

Binary Proportional Interpolation for Color Space Conversion

Steven F. Weed ▲ and Tomasz J. Cholewo

Lexmark International, Inc. Software Research, Lexington, Kentucky, USA

An algorithm which exploits binary data representation for efficiently interpolating lookup tables (LUTs) is developed. Some implementation cost and accuracy trade-offs among trilinear, tetrahedral and binary proportional interpolation (BPI) when constrained to no more than one LUT access per pixel are compared for color space conversion of digital images. A simpler sample dither implementation of BPI, called Neighborhood Mask Dither Interpolation (NMDI), is also described.

Journal of Imaging Science and Technology 47: 525–530 (2003)

Introduction

Interpolation among sparse table entries is venerable technology. Often employed for color space conversions,¹ with printing as the predominant application, some video display technologies such as LCDs also require relatively complex processing for good color rendition. Color space representations with higher information coding density than RGB are also candidates for conversion by table interpolation. Digital processing costs continue to decrease, but simpler and faster color conversion methods with good quality could accelerate the acceptance of advanced color devices and interchange representations.²

There are many applications for which color conversions by other methods, including matrix multiplication, neural networks, Neugebauer or other algebraic equations³ can be digitally implemented for a required accuracy with better efficiency than conventional interpolations with multidimensional LUTs. For applications where use of LUTs is indicated, a wealth of optimization techniques is available.⁴ This article examines other approaches for increased efficiency using LUTs.

LUTs capture relationships between perceived colors and control values for processes such as printing which often defy terse analytical definition and may also change over time. Sparse LUTs work well when these relationships can be approximated as linear among adjacent table entries. For custom digital processing, in-

terpolation speed is ultimately constrained by access times for LUT entries stored in random access memory. Faster memory supports higher pixel conversion rates, but reducing LUT accesses per conversion is more efficient. Algorithms based upon relatively few primitive logic functions with simple implementations in ASIC and FPGA random logic (such as AND, OR, XOR, bit shifts, addition and subtraction) can be more economical than algorithms requiring general purpose or digital signal processor cores.

Recently proposed spectral based and principal component image representations raise additional challenges for geometrically-based color space interpolations. For example, n -linear interpolation involves 2^n LUT entries per conversion, while n -dimensional simplex interpolation involves a somewhat cumbersome task of selecting the correct set of $n + 1$ from 2^n LUT entries. Of course, selected LUT entries need not be accessed if associated interpolation weights are zero. Zero weights correspond to reduction in dimensionality of interpolation, for example when an input value resolves to a single LUT entry or lies along an axis between two LUT entries in a multidimensional table.⁵ As with n -linear interpolation, Binary Proportional Interpolation (BPI) assigns a weight for each of 2^n LUT entries. However, the number of non-zero LUT entry weights will be no more than one (corresponding to an all-zero bit tuple for the base index) plus the bit precision of interpolation, since that is the maximum number of unique bit tuples which can increment base indices.⁶

Linear Interpolations and Trichromatic Color Tables

Characterizing relatively complicated numeric relationships using sparse tables is well understood. Although variable grid spacing has been demonstrated to reduce color table size for a given error tolerance,² integer tables with grid points uniformly distributed in input space allow for simpler implementations. Linear interpolation accuracy deteriorates when local relationships are not approximately linear. Splines improve interpolation accuracy for sparser tables, but each three-dimensional cubic spline interpolation requires 32 LUT accesses.

Original manuscript received February 3, 2003

▲ IS&T Member

phone: (859) 825-4377; fax: (859) 232-7345; email: weed@lexmark.com

Color Plates 13–16 are printed in the color plate section of this issue, pp. 586–603.

Supplemental Materials—An Appendix containing Figures A1 and A2 can be found on the IS&T website (www.imaging.org) for a period of no less than two years from the date of publication.

