Article

An Analytic Transform Kernel Derivation Method for Video Codecs

Ankit Kumar and Bumshik Lee

Department of Information and Communication Engineering, Chosun University, Gwangju 61452, Korea; ankitku@chosun.kr

* Correspondence: bslee@chosun.ac.kr

Abstract: In the standardization of versatile video coding (VVC), discrete cosine transform (DCT)-2, discrete sine transform (DST)-7, and DCT-8 are regarded as the primary transform kernels. However, DST-4 and DCT-4 can also be considered as the transform kernels instead of using DST-7 and DCT-8 owing to their effectiveness in smaller resolution test sequences. To implement these different block size transform kernels, a considerable amount of memory has to be allocated. Moreover, memory consumption to store different block size transform kernels is regarded as a major issue in video coding standardization. To address this problem, a common sparse unified matrix concept is introduced in this study, where any block size transform kernel matrix can be obtained after some mathematical operations. The proposed common sparse unified matrix saves approximately 80% of the static memory by storing only a few transform kernel elements for DCT-2, DST-7, and DCT-8. Full-required transform kernels are derived using the stored transform kernels and generated unit-element matrices and a permutation matrix. The static memory required is only for 1648 elements instead of 8180 elements, each with 8-bit precision. The defined common sparse unified matrix is composed of two parts: a unified DST-3 matrix and a grouped DST-7 matrix. The unified DST-3 matrix is used to derive different points of DCT-2 transform kernels, and the grouped DST-7 matrix is used to derive different points of DST-7 and DCT-8 transform kernels. The new technique of grouping concept is introduced, which shows the relationship between different rows of DST-7 transform kernels with various block sizes. The proposed grouping concept supports the fast algorithm of DST-7 by implementing the proposed method of the “one group one feature” principle. The simulation was conducted using the VTM-3.0 reference software under common test conditions. The simulation result of the all intra (AI) configuration is $Y = 0.00\%$, $U = -0.02\%$, $V = 0.00\%$ with an encoding time of 100%, and a decoding time of 100%. Similarly, the simulation results of random access (RA) configuration are $Y = -0.01\%$, $U = 0.09\%$, $V = 0.06\%$, and the encoding and decoding times are 101% and 100%, respectively. The simulation result of the low delay B (LDB) configuration is $Y = 0.01\%$, $U = 0.08\%$, and $V = -0.27\%$, for encoding and decoding times of 101% and 100%, respectively.

Keywords: VVC; DCT-2; DST-7; DCT-8; DST-3; DST-4; DCT-4; common sparse unified matrix; unified DST-3 matrix; unit element matrices; grouped DST-7 matrix; permutation matrix

Citation: Kumar, A.; Lee, B. An Analytic Transform Kernel Derivation Method for Video Codecs. *Appl. Sci.* **2021**, *11*, 9280. <https://doi.org/10.3390/app11199280>

Academic Editor: Andrea Prati

Received: 25 August 2021

Accepted: 29 September 2021

Published: 6 October 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Various standardization works for video compression have been performed, including H.261, H.262, H.263 [1], H.264/advanced video coding [2], and H.265/ high-efficiency video coding (HEVC) [3]. Recently, the standardization process of versatile video coding (VVC) [4], which aims to provide a significant improvement in compression performance over the existing HEVC standard and aid the deployment of higher quality video services and emerging applications, such as 360° omnidirectional immersive multimedia and high-dynamic-range video, has been standardized. In the standardization of VVC, discrete cosine transform-2 (DCT-2), discrete sine transform-7 (DST-7), and DCT-8 are regarded as three vital primary transform kernels [5–9], which can also be termed as the multiple

transform set (MTS) [10,11]. The benefits of MTS are usually to enlarge transform sets when necessary, that is, reduced distortion in the R-D trade-off as complex residuals exploit the multiple transforms and reduce transform sets when necessary. This means reduced bitrate in the R-D trade-off by avoiding wasting bits signaling the transforms when necessary [12]. Similarly, DST-7 approximates the optimal transform better than DCT-2 along the prediction direction, that is, for intra prediction residuals [11].

