Correspondence

Reversible Integer Color Transform

Soo-Chang Pei and Jian-Jiun Ding

***Abstract*—In this correspondence, we introduce a systematic algorithm that can convert any 3 × 3 color transform into a reversible integer-to-integer transform. We also discuss the ways to improve accuracy and reduce implementation complexity. We derive the integer RGB-to-KLA, $\mathrm{IV_1V_2}$, $\mathrm{YC_bC_r}$, DCT, YUV, and YIQ transforms that are optimal in accuracy.**

***Index Terms*—Color image processing, color transform, integer transform.**

I. Introduction

In color image processing, there are a variety of color-coordinate systems. Most of them consist of three components. Some popular and well-known ones are RGB, Karhunen–Loeve average (KLA), highest ability for color decorrelation, $\mathrm{IV_1V_2}$, $\mathrm{YC_bC_r}$, DCT, YUV, and YIQ [1], [2]. In many applications, we usually have to transform one color coordinate system into another one. The transformation is done by a 3 × 3 matrix. For example, the RGB-to-KLA transform is [2]

$$\mathbf{A_{KLA}} = \begin{bmatrix} 0.54933 & 0.60238 & 0.57912 \\ 0.80429 & -0.19322 & -0.56194 \\ 0.22661 & -0.77447 & -0.59063 \end{bmatrix}. \quad (1)$$

Since the entries of color transform matrices are usually not of binary form, we should use the floating-point processor to implement them. When using the fixed-point processor, we should approximate them by binary matrices. Unfortunately, the approximate2d matrices are always irreversible. For example, suppose that we apply rounding to approximate $\mathbf{A_{KLA}}$ by a binary matrix, i.e., $\mathbf{R_{KLA}} = \mathrm{round}(\mathbf{A_{KLA}}2^k)2^{-k}$ and $\mathbf{S_{KLA}} = \mathrm{round}(\mathbf{A_{KLA}^{-1}}(2^k)2^{-k}$. From computer experiments, we find that

$$\mathbf{S_{KLA}} \cdot \mathbf{R_{KLA}} \neq \mathbf{I} \quad \text{no matter how large } k \text{ is.} \quad (2)$$

Thus, after approximating a color transform by sums of powers of two, the reversibility property is always lost. It affects the performances of many image processing applications. For example, we usually hide the watermark in the least significant bit. If we cannot recover the original image, even if only the least significant bit is wrong, the watermark will be destroyed.

If we want to preserve the reversibility property, we should use a floating-point processor, which is more time-consuming and inefficient. To overcome this problem, some integer color transforms used for approximating noninteger color transforms were developed [2]–[6], [14], [15].

Manuscript received May 4, 2006; revised January 16, 2007. This work was supported by the National Science Council, R.O.C., under Contracts 93-2219-E-002-004 and NSC 93-2752-E-002-006-PAE. The associate editor coordinating the review of this paper and approving it for publication was Dr. G. Marcu.

The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 10617, R.O.C. (e-mail: pei@cc.ee.ntu.edu.tw; djj@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TIP.2007.896617

TABLE I
Parameters of the Reversible Integer Color Transforms (Part 1)

	RGB to KLA	RGB to $\mathrm{IV_1V_2}$	RGB to XYZ	RGB to UVW	RGB to YIQ
g_1	13/32	0	-175/128	-75/256	35/256
g_2	641/256	-47/128	-89/256	313/256	227/256
g_3	-115/256	1/2	205/256	141/256	-111/128
g_4	-77/128	-209/256	71/256	-25/128	-81/256
g_5	21/64	-157/256	95/256	7/256	-81/256
g_6	73/128	209/256	-27/256	97/256	57/256
g_7	-57/256	-101/128	-349/256	57/128	51/128
g_8	193/256	247/256	57/64	29/128	-31/128
$\mathbf{P_1^T}$ $\mathbf{\times D_1}$	$\begin{bmatrix} 0 & 2^{-1} & 0 \\ 0 & 0 & 1 \\ 2^{-2} & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2^{-1} & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 2^{-1} \\ 2^{-1} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2^{-1} & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 2^{-1} & 0 & 0 \\ 0 & 2^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\mathbf{D_2\times}$ $\mathbf{P_2^T}$	$\begin{bmatrix} 0 & -4 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 2 \\ 0 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
NR MSE	0.1609%	0.1842%	0.2156%	0.1291%	0.2703%

II. Previous Works About Integer Transforms and Matrix Factorization

In [2], the reversible integer RGB-to-KLA transform was introduced. In [3], the reversible integer RGB-to-$\mathrm{YC_bC_r}$ was developed. Then, in [4]–[6], [14], Hao, Shi, and Chen used a systematic way based on matrix factorization to derive the integer color transforms.

Matrix factorization is an important problem in linear algebra [12]. In [9]–[11], the ways that use three triangular matrices to decompose a unitary matrix were proposed. In [13], the way to decompose a matrix into a product of one-row matrices was also proposed. In [4]–[6], [14], Hao, Shi, and Chen apply these matrix decomposition methods to derive the reversible integer color transforms.

In this paper, we improve the previous works about integer color transform derivation. Our algorithm is also based on matrix decomposition. The improvement is that the proposed algorithm can achieve all the following five goals at the same time.