©2003, IS&T—The Society for Imaging Science and Technology

```

for (c = 0; c < numcolors; c++)
    colorant[c] = 0; /* interpolate within unit cube of LUT */

for(t = 8; t; t/=2) { /* t is bit mask and interpolation weight */
    for (c = 0; c < 3; c++) {
        cmy[c] = 255 - rgb[c];
        bit = t & cmy[c]; /* test masked fraction bit */
        cmy[c] = cmy[c]/16; /* base index */
        if (bit)
            cmy[c]++; /* increment index */
    }
    for (c = 0; c < numcolors; c++)
        colorant[c] += t * lut[cmy[0]][cmy[1]][cmy[2]][c]; /* lut access */
}

for (c = 0; c < 3; c++) {
    cmy[c] = (255 - rgb[c])/16; /* LUT unit cube origin */
}
for (c = 0; c < numcolors; c++) {
    colorant[c] = (lut[cmy[0]][cmy[1]][cmy[2]][c] + colorant[c] + 8) / 16;
}

```

Figure 1. Pseudo-code for 4-bit binary proportional interpolation of RGB to CMYK

Integer processing for final LUT entry intervals deserves special consideration. Consider 4-bit integer interpolation of 8-bit values. 4-bit interpolation fractions leave 4 bit indices to select among 16 intervals, requiring 17 LUT entries. For some applications, it may be desirable to rescale 8-bit inputs for power of two table dimensions with more compact binary addressing. However, round-off of rescaled inputs and reduction of table size sacrifice precision. Without input rescaling, final LUT entries cannot be directly indexed. Consider a perfectly linear (identity) sparse 4-bit table with final interval values 240 and 255. With appropriate rounding, an input of 255 can be interpolated to a result of 255, but 248 will interpolate to 247. While the difference between 247/255 and 248/255 dot coverage may not be perceptible for printing, a difference between 7/255 and 8/255 is more likely problematic. Consequently, it is generally desirable to concentrate smaller table values nearer the origin. When table values represent subtractive device colorants, this requires inverting additive inputs such as RGB or CIEXYZ before generating interpolation fractions and table indices.

Other approaches for handling final table intervals with 8-bit values include treating final intervals as one unit shorter than others or rescaling final table entries. However, table value rescaling may require values outside the valid range. For the example above, a rescaled final table value of 256 exceeds the range of a byte and would require increased memory to be allocated. In some closed systems, integer interpolation may be employed without special treatment for final intervals by appropriate compensations in subsequent processing, such as one-dimensional LUTs or halftone threshold array values.

Table interpolation proceeds by extracting LUT indices from input values, selecting a nearest set of table entries, then applying weights to selected entries. For example, consider a two-dimensional table T representing a function of arguments X and Y . Values X and Y are converted to grid indices x, y and interpolation fractions $0 \leq a, b \leq 1$, respectively.

Two-dimensional *simplex interpolation* (triangular interpolation) uses only three LUT entries surrounding the interpolated point. If $a > b$:

$$f(X, Y) = (1 - a)T_{x,y} + (a - b)T_{x+1,y} + bT_{x+1,y+1}, \quad (1)$$

else, for $a \leq b$:

$$f(X, Y) = (1 - b)T_{x,y} + (b - a)T_{x,y+1} + aT_{x+1,y+1}. \quad (2)$$

Two-dimensional *n-linear interpolation* (bilinear interpolation) calculates

$$\begin{aligned} f(X, Y) &= (1 - a)(1 - b)T_{x,y} + a(1 - b)T_{x+1,y} \\ &+ (1 - a)bT_{x,y+1} + abT_{x+1,y+1}. \end{aligned} \quad (3)$$

While triangular interpolation requires numerically ranking fractions, BPI implicitly assigns powers of two weights according to bit significance within fractions. A two-dimensional BPI with 4-bit fractions a and b calculates:

$$\begin{aligned} f(X, Y) &= \\ &(1/16 + (\bar{a} \& \bar{b}))T_{x,y} + (a \& \bar{b})T_{x+1,y} \\ &+ (\bar{a} \& b)T_{x,y+1} + (a \& b)T_{x+1,y+1}. \end{aligned} \quad (4)$$

where \bar{a} represents bit-wise logical negation of a , and $\&$ represents bit-wise logical AND.

Although this article focuses on three-dimensional interpolation, the above equations can be generalized to higher dimensions. Ranking fractions for simplex interpolation becomes increasingly expensive in higher dimensions, while BPI complexity increases with interpolation precision rather than with input dimension. Being fundamentally geometric (albeit with iterations for ranking and selection), simplex interpolation is readily portrayed graphically. BPI is an inherently binary procedure, more naturally illustrated by table or pseudo-code. Comparable integer pseudo-codes for BPI and tetrahedral interpolation, both interpolating for the four least significant bits of 24-bit RGB, are shown in Figs. 1 and 2.

