The Gray-Code Filter Kernels

Gil Ben-Artzi, Hagit Hel-Or, *Member, IEEE*, and Yacov Hel-Or, *Member, IEEE,*

Manuscript Submitted September, 2005.

G. Ben-Artzi is with the Dept of Computer Science at Bar-Ilan University, Ramat-Gan, Israel.

H. Hel-Or is with the Dept of Computer Science at The University of Haifa, Haifa, Israel.

Y. Hel-Or is with the School of Computer Science at The Interdisciplinary Center, Herzeliya, Israel.

Abstract

In this paper we introduce a family of filter kernels - the *Gray-Code Kernels (GCK)* and demonstrate their use in image analysis. Filtering an image with a sequence of Gray-Code Kernels is highly efficient and requires only *2 operations per pixel* for each filter kernel, independent of the size or dimension of the kernel. We show that the family of kernels is large and includes the Walsh-Hadamard kernels amongst others. The GCK can be used to approximate any desired kernel and as such forms a complete representation. The efficiency of computation using a sequence of GCK filters can be exploited for various real-time applications, such as, pattern detection, feature extraction, texture analysis, texture synthesis, and more.

Index Terms

Image Filtering, Filters, Filter Kernels, Convolution, Walsh-Hadamard, Pattern Matching, Block Matching, Pattern Detection.

I. INTRODUCTION

Many image processing and vision applications require filtering of images with a successive set of filter kernels; pattern classification, texture analysis, image de-noising, pattern detection are a few examples. In many such applications, however, applying a large set of filter kernels is prohibited due to time limitations. This limitation is even more severe when dealing with video data in which spatio-temporal filtering is required. Even when exploiting the convolution theorem and the Fast Fourier Transform (FFT) algorithm, the complexity remains high. A possible approach to increase efficiency is to design a set of specific kernels which are efficient to apply. Studies that took this course of action include the integral image [1], summed-area tables [2], and a generalized version of these called boxlets [3]. The main drawback of these approaches are that they allow only a limited set of filter kernels to be computed efficiently.

In this paper we aim to improve run-time and approach real-time performance for image filtering. Our work is motivated by a previous study [4], [5] in which the authors have shown that real-time pattern matching can be achieved using successive image filtering with a set of carefully chosen filter kernels.

The goal of this paper is to form a set of filter kernels that can be applied efficiently in various real-time applications. Towards this end, the suggested kernels should have the following desired characteristics:

- **Informative:** The kernels should be "informative" with respect to the relevant task.
- **Efficiency:** The kernels should be efficient to apply enabling real-time performance.
- **Variety:** The kernel set should consist of a large variety of kernels so that it can be used in various applications. It is advantageous to have a kernel set that forms a complete basis, enabling approximations of *any* desired kernel.

In this paper, we introduce a family of filter kernels such that successive convolution of an image with a set of such filters is highly efficient and requires only *2 operations per pixel* for each filter kernel, regardless of the size or dimension of the filter. Moreover, the memory required is at most 2 times the size of the original image. This family which we named *Gray-Code Kernels (GCK)* consists of a very large set of filter kernels, including the Walsh-Hadamard basis kernels, which can be used in a wide variety of applications. A specific application, demonstrating the method's efficiency, is presented in Section VI.

II. Previous Work

Image filtering is a very common operation in image processing, yet its computational complexity poses severe limitations in many applications. Numerous techniques have been proposed to expedite this operation (e.g. [6], [7], [8], [9], [10], [3]). These techniques can be categorized into three main classes of approaches: 1) Computational speed up of the filtering process independent of the kernel used. 2) Design of special families of kernels for which each kernel can be applied efficiently. 3) Design of special families of kernels for which a sequence of filters can be applied efficiently in a cascade manner.

The first class deals with reducing run time of the filtering operation which can be applied to any given filter kernel. The most common approach in this category is to exploit the convolution theorem and apply filtering in the frequency domain using the Fast Fourier Transform (FFT) [11]. In spite of the versatility of this approach, the scheme is efficient only for kernels with wide support, due to the overhead calculations of the FFT. Its performance in real-time applications is still inadequate (see e.g. [5]). Another approach in this class is to apply the filtering process in the Wavelets domain while ignoring high frequency coefficients and exploiting the energy compactization of the image in this domain [7]. Here too, in addition to the lossy results, the overhead of the Wavelet transform limits the profitability of this scheme in real-time applications.

The second class of approaches suggest fast image-filtering with filter kernels of specially defined families of kernels. These families of kernels have special characteristics that are exploited to reduce filtering complexity. Studies that took this course of action include the integral image [1], summed-area tables [2] and a generalized version of these called boxlets [3]. Another example are kernels that belong to known function spaces that are fast to apply [6]. These techniques are restricted to the specially defined families of kernels and do not generalize to allow filtering with any given kernel.

The third class includes techniques for fast filtering with a cascade of kernels. In such cases, efficiency of computation is achieved by exploiting relationships between the applied kernels. One approach in this direction is to find a reduced subspace in which the kernel set is approximately or exactly embedded. Filtering is performed with a small number of kernels that span this subspace. Then, due to linearity of the filtering process, the original kernel filtering results are computed as linear combinations of these few filtering results. The steerable filters technique [10], [12], deformable kernels [9], and the SVD filtering [8] follow this scheme.

This paper introduces a novel technique that belongs to the third class of approaches. A preliminary study was presented in [13]. Our work is motivated by a previous study [4], [5] where a fast filtering scheme for the Walsh-Hadamard (WH) kernel set was used for pattern detection. In this earlier study, the computational cost of convolving an image with each WH kernel is between 1 ops/pixel and up to at most $2logk$ ops/pixel for kernels of size $k \times k$. This performance is achieved by exploiting the recursive structure of the WH kernels. This previous approach, however, is constrained by several limitations:

- The method applies only to the Walsh-Hadamard Kernels.
- Filtering with each kernel requires $O(1)$-$O(d\ logk)$ operations per pixel (d being the kernel dimension and k its width).
- The fast filtering approach is limited to filtering in a fixed order of kernels (defined by the linear scanning of the leaves of the Walsh-Hadamard tree. See [5] for more details).
- Filtering is restricted to dyadic sized kernels.
- The method requires maintaining $d\ logk$ images in memory. This requirement might be prohibitive when dealing with 3D or higher-dimensional images.

In this paper we introduce the Gray-Code Kernels (GCK) and demonstrate their advantages:

- The GCK family of kernels enables filtering in O(1) operations per pixel per kernel, independent of the kernel size and dimension!
- The GCK family consists of a very large set of kernels.
- The GCK set includes non-dyadic kernels.
- The GCK method requires maintaining only 2 images in memory.

The GCK filters can be exploited in real-time applications, including, pattern detection, feature extraction, texture analysis, texture synthesis, and more.

III. The Gray-Code Kernels (GCK) - 1D Case

Consider first the 1D case where signal and kernels are 1 dimensional vectors. Denote by $V_s^{(k)}$ a set of 1D filter kernels expanded recursively from an initial seed vector $\mathbf{s}$ as follows:

Definition 3.1:

$$\begin{aligned} V_s^{(0)} &= \mathbf{s} \\ V_s^{(k)} &= \{[\mathbf{v}_s^{(k-1)} \;\; \alpha_k \mathbf{v}_s^{(k-1)}]\} \quad s.t. \quad \mathbf{v}_s^{(k-1)} \in V_s^{(k-1)}, \\ & \qquad\qquad\qquad\qquad\quad \alpha_k \in \{+1, -1\} \end{aligned}$$

where $\alpha_k \mathbf{v}$ indicates the multiplication of kernel $\mathbf{v}$ by the value α_k and $[\ldots]$ denotes concatenation.

The set of kernels and the recursive definition can be visualized as a binary tree of depth k. An example is shown in Figure 1 for $k = 3$. The nodes of the binary tree at level i represent the kernels of $V_s^{(i)}$. The leaves of the tree represent the 8 kernels of $V_s^{(3)}$. The branches are marked with the values of α used to create the kernels (where $+/-$ indicates $+1/-1$).

Denote $|s| = t$ the length of s. It is easily shown that $V_s^{(k)}$ is an orthogonal set of 2^k kernels of length $2^k t$. Furthermore, given an orthogonal set of seed vectors $\mathbf{s}_1, \ldots \mathbf{s}_n$, it can be shown that the union set $V_{s_1}^{(k)} \bigcup \ldots \bigcup V_{s_n}^{(k)}$ is orthogonal with $2^k n$ vectors of length $2^k t$. If $n = t$ the set forms a basis.

Figure 1 also demonstrates the fact that the values, $\alpha_1 \ldots \alpha_k$ along the tree branches, uniquely define a kernel in $V_s^{(k)}$.

Definition 3.2: The sequence $\boldsymbol{\alpha} = \alpha_1 \ldots \alpha_k, \quad \alpha_i \in \{+1,-1\}$ that uniquely defines a kernel $\mathbf{v} \in V_s^{(k)}$ is called the $\boldsymbol{\alpha}$**-index** of $\mathbf{v}$.