- [**Goal 1**] The integer color transform should be reversible.
- [**Goal 2**] No floating-point processor is required for both the forward and the inverse transforms.
- [**Goal 3**] The bit-length of the output should be constrained.
- [**Goal 4**] Less complexity for implementation
- [**Goal 5**] Accuracy: The integer transforms should well approximate the original transform.

Compared with the works in [2] and [3], the integer color transforms derived here are much more accurate (see Section VII). Compared with the advanced works in [4]–[6], [14], the improvements are: 1) the ways to save the number of time cycles for implementation (Section IV) and 2) the method to analyze the accuracy by normalized root mean square error are proposed (Section V). 3) The closed-form solutions of coefficients are shown. 4) Instead of ±1, the entries of diagonal matrices can be $\pm 2^k$. Moreover, two diagonal matrices were applied. Since our algorithm is more general, the integer color transform with even higher accuracy can be obtained. 5) We use our algorithm to derive the integer

TABLE II
PARAMETERS OF THE REVERSIBLE INTEGER COLOR TRANSFORMS (PART 2)

	RGB to KLA	RGB to IV_1V_2	RGB to YCrCb	RGB to DCT	RGB to YIQ
g_1	-215/1024	115/256	289/1024	53/128	139/1024
g_2	1313/1024	47/256	343/1024	325/1024	227/512
g_3	-107/512	-341/1024	99/1024	115/512	-81/256
g_4	221/256	-209/512	-347/512	-181/256	-443/1024
g_5	-857/1024	1	-137/256	-209/512	-423/512
g_6	1047/1024	141/256	-125/1024	-245/1024	57/128
g_7	-149/1024	119/1024	201/1024	53/128	205/512
g_8	-7/1024	-557/1024	-79/512	325/1024	-31/256
$\mathbf{P_1^T}\times\mathbf{D_1}$	$\begin{bmatrix} 2^{-1} & 0 & 0 \\ 0 & 2^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2^{-1} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 2^{-1} & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 2^{-1} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2^{-1} & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 2^{-1} \end{bmatrix}$
$\mathbf{D_2}\times\mathbf{P_2^T}$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 2 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}$
NR MSE	0.187%	0.175%	0.288%	0.267%	0.297%

RGB-to-KLA, $\mathrm{IV_1V_2}$, XYZ, UVW, YIQ, $\mathrm{YC_bC_r}$, YUV, $\mathrm{R_SG_SB_S}$, and $\mathrm{R_CG_CB_C}$ transforms that are optimal in accuracy successfully. They are listed in Tables I and II.

III. DECOMPOSITION AND INTEGERIZATION

First, we normalize the original 3×3 color transform $\mathbf{A_0}$ as $\mathbf{A}$ such that $\det(\mathbf{A}) = \pm 1$

$$\mathbf{A} = \sigma \cdot \mathbf{A}_0. \quad \sigma = |\det(\mathbf{A_0})|^{-1/3}. \tag{3}$$

Note that, if $\mathbf{y} = \mathbf{Ax}$ and $\mathbf{y_1} = \mathbf{A_0 x}$, then $\mathbf{y} = \sigma \mathbf{y_1}$. If the most and least significant bits of $\mathbf{y_1}$ are 2^{k_1} and 2^{k_2}, then those of $\mathbf{y}_1$ are 2^{k_3} and 2^{k_4}, where $k_3 \approx k_1 + \log_2 \sigma$ and $k_4 \approx k_2 + \log_2 \sigma$. Since $k_3 - k_4 + 1 \approx k_1 - k_2 + 1$, the number of bits of $\mathbf{y}$ is almost the same as that of $\mathbf{y}_1$. Thus, scaling just shifts the dynamic range in density values and does not enlarge or shorten it.

Then we do permutation, scaling, and sign changing operations for $\mathbf{A}$

$$\mathbf{C} = \mathbf{D}_1\mathbf{P}_1\mathbf{A}\mathbf{P}_2\mathbf{D}_2 \tag{4}$$

where $\mathbf{P_1}$ is a row-permuting matrix and $\mathbf{P_2}$ is a column-permuting matrix. In each row and column of a permuting matrix, only one entry is 1 and others are 0. $\mathbf{D_1}$ and $\mathbf{D_2}$ are diagonal matrices

$$D_1[m,m] = \pm 2^{-k_m}$$
$$D_2[m,m] = \pm 2^{k_m} \quad (m = 1, 2, \text{ and } 3) \tag{5}$$
$$D_{1\text{ or }2}[m,n] = 0 \text{ when } m \neq n. \tag{6}$$

$\mathbf{D}_1$ and $\mathbf{D}_2$ have the effects of scaling and sign changing. Note that there are 3! choices for $\mathbf{P}_1$, 3! choices for $\mathbf{P}_2$, and 2^6 choices for the signs of the diagonal entries of $D_{1\text{ or }2}[m,m]$. If we do not consider the case where $\det(\mathbf{C}) = -1$ and constrain that

$$0 \leq k_m \leq p_k \tag{7}$$

then, in sum, there are

$$(3!)^2(p_k+1)^3 2^6/2 = 1152(p_k+1)^3 \tag{8}$$

choices for $\mathbf{C}$. In theory, we can set p_k (the upper bound of k_m) to infinite, i.e., there are infinite choices for $\mathbf{C}$. However, from our experiments, the probability that the optimal integer transform is obtained when k_m has a large value is very small. Thus, in (7), setting $p_k = 2$ is usually sufficient to find the optimal integer transform. In this case, there are 31 104 choices for $\mathbf{C}$.