Along with these three transform kernels, DST-4 and DCT-4 can also be regarded as the replacement of DST-7 and DCT-8, respectively, because they exhibit similar kernel behavior and show better coding efficiency for smaller resolution sequences [13].

The used transform kernel elements are derived based on the following mathematical equations.

DCT-2

$$[C_2]_{n,k} = \sqrt{\frac{2}{N}} \varepsilon_k \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad (1)$$

$$\text{where } \varepsilon_k = \begin{cases} \frac{1}{\sqrt{2}} & k = 0 \\ 1 & \text{otherwise} \end{cases}$$

DCT-4

$$[C_4]_{n,k} = \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)(2k+1)}{4N}\right) \quad (2)$$

DCT-8

$$[C_8]_{n,k} = \frac{2}{\sqrt{2(N-1)}} \cos\left(\frac{\pi(2n+1)(2k+1)}{4N-2}\right) \quad (3)$$

DST-4

$$[S_8]_{n,k} = \sqrt{\frac{2}{2N}} \varepsilon_k \sin\left(\frac{\pi(2n+1)(k+1)}{2N}\right) \quad (4)$$

$$\text{where, } \varepsilon_k = \begin{cases} \frac{1}{\sqrt{2}} & k = N-1 \\ 1 & \text{otherwise} \end{cases}$$

DST-7

$$[S_7]_{n,k} = \frac{2}{\sqrt{2N+1}} \sin\left(\frac{\pi(2k+1)(n+1)}{2N+1}\right) \quad (5)$$

where N represents the block size.

Several design aspects of HEVC transform coding are inherited in the VVC codec. In addition to the conventional DCT-2, alternate transform types like DST-7 and DCT-8 are also adopted in the VVC, which is generally referred to as primary transform because it is first applied to the predicted residual pixels and named as contrast meaning of the secondary transform [14] in VVC. The size of DCT-2 ranges from 4-point to 64-point, whereas that of DST-7 and DCT-8 ranges from 4-point to 32-point. The kernels elements defined in VVC are composed of 8-bit signed integer values. The additional integer transform kernels defined in VVC are derived by scaling the floating-point transform kernel with $64 \sqrt{N}$, where N represents the transform size [14]. For the alignment of the worst-case multiplications per coefficients with HEVC, for 64-point DCT-2 and 32-point DST-7/DCT-8, only the first 32 and 16 low-frequency coefficients are kept, and the higher frequency coefficients are zeroed out respectively, which is also considered in the last coefficient position coding and the coefficient group scanning [15].

All the kernels elements are stored in the reference software VTM-3.0 in an 8-bit representation. A single 64-point DCT-2 kernel matrix is newly defined in VCC, and other sizes of DCT-2 are kept unchanged with HEVC. Other individual sets of 4-, 8-, 16-, and 32-point DST-7 and DCT-8 kernel matrices are also stored in the reference software [16], adding together to the storage of 8180 elements which is a prime concern for the memory. Unlike the conventional video codec such as HEVC, the partial butterfly structure for the transform of DST-7 and DCT-8 is avoided in the transform process to be compatible with the virtual pipeline process and low delay in hardware design, which is thoroughly

investigated and agreed in the standardization works of VVC. Instead, the fast transform method is designed to support the dual implementation of matrix multiplication and butterfly operation [17]. Thus, all kernel elements for matrix multiplication operation should be stored in the codec devices or software. However, the number of multiplication and addition is significantly reduced even though matrix operations are performed with matrix multiplications by utilizing several features of DST-7 and DCT-8 kernels [17]. Table 1 shows the use of MTS transform kernels based on the prediction modes and block sizes in VVC.

Table 1. MTS transform kernels based on prediction and block sizes in VTM-3.0.

