Indexing of Multidimensional Lookup Tables in Embedded Systems

Michael J. Vrhel, *Senior Member, IEEE*

***Abstract*—The proliferation of color devices and the desire to have them accurately communicate color information has led to a need for embedded systems that perform color conversions. A common method for performing color space conversions is to characterize the device with a multidimensional lookup table (MLUT). To reduce cost, many of the embedded systems have limited computational abilities. This leads to a need for the design of efficient methods for performing MLUT indexing and interpolation. This paper examines and compares two methods of MLUT indexing within embedded systems. The comparison is made in terms of colorimetric accuracy and computational cost.**

I. Introduction

COLOR imaging and reproduction devices are extremely common today. To accurately communicate color image data between these devices, it is necessary to perform transformations that account for the response of the source device (e.g., scanner or digital still camera) and the destination device (e.g., LCD, printer, etc). A high level description of these transformations is contained in [1], [10].

With the proliferation of standard RGB color spaces such as sRGB [2], many modern devices perform transformations from (to) sRGB to (from) the device native color space within an embedded system. Devices such as multifunction peripherals (MFPs) contain a scanner, a printer, and a FAX. In this case, the embedded system must perform a number of different transformations (e.g., scan-to-print, scan-to-FAX, scan-to-PC, FAX-to-print, etc.). Such embedded systems are limited in terms of memory and word width compared to desktop computers. In many cases, floating point operations are not available.

Multidimensional lookup tables (MLUTs) have been used for color space transformations for several years [3]–[6], [10]. An MLUT is included in the international color consortium (ICC) format for color device characterization [7]. While there is much literature on these tables and on extensions such as nonuniform tables [8], [9], [11, Section 11.3], there has not been published information on using these tables within the constraints of an embedded system. In [11, Section 11.2], Bala briefly discusses indexing methods but does not look at the errors that these methods can introduce. To fill that void, we look at the errors that are introduced by various indexing methods in MLUTs, and recommend an approach for embedded systems.

Manuscript received August 5, 2003; revised January 30, 2004. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Luca Lucchese.

The author is with ViewAhead Technology, Redmond, WA 98052 USA (e-mail: michael.vrhel@viewahead.com).

Digital Object Identifier 10.1109/TIP.2004.834658

The paper is organized as follows. Section II introduces some notation and reviews the uniform sampling of a color space. Section III introduces the mathematics for exact determination of the index and subindex values in the table for floating point and integer sampling. Section IV discusses and compares two methods for determining index and subindex values in an embedded system. Section V compares the computational cost of the two methods. Section VI compares the methods on real image data and finally Section VII provides a brief summary.

II. MLUT Sample Points

Let the true mapping between the M dimensional color space C and the N dimensional color space D be given by $\mathcal{F}$ where

$$\mathcal{F}(\mathbf{c}) \in D \tag{1}$$

for $\mathbf{c} \in C$.

For simplicity, (and to consider only cases of interest) let us assume that the elements of the vectors in C range from 0 to P. In addition, we will assume that $M = 3$. In this case, the entries for a uniform sampled MLUT of size [1]R, which approximates the mapping $\mathcal{F}$ are the values

$$\mathcal{F}(\mathbf{c}_{i,j,k}) = \mathbf{d}_{i,j,k} \quad i,j,k \in [0,\ldots,R-1] \tag{2}$$

where

$$\mathbf{c}_{i,j,k} = \left[\frac{P*i}{R-1}, \frac{P*j}{R-1}, \frac{P*k}{R-1}\right]^T. \tag{3}$$

T is used to denote the transpose operation and we assume vectors to be in column format. The approximation of a value $\mathcal{F}(\mathbf{c})$, where $\mathbf{c}$ is not on a sample point is calculated by an interpolation method. Two common MLUT interpolation methods are trilinear interpolation and tetrahedral interpolation. For ease of discussion later, let us define a function $\mathcal{H}$, which takes table indices as input and provides the table value at that location as output, which implies

$$\mathcal{H}(i,j,k) = \mathbf{d}_{i,j,k} \quad i,j,k \in [0,\ldots,R-1]. \tag{4}$$

As an example, consider an MLUT where $P = 255$ and $R = 17$. For this sampling step size, the sample values at the grid points are given by

$$0,\ 15.9375,\ 31.875,\ 47.8125,\ 63.75,\ 79.6875,\ 95.625$$
$$111.5625,\ 127.5,\ 143.4375,\ 159.375,\ 175.3125$$
$$191.25,\ 207.1875,\ 223.125,\ 239.0625,\ 255.$$

[1]The table will contain $N*R^3$ values.

TABLE I
TABLE SIZES WITH UNIFORM INTEGER STEP SIZES FOR $P = 255$

Table Size	Step Size
6	51
16	17
18	15
52	5

In this case, the value of the table at location $[i, j, k] = [14, 6, 5]$ is denoted by

$$\mathcal{H}(14, 6, 5) = \mathcal{F}([223.125\ 95.625\ 79.6875]) \tag{5}$$

which is the device output for an input of [223.125, 95.625, 79.6875]. Note that having float sample points is the defined method for MLUTs in the ICC profile format [7, pages 51,55]. In practice, the actual table is often created using rounded values, which in the above example are given by

0, 16, 32, 48, 64, 80, 96, 112, 128,
143, 159, 175 191, 207, 223, 239, 255.