Then, applying the lifting scheme and the single-row elementary reversible matrix scheme [6], [7] with several modifications, we can decompose $\mathbf{C}$ into **four one-row matrices**

$$\mathbf{C} = \mathbf{T_4T_3T_2T_1} \tag{9}$$

$$\mathbf{T_1} = \begin{bmatrix} 1 & t_1 & t_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ where } t_1 = (c_{22}-1)/c_{21}$$
$$t_2 = -(t_1 z_2 + z_1) \tag{10}$$
$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}^{-1} \begin{bmatrix} -c_{23} \\ 1 - c_{33} \end{bmatrix}$$
$$c_{m\cdot n} \text{ are the entries of C} \tag{11}$$

$$\mathbf{T_2} = \begin{bmatrix} 1 & 0 & 0 \\ t_3 & 1 & t_4 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\text{where } t_3 = c_{21} \text{ and } t_4 = -z_2 \tag{12}$$

$$\mathbf{T_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_5 & t_6 & 1 \end{bmatrix}, \text{ where } t_5 = c_{22}c_{31} - c_{21}c_{32}$$
$$\text{and } t_6 = c_{32} - t_1 c_{31} \tag{13}$$

$$\mathbf{T_4} = \begin{bmatrix} 1 & t_7 & t_8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ where } t_7 = c_{12} - t_1 c_{11} - t_6 t_8$$
$$\text{and } t_8 = c_{13} + z_1 c_{11} + z_2 c_{12}. \tag{14}$$

We then use binary values g_n to approximate t_n

$$g_n = Q_b(t_n), \quad n = 1, 2, \ldots.., 8 \tag{15}$$

where Q_b is a truncation operation, which throws the bits that are less than 2^{-b}

$$Q_b\left[\pm \sum_{n=-\infty}^{\infty} d_n 2^{-n}\right] = \pm \sum_{n=-\infty}^{b} d_n 2^{-n} \quad (d_n = 0 \text{ or } 1). \tag{16}$$

Then we derive the reversible integer transform $\mathbf{B}$ that approximates $\mathbf{A}$ from

$$\mathbf{B} = \mathbf{P_1^T D_1 V_4 V_3 V_2 V_1 D_2 P_2^T} \tag{17}$$

where

$$\mathbf{V_1} = \begin{bmatrix} 1 & g_1 & g_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{V_2} = \begin{bmatrix} 1 & 0 & 0 \\ g_3 & 1 & g_4 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{V_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ g_5 & g_6 & 1 \end{bmatrix}, \quad \mathbf{V_4} = \begin{bmatrix} 1 & g_7 & g_8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{18}$$

Theorem 1: In (17), $\mathbf{B}$ is a binary matrix that approximates $\mathbf{A}$. Moreover, we can define the inverse integer transform as $\mathbf{B}' = \mathbf{P_2 D_2^{-1} V_1^{-1} V_2^{-1} V_3^{-1} V_4^{-1} D_1^{-1} P}_1$. It is easy to prove that

(1)

$$\mathbf{B}'\mathbf{B} = \mathbf{I}. \tag{19}$$

(2) Since the inverse of a binary one-row matrix is also a binary one-row matrix

$$\mathbf{V_1^{-1}} = \begin{bmatrix} 1 & -g_1 & -g_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{V_2^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ -g_3 & 1 & -g_4 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{V_3^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -g_5 & -g_6 & 1 \end{bmatrix}$$
$$\mathbf{V_4^{-1}} = \begin{bmatrix} 1 & -g_7 & -g_8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{20}$$

therefore, $\mathbf{B}'$ is bound to be a binary matrix and the first two goals listed in Section II are satisfied.

IV. Bit Constraint

Then we try to satisfy Goals 3–5 in Section II. Note that, in (15)–(18), if the least significant bit of g_n is 2^{-b}, and 2^{-a} and 2^{-c} are the least significant bits of $\mathbf{x}$ and $\mathbf{z}$ ($\mathbf{z} = \mathbf{B}\mathbf{x}$), then

$$c = a + 4b. \tag{21}$$

Thus, the bit-length of $\mathbf{z}$ is much longer than that of $\mathbf{x}$. For example, if the least significant bit of g_n is $1/2^8$, then $h_2 = h_1 + 32$. To solve the problem, we can convert $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V}_3$ and $\mathbf{V}_4$ in (18) into addition-truncation operations. For example, we can convert $\mathbf{x}_2 = \mathbf{V_1 x_1}$ into

$$\begin{aligned} x_2[1] &= x_1[1] + Q_r\left\{g_1 x_1[2] + g_2 x_1[3]\right\} \\ x_2[2] &= x_1[2],\ x_2[3] = x_1[3],\ Q_r \text{ is defined in (16)}. \end{aligned} \tag{22}$$

Note that, if the least significant bit of $x_1[m], x_2[1]$, and g_n are 2^{-q_1}, 2^{-q_2}, and 2^{-b}, respectively, then $q_2 = \max[q_1, \min(r, q_1 + b)]$. If we set r to satisfy that $q_1 \le r < q_1 + b$ and $b > 0$, then

$$q_2 = r \quad (\text{independent of } b). \tag{23}$$

That is, the least significant bit of $x_2[1]$ is increased from 2^{-q_1-b} to 2^{-r} and the bit length is reduced. Although **the bit length is reduced, the reversibility property is preserved**. For example, in (22), if we want to recover $\mathbf{x_1}$ from $\mathbf{x_2}$, then we can apply

$$\begin{aligned} x_1[1] &= x_2[1] - Q_r\left\{g_1 x_2[2] + g_2 x_2[3]\right\} \\ x_1[2] &= x_2[2],\ x_1[3] = x_2[3]. \end{aligned} \tag{24}$$

