The Segmented Gray-Code Kernels for Fast Pattern Matching

Wanli Ouyang*, *Member, IEEE,* Renqi Zhang, *Student Member, IEEE,* and Wai-Kuen Cham, *Senior Member, IEEE*

***Abstract*—The Gray-Code Kernels (GCK) family which has Walsh Hadamard Transform (WHT) on sliding windows as a member is a family of kernels that can perform image analysis efficiently using a fast algorithm such as the GCK algorithm. The GCK has been successfully used for pattern matching. In this paper, we propose the G4-GCK algorithm that is more efficient than the previous algorithm in computing GCK. The G4-GCK algorithm requires 4 additions per pixel for 3 basis vectors independent of transform size and dimension. Based on the G4-GCK algorithm, we then propose the segmented GCK. By segmenting input data into L_s parts, the SegGCK requires only 4 additions per pixel for $3L_s$ basis vectors. Experimental results show that the proposed algorithm can significantly accelerate the full-search equivalent pattern matching process and outperforms state-of-the-art methods.**

***Index Terms*—Fast algorithm, Walsh Hadamard Transform, pattern matching, template matching, feature extraction, block matching.**

I. Introduction

PATTERN matching, also called template matching, aims at locating a given pattern or template in a given image as shown in Fig. 1. Pattern matching has found applications in signal processing, computer vision, image and video processing. It has been applied for quality control [1], image based rendering [2], image compression [3], super-resolution [4], texture synthesis [5], block matching in motion estimation [6], [7], image denoising [8], [9], tracking [10], tone mapping [11] and image matching [12].

Pattern matching requires intensive computation because the search space is usually huge. Hence, many fast algorithms have been proposed to relieve this computational burden. The algorithms can be grouped into full search nonequivalent algorithms and full search equivalent algorithms. *Full search nonequivalent* algorithms achieve computational savings by reducing the search space [13], [14], [15] or by approximating patterns and windows using polynomials [16], [17], [18], [19] or linear combination of features [20].

On the other hand, *full search equivalent* algorithms accelerate the pattern matching process and, at the same time, yield exactly the same result as that of the full search (FS). The FFT-based approach, which has been implemented in OpenCV [21], is FS equivalent. The FS equivalent algorithms in [22], [23], [24], [25], [26], [27] construct rejection schemes which reject a large portion of mismatched candidates in computationally efficient ways and save the complex computation originally required by the FS for these rejected candidates. FS equivalent algorithms can be modified to FS nonequivalent algorithms. For example, the novel FS equivalent algorithms in [23], [24] have been modified to FS nonequivalent algorithms and applied for block matching in motion estimation [6], [7].

W. Ouyang, R. Zhang and W.K. Cham are with the Chinese University of Hong Kong, China. e-mail: wlouyang@ee.cuhk.edu.hk; wkcham@ee.cuhk.edu.hk.

Manuscript received Dec. 17, 2011; revised Apr. 29, 2012.

A. Pattern Matching Using Transformation

Suppose a template/pattern is to be sought in a given image as shown in Fig. 1. The template/pattern will be compared with candidate windows of similar size in the image. Consider the 1D pattern matching case, we represent the template/pattern as vector $\vec{\mathbf{x}}_t^{(N)}$ having length N and represent the jth candidate window as $\vec{\mathbf{x}}_w^{(N,j)}$ having length N and $j = 0, 1, \ldots, W-1$. For example, if a pattern having length $N = 16$ is sought in a vector having length 256, we have $W = 256 - 16 + 1 = 241$. In order to compare $\vec{\mathbf{x}}_t^{(N)}$ with $\vec{\mathbf{x}}_w^{(N,j)}$, we can compute the distance between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$, which is denoted by $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$ and used for measuring the dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$. The distance $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$ should increase as the dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ increases. If T is the threshold that discriminates between matched and mismatched candidates, then $\vec{\mathbf{x}}_w^{(N,j)}$ is considered to match $\vec{\mathbf{x}}_t^{(N)}$ when $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)}) < T$. In this paper, the distance between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ is measured by $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2$ which is also called the sum of squared differences (SSD) between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$. As pointed out in [23], although there are arguments against the SSD as a dissimilarity measure for images, it is still commonly used due to its simplicity. Discussion on the SSD as a dissimilarity metric can be found in [28], [29], [30].

The transformation that projects a vector $\vec{\mathbf{x}}^{(N)} \in \mathbb{R}^N$ onto a linear subspace spanned by U basis vectors

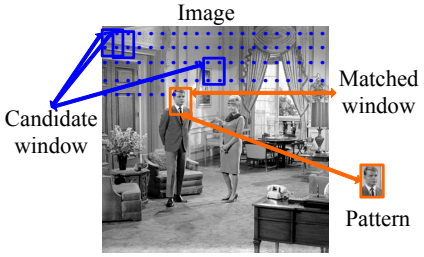


Fig. 1. Pattern matching in image 'couple'

TABLE I
TRANSFORM DOMAIN PATTERN MATCHING

Overall procedure:
Initially, set_{can} contains all candidate windows $\vec{\mathbf{x}}_w^{(N,j)}$;
Rejection Step: For $u = 1$ to N_{Maxu}:{
For $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}: {*Step a.1; Step a.2;*}
}
FS-Step: The remaining candidate windows in set_{can} undergo FS.

Step a.1: $\vec{\mathbf{y}}_t^{(u)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)}$; $\vec{\mathbf{y}}_w^{(u,j)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}$.
Step a.2: If $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$, then $\vec{\mathbf{x}}_w^{(N,j)}$ is removed from set_{can}.

$\vec{\mathbf{v}}^{(N,0)}, \ldots \vec{\mathbf{v}}^{(N,U-1)}$ can be represented as follows:

$$\vec{\mathbf{y}}^{(U)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{x}}^{(N)} = [\vec{\mathbf{v}}^{(N,0)} \ \ldots \ \vec{\mathbf{v}}^{(N,U-1)}]^T\vec{\mathbf{x}}^{(N)}, \quad (1)$$

where $\cdot^T$ is matrix transposition, vector $\vec{\mathbf{x}}^{(N)}$ of length N is the input window, vector $\vec{\mathbf{y}}^{(U)}$ of length U is the projection value vector, the U elements in vector $\vec{\mathbf{y}}^{(U)}$ are projection values and $\mathbf{V}^{(U\times N)}$ is a $U \times N$ matrix that contains U orthogonal basis vectors $\vec{\mathbf{v}}^{(N,i)}$ of length N for $i = 0, \ldots, U-1$. When $U = N$, we denote $\mathbf{V}^{(U\times N)}$ as $\mathbf{V}^{(N)}$. In [23], [31], the transformation is called the projection. Basis vectors are called projection kernels in [23] and called filter kernels in [24].

The following inequality is proved in [23] when the u basis vectors in $\mathbf{V}^{(u\times N)}$ are orthonormal:

$$\begin{aligned}||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2 &\geq ||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2 \\ &= ||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2, \end{aligned} \quad (2)$$

where $\vec{\mathbf{y}}_t^{(u)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{y}}_w^{(u,j)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}$.

If $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$, then we have $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2 > T$ from (2), and so we can safely prune candidate window $\vec{\mathbf{x}}_w^{(N,j)}$ from set_{can}. In this way, $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$ is a sufficient condition for rejecting candidate window $\vec{\mathbf{x}}_w^{(N,j)}$. Denote the set of candidates as set_{can}, which initially contains all candidates. For each iteration of u, where u increases from 1, $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ are projected onto transform domain using $\mathbf{V}^{(u\times N)}$ and the rejection condition is checked for the remaining candidates in set_{can}. Finally, after N_{Maxu} iterations, the remaining candidate windows in set_{can} undergo FS for finding out the matched windows. This procedure is summarized in Table I. Such a pattern matching approach is FS equivalent. The u basis vectors in $\mathbf{V}^{(u\times N)}$ are selected from the U orthonormal vectors $\vec{\mathbf{v}}^{(N,0)} \ \ldots \ \vec{\mathbf{v}}^{(N,U-1)}$ in $\mathbf{V}^{(U\times N)}$.

There are two possible ends in pattern matching: 1) detect all candidate windows having $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)}) < T$, for a given threshold T; 2) find the window that leads to the minimum value of $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$ among all candidate windows. The procedure given in Table I can be modified to find the window that has the minimum $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ using the approaches proposed in [32], [23] and [33]. Take the approach in [23] as an example. The threshold T can be adapted in each loop of u based on the minimum lower bound found in the uth loop.

The main advantage of using a transformation is that $||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2$ can be computed more efficiently than $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2$ and a small number of projection values can eliminate a large number of mismatched windows. According to [23], pattern matching using Walsh Hadamard Transform (WHT) as the transform is almost two orders of magnitude faster than the FS. The other advantages are:

- Transform domain pattern matching is FS-equivalent.
- It can be modified to FS nonequivalent algorithms, e.g. in [6], [7].
- As shown in [23], it can deal with illumination effect and multi-scale pattern matching. This property is used in [12].
- The new transforms and algorithms developed can be used as a feature extraction approach. WHT coefficients are used as features for further analysis in tracking [10] and wide-baseline image matching [12].

Because of these advantages, transform domain pattern matching has found application in block matching in motion estimation for video coding [6], [7], tracking [10], feature point based image matching [12], texture synthesis [34] and augmented reality [35].

Unlike the FFT approach, transform domain pattern matching does not obtain exact SSD results for all candidates, which might be required in some applications. Transform domain pattern matching only finds the k-minimum SSD results or the SSD results below threshold T using the FS-Step in Table I. More comparison of WHT and FFT is detailed in [23].

As analyzed in [23], the computational efficiency of transform domain pattern matching is dependent on two factors: 1) the cost of computing transformation for Step a.1 in Table I; 2) the ability of the rejection condition $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$ for Step a.2 in rejecting mismatched windows and thus saving the computation required afterwards, which is determined by the energy packing ability of the transform. The energy packing ability of a transform corresponds to its ability to compact the energy of input data, i.e. the ability in using small u to obtain large $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2$. The larger is $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2$, the more powerful is the rejection condition $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$ in pruning mismatched windows. In summary, the transform should be computationally efficient in packing energy.

Hel-Or and Hel-Or found in [23] that WHT is efficient for transform domain pattern matching. Their algorithm requires $2N - 2$ additions for obtaining all WHT projection values in each window of size N. The Gray-Code Kernel (GCK) algorithm proposed in [24] requires a similar number of additions as [23] when all projection values are computed and requires fewer number of additions when only a small number of projection values are computed. Recently, we proposed a faster algorithm for computing WHT in [25].

Meanwhile, new families of transforms that can be efficiently computed by fast algorithms were also proposed. The GCK [24], which has WHT on sliding windows as a member, is a family of transforms that can be computed efficiently by the GCK algorithm. The generalized GCK [36], which is a superset of GCK, can be computed efficiently by the approach in [36].

B. Overview

In this paper, we shall first develop a new algorithm for computing GCK, which requires 4/3 additions per datum per basis vector independent of transform size and dimension. This

Fig. 2. Order-8 WHT basis vectors in sequency order and dyadic order. White represents the value +1 and grey represents the value -1. Normalization factor of basis vectors is skipped.

algorithm is called G4-GCK algorithm. Limited by WHT or GCK, fast algorithms such as the GCK algorithm, our previous algorithm in [25] and the new G4-GCK algorithm require more than one addition for one basis vector. We then develop a new Segmented GCK (SegGCK) that can be computed more efficient than existing algorithms for WHT and GCK. By segmenting input data into L_s parts, the proposed fast SegGCK algorithm requires 4/($3L_s$) addition(s) per datum per basis vector. For example, when L_s is 8, the SegGCK requires only 1/6 addition per datum per basis vector. As shown by experimental results, pattern matching can be significantly accelerated by performing the transformation more efficiently.