This rounding introduces a nonuniformly sampled table by one count in the middle of the table (in the above example, all the sample values are 16 steps apart except when going from 128 to 143).

Note that there are a few table sizes, for $P = 255$, which have uniform integer step sizes. These table sizes are given in Table I.

III. INTERPOLATION

Given an arbitrary value $\mathbf{c} \in C$, the problem is to use the MLUT to approximate the value $\mathcal{F}(\mathbf{c})$. The approximation using trilinear interpolation consists of three steps:

1) determining the cube that contains the point $\mathbf{c}$ (which we will refer to as the cube index);
2) determining the subindexing or weight values within the cube;
3) computing the interpolation.

A. Finding the Cube Index

Since $M = 3$, the MLUT consists of $(R-1)^3$ cubes. Mathematically, the root index of the containing cube can be determined using

$$\begin{aligned}[\mathbf{d}]_i &= \text{FLOOR}\left\{\frac{[\mathbf{c}]_i(R-1)}{P}\right\} \quad [\mathbf{c}]_i < P \quad i = 1, 2, 3 \\ &= R-2 \qquad [\mathbf{c}]_i = P \end{aligned} \tag{6}$$

$$[I, J, K]^T = \mathbf{d} \tag{7}$$

where $[\mathbf{c}]_i$ denotes the ith element of the vector $\mathbf{c}$ and $[I, J, K]$ is used to denote the root index of the cube containing the point $\mathbf{c}$. Note that the elements of $[I, J, K]$ are in the range 0 to $R-2$.

Given this root index, it is possible to determine the table values that will be used in the trilinear interpolation. If the root index is given by the values $[I, J, K] < R-1$, then the eight table values that are used in the calculation are given by

$$\begin{aligned}&\{\mathcal{H}([I, J, K]), \mathcal{H}([I+1, J, K)\\ &\mathcal{H}([I, J+1, K]), \mathcal{H}([I, J, K+1])\\ &\mathcal{H}([I+1, J+1, K]), \mathcal{H}([I+1, J, K+1])\\ &\mathcal{H}([I, J+1, K+1]), \mathcal{H}([I+1, J+1, K+1])\}.\end{aligned}$$

B. Finding Subindex Values

In addition to the above table values, the trilinear interpolation computation requires subindexes into the cube along each of the three dimensions to determine how much to weight each of the eight table values in a summation. Assuming float values were used to create the table, the exact subindexes are given by the values

$$\mathbf{s_c} = \mathbf{c} - \mathbf{c}_{I,J,K} \tag{8}$$

and the weights are given by products of the elements of the vectors

$$\mathbf{w_c} = \frac{\mathbf{s_c}(R-1)}{P} \tag{9}$$

$$\mathbf{v_c} = 1 - \mathbf{w_c}. \tag{10}$$

Note that the elements of $\mathbf{w_c}$ and $\mathbf{v_c}$ are in the range 0 to 1.

C. Trilinear Interpolation

In the case of trilinear interpolation, the interpolated value is computed as

$$\begin{aligned}\mathbf{t} = &\mathcal{H}([I, J, K])[\mathbf{v_c}]_1[\mathbf{v_c}]_2[\mathbf{v_c}]_3\\ &+ \mathcal{H}([I+1, J, K)[\mathbf{w_c}]_1[\mathbf{v_c}]_2[\mathbf{v_c}]_3\\ &+ \mathcal{H}([I, J+1, K])[\mathbf{v_c}]_1[\mathbf{w_c}]_2[\mathbf{v_c}]_3\\ &+ \mathcal{H}([I, J, K+1])[\mathbf{v_c}]_1[\mathbf{v_c}]_2[\mathbf{w_c}]_3\\ &+ \mathcal{H}([I+1, J+1, K])[\mathbf{w_c}]_1[\mathbf{w_c}]_2[\mathbf{v_c}]_3\\ &+ \mathcal{H}([I+1, J, K+1])[\mathbf{w_c}]_1[\mathbf{v_c}]_2[\mathbf{w_c}]_3\\ &+ \mathcal{H}([I, J+1, K+1])[\mathbf{v_c}]_1[\mathbf{w_c}]_2[\mathbf{w_c}]_3\\ &+ \mathcal{H}([I+1, J+1, K+1])[\mathbf{w_c}]_1[\mathbf{w_c}]_2[\mathbf{w_c}]_3\end{aligned} \tag{11}$$

where $[\mathbf{v_c}]_i$ denotes the ith element of vector $\mathbf{v_c}$.

D. Example

Let us return to our earlier example where $P = 255$ and $R = 17$. Now consider a particular RGB input point given by $\mathbf{c} = [230,\ 100,\ 95]^T$. The root cube index is computed using (6) giving

$$[I, J, K]^T = [14,\ 6,\ 5]^T$$

which, using (3) is for the table sample value $\mathbf{c}_{14,\,6,\,5} = [223.1250,\ 95.6250,\ 79.6875]^T$.