V. Time Cycle Problem and Implementation

There is another problem for implementing the integer color transform. That is, too many time cycles are required. Suppose that in each time cycle we can only do one addition and one multiplication in each entry. Quantization (just throwing bits), permuting (just twisting circuit in hardware), and multiplying 2^{-k} (just doing bit shifting) does not require any time cycle. Then, in (18), each of $\mathbf{V_1}, \mathbf{V_2}, \mathbf{V_3}$, and $\mathbf{V_4}$ requires three time cycles. For example, in (22), to implement $\mathbf{V_1}$, we must calculate $l_1 = g_1 x_1[2]$ in the first time cycle, calculate $l_1 = l_1 + g_2 x_1[3]$ in the second time cycle, and calculate $x_2[1] = x_1[1] + Q_r\{l_1\}$ in the third time cycle. Thus, we need **12 time cycles** to implement the integer color transform if we apply (17) and (18) directly.

However, from the facts that ① two adjacent one-row matrix can be implemented together and ② if r is larger than q_1 where 2^{-q_1} is the least significant bit of $x_1[1]$, then (22) is equivalent to

$$\begin{aligned} x_2[1] &= x_1[1] + Q_r\left\{g_1 x_1[2] + g_2 x_1[3]\right\} \\ &= Q_r\left\{x_1[1] + g_1 x_1[2] + g_2 x_1[3]\right\} \end{aligned} \tag{25}$$

we can **reduce the number of time cycles into 5**, as (26)–(30).

The process of the forward integer color transform is as follows.

Step 1)

$$f[1] = f[1] + g_1 f[2],\ f[2] = f[2],\ f[3] = f[3] \quad \text{where } \mathbf{f} = \mathbf{D_2 P_2^T x}. \tag{26}$$

Step 2)

$$\begin{aligned} f[1] &= Q_{r-k_1}\left\{f[1] + g_2 f[3]\right\} \\ f[2] &= f[2] + g_4 f[3],\ f[3] = f[3]. \end{aligned} \tag{27}$$

Step 3)

$$\begin{aligned} f[1] &= f[1],\ f[2] = Q_{r-k_2}\left\{f[2] + g_3 f[1]\right\} \\ f[3] &= f[3] + g_5 f[1]. \end{aligned} \tag{28}$$

Step 4)

$$\begin{aligned} f[1] &= f[1] + g_7 f[2],\ f[2] = f[2] \\ f[3] &= Q_{r-k_3}\left\{f[3] + g_6 f[2]\right\}. \end{aligned} \tag{29}$$

Step 5)

$$f[1] = Q_{r-k_1}\left\{f[1] + g_8 f[3]\right\},\ f[2] = f[2],\ f[3] = f[3] \quad \mathbf{z} = \mathbf{P_1^T D_1 f} \tag{30}$$

where Q_r is defined in (16) and k_m, g_m, $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{P}_1$, and $\mathbf{P}_2$ are defined in Section III. **The process of the inverse integer color transform** is as follows.

Step 1)

$$f[1] = f[1] - g_7 f[2],\ f[2] = f[2],\ f[3] = f[3] \quad \text{where } \mathbf{f} = \mathbf{D_1^{-1} P_1 z}. \tag{31}$$

Step 2)

$$\begin{aligned} f[1] &= Q_{r-k_1}\left\{f[1] - g_8 f[3]\right\},\ f[2] = f[2] \\ f[3] &= f[3] - g_6 f[2]. \end{aligned} \tag{32}$$

Step 3)

$$\begin{aligned} f[1] &= f[1],\ f[2] = f[2] - g_3 f[1] \\ f[3] &= Q_{r-k_3}\left\{f[3] - g_5 f[1]\right\}. \end{aligned} \tag{33}$$

Step 4)

$$\begin{aligned} f[1] &= f[1] - g_2 f[3],\ f[2] = Q_{r-k_2}\left\{f[2] - g_4 f[3]\right\} \\ f[3] &= f[3]. \end{aligned} \tag{34}$$

Step 5)

$$f[1] = Q_{r-k_1}\left\{f[1] - g_1 f[2]\right\},\ f[2] = f[2],\ f[3] = f[3] \quad \mathbf{x} = \mathbf{P_2 D_2^{-1} x_1}. \tag{35}$$

We can prove that

Theorem 2: If 2^{-a} is the least significant bit of $\mathbf{x}$, 2^{-b} is the least significant bit of g_n, and

$$a \le r \le a + b + k_n - k_m, \quad b > k_m - k_n \text{ for any } m, n, \tag{36}$$

then the least significant bit of the output $\mathbf{z}$ is

$$2^{-r} \quad \text{(independent of } b\text{)}. \tag{37}$$

Therefore, for our algorithm, if b is sufficient large such that (36) is satisfied, then the least significant bit of the output is determined by how many bits are preserved by the truncation operation Q and is independent of b. That is, in (15), **no matter how many bits we use for approximating $\mathbf{t_n}$, the least significant bit of the output is remained to be $\mathbf{r}$**.