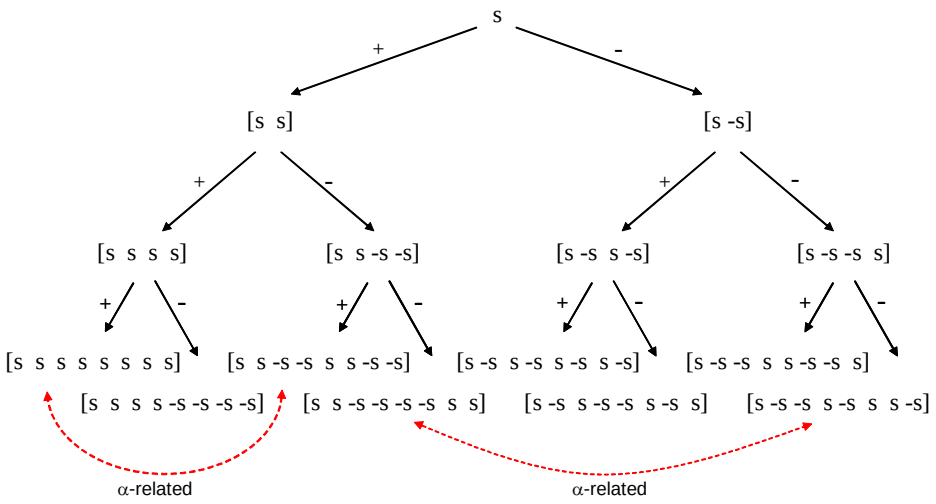

Fig. 1. The set of kernels and the recursive definition can be visualized as a binary tree. In this example the tree is of depth $k = 3$ and creates $2^3 = 8$ kernels of length 8. Arrows indicate pairs of kernels that are $\boldsymbol{\alpha}$-related.

We now define the notion of $\boldsymbol{\alpha}$**-relation** between two filter kernels:

Definition 3.3: Two kernels $\mathbf{v}_i, \mathbf{v}_j \in V_s^{(k)}$ are $\boldsymbol{\alpha}$**-related** iff the hamming distance of their α*-index* is one.

Without loss of generality, the α-indices of two α-related kernels are $(\alpha_1 \dots \alpha_{r-1}, +1, \dots \alpha_k)$ and $(\alpha_1 \dots \alpha_{r-1}, -1, \dots \alpha_k)$. We denote the corresponding kernels as $\mathbf{v}_+$ and $\mathbf{v}_-$ respectively. Since $\alpha_1 \dots \alpha_{r-1}$ uniquely define a kernel in $V_s^{(r-1)}$, two α-related kernels always share the same prefix vector of length $2^{r-1}t = \Delta$. The arrows of Figure 1 indicate examples of α-related kernels in the binary tree of depth $k = 3$. Note that not all possible pairs of kernels are α-related. Of special interest are sequences of kernels that are consecutively α-related.

Definition 3.4: An ordered set of kernels $\mathbf{v}_0 \dots \mathbf{v}_n \in V_s^{(k)}$ that are consecutively α-related form a sequence of **Gray Code Kernels (GCK)**. The sequence is called a **Gray Code Sequence (GCS)**.

The term Gray Code relates to the fact that the series of α-indices associated with a GCS forms a Gray Code [14], [15], [16]. The kernels at the leaves of the tree in Figure 3 in a left to right scan, are in fact consecutively α-related, and form a Gray Code Sequence. Note, however that this sequence is not unique and that there are many possible ways of reordering the kernels to form a Gray Code Sequence.

The main idea of this paper relies on the fact that two α-related kernels share a special relationship: Given two α-related kernels $\mathbf{v}_+, \mathbf{v}_- \in V_s^{(k)}$ their sum $\mathbf{v}_p$ and their difference $\mathbf{v}_m$ are defined as follows:

Definition 3.5:

$$\begin{aligned} \mathbf{v}_p &= \mathbf{v}_+ + \mathbf{v}_- \\ \mathbf{v}_m &= \mathbf{v}_+ - \mathbf{v}_- \end{aligned}$$

Theorem 3.6: Given two α-related kernels, $\mathbf{v}_+$, $\mathbf{v}_- \in V_s^{(k)}$ with a common prefix vector of length Δ, the following relation holds:

$$[\mathbf{0}_\Delta \quad \mathbf{v}_p] = [\mathbf{v}_m \quad \mathbf{0}_\Delta]$$

where $\mathbf{0}_\Delta$ denotes a vector with Δ zeros.

Proof is given in Appendix I. For example, consider the two α-related kernels from Figure 1 whose α-indices are $[+\,+\,+]$ and $[+\,-\,+]$ respectively :

$$\begin{aligned} \mathbf{v}_+ &= [s \;\; s \;\; s \;\; s \;\; s \;\; s \;\; s \;\; s] \\ \mathbf{v}_- &= [s \;\; s \;\; -s \;\; -s \;\; s \;\; s \;\; -s \;\; -s] \end{aligned}$$

They share a common prefix of length $\Delta = 2t$. Then

$$\begin{aligned} \mathbf{v}_p &= [\; 2s \;\; 2s \;\; \mathbf{0} \;\; \mathbf{0} \;\; 2s \;\; 2s \;\; \mathbf{0} \;\; \mathbf{0}] \\ \mathbf{v}_m &= [\; \mathbf{0} \;\; \mathbf{0} \;\; 2s \;\; 2s \;\; \mathbf{0} \;\; \mathbf{0} \;\; 2s \;\; 2s] \end{aligned}$$

and Theorem 3.6 holds with:

$$[\mathbf{0}_{2t} \quad \mathbf{v}_p] = [\mathbf{0} \;\; \mathbf{0} \;\; 2s \;\; 2s \;\; \mathbf{0} \;\; \mathbf{0} \;\; 2s \;\; 2s \;\; \mathbf{0} \;\; \mathbf{0}] = [\mathbf{v}_m \quad \mathbf{0}_{2t}]$$

For simplicity of explanation, we now expand $\mathbf{v} \in V_s^{(k)}$ to an infinite sequence such that $\mathbf{v}(i) = 0$ for $i < 0$ and for $i \geq 2^k t$. Using this convention, the relation $[\mathbf{0}_\Delta \quad \mathbf{v}_p] = [\mathbf{v}_m \quad \mathbf{0}_\Delta]$ can

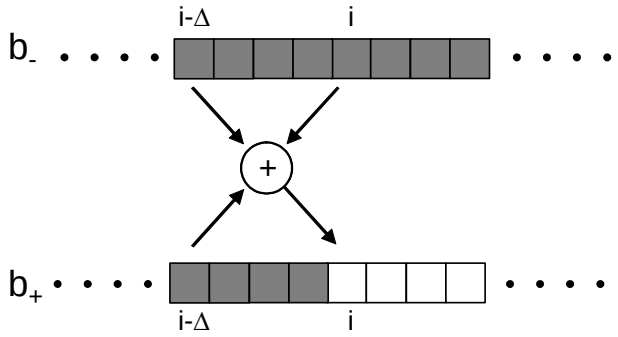

Fig. 2. Given $\mathbf{b}_-$, the convolution of a signal with the filter kernel $\mathbf{v}_-$, the convolution result $\mathbf{b}_+$ can be computed using 2 ops/pixel regardless of kernel size.

be rewritten in a new notation:

$$\mathbf{v}_p(i-\Delta) = \mathbf{v}_m(i)$$

With the new notation, Theorem 3.6 gives rise to the following Corollary:

Corollary 3.7:

$$\begin{aligned}\mathbf{v}_+(i) &= +\mathbf{v}_+(i-\Delta) + \mathbf{v}_-(i) + \mathbf{v}_-(i-\Delta)\\ \mathbf{v}_-(i) &= -\mathbf{v}_-(i-\Delta) + \mathbf{v}_+(i) - \mathbf{v}_+(i-\Delta)\end{aligned}$$

Corollary 3.7 is the core principle behind the efficient filtering scheme introduced in this paper.

Let $\mathbf{b}_+$ and $\mathbf{b}_-$ be the signals resulting from convolving a signal $\mathbf{x}$ with filter kernels $\mathbf{v}_+$ and $\mathbf{v}_-$ respectively:

$$\begin{aligned}\mathbf{b}_+(i) &= \textstyle\sum_j \mathbf{x}(j)\mathbf{v}_+(i-j)\\ \mathbf{b}_-(i) &= \textstyle\sum_j \mathbf{x}(j)\mathbf{v}_-(i-j)\end{aligned}$$

Then, by linearity of the convolution operation and Corollary 3.7 we have the following:

$$\begin{aligned}\mathbf{b}_+(i) &= +\mathbf{b}_+(i-\Delta) + \mathbf{b}_-(i) + \mathbf{b}_-(i-\Delta)\\ \mathbf{b}_-(i) &= -\mathbf{b}_-(i-\Delta) + \mathbf{b}_+(i) - \mathbf{b}_+(i-\Delta)\end{aligned}$$

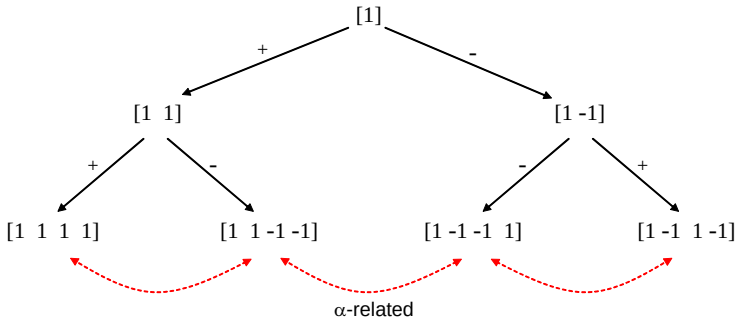

Fig. 3. Using initial vector $s = [1]$ and depth $k = 2$ a binary tree creates the Walsh-Hadamard basis set of order 4. Consecutive kernels are α-related, as shown by the arrows.

This forms the basis of an efficient scheme for convolving a signal with a set of GCK kernels. Given the result of convolving the signal with the filter kernel $\mathbf{v}_-$ ($\mathbf{v}_+$), convolving with the filter kernel $\mathbf{v}_+$ ($\mathbf{v}_-$) requires ***only*** 2 ***operations per pixel*** independent of the kernel size (Figure 2).

Example - The 1D Walsh-Hadamard Kernels

Following is a specific example for the above definitions and discussions.