The rest of the paper is organized as follows. Section II introduces WHT and GCK. Section III illustrates the proposed G4-GCK algorithm. Section IV extends the algorithm to high dimensional GCK. Section V introduces the SegGCK and its fast algorithm. Section VI gives the experimental results. Finally, Section VII presents the conclusions.

II. The WHT and GCK

A. The WHT

1D order-$N_{\mathbf{M}}$ WHT transforms $N_{\mathbf{M}}$ numbers into $N_{\mathbf{M}}$ projection values. Let $\mathbf{M}^{(N_{\mathbf{M}})}$ be an order-$N_{\mathbf{M}}$ WHT matrix and:

$$\begin{aligned}\mathbf{M}^{(N_{\mathbf{M}})} &= [\vec{\mathbf{m}}^{(N_{\mathbf{M}},0)}, \dots \vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})}, \dots \vec{\mathbf{m}}^{(N_{\mathbf{M}},N_{\mathbf{M}}-1)}]^T \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{M}^{(N_{\mathbf{M}}/2)} & \mathbf{M}^{(N_{\mathbf{M}}/2)} \\ \mathbf{M}^{(N_{\mathbf{M}}/2)} & -\mathbf{M}^{(N_{\mathbf{M}}/2)} \end{bmatrix}, \end{aligned} \quad (3)$$

where $\mathbf{M}^{(1)} = 1$, $\mathbf{M}^{(N_{\mathbf{M}})}$ is an $N_{\mathbf{M}} \times N_{\mathbf{M}}$ matrix, $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})}$ for $i_{\mathbf{M}} = 0, \dots, N_{\mathbf{M}} - 1$ is the $i_{\mathbf{M}}$th WHT basis vector having length $N_{\mathbf{M}}$. $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})^T}$ is the $i_{\mathbf{M}}$th row of $\mathbf{M}^{(N_{\mathbf{M}})}$ in (3). When $N_{\mathbf{M}} = 8$, Fig. 2 shows the order-8 WHT in dyadic order and sequency order. For example, the $\vec{\mathbf{m}}^{(N_{\mathbf{M}},2)}$ of dyadic-ordered WHT is the $\vec{\mathbf{m}}^{(N_{\mathbf{M}},3)}$ of sequency-ordered WHT. For sequency-ordered WHT, the extracted spatial frequency increases as the index $i_{\mathbf{M}}$ of basis vector $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})}$ increases. The relationship between dyadic-ordered and sequency-ordered WHT is detailed in [37]. For ease of explanation, dyadic-ordered WHT will be used in the following of this paper if not specified.

WHT has long been used for image representation under numerous applications. More discussions on applying WHT for pattern matching are available in [23], [36] and [32].

B. The GCK

The GCK was introduced as a filter in [24]. This paper explains the GCK by transform using Kronecker product which is denoted by $\otimes$. If $\mathbf{A}$ is a $U_1 \times Q_1$ matrix (a_{n_1,n_2}) and $\mathbf{B}$ is a $U_2 \times Q_2$ matrix (b_{m_1,m_2}), then $\mathbf{A} \otimes \mathbf{B}$ is a $U_1U_2 \times Q_1Q_2$ matrix as follows:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} & \cdots & a_{0,Q_1-1}\mathbf{B} \\ a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} & \cdots & a_{1,Q_1-1}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{U_1-1,0}\mathbf{B} & a_{U_1-1,1}\mathbf{B} & \cdots & a_{U_1-1,Q_1-1}\mathbf{B} \end{bmatrix}. \quad (4)$$

More information on Kronecker product can be found in [38].

Let matrix $\mathbf{S} \in \mathbb{R}^{R \times R}$ be:

$$\mathbf{S} = [\vec{\mathbf{s}}^{(0)}, \dots, \vec{\mathbf{s}}^{(i_r)}, \dots, \vec{\mathbf{s}}^{(R-1)}]^T. \quad (5)$$

The 1D order-N GCK matrix of $\mathbf{S}$ and WHT matrix $\mathbf{M}^{(N_{\mathbf{M}})}$ can be represented as:

$$\begin{aligned}\mathbf{V}^{(N)} &= [\vec{\mathbf{v}}^{(N,0)}, \dots \vec{\mathbf{v}}^{(N,i)}, \dots \vec{\mathbf{v}}^{(N,N-1)}]^T \\ &= \mathbf{M}^{(N_{\mathbf{M}})} \otimes \mathbf{S}, \quad (6) \\ \text{where} \quad & N = N_{\mathbf{M}}R, \quad (7)\end{aligned}$$

$\mathbf{V}^{(N)}$ is an $N \times N$ matrix, $\vec{\mathbf{v}}^{(N,i)}$ for $i = 0, \dots, N-1$ denotes the ith GCK basis vector given by $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})}$ and $\vec{\mathbf{s}}^{(i_r)}$ as follows:

$$\vec{\mathbf{v}}^{(N,i)} = \vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})} \otimes \vec{\mathbf{s}}^{(i_r)}. \quad (8)$$

Consider J input elements x_j for $j = 0, 1, \dots, J-1$, which are divided into overlapping windows of size N ($J > N$). Let the jth length-N input window for $j = 0, 1, \dots, J - N$ be:

$$\vec{\mathbf{x}}_w^{(N,j)} = [x_j, x_{j+1}, \dots, x_{j+N-1}]^T. \quad (9)$$

For 1D pattern matching , $\vec{\mathbf{x}}_w^{(N,j)}$ are sliding candidate windows which will be compared with the given pattern $\vec{\mathbf{x}}_t^{(N)}$. Let $y(N,i,j)$ for $i = 0, 1, \dots, N-1; j = 0, 1, \dots, J-N$ be the ith GCK projection value for the jth window and

$$y(N,i,j) = < \vec{\mathbf{v}}^{(N,i)}, \vec{\mathbf{x}}_w^{(N,j)} > = \vec{\mathbf{v}}^{(N,i)^T} \vec{\mathbf{x}}_w^{(N,j)}. \quad (10)$$

In other words, $y(N,i,j)$ is obtained by projecting the jth input window $\vec{\mathbf{x}}_w^{(N,j)}$ defined in (9) onto the ith GCK basis vector $\vec{\mathbf{v}}^{(N,i)}$ defined in (8). Let $\vec{\mathbf{y}}^{(N,j)}$ be the projection value vector containing all order-N GCK projection values at the jth window and

$$\vec{\mathbf{y}}^{(N,j)} = \mathbf{V}^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \quad (11)$$

For example, let $\mathbf{M}^{(N_{\mathbf{M}})} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $\mathbf{S} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Hence,

$$\begin{aligned}\vec{\mathbf{y}}^{(4,j)} &= \mathbf{V}^{(4)} \vec{\mathbf{x}}_w^{(4,j)} = \left[\mathbf{M}^{(2)} \otimes \mathbf{S}\right] \vec{\mathbf{x}}_w^{(4,j)} \\ &= \begin{bmatrix} y(4,0,j) \\ y(4,1,j) \\ y(4,2,j) \\ y(4,3,j) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & -1 & -2 \\ 3 & 4 & -3 & -4 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ x_{j+3} \end{bmatrix}. \end{aligned} \quad (12)$$

As pointed out in [36], the GCK matrix $\mathbf{V}^{(N)}$ in (6) is orthogonal if the $\mathbf{S}$ in (6) is orthogonal. The $\mathbf{S}$ in (6) corresponds to the "seed" vectors in [24]. The 1D generalized GCK proposed in [36] is defined as:

$$\mathbf{V}^{(N)} = \left(\begin{bmatrix} 1 & a_0 \\ 1 & -a_0 \end{bmatrix} \otimes \dots \begin{bmatrix} 1 & a_k \\ 1 & -a_k \end{bmatrix} \otimes \dots \begin{bmatrix} 1 & a_{G-1} \\ 1 & -a_{G-1} \end{bmatrix} \right) \otimes \mathbf{S}, \quad (13)$$

where $a_k \in \mathbb{Z}^+, k = 0, \dots, G-1$. WHT is a member of the GCK with $\mathbf{S} = \mathbf{I}_1$ in (6). GCK is a subset of generalized GCK with $a_k = 1$ in (13).

C. Definition of α-index and Being α^2-related

The α-index and α^2 relation defined in this paper is used for the G4-GCK algorithm proposed in the next section.

Let the row index of WHT matrix $\mathbf{M}^{(N_\mathbf{M})}$ be $i_\mathbf{M}$. The binary sequence α=$[\alpha_{G_\mathbf{M}-1}\ldots\alpha_g\ldots\alpha_1\ \alpha_0]$ for $\alpha_g \in \{0,1\}$ and $g = 0,\ldots,G_\mathbf{M}-1$ is called the α-index of $i_\mathbf{M}$, where

$$i_\mathbf{M} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\cdots+\alpha_1 2^1+\alpha_0 2^0, N_\mathbf{M} = 2^{G_\mathbf{M}}. \quad (14)$$

Thus the α-index is the binary representation of $i_\mathbf{M}$.

Let $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ be four possible values of $i_\mathbf{M}$. For the fixed $\vec{\mathbf{s}}^{(i_r)}$, if the α-indices of $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ are only different at α_g and α_{g+1}, then we say that: 1) WHT basis vectors $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(0)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(1)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(2)})}$ and $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(3)})}$ are α^2-related at g; 2) the corresponding GCK projection values $y(N,i^{(0)},j)$, $y(N,i^{(1)},j)$, $y(N,i^{(2)},j)$ and $y(N,i^{(3)},j)$ represented as follows are α^2-related at g:

$$\begin{bmatrix} y(N,i^{(0)},j)\\ y(N,i^{(1)},j)\\ y(N,i^{(2)},j)\\ y(N,i^{(3)},j)\end{bmatrix}=\left(\begin{bmatrix}\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(0)})^T}\\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(1)})^T}\\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(2)})^T}\\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(3)})^T}\end{bmatrix}\otimes\vec{\mathbf{s}}^{(i_r)^T}\right)\vec{\mathbf{x}}_w^{(N,j)}. \quad (15)$$

Note that the basis vectors $\vec{\mathbf{v}}^{(N,i^{(n)})}$ generating $y(N,i^{(n)},j)$ for $n = 0,1,2,3$ in (15) have the same $\vec{\mathbf{s}}^{(i_r)}$ but different $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(n)})}$. Without losing generality, let the $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ in (15) be represented by:

$$i_\mathbf{M}^{(0)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\ldots+0\cdot 2^{g+1}+0\cdot 2^g+\cdots+\alpha_0 2^0, \quad (16)$$
$$i_\mathbf{M}^{(1)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\ldots+0\cdot 2^{g+1}+1\cdot 2^g+\cdots+\alpha_0 2^0, \quad (17)$$
$$i_\mathbf{M}^{(2)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\ldots+1\cdot 2^{g+1}+0\cdot 2^g+\cdots+\alpha_0 2^0, \quad (18)$$
$$i_\mathbf{M}^{(3)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\ldots+1\cdot 2^{g+1}+1\cdot 2^g+\cdots+\alpha_0 2^0. \quad (19)$$

Some WHT basis vectors $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})}$ and their α-indices for $N_\mathbf{M}$= 4, 8 are given in Table II. For example, the α-indices [00], [01], [10] and [11] of 0, 1, 2 and 3 are only different at α_0 and α_1, thus $\vec{\mathbf{m}}^{(4,0)}$, $\vec{\mathbf{m}}^{(4,1)}$, $\vec{\mathbf{m}}^{(4,2)}$ and $\vec{\mathbf{m}}^{(4,3)}$ are α^2-related at 0. The α-indices [000], [010], [100] and [110] of 0, 2, 4 and 6 are only different at α_1 and α_2, thus $\vec{\mathbf{m}}^{(8,0)}$, $\vec{\mathbf{m}}^{(8,2)}$, $\vec{\mathbf{m}}^{(8,4)}$ and $\vec{\mathbf{m}}^{(8,6)}$ are α^2-related at 1.

