Improved color table inversion near the gamut boundary

Miheer Bhachech[1], Mark Shaw[2] and Jeffrey M. DiCarlo[1]
[1]Hewlett-Packard Laboratories, Palo Alto, CA, [2]Hewlett-Packard Company, Boise, ID

Abstract

Color inverse tables, although seemingly accurate, introduce color errors and artifacts near the gamut boundary of a device. These artifacts are due to a combination of gamut mapping and interpolation: Gamut mapping breaks the correspondence between connection and device space vertices, and interpolation subsequently fails. An algorithm has been designed for more accurate inversions at the gamut boundary. It is based on the extrapolation of vertices instead of gamut mapping and interpolation. Results on different printers and media types show a significant improvement in color accuracy near the gamut boundary compared to standard inversion techniques.

Introduction

Typically, an image captured with a camera or scanner has various color transformations applied to it before it is printed in order to produce a pleasing and faithful reproduction of the original scene[1-3]. For example, a camera image is usually transformed from the camera sensor space to a connection space (usually XYZ or Lab) and then transformed to the printer colorant space. These transformations are applied either within the ICC framework[4] or directly within the device, but either way, they are vital for high-quality color reproductions.

Standard techniques for designing these color transforms[5-7], however, can introduce significant color errors and artifacts near the gamut boundary of a device. This is illustrated in Figure 1. It shows an original test image (Panel A) and the same test image after two color transformations that were designed to be completely invertible to each other (Panel B). Ideally, these images should be identical, but as the difference image shows (Panel C), colors that were highly saturated are less saturated and contouring artifacts have been introduced by the color transforms.

These color errors and artifacts occur in standard table-based (not matrix-based) color transforms due to the combination of gamut mapping and interpolation used in building the inverse table. Typically, the *forward table*, which is the mapping from a device space to a connection space, is built by characterizing the colors that span the device gamut using a color measuring device. The *inverse table* for a device, which is the mapping from a connection space to a device space, is usually determined by a combination of gamut mapping[8-11] colors outside of the device gamut and tetrahedral inversion[5-7] for colors inside the gamut. It is the process used to build these inverse tables that causes the desaturation of colors near the gamut and the color artifacts.

Figure 2 illustrates how gamut mapping and interpolation combine in standard inverse tables to cause color desaturation and artifacts. Panel A represents a tetrahedron in Lab space (connection space). Ideally, the inverse RGB values (device values) are determined by inverting each Lab vertex (a, b and c) using the forward table and tetrahedral inversion. This, however, can be done only for vertices within the device gamut (vertex c).

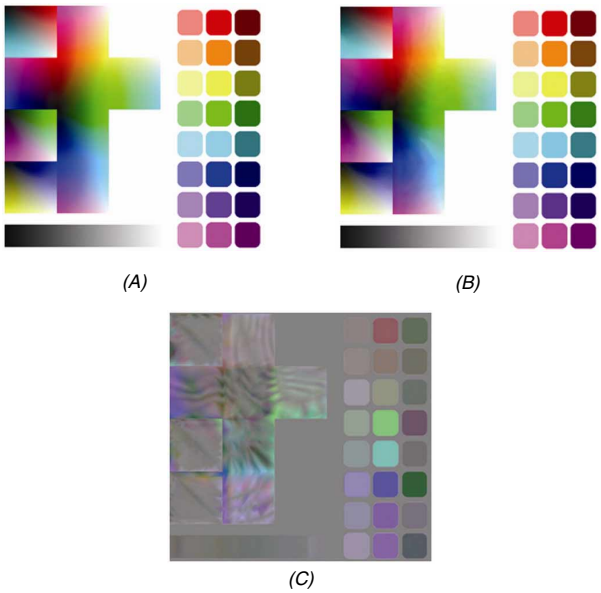

Figure 1. *Example of color errors and artifacts introduced by standard color transformations. Panel A shows the original test image. Panel B shows the same test image after two color transformations designed to be invertible using standard techniques, and Panel C shows the difference between A and B.*