The subindexes are given by (8), which gives

$$\mathbf{s_c} = [6.8750,\ 4.3750,\ 15.3125]^T$$

and from (9) and (10), the weights for the interpolation are computed from various products of the values

$$\mathbf{w_c} = [0.4314,\ 0.2745,\ 0.9608]^T$$
$$\mathbf{v_c} = [0.5686,\ 0.7255,\ 0.0392]^T.$$

E. Integer Sampling

If integer values were used in the table construction, then the table is actually nonuniform as mentioned earlier. In this case, the cube index is determined exactly as outlined in Section III-A. The subindexes are computed using

$$\mathbf{r}_{I,J,K} = \mathrm{ROUND}(\mathbf{c}_{I,J,K}) \tag{12}$$
$$\mathbf{s_c} = \mathbf{c} - \mathbf{r}_{I,J,K} \tag{13}$$

and the weights are computed using

$$[\mathbf{w_c}]_1 = \frac{[\mathbf{s_c}]_1}{[\mathbf{r}_{I+1,J,K}]_1 - [\mathbf{r}_{I,J,K}]_1}$$
$$[\mathbf{w_c}]_2 = \frac{[\mathbf{s_c}]_2}{[\mathbf{r}_{I,J+1,K}]_2 - [\mathbf{r}_{I,J,K}]_2}$$
$$[\mathbf{w_c}]_3 = \frac{[\mathbf{s_c}]_3}{[\mathbf{r}_{I,J,K+1}]_3 - [\mathbf{r}_{I,J,K}]_3}$$
$$\mathbf{v_c} = 1 - \mathbf{w_c} \tag{14}$$

where, again, $[\mathbf{s_c}]_i$ denotes the ith element of vector $\mathbf{s_c}$. The above weight calculations account for the nonuniformity of the table. The interpolation is performed using (11).

F. Tetrahedral Interpolation

Tetrahedral interpolation and other methods such as prism and pyramid interpolation, reduce the number of points used in the interpolation calculation [i.e., (11)] by dividing the cube containing the vector $\mathbf{c}$ into subsections. In the case of tetrahedral interpolation, an additional test consisting of three compares is required to determine what subsection contains the vector $\mathbf{c}$. Each subsection uses only four points to compute an interpolation as opposed to the eight points used in the trilinear interpolation. The reader is referred to [6] for additional details.

For all of these interpolation methods, the indexing into the table is performed identically to that of the trilinear interpolation example described in the above section. The difference occurs after this indexing process with the introduction of subindex compares and a computationally simpler replacement for (11).

IV. Efficient Indexing Methods

The indexing problem is to determine the values given by (6), (9), and (10) in the floating point case, or (6) and (14) in the integer case, in a fast efficient manner for an embedded system. The normalizations (divisions) in (9) and (14) are typically prohibitively expensive operations. In addition, if floating point calculations are not available, then care must be taken in scaling the data to achieve minimal error.

TABLE II
Table Sizes Suited for Bit-Wise Shift and Mask Indexing

Table Size R	Step Size ($P = 255$)
3	127.5
5	63.750
9	31.875
17	15.938
33	7.969
65	3.984
127	2.024

A. Bit Shift and Mask

One method for reducing computational cost, is to use the upper bits of the elements of $\mathbf{c}$ for the root index, and the lower bits for the subindexes. This method works when the step size is a power of 2, since simple shifting and masking can be used to compute the indices. Unfortunately, the step size on the range 0 to 255 can never be a power of 2. There are table sizes where the step size is close to a power of 2. These values are given in Table II.

Note that since the step size is not exactly a power of two in these tables, the method of bit-wise shifting and masking will introduce an error.

Typically, the computation is performed as

$$L = \mathrm{ROUND}\left(\frac{P}{R-1}\right) - 1 \tag{15}$$
$$S = \log_2(L+1) \tag{16}$$
$$[I, J, K]^T = \mathbf{c} \gg S \tag{17}$$
$$\mathbf{s_c} = \mathbf{c}\ \&\ L \tag{18}$$

where the values of L and S are precomputed and coded in the algorithm to compute the bit-wise AND operation & in (18) and the bit-wise right SHIFT operation $\gg$ in (17). The weights are computed using

$$\mathbf{w_c} = \frac{\mathbf{s_c}}{L+1} \tag{19}$$
$$\mathbf{v_c} = \frac{(L+1) - \mathbf{s_c}}{L+1}. \tag{20}$$

Note that if $L+1$ is a power of 2, then the normalization (division) is easily implemented with a bit-wise shift operation. If (11) is used for the interpolation, then a final division by $(L+1)^3$ could be performed assuming there is sufficient bits to hold the temporary results. If there is not sufficient bits, multiple shifts will be required.

B. LUT-Based Indexing

An alternative approach to using the bit-wise shift and mask method is to use two 1-D LUTs of size $P+1$ to compute the index values. Note that the same table can be used for each dimension since the MLUT is uniformly sampled. The LUT-based method has the advantage of working with any table size as well as creating less error compared to the mask and shift method. In the case of integer sampled points, the root index table value for input i is given by

$$\begin{aligned} LUT_{IJK}[i] &= \text{FLOOR}\left(\frac{i*(R-1)}{P}\right) i=0,\ldots,P-1 \\ &= R-2 \quad i=P. \end{aligned} \tag{21}$$

The subindex weight table value is given by

$$LUT_{\mathbf{w_c}}[i] = \frac{i - Q[i]}{\text{R}\left((LUT_{IJK}[i]+1)*\frac{P}{R-1}\right) - Q[i]} \quad i=[0,\ldots,P] \tag{22}$$

where

$$Q[i] = \text{R}\left(LUT_{IJK}[i]*\frac{P}{R-1}\right) \tag{23}$$

and $R(.)$ is the rounding operation.