Note that both the forward and inverse integer color transforms **require only five time cycles**. Moreover, **only three storages** are required during the process (for $f[1]$, $f[2]$, and $f[3]$). Thus, if we use (26)–(35) for implementing the integer color transform, Goals 3 and 4 in Section II are satisfied.

VI. Accuracy Analysis

Then we discuss the problem of accuracy, i.e., Goal 5 in Section II. Note that, in Steps 2)–5), the truncation operation Q_{r-k_m} is equivalent to adding a small number

$$Q_{r-k_m}\{a\} = a + \tau, \text{ where } -2^{-r+k_m-1} \le \tau \le 2^{-r+k_m-1}. \tag{38}$$

If the input of Q_{r-k_m} is not known, τ can be treated as a random variable that uniformly distributes in $(-2^{-r+k_m-1}, 2^{-r+k_m-1})$ and

$$E[\tau] = 0, \quad E[\tau^2] = 4^{-r+k_m}/12 \tag{39}$$

where E means the expected value. Thus, the process in (26)–(30) can be rewritten as

$$\mathbf{z} = \mathbf{P}_1^{\mathrm{T}}\mathbf{D}_1\left(\mathbf{V}_4\left\{\mathbf{V}_3\left[\mathbf{V}_2\left(\mathbf{V}_1\mathbf{D}_2\mathbf{P}_2^{\mathrm{T}}\mathbf{x} + \mathbf{\Delta}_1\right) + \mathbf{\Delta}_2\right] + \mathbf{\Delta}_3\right\} + \mathbf{\Delta}_4\right) \tag{40}$$

where $\mathbf{V_k}$ is defined in (18) and $\mathbf{\Delta_1}, \mathbf{\Delta_2}, \mathbf{\Delta_3}$, and $\mathbf{\Delta_4}$ are 3×1 random vectors

$$\begin{aligned}\Delta_1[2] &= \Delta_1[3] = \Delta_2[1] = \Delta_2[3] \\ &= \Delta_3[1] = \Delta_3[2] = \Delta_4[2] = \Delta_4[3] = 0\end{aligned} \tag{41}$$

and $\Delta_1[1]$, $\Delta_2[2]$, $\Delta_3[3]$, and $\Delta_4[1]$ are random variables whose statistical characteristics are the same as in (39). We can compare (40) with the original noninteger color transform. If $\mathbf{y} = \mathbf{A}\mathbf{x}$ (i.e., the original noninteger color transform of $\mathbf{x}$), then $\mathbf{y} = \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{T_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}$, $\mathbf{T_k}$ is defined in (10)–(14)

$$\begin{aligned}\mathbf{z} - \mathbf{y} = {} & \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{V_4}\mathbf{V_3}\mathbf{V_2}\mathbf{\Delta_1} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{V_4}\mathbf{V_3}\mathbf{\Delta_2} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{V_4}\mathbf{\Delta_3} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{\Delta_4} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}(\mathbf{V_4}\mathbf{V_3}\mathbf{V_2}\mathbf{V_1} - \mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{T_1}) \\ & \times \mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}.\end{aligned} \tag{42}$$

Suppose that b in (15) is large enough such that $\mathbf{V_n} = \mathbf{T_n} + \nabla_\mathbf{n}$, where the entries of $\nabla_\mathbf{n}$ is very small compared with those of $\mathbf{T_n}$, $n = 1, 2, 3, 4$

$$\begin{aligned}&\mathbf{V_4}\mathbf{V_3}\mathbf{V_2}\mathbf{V_1} - \mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{T_1} \\ &\quad = (\mathbf{T_4} + \mathbf{\nabla_4})(\mathbf{T_3} + \mathbf{\nabla_3})(\mathbf{T_2} + \mathbf{\nabla_2})(\mathbf{T_1} + \mathbf{\nabla_1}) \\ &\qquad - \mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{T_1} \\ &\quad \approx \mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{\nabla_1} + \mathbf{T_4}\mathbf{T_3}\mathbf{\nabla_2}\mathbf{T_1} + \mathbf{T_4}\mathbf{\nabla_3}\mathbf{T_2}\mathbf{T_1} \\ &\qquad + \mathbf{\nabla_4}\mathbf{T_3}\mathbf{T_2}\mathbf{T_1}.\end{aligned} \tag{43}$$

Theorem 3: The error of the integer color transform can be approximated by [from (42) and (43)]

$$\begin{aligned}\mathbf{z} - \mathbf{y} \approx {} & \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{\Delta_1} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{\Delta_2} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{\Delta_3} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{\Delta_4} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\nabla_1\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\nabla_2\mathbf{T_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\nabla_3\mathbf{T_2}\mathbf{T_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\nabla_4\mathbf{T_3}\mathbf{T_2}\mathbf{T_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}.\end{aligned} \tag{44}$$

1) The truncations in Steps 2)–5) cause $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{\Delta_1}$, $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{\Delta_2}$, $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{\Delta_3}$, and $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{\Delta_4}$, respectively.
2) Quantizing $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, and $\mathbf{T_4}$ into $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V_3}$, and $\mathbf{V_4}$ cause $\mathbf{P}_1^{\mathrm{T}}\mathbf{D}_1\mathbf{T}_4\mathbf{T}_3\mathbf{T_2}\mathbf{\nabla_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}$, $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{\nabla}_2\mathbf{T_1}\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}$, $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{\nabla_3}\mathbf{T_2}\mathrm{T}_1\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}$, and $\mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{\nabla}_4\mathbf{T_3}\mathbf{T_2}\mathrm{T}_1\mathbf{D_2}\mathbf{P}_2^{\mathrm{T}}\mathbf{x}$, respectively.