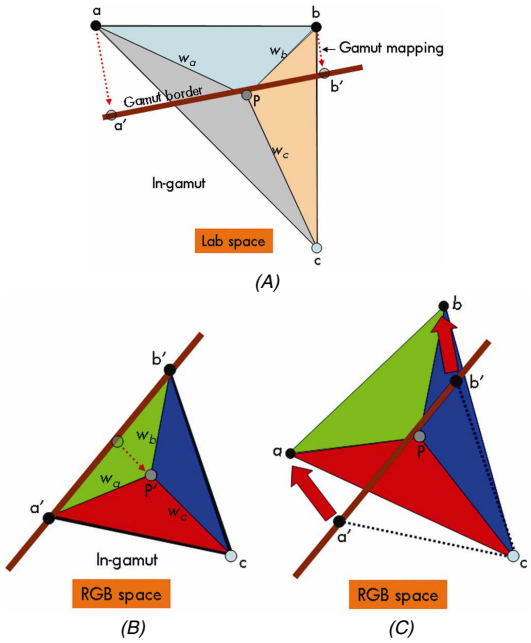

Figure 2: *Inverse color tables introduce color desaturation and artifacts near the gamut boundary. See text for details.*

Vertices outside the device gamut (a and b) are usually mapped to the gamut boundary (a' and b') and then inverted. Gamut mapping causes a difference between the tetrahedron vertices in Lab space and those in RGB space. In Lab space, the vertices are points a, b and c, but in RGB space the vertices are a', b' and c (see Panel B).

When these tables are applied in a device or ICC framework, interpolation will cause color desaturation and artifacts due to the lack of correspondence between vertices in Lab and RGB spaces. To illustrate, suppose we had a color on the gamut border in Lab space, e.g. point P in Panel A. A color management module (CMM) would determine it was in the tetrahedron defined by a, b and c in Lab space. Next, it would determine the distances to each of the tetrahedron vertices and compute interpolation weights. These are illustrated by w_a, w_b and w_c. Next, the CMM would use these weights and the RGB vertices to compute the corresponding color in device space. The result would cause the point P to move into the device gamut due to lack of vertex correspondence (point P' in Panel B), effectively desaturating the color.

In this paper, we present a solution to color inverse tables that cause color desaturation and color artifacts. Specifically, instead of gamut mapping any out-of-gamut connection space vertex before inversion, we extrapolate these vertices in the device space. This is illustrated in Panel C of Figure 2. The algorithm, which we refer to as *color extrapolation*, preserves the correspondence between device and connection space vertices and thus does not produce color desaturation or color artifacts. Vertex extrapolation methods have appeared in the literature that produce more accurate forward printer and scanner tables from limited color measurements[12, 13]. Here, we assume an accurate forward table and focus on developing an accurate inverse table that fully utilizes the device's gamut by not desaturating colors.

Color extrapolation

The first step of the color extrapolation algorithm is to classify the connection space vertices in the inverse table into one of three categories: *in-gamut*, *border* or *non-border* vertices. In-gamut vertices are defined to be completely contained within the device gamut. Border vertices are defined to be outside the device gamut, but they reside very close to the gamut boundary that they influence the interpolation of in-gamut colors. These are the vertices that must be extrapolated. Finally, non-border vertices are also outside the device gamut, but they are far enough from the gamut boundary that they have no influence on the interpolation of any in-gamut colors.

Figure 3 shows a two-dimensional example of vertex classification. In the figure, black vertices are in-gamut vertices; red vertices are border vertices; and green vertices are non-border vertices. The triangles interconnecting the vertices illustrate the interpolation or tessellation structure of the two-dimensional table. Specifically, these triangles in two-dimensions (tetrahedra in three-dimensions) define the vertices that are used to interpolate colors within the triangle. Because border vertices by definition influence the interpolation of in-gamut colors, border vertices can be found by identifying the out-of-gamut vertices of tetrahedra that cross the device gamut.

After classification of the inverse table vertices as in-gamut, border or non-border, the next step of the algorithm is the extrapolation of the border vertices. To extrapolate a border vertex, we must first select forward table vertices that can train or

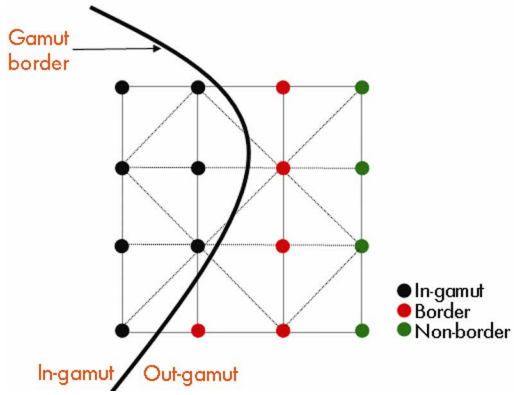

Figure 3: *Classification of the inverse table connection space vertices into in-gamut vertices (black), border vertices (red), and non-border vertices (green). Border vertices are outside the gamut, but influence the interpolation of in-gamut colors.*