Note that the weights $LUT_{\mathbf{w_c}}[i]$ as expressed above are float values between 0 and 1. In practice, these values would be approximated with a fixed-point representation. This is possible to implement exactly with only a small error within one region as will be shown.

C. Acceleration Methods

There are several acceleration methods for MLUT interpolation discussed in [11, Section 11.4]. These methods are designed to reduce the computational cost of indexing as well as the interpolation in (11). One approach is to make use of a cache or hash coding method to avoid recomputing recently computed values. For an embedded system, it would be necessary to consider the trade-off of the cost of increased memory these methods require, versus the reduction in computational cost. The size of the cache, the statistics of the image data, and the hash coding function will greatly affect the efficiency of these methods. Note that using a cache, or performing hash coding will have no affect on the errors or the efficiency of the indexing method, which is the focus of this paper.

Another class of acceleration methods discussed in [11] takes advantage of the spatial frequency sensitivity properties of the human visual system (HVS). Since the HVS is less sensitive to high frequency chrominance errors compared to luminance errors, methods which save computations at the cost of introducing high frequency chrominance errors are of interest.

To work efficiently, these methods require the input color space to be a chrominance/luminance type such as YCrCb, Kodak YCC, or CIELAB. Embedded systems that process images for Color FAX (CIELAB) or JPEG images (YCC) may use MLUTs that have such a property. Color copy operation in MFPs will typically transform the scanner RGB data to the printer CMY(K) space. In this case, it is not straight forward to take advantage of these acceleration techniques. Since these methods introduce errors in the interpolation calculation, and our focus is errors from indexing, these techniques will not be considered here.

D. Examples

The indexing locations for the above approaches can be easily visualized in the case of $P = 15$ (signal ranges from 0 to 15) with $R = 5$. Fig. 1 displays the float sample points as lines with

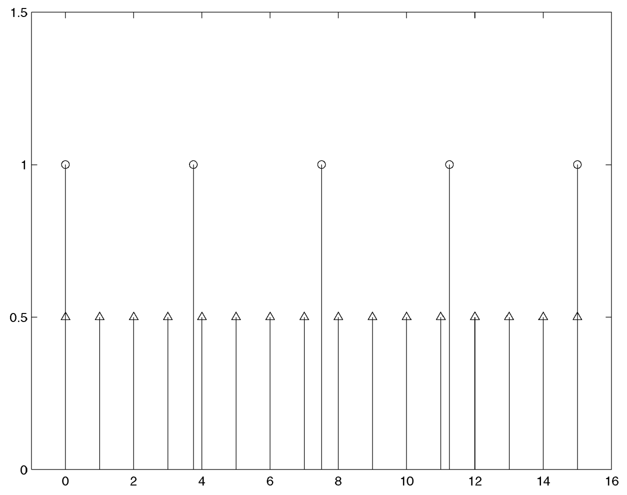
Fig. 1. Circles denote uniform sampling at float values for $P = 15$ and $R = 5$. Arrows denote the values to be interpolated.

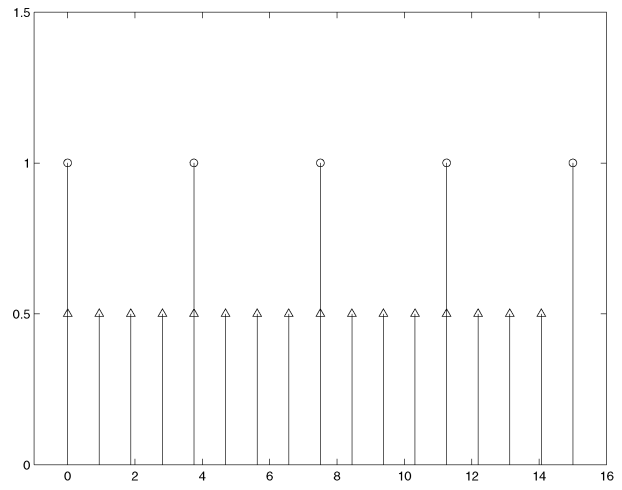
Fig. 2. Circles denote uniform sampling at float values for $P = 15$ and $R = 5$. Arrows denote the values interpolated by bit shift and mask indexing.

circles (these are the sample points of the grid as defined by the ICC). The actual values that are to be interpolated are show as arrows and occur at the integer values 0 to 15. Note that the input values do not fall on the sample points except at 0 and 15.

If the table was generated with float sample points, and bit shift and mask indexing is used, then the interpolated values will be computed as shown by the arrows in Fig. 2. Note that these values are different than the ideal points which occur at the integer values. Also note that the point 15 never occurs.