Like simplex interpolation, BPI typically accesses fewer LUT entries than n -linear interpolation, with an

```

/* rank the four LSBs of each trichromatic input component */
for (c = 0; c < 3; c++)
    cmy[c] = 0x0F & (255 - rgb[c]);

if (cmy[0] < cmy[1])
    axis = 1;
else
    axis = 0;
minor = 1 - axis;
if (cmy[axis] < cmy[2])
    axis = 2;
/* cmy[axis] is now >= other components */
major = 3 - (minor + axis);
if (cmy[minor] < cmy[major]) {
    minor = major;
    major = 3 - (minor + axis);
}
/* cmy[major] is now <= other components */
for (c = 0; c < 3; c++) {
    cmy[c] = (255 - rgb[c]) / 16;    /* tetrahedron origin by truncation */
    if (axis == c)
        cmyaxis[c] = cmy[c] + 1;    /* axis vertex */
    else
        cmyaxis[c] = cmy[c];
    if (major != c)
        cmyminor[c] = cmy[c] + 1;    /* minor diagonal vertex */
    else
        cmyminor[c] = cmy[c];
}
wmajor = cmy[major];                /* major diagonal (neutral) weight */
wminor = cmy[minor] - cmy[major];    /* minor (rgb) diagonal weight */
waxis = cmy[axis] - cmy[minor];      /* axis (cmy) weight */
worigin = 16 - cmy[axis];            /* origin weight */
for (c = 0; c < numcolors; c++) {
    x = wmajor * lut[cmy[0] + 1 ][cmy[1] + 1 ][cmy[2] + 1 ][c];
    x += wminor * lut[cmyminor[0]][cmyminor[1]][cmyminor[2]][c];
    x += waxis * lut[cmyaxis[0] ][cmyaxis[1] ][cmyaxis[2] ][c];
    x += worigin * lut[cmy[0]    ][cmy[1]    ][cmy[2]    ][c];
    colorant[c] = (x + 8) / 16;
}

```

Figure 2. Pseudo-code for 4-bit tetrahedral interpolation from RGB to CMYK.

upper bound of one more access than the bit precision of interpolation. As fewer entries are used for each interpolation, noise in individual LUT entries becomes more disruptive. LUT entries directly inverted from data measurements by numerical methods are inherently more noisy than tables generated from models. Since it is an infrequent offline activity, using more sophisticated LUT generation algorithms⁷ to maintain quality with fewer accesses per conversion is an easy trade-off.

Color information is often represented by three eight-bit bytes, one for each trichromatic channel. Since humans can visually discern roughly 100 increments along each trichromatic axis, quantizing to more than twice that precision supports the illusion of continuously varying colors. A 24-bit color space encoding can be directly converted by a LUT with 2^{24} entries, but RAM storage for 50-megabyte tables is still considered extravagant. Many color spaces are defined with explicit and separable algebraic conversions, but these conversions of-

ten involve mathematical functions for which sufficiently fast and accurate digital implementations remain too costly for commodity deployment. Sparse LUTs with integer interpolation remain a viable option.

For a number of trichromatic color spaces, including sRGB and CIELab, $17 \times 17 \times 17$ LUTs are common. Starting with 8 bits for each trichromatic component value, an interpolation unit cube is selected by truncating each value to its 4 most significant bits. Trilinear interpolation is popular for trichromatic color spaces, but requires access to eight lookup table entries for each conversion. Tetrahedral interpolation and BPI are relatively disadvantaged when interpolating near-neutrals from a space such as CIELab, where neutrals can lie along a LUT axis (Fig. 3). On the other hand, Fig. A1, (*found in the Appendix published as Supplemental Material on the IS&T website (www.imaging.org) for a period of no less than two years from the date of publication*) shows errors of noisy gray