There are some things to be noticed.

A) From Theorem 2, the least significant bit of $\mathbf{z}$ is independent of b. Thus, in (15) we can choose a large b. If b is very large, then ∇_1, ∇_2, ∇_3, and ∇_4 are very small and the last four terms in (44) can be ignored

$$\begin{aligned}\mathbf{z}-\mathbf{y} \approx {} & \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{T_2}\mathbf{\Delta_1} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{T_3}\mathbf{\Delta_2} \\ & + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{T_4}\mathbf{\Delta_3} + \mathbf{P}_1^{\mathrm{T}}\mathbf{D_1}\mathbf{\Delta_4}.\end{aligned} \tag{45}$$

For example, for the case of the integer KLA transform, when $r = 0$ and $b = 8$, the last four terms affect only 8.05% of error. When $r = 0$ and $b = 12$, these terms contribute only 0.07% of error.

Note that in this case the error is independent of the input $\mathbf{x}$.

B) From (44) and (45), we can see that the entries of $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, and $\mathbf{T_4}$ affect the error of approximation. Especially, from (45), the entries of $\mathbf{T_3}$ and $\mathbf{T_4}$ have larger effects than $\mathbf{T_1}$ and $\mathbf{T_2}$. Thus, to make error small, we should adjust $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{D_1}$, and $\mathbf{D_2}$ to make the values of t_5, t_6, t_7, and t_8 in (13) and (14) as small as possible.

C) After the integer transform is designed, we can use (44) together with the following equation to estimate the normalized root mean square error (NRMSE):

$$\text{NRMSE} = \sqrt{\frac{E\left[(\mathbf{z}-\mathbf{y})^{\mathrm{T}}(\mathbf{z}-\mathbf{y})\right]}{E[\mathbf{y}^{\mathrm{T}}\mathbf{y}]}} \tag{46}$$

where $E[\,]$ means the expected value. In (4), we can vary the permuting matrices $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{D_1}$, and $\mathbf{D_2}$ iteratively and calculate the NRMSE of the integer transform for each time. In (8), we discussed that there are $1152(p_k+1)^3$ possible integer color transforms we can obtain. We can calculate NRMSE for each integer color transform and choose the optimal one with the minimal NRMSE.

VII. Examples

In this section, we show some integer color transforms we derived. Suppose that the least significant bit of the input data is 1 (i.e., in Theorem 2, $a = 0$). We choose the values of b and r as

$$b = 8, \quad r = 0. \tag{47}$$

Then, from Theorem 2, the least significant bit of the output is

$$2^{-0} = 1. \tag{48}$$

Therefore, *the least significant bits of both the input and the output data are 1*. We also suppose that the input signal is uniformly distributed in [0, 255] such that

$$E(x[n]) = 127.5, \quad E\left(x^2[n]\right) = 255^2/3$$
$$E^2(x[n]) = 127.5^2. \tag{49}$$

Using (49) together with (39), (44), (46), and the fact that $\mathbf{y^T y} = \mathbf{x^T A^T A x}$, we can calculate the NRMSE. We try to convert the ten color transforms in (50)–(54) into reversible integer transforms. The results are shown in Table I and II, where $g_k\,(k = 1 \sim 8)$ $\mathbf{D_1}, \mathbf{D_2}, \mathbf{P}_1$, and $\mathbf{P_2}$ are the parameters, diagonal matrix, and permuting matrices using in the processes of (26)–(30) (for the forward transform) and (31)–(35) (for the inverse transform)

$$(\mathbf{1})\ \mathbf{RGB\ to\ KLA}$$
$$\begin{bmatrix} 0.8185 & 0.8975 & 0.8629 \\ 1.1984 & -0.2879 & -0.8373 \\ 0.3376 & -1.1539 & -0.8800 \end{bmatrix}$$
$$(\mathbf{2})\mathbf{RGB\ to\ IV_1V_2}$$
$$\begin{bmatrix} 1 & 1 & 1 \\ -1/\sqrt{6} & -1/\sqrt{6} & 2/\sqrt{6} \\ 1/\sqrt{6} & -1/\sqrt{6} & 0 \end{bmatrix} \tag{50}$$
$$(\mathbf{3})\mathbf{RGB\ to\ XYZ}$$
$$\begin{bmatrix} 0.8706 & 0.2496 & 0.2868 \\ 0.4288 & 0.8419 & 0.1635 \\ 0 & 0.0947 & 1.6006 \end{bmatrix}$$
$$(\mathbf{4})\mathbf{RGB\ to\ UVW}$$
$$\begin{bmatrix} 0.8387 & 0.2402 & 0.2754 \\ 0.6192 & 1.2156 & 0.2361 \\ 0.3003 & 1.7125 & 1.2984 \end{bmatrix} \tag{51}$$
$$(\mathbf{5})\ \mathbf{RGB\ to\ YIQ}$$
$$\begin{bmatrix} 0.4722 & 0.9270 & 0.1800 \\ 0.9412 & -0.4327 & -0.5085 \\ 0.3332 & -0.8259 & 0.4927 \end{bmatrix}$$
$$(\mathbf{6})\mathbf{RGB\ to\ DCT}$$
$$\begin{bmatrix} 0.5774 & 0.5774 & 0.5774 \\ 0.7071 & 0 & -0.7071 \\ 0.4082 & -0.8165 & 0.4082 \end{bmatrix} \tag{52}$$
$$(\mathbf{7})\ \mathbf{RGB\ to\ YC_bC_r}$$
$$\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.500 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.500 \end{bmatrix}$$
$$(\mathbf{8})\ \mathbf{RGB\ to\ YUV}$$
$$\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.148 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.1 \end{bmatrix} \tag{53}$$
$$(\mathbf{9})\ \mathbf{RGB\ to\ R_CG_CB_C}$$
$$\begin{bmatrix} 1.609 & -0.447 & -0.104 \\ -0.058 & 0.977 & 0.051 \\ -0.025 & -0.037 & 1.162 \end{bmatrix}$$
$$(\mathbf{10})\ \mathbf{RGB\ to\ R_SG_SB_S}$$
$$\begin{bmatrix} 1.167 & -0.146 & -0.151 \\ 0.114 & 0.753 & 0.159 \\ -0.001 & 0.059 & 1.128 \end{bmatrix}. \tag{54}$$