If the table was instead constructed using integer sample points by rounding the float sample points, then bit shift and mask indexing results in the interpolated values computed as shown in Fig. 3. Again, the point at 15 is missing. This missing point is taken up by a finer interpolation in the range from 8 to 11. A similar finer interpolation occurs in the $P = 255$ $R = 17$ sized table for the range from 128 to 143. This finer interpolation has an advantage in that it keeps the number of segments within each cube the same and allows the same normalization (division) to be used in computing the weights. In other words, the normalization of the weights can be implemented using (19) and (20), which is a division of $L + 1$ (an easy division

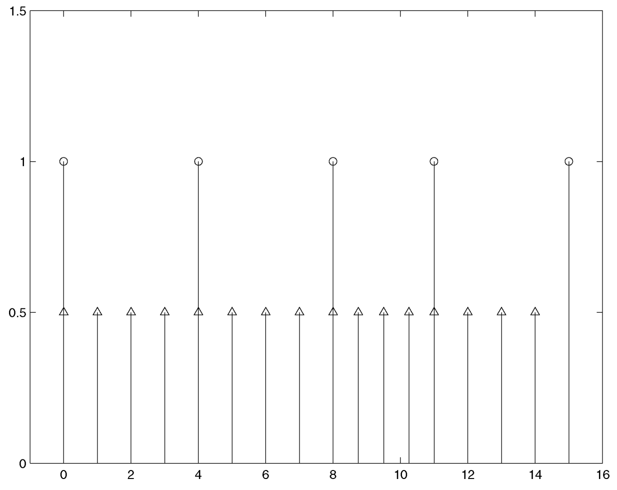

Fig. 3. Circles denote uniform sampling at integer values for $P = 15$ and $R = 5$. Arrows denote the values interpolated by bit shift and mask indexing.

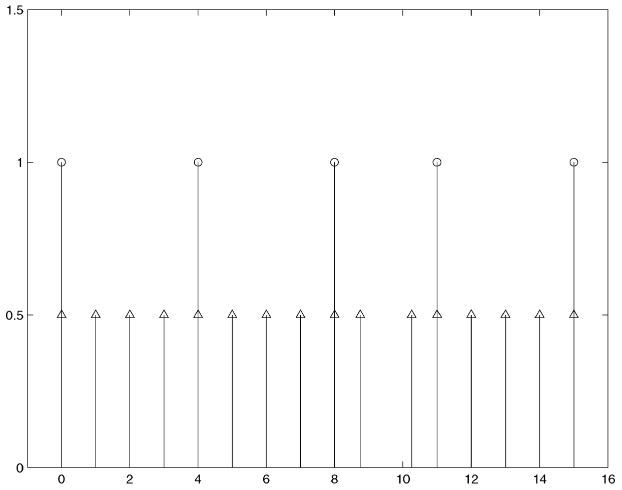

Fig. 4. Circles denote uniform sampling at integer values for $P = 15$ and $R = 5$. Arrows denote the values interpolated by LUT-based indexing.

TABLE III
INDEX LUTS FOR CASE $P = 15$ $R = 5$

i	$LUT_{IJK}[i]$	$LUT_{\mathbf{w_c}}[i]$	$\mathrm{ROUND}(LUT_{\mathbf{w_c}}[i]*4)$
0	0	0	0
1	0	0.25	1
2	0	0.5	2
3	0	0.75	3
4	1	0	0
5	1	0.25	1
6	1	0.5	2
7	1	0.75	3
8	2	0	0
9	2	0.3333	1
10	2	0.6667	3
11	2	1	4
12	3	0.25	1
13	3	0.5	2
14	3	0.75	3
15	3	1	4

TABLE IV
INTERPOLATED VALUES $P = 15$ $R = 5$

Float Point	Integer Shift	LUT Indexing
0.00	0.00	0.00
1.00	1.00	1.00
2.00	2.00	2.00
3.00	3.00	3.00
4.00	4.00	4.00
5.00	5.00	5.00
6.00	6.00	6.00
7.00	7.00	7.00
8.00	8.00	8.00
9.00	8.75	8.75
10.00	9.50	10.25
11.00	10.25	11.00
12.00	11.00	12.00
13.00	12.00	13.00
14.00	13.00	14.00
15.00	14.00	15.00

when R is one of the values in Table II.) The disadvantage of this method is that the value P is never created, and in fact the samples greater than $(P + 1)/2$ have an error. Note that a 1 count error exactly at the black point or white point can often introduce a large visual error, especially when the output is halftoned. Some solutions to this problem in the 8-bit case include the following.

1) If the input is white i.e., [255 255 255], then do not go through the MLUT calculation, simply force the output to be [255 255 255] (white).
2) If the MLUT output is 254, make sure it is mapped to 255 in a 1-D LUT that follows the MLUT operation.

These solutions can introduce artifacts near white.

The LUT-based method of indexing with uniform sampling provides the values shown in Fig. 4. Note that there is an error introduced in the range between 8 and 11. The values for the LUTs are given in Table III, where as mentioned earlier, the float values must be represented with fix point arithmetic. In Table III, the index values are scaled by 4 and rounded. These are the actual values that would be used in the computation of (11), with a final division by 2^6 or a right shift by 6 bits (since there is a product of three of these values that were scaled by 4). The actual values computed by the methods are given in Table IV.

E. Error Comparison

It is interesting to compare the errors of the bit shift and mask indexing method and the fixed-point LUT-based method. Consider again the example where $P = 255$ and $R = 17$. If the float weights are scaled by 2^4, then the only error that occurs is in the range from 128 to 143 due to the limited bits for float representation (similar to the error shown in Fig. 4 for the range 8 to 11). The error is shown in Fig. 5 where note that the error is always less than 0.5.The error for the shift-based approach is shown in Fig. 6. Note that the error is 1 for values greater than or equal to 143. As mentioned earlier, an error at the white point is particularly troublesome in many cases due to the sensitivity of the human visual system to the reference white.