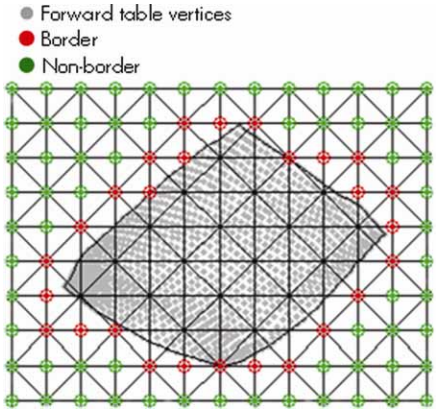

Figure 4: *The training data used for color extrapolation of border vertices are selected from the forward table vertices (gray).*

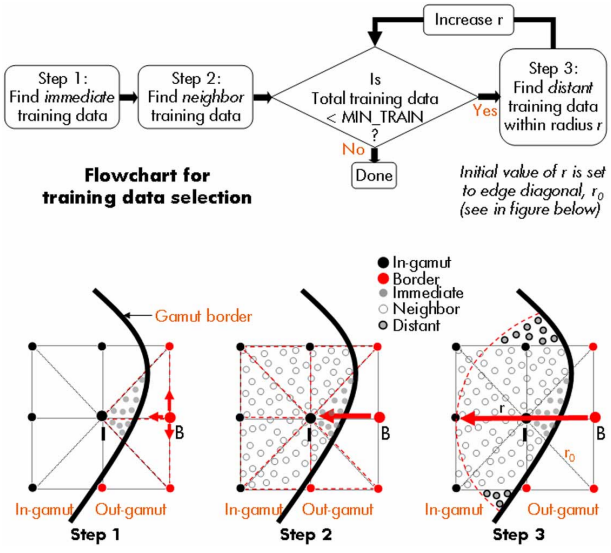

Figure 5: *Training data selection flowchart and geometry for the extrapolation of the border vertex B.*

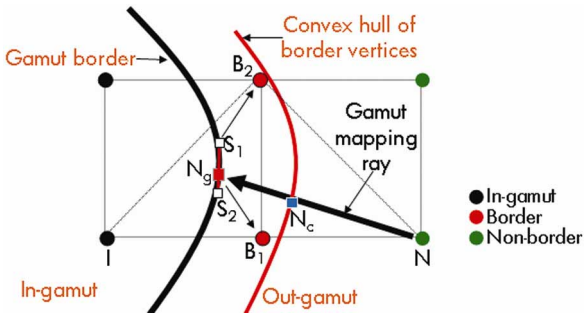

Figure 6. *Geometry illustrating the inversion of non-border vertices. See text for details.*

Figure 7. *ICC color workflow architecture for standard printers.*

guide the extrapolation algorithm. These forward table vertices, which we refer to as *training data,* enable extrapolation because they contain both accurate device and connection space values. Figure 4 illustrates example forward table vertices for an a*-b* plane in connection space. Their corresponding RGB values are not shown.

A flowchart and illustration of the steps involved in selecting the training data from the forward vertices is given in Figure 5. Here, we assume B is the border vertex that needs to be extrapolated. In Step 1, the forward table vertices that are in the region of influence of B are first selected. We find these vertices by testing if each vertex is contained within any of the tetrahedra connected to B. If it is contained in a tetrahedron, we select the vertex. We refer to these selected vertices as *immediate training data*, and they are shown by the solid gray circles in the Step 1 panel.

In Step 2 of the training data selection, we find the *neighbor training data*. These are defined as forward vertices that do not directly fall in the tetrahedra connected to B but instead fall in tetrahedra that are connected to all in-gamut neighbors of B. In this particular figure, the border vertex, B, has one in-gamut vertex, I, directly connected to it. The neighbor training data, which are shown by the open gray circles, are all contained within the tetrahedra defined by the in-gamut vertex, I.