Prediction		Prediction Tools	Block Size Checks	Transform Kernels
MTS	Intra	ISP [18] (intra sub-partition)	4×4 up to 16×16	DCT-2, DST-7
		Normal intra-prediction mode	4×4 up to $N \times 32/32 \times N$	DCT-2, DST-7
	Inter	SBT [19] (sub-block transform)	4×4 up to $N \times 32/32 \times N$	DST-7, DCT-8 depends on SBT position

Table 1 shows the use of multiple transform set (MTS) for intra and inter prediction. In the intra prediction, various prediction tools have been introduced in VVC, which are intra-sub-partition (ISP) and normal prediction. For ISP, the combinations of DCT-2 and DST-7 transform kernels are used as the horizontal and vertical transform types with a transform block size of 4×4 up to 16×16 . For the normal intra-prediction modes, the combinations of DCT-2 and DST-7 transform kernel sets are used, based on the prediction mode and comparison of width and height of the block sizes, ranging from 4 to 16 pixels in accordance with the spec text [15]. The 16-point or 32-point DST-7 is applied to non-zero coefficients transform blocks in either the horizontal or vertical direction. For inter prediction, using sub-block transform (SBT) [19], as a prediction tool, the combinations of DST-7 and DCT-8 are applied on the partitioned transform unit block for a block size up to 32 pixels, depending on the `cu_sbt_horizontal_flag` and `cu_sbt_vertical_flag` [15] in VTM-3.0.

Despite significant coding gain improvement owing to the usage of MTS for different block sizes and kernels, memory storage of kernel elements is regarded as one of the crucial issues for the standardization of VVC. In MTS, DST-4 and DCT-4 can be used as a replacement for the DST-7 and DCT-8 transform kernels, respectively, owing to their relatively easier kernel derivation by sub-sampling of even and odd rows of higher-order DCT-2 transform kernels [20].

DST-4/DCT-4 provides significant improvement for lower resolution test sequences, whereas the use of DST-7 and DCT-8 provides significant results for all resolution test sequences; however, the derivation of different points DST-7 and DCT-8 transform kernels cannot be achieved by subsampling the larger block size DCT-2 transform kernels. Hence, separate memory must be allocated for the storage of these transform kernels, which results in the memory issue in VVC. The total number of elements for which memory has to be allocated is 8180 elements with 8-bit precision. To address this scenario, various proposals have been presented in VVC.

In [21], a 64×64 compound orthonormal transform (COT) matrix is introduced, which comprises two aspects: (1) for 4-point and 8-point DST-4/DCT-4 transform kernels replacing DST-7/DCT-8 transform kernels, and (2) 16-point and 32-point DST-7/DCT-8 transform kernels embedded into 64-point DCT-2. It is implemented using a single 64×64 matrix, which has a total of 8180 elements and provides all three types of transform kernels. In [21], 2-, 4-, 8-, 16-, and 32-point DCT-2 can be extracted from the even rows 64-point DCT-2, whereas 4-, 8-point DST-4/DCT-4 and 16- and 32-point DST-7/DCT-8 can be extracted from the odd rows 64-point DCT-2. The metrics of the operation counts and minimum bit-precision remain the same as VTM-3.0. With this approach of using a single 64×64 matrix, 33% of transform storage, i.e., 2.7 kilobytes of memory, can be saved compared to VTM-3.0.

Although the proposed method provides good gains, it fails to provide the precise 64-point DCT-2 because some of the kernel values of DST-7/DCT-8 are embedded in some rows of 64-point DCT-2. Owing to the mismatch of the proposed 64-point DCT-2 transform kernel with the original DCT-2 transform kernel, coding performance improvement is not significant when only the DCT-2 transform kernel is used. Conversely, the fast computation algorithm [22] for the 64-point DCT-2 transform cannot be used. Similarly, the number of transform kernel matrices is also increased to 5, that is DCT-2, DST-7, DCT-8, DST-4, and DCT-4, even though it was aimed to use only three transform kernels.