Considering Definition 3.1, and setting the prefix string to $s = [1]$, we obtain that $V_s^{(k)}$ is the Walsh-Hadamard basis set of order 2^k. A binary tree can be designed such that its leaves are the Walsh-Hadamard kernels ordered in dyadically increasing sequency and they form a Gray Code sequence (i.e. are consecutively α-related). Such a tree and a discussion of its efficiency in pattern detection is described in [4], [5]. An example for $k = 2$ is shown in Figure 3 where every two consecutive kernels are α-related. For example, the first two kernels are:

$$\begin{aligned} \mathbf{v_0} &= [1 \quad 1 \quad 1 \quad 1] \\ \mathbf{v_1} &= [1 \quad 1 \quad -1 \quad -1] \end{aligned}$$

They share the prefix string $[1 \quad 1]$, thus $\Delta = 2$. Their sum and difference are respectively $\mathbf{v}_p = [2 \quad 2 \quad 0 \quad 0]$ and $\mathbf{v}_m = [0 \quad 0 \quad 2 \quad 2]$ and Theorem 3.6 holds with:

$$\mathbf{v}_p(i-2) = \mathbf{v}_m(i)$$

which yields:

$$\mathbf{v}_1(i) = -\mathbf{v}_1(i-2) + \mathbf{v}_0(i) - \mathbf{v}_0(i-2)$$

Thus, given the result of filtering an image with the first Walsh-Hadamard kernel, filtering with the second kernel requires only 2 operations (additions/subtractions) per pixel.

Subsequently, by ordering the Walsh-Hadamard kernels to form a Gray-Code Sequence, the windowed Walsh-Hadamard transform can be performed using only 2 operations per pixel per kernel regardless of signal and kernel size.

IV. Extension of GCK to Higher Dimensions

The previous sections can be generalized to higher dimensions. The most common use would be in 2D where input signals and filter kernels are 2D images. Thus, in this section we present only the 2-dimensional extension. However, the advantages of the approach are even more significant in 3D and higher dimensions. Extension to higher dimensions and proofs can be found in the Appendix.

In the previous sections, it was shown that successive filtering with α-related kernels can be applied efficiently using at most 2 operations per pixel. We now define the conditions under which higher dimensional filter kernels can be applied efficiently in a similar manner. We show that computation cost remains at 2 operations per pixel per kernel regardless of the dimension.

Definition 4.1: Two filter kernels $\mathbf{v}_{f_1}$ and $\mathbf{v}_{f_2}$ are considered **efficiently computable** if given an image filtered with one of the kernels, filtering the image with the second kernel is possible using two operations per pixel.

The following Lemma forms the basis of the Gray Code Kernel results for 2-dimensions:

Lemma 4.2: Assume $\mathbf{v}_{01}(i_1, i_2)$, $\mathbf{v}_{02}(i_1, i_2)$ are two filter kernels in 2 dimensions. $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$ are **efficiently computable** if both kernels are separable and can be factored into 1D kernels: $\mathbf{v}_{01} = \mathbf{v}_0 \times \mathbf{v}_1$ and $\mathbf{v}_{02} = \mathbf{v}_0 \times \mathbf{v}_2$ or $\mathbf{v}_{01} = \mathbf{v}_1 \times \mathbf{v}_0$ and $\mathbf{v}_{02} = \mathbf{v}_2 \times \mathbf{v}_0$, such that $\mathbf{v}_1$ and $\mathbf{v}_2$ are α-related.

The symbol "$\times$" denotes the outer product: $[\mathbf{v}_0 \times \mathbf{v}](i_1, i_2) = \mathbf{v}_0(i_1)\mathbf{v}(i_2)$. Proof is given in Appendix II.

As an example, consider the following two filter kernels:

$$\mathbf{v}_{01} = \begin{bmatrix} 10 & 10 & -10 & -10 \\ 5 & 5 & -5 & -5 \\ 5 & 5 & -5 & -5 \\ 10 & 10 & -10 & -10 \end{bmatrix} \quad \mathbf{v}_{02} = \begin{bmatrix} 10 & 10 & 10 & 10 \\ 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \\ 10 & 10 & 10 & 10 \end{bmatrix}$$

These kernels are separable:

$$\begin{matrix} \mathbf{v}_{01}: \\ \mathbf{v}_{02}: \end{matrix} \quad \mathbf{v}_0 = \begin{bmatrix} 10 \\ 5 \\ 5 \\ 10 \end{bmatrix} \times \begin{matrix} \mathbf{v}_1 = [1 & 1 & -1 & -1] \\ \mathbf{v}_2 = [1 & 1 & 1 & 1] \end{matrix}$$

If $\mathbf{s} = [1]$, the α-indices of $\mathbf{v}_1, \mathbf{v}_2$ are:

$$\boldsymbol{\alpha_1} = [+, \quad -]$$
$$\boldsymbol{\alpha_2} = [+, \quad +]$$

and therefore, by Definition 3.3, the kernels are α-related.

Given the filtering result $\mathbf{b}_1 = I * \mathbf{v}_{01}$ of the 2-dimensional image I with kernel $\mathbf{v}_{01}$, the filtering $\mathbf{b}_2 = I * \mathbf{v}_{02}$ can be calculated using 2 operations per pixel:

$$\mathbf{b}_2(i_1, i_2) = \mathbf{b}_2(i_1, i_2 - 2) + \mathbf{b}_1(i_1, i_2) + \mathbf{b}_1(i_1, i_2 - 2)$$

The operations in this example are along the 2nd dimension.

A. Separable Gray Code Kernels

Considering sets of 2-dimensional separable kernels, a set that spans a 2-dimensional image window is often required. Of special interest are separable kernels of the form $\mathbf{v} = \mathbf{v}_1 \times \mathbf{v}_2$ where $\mathbf{v}_1$ and $\mathbf{v}_2$ are each from a one dimensional set of Gray Code Kernels. The 2-dimensional version of $V_s^{(k)}$ is defined as:

$$V_{s_1,s_2}^{(k_1,k_2)} = \{\mathbf{v}_1 \times \mathbf{v}_2 \mid \mathbf{v}_i \in V_{s_i}^{(k_i)}\} \tag{1}$$

That is for $\mathbf{v} \in V_{s_1,s_2}^{(k_1,k_2)}$, $\mathbf{v}(i_1, i_2) = \mathbf{v}_1(i_1)\mathbf{v}_2(i_2)$. For example, $V_{[1],[1]}^{(2,3)}$ is the set of 2^5 2-dimensional kernels of size 4×8. The set $V_{[1],[1]}^{(2,2)}$ is shown in Figure 4 and forms the 4×4 Walsh-Hadamard kernels.

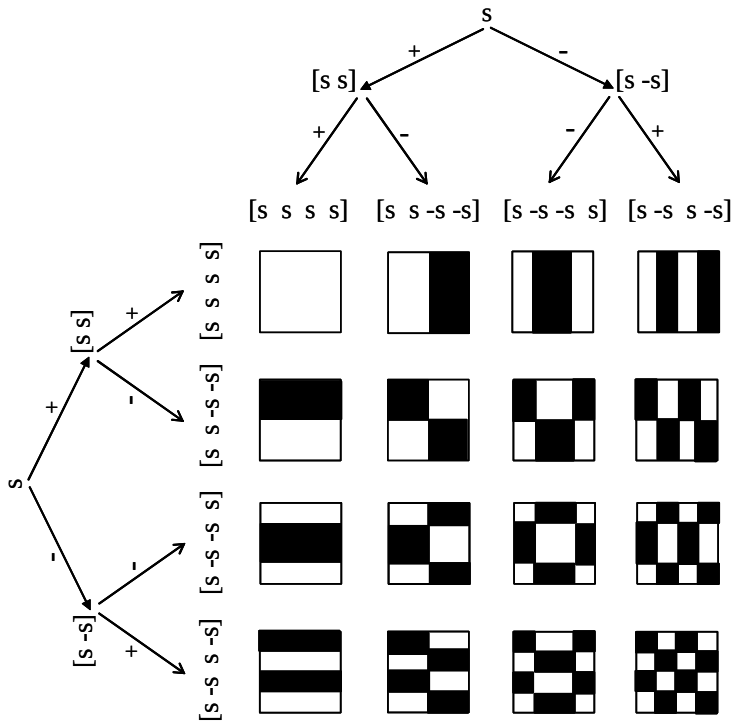

Fig. 4. The outer product of two sets of one-dimensional Gray Code Kernels forms the set of 2-dimensional kernels. In this case, the Walsh-Hadamard kernels of size 4×4 are obtained.

Since the 2-dimensional kernel $\mathbf{v}$ is separable, and can be defined by 2 one-dimensional kernels, the associated 2 α-indices uniquely define $\mathbf{v}$. Thus the following definition is consistent with the one dimensional case (Definition 3.2):

Definition 4.3: For $\mathbf{v} \in V_{s_1,s_2}^{(k_1,k_2)}$ such that $\mathbf{v} = \mathbf{v}_1 \times \mathbf{v}_2$, with associated α-indices $\boldsymbol{\alpha_1}$ and $\boldsymbol{\alpha_2}$, the sequence $\boldsymbol{\alpha} = [\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}]$ uniquely defines $\mathbf{v}$ and is called the $\boldsymbol{\alpha}$**-index** of $\mathbf{v}$.

The set of kernels can then be computed using a binary tree of depth $k_1 + k_2$ such that k_1 levels of the tree operate on the first dimension and k_2 on the second (see Figure 5).

Accordingly, the notion of $\boldsymbol{\alpha}$**-relation** between two 2-dimensional kernels is defined:

Definition 4.4: Two kernels $\mathbf{v}_i, \mathbf{v}_j \in V_{s_1,s_2}^{(k_1,k_2)}$ are $\boldsymbol{\alpha}$**-related** iff the hamming distance of their α-indices is one.