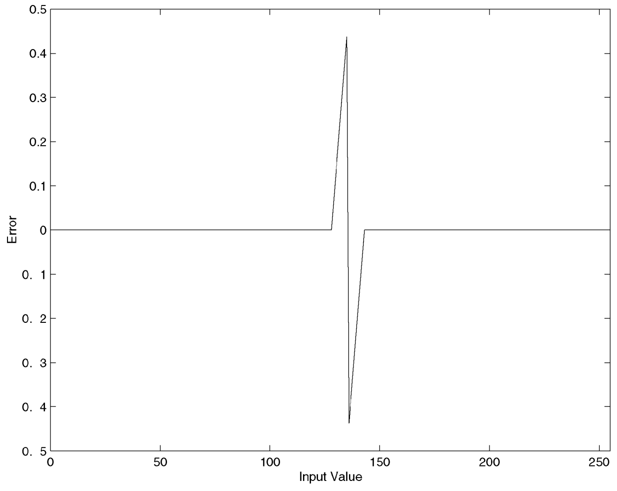

Fig. 5. Fixed-point arithmetic LUT-based error for $P = 255$ and $R = 17$.

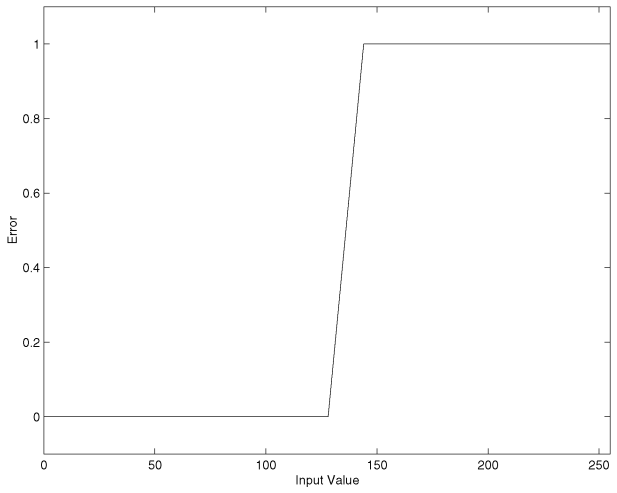

Fig. 6. Fixed-point arithmetic Shift-based error for $P = 255$ and $R = 17$.

V. Computational Cost

The exact computational cost depends upon the types of operations and memory access that can be performed in parallel. Computationally, the only difference between the LUT-based method and the bit shift and mask method is the computation of the weight vectors $\mathbf{w_c}$, $\mathbf{v_c}$ and the cube index location $[I, J, K]^T$. Given an input vector $\mathbf{c}$ and $P = 255$, $R = 17$, the LUT-based approach performs the calculation

$$\begin{aligned} I &= LUT_{IJK}[[\mathbf{c}]_1] \\ J &= LUT_{IJK}[[\mathbf{c}]_2] \quad K = LUT_{IJK}[[\mathbf{c}]_3] \end{aligned} \tag{24}$$

$$\begin{aligned} [\mathbf{w_c}]_1 &= LUT_{\mathbf{w_c}}[[\mathbf{c}]_1] \\ [\mathbf{w_c}]_2 &= LUT_{\mathbf{w_c}}[[\mathbf{c}]_2] \quad [\mathbf{w_c}]_3 = LUT_{\mathbf{w_c}}[[\mathbf{c}]_3] \end{aligned} \tag{25}$$

$$\begin{aligned} [\mathbf{v_c}]_1 &= 16 - [\mathbf{w_c}]_1 \\ [\mathbf{v_c}]_2 &= 16 - [\mathbf{w_c}]_2 \quad [\mathbf{v_c}]_3 = 16 - [\mathbf{w_c}]_3 \end{aligned} \tag{26}$$

and the shift-based method performs the calculations

$$\begin{aligned} I &= [\mathbf{c}]_1 \gg 4 \\ J &= [\mathbf{c}]_2 \gg 4 \quad K = [\mathbf{c}]_3 \gg 4 \end{aligned} \tag{27}$$

$$\begin{aligned} [\mathbf{w_c}]_1 &= [\mathbf{c}]_1 \& 0\mathrm{Fh} \\ [\mathbf{w_c}]_2 &= [\mathbf{c}]_2 \& 0\mathrm{Fh} \quad [\mathbf{c}]_3 \& 0\mathrm{Fh} \end{aligned} \tag{28}$$

$$\begin{aligned} [\mathbf{v_c}]_1 &= 16 - [\mathbf{w_c}]_1 \\ [\mathbf{v_c}]_2 &= 16 - [\mathbf{w_c}]_2 \quad [\mathbf{v_c}]_3 = 16 - [\mathbf{w_c}]_3. \end{aligned} \tag{29}$$

TABLE V
Computational Cost of Indexing Methods

Method	MULT	ADD	SHIFT	AND	LOOK-UP
LUT-Based	0	3	0	0	6
Shift-Based	0	3	3	3	0

The number of arithmetic operations required by each of these methods is given in Table V. As mentioned, the computational cost of these two methods will depend upon the exact architecture of the hardware (e.g., number of MACs, ALUs, and parallel memory banks). Looking at Table V, neither algorithm appears to have a significant computational advantage.