In [23], a 64-point unified matrix was proposed. It is implemented using the 64×64 unified matrix with a total of 8180 elements and provides all three types of transform with all block sizes from 4×4 to 64×64 by sampling a subset of coefficients and basis vectors of larger transforms. This method is designed with an 8-bit representation. Using simple calculations from the proposed matrix, different kernel types with different sizes were derived. This method uses 131,072 bits amount of kernel elements which account for 16 kilobytes of memory for the total bits amount and is 19% lesser than the total memory of bits used in VTM-3.0. Experimental results show that no gain is observed for all intra (AI) configurations for higher resolution test sequences, that is, class A1 and A2, but they comprise some gain for lower resolution test sequences, that is, classes B, C, D, and E. For random access (RA) and low delay B (LDB) configurations, the overall result shows no gain. The losses are because of the mismatch between the original and derived DST-7 and DCT-8 transform kernels.

Similarly, in [24], the adjustment stage concept for memory reduction was introduced. The adjustment stages are defined as sparse block-band orthogonal matrices and are similar to filters with small numbers of taps. It is proposed to approximate different sizes and types of cosine and sine transform such as DCT-5, DCT-8, DST-1, and DST-7 by applying different adjustment matrices (stages). The major drawback of this method is that the number of multiplications increases by using these adjustment stages, and the normalized values also differ for different transform kernels, which results in a mismatch with the original transform kernels. A transform adjustment filter (TAF) was introduced in [25], which is similar to the adjustment stage in [24]. A sparse matrix is used as a preprocessor to the partial butterfly DCT-2 algorithm. It approximates the DCT-8 and DST-7 with an adjustment stage followed by DCT-2. The DCT-8/DST-7 kernels are approximated by the combination of a block band matrix and a DCT-2 kernel. The size-16, size-32, and size-64 adjustment matrices require the overall 356 8-bit coefficients. Also, the adjustment stage is efficiently implemented using the SIMD instructions. Symmetries are applied for the 32-point and 64-point adjustment matrix having the storage requirement of 93 and 189 8-bit coefficients, respectively, whereas, for the 16-point adjustment matrix, 74 coefficients are required. All the adjustment matrix coefficients require 8-bit storage. This method reduces the complexity of the DST-7/DCT-8 transform and reduces the memory usage by storing only the different point DCT-2 transform kernels. Although the number of TAFs is reduced compared to [24], the total number of multiplications increases, i.e., total multiplication required is 160, 482, and 1468 for 16-, 32- and 64-point, respectively. This is because additional multiplication with the TAF has to be performed, and the normalization values of TAF are also not identical, thus resulting in the mismatch of the derived transform kernels with the original transform kernels.

In this study, an analytical derivation of the transform kernel using a common sparse unified matrix is introduced. The proposed method reduces the total amount of memory for the MTS kernels by storing only 1648 elements instead of 8180 elements with 8-bit precision. The transform kernels used in this study were DCT-2, DST-7, and DCT-8. Using the proposed method, DST-4 and DCT-4 transform kernels can also be achieved without affecting the memory. Similarly, the proposed method supports the implementation of the fast algorithm of DCT-2 [22] and DST-7 [17]. Figure 1 shows the proposed common sparse unified matrix, composed of two parts: Unified DST-3 matrix (U) and grouped DST-7 matrix. The U matrix is composed of different block sizes of DST-4 and is used to derive

different block sizes of DCT-2 transform kernels. It stores 1368 elements with 8-bit precision. Similarly, grouped DST-7 is used to derive different block sizes of DST-7 transform kernels. It comprises **P**, **Q**, **R**, and **S** matrices, which are different selected rows from 32-point, 16-point, 8-point, and 4-point DST-7 transform kernels. It stores 280 elements with 8-bit precision. Hence, the total memory allocation must be appointed for only 1648 elements for the overall common sparse unified matrix.

Figure 1. Common sparse unified matrix.

The remainder of this paper is organized as follows. In Section 2, the proposed MTS kernel derivation method is described. In Section 3, the experimental results are provided, and the concluding statements are provided in Section 4.