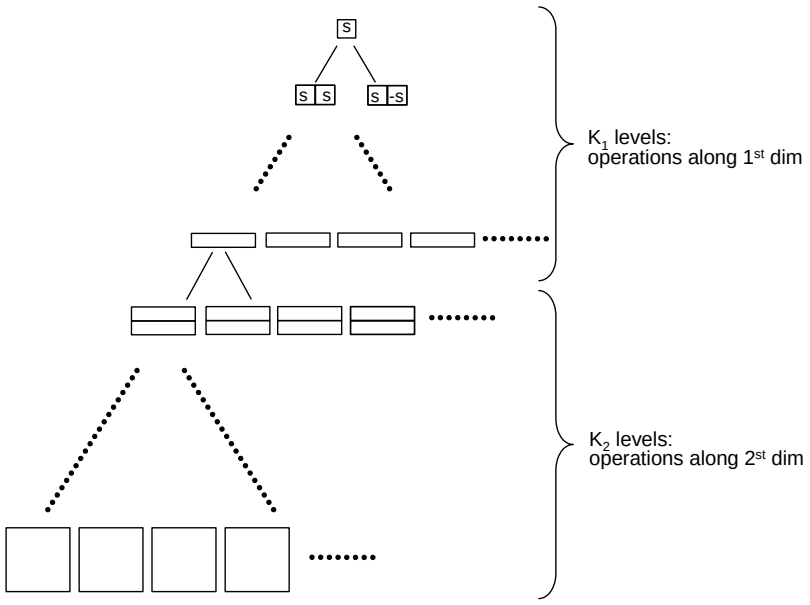

Fig. 5. The set of 2D kernels $V_{s_1,s_2}^{(k_1,k_2)}$ can be computed using a binary tree of depth $k_1 + k_2$ such that k_1 levels of the tree operate on the first dimension and k_2 on the second.

For example, in Figure 4, every pair of horizontally or vertically neighboring kernels are α-related.

The notion of a *Gray Code Sequence* of kernels can be extended to 2-dimensions:

Definition 4.5: An ordered set of 2-dimensional kernels, $\mathbf{v}_0 \ldots \mathbf{v}_n$ such that every consecutive pair are α-related, is called a **Gray Code Sequence** of kernels.

From Lemma 4.2 and Definition 4.5 we have the following corollary:

Corollary 4.6: Every two consecutive 2D kernels in a Gray Code Sequence are **efficiently computable**.

The GCK definitions extend naturally to higher dimensions. The d-dimensional version of $V_s^{(k)}$ is defined as:

$$V_{s_1,\ldots,s_d}^{(k_1,\ldots,k_d)} = \{\mathbf{v}_1 \times \ldots \times \mathbf{v}_d \mid \mathbf{v}_i \in V_{s_i}^{(k_i)}\} \tag{2}$$

and the d-dimensional kernel $\mathbf{v} \in V_{s_1,\ldots,s_d}^{(k_1,\ldots,k_d)}$ is defined as $\mathbf{v}(i_1,\ldots,i_d) = \mathbf{v}_1(i_1)\mathbf{v}_2(i_2)\ldots\mathbf{v}_d(i_d)$ where $\mathbf{v}_i$ are one dimensional Gray Code kernels.

The d α-indices associated with $\mathbf{v}_i$ uniquely define $\mathbf{v}$:

Definition 4.7: For $\mathbf{v} = \mathbf{v}_1 \times \ldots \times \mathbf{v}_d$, such that $\mathbf{v}_i \in V_{s_i}^{(k_i)}$ with associated α-index $\boldsymbol{\alpha_i}$, the sequence $\boldsymbol{\alpha}$ = $[\boldsymbol{\alpha_1}\ldots\boldsymbol{\alpha_d}]$ uniquely defines $\mathbf{v}$ and is the **$\boldsymbol{\alpha}$-index** of $\mathbf{v}$.

These kernels can be computed using a binary tree of depth $k_1 + \ldots + k_d$ similar to the tree of Figure 5.

If $s_1 = s_2 = \ldots = s_d = s$ and $k_1 = k_2 = \ldots = k_d = k$ then the set is denoted $V_s^{(k)^d}$ and contains d-dimensional square kernels.

Definitions and Lemmas for the separable d-dimensional GCKs are similar to the 2D case and are given in Appendix III.

Thus, 2 consecutive kernels in a GCS are efficiently computable, requiring only 2 ops/pixel, even in higher dimensions. Consequently, the use of GCK in higher dimensions is even more advantageous.

In summary, successive convolution of a signal with kernels of a GCS requires only 2 ops/pixel/kernel regardless of the kernel size or dimension. Furthermore, the successive convolution scheme requires maintaining in memory only the results of the previous convolution, thus only memory space the size of 2 times the original signal size is required throughout the process.

V. Sequencing the Gray-Code kernels

In previous sections we presented the GCK as a set of filter kernels. It was shown that successive filtering with α-related kernels of this set can be applied efficiently using 2 operations per pixel per kernel. However, the efficiency of using the GCK in a particular application is determined not only by the computational complexity of applying each kernel, but also by the total number of kernels taking part in the process. This, in turn, depends upon the order in which the kernels are applied. In this section we discuss the issue of ordering kernels into sequences of Gray Code Kernels.

Consider the set $V_s^{(k)^d}$ of d-dimensional separable kernels of size $2^k t$ (as defined in Section IV-A). The kernels in this set are uniquely represented by α-indices of length kd (Definition 4.3). A Gray Code Sequence is then equivalent to an ordering of these binary kd-vectors. As demonstrated above, many such sequences are possible. In this case, the question arises as to how many GCS are possible and how is the optimal GCS for a particular application chosen.

It is easily shown that the set $V_s^{(k)^d}$ is isomorphic to a kd-dimensional hypercube graph of 2^{kd} vertices; every kernel $\mathbf{v} \in V_s^{(k)^d}$ is associated with a vertex whose coordinates are equal to the α-index of $\mathbf{v}$. Edges in this hypercube connect vertices associated with α-related kernels. An example is shown in Figure 6a where the $2 \times 2 \times 2$ kernels of $V_s^{(1)^3}$ are represented by vertices of the 3-dimensional hypercube. The vertices are marked by the α-index associated with the kernel ($+, -$ are represented here by $0, 1$ respectively). Pairs of vertices connected by an edge represent α-related kernels.

A Gray Code Sequence is a sequence of α-related kernels thus it is isomorphic to a path in the hypercube graph. A GCS containing all the kernels of $V_s^{(k)^d}$ is isomorphic to a Hamiltonian path in the kd-dimensional hypercube [14], [15]. For example, in Figure 6b, a Hamiltonian path is marked on the graph and represents a complete sequence of Gray Code Kernels. Given this relationship between Gray Code sequencing and Hamiltonian paths in graphs, it is evident that the number of possible Gray Code Sequences containing all kernels of the set $V_s^{(k)^d}$ is equal to the number of Hamiltonian paths in a kd-dimensional hypercube: $2, 8, 144, 91392, \ldots$ for $kd = 1, 2, 3, 4, \ldots$ [17], [15].

Hamiltonian paths are isomorphic to linear GCS in which a filter kernel appears only once. However, other variations may be considered as well, including allowing multiple appearances of a kernel (see Backtracking in [18]) and sequences that are isomorphic to a spanning tree on the hypercubic graph (see Increasing Memory Allowance in [18]) In this paper we concentrate on linear GCS.

In typical situations where filtering with GCKs is used, only a subset of filters from the filter set $V_s^{(k)^d}$ are required to complete the process. In such cases, the *Optimal* Gray Code Sequence should be chosen from amongst the numerous possible GCS. To do so, a priority value is assigned to each kernel, representing its contribution in achieving the goal of the process. The priority value strongly depends on the application and possibly the input data. For example, using the projected values as features in a classification process, the priority value may reflect

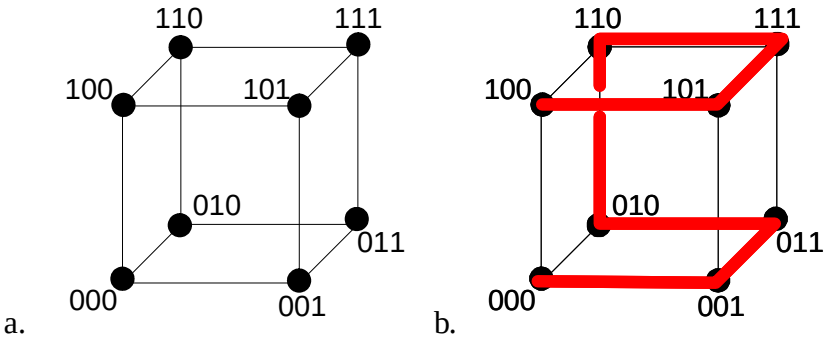

Fig. 6. a. The set $V_s^{(1)^3}$ represented by a 3-dimensional hypercubic graph. The vertices are marked by the α-index associated with the kernel (for simplicity, 0 and 1 replace + and - respectively). Every pair of α-related kernels share an edge in the hypercubic graph. b. A Hamiltonian Path is marked on the graph, representing a GCS.

the discrimination power of each kernel, i.e. the ability of classifying examples based on the projection values of the specific kernel. A detailed example is given in Section VI.

Assigning a priority value to each kernel is analogous to associating a weight with every node in the isomorphic hypercube graph. Given the priority values, the optimal Gray Code Sequence and accordingly the chosen path in the hypercube graph can be determined. In this paper we determined the optimal GCS of a given length as that which maximizes the accumulated priority values. This definition of optimality is appropriate for applications with limited run time that allow only a fixed predefined number of filter kernels to be applied. Note that the order of the kernels within the sequence is insignificant. This is analogous to finding the maximally weighted path of a given length in a hypercube graph. It can be shown that this problem is a special case of the Travelling Salesman's Subtour Problem [19] and the Orienteering Problem (OP) [20] shown to be NP-hard. Approximation algorithms for this problem have been suggested [21], [22].