VI. Color Example

Since our application is color imaging, we will look at the color error introduced by these indexing methods. Consider the problem of transforming from sRGB to CIELAB. The transformation is known analytically, and is given in the Appendix for completeness. One MLUT of size $R = 33$ was created to approximate the sRGB to CIELAB mapping. The interpolation of the MLUT values was implemented with integer sample points in sRGB space and indexed as follows:

1) indexed using the method described in Section III-E, which we will denote as IDEAL_INDEX;
2) indexed using the method described in Section IV-A, which we will denote as SHIFT_INDEX;
3) indexed using the method described in Section IV-B, which we will denote as LUT_INDEX.

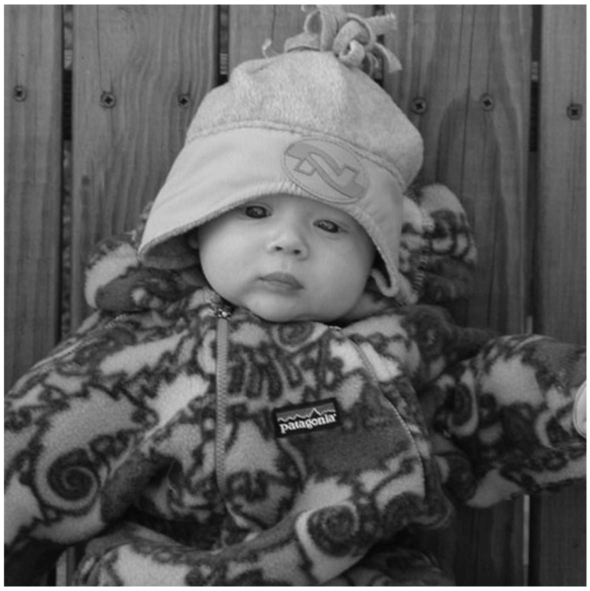

Fig. 7. Input sRGB image.

The sRGB image shown in Fig. 7 was operated on by the table using the above indexing methods. This operation created three CIELAB images. The ΔE^*_{94} color difference was computed between these images and the analytically transformed image

TABLE VI
AVERAGE ΔE^*_{94} ERRORS, MAPPING TO CIELAB

Method	Starting color space			
	sRGB	$\gamma = 5$	$\gamma = 7.5$	$\gamma = 10$
IDEAL_INDEX	0.029	0.057	0.060	0.056
SHIFT_INDEX	0.175	0.250	0.249	0.194
LUT_INDEX	0.040	0.068	0.065	0.057

TABLE VII
MAXIMUM ΔE^*_{94} ERRORS, MAPPING TO CIELAB

Method	Starting color space			
	sRGB	$\gamma = 5$	$\gamma = 7.5$	$\gamma = 10$
IDEAL_INDEX	0.289	0.230	0.275	0.363
SHIFT_INDEX	0.593	0.745	1.050	1.363
LUT_INDEX	0.470	0.586	0.468	0.375

for each pixel. The definition of ΔE^*_{94} is given in the Appendix . To reduce errors from nonindexing sources, the table values were double precision floating point CIELAB values and floating point operations were used to calculate (11).

The magnitude of error is highly dependent upon the nonlinearity that is approximated by the MLUT. To demonstrate this, consider a mapping from RGB to CIEXYZ given by

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix} \tag{30}$$

where r, g, and b are computed using (assuming an 8-bit range on the input RGB values)

$$r = \left(\frac{R}{255}\right)^{\gamma} \tag{31}$$
$$g = \left(\frac{G}{255}\right)^{\gamma} \tag{32}$$
$$b = \left(\frac{B}{255}\right)^{\gamma}. \tag{33}$$

Tables VI and VII contain the ΔE^*_{94} errors for the three methods for several values of γ and for the case of sRGB, which has a gamma of approximately 2.4. Note that as the nonlinearity increases, the maximum error introduced by the shift-based method increases significantly.

VII. SUMMARY

The mathematics of MLUT interpolation were reviewed. A comparison was made between a LUT-based indexing method and a bit shift/mask method. The LUT-based method resulted in less error with no additional computational cost. In addition, the errors in the LUT-based method resulted in less visually noticeable artifacts compared to the bit shift/mask based method.

APPENDIX

A. sRGB to CIELAB

The transformation from sRGB [2] to CIELAB is as follows. First transform from sRGB to CIEXYZ using

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix} \tag{34}$$

where r is computed using (assuming an 8-bit range on the input sRGB values)

$$r = \begin{cases} \left(\frac{R}{255}\right) * 12.92 & \left(\frac{R}{255}\right) \le 0.040\,45 \\ \left(\frac{\left(\left(\frac{R}{255}\right)+0.055\right)}{1.055}\right)^{2.4} & \text{else} \end{cases} \tag{35}$$

and g and b are similarly computed. Given a computed $[XYZ]$ value, the transformation to CIELAB is given by

$$L^* = 116\left(\frac{Y}{Y_n}\right)^{1/3} - 16 \tag{36}$$
$$a^* = 500\left[\left(\frac{X}{X_n}\right)^{1/3} - \left(\frac{Y}{Y_n}\right)^{1/3}\right] \tag{37}$$
$$b^* = 200\left[\left(\frac{Y}{Y_n}\right)^{1/3} - \left(\frac{Z}{Z_n}\right)^{1/3}\right] \tag{38}$$

for $X/X_n, Y/Y_n, Z/Z_n > 0.01$. Since the assumed white point of the sRGB color space is D65, $[X_nY_nZ_n] = [0.3127, 0.3290, 0.3583]$.