For example, from Table I, the process of the forward integer RGB to KLA transform is

Step 1)
$$f[1] = -4x[2] + 13x_1[1]/16,\ f[2] = 2x[1]$$
$$f[3] = -x[3]. \tag{55}$$

Step 2)
$$f[1] = Q_{-2}\{f[1] + 641f[3]/256\}$$
$$f[2] = f[2] - 77f[3]/128$$
$$f[3] = f[3]. \tag{56}$$

Step 3)
$$f[1] = f[1],\ f[2] = Q_{-1}\{f[2] - 115f[1]/256\}$$
$$f[3] = f[3] + 21f[1]/64. \tag{57}$$

Step 4)
$$f[1] = f[1] - 57f[2]/256,\ f[2] = f[2]$$
$$f[3] = Q_0\{f[3] + 73f[2]/128\}. \tag{58}$$

Step 5)
$$z[1] = f[2]/2,\ z[2] = f[3]$$
$$z[3] = Q_{-2}\{f[1] + 193f[3]/256\}/4. \tag{59}$$

Note that, in Tables I and II, the NRMSEs of all the integer color transforms are smaller than 0.3%. In contrast, the NRMSE of the simple integer RGB-to-KLA transform in [2] is 68.80%. The NRMSE of the integer RGB-to-$\mathrm{YC_bC_r}$ transform in [3] is 41.14%. For the integer RGB-to-DCT transform in [5], the NRMSE is 0.3363% if we also choose $r = 0$ and $b = 8$. For the RGB-to-DCT transform, we derive

$$\mathrm{NRMSE} = 0.2679\%. \tag{60}$$

Due to the fact that $|D_{1\ \mathrm{or}\ 2}[m, m]|$ in (5) can be nonunitary and that we try all the 31 104 possible ways for decomposition, we can obtain more accurate integer color transforms.

VIII. Conclusion

We introduced a systematic way to convert a 3×3 color transform matrix by a reversible integer transform, which can satisfy all the five goals (binary entry, reversibility, short bit length, less time cycle, and accuracy) listed in Section II at the same time. With the proposed integer color transforms, we can use the fixed-point processor instead of the floating-point one to do lossless color coordinate transformation. It is useful for improving the efficiency and accuracy of image processing.

References

[1] W. K. Pratt, *Digital Image Processing*, 2nd ed. New York: Wiley, 1991.

[2] B. Deknuydt, J. Smolders, L. V. Eycken, and A. Oosterlinck, "Color space choice for nearly reversible image compression," *Proc. SPIE*, vol. 1818, pp. 1300–1311, 1992.

[3] M. J. Gormish, E. L. Schwartz, and A. Keith *et al.*, "Lossless and nearly lossless compression for high quality images," *Proc. SPIE*, vol. 3025, pp. 62–70, Feb. 1997.

[4] P. Hao and Q. Shi, "Comparative study of color transforms for image coding and derivation of integer reversible color transform," in *Proc. Int. Conf. Pattern Recognition*, Sep. 2000, vol. 3, pp. 224–227.

[5] Y. Chen and P. Hao, "Integer reversible transformation to make JPEG lossless," in *Proc. Int. Conf. Signal Processing*, 2004, pp. 835–838.

[6] P. Hao and Q. Shi, "Matrix factorizations for reversible Integer mapping," *IEEE Trans. Signal Process.*, vol. 49, no. 10, pp. 2314–2324, Oct. 2001.

[7] S. Oraintara, Y. J. Chen, and T. Q. Nguyen, "Integer fast Fourier transform," *IEEE Trans. Signal Process.*, vol. 50, no. 3, pp. 607–618, Mar. 2002.

[8] M. D. Adams, F. Kossentini, and R. K. Ward, "Generalized S transform," *IEEE Trans. Signal Process.*, vol. 50, no. 11, pp. 2831–2842, Nov. 2002.

[9] K. R. Nagarajan, M. P. Devasahayam, and T. Soundarajan, "Product of three triangular matrices," *Lin. Algebra Appl.*, vol. 292, pp. 61–71, Jul. 1999.

[10] T. Toffoli, "Almost every unit matrix is a ULU," *Lin. Algebra Appl.*, vol. 259, pp. 31–38, Jul. 1997.

[11] G. Strang, "Every unit matrix is a LULU," *Lin. Algebra Appl.*, vol. 265, pp. 165–172, Nov. 1997.