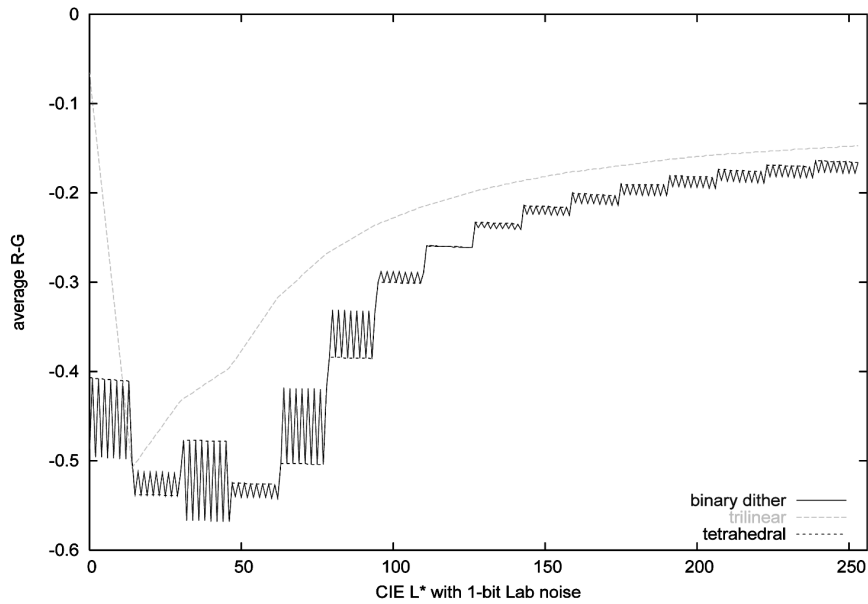


Figure 3. Noisy gray interpolation from 24-bit CIELab to floating point sRGB.

```

BYTE mask[] = {8,4,8,2,8,4,8,1,8,4,8,2,8,4,8,0};
#define DITHER( x, m ) ( ((x)>>BITS) + (((x)&(m)) ? 1 : 0) )

i = i % sizeof(mask);
k = mask[i++];
C = DITHER(255-*iptr++, k);
M = DITHER(255-*iptr++, k);
Y = DITHER(255-*iptr++, k);
lutptr = LUT[C][M][Y];
for (c = 0; c < numcolors; c++) {
    colorant[c] = lutptr[c]
}

```

Figure 4. Pseudo-code for 4-bit neighborhood mask dither interpolation from RGB to CMYK.

interpolation from 24-bit sRGB to floating point CIELab. Trilinear interpolation can generate anomalous results, for example when calculating near-neutral values along diagonals in RGB-like spaces. These plots were generated by averaging results from inputs uniformly distributed around neutral values. Inputs average to neutral, and outputs linearly proportional to inputs should ideally also average to neutral. The relationship between sRGB and CIELab is non-linear, and these plots illustrate relative robustness of different linear interpolations to nonlinearity and quantization. Slightly off-diagonal trilinear interpolations are relatively more perturbed by non-linear differences among non-diagonal corner contributions, while off-axis tetrahedral interpolations are more perturbed by diagonal corner contributions.

Tradeoffs between trilinear and tetrahedral interpolations are well understood. Trilinear interpolation incurs consistently higher memory access requirements with inferior rendition of near-neutral colors from RGB.⁸ Since BPI behaves more similarly to tetrahedral, subsequent comparisons concentrate on them.

Binary Proportional Interpolation

While tetrahedral interpolation uses no more than 4 LUT entries, four-bit BPI uses no more than 5 LUT entries for 3 or more dimensions.

Consider hexadecimal 24-bit RGB color value {0xC8, 0x64, 0x96}. For lookup table T with 17^3 entries, the most significant 4 bits of each input component are used as indices for an interpolation unit cube; in this case its origin is $T_{C,6,9}$, Fig. A2, (found in the Appendix published as Supplemental Material on the IS&T website (www.imaging.org) for a period of no less than two years from the date of publication). The least significant 4 bits (8,4,6) of input components are employed as interpolation fractions within this sub-cube.

Tetrahedral weights¹ for this color are 4, 2, 2, and 8. Red is the largest fraction, so the tetrahedron's LUT axis is to $T_{D,6,9}$. The median fraction is blue, so the minor diagonal is to $T_{D,6,A}$. Therefore the tetrahedral calculation is:

$$(4T_{D,7,A} + 2T_{D,6,A} + 2T_{D,6,9} + 8T_{C,6,9})/16. \quad (5)$$

TABLE I. BDI Processing for Example in Text

BPI weights	truncated inputs			BPI LUT vertices
	8	4	6	
8	1	0	0	$T_{D,6,9}$
4	0	1	1	$T_{C,7,A}$
2	0	0	1	$T_{C,6,A}$
1	0	0	0	$T_{C,6,9}$

In 4-bit BPI, enumerated bit weights sum to 15/16, with an additional 1/16 always contributed by the origin. For each significant weight, a corresponding bit is added to its component of the origin index. Thus the BPI calculation for the same color is:

$$(8T_{D,6,9} + 4T_{C,7,A} + 2T_{C,6,A} + T_{C,6,9} + T_{C,6,9})/16. \quad (6)$$

Table I summarizes BPI processing for this example:

For another example, trichromatic binary fractions 1100, 1100, 1100 need only access table entries, $T_{C,6,9}$ and $T_{D,7,A}$, with weights 4 and 12 respectively.