As an example for α^2-related GCK projection values, when $\mathbf{M}^{(N_\mathbf{M})} = \mathbf{M}^{(4)}$ and $\mathbf{S} \in \mathbb{R}^{2\times 2}$, we have:

$$\mathbf{V}^{(8)}\vec{\mathbf{x}}_w^{(N,j)}=\left(\begin{bmatrix}\vec{\mathbf{m}}^{(4,0)^T}\\ \vec{\mathbf{m}}^{(4,1)^T}\\ \vec{\mathbf{m}}^{(4,2)^T}\\ \vec{\mathbf{m}}^{(4,3)^T}\end{bmatrix}\otimes\begin{bmatrix}\vec{\mathbf{s}}^{(0)^T}\\ \vec{\mathbf{s}}^{(1)^T}\end{bmatrix}\right)\vec{\mathbf{x}}_w^{(N,j)}$$
$$=\begin{bmatrix}\vec{\mathbf{m}}^{(4,0)^T}\otimes\vec{\mathbf{s}}^{(0)^T}\\ \vec{\mathbf{m}}^{(4,0)^T}\otimes\vec{\mathbf{s}}^{(1)^T}\\ \vec{\mathbf{m}}^{(4,1)^T}\otimes\vec{\mathbf{s}}^{(0)^T}\\ \vec{\mathbf{m}}^{(4,1)^T}\otimes\vec{\mathbf{s}}^{(1)^T}\\ \vec{\mathbf{m}}^{(4,2)^T}\otimes\vec{\mathbf{s}}^{(0)^T}\\ \vec{\mathbf{m}}^{(4,2)^T}\otimes\vec{\mathbf{s}}^{(1)^T}\\ \vec{\mathbf{m}}^{(4,3)^T}\otimes\vec{\mathbf{s}}^{(0)^T}\\ \vec{\mathbf{m}}^{(4,3)^T}\otimes\vec{\mathbf{s}}^{(1)^T}\end{bmatrix}\vec{\mathbf{x}}_w^{(N,j)}=\begin{bmatrix}y(8,0,j)\\ y(8,1,j)\\ y(8,2,j)\\ y(8,3,j)\\ y(8,4,j)\\ y(8,5,j)\\ y(8,6,j)\\ y(8,7,j)\end{bmatrix}, \quad (20)$$

where WHT basis vectors $\vec{\mathbf{m}}^{(4,0)}$, $\vec{\mathbf{m}}^{(4,1)}$, $\vec{\mathbf{m}}^{(4,2)}$ and $\vec{\mathbf{m}}^{(4,3)}$ are α^2-related at 0. $y(8,0,j)$, $y(8,2,j)$, $y(8,4,j)$ and $y(8,6,j)$ in (20) are α^2-related since they can be represented as follows:

$$\begin{bmatrix} y(8,0,j)\\ y(8,2,j)\\ y(8,4,j)\\ y(8,6,j)\end{bmatrix}=\left(\begin{bmatrix}\vec{\mathbf{m}}^{(4,0)^T}\\ \vec{\mathbf{m}}^{(4,1)^T}\\ \vec{\mathbf{m}}^{(4,2)^T}\\ \vec{\mathbf{m}}^{(4,3)^T}\end{bmatrix}\otimes\vec{\mathbf{s}}^{(0)^T}\right)\vec{\mathbf{x}}_w^{(N,j)}, \quad (21)$$

where $i^{(0)}$=0, $i^{(1)}$=2, $i^{(2)}$=4, $i^{(3)}$=6, $i_\mathbf{M}^{(0)}$=0, $i_\mathbf{M}^{(1)}$=1, $i_\mathbf{M}^{(2)}$=2, $i_\mathbf{M}^{(3)}$=3 for (15)-(19). Similarly, $y(8,1,j)$, $y(8,3,j)$, $y(8,5,j)$ and $y(8,7,j)$ in (20) are α^2-related. Table III summarizes the main definitions used for GCK.

TABLE II
THE α-INDEX FOR DYADIC-ORDERED WHT BASIS VECTOR.

Basis vector	$[1\ 1\ 1\ 1]^T$	$[1\ 1\ -1\ -1]^T$	$[1\ -1\ 1\ -1]^T$	$[1\ -1\ -1\ 1]^T$
$\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})}$	$\vec{\mathbf{m}}^{(4,0)}$	$\vec{\mathbf{m}}^{(4,1)}$	$\vec{\mathbf{m}}^{(4,2)}$	$\vec{\mathbf{m}}^{(4,3)}$
α-index	[00]	[01]	[10]	[11]

Basis vector	$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^T$	$[1\ 1\ 1\ 1\ -1\ -1\ -1\ -1]^T$	…
$\vec{\mathbf{m}}^{(N_M,i_M)}$	$\vec{\mathbf{m}}^{(8,0)}$	$\vec{\mathbf{m}}^{(8,1)}$	…
α-index	[000]	[001]	…

TABLE III
SYMBOLS AND TERMS DEFINED FOR GCK.

	Meanings
N, $N_\mathbf{M}$	Length of vector.
i, $i_\mathbf{M}$	Index of basis vector in a matrix.
j	Index of input elements and windows.
$\vec{\mathbf{x}}_w^{(N,j)}$	The input window starting at x_j having length N: $[x_j, x_{j+1}, \ldots, x_{j+N-1}]^T$.
$\mathbf{M}^{(N_\mathbf{M})}$	1D $N_\mathbf{M}\times N_\mathbf{M}$ WHT matrix.
$\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})}$	The $i_\mathbf{M}$th WHT basis vector having length $N_\mathbf{M}$.
$\mathbf{V}^{(N)}$	1D GCK matrix $\mathbf{V}^{(N)} = \mathbf{M}^{(N_\mathbf{M})}\otimes\mathbf{S}$.
$\vec{\mathbf{v}}^{(N,i)}$	The ith GCK basis vector having length N.
$\mathbf{V}_s^{(N)}$	The SegGCK matrix $\mathbf{V}_s^{(N)} = \mathbf{I}_{L_s}\otimes\mathbf{V}^{(N_S)}$. Details are given in Section V.
$y(N,i,j)$	$y(N,i,j) = \vec{\mathbf{v}}^{(N,i)^T}\vec{\mathbf{x}}_w^{(N,j)}$. The ith order-N GCK projection value at the jth window.
α-index of $i_\mathbf{M}$	Sequence $[\alpha_{G_\mathbf{M}-1}\ldots\alpha_g\ldots\alpha_1\ \alpha_0]$, which is the binary representation of $i_\mathbf{M}$: $i_\mathbf{M} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1}+\ldots+\alpha_0 2^0$.
$\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(0)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(1)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(2)})}$ and $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(3)})}$ are α^2-related at g	The α-indices of $i_\mathbf{M}^{(0)}$, $i_\mathbf{M}^{(1)}$, $i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ are only different at α_g and α_{g+1}.

III. THE G4-GCK ALGORITHM FOR GCK

This section gives an example of computing 1D order-4 WHT on sliding windows using the G4-GCK algorithm in Section III-A, and then describes the G4-GCK algorithm for 1D order-N GCK.

A. The G4-GCK algorithm for Order-4 WHT

If $\mathbf{S} = \mathbf{I}_1$, then GCK matrix $\mathbf{V}^{(N)}$ is the order-N WHT matrix $\mathbf{M}^{(N)}$. When $N = 4$, we have:

$$\vec{\mathbf{y}}^{(4,j)}=\begin{bmatrix}y(4,0,j)\\ y(4,1,j)\\ y(4,2,j)\\ y(4,3,j)\end{bmatrix}=\frac{1}{2}\begin{bmatrix}1&1&1&1\\ 1&1&-1&-1\\ 1&-1&1&-1\\ 1&-1&-1&1\end{bmatrix}\begin{bmatrix}x_j\\ x_{j+1}\\ x_{j+2}\\ x_{j+3}\end{bmatrix}. \quad (22)$$

Let $\Delta(4,j) = y(4,0,j) - y(4,0,j+1)$. Equation (22)

implies that:

$$\begin{aligned}
&\begin{bmatrix} y(4,0,j)-y(4,0,j+1) \\ y(4,3,j)+y(4,1,j+1) \\ y(4,2,j)+y(4,2,j+1) \\ y(4,1,j)-y(4,3,j+1) \end{bmatrix} \\
=&\frac{1}{2}\begin{bmatrix} (x_j+x_{j+1}+x_{j+2}+x_{j+3})-(x_{j+1}+x_{j+2}+x_{j+3}+x_{j+4}) \\ (x_j-x_{j+1}-x_{j+2}+x_{j+3})+(x_{j+1}+x_{j+2}-x_{j+3}-x_{j+4}) \\ (x_j-x_{j+1}+x_{j+2}-x_{j+3})+(x_{j+1}-x_{j+2}+x_{j+3}-x_{j+4}) \\ (x_j+x_{j+1}-x_{j+2}-x_{j+3})-(x_{j+1}-x_{j+2}-x_{j+3}+x_{j+4}) \end{bmatrix} \\
=&\frac{1}{2}\begin{bmatrix} x_j-x_{j+4} \\ x_j-x_{j+4} \\ x_j-x_{j+4} \\ x_j-x_{j+4} \end{bmatrix}=\begin{bmatrix} y(4,0,j)-y(4,0,j+1) \\ y(4,0,j)-y(4,0,j+1) \\ y(4,0,j)-y(4,0,j+1) \\ y(4,0,j)-y(4,0,j+1) \end{bmatrix}=\begin{bmatrix} \Delta(4,j) \\ \Delta(4,j) \\ \Delta(4,j) \\ \Delta(4,j) \end{bmatrix}.
\end{aligned} \tag{23}$$

Hence, $\Delta(4,j)$ relates the projection values in window j and $j+1$. Equation (23) implies

$$\begin{bmatrix} y(4,1,j+1) \\ y(4,2,j+1) \\ y(4,3,j+1) \end{bmatrix}=\begin{bmatrix} -y(4,3,j) \\ -y(4,2,j) \\ y(4,1,j) \end{bmatrix}+\begin{bmatrix} \Delta(4,j) \\ \Delta(4,j) \\ -\Delta(4,j) \end{bmatrix}. \tag{24}$$

When we compute the projection values in window $j+1$ on sliding windows, the projection values in windows 0, 1, ..., j have been computed. Suppose $y(4,0,j+1)$ has been obtained by other approaches, the G4-GCK algorithm: 1) computes $\Delta(4,j)$ as $\Delta(4,j)=y(4,0,j)-y(4,0,j+1)$ by 1 addition; 2) uses $\Delta(4,j)$ and projection values in window j for computing the 3 projection values in window $j+1$ by 3 additions using (24). Computation of $y(4,0,j+1)$ is analyzed in Section III-B.