At this stage in the selection process, we must check that enough forward vertices have been selected to produce a robust extrapolation: Too few vertices can lead to extrapolation results that are imprecise. Hence, we confirm that the total number of immediate and neighbor training data is greater than MIN_TRAIN, which we set to twenty. If the total is greater, the selection process is complete and the border vertex can be extrapolated. Otherwise, we continue to Step 3.

In Step 3, additional forward table vertices must be selected to produce precise extrapolations. To select more data, we find the forward vertices that lie within a specific distance, r, from border vertex B. This additional training data, which we refer to as *distant training data*, are shown by solid gray circles with black borders in the Step 3 panel of Figure 5. After adding the distant data to the training data, we again test against MIN_TRAIN. If we still do not have enough vertices, the distance r is increased until at least MIN_TRAIN training points are selected.

Once enough training data are selected, the border vertex is extrapolated. There are many methods to extrapolate border vertices, but we found linear extrapolation to be simple and work well[14]. To perform linear extrapolation for a particular border vertex, we first place all the training data, immediate, neighbor and distant, into two matrices (See Equation 1). One matrix, $\mathbf{L}_T$, contains the training data values in the connection space, and the other matrix, $\mathbf{R}_T$, contains the same training data in the device space. (The subscript m denotes the total number of training samples.) In the equation, we assumed Lab for the connection space and RGB for the device space, but any color space can be used. Next, using the connection space value of the border vertex, $\mathbf{l}_B$, and the two matrices, $\mathbf{L}_T$ and $\mathbf{R}_T$, the device space value at the border vertex, $\mathbf{r}_B$, is extrapolated using Equation 2. (The symbol $\mathbf{X}^+$ denotes the pseudo-inverse of $\mathbf{X}$.)

$$\mathbf{L}_T = \begin{bmatrix} L_1 & L_2 & L_3 & \cdots & L_m \\ a_1 & a_2 & a_3 & \cdots & a_m \\ b_1 & b_2 & b_3 & \cdots & b_m \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$\mathbf{R}_T = \begin{bmatrix} R_1 & R_2 & R_3 & \cdots & R_m \\ G_1 & G_2 & G_3 & \cdots & G_m \\ B_1 & B_2 & B_3 & \cdots & B_m \end{bmatrix} \quad (1)$$

$$\mathbf{r}_B = \mathbf{R}_T \, \mathbf{L}_T^+ \, \mathbf{l}_B \quad (2)$$

where $\mathbf{l}_B = [L_B \; a_B \; b_B \; 1]^T$

This completes the process of extrapolating the border vertices in the inverse table.

After extrapolation of the border vertices, we must invert the non-border vertices. Traditionally, non-border vertices are simply gamut mapped to the device gamut and inverted. This normally is an acceptable solution because non-border vertices do not influence any in-gamut colors; however, due to the extrapolation of the border vertices in the previous step, we must take steps to gamut map in a way consistent with the extrapolation to prevent artifacts. (While additional steps are needed in the gamut mapping process, no restrictions have been placed on the gamut mapping algorithm by the color extrapolation algorithm.)

Figure 6 illustrates all the steps required to gamut map non-border vertices to a color extrapolated table. First, a convex hull is built around the border vertices, not the device gamut. This convex hull is illustrated by the red curve. Next, each non-border vertex (N in this example) is mapped in a direction defined by the gamut mapping algorithm. This direction is denoted as the *gamut mapping ray*. In typical gamut mapping applications, this ray is intersected with the device gamut, N_g, and the device value at N_g is encoded into the color table at vertex N. In the color extrapolation case, however, the N_g point is used to locate the vertices of the intersected gamut hull triangle. These vertices, S_1 and S_2, are used to find the closest border vertices (B_1 and B_2 in this case). Now, the exact same extrapolation process is used as described earlier

with the exception that the forward table vertices selected as training data come from the training data used for vertices B_1 and B_2, and the data is extrapolated to point N_c, the intersection of the gamut mapping ray with the convex hull, instead of N. By using the same extrapolation process and training data for non-border vertices as the border vertices, gamut mapping artifacts due to color extrapolation are eliminated.

Finally, we invert the in-gamut vertices. Nothing special is required for these vertices. We used standard tetrahedral inversion[5-7], but any other alternative inversion techniques can be used.