Table II compares processing operations for these 4-bit trichromatic interpolations, excluding LUT index generation. Maximum table accesses are typically important for embedded applications; average accesses better predict application performance on general purpose computers.

Referring again to Fig. 3 and Fig. 4, systematic interpolation artifacts can be visible as periodic color bands in noisy near-neutral gradients. Since one-bit changes in input values can cause BPI to select substantially different LUT entries, its periodic error structure is of consistently higher frequencies than n -linear and n -simplex. Whether they are immediately more visible depends upon the rate of change in samples being interpolated. Our experience is that interpolation artifacts are typically more noticeable in areas of slowly changing color. Interpolation artifacts in images rendered using BPI and tetrahedral are typically more prominent than when using trilinear. Whether this is simply a consequence of more LUT entries contributing to n -linear solutions requires further investigation.

Single LUT Access per Pixel

Constrained to no more than one LUT access per pixel, uncached color conversion becomes a quantized sampling process called dither. Depending upon other system behavior (including human perception) for averaging or smoothing, dithering simplifies conversion by alternating selections among table entries according to some estimate of proximity to the input. Dither techniques are distinguished by their selection methods. Error diffusion is a well regarded dithering technique, albeit computationally intensive. A simpler method adds zero mean pseudo-random values to inputs before quantizing to table indices.⁹ When used in conjunction with any other sampling process, aliasing artifacts can result. Of course, aliasing against high frequency content in the image is also possible. For example, the image of leaves converted with Neighborhood Mask Dither Interpolation NMDI (**Color Plate 13, p. 591**) may show moire patterns, depending upon how it is rendered in print. Similar effects are to be expected with other dither interpolations. When luminance-chrominance color inputs are available, interpolation in luminance with chrominance dither has been shown to be effective.²

TABLE II. Processing Operations

Operations	trilinear	tetrahedral	BPI
maximum table accesses	8	4	5
average table accesses	6.59	3.64	3.90
multiplications	36	22	15
additions and subtractions	24	34	12
ANDs and NOTs	0	0	15
bit shifts	3x12-bit	3x4-bit	3x4-bit

Although more complicated to implement than interpolations, caching is a popular technique for reducing accesses to mass storage. Empirically, a cache of 8 recently-used entries was found to be adequate for color conversion for BPI or tetrahedral interpolation constrained to single LUT access per input when used with halftoning by threshold array (another spatial sampling process). Conversion accuracy is affected relatively weakly by cache size (for example 3.8 RMS for cache size 4, 2.35 for cache size 8 and 1.95 for cache size 32 with a relatively challenging image), so a key consideration is substantially caching multiple colors to minimize moire with halftones. Even though it does not have appreciably larger RMS errors for the same cache size, we recommend a minimum cache size of 16 with cached trilinear interpolation for moire considerations. The missing LUT entry (if any) with largest interpolation weight for each pixel updates the cache. BPI weights are implicitly ordered, with largest weight at least 8/16. Tetrahedral interpolation requires an additional sorting of interpolation weights to prioritize cache updates. The largest weight for tetrahedral interpolation can be as small as 4/16, with a correspondingly larger worst case errors for identity interpolation. Consistent with predicted table accesses, 4-bit trichromatic BPI averages more cache misses than tetrahedral interpolation.

For single clock cycle execution, simple hardware caching may employ a suboptimal strategy such as preselecting a cache entry to be replaced before determining whether that entry is used for the current pixel. Magnified pixels with color errors from LUT cache misses are shown in **Color Plate 14(b) and 14(c), p. 591**. Since relatively small differences may not be apparent in print, color value difference differences (**Color Plate 15, p. 592**) are multiplied by 16.

For this example, with LUT access restricted to no more than one per pixel, BPI reported 44 unavailable LUT entries, while tetrahedral reported 25 cache misses. The largest primary color differences were 7/255 for BPI and 9/255 for tetrahedral, both in red. The largest average absolute pixel color differences were 0.63/255 in red for BPI and 0.60/255 in blue for tetrahedral. Eliminating cache misses reduced BPI average absolute difference to 0.15/255 in blue. Cache misses have another consequence: weights generally do not sum to a power of two. This in turn precludes simple bit-shifting to normalize results. Cached 4-bit BPI requires the equivalent of division by integer values from 8 to 16, while cached 4-bit tetrahedral interpolation denominators range from 4 to 16. These can be approximated by integer multiplications followed by bit-shifting, but the problem is more severe for cached 4-bit trilinear interpolation, with denominators from 512 to 4096. Despite these complications, color conversion by cached trilinear interpolation has been observed to be more robust than BPI and tetrahedral in some instances. Relative robustness of interpolations deserves more research.