The GCK algorithm in [24] computes the WHT projection values in window $j+1$ from the projection values in window j and $j-1$ as follows:

$$y(4,1,j+1)=y(4,0,j-1)-y(4,0,j+1)-y(4,1,j-1), \tag{25}$$
$$y(4,3,j+1)=y(4,1,j)-y(4,1,j+1)-y(4,3,j), \tag{26}$$
$$y(4,2,j+1)=y(4,2,j-1)-y(4,3,j-1)-y(4,3,j+1). \tag{27}$$

Given $y(4,0,j+1)$, the GCK algorithm will: 1) compute the 1st projection value, i.e. $y(4,1,j+1)$, from the 0th projection values $y(4,0,j-1)$ and $y(4,0,j+1)$ by 2 additions in (25); 2) compute the 3rd projection value from the 1st projection values by 2 additions in (26); 3) compute the 2nd projection value from the 3rd projection values by 2 additions in (27).

B. The G4-GCK algorithm for Order-N GCK

The theorem below is proved in Appendix A in [39].

Theorem 1: If four order-N GCK projection values $y(N,i^{(n)},j)=[\vec{\mathbf{m}}^{(N_{\mathbf{M}},i^{(n)}_{\mathbf{M}})}\otimes\vec{\mathbf{s}}^{(i_r)}]^T\vec{\mathbf{x}}_w^{(N,j)}$ for $n=0,\ldots,3$ as defined in (10) are α^2-related at g, where $i^{(n)}_{\mathbf{M}}$ are represented in (16)-(19), $\vec{\mathbf{s}}^{(i_r)}$ has length R, $2^{G_{\mathbf{M}}}=N_{\mathbf{M}}$, then we have:

$$\begin{bmatrix} y(N,i^{(1)},j+R_4) \\ y(N,i^{(2)},j+R_4) \\ y(N,i^{(3)},j+R_4) \end{bmatrix}=\begin{bmatrix} -y(N,i^{(3)},j) \\ -y(N,i^{(2)},j) \\ y(N,i^{(1)},j) \end{bmatrix}+\begin{bmatrix} \Delta(N,j,R_4) \\ \Delta(N,j,R_4) \\ -\Delta(N,j,R_4) \end{bmatrix}, \tag{28}$$

$$\text{where}\quad R_4=2^{G_{\mathbf{M}}-g-2}R, \tag{29}$$

$$\Delta(N,j,R_4)=y(N,i^{(0)},j)-y(N,i^{(0)},j+R_4). \tag{30}$$

Table IV shows the steps and corresponding number of additions required using Theorem 1 to compute the 3 projection values $y(N,i^{(n)},j+R_4)$ for $n=1,2,3$ from $y(N,i^{(0)},j+R_4)$. The computation in (28) corresponds to the example in (24), where N=4, $i^{(0)}$=0, $i^{(1)}$=1, $i^{(2)}$=2, $i^{(3)}$=3, $y(4,i,j)$ for $i=0,1,2,3$ are α^2-related at 0 (thus $g=0$), $G_{\mathbf{M}}=2$, $R=1$, so $R_4=2^{2-0-2}\cdot 1=1$ and $\Delta(N,j,R_4)=\Delta(4,j)$.

TABLE IV
COMPUTATION OF ORDER-N GCK USING THE G4-GCK ALGORITHM

Step a $y(N, i^{(0)}, j+R_4)$ is provided by other approaches.
- The computation required is not counted.

Step b $\Delta(N, j, R_4) = y(N, i^{(0)}, j) - y(N, i^{(0)}, j+R_4)$
- One addition is required.

Step c $y(N, i^{(1)}, j+R_4) = \Delta(N, j, R_4) - y(N, i^{(3)}, j)$,
$y(N, i^{(2)}, j+R_4) = \Delta(N, j, R_4) - y(N, i^{(2)}, j)$,
$y(N, i^{(3)}, j+R_4) = y(N, i^{(1)}, j) - \Delta(N, j, R_4)$.
- Three additions are required. Note that $y(N, i^{(n)}, j)$ for n=0, …, 3 are obtained during previous computation.

Theorem 1 shows how to compute the other 3 projection values from $y(N,i^{(0)},j+R_4)$. The following corollary shows that we can compute $y(N,i^{(0)},j+R_4)$ from one of the other 3 projection values:

Corollary 2:

$$y(N,i^{(0)},j+R_4)=y(N,i^{(0)},j)-\Delta(N,j,R_4), \tag{31}$$

$$\begin{aligned}
\Delta(N,j,R_4)&=y(N,i^{(3)},j)+y(N,i^{(1)},j+R_4) \\
&=y(N,i^{(2)},j)+y(N,i^{(2)},j+R_4) \\
&=y(N,i^{(1)},j)-y(N,i^{(3)},j+R_4).
\end{aligned} \tag{32}$$

Equation (31) is derived from (30); (32) is from (28). The computation of $\vec{\mathbf{y}}^{(N,j)}$ using (28)-(32) is called the G4-GCK algorithm because it groups 4 GCK projection values for computation. Here is a summary of the G4-GCK algorithm. If one of the 4 α^2-related projection values $y(N,i^{(n)},j+R_4)$ for $n=0,\ldots,3$ is provided, then we can: 1) use this provided projection value to obtain $\Delta(N,j,R_4)$ by 1 addition using (30) or (32); 2) obtain the other 3 projection values by 3 additions using (28), (31). Therefore, the G4-GCK algorithm requires 4 additions for obtaining 3 projection values, i.e. 4/3 additions per window per projection value independent of transform size N. In comparison, the GCK algorithm requires 2 additions per projection value and the algorithm in [25] requires 3/2 additions.

The GCK algorithm, the algorithm in [25] and the G4-GCK algorithm assume that one projection value is provided on all window positions. Let the number of additions per window required for computing this projection value be B_1. The GCK algorithm requires $2(u-1)+B_1$ additions per window for obtaining u projection values and the G4-GCK algorithm requires $4(u-1)/3+B_1$ additions. For example, the G4-GCK algorithm requires $4(N-1)/3+B_1$ additions per window and the GCK algorithm requires $2(N-1)+B_1$ additions per window for computing N WHT projection values. In general, the G4-GCK algorithm requires $4(N-R)/3+B_1R$ additions for computing N GCK projection values when R is not necessarily 1. Normally, this provided projection value is the dc component for 2D WHT and can be computed using box-technique [40] by $B_1=4$ additions per window, or using integral image [41] by $B_1=3$ additions per window or using strip sum [42] by $B_1=3$ additions per window. Since B_1 is a constant and is relatively small compared with the computation required by u when u is large, B_1 is skipped in [24] and in the following of this paper.

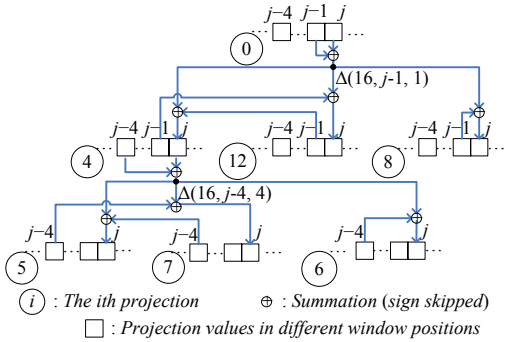


Fig. 3. Utilization of the G4-GCK algorithm for obtaining the other projection values from the 0th projection value. The number i in circle denotes the ith projection. The rectangles denote projection values in different window positions. The signs are skipped for the summation operations in this figure.

Generally, if we are provided with one of the $N_{\mathbf{M}}$ projection values in window j, i.e. $y(N,i,j) = [\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})} \otimes \vec{\mathbf{s}}^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$ for $i_{\mathbf{M}} = 0,\ldots,N_{\mathbf{M}}-1$, then we can efficiently compute the remaining $N_{\mathbf{M}}-1$ projection values. As a special case, when $\vec{\mathbf{s}} = \mathbf{I}_1$, GCK is WHT and we have $R = 1$. Using order-16 WHT as an example, we show in Fig. 3 how to obtain all projection values in window j with the 0th projection value provided:

1) Since the 0th, 4th, 8th and 12th WHT projection values having α-indices [0000], [0100], [1000] and [1100] respectively are α^2-related at 2, i.e. $g = 2$, the G4-GCK algorithm computes the $y(16,4,j)$, $y(16,8,j)$ and $y(16,12,j)$ from the $y(16,0,j)$, which is shown in Fig. 3. In this case, we have $g = 2$, $R = 1$, $N = 16$ and $G_{\mathbf{M}} = 4$ and the following from Theorem 1:

$$\begin{bmatrix} y(16,4,j) \\ y(16,8,j) \\ y(16,12,j) \end{bmatrix} = \begin{bmatrix} -y(16,12,j-1) \\ -y(16,8,j-1) \\ y(16,4,j-1) \end{bmatrix} + \begin{bmatrix} \Delta(16,j-1,1) \\ \Delta(16,j-1,1) \\ -\Delta(16,j-1,1) \end{bmatrix},$$

$$\text{where} \quad R_4 = 2^{G_{\mathbf{M}}-g-2} R = 2^{4-2-2} \cdot 1 = 1,$$

$$\Delta(16,j-1,1) = y(16,0,j-1) - y(16,0,j). \tag{33}$$

2) The projection value $y(16,4,j)$ obtained in the previous step can be used to obtain the projection values $y(16,5,j)$, $y(16,6,j)$ and $y(16,7,j)$ because these 4 projection values are α^2-related at $g = 0$, which is shown in Fig. 3. In this case, we have $g = 0$, $R = 1$, $N = 16$, $G_{\mathbf{M}} = 4$ and the following from Theorem 1:

$$\begin{bmatrix} y(16,5,j) \\ y(16,6,j) \\ y(16,7,j) \end{bmatrix} = \begin{bmatrix} -y(16,7,j-4) \\ -y(16,6,j-4) \\ y(16,5,j-4) \end{bmatrix} + \begin{bmatrix} \Delta(16,j-4,4) \\ \Delta(16,j-4,4) \\ -\Delta(16,j-4,4) \end{bmatrix},$$

$$\text{where} \quad R_4 = 2^{G_{\mathbf{M}}-g-2} R = 2^{4-0-2} \cdot 1 = 4,$$

$$\Delta(16,j-4,4) = y(16,4,j-4) - y(16,4,j). \tag{34}$$

3) The remaining projection values are similarly obtained.

IV. The G4-GCK algorithm for High Dimensions

In the previous sections, we have illustrated the proposed algorithm for 1D GCK where the input window is a vector. The proposed algorithm can be used for computing GCK of higher dimensions, e.g. 2D for images. As proved in Appendix C in [39], the computation required is 4/3 additions per projection value per window independent of dimension and size.

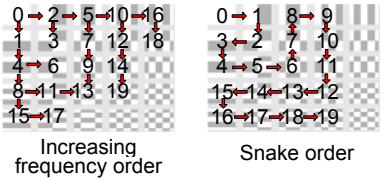


Fig. 4. Snake order and increasing frequency order proposed in [7] for WHT. Numbers denote the order. Arrows denote the computation dependence. Projection value 1 is computed from 0 in both orders.

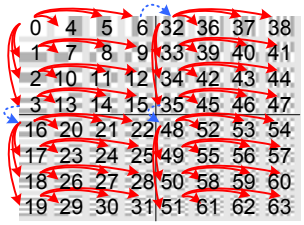


Fig. 5. Ordering 2D 8×8 WHT for the proposed algorithm. Numbers denote the order. Solid arrows denote the computation dependence using the proposed algorithm while dashed arrows denote the computation dependence using the GCK algorithm in [7]. For example, projection values 4, 5 and 6 are computed from 0 by the proposed algorithm while 32 is computed from 6 by the GCK algorithm in [7].

A. Ordering GCK Projection Values

The Gray-code sequence (GCS) is proposed in [24] to order the GCK projection values so that the current projection value can be efficiently computed from the previously computed projection values using the GCK algorithm. To order 2D projection values, Moshe and Hel-Or propose two orders for computing GCK on images in [7]. They are the snake order and increasing frequency order. Fig. 4 shows the two orders for the first 20 2D WHT projection values.