In order to demonstrate the advantage of the optimal GCS over other orderings of the Gray Code Kernels, we compared three types of Gray Code sequencing:

- **Greedy** - This algorithm orders the kernels in decreasing priority value. This order maximizes the accumulated priority value for any given length of sequence, however, the sequence produced is not necessarily a Gray Code Sequence since consecutive kernels are not necessarily α-related. In terms of computation, filtering with each $k \times k$ kernel of the sequence, naively requires $2logk$ ops/pixel (e.g. by computing projection onto each kernel

along a branch of the Walsh-Hadamard tree as explained in [5]).

- **Sequency** - This algorithm creates a GCS of Walsh-Hadamard kernels ordered with increasing sequency (the number of sign changes along each dimension of the kernel - analogous to frequency). For the 2D Walsh-Hadamard kernels, the 'snake' order as depicted in Figure 7 is used. In Section IV-A, we have shown that neighboring kernels in the 2D Walsh-Hadamard array are α-related. Thus the Walsh-Hadamard kernels, ordered in this manner form a Gray Code Sequence and the computation cost for each kernel reduces to 2 ops/pixel. The Sequency order is known to perform well on natural images due to energy compactization in the low order sequencies [23], [24], [5]. Note that, in contrast to the Greedy ordering, this computation scheme does not depend on the priority values of the kernels.
- **Optimal** - Given a priority value associated with each kernel, this algorithm returns a GCS of a given length with maximum accumulated priority values. This, in theory is an NP-hard problem, however for short sequences (length $n \leq 10$) an exhaustive search can be implemented in reasonable time. For longer sequences ($n > 10$), we implemented a pseudo-optimal GCS which is created by concatenating several (n/10) optimal GCS of short length with an additional constraint that requires the last kernel in each short sequence to be α-related to the first kernel in the following short sequence. Recurrent kernels are allowed, thus kernels may appear more than once in the Optimal GCS, however their priority value is accumulated only once.

VI. Experimental Results

The most attractive property of the GCK framework is that it enables filtering with each kernel using only 2 ops/pixel. However, this can only be achieved if the kernels are used in an order which forms one of the many possible Gray-Code Sequences. This section tests the implications of this requirement on the GCK efficiency in a specific application: Pattern-Matching, where a given pattern is sought in an image. We follow the scheme suggested in [4], [5] where a framework for real-time pattern matching was introduced. In this section we compare performance of filtering using the 2D Walsh-Hadamard kernels as projection vectors when ordered as GCS and when ordered otherwise. We also compare performance with another known fast filtering scheme, namely, the Integral Image kernels as used in [1].

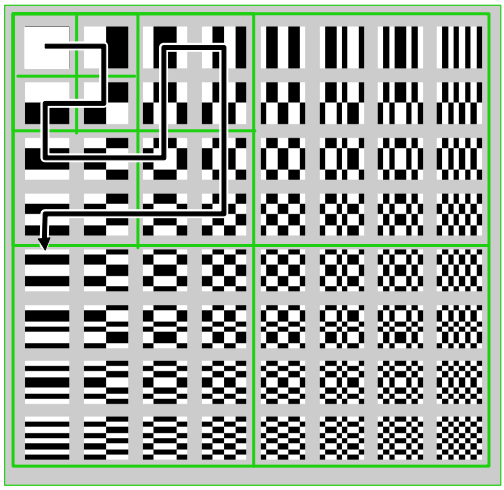

Fig. 7. An array of Walsh-Hadamard kernels of order $n = 8$ ordered with increasing sequency in each column and row. White represents the value 1, and black represents the value -1. A 'snake' ordering of these kernels is shown by the overlayed arrows. This ordering of kernels forms a GCS of Walsh-Hadamard kernels ordered with pseudo-increasing sequency.

A. Projection-Based Pattern Matching

Finding a given pattern in an image is typically performed by scanning the entire image, and evaluating the similarity between the pattern and a local 2D window about each pixel. In our experiments, we assume the most common measure of similarity - the Euclidean distance.

Assume a $k \times k$ pattern $\mathbf{p}$ is to be sought within a given image. Pattern $\mathbf{p}$ is matched against a similarly sized window $\mathbf{w}$ at every location in the image. Referring to the pattern $\mathbf{p}$ and the window $\mathbf{w}$ as vectors in $\Re^{k^2}$, the Euclidean distance between them is given as:

$$d_E^2(\mathbf{p}, \mathbf{w}) = \|\mathbf{p} - \mathbf{w}\|^2 \tag{3}$$

The smaller the distance value, the more similar are $\mathbf{w}$ and $\mathbf{p}$. If the distance is found to be below a given threshold, then it is concluded that window $\mathbf{w}$ is similar to the pattern $\mathbf{p}$. Now, assume that $\mathbf{p}$ and $\mathbf{w}$ are not given, but only the values of their projection ${\mathbf{v}_1}^T\mathbf{p}$ and $\mathbf{v}_1^T\mathbf{w}$ onto a particular projection vector $\mathbf{v}_1$, where $\|\mathbf{v}_1\| = 1$. Since the Euclidean distance is a norm, it follows from the Cauchy-Schwartz inequality, that a lower bound on the actual Euclidean distance can be inferred from the projection values [5]:

$$d_E^2(\mathbf{p}, \mathbf{w}) \geq d_E^2(\mathbf{v}_1^T\mathbf{p}, \mathbf{v}_1^T\mathbf{w}) \tag{4}$$

If an additional projection vector $\mathbf{v}_2$ is given along with the projection values: $\mathbf{v}_2^T\mathbf{p}$ and $\mathbf{v}_2^T\mathbf{w}$, it is possible to tighten the lower bound on the distance. Define the distance vector $\mathbf{d} = \mathbf{w} - \mathbf{p}$, and assume that the projection values of $\mathbf{d}$ onto r orthonormal projection vectors are given:

$$M^T\mathbf{d} = \mathbf{b}$$

where $M = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots \mathbf{v}_r]$ is a matrix composed of the r orthonormal projection vectors, and $\mathbf{b} = [b_1\ b_2\ \cdots b_r]$ is a vector of projection values $b_i = \mathbf{v}_i^T\mathbf{d} = \mathbf{v}_i^T\mathbf{w} - \mathbf{v}_i^T\mathbf{p}$. It is straightforward to verify that the lower-bound on the distance is:

$$d_E^2(\mathbf{p}, \mathbf{w}) = \mathbf{d}^T\mathbf{d} \geq \mathbf{b}^T\mathbf{b}$$

A similar expression can be obtained when $\mathbf{v}_i$ are not orthonormal [5].

Note, that as the number of projection vectors increases, the lower bound on the distance $d_E(\mathbf{p}, \mathbf{w})$ becomes tighter. In the extreme case where $r = k^2$ and the projection vectors are linearly independent, the lower bound reaches the actual Euclidean distance.

The above implies that a lower bound on the distance between a window and the pattern can be estimated from the projections. Thus, complexity and run time of the pattern matching process can be significantly reduced by rejecting windows with lower bounds exceeding a given threshold value. This can be utilized within the following process:

1) The sought pattern $\mathbf{p}$ is projected onto a set of n normalized projection vectors $\{\mathbf{v}_i\}$, resulting in n values: $\hat{p}^i = \mathbf{v}_i^T\mathbf{p}$, for $i = 1 \ldots n$.
2) All signal windows $\{\mathbf{w}_j\}$ are projected onto the first projection vector $\mathbf{v}_1$: $\hat{w}_j^1 = \mathbf{v}_1^T\mathbf{w}_j$
3) This projection sets a lower bound on the true distance between each window $\mathbf{w}_j$ and the pattern: $LB_j^1 = (\hat{w}_j^1 - \hat{p}^1)^2$. According to the lower bound values, any window j whose LB_j^1 value is greater than the given threshold can be rejected.
4) The windows of the image that have not been rejected, are projected onto the second projection vector $\mathbf{v}_2$: $\hat{w}_j^2 = \mathbf{v}_2^T\mathbf{w}_j$. This produces updated lower bounds: $LB_j^2 = LB_j^1 + (\hat{w}_j^2 - \hat{p}^2)^2$.
5) Steps 3 and 4 are repeated for the subsequent projection vector.
6) The process terminates after all n kernels have been processed or until the number of non-rejected image windows reaches a predefined percentage.

Former experiments (e.g. see [4], [5]) showed that for reasonable threshold values, almost all non-matching windows in the image are rejected when the lower bound (LB_j) reaches 80% of the true Euclidean distance for each window.

The main advantage using this framework is that by carefully choosing the appropriate projection vectors, this lower bound can be reached using a small number of projections. Note, that projecting all image windows onto a projection vector can be implemented using convolution. Thus an efficient scheme for convolving an image with a sequence of projection vectors is advantageous within this pattern matching framework. In this section we show the advantage of using the Gray Code Kernels as projection vectors for pattern matching. For detailed description of the rejection framework the reader is referred to [4], [5].

B. Experimental Results I - General Pattern Case

The efficiency of the pattern matching process is measured by the total number of operations required to find pattern appearances in the image. Using the above described rejection framework, the total number of operations is dependent on two factors: the number of projection vectors required to reject the non-matching windows and the cost of computing the projections onto each of the projection vectors. In turn, the number of required projection vectors is dependent on the order of the vectors used in the process. Thus, fewer vectors would be needed if the kernels with strong rejection power are applied early in the process.