Unlike tetrahedral interpolation, BPI can also be implemented as a neighborhood dither, constrained to a single LUT access per pixel, by the use of a sample bit mask:

$$\begin{pmatrix} 8 & 2 & 8 & 4 \\ 4 & 8 & 0 & 8 \\ 8 & 4 & 8 & 2 \\ 1 & 8 & 4 & 8 \end{pmatrix}$$

This bit mask tiles over samples for neighborhood mask dither interpolation. Sample component values become base LUT indices and interpolation fractions. Base indices are incremented when corresponding mask and fraction bits for a component are both 1. Resulting indices select one LUT entry for each pixel. This NMDI algorithm can be implemented in fewer than 100 logic gates. The relative frequencies of mask values determine neighborhood interpolation weights, so that more significant fraction bits have double the weight of next less significant bits by being sampled twice as often. Figure 4 is pseudocode for NMDI. An n -bit BPI converges to mean value over 2^n input samples. Spatial mask size for periodically refreshed processes can be reduced by temporal dither.

Results for NMDI followed by error diffusion have been very promising. Perturbation of colorant values by NMDI appreciably reduces the severity of “worm” artifacts generated by some error diffusion halftone algorithms. No dither occurs when truncated input bits are all zero, since conversion is exact.

Dithering of individual pixels is evident at extremely low spatial resolutions (**Color Plate 14, p. 591**) Unlike cached interpolations, where errors are concentrated in areas of rapidly changing color (**Color Plate 16(a) and 16(b), p. 592**), distribution of NMDI errors (**Color Plate 16(c), p. 592**) is relatively uniform.

As is the case with other table dithering techniques, NMDI has applications beyond color display and printing. Control and data conversion in applications such as automotive engine management, HVAC and many industrial processes involving repetitive sampling of multiple inputs are candidates for table dithering when convergence to mean value is required only for local intervals.


Although statistically significant user preference data has yet to be collected, experienced inkjet developers have rated typical test pages rendered by NMDI and error diffusion at 600 dpi and higher over trilinear

and error diffusion, where differences can be discerned, while processing time is reduced by about 30% for a prototype software filter to read, color convert, error diffuse and format images for host-based inkjet printing.

Conclusions

Neighborhood Mask Dither Interpolation with error diffusion halftoning applied to inkjet printing at 600 dpi causes no discernible artifact exacerbation and reduces processing time for software based interpolation.

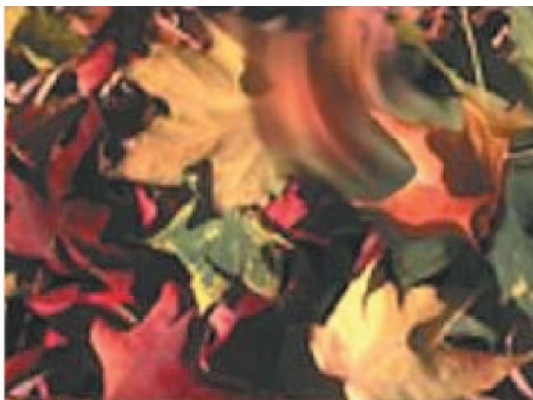
Practical embedded applications include color conversions for error diffused printing and nonlinear color video displays. In custom digital logic implementations constrained by memory access speed, NMDI will be eight times faster than uncached trilinear and four times faster than tetrahedral interpolation. When color conversion by dither is inappropriate, cached tetrahedral and tetrahedral interpolations are competitive in speed but with greater complexity and typically somewhat larger errors than cached BPI.

Tetrahedral interpolation typically shows smaller average differences than BPI for RGB images interpolated with identity LUTs. This may not generalize to all non-identity LUTs, and CMYK prints with BPI are considered to have comparable quality. Relative robustness and user preference among interpolations remain topics for future research. 