2. Proposed Common Sparse Unified Matrix

Figure 1 shows the proposed common sparse unified matrix. It comprises two parts: a unified DST-3 matrix (**U**) and a grouped DST-7 matrix. The unified DST-3 matrix (**U**) is used to derive the different sizes of DCT-2 transform kernels, whereas the grouped DST-7 matrix is used to derive the different sizes of DST-7 and DCT-8 transform kernels, where DCT-2, DST-7, and DCT-8 are the transform kernels of MTS in VVC. The **U** matrix includes 32-, 16-, 4-, and 2-point DST-4 transform kernels and a 2-point DST-3 transform kernel. Similarly, the grouped DST-7 matrix comprises rows selected from different point DST-7 transform kernels, that is, 6×32 block size **P**, 4×16 block size **Q**, 2×8 block size **R**, and 2×4 block size **S** matrices are the row elements selected from 32, 16, 8, and 4-point DST-7 transform kernels, respectively.

Note that the overall size of the proposed common sparse unified matrix is 1648 bytes (integer, 8 bit-precision). The original size of the transform kernel is 8180 bytes (integer, 8-bit precision) for storing different block sizes of DCT-2, DST-7, and DCT-8.

2.1. Unified DST-3 Matrix (**U**)

The proposed unified DST-3 matrix (**U**) was used to derive DCT-2 transform kernels of different sizes. It stores only 1368 elements with 8-bit precision for deriving any block size of the DCT-2 transform kernel, whereas the current VVC allocates memory for 5460 elements with 8-bit precision for deriving different block sizes of the DCT-2 transform kernels. The proposed basic structure of the unified DST-3 matrix is shown in Figure 2.

Figure 2. Structure of unified DST-3 matrix (**U**).

The matrix **U** comprises 32, 16, 8, 4, and 2-point DST-4 and a 2-point DST-3 matrix. The remaining elements of **U** were set to zero. A relationship exists between DST-3 and DCT-2 [26], which can be expressed as Equation (6).

$$\mathbf{C}_2 = \mathbf{F} \times \mathbf{S}_3 \times \mathbf{S} \quad (6)$$

where $\mathbf{C}_2$ is DCT-2, $\mathbf{S}_3$ is DST-3, and **F** and **S** are flipping and sign change matrices, respectively. The elements in **F** and **S** are defined as

$$F_{m,n} = \begin{cases} 1, & \text{if } n = N - 1 - m, \\ 0, & \text{otherwise,} \end{cases} \quad (7)$$

$$S_{m,n} = \begin{cases} (-1)^m, & \text{if } n = m, \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

where m and n are matrix indices, and N is the size of the matrix.

To retrieve any block size of the DCT-2 kernel, different block sizes of the DST-3 kernel are required, as in (6). As shown in Figure 2, a 2-point DST-3 transform kernel matrix exists at the right bottom part of **U**. Using (6) and the 2-point DST-3 transform kernel, a 2-point DCT-2 transform kernel can be derived. Similarly, different block sizes of DST-3 transform kernels are required to derive other sizes of DCT-2 transform kernels. To obtain different block sizes of DST-3 transform kernels from **U**, the predefined 64-point unit-element matrices, **A**, **B**, **C**, **D**, and **E**, are multiplied by **U**.

Figure 3 shows the DST-3 derivation mechanism from unified DST-3 matrix (**U**). The unit-element matrices **A**, **B**, **C**, **D**, and **E** are the matrices with values of -1 , 0 , or 1 , and their elements are defined as Equations (9)–(13), respectively.