Color encoding

A consequence of color extrapolation is that both border and non-border vertices will be assigned device values beyond the gamut of the device, i.e. less than 0 or greater than 255 for an 8-bit device. These beyond gamut values cannot be directly encoded in the color tables of most printers and ICC workflows. Color table encodings are usually optimized for the device color gamut and usually do not allow negative values or values greater than the maximum device value. To circumvent this problem, we take advantage of the multiple one-dimensional tables that are usually available in color workflows both before and after the main three-dimensional color table.

Architecturally, it is quite common that printers and ICC workflows do not contain only a single three-dimensional table. They also contain multiple one-dimensional tables to transform the data before and after the full three-dimensional table. Figure 7 illustrates an example ICC workflow for a printer. Here, the workflow contains three one-dimensional tables before the main color table and four one-dimensional tables after the main color table. These additional tables allow us to encode color extrapolated values in ICC workflows and general printer workflows that we would otherwise not be able to encode.

To encode the border and non-border vertices in these workflows, the maximum and minimum device values of all the vertices are computed for each color channel. The vertices are then scaled and offset using the maximum and minimum values to ensure that all vertices are within the normal device range and encoded in the full three-dimensional table. Then to compensate for the applied scale and offset, we use the one-dimensional output tables to undo their effect.

The encoding strategy, however, comes with a cost. We sacrifice approximately one-bit in the main table because roughly half of the table is used for device colors, while the other half is used for extrapolation. Thus, to properly encode a color extrapolated table with a bit depth equal to a normal 8-bit color table, we need at least a 9-bit encoding for the main color table. The output tables, however, can still be encoded with 8-bits so that the processed color data is consistent with typical workflows. The ICC workflow in Figure 7 supports exactly this architecture.

Results

We compared the performance of the extrapolation method with the standard tetrahedral inverse method. In Figure 8, the gamut boundary of a printer is shown unwrapped as a function of hue and lightness. The colors in each plot indicate the magnitude of the error, CIE ΔE, between the ideal inversion and that obtained by each of the two methods for colors near the gamut surface. For

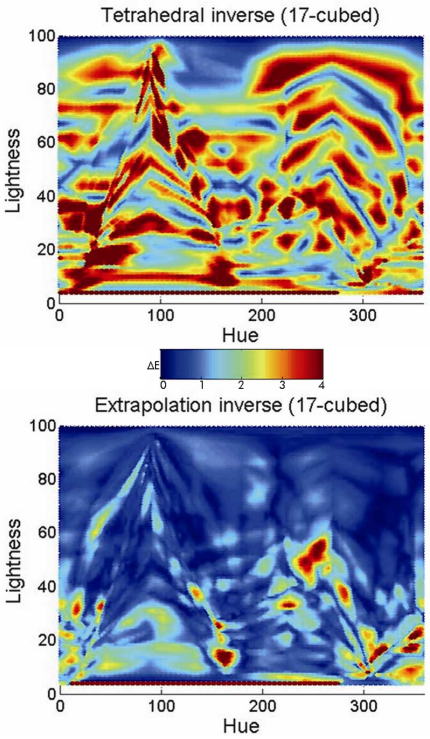

Figure 8. *Color errors introduced on the gamut boundary by different profiling techniques for a 17^3 inverse table. Each plot shows the gamut boundary of a printer unwrapped as a function of lightness and hue.*

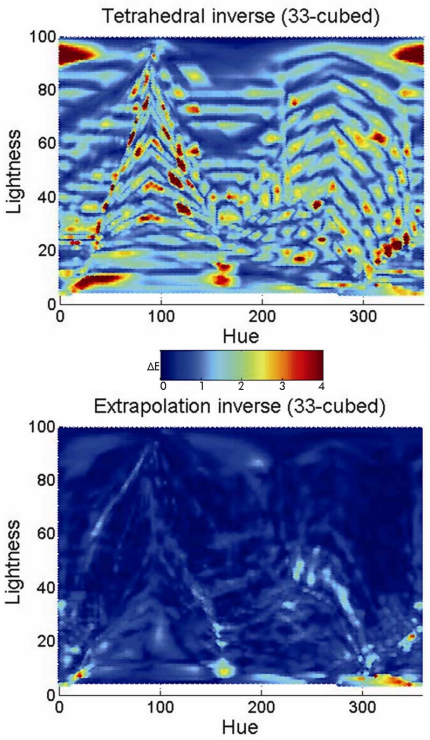

Figure 9. *Color errors introduced on the gamut boundary by two different profiling techniques for a 33^3 inverse table.*