To efficiently compute projection values using the G4-GCK algorithm for 2D natural images, we can use the order shown in Fig. 5. For this order, 16 projection values form a group. Groups are arranged in an increasing frequency order and projection values within a group have a fixed order. This order for larger sizes of WHT or GCK can be similarly obtained. The snake order in Fig. 4 is suggested in [7] for the GCK algorithm. It can be seen that the first 16 projection values in snake order are the same as the group of 16 projection values in Fig. 5. In the experimental results, we show that the G4-GCK algorithm using this order is faster than the GCK algorithm using the increasing frequency order for pattern matching.

When specific projection values are required, we recommend constructing the GCS first and then examining the α^2-related projection values in the GCS for using the G4-GCK algorithm.

V. The Segmented GCK

A. The Definition of Segmented GCK

Define the order-N Segmented GCK (SegGCK) matrix $\mathbf{V}_s^{(N)}$ of identity matrix $\mathbf{I}_{L_s}$ and GCK matrix $\mathbf{V}^{(N_s)}$ as

follows:

$$\begin{aligned}\mathbf{V}_s^{(N)} &= [\vec{\mathbf{v}}_s^{(N,0)}, \ldots, \vec{\mathbf{v}}_s^{(N,i)}, \ldots, \vec{\mathbf{v}}_s^{(N,N-1)}]^T \\ &= \mathbf{I}_{L_s} \otimes \mathbf{V}^{(N_s)} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} & 0 & 0 & 0 \\ 0 & \mathbf{V}^{(N_s)} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{V}^{(N_s)} \end{bmatrix} \end{aligned} \quad (35)$$

where $L_s = 2, 3, 4, \ldots$ is the segmentation parameter, $N = L_s N_s$, $\vec{\mathbf{v}}_s^{(N,i)}$ for $i = 0, \ldots, N-1$ is the ith SegGCK basis vector having length N and subscript $\cdot_s$ is used for denotation related to SegGCK. The order-N SegGCK matrix is an $N \times N$ matrix. It is easy to see from (35) that $\mathbf{V}_s^{(N)}$ is orthogonal if $\mathbf{V}^{(N_s)}$ is orthogonal. Thus we can make $\mathbf{V}^{(N_s)}$ orthogonal to obtain orthogonal SegGCK matrix and apply SegGCK for transform domain pattern matching introduced in Section I-A.

Let $\vec{\mathbf{y}}_s^{(N,j)}$ for $j = 0, \ldots, W-1$ be the SegGCK projection value vector containing all projection values at the jth window and

$$\begin{aligned}\vec{\mathbf{y}}_s^{(N,j)} &= \mathbf{V}_s^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \\ &= [y_s(N,0,j), \ldots, y_s(N,i,j) \ldots, y_s(N,N-1,j)]^T, \end{aligned} \quad (36)$$

where $y_s(N,i,j) = \vec{\mathbf{v}}_s^{(N,i)^T} \vec{\mathbf{x}}_w^{(N,j)}$ is the ith SegGCK projection value for the jth window.

When the GCK matrix $\mathbf{V}^{(N_s)}$ of SegGCK matrix is a WHT matrix $\mathbf{M}^{(N_s)}$, we call it the Segmented WHT (SegWHT). For example, when $L_s = 2, N_s = 4, N = 8$ and $\mathbf{V}^{(N_s)} = \mathbf{M}^{(4)}$ in (35), we have an order-8 SegWHT matrix as follows:

$$\mathbf{V}_s^{(8)} = \mathbf{I}_2 \otimes \mathbf{M}^{(4)} = \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix}. \quad (37)$$

B. Fast Segmented GCK Algorithm

The jth input window of length N can be represented by L_s subwindows of length N_s as follows:

$$\vec{\mathbf{x}}_w^{(N,j)} = \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ \vdots \\ x_{j+N-1} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vec{\mathbf{x}}_w^{(N_s,j+2\cdot N_s)} \\ \vdots \\ \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)} \end{bmatrix}, \quad (38)$$

From (35), (36) and (38), the SegGCK projection value vector can be represented by GCK projection value vectors as follows:

$$\begin{aligned}\vec{\mathbf{y}}_s^{(N,j)} &= \mathbf{V}_s^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} & 0 & 0 & 0 \\ 0 & \mathbf{V}^{(N_s)} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{V}^{(N_s)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vdots \\ \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)}) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vdots \\ \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)}) \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(N_s,j)} \\ \vec{\mathbf{y}}^{(N_s,j+N_s)} \\ \vdots \\ \vec{\mathbf{y}}^{(N_s,j+(L_s-1)\cdot N_s)} \end{bmatrix}, \end{aligned} \quad (39)$$

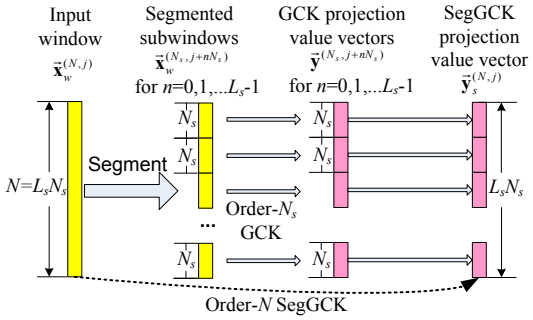


Fig. 6. The order-N SegGCK that can be computed by segmenting input window into L_s subwindows having length N_s and then computing order-N_s GCK on the L_s subwindows.

where $\vec{\mathbf{x}}_w^{(N,j)}$ is the jth input window having length $N(= L_s N_s)$ and $\vec{\mathbf{x}}_w^{(N_s,j)}$ is the jth window having length N_s; $\vec{\mathbf{y}}_s^{(N,j)}$ is the SegGCK projection value vector at the jth window and $\vec{\mathbf{y}}^{(N_s,j)}$ is the GCK projection value vector at the jth window. In (39), we segment the length-N input window $\vec{\mathbf{x}}_w^{(N,j)}$ into L_s length-N_s subwindows $\vec{\mathbf{x}}_w^{(N_s,j)}, \ldots, \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)N_s)}$ and then compute order-N_s GCK on these L_s subwindows. This procedure is shown in Fig. 6. Therefore, the projection of a window is decomposed into projections of its subwindows.

According to the result in (39), obtaining the SegGCK projection value vector $\vec{\mathbf{y}}_s^{(N,j)}$ in the jth window is equivalent to obtaining the L_s GCK projection value vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 0, \ldots, L_s - 1$. Similarly, obtaining $\vec{\mathbf{y}}_s^{(N,j+N_s)}$ in the $j + N_s$th window is equivalent to obtaining L_s vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 1, \ldots, L_s$, where the $L_s - 1$ vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 1, \ldots, L_s - 1$ have been computed previously in $\vec{\mathbf{y}}_s^{(N,j)}$ and only $\vec{\mathbf{y}}^{(N_s,j+L_sN_s)}$ is not computed previously. Fig. 7 shows the relationship between $\vec{\mathbf{y}}_s^{(N,j)}$ and $\vec{\mathbf{y}}_s^{(N,j+N_s)}$. When we compute $\vec{\mathbf{y}}_s^{(N,j+N_s)}$, we need only obtain $\vec{\mathbf{y}}^{(N_s,j+L_sN_s)}$ by about $4N_s/3$ additions using the G4-GCK algorithm. Thus the SegGCK algorithm requires about $4N_s/3$ additions to obtain $N(= L_sN_s)$ SegGCK projection values per window, while the GCK algorithm requires $2N$ additions and the algorithm in [25] requires $3N/2$ additions for obtaining N GCK projection values. On average, the fast SegGCK algorithm requires $4/(3L_s)$ addition(s) per projection value.

As an example, we have the following for the order-8 SegWHT in (37):

$$\begin{aligned}\vec{\mathbf{y}}_s^{(8,j)} &= \mathbf{V}_s^{(8)} \vec{\mathbf{x}}_w^{(8,j)} = (\mathbf{I}_2 \otimes \mathbf{M}^{(4)}) \vec{\mathbf{x}}_w^{(8,j)} \\ &= \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(4,j)} \\ \vec{\mathbf{x}}_w^{(4,j+4)} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(4,j)} \\ \vec{\mathbf{y}}^{(4,j+4)} \end{bmatrix} \end{aligned} \quad (40)$$

$$\begin{aligned}\vec{\mathbf{y}}_s^{(8,j+4)} &= \mathbf{V}_s^{(8)} \vec{\mathbf{x}}_w^{(8,j+4)} = (\mathbf{I}_2 \otimes \mathbf{M}^{(4)}) \vec{\mathbf{x}}_w^{(8,j+4)} \\ &= \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(4,j+4)} \\ \vec{\mathbf{x}}_w^{(4,j+8)} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(4,j+4)} \\ \vec{\mathbf{y}}^{(4,j+8)} \end{bmatrix} \end{aligned} \quad (41)$$

The relationships in (40) and (41) are examples for (39), where $\mathbf{V}^{(N_s)} = \mathbf{M}^{(4)}$, $L_s = 2, N_s = 4$ and $N = 8$. According to the result in (40) and (41), obtaining $\vec{\mathbf{y}}_s^{(8,j)}$ is equivalent to obtaining $\vec{\mathbf{y}}^{(4,j)}$ and $\vec{\mathbf{y}}^{(4,j+4)}$. Similarly, obtaining $\vec{\mathbf{y}}_s^{(8,j+4)}$ is equivalent to obtaining $\vec{\mathbf{y}}^{(4,j+4)}$ and $\vec{\mathbf{y}}^{(4,j+8)}$, where $\vec{\mathbf{y}}^{(4,j+4)}$ has been obtained in $\vec{\mathbf{y}}_s^{(8,j)}$. This example is depicted in Fig. 7. If we compute $\vec{\mathbf{y}}_s^{(8,j+4)}$ on sliding windows: 1) its 4 elements in vector $\vec{\mathbf{y}}^{(4,j+4)}$ have been computed when we compute $\vec{\mathbf{y}}_s^{(8,j)}$; 2) the other 4 elements in $\vec{\mathbf{y}}^{(4,j+8)}$ can be computed by

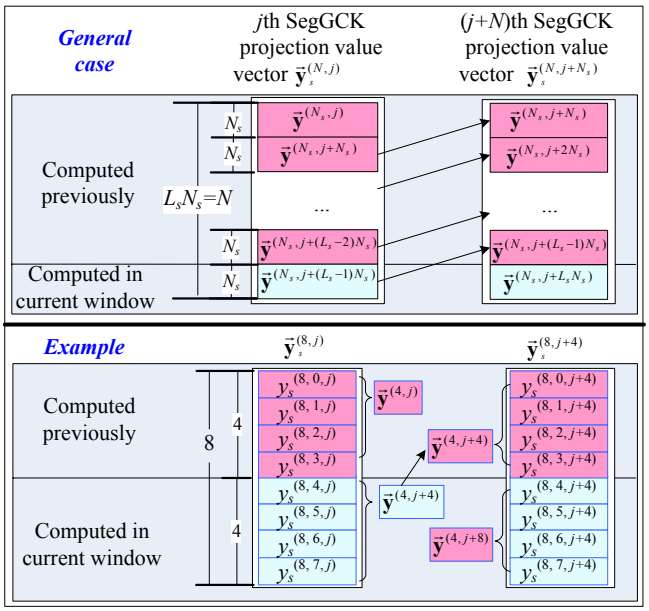


Fig. 7. Computing order-N SegGCK on sliding windows. For general case, SegGCK projection value vectors $\vec{\mathbf{y}}_s^{(N,j)}$ and $\vec{\mathbf{y}}_s^{(N,j+N_s)}$ share L_s-1 GCK projection value vectors. Arrows in the figure point out the L_s-1 shared GCK projection value vectors. For (37), we have $L_s=2$, $N_s=4$ and $N=8$.

about $16/3(=4\cdot 4/3)$ additions using the G4-GCK algorithm. Thus the fast SegGCK algorithm requires about $16/3$ additions for obtaining the 8 projection values in $\vec{\mathbf{y}}_s^{(8,j+4)}$, i.e. 2/3 additions per projection value on average.