$$A_{m,n} = \begin{cases} 1, & \text{if } (m = n, \forall 0 < (m,n) < 60) \parallel \\ & (m = n, \forall 60 < (m,n) < 62) \parallel \\ & \left(\begin{array}{l} m + n = 123 \forall (61 < m < 64) \text{ and} \\ (60 < n < 62) \end{array} \right) \parallel \\ & \left(\begin{array}{l} i - j = -2, \quad \forall (60 < m < 62) \text{ and} \\ (61 < n < 64) \end{array} \right) \\ -1, & \text{if } (m + n = 125, \forall 61 < (m,n) < 64) \\ 0, & \text{elsewhere} \end{cases} \quad (9)$$

$$B_{m,n} = \begin{cases} 1, & \text{if } (m = n, \forall 0 < (m,n) < 56) \parallel \\ & (m = n, \forall 55 < (m,n) < 60) \parallel \\ & \left(\begin{array}{l} m + n = 119, \forall (59 < m < 64) \text{ and} \\ (55 < n < 60) \end{array} \right) \parallel \\ & \left(\begin{array}{l} i - j = -4, \forall (55 < m < 60) \text{ and} \\ (59 < n < 60) \end{array} \right) \parallel \\ -1, & \text{if } (m + n = 123, \forall 59 < (m,n) < 64) \\ 0, & \text{elsewhere} \end{cases} \quad (10)$$

$$C_{m,n} = \begin{cases} 1, & \text{if } (m = n, \forall 0 < (m,n) < 48) \parallel \\ & (m = n, \forall 47 < (m,n) < 56) \parallel \\ & \left(\begin{array}{l} m + n = 111, \forall (55 < m < 64) \text{ and} \\ (47 < n < 56) \end{array} \right) \parallel \\ & \left(\begin{array}{l} i - j = -8, \forall (47 < m < 56) \text{ and} \\ (55 < n < 64) \end{array} \right) \parallel \\ -1, & \text{if } (m + n = 119, \forall 55 < (m,n) < 64) \\ 0, & \text{elsewhere} \end{cases} \quad (11)$$

$$D_{m,n} = \begin{cases} 1, & \text{if } (m = n, \forall 0 < (m,n) < 32) \parallel \\ & (m = n, \forall 31 < (m,n) < 48) \parallel \\ & \left(\begin{array}{l} m + n = 95, \forall (47 < m < 64) \text{ and} \\ (31 < n < 48) \end{array} \right) \parallel \\ & \left(\begin{array}{l} i - j = -16, \forall (31 < m < 48) \text{ and} \\ (47 < n < 64) \end{array} \right) \parallel \\ -1, & \text{if } (m + n = 111, \forall 47 < (m,n) < 64) \\ 0, & \text{elsewhere} \end{cases} \quad (12)$$

$$E_{m,n} = \begin{cases} 1, & \text{if } (m = n, \forall 0 < (m,n) < 32) \parallel \\ & \left(\begin{array}{l} m + n = 63, \forall (31 < m < 64) \text{ and} \\ (0 < n < 32) \end{array} \right) \parallel \\ & \left(\begin{array}{l} i - j = -32, \forall (0 < m < 32) \text{ and} \\ (31 < n < 64) \end{array} \right) \parallel \\ -1, & \text{if } (m + n = 95, \forall 31 < (m,n) < 64) \\ 0, & \text{elsewhere} \end{cases} \quad (13)$$

Figure 3. DST-3 derivation mechanism from unified DST-3 matrix (U).

The relation between **U** and the unit-element matrices can be expressed as Equation (14).

$$\mathbf{S}_{3,64} = \mathbf{U} \times \mathbf{A} \times \mathbf{B} \times \mathbf{C} \times \mathbf{D} \times \mathbf{E} \quad (14)$$

where $\mathbf{S}_{3,64}$ is the 64-point DST-3 transform kernel, **U** is the proposed unified DST-3 matrix, and **A**, **B**, **C**, **D**, and **E** are the 64-point unit-element matrices. Using **U** and the unit-element

matrices, DST-3 transform kernels with different sizes can be obtained, which is explained using Equations (15)–(19)

$$\mathbf{S}_{3,4} = \mathbf{T}[\mathbf{U}_{64} \times \mathbf{A}_{64}]_4 \quad (15)$$

$$\mathbf{S}_{3,8} = \mathbf{T}[\mathbf{U}_{64} \times \mathbf{M}_{64}]_8 \quad (16)$$

$$\mathbf{S}_{3,16} = \mathbf{T}[\mathbf{U}_{64} \times \mathbf{N}_{64}]_{16} \quad (17)$$