B. ΔE^*_{94} Color Difference Metric

The CIE has updated ΔE^*_{ab} with a new weighted version, which is designated ΔE^*_{94}[12]. The new measure weights the hue and chroma components in the ΔE^*_{ab} measure by a function of chroma. Specifically, given two CIELAB values $[L_1, a^*_1, b^*_1]$ and $[L_2, a^*_2, b^*_2]$, the measure is given by

$$\Delta E^*_{94} = \sqrt{\left(\frac{\Delta L^*}{k_L S_L}\right)^2 + \left(\frac{\Delta C^*_{ab}}{k_C S_C}\right)^2 + \left(\frac{\Delta H^*_{ab}}{k_H S_H}\right)^2} \tag{39}$$

where

$$\begin{aligned} S_L &= 1 \\ S_C &= 1 + 0.045\, C^*_{ab} \\ S_H &= 1 + 0.015\, C^*_{ab} \end{aligned} \tag{40}$$
$$C^*_{ab1} = \sqrt{(a^*_1)^2 + (b^*_1)^2} \tag{41}$$
$$\Delta L^* = \sqrt{(L_1 - L_2)^2} \tag{42}$$
$$\Delta C^*_{ab} = \sqrt{(C^*_{ab1} - C^*_{ab2})^2} \tag{43}$$
$$\Delta H^*_{ab} = \sqrt{(\Delta E^*_{ab})^2 - (\Delta L^*)^2 - (\Delta C^*_{ab})^2} \tag{44}$$
$$\Delta E^*_{ab} = \sqrt{(L_1 - L_2)^2 + (a^*_1 - a^*_2)^2 + (b^*_1 - b^*_2)^2} \tag{45}$$

and typically the weighting values are set such that

$$k_L = k_C = k_H = 1. \tag{46}$$

The reference color C^*_{ab} is used in (40). If neither color is the reference, then the geometric mean of C^*_{ab1} and C^*_{ab2} is used.

References

[1] M. J. Vrhel and H. J. Trussell, "Color device calibration: A mathematical formulation," *IEEE Trans. Image Processing*, vol. 8, pp. 1796–1806, Dec. 1999.

[2] M. Anderson, R. Motta, S. Chandrasekar, and M. Stokes, "Proposal for a standard default color space for the internet–sRGB," in *Proc. IS&T/SID 4th Color Imaging Conf.: Color Science, Systems, and Applications*, Nov. 1996, pp. 238–246.

[3] W. F. Schreiber, "A color pre-press system using appearance variables," *J. Imag. Technol.*, vol. 17, no. 4, pp. 200–211, Aug. 1986.

[4] M. C. Stone, W. B. Cowan, and J. C. Beatty, "Color gamut mapping and the printing of digital color images," *ACM Trans. Graph.*, vol. 7, no. 3, Oct. 1988.

[5] P. Hung, "Colorimetric calibration in electronic image devices using a look-up-table model and interpolations," *J. Electron. Imag.*, vol. 2, pp. 53–61, 1993.

[6] H. R. Kang, *Color Technology for Electronic Devices*. Bellingham, WA: SPIE, 1997.

[7] "File Format for Color Profiles," International Color Consortium, Specification ICC.1:2001–12.

[8] A. U. Agar and J. P. Allebach, "A minimax method for function interpolation using an SLI structure," in *Proc. ICIP* , vol. 1, 1997, pp. 671–674.

[9] J. Z. Chang, J. P. Allebach, and C. A. Bouman, "Sequential linear interpolation of multidimensional functions," *IEEE Trans. Image Processing*, vol. 6, pp. 1231–1245, Sept. 1997.

[10] R. Bala, "Device characterization," in *Digital Color Imaging Handbook*, G. Sharma, Ed. Boca Rotan, FL: CRC, 2003, ch. 5.

[11] R. Bala and R. V. Klassen, "Efficient color transformation implementation," in *Digital Color Imaging Handbook*, G. Sharma, Ed. Boca Rotan, FL: CRC, 2003, ch. 11.

[12] "Industrial Color Difference Evaluation," CIE, Tech. Rep. 116–1995, 1995.

Michael J. Vrhel (SM'93) received the B.S. degree in electrical engineering from Michigan Technological University, Houghton, in 1987 and the M.S. and Ph.D. degrees from North Carolina State University, Raleigh, in 1989 and 1993, respectively.

From 1993 to 1996, he was a National Research Council Research Associate at the National Institutes of Health (NIH), Bethesda, MD, where he researched biomedical image and signal processing problems. From 1997 to 2002, he was the Senior Scientist at Color Savvy Systems Limited, Springboro, OH, where he developed color device characterization software and low-cost color measuring instrumentation. Since 2002, he has been the Senior Scientist at ViewAhead Technology, Redmond, WA. His current research interests include color imaging and measurement, device characterization, and efficient algorithms.

Dr. Vrhel is a member of the SPIE and is currently serving as a Guest Editor for the IEEE SIGNAL PROCESSING MAGAZINE, Special Issue on Color Image Processing.