Since the memory storing $\vec{\mathbf{y}}^{(N_s,j)}$ are regarded as SegGCK projection values at L_s different input window positions, $\vec{\mathbf{y}}^{(N_s,j)}$ will be accessed multiple times, which improves memory utilization and saves memory access time when these data are found multiple times in the data cache.

For input window having length N, the L_s basis vectors proposed in [22] can be represented by SegGCK basis vectors while SegGCK contains the other $N-L_s$ basis vectors that cannot be represented in [22]. The method in [31] segments non-rectangular patterns into certain number of rectangles aiming at dealing with non-rectangular pattern matching; the SegGCK segments rectangular pattern into L_s rectangles aiming at improving the computational efficiency. The SegGCK is inspired by the Incremental Dissimilarity Approximations (IDA) algorithm [26], in which Tombari *et al.* achieve computational efficiency by segmenting input data into several parts. The IDA algorithm determines a succession of rejection conditions characterized by increasing rejection ability and computational complexity. The differences between SegGCK and IDA are as follows: 1) the IDA is not transform domain pattern matching algorithm while SegGCK is used for transform domain pattern matching; 2) the IDA segments input window into subwindows and uses the triangular inequality on subwindows as the rejection condition while the SegGCK aims at efficiently computing transformation on sliding windows; 3) as will be shown in the experimental results, the pattern matching using SegGCK is faster than IDA.

C. Relationship Between GCK and SegGCK

Let $\%$ denote the modulo operation. Denote $span(\mathbf{V}^{(u\times N)})$ as the space spanned by the u basis vectors in matrix

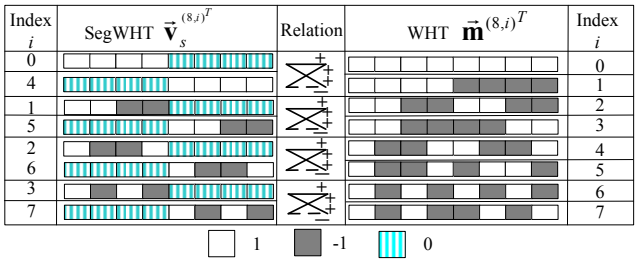


Fig. 8. The linear relationship among order-8 SegWHT and order-8 WHT, e.g. $\vec{\mathbf{m}}^{(8,0)}=\vec{\mathbf{v}}_s^{(8,0)}+\vec{\mathbf{v}}_s^{(8,4)}$, $\vec{\mathbf{m}}^{(8,1)}=\vec{\mathbf{v}}_s^{(8,0)}-\vec{\mathbf{v}}_s^{(8,4)}$. White represents the value +1, grey represents the value -1 and vertical strips represent the value 0. Normalization factors of basis vectors are skipped.

$\mathbf{V}^{(u\times N)}=[\vec{\mathbf{v}}^{(N,0)}\ \ldots\vec{\mathbf{v}}^{(N,u-1)}]^T$. The following theorem describes the links between GCK and SegGCK:

Theorem 3: If L_s GCK basis vectors in $\mathbf{V}_{GCK}^{(L_s\times N)}$ are represented as:

$$\vec{\mathbf{v}}^{(N,i)}=\vec{\mathbf{m}}^{(N_{\mathbf{M}},kL_s+i_s)}\otimes\vec{\mathbf{s}}^{(i_r)},\text{ for } i_s=0,\ldots,L_s-1, \quad (42)$$

where $N=N_{\mathbf{M}}R$, $N_{\mathbf{M}}\%L_s=0, k=0,\ldots,\frac{N_{\mathbf{M}}}{L_s}-1$, $\vec{\mathbf{s}}^{(i_r)}$ is fixed and WHT vectors $\vec{\mathbf{m}}^{(N_{\mathbf{M}},\cdot)}$ are in dyadic order or sequency order, then we can find L_s SegGCK basis vectors in $\mathbf{V}_{SegGCK}^{(L_s\times N)}$ so that:

1) $span(\mathbf{V}_{SegGCK}^{(L_s\times N)})=span(\mathbf{V}_{GCK}^{(L_s\times N)})$.
2) $\forall\vec{\mathbf{x}},\ ||\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2=||\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2$.
3) $4/(3L_s)$ additions and $4/3$ additions per projection value per window are required for computing $\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}$ and $\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}$ respectively, thus the computation required for SegGCK is $1/L_s$ of that required for GCK.

The proof for the theorem above is provided in Appendix B in [39]. According to Theorem 3, $||\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2=||\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2$ and so u GCK and SegGCK basis vectors pack the same energy and reject the same mismatched candidates when $u=L_s,2L_s,3L_s,\ldots$ for transform domain pattern matching in Table I. Theorem 3 can be used for WHT and SegWHT when we set $\vec{\mathbf{s}}^{(i_r)}=\mathbf{I}_1$ in (42). We use the example in (37) for illustrating the relationship between GCK and SegGCK. Fig. 8 shows that all order-8 WHT basis vectors can be linearly represented by SegWHT basis vectors. For example, when $k=0$, we have the dyadic ordered WHT vectors $\vec{\mathbf{m}}^{(8,0)}$ and $\vec{\mathbf{m}}^{(8,1)}$, where $N=N_{\mathbf{M}}=8, L_s=2, i_s=0,1, \vec{\mathbf{s}}^{(i_r)}=\mathbf{I}_1, R=1$ and $N_{\mathbf{M}}\%L_s=8\%2=0$ in (42). And we select $L_s(=2)$ SegWHT vectors $\vec{\mathbf{v}}_s^{(8,0)}$ and $\vec{\mathbf{v}}_s^{(8,4)}$ in (37). Orthonormal WHT basis vectors $\vec{\mathbf{m}}^{(8,0)}$ and $\vec{\mathbf{m}}^{(8,1)}$ can be linearly represented by orthonormal SegWHT basis vectors $\vec{\mathbf{v}}_s^{(8,0)}$ and $\vec{\mathbf{v}}_s^{(8,4)}$. Thus $span([\vec{\mathbf{m}}^{(8,0)}\ \vec{\mathbf{m}}^{(8,4)}]^T)=span([\vec{\mathbf{v}}_s^{(8,0)}\ \vec{\mathbf{v}}_s^{(8,4)}]^T)$ and $\forall\vec{\mathbf{x}},\ |||[\vec{\mathbf{m}}^{(8,0)}\ \vec{\mathbf{m}}^{(8,4)}]^T\vec{\mathbf{x}}||^2=|||[\vec{\mathbf{v}}_s^{(8,0)}\ \vec{\mathbf{v}}_s^{(8,4)}]^T\vec{\mathbf{x}}||^2$. As for computational requirement, the fast SegGCK algorithm computes the SegWHT projection values by about $2/3$ additions per projection value per window as we have illustrated in the example given by (40) and (41). The G4-GCK algorithm computes the WHT projection values by $4/3$ additions per window. Thus the computation required for SegWHT is $1/L_s(=1/2)$ of that required for WHT.

TABLE V
DATASETS WITH CORRESPONDING SIZES OF IMAGES AND PATTERNS USED IN THE EXPERIMENTS AND THE AVERAGE TIME IN SECONDS REQUIRED BY THE FS ON THESE DATASETS FOR ONE IMAGE-PATTERN PAIR.

Dataset	Image Size	Pattern Size	FS time
$S1$	160×120	16×16	0.0087
$S2$	320×240	32×32	0.1197
$S3$	640×480	64×64	1.8825
$S4$	1280×960	128×128	28.1889
$S5$	1280×960	64×64	8.5514
$S6$	1280×960	32×32	2.2929

VI. EXPERIMENTAL RESULTS

This section evaluates the performance of the proposed algorithm and SegGCK by comparing them with the FS and the other FS equivalent algorithms in pattern matching. All of the experiments were implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0. SSD is used for measuring the dissimilarity between a pattern and a candidate window.

A. Dataset and Algorithms Used for the Experiments

To investigate the computational efficiency of the proposed SegGCK for pattern matching, we shall compare the following FS equivalent algorithms with the FS:

- 1) FFT: the FFT-based approach in OpenCV [21];
- 2) IDA: the IDA algorithm in [26];
- 3) WHT: the WHT algorithm for WHT in [23];
- 4) GCK: the GCK algorithm for WHT in [24];
- 5) WHT_{G4}: the G4-GCK algorithm for WHT;
- 6) SegGCK: the SegGCK algorithm for SegGCK.

The code for WHT is available online [43] and the code for IDA is provided by the authors of [26]. The parameters for WHT use the default values in [43] and those for IDA are chosen according to [26]. For GCK, we choose the sequency-ordered WHT and the code is based on the code used for motion estimation in [7]. WHT_{GCK} is in the order introduced in Fig. 5.

As explained in [23], when the percentage of remaining candidate windows is smaller than certain number, denoted by ϵ, it is more efficient to use the FS-step for finding the matched windows instead of using transformation. In the experiments, the default setting is $\epsilon = 0.02\%$ and $L_s = 8$ for SegGCK. In the experiments, $\mathbf{S} = I_1$ is used for SegGCK, which corresponds to SegWHT. According to the authors' source code for WHT in [43], we set $\epsilon = 2\%$ as the default value for WHT and GCK. Variations of L_s and ϵ are given in Section VI-D.

Table V shows the 6 datasets used for evaluating the performance of the compared algorithms. The dataset includes different sizes of patterns and images. Our experiments include a total of 120 images chosen among three databases: MIT [44], medical [45], and remote sensing [46]. The MIT database is mainly concerned with indoor, urban, and natural environments, plus some object categories such as cars and fruits. The two other databases contain radiographs and Landsat satellite images. The 120 images have 4 resolutions which are 160×120, 320×240, 640×480 and 1280×960. Each resolution has 30 images. The OpenCV function 'cvResize' with linear interpolation is used for producing the desired resolution of images. For each image, 10 patterns were randomly selected among those showing a standard deviation of pixel intensities higher than a threshold (i.e., 45). Six datasets $S1$ to $S6$ with image and pattern sizes given in Table V were formed. Each dataset has 300 image-pattern pairs. Datasets $S1$ to $S4$ are the same as those in [26]. Datasets $S5$ and $S6$ are to investigate the effect of pattern size in pattern matching.

In the experiments, if the SSD between a candidate window and the pattern is below a threshold T, the candidate window is regarded to have matched the pattern. Similarly to the experiment in [26], the threshold T for an $N_1 \times N_2$ pattern is:

$$T = 1.1 \cdot SSD_{min} + N_1 N_2, \tag{43}$$

where SSD_{min} is the SSD between the pattern and the best matching window. In the experiments, SSD_{min} is decided by noise levels and types. Experiments on different noises will be shown in Section VI-B and VI-C.

Since all evaluated algorithms find the same matching windows as the FS, the only concern is computational efficiency which is measured by execution time in the experiments. The time speed-up of algorithm A over algorithm B is measured by the execution time required by B divided by that required by A. As an example, the time speed-up of GCK over FS is measured as the execution time required by FS divided by that required by GCK. The larger is the speed-up, the faster is GCK compared with FS.