$$\mathbf{S}_{3,32} = \mathbf{T}[\mathbf{U}_{64} \times \mathbf{O}_{64}]_{32} \quad (18)$$

$$\mathbf{S}_{3,64} = \mathbf{U}_{64} \times \mathbf{P}_{64} \quad (19)$$

where $\mathbf{S}_{3,N}$ is an N -point DST-3 transform kernel, $\mathbf{T}[\cdot]_N$ represents a function that takes an $N \times N$ block of the right-bottom part of the matrix, and $\mathbf{M}$, $\mathbf{N}$, $\mathbf{O}$, and $\mathbf{P}$ are defined using unit elements (-1 , 0 , and 1) as in Equation (20).

$$\begin{aligned} \mathbf{M} &= \mathbf{A} \times \mathbf{B} \\ \mathbf{N} &= \mathbf{A} \times \mathbf{B} \times \mathbf{C} \\ \mathbf{O} &= \mathbf{A} \times \mathbf{B} \times \mathbf{C} \times \mathbf{D} \\ \mathbf{P} &= \mathbf{A} \times \mathbf{B} \times \mathbf{C} \times \mathbf{D} \times \mathbf{E} \end{aligned} \quad (20)$$

For example, to derive the 16-point DCT-2 kernel, a 16-point DST-3 is required, as in Equation (6). First, $\mathbf{U}$ is multiplied by the $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ unit-element matrices as in (17) in sequential order, thus leading to 32-point DST-4, 16-point DST-4, and 16-point DST-3 transform kernel matrices. Using the resulting matrices, the 16-point DST-3 matrix at the bottom right of the generated matrix is obtained. Finally, the 16-point DCT-2 transform kernel can be obtained using (6) and a 16-point DST-3. Figure 4 shows the overall process of generation of the 16-point DST-3 transform kernel.

Figure 4. Proposed method to obtain the 16-point DST-3 transform kernel.

Figure 5 shows the proposed derivation of the N -point DCT-2 transform kernel. First, the value of N is equal to 2. If the condition is satisfied, $\mathbf{S}_{3,2}$ is obtained by the selection of a 2×2 block of the right-bottom part of the $\mathbf{U}$ matrix; otherwise, $\mathbf{S}_{3,N}$ is achieved by the selection of the $N \times N$ block of the bottom right of the matrix obtained after the multiplication of $\mathbf{U}$, and $\mathbf{E}_{64}$. $\mathbf{E}_{64}$ represents the unit-element matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$. $\mathbf{E}_{64,N}$ can be obtained by multiplying the unit-element matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$, depending on N . For example, $\mathbf{E}_{64,N}$ is calculated as $\mathbf{A} \times \mathbf{B} \times \mathbf{C} \times \mathbf{D}$, $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{A}$ for $N = 32$, $N = 16$, $N = 8$, and $N = 4$, respectively. Finally, the N -point DCT-2 transform kernel was obtained using (6). Figure 5 shows an illustration of the proposed DCT-2 transform kernel derivation.

Figure 5. Illustration of the proposed DCT-2 transform kernel.

2.2. Proposed Grouped DST-7 Matrix

To derive the full DST-7 and DCT-8 matrices with less memory, a grouped DST-7 matrix is proposed. The proposed grouped DST-7 matrix is based on the principle of storing only a few selected rows of different block-sized DST-7 transform kernels. The rows selected from the original kernel matrix are used to generate the group of elements of the respective block size DST-7 transform kernel with the help of the permutation matrix (**G**) comprising $-1, 0$, or 1 . The matrix **G** is multiplied by the selected rows, resulting in a full block-sized DST-7 transform kernel. The matrices **P**, **Q**, **R**, and **S** in Figure 1 indicate the rows for the derivation of 32, 16, 8, and 4-point DST-7 transform kernels, respectively, and they are only stored for DST-7 and DCT-8 kernel derivation, leading to a significant reduction in memory storage. The grouped DST-7 matrix is defined as a 14×64 matrix of the lower part, as shown in Figure 1, and the total memory allocation is required for only 280 elements with 8-bit precision, which significantly reduces the memory sizes of the current DST-7 in VVC with 1360 elements each with 8-bit precision.