In the first experiment we tested the rejection power of different orders of kernels. The experimental setting assumes an unknown pattern and an unknown window, both sampled from natural scenes. This setting represents applications where image windows are given online and no specific knowledge on the patterns is known apriori, (e.g. match-based texture synthesis [25], or video coding [26]). Thus, over 3000 pattern-window pairs of size 8×8 were randomly chosen from a collection of images and the Euclidean distance between them were computed. Each 8×8 Walsh-Hadamard kernel $\mathbf{v}_i$, was assigned a priority value which indicates the percentage of the distance between the pattern-window pairs captured by the given kernel:

$$\delta_i = E_j\{\frac{(\mathbf{v}_i^T(\mathbf{p}_j - \mathbf{w}_j))^2}{(\mathbf{p}_j - \mathbf{w}_j)^2}\} \tag{5}$$

Here the expectation is calculated over all randomly chosen pattern-window pairs. Given the priority values, the Walsh-Hadamard kernels were ordered according to the three methods described in Section V: Greedy, Sequency, and Optimal. We compared the rejection power of these

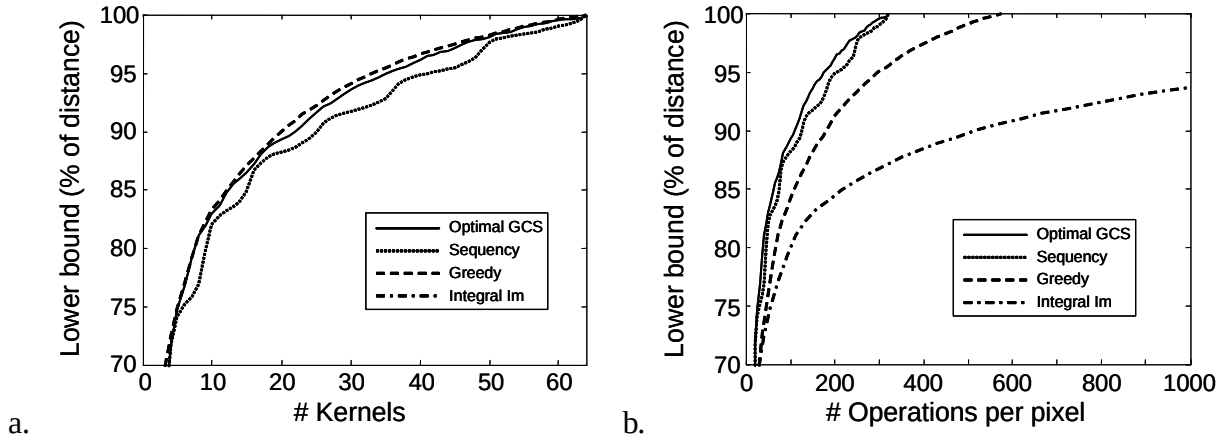

Fig. 8. Filtering by projection using WH kernels in 3 different Sequences and using Integral Image Kernels. The three WH orders include: Greedy, Sequency and Optimal GCS. a. The lower bound as a function of the number of kernel projections. b. The lower bound as a function of the number of operations per pixel required to compute the lower bound. The lower bound is given as the average percentage of the actual distance between pattern and window. All values are the average over 3000 pattern-window pairs sampled randomly from natural scenes.

three sequences by evaluating the lower bound calculated from the kernels in the sequence. The tighter the lower bound, the greater the rejection power of the sequence and accordingly the expected performance in a pattern matching application.

Figure 8 compares filtering performance and run times of the three Sequences of Walsh-Hadamrd kernels. Figure 8a shows the lower bound (given as the percentage of the actual distance between pattern and window) as a function of the number of kernel projections. Values are the average over the 3000 pattern-window pairs. The Greedy order of Walsh-Hadamard kernels creates tight lower bounds with fewer kernels than the Optimal and Sequency GCS. These results, however, do not exhibit the run times required to obtain the lower bounds. Figure 8b shows the lower bound (given as the average percentage of the actual distance between pattern and window) as a function of the number of operations per pixel required to compute the lower bound. The Optimal GCS outperforms the Sequency and Greedy sequences. The Sequency GCS performs relatively well as expected for pattern-window pairs chosen randomly from natural images.

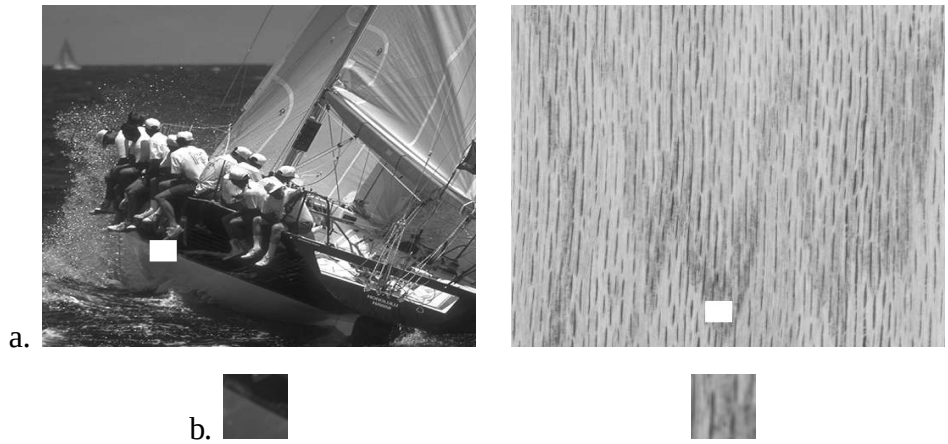

Fig. 9. The natural and texture images (a) and patterns (b) that were used in the experiments. Each of the 2 patterns was sought in each of the 2 images. The origins of the patterns within the images are marked with a white square. The images are 512×512 and the patterns are 32×32.

C. Experimental Results II - Specific Pattern Case

In this experiment , the performance of the three computation schemes (Greedey, Sequency, and Optimal) were compared over four pattern-image scenarios. Two images of size 512×512 (figure 9.a) were chosen, representing a 'natural' and a 'texture' image. A 32×32 window (figure 9.b) was chosen randomly from each image and these served as the patterns, denoted as 'natural pattern' and 'texture pattern'. Each case of pattern and image pair was tested both with and without the DC kernel (to eliminate illumination effects). Our main interest is the comparison of the total number of operations required per pixel by each of the computation schemes. Comparison was based on the ability to reach the 80% lower bound on the average Euclidean distance between the pattern and image windows as described in Section VI-A.

Figure 10 presents the number of kernels required by each computation scheme in order to reach the 80% goal. This number is dictated by the order of the kernels. As expected, the Greedy sequence required the least number of kernels and the Sequency order required the most. Since the Optimal Sequence imposes constraints on the Greedy order, it requires a few more kernels than the Greedy sequence. For the case of natural pattern - natural image, the Sequency order of kernels performs equally well as the optimal GCS as expected. This is not true for the other three cases.

Figure 11 shows the total number of ops/pixel required by the 3 computation schemes for

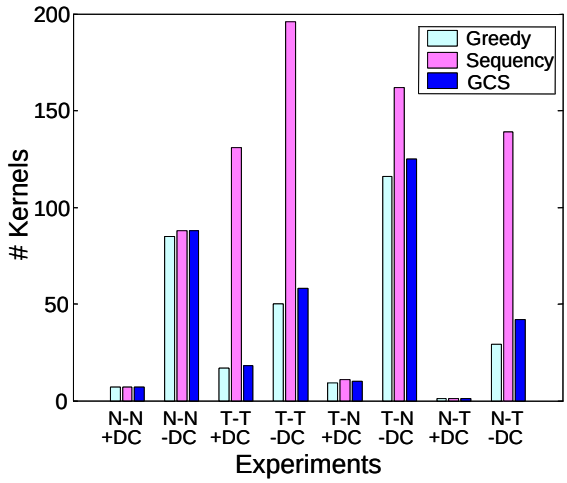

Fig. 10. The number of kernels dictated by the order of each computation scheme. Each experiment is denoted by the pattern-image pair (N=Natural, T=Texture) and whether the DC was included (+DC) or not (-DC).

the 4 pattern-image cases with and without the DC value. It can be seen that the Optimal GCS scheme always required fewer ops/pixel than the other two orderings even though the actual number of kernels used is greater. The Greedy scheme always required more ops/pixel even though it used the fewest number of kernels.

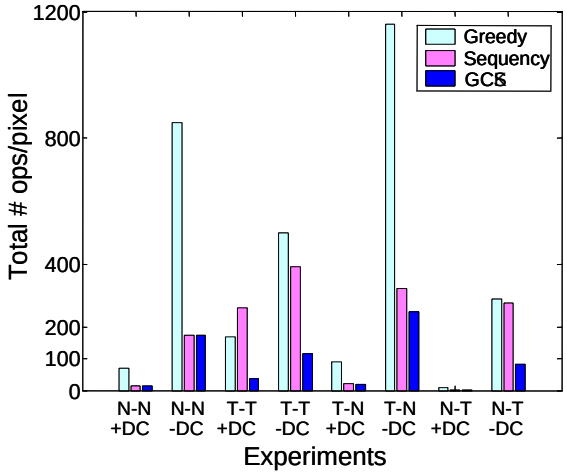

Fig. 11. The total number of ops/pixel required by each of the computation schemes. Each experiment is denoted as in Figure 10.