All necessary preprocessing like the time required for transformation on patterns and the FS-step has been included in all experiments. The G4-GCK algorithm and the GCK algorithm require the availability of one projection value on all window positions, which needs to be computed by other approaches. This projection value is the dc component in the experiment and computed using the box-technique [40] by 4 additions per window position. The computation required for the dc component has been included in all experiments for the related algorithms.

B. Experiment 1 – Different Image-Pattern Sizes

In this experiment, we compare the time speed-ups yielded by the considered algorithms in pattern matching on the datasets $S1 - S4$ which have different sizes of image-pattern pairs. The time speed-ups yielded by GCK, IDA and SegGCK over FS in pattern matching for each dataset are shown in Fig. 9. It can be seen that SegGCK outperforms the other compared algorithms in the four different datasets. When there is no noise, small u (less than 4) is sufficient for rejecting a large amount of candidates. When u is small, WHT_{G4} does not obviously outperform WHT. SegGCK performs better when $\epsilon = 0.02\%$. In our experiments, we fix ϵ for each algorithm. As illustrated later in Fig. 14, setting $\epsilon = 0.02\%$ for GCK and WHT_{G4} is not a good choice if we take all noise levels into account.

C. Experiment 2 – Different Pattern Sizes and Noise Levels

To evaluate the performance of algorithms on the variation of pattern sizes with image size unchanged, we shall examine the experimental results on datasets $S4$-$S6$. In $S4$-$S6$, the

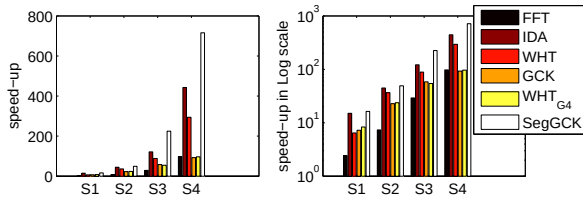


Fig. 9. Time speed-up over FS on datasets $S1-S4$ for different algorithms measured by normal scale (*left*) and log scale (*right*). The bars for each dataset from left to right correspond to algorithms FFT, IDA, WHT, GCK, WHT$_{G4}$ and SegGCK.

image size is always 1280×960, but the pattern size changes from 128×128 to 32×32. Moreover, there are 3 different kinds of noises having 4 noise levels added to each image. Four different levels of Gaussian noise having variances 100, 200, 400 and 800 were used for distorting images, which are referred to as $G(1)$, $G(2)$, $G(3)$ and $G(4)$ respectively. In this setting, the 512×512 distorted "couple" images have PSNR values 28.1, 25.1, 22.1 and 19.2 respectively when compared with the original image in Figure 1. Four different levels of Gaussian low-pass filters of standard deviation σ=0.2, 0.9, 1.6 and 2.3 were used for blurring each image, which are referred to as $B(1)$, $B(2)$, $B(3)$ and $B(4)$ respectively. In this setting, the 512×512 blurred "couple" images have PSNR values 27.79, 27.18, 25.36 and 24.14 respectively. Images of different JPEG compression quality levels were tested, which are referred to as $J(1)$, $J(2)$, $J(3)$ and $J(4)$, corresponding to quality measure $Q_{JPG} = 90, 70, 50$ and 30 respectively. Higher Q_{JPG} means higher quality. In this setting, the 512×512 compressed "couple" images have PSNR values 39.88, 34.93, 33.10 and 31.52 respectively.

The SSD_{min} in (43) is the SSD between the undistorted pattern and the distorted pattern. The distorted pattern and the distorted image have the same noise, e.g. $G(1)$. Since the threshold T in (43) is a bit greater than this SSD_{min}, the distorted pattern will always be found as a matched window if we put it into the image. Therefore, the miss rate is 0. As shown in Fig. 11, the false positives using the threshold in (43) are not greater than 0.0025% in $S4$-$S6$ for the 4 noise levels in 3 different noise types.

Fig. 10 shows the speed-ups of examined fast algorithms over FS in different sizes of patterns and different levels of noises. It can be seen that SegGCK is the fastest except for $B(3)$ and $B(4)$ in dataset $S4$ and WHT transform using the G4-GCK algorithm is usually faster than WHT transform using the GCK algorithm.

As the noise level increases, it is more difficult for the rejection step to eliminate mismatched candidates and the FS-step takes more time. The speed-up decreases for all rejection based algorithms, i.e. WHT, IDA, GCK, WHT$_{G4}$ and SegGCK. Therefore the improvement of WHT$_{G4}$ of GCK is not obvious in this case, e.g. B(4) for dataset S4. When we take Fig. 10 and 9 into account, WHT$_{G4}$ performs better than GCK when there is median noise level, where more than 4 projection values are required and the transformation time mainly influences the whole process. When the pattern size is 128×128 for dataset $S4$, the number of projection values used is the largest, and advantage of SegGCK over GCK is more obvious when there are more projection values computed.

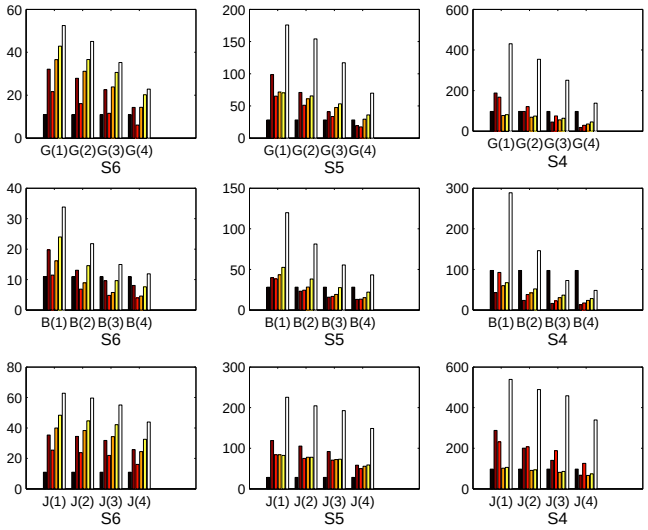


Fig. 10. Time speed-ups yielded by different algorithms over FS for Gaussian noise (upper row), image blur (middle row) and JPEG compression (bottom row) in different noise levels and different sizes of image-pattern pairs in pattern matching. Label of bars are the same as Fig. 9.

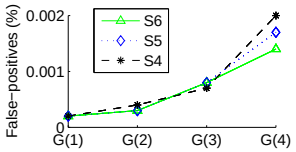


Fig. 11. False-positives (%) for noises $G(1)$-$G(4)$ in dataset $S4$-$S6$.

D. Experiment 3 – Parameters Selection

Experiments 1 and 2 use the default setting introduced in Section VI-A. This section investigates the influence of the parameters L_s and ϵ on the performance of pattern matching. First of all, in this section, we study the effect of parameter L_s for SegGCK on pattern matching performance. Then we investigate the effect of parameter ϵ for SegGCK and GCK on the performance in pattern matching.

Fig. 12 shows the influence of L_s on time speed-up of SegGCK over FS in dataset $S4$. In this experiment, SegGCK under different settings of L_s always outperforms WHT in pattern matching. And we can find that the speed-up increases as L_s increases from 4 to 8 while speed-up decreases as L_s increases to 32 and 64. Both 8 and 16 are good choices of L_s. 8 is chosen as a default setting because the best setting of L_s is 8 in most cases although 16 is slightly better than 8 for noise level $G(1)$. Here is an analysis of the result in Fig. 12. The computational efficiency of transform domain pattern matching is determined by both the energy packing ability of transform and the efficiency in computing transformation. Since the SegGCK requires $4/(3L_s)$ addition(s) per projection value, the larger is L_s, the more efficient is the computation of transformation. However, the energy packing ability degrades as L_s increases. In the extreme, when $N_s = 1$, the SegGCK matrix is simply identity matrix $\mathbf{I}_{L_s}$. In this case, no computation is required for computing projection values, but the energy packing ability of SegGCK is the worst. Fig. 13(a) shows the number of remaining windows after each projection for WHT and SegGCK with different L_s on dataset $S4$ with Noise $G(4)$. It can be seen that as L_s increases, the rejection power of SegGCK decreases. For L_s being 4 and

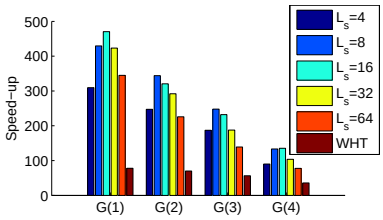


Fig. 12. The time speed-up over FS yielded by WHT and SegWHT with different L_s on dataset $S4$ with noise $G(1)$-$G(4)$.

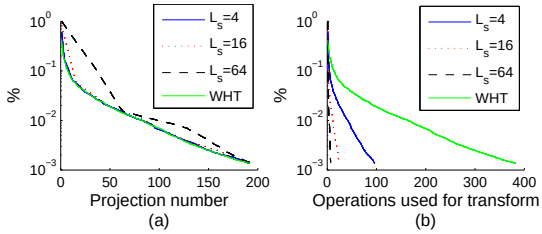


Fig. 13. The percentage of remaining windows, which is measured by the number of remaining windows divided by the number of all candidate windows, as a function of the number of projections (**a**) and the number of operations required by transform (**b**) on dataset $S4$ with Noise $G(4)$.

16, the rejection power of SegGCK is close to WHT. As L_s increases to 64, the rejection power is seriously affected, which explains why $L_s = 64$ is less efficient than $L_s = 16$ in Fig. 12. Thus the decision of L_s is dependent on the trade-off between energy packing ability of projection values and the computation required for obtaining these projection values. As shown in Fig. 13(b), SegGCK using the SegGCK algorithm requires much fewer operations than WHT using the GCK algorithm in rejecting the same number of candidate windows. This results in faster speed of SegGCK compared with GCK in Fig. 12.

To make the percentage of remaining candidate windows below ϵ for dataset $S4$, the average numbers of projection values required by SegGCK are: 42, 66, 101 and 123 for noise levels $G(1), G(2), G(3)$ and $G(4)$ respectively; 59, 83, 98 and 104 for $B(1), B(2), B(3)$ and $B(4)$ respectively; 26, 31, 36 and 54 for $J(1), J(2), J(3)$ and $J(4)$ respectively.

As introduced in Section V-C, it can be inferred from Theorem 3 that u 1D WHT and SegGCK basis vectors reject the same mismatched candidates when $u = L_s, 2L_s, 3L_s, \ldots$ for transform domain pattern matching in Table I. This can be similarly used for 2D transforms. Thus we can see from Fig. 13(a) that SegGCK rejects similar amount of candidate windows at certain projection number, e.g. 16 for $L_s = 16$, 64 for $L_s = 64$. The theoretical analysis in Appendix E shows that SegWHT is almost the same as WHT in energy packing ability.