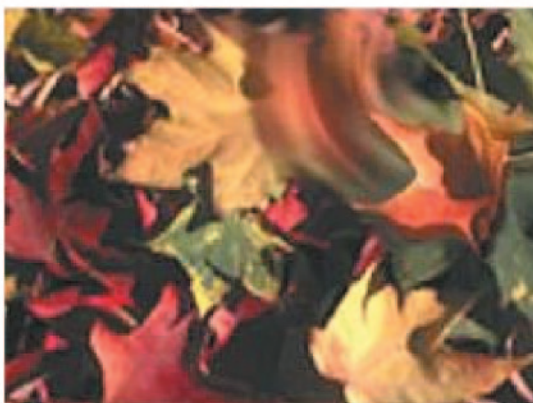
References

1. H. R. Kang, *Color technology for electronic imaging devices*, SPIE, Optical Engineering Press, Bellingham, WA, 1996.
2. R. Balasubramanian, Reducing the cost of lookup table based color transformations, *J. Imaging Sci. Technol.* **44**(4), 321–327 (2000).
3. Y. Azuma, M. Kaji, S. Otake, and J. Arney, Evaluation of an algebraic technique for colorimetric calibration of a printing system, *J. Imaging Sci. Technol.* **45**(2), 93–99 (2001).
4. R. Bala and V. Klassen, Efficient color transformation implementation, in *Digital Color Imaging Handbook*, G. Sharma, Ed., 2003.
5. P. Hemingway, *N-simplex interpolation*, Technical Report HPL-2002-320, HP Labs, Palo Alto, CA, 2002.
6. S. F. Weed and T. J. Cholewo, Color space binary dither interpolation, in *Proc. IS&T/SID 10th Color Imaging Conference*, IS&T, Springfield, VA, 2002, pp. 183–189.
7. T. J. Cholewo, Printer model inversion by constrained optimization, in *Proc. IS&T/SPIE's 12th Annual Symposium, Electronic Imaging 2000: Science and Technology*, SPIE, Optical Engineering Press, Bellingham, WA, 2000, pp. 349–357.
8. K. Kanamori, A study on interpolation errors and ripple artifacts of 3D lookup table method for nonlinear color conversion, *Proc. SPIE* **3648**, 167–178 (1999).
9. K. Spaulding, Method and apparatus employing mean preserving spatial modulation for transforming a digital color image signal, US Patent 5,377,041 (1994).

a.



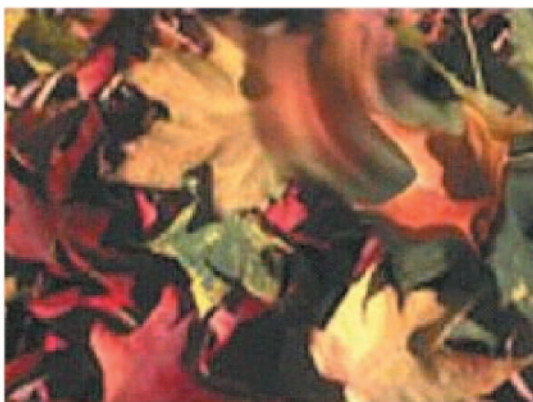
b.



c.



d.



Color Plate 13. Effects of caching on 4-bit interpolations using identity LUT: a) original; b) cached tetrahedral; c) cached binary proportional; and d) neighborhood binary dither. (Weed and Cholewo, pp. 525–530)

a.



b.



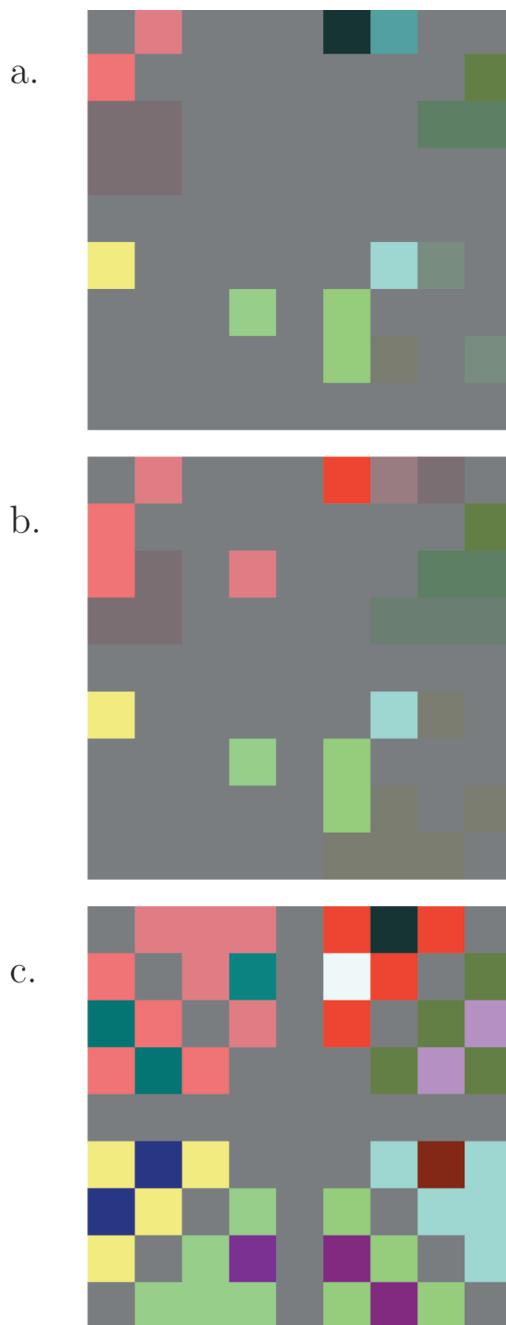
c.