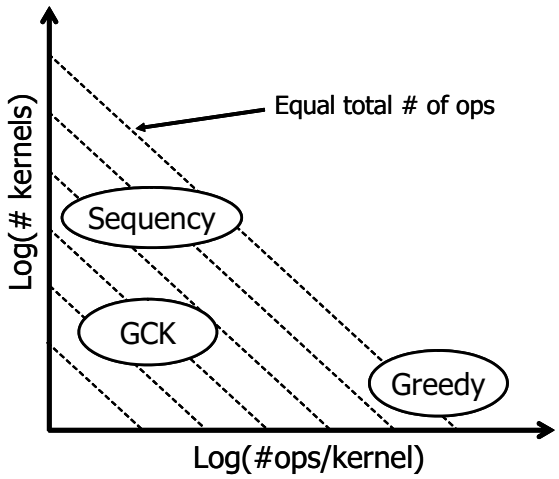

Fig. 12. A comparison between the three computation schemes. Each scheme is positioned in the plot according to the number of kernels required and the number of ops/kernel. The dashed lines represent lines of equal Total number of Operations.

The comparison of the three schemes is graphically displayed in Figure 12 where each of the three schemes is positioned in a plot of the number of kernels required vs. the number of ops/kernel (log-log scale). The dashed lines represent lines of equal total number of operations. Thus, although the Greedy scheme requires fewer kernels, the cost per kernel causes this scheme to be expensive in terms of total number of operations. The Optimal GCS scheme requires more kernels but since the computation cost per kernel is very low, the total number of operations required is minimal.

Details of the experimental procedure, and analysis can be found in [18].

D. Experimental Results III - Comparison with the Integral Image Kernels

The idea of choosing projection kernels that are fast to apply was also suggested in [1], [27] in the context of classification. Although similar in spirit, we emphasize the distinction between our approach and that of Viola et. al. [1]. Whereas Viola et. al. perform efficient classification using rapidly computed projection kernels, our suggested approach performs efficient filtering that can be exploited in block matching in general. Thus, classification approaches are impractical when patterns are given online e.g. in the case of texture synthesis and video coding (see GCK implementation for video coding in [26]). Nevertheless, the use of the Integral Image Kernels themselves, can be considered in the rejection based Pattern Matching application. Kernels of

this set are efficient to compute when they are of low sequency (computational complexity of these kernels increases exponentially with sequency). Moreover, the non-orthogonality of the kernels adds additional computational costs due to the redundancy in feature content captured by the kernels. This tendency is shown in the following experimental results.

We compared the performance of the GCK filtering of Walsh-Hadamard kernels with that of the Integral Image kernels used in [1]. A priority value as defined in Equation 5 was assigned to a set of Integral Image kernels. The kernels of the set that are the most efficient to apply are the first order kernels which sum a single rectangular region within the window. Assuming the integral image is given, these kernels require 3 ops per pixel to apply. We consider the set of all possible first order integral image kernels of size 8×8. These kernels were ordered in decreasing priority order. Since these kernels are not orthogonal, we considered the contribution of each kernel to the lower bound (this is achieved using the Grahm-Schmidt orthogonalization method in which the kernels are iteratively projected onto the subspace spanned by the already sorted kernels. See Appendix 3 in [5]). Filtering performance with these filters were compared with those obtained by the three sequences of Walsh-Hadamard kernels described above. Results for the Integral image kernels are also shown in Figure 8. The lower bound as a function of the number of Integral Image kernels applied is identical to that of the Greedy sequence (the lines overlap perfectly in Figure 8a). This is due to the fact that both the Walsh-Hadamard kernels and the first order Integral Image kernels span the 8×8 pattern-window space and a non-restricted ordereing of both kernel sets produce the tightest possible lower bounds for any given number of kernels. In terms of run-time, however, all three orders of Walsh-Hadamard kernels outperform the Integral Image kernels in the number of operations per pixel as shown in Figure 8b. This is due to the fact that computation of the lower bound for the non-orthogonal Integral Image kernels requires significantly more computation than for the orthogonal Walsh-Hadamard kernels (see Appendix 3 in [5]).

VII. Conclusions

In this paper we introduced a family of filter kernels called the Gray Code Kernels (GCK). A special relationship between pairs of such kernels allow filtering of images with a cascade of such kernels to be performed very fast. The GCK framework is a highly-efficient computational scheme mainly due to the following advantages:

- The ability of filtering an image using only 2 operations per pixel per kernel.
- The computation cost is independent of the kernel size and kernel dimension.
- The computations are performed using only integer additions and subtractions (if the seed s is integer).
- Only a single image (the filtering results with a preceding kernel) needs to be maintained in memory in addition to that currently being computed.
- A wide variety of kernels can be used within this framework.
- The kernels can be computed in a variety of different orders.
- The kernel set forms an orthogonal basis.

We note the following limitations of the GCK:

- The filtering with each kernel depends on the filtering result of a preceding kernel. Thus when a single kernel computation is required, the advantages of this framework can not be exploited. This also poses a limitation on the order in which the kernels can be computed.
- The framework offers efficient filtering for a group of image windows. Computing the projection of a single image window might require more than 2 ops/pixel.

We note that it is possible to extend the family of GCKs even further by allowing the α-index to be of any integer value (rather than only $\{+1, -1\}$). This, however, incurs an additional 1 ops/pixel. Details can be found in [18].

The unique properties of the GCK framework makes it an attractive choice for many applications requiring a cascade of kernel computation scheme such as feature extraction, block matching for motion detection (e.g. [26]), texture analysis and synthesis, classification and more.

APPENDIX I
PROOF OF THEOREM 3.6

We prove the following Theorem.

Theorem 3.6 Given two α-related kernels, $\mathbf{v}_+$, $\mathbf{v}_- \in V_s^{(k)}$ with a common prefix vector of length $\Delta = 2^{r-1}t$, where $t = |\mathbf{s}|$, the following relation holds:

$$[\mathbf{0}_\Delta \quad \mathbf{v}_p] = [\mathbf{v}_m \quad \mathbf{0}_\Delta]$$

where $\mathbf{0}_\Delta$ denotes a vector with Δ zeros.

Proof:

Denote by $\mathbf{v}^{(l)}$ the prefix of vector $\mathbf{v}$ of length $2^l t$. Since $\mathbf{v}_+$ and $\mathbf{v}_-$ are α-related there exists an entry r, $1 \leq r \leq k$, for which their two α-indices differ. We prove that the following holds for all l, $r \leq l \leq k$:

$$[\mathbf{0}_\Delta \quad \mathbf{v}_p^{(l)}] = [\mathbf{v}_m^{(l)} \quad \mathbf{0}_\Delta]$$

Proof is by induction: for $l = r$ we have by definitions 3.1 and 3.5 that

$$\begin{aligned} \mathbf{v}_+^{(r)} &= [\mathbf{v}_+^{(r-1)} \quad \mathbf{v}_+^{(r-1)}] \\ \mathbf{v}_-^{(r)} &= [\mathbf{v}_-^{(r-1)} \quad -\mathbf{v}_-^{(r-1)}] = [\mathbf{v}_+^{(r-1)} \quad -\mathbf{v}_+^{(r-1)}] \end{aligned}$$

thus

$$\begin{aligned} \mathbf{v}_p^{(r)} &= \mathbf{v}_+^{(r)} + \mathbf{v}_-^{(r)} = [2\mathbf{v}_+^{(r-1)} \quad \mathbf{0}_\Delta] \\ \mathbf{v}_m^{(r)} &= \mathbf{v}_+^{(r)} - \mathbf{v}_-^{(r)} = [\mathbf{0}_\Delta \quad 2\mathbf{v}_+^{(r-1)}] \end{aligned}$$

and we have

$$[\mathbf{0}_\Delta \quad \mathbf{v}_p^{(r)}] = [\mathbf{0}_\Delta \quad 2\mathbf{v}_+^{(r-1)} \quad \mathbf{0}_\Delta] = [\mathbf{v}_m^{(r)} \quad \mathbf{0}_\Delta]$$

By induction, we assume true for $l - 1 \geq r$ and prove for l (note that α_l is identical for both α-indices):

$$\begin{aligned} [\mathbf{0}_\Delta \quad \mathbf{v}_p^{(l)}] &= [\mathbf{0}_\Delta \quad [\mathbf{v}_+^{(l)} + \mathbf{v}_-^{(l)}]] \\ &= [\mathbf{0}_\Delta \quad [[\mathbf{v}_+^{(l-1)} \quad \alpha_l \mathbf{v}_+^{(l-1)}] + [\mathbf{v}_-^{(l-1)} \quad \alpha_l \mathbf{v}_-^{(l-1)}]]] \\ &= [\mathbf{0}_\Delta \quad [\mathbf{v}_p^{(l-1)} \quad \alpha_l \mathbf{v}_p^{(l-1)}]] \\ &= [[\mathbf{0}_\Delta \quad \mathbf{v}_p^{(l-1)}] \quad \alpha_l \mathbf{v}_p^{(l-1)}] \end{aligned}$$

from the induction assumption:

$$\begin{aligned} &= [[\mathbf{v}_m^{(l-1)} \ \mathbf{0}_\Delta] \ \alpha_l \mathbf{v}_p^{(l-1)}] \\ &= [\mathbf{v}_m^{(l-1)} \ [\mathbf{0}_\Delta \ \alpha_l \mathbf{v}_p^{(l-1)}]] \\ &= [\mathbf{v}_m^{(l-1)} \ \alpha_l [\mathbf{0}_\Delta \ \mathbf{v}_p^{(l-1)}]] \end{aligned}$$

and again from the induction assumption:

$$\begin{aligned} &= [\mathbf{v}_m^{(l-1)} \ \alpha_l [\mathbf{v}_m^{(l-1)} \ \mathbf{0}_\Delta]] \\ &= [[\mathbf{v}_m^{(l-1)} \ \alpha_l \mathbf{v}_m^{(l-1)}] \ \mathbf{0}_\Delta] \\ &= [\mathbf{v}_m^{(l)} \ \mathbf{0}_\Delta] \end{aligned}$$

APPENDIX II

EFFICIENTLY COMPUTABLE FILTER KERNELS IN D-DIMENSIONS

We now prove the efficiency of computation with d-dimensional GCKs. To simplify the notations we define a short notation for a sequence of indices. A complete d-dimensional kernel is defined as:

$$\mathbf{v}(\mathbf{i}) = \mathbf{v}(i_1, i_2, \cdots, i_d)$$

and a $(d-1)$-dimensional kernel, lacking the dimension m as:

$$\mathbf{v}(\{\sim i_m\}) = \mathbf{v}(i_1, \cdots, i_{m-1}, i_{m+1}, \cdots, i_d)$$

Recall that two kernels can be regarded as *efficiently computable* if their computation cost is 2 ops/pixel.