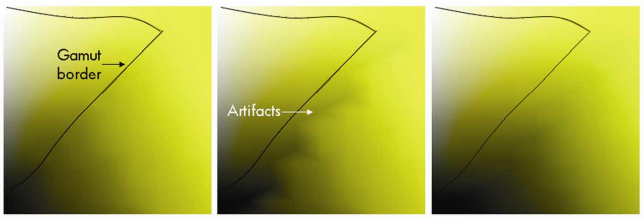

Figure 10. *Image artifacts can be introduced when gamut mapping color extrapolated tables if not properly compensated. Each plot shows the results of minimum ΔE gamut mapping on a hue plane at 91º using a standard tetrahedral table (Panel A), a color extrapolated table with no gamut mapping compensation (Panel B), and a color extrapolated table with gamut mapping compensation (Panel C).*

each method, we used the same number of vertices for the inverse table, 17^3. As evident in the plots, extrapolation has significantly reduced the color errors near the gamut boundary.

Figure 9 shows similar plots to those in Figure 8 except higher resolution inverse tables were constructed. Specifically, 33^3 vertices were used in the inverse table. In the figure, we can see the error for all methods decreased, as we would expect, but color extrapolation still produced significantly better results. Furthermore by comparing Figure 8 and 9, it can be seen that color extrapolated, low resolution (17^3) tables, which use much less memory than the high resolution tables, produced results similar to non-extrapolated, high-resolution (33^3) tables.

Figure 10 compares the results of gamut mapping to traditional tables and gamut mapping to color extrapolated tables for a particular hue slice. Specifically, Panel A shows minimum ΔE gamut mapping to a non-extrapolated color gamut; Panel B shows the same gamut mapping on a color extrapolated gamut with no compensation for extrapolation; and Panel C shows the same gamut mapping to a color extrapolated gamut but with the proposed compensation for the color extrapolated gamut. In Figure 10, we can see that traditional gamut mapping to a color extrapolated gamut produces artifacts (Panel B). These artifacts are caused by the differences between gamut mapping algorithm and the color extrapolation algorithm. If, however, the gamut mapping algorithm properly takes into account the color extrapolated gamut, the artifacts are eliminated (Panel C).

To gauge the color extrapolation method for different printing technologies and media types, Figure 11 shows the average ΔE values for in-gamut colors near the gamut surface for the two methods and for four technologies and media types: DesignJet130 Gloss, DesignJet130 Photo Satin, PhotoSmart7960 Gloss and ColorLaserJet4700 Gloss. As evident from the chart, extrapolation reduces the color errors near the gamut boundary for all media types.

Finally, Figure 12 shows color extrapolation algorithm applied to the image from Figure 1. Panel A shows the original image, Panel B shows the inverted image after being applied through a color extrapolated table, and Panel C shows the difference image. If we compare the color extrapolated difference image (Panel C) to the difference image from the non-extrapolated color table from Figure 1, which is repeated in Panel D to aid

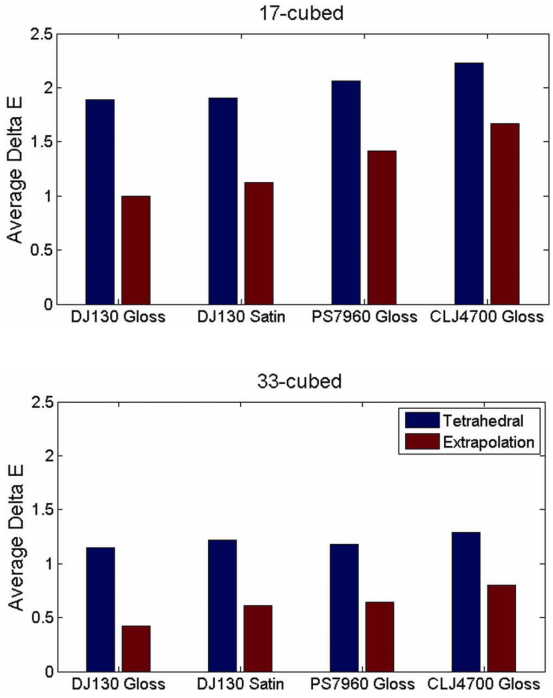

Figure 11. *Average ΔE values for in-gamut colors near the gamut boundary using different profiling techniques and different printers and media.*

