely movable. The colored trails mark the movement of the knots under the minvar algorithm.*

triangulation, with one knot connected to knots on all six faces of the cube. The former ends up with better error statistics even though it has less parameters. This underlines the importance of the triangulation as a parameter in the PL representation.

The remaining two LUTs have huge numbers of parameters and very good error statistics. The 6x6x6 LUT has about 5 times as many parameters as the highest complexity PL function presented here, and a little less than half the MSE. Unfortunately we can not present a comparatively parameterized PL approximation at this time, because progress on complex triangulations is inhibited by our lack of a local retriangulation scheme.

Anecdotally, the color space transformation is more complicated along the boundary of the domain. From numerical experience with the Xerox data, it was found that the points have the highest error norm were generally on the exterior. Generally the more knots a cell has on the boundary, the higher the MSE, suggesting that extra knots should be put on the boundary to drive down the errors, which is what PL(26,0,0,8), PL(28,0,0,8) and PL(28,0,0,9) do. For example, PL(26,0,0,8) has fixed knots on the cube on each corner (8), the middle of each edge (12) and the center of each face, for a total of 26 knots with fixed domain. Allowing constrained movement of the knots on the boundary greatly improves the efficacy of the PL approximations. Note that PL(28,0,0,8) and PL(23,8,4,2) have the same number of knots, but the extra degrees of freedom provided by the constrained motion greatly improves the error statistics.

As the minvar algorithm continues to mature and we improve our treatment of triangulation and retriangulation, we expect PL to perform better in terms of error as compared to LUT for approximations with a similar number of parameters. The PL representation has the benefit of being closed form invertible.

6. Conclusions

Our present version of the minvar algorithm requires a great deal of hand tuning to yield good results. Many of the parameters that require such tuning are being automated. Currently boundary vertex constraints must be hand specified, but yield significant improvements. This fits well with anecdotal claims that much of the non-linearity of the color space transformation are near the boundaries of the cube where colors saturate.

Another area for algorithm improvement is retriangulation. By locally reworking the triangulation based on some local criterion it might be possible to completely avoid mesh tangles, and guarantee convergence of the algorithm. Triangulations of spaces with dimension higher than three have received relatively little study in the past.

We remain cautiously optimistic that a more systematic approach to the placement of knots may yield PL approximants to color space transformations that are far more parsimonious than the industrial standard LUT yet offer the benefits of closed form invertibility as well.

The performance evaluation of color calibration algorithms resides, as always, in the analysis of printed output. Once the remaining parameterization issues are resolved, we will exercise the resulting calibration statically by printing a set of standard test images. The goal of this research is to provide printers with real-time calibrations that stabilize color predictability from monitor to print. With the a parsimonious representation of the transformation in hand, our focus will move to the level 4 control algorithms.

7. References

[1] Raja Balasubramanian, The Use of Spectral Regression in Modeling Halftone Color Printers, *Optical Society of America Annual Meeting*, October, 1996.

[2] H. Edelsbrunner and N.R. Shah, Incremental topological flipping works for regular triangulations, *Algorithmica*, 15(3):223–41, 1996.

[3] Richard E. Groff, Daniel E. Koditschek, and Pramod P. Khargonekar, Training Piecewise Linear Homeomorphisms for Approximation of Maps with Known Invariants: The Scalar Case, *Journal of Complexity*, submitted.

[4] Richard E. Groff, Daniel E. Koditschek, and Pramod P. Khargonekar, Piecewise Linear Homeomorphisms: the Scalar Case, *IJCNN 2000*, to appear.

[5] K. Kanamori and H. Kotera, Color-correction technique for hard copies by 4-neighbors interpolation method, *J. Imaging Sci. Technol.*, 36(1):73–80, 1992.

[6] Henry R. Kang, *Color Technology for Electronic Imagining Devices*, SPIE Optical Engineering Press, 1997.

[7] J. M. Kasson, S. I. Nin, W. Plouffe, and J. L. Hafner, Performing color space conversions with three dimensional linear interpolation, *J. Electron Imaging*, 4(3):226–50, 1995.

[8] Tracy E. Thieret, Thomas A. Henderson, Michael A. Butler, *Method and Control System Architecture for Controlling Tone Reproduction in a Printing Device*, U.S. Patent 5,471,313, November 28, 1995.