[12] P. Dita, "Factorization of unitary matrices," *J. Phys. A: Math. Gen.*, vol. 39, pp. 2781–2789, 2003.

[13] B. Chen and A. Kaufman, "3D volume rotation using shear transforms," *Graph. Models*, vol. 62, pp. 308–322, 2000.
[14] P. Hao, "Customizable triangular factorizations of matrices," *Lin. Algebra Appl.*, vol. 382, pp. 135–154, May 2004.
[15] S. C. Pei and J. J. Ding, "Reversible integer color transform with bit-constraint," in *Proc. Int. Conf. Image Processing*, 2005, vol. 3, pp. 964–967.

Objective Distortion Measure for Binary Text Image Based on Edge Line Segment Similarity

Jun Cheng and Alex C. Kot, *Fellow, IEEE*

***Abstract*—This paper proposes a new approach to measure the distortion introduced by changing individual edge pixels in binary text images. The approach considers not only how many pixels are changed but also where the pixels are changed and how the flipping affects the overall shape formed by the edge line. Similarities between the edge line segments in the original and distorted image are compared to measure the distortion. Subjective testing shows that the new distortion measure correlates well with human visual perception.**

***Index Terms*—Binary text image, distortion measure.**

I. Introduction

Distortion measure is an important topic in image processing including data hiding. An important application of the distortion measure in data hiding is to evaluate the performance of the data hiding algorithms as well as to provide insights in data hiding. Distortion measure can be categorized into subjective measurement and objective measurement [1]. As discussed in [2], subjective distortion measure quantifies the dissatisfaction of the viewer in observing the distorted image in place of the original. A common way to evaluate the dissatisfaction is through subjective testing. In these tests, observer views a series of distorted images and rate them based on the visibility of the artifact. The results of subjective testing depend on various factors such as the observer's background. Subjective measurement is important for image quality evaluation since the images are ultimately viewed by human beings. However, subjective testing is inconvenient, time consuming, and expensive. The objective distortion measure gives the distortion between the original and the distorted image mathematically such as mean-square error (MSE), peak signal-to-noise ratio (PSNR), and signal-to-noise ratio (SNR). However, it may not reflect the observer's visual perception of distortion. An objective distortion measure that can accurately reflect the subjective ratings would be quite useful in designing data hiding algorithms. Several authors have discussed the gaps between subjective measurement and objective measurement and they have proposed solutions for multilevel images [2]–[4]. Little work has been done for objective measurement for binary text images. In recently years, data hiding in binary text images have received much attention and a series of data hiding schemes for binary images by flipping individually selected edge pixels have been proposed [5]–[10]. These schemes are called pixel flipping techniques. One of the important issues of the pixel flipping technique is how to select the flippable pixels to minimize distortion. Also, the comparison of the distortion introduced by different schemes is important. However, performance evaluation is difficult due to the lack of objective distortion measure that reflects the subjective test results. There is an urgent need to develop a good objective distortion measure.

Manuscript received April 13, 2006; revised February 4, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Zhigang (Zeke) Fan.

J. Cheng is with the Workstation Resource Laboratory, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: cjun@pmail.ntu.edu.sg).

A. C. Kot is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: eackot@ntu.edu.sg).

Digital Object Identifier 10.1109/TIP.2007.896619

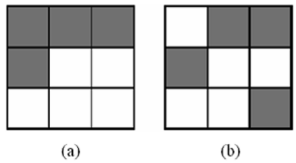

Fig. 1. Two 3 × 3 pixel patterns with same DRDM distortion score [11].

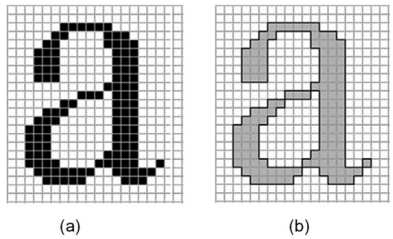

Fig. 2. Illustration of edges: (a) enlarged "a" and (b) edge line of "a."

Distance reciprocal distortion measure (DRDM) proposed in [11] uses a weight matrix to calculate the distortion caused by the flipping of pixels. It shows a better correlation with human perception than PSNR, and this measure works well if salt and pepper noise is involved. For example, flipping a nonedge pixel usually causes a larger visual distortion than flipping an edge pixel. However, most good data hiding techniques for binary images involve flipping edge pixels only. If connectivity is also taken into consideration, the measure of distortion would be more accurate. For example, flipping the center pixels of the two patterns in Fig. 1 have the same distortion score according to DRDM, but the visual distortions by flipping them are quite different. The change in smoothness and connectivity measure (CSCM) by using a modified weight matrix in [12] gives reasonable distortion scores. However, it has a limitation for patterns such as sharp corner and pixels along straight line. In this paper, we propose a new objective distortion measure based on edge line segment to evaluate the distortion introduced by flipping individually selected edge pixels for binary text images. The new proposed method is introduced in Section II, followed by experimental results and a conclusion.

II. Proposed Method

We define an edge line as the common sharing 'line' between two neighboring pixels where the pixel values for the two pixels are different. The edge pixel refers to pixels (either white or black) on the edge. For example, the edge lines in Fig. 2(a) are those black lines shown in Fig. 2(b). When the edge line changes its direction by 90 degrees, it forms a sharp "corner." There are eight different types of corners shown in Fig. 3 that make up all possible corners in a 2 × 2 pixel block.