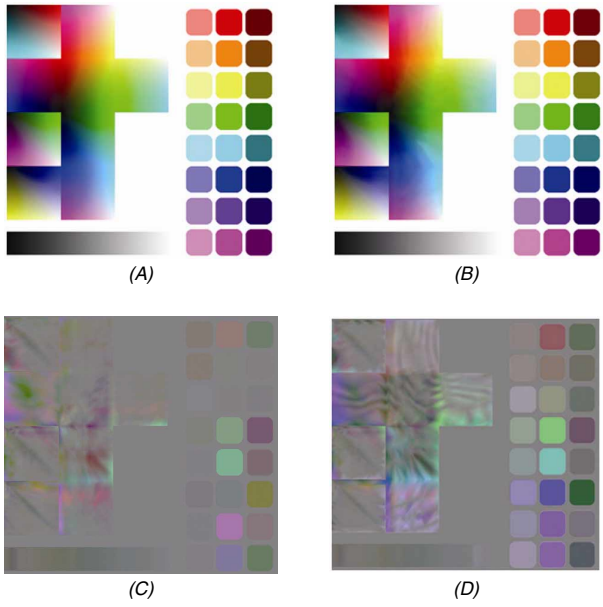

Figure 12. *Image color artifacts are significantly reduced using color extrapolation. Panel A shows the test image. Panel B shows the same test image with transformations built using color extrapolation. Panel C shows the difference between images A and B, and Panel D shows the difference image from Figure 1 that was produce using a standard tetrahedral inverse table.*

comparison, we see that color extrapolation significantly reduces color contouring artifacts and produces a more accurate inversion of colors near the gamut boundary.

Conclusions

We described a color extrapolation method for improving color table inversion near the gamut boundary. We introduced how existing gamut mapping algorithms can be used within the extrapolation method and how the resulting color tables can be encoded within an ICC or printer workflow. Results show the improved color tables have lower color errors near the gamut boundary; reduce the color artifacts in images near the gamut boundary; and utilize the full color capabilities of the device over standard tetrahedral inverse techniques. Finally, the results appear to be consistent across printing technologies and media types.

References

1. R. S. Berns, "Billmeyer and Saltzman's principles of color technology," (2000).
2. E. J. Giorgianni and T. E. Madden, Digital Color Management: Encoding Solutions (Addison-Wesley, Reading, Massachusetts, 1997).
3. M. Has, "Color management: current approaches, standards and future perspectives," presented at the Recent Progress in Color Management and Communications, 1998.
4. ISO 15076-1, "Image technology colour management: architecture, profile format and data structure," (2005).
5. H. R. Kang, Color technology for electronic imaging devices (1997).
6. I. E. Bell and W. Cowan, "Characterizing printer gamuts using tetrahedral interpolation," presented at the 1st IS&T/SID Color Imaging Conference, Scottsdale, Arizona, 1993.
7. R. Bala, "Inverse problems in color device characterization," presented at the SPIE: Computational Imaging, Santa Clara, California, 2003.
8. J. Morovic and P.-L. Sun, "Non-iterative minimum delta E gamut clipping," presented at the CIC 9: Color Science and Engineering Systems, Technologies, Applications, Scottsdale, Arizona, 2001.
9. G. Marcu, "Gamut mapping: an overview of the problem," presented at the PICS 1999: Image Processing, Image Quality, Image Capture, Systems Conference, Savannah, Georgia, 1999.
10. H. Zeng, "Gamut mapping in a composite color space," presented at the NIP17: International conference on digital printing technologies, Fort Lauderdale, Florida, 2001.
11. G. J. Braun and M. D. Fairchild, "Color gamut mapping in a hue-linearized CIELAB color space," presented at the CIC 6: Color Science, Systems and Applications, Scottsdale, Arizona, 1998.
12. P. Hung, "Colorimetric calibration for scanners and media," presented at the SPIE, 1991.
13. S. A. Rajala, H. J. Trussell, and A. P. Kakodkar, "The use of extrapolation in computing color look-up tables," presented at the SPIE, 1994.
14. G. Strang, Introduction to linear algebra, 2nd ed. (Wellesley-Cambridge Press, Wellesley, MA, 1998).