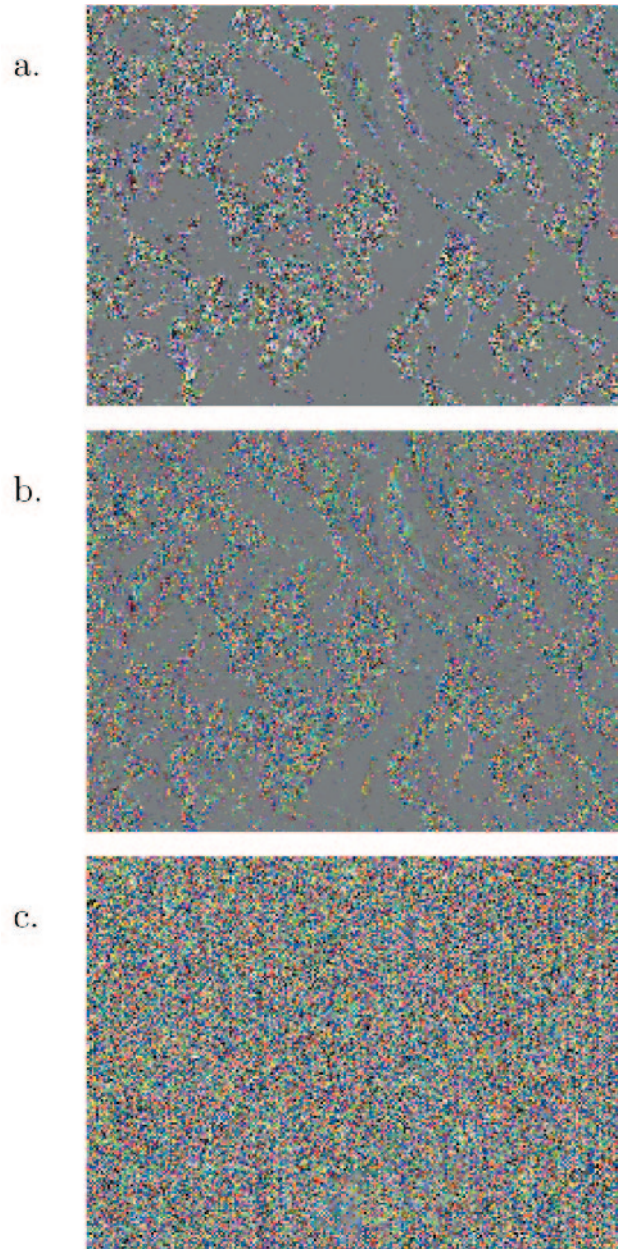
d.



Color Plate 14. Effects of caching on interpolation using 17×17 identity 24-bit LUT: a) original 9×9 image; b) cached tetrahedral; c) cached binary proportional; and d) neighborhood binary dither. (Weed and Cholewo, pp. 525–530)



Color Plate 15. Effects of caching on interpolation using 17×17 identity 24-bit LUT: exaggerated difference images between the identity mapping and a) cached tetrahedral; b) cached binary proportional; and c) neighborhood binary dither. (Weed and Cholewo, pp. 525–530)



Color Plate 16. Effects of caching on interpolation using $17 \times 17 \times 17$ identity 24-bit LUT: exaggerated difference images between the identity mapping and (a) cached tetrahedral; (b) cached binary proportional; and (c) neighborhood binary dither. (Weed and Cholewo, pp. 525–530)