Lemma 2.1: Assume $\mathbf{v}_{01}(i_1, i_2)$, $\mathbf{v}_{02}(i_1, i_2)$ are two filter kernels in d dimensions. $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$ are **efficiently computable** if both kernels can be factored into α-related kernels: $\mathbf{v}_{01} = \mathbf{v}_0 \times \mathbf{v}_1$ and $\mathbf{v}_{02} = \mathbf{v}_0 \times \mathbf{v}_2$, where $\mathbf{v}_0$ is $(d$–$1)$-dimensional, $\mathbf{v}_1$ and $\mathbf{v}_2$ are 1-dimensional α-related kernels and $\times$ denotes the outer product along the m-th dimension, i.e. $[\mathbf{v}_0 \times \mathbf{v}](\mathbf{i}) = \mathbf{v}_0(\{\sim i_m\})\mathbf{v}(i_m)$.

Proof:

Assume $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$ are two such filters. Since $\mathbf{v}_1, \mathbf{v}_2$ are α-related, they share a common prefix vector of length $\Delta = 2^{r-1}t$ and Corollary 3.7 holds. Thus w.l.o.g. we assume:

$$\mathbf{v}_2(i) = +\mathbf{v}_2(i - \Delta) + \mathbf{v}_1(i) + \mathbf{v}_1(i - \Delta)$$

Then we have

$$\begin{aligned}
\mathbf{v}_{02}(\mathbf{i}) &= \mathbf{v}_0(\{\sim i_m\})\mathbf{v}_2(i_m) \qquad (6)\\
&= \mathbf{v}_0(\{\sim i_m\})\Big(\mathbf{v}_2(i_m-\Delta)+\mathbf{v}_1(i_m)+\mathbf{v}_1(i_m-\Delta)\Big)\\
&= \mathbf{v}_0(\{\sim i_m\})\mathbf{v}_2(i_m-\Delta)+\mathbf{v}_0(\{\sim i_m\})\mathbf{v}_1(i_m)+\mathbf{v}_0(\{\sim i_m\})\mathbf{v}_1(i_m-\Delta)\\
&= \mathbf{v}_{02}(i_1,\ldots,(i_m-\Delta),\ldots,i_d)+\mathbf{v}_{01}(\mathbf{i})+\mathbf{v}_{01}(i_1,\ldots,(i_m-\Delta),\ldots,i_d)
\end{aligned}$$

Given a d-dimensional signal $\mathbf{s}$, denote by $\mathbf{b}_1, \mathbf{b}_2$ the d-dimensional convolution of $\mathbf{s}$ with $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$ respectively. From Equation 6 and linearity of the convolution we have:

$$\mathbf{b}_2(\mathbf{i}) = \mathbf{b}_2(i_1,\quad \ldots\quad ,(i_m-\Delta),\ldots i_d)+\mathbf{b}_1(\mathbf{i})-\mathbf{b}_1(i_1,\ldots,(i_m-\Delta),\ldots i_d)$$

Therefore, given $\mathbf{b}_1$, $\mathbf{b}_2$ can be calculated in scan order using 2 operations per pixel, and thus, $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$ are *efficiently computable*.

APPENDIX III
SEPARABLE GCK IN D-DIMENSIONS

As in the 2-dimensional case, a special class of d-dimensional kernels are of interest (Equation 2 in Section IV):

$$V^{(k_1,\ldots,k_d)}_{s_1,\ldots,s_d} = \{\mathbf{v}_1 \times \ldots \times \mathbf{v}_d \mid \mathbf{v}_i \in V^{(k_i)}_{s_i}\}$$

Thus the d-dimensional kernel $\mathbf{v} \in V^{(k_1,\ldots,k_d)}_{s_1,\ldots,s_d}$ is defined as $\mathbf{v}(i_1,\ldots,i_d) = \mathbf{v}_1(i_1)\mathbf{v}_2(i_2)\ldots\mathbf{v}_d(i_d)$ where $\mathbf{v}_i$ are one dimensional Gray Code kernels.

Since the d-dimensional kernel $\mathbf{v}$ is separable, and can be defined by d one dimensional kernels, the associated d α-indices uniquely define $\mathbf{v}$. Thus the following definition is consistent with the one and two dimensional cases, Definitions 3.2 and 4.3):

Definition 3.1: For $\mathbf{v} \in V^{(k_1,\ldots,k_d)}_{s_1\ldots,s_d}$ such that $\mathbf{v} = \mathbf{v}_1 \times \ldots \times \mathbf{v}_d$, with associated α-indices $\boldsymbol{\alpha_1},\ldots,\boldsymbol{\alpha_d}$, the sequence $\boldsymbol{\alpha} = [\boldsymbol{\alpha_1},\ldots,\boldsymbol{\alpha_d}]$ uniquely defines $\mathbf{v}$ and is called the $\boldsymbol{\alpha}$**-index** of $\mathbf{v}$.

Similar to the 2D case, the notion of $\boldsymbol{\alpha}$**-relation** between two d-dimensional kernels is defined:

Definition 3.2: Two kernels $\mathbf{v}_i, \mathbf{v}_j \in V^{(k_1,\ldots,k_d)}_{s_1\ldots s_d}$ are $\boldsymbol{\alpha}$**-related** iff the hamming distance of the signs of their α-index is one.

The notion of *Gray Code Sequence* of kernels can be extended to d-dimensions:

Definition 3.3: An ordered set of d-dimensional kernels, $\mathbf{v}_0 \ldots \mathbf{v}_n$ such that every consecutive pair are α-related, are called a **Gray Code Sequence** of d-dimensional kernels.

From Lemma 2.1 and Definition 3.3 we have the following corollary:

Corollary 3.4: Every two consecutive d-dimensional kernels in a Gray Code Sequence are *efficiently computable*.

REFERENCES

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

[2] F. C. Crow, "Summed-area tables for texture mapping," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. ACM Press, 1984, pp. 207–212.

[3] P. Simard, L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: a fast convolution algorithm for neural networks and signal processing," in *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.

[4] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," in *International Conference on Computer Vision*, Nice, France, 2003, pp. 1486–1493.

[5] ——, "Real time pattern matching using projection kernels," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 9, pp. 1430–1445, 2005.

[6] M. Werman, "Fast convolution," *Journal of WSCG*, vol. 11, no. 1, 2003.

[7] I. Drori and D. Lischinski, "Fast multiresolution image operations in the wavelet domain," *IEEE Trans. on Visualization and Computer Graphics*, vol. 9, no. 3, 2003.

[8] C. Gotsman, "Constant-time filtering by singular value decomposition," *Computer Graphics Forum, special issue on Rendering*, vol. 13, no. 2, pp. 153–163, 1994.

[9] P. Perona, "Deformable kernels for early vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 5, pp. 488–499, 1995.

[10] W. Freeman and E. Adelson, "The design and use of steerable filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 9, pp. 891–906, 1991.

[11] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Prentice-Hall, 2002.

[12] Y. Hel-Or and P. Teo, "Canonical decomposition of steerable functions," *J. Math. Imaging Vis.*, vol. 9, no. 1, pp. 83–95, 1998.

[13] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "Filtering with gray-code kernels," in *Proceedings of the 17th International Conference on Pattern Recognition*, Cambridge, UK, Sept 2004, pp. 556–559.

[14] S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Massachusetts: Addison-Wesley, 1990.

[15] M. Gardner, "The binary gray code," in *Knotted Doughnuts and Other Mathematical Entertainments*. New York: W.H. Freeman, 1986, pp. 23–24.

[16] N. Sloane, "The on-line encyclopedia of integer sequences, sequence number A014550." [Online]. Available: http://www.research.att.com/ njas/sequences/

[17] ——, "The on-line encyclopedia of integer sequences, sequence number A091299." [Online]. Available: http://www.research.att.com/ njas/sequences/

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

[19] D. Gensch, "An industrial application of the traveling salesman's subtour problem," *AIIE Transactions*, vol. 10, pp. 362–370, 1978.

[20] B. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics*, vol. 34, p. 307318, 1987.

[21] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, "Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen," in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995, pp. 277–283.

[22] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward tsp," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003, pp. 11–14.

[23] H. Kitajima, "Energy packing efficiency of the hadamard transform," *IEEE T-Comm.*, pp. 1256–1258, 1976.

[24] D. Ruderman, "Statistics of natural images," *Network: Computation in Neural Systems*, vol. 5, no. 4, pp. 517–548, 1994.

[25] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," in *IEEE International Conference on Computer Vision*, Corfu, Greece, Sept 1999, pp. 1033–1038.

[26] Y. Moshe and H. Hel-Or, "A fast block motion estimation algorithm using gray code kernels," in *submitted to ISSPIT*, Vancouver, canada, 2006.

[27] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Sixth International Conference on Computer Vision (ICCV'98)*, Bombay, India, Jan 1998, pp. 555–562.

PLACE PHOTO HERE

Gil Ben-Artzi Biography text here.

PLACE PHOTO HERE

Hagit Hel-Or Biography text here.

PLACE PHOTO HERE

Yacov Hel-Or Biography text here.