As explained in [23], when the percentage of remaining candidate windows is smaller than certain number ϵ, it is more efficient to use SSD directly for finding the matched windows instead of using transformation. Fig. 14(a) shows the influence of ϵ for both GCK and SegGCK on dataset $S4$ with Gaussian noise $G(4)$. It can be seen that 2% is better for GCK while 0.02% is better for SegGCK. Setting $\epsilon = 2\%$ for both GCK and SegGCK is not fair for SegGCK while setting $\epsilon = 0.02\%$ for both GCK and SegGCK is not fair for GCK. Thus we have chosen their best setting, i.e. $\epsilon = 2\%$ for GCK and $\epsilon = 0.02\%$ for SegGCK, for evaluation. Fig. 14(b) shows that the transformation time required by SegGCK is obviously less

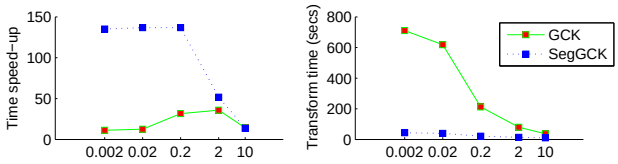


Fig. 14. **(a)** The speed-up over FS as a function of ϵ on the left figure and **(b)** the total transformation time in seconds required for 300 image-pattern pairs as a function of ϵ on the right figure. The FS-step in Table I is included for measuring time speed-up. Experiments were done on dataset $S4$ with Noise $G(4)$. ϵ denotes the percentage of remaining window below which FS is used for pattern matching, e.g. 2 and 10 in the X axis correspond to 2% and 10% respectively.

than the transformation time required by GCK for different situations of ϵ. When the whole time, i.e. transformation time and the time required for the FS step, is considered, Fig. 14(a) shows that SegGCK is more efficient than GCK for different settings of ϵ. As shown in Fig. 14, the speed-up increases as transformation time decreases when $\epsilon < 0.2\%$. When ϵ is too large, e.g. $\epsilon = 10\%$, the FS-step requires too much computational time and the whole process is less efficient; when ϵ is smaller, the transformation time is longer but the whole process is more efficient.

VII. Conclusions

This paper develops the G4-GCK algorithm for Gray code kernels (GCK). We then develop the Segmented GCK (SegGCK) which can be computed using a fast algorithm that is more efficient than existing algorithms for WHT and GCK.

The advantages of G4-GCK algorithm and SegGCK are summarized as follows.

- The G4-GCK algorithm which requires 4/3 additions per datum per projection value is faster than any known fast GCK algorithm.
- By segmenting input data into L_s parts, the SegGCK algorithm requires $4/(3L_s)$ addition(s) per projection value using the G4-GCK algorithm.
- The computational cost of G4-GCK algorithm and SegGCK algorithm are independent of the transform size and dimension.
- GCK and SegGCK are orthogonal transforms.

This paper describes the G4-GCK algorithm and SegGCK in the context of transform domain pattern matching. However, pattern matching is only an application example. The properties of the G4-GCK algorithm and SegGCK make them attractive for applications which require transformation on sliding windows such as image based rendering, image compression, super-resolution, object detection, texture synthesis, block matching in motion estimation, image denoising, action recognition and wide baseline image matching.

Acknowledgment

The authors wish to thank Prof. Yacov Hel-Or and Prof. Hagit Hel-Or for providing the thesis on generalized GCK and their code implementing GCK and WHT, Prof. Stefano Mattoccia and Dr. Federico Tombari for providing their code implementing IDA, image datasets and helpful discussion, Prof. Antonio Torralba and CSAIL for the use of the MIT database, Prof. Rainer Koster and the Institute for Clinical Radiology, Nuclear Medicine of the Lukas Hospital Neuss for

the use of the medical image database, and NASA for the use of the remote sensing image database.

REFERENCES

[1] M. S. Aksoy, O. Torkul, and I. H. Cedimoglu, "An industrial visual inspection system that uses inductive learning," *J. of Intelligent Manufacturing*, vol. 15, no. 4, pp. 569–574, 2004.

[2] A. Fitzgibbon, Y. Wexler, and A. Zisserman, "Image-based rendering using image-based priors," in *ICCV*, vol. 2, 2003, pp. 1176–1183.

[3] T. Luczak and W. Szpankowski, "A suboptimal lossy data compression based on approximate pattern matching," *IEEE Trans. Information Theory*, vol. 43, no. 5, pp. 1439–1451, 1997.

[4] W. Freeman, T. Jones, and E. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, Mar./Apr 2002.

[5] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," in *ICCV*, Sept. 1999, pp. 1033–1038.

[6] C. M. Mak, C. K. Fong, and W. K. Cham, "Fast motion estimation for H.264/AVC in Walsh Hadamard domain," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 5, pp. 735–745, Jun. 2008.

[7] Y. Moshe and H. Hel-Or, "Video block motion estimation based on Gray-code kernels," *IEEE Trans. Image Process.*, vol. 18, no. 10, pp. 2243–2254, Oct. 2009.

[8] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *CVPR*, vol. 2, Jun. 2005, pp. 60– 65.

[9] R. Zhang, W. Ouyang, and W. K. Cham, "Image postprocessing by non-local kuan's filter," *J. of Visual Communication and Image Representation*, vol. 22, pp. 251–262, 2011.

[10] Y. Alon, A. Ferencz, and A. Shashua, "Off-road path following using region classification and geometric projection constraints," in *CVPR*, vol. 1, Jun. 2006, pp. 689–696.

[11] Y. Hel-Or, H. Hel-Or, and E. David, "Fast template matching in non-linear tone-mapped images," in *ICCV*, 2011.

[12] Q. Wang and S. You, "Real-time image matching based on multiple view kernel projection," in *CVPR*, 2007.

[13] A. Goshtasby, *2-D and 3-D Image Registration for Medical, Remote Sensing and Industrial Applications*. New York: Wiley, 2005.

[14] B. Zitova and J. Flusser, "Image registration methods:a survey," *Image Vis. Comput.*, vol. 21, no. 11, pp. 977–1000, 2003.

[15] W. Krattenthaler, K. Mayer, and M. Zeiler, "Point correlation: A reduced-cost template matching technique," in *Proc. 1st IEEE Int. Conf. Image Processing*, vol. 1, Austin, TX, 1994, pp. 208–212.

[16] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Proc. SPIE AeroSense Symp.*, vol. 4387. SPIE, 2001, pp. 95–102. [Online]. Available: http://link.aip.org/link/?PSI/4387/95/1

[17] P. S. Heckbert, "Filtering by repeated integration," in *Proc. SIG-GRAPH*, 1986, pp. 315–321.

[18] H. Schweitzer, J. W. Bell, and F. Wu, "Very fast template matching," in *ECCV*, 2002, pp. 358–372.

[19] P. Simard, L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: A fast convolution algorithm for signal processing and neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 11, pp. 571–577, 1999.

[20] F. Tang, R. Crabb, and H. Tao, "Representing images using nonorthogonal Haar-like bases," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2120–2134, Dec. 2007.

[21] ;, "http://sourceforge.net/projects/opencvlibrary," accessed in 2012.

[22] M. G. Alkhansari, "A fast globally optimal algorithm for template matching using low-resolution pruning," *IEEE Trans. Image Process.*, vol. 10, no. 4, pp. 526–533, Apr 2001.

[23] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 9, pp. 1430–1445, Sept. 2005.

[24] G. Ben-Artz, H. Hel-Or, and Y. Hel-Or, "The Gray-code filter kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 382–393, Mar. 2007.

[25] W. Ouyang and W. K. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 165–171, Jan. 2010.

[26] F. Tombari, S. Mattoccia, and L. D. Stefano, "Full search-equivalent pattern matching with incremental dissimilarity approximations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 1, pp. 129–141, Jan. 2009.

[27] S. Mattoccia, F. Tombari, and L. D. Stefano, "Fast full-search equivalent template matching by enhanced bounded correlation," *IEEE Trans. Image Process.*, vol. 17, no. 4, pp. 528–538, Apr. 2008.

[28] B. Girod, *Whats Wrong with Mean-Squared Error?* MIT Press, 1993, ch. 15.

[29] S. Santini and R. Jain, "Similarity measures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 9, pp. 871–883, Sept. 1999.

[30] A. Ahumada, "Computational image quality metrics: A review," in *Proc. Soc. Information Display Intl Symp.*, vol. 24, 1998, pp. 305–308.

[31] M. Ben-Yehuda, L. Cadany, and H. Hel-Or, "Irregular pattern matching using projections," in *Proc. 12th Int'l Conf. Image Processing (ICIP)*, vol. 2, 2005, pp. 834–837.

[32] H. Schweitzer, R. Deng, and R. F. Anderson, "A dual bound algorithm for very fast and exact template-matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 459–470, Mar. 2011.

[33] S.-D. Wei and S.-H. Lai, "Fast template matching based on normalized cross correlation with adaptive multilevel winner update," *IEEE Trans. Image Process.*, vol. 17, no. 11, pp. 2227–2235, Nov. 2008.

[34] Y. Hel-Or, T. Malzbender, and D. Gelb, "Synthesis and rendering of 3d textures," in *Texture 2003 Workshop accomp. ICCV 2003*,, 2003, pp. 53–58.

[35] Q. Wang, J. Mooser, S. You, and U. Neumann, "Augmented exhibitions using natural features," *Int'l. J. Virtual Reality*, vol. 7, no. 4, pp. 1–8, 2008.

[36] G. Ben-Artzi, "Gray-code filter kernels (GCK)-fast convolution kernels," Master's thesis, Bar-Ilan Univ., , Ramat-Gan, Israel, 2004.

[37] W. K. Cham and R. Clarke, "Dyadic symmetry and Walsh matrices," *IEE Proceedings, Pt.F.*, vol. 134, no. 2, pp. 141–145, April 1987.

[38] A. Graham, *Kronecker Products and Matrix Calculus: With Applications*. 605 THIRD AVE., NEW YORK, NY 10158: JOHN WILEY & SONS, INC.,, 1982.

[39] ;, "Currently in the supplementary file "TPAMI-2011-03-0149-supp.pdf". Will be put online with the help of IEEE."

[40] M. J. McDonnell, "Box-filtering techniques," *Comput. Graph. Image Process.*, vol. 17, pp. 65–70, 1981.

[41] P. Viola and M. Jones, "Robust real-time face detection," *Int'l J. Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[42] W. Ouyang, R. Zhang, and W. K. Cham, "Fast pattern matching using orthogonal Haar transform," in *CVPR*, 2010.

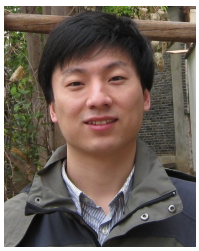
[43] ;, "www.faculty.idc.ac.il/toky/software/software.htm," accessed in 2012.

[44] ——, "http://people.csail.mit.edu/torralba/images," accessed in 2012.

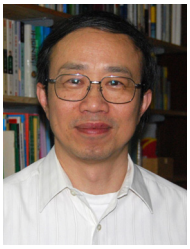
[45] ——, "www.data-compression.info/corpora/lukascorpus." accessed in 2012.

[46] ——, "http://zulu.ssc.nasa.gov/mrsid," accessed in 2012.

Wanli Ouyang received the B.S. degree in computer science from Xiangtan University, Hunan, China, in 2003. He received the M.S. degree in computer science from the College of Computer Science and Technology, Beijing University of Technology, Beijing, China. He received the PhD degree in the Department of Electronic Engineering, The Chinese University of Hong Kong, in which he is now a Postdoctoral Fellow. His research interests include image processing, computer vision and pattern recognition. He is a member of the IEEE.



Renqi Zhang received the B.S. degree in Electronic Engineering from Xidian University, Xi'an, China, in 2006. He received his Ph.D. degree from the Department of Electronic Engineering, The Chinese University of Hong Kong, in 2012. His research interests include image/video coding, computer vision and machine learning.



Wai-Kuen Cham graduated from The Chinese University of Hong Kong in 1979 in Electronics. He received his M.Sc. and Ph.D. degrees from Loughborough University of Technology, U.K., in 1980 and 1983 respectively. From June 1984 to April 1985, he was a senior engineer in Datacraft Hong Kong Limited and a lecturer in the Department of Electronic Engineering, Hong Kong Polytechnic (now The Polytechnic University of Hong Kong). Since May 1985, he has been with the Department of Electronic Engineering, The Chinese University of Hong Kong.