To derive the full-size 32-point DST-7 transform kernel, only six-row vectors from the 32-point DST-7 are selected, that is, $\mathbf{P} = [\mathbf{S}_{7,0} \ \mathbf{S}_{7,1} \ \mathbf{S}_{7,2} \ \mathbf{S}_{7,3} \ \mathbf{S}_{7,5} \ \mathbf{S}_{7,6}]^T$ and can be base vectors for each group of DST-7 matrix in the proposed method. Similarly, for the derivation of full-size 16, 8, and 4-point DST-7 transform kernels, **Q** with four rows, **R** with two rows, and **S** with two rows, that is, $\mathbf{Q} = [\mathbf{S}_{7,0} \ \mathbf{S}_{7,1} \ \mathbf{S}_{7,2} \ \mathbf{S}_{7,5}]^T$, $\mathbf{R} = [\mathbf{S}_{7,0} \ \mathbf{S}_{7,1}]^T$, and $\mathbf{S} = [\mathbf{S}_{7,0} \ \mathbf{S}_{7,1}]^T$, are selected from 16, 8, and 4-point DST-7, respectively and are regarded as the base vectors. Note that $\mathbf{S}_{7,i}$ represents the i -th row of DST-7. Using a few selected rows and **G**, the full-size DST-7 transform kernel can be derived by retrieving the row vectors of each group. Figure 6 shows an illustration of kernel derivation using a grouped 32-point DST-7 matrix. As shown in Figure 6, the row vectors of **P** in the grouped DST-7 matrix are used to derive the row vectors of the DST-7 kernel in each group.

Figure 6. Illustration of kernel derivation using grouped DST-7 matrix.

Figure 7 shows the details of the derivation process of the kernel for each group. The 0-th group is taken as an example, and the other groups follow the same process using different row vectors of matrix **P**.

Figure 7. An example of derivation process for the i -th group of 32-point DST-7.

In Figure 7, $[S_{7,(i,j)}]_N$ denotes the j -th row vector of the i -th group for the N -point DST-7. In the proposed method, matrix **P** comprises only the 0-th row vectors of the i -th group of the N -point DST-7. First, the 0-th row vector $[S_{7,(i,0)}]_N$ from **P** is obtained from the i -th group of DST-7. Subsequently, the absolute values of e_l and e_f , which are the last (the rightmost) and first (the leftmost) elements in the row vector, are compared. If these two absolute values are identical, the row vector can be one row of the DST-7 kernel with the same index as stored in **P**. Otherwise, the row vector is multiplied by **G** and a comparison of e_l and e_f in the resulting row vector is performed as in the previous step. If the first element in each row is a negative value, then the sign of every element in each row is altered. It should be noted that **G** is designed such that the absolute value of the first element in the previous row equals the absolute value of the last element of the next row vector after multiplying with **G**. This process is repeated until the end of the last row vector of the current group.

Figure 8 shows a 16-point DST-7 transform kernel derived using Figure 7, and the elements in each row vectors are perfectly matched with the definition of DST-7 in (5). Similarly, 32-, 8-, and 4-point DST-7 can be derived using **P**, **R**, **S**, and **G** matrices, which are defined according to the size, which will be discussed in the next section.

$$\begin{aligned}
[S_{7,(0,0)}]_{16} &= [0.033 \ 0.066 \ 0.098 \ 0.129 \ 0.160 \ 0.188 \ 0.215 \ 0.240 \ 0.263 \ 0.284 \ 0.302 \ 0.317 \ 0.329 \ 0.338 \ 0.345 \ 0.348] \\
[S_{7,(0,15)}]_{16} &= [0.066 \ -0.129 \ 0.188 \ -0.240 \ 0.284 \ -0.317 \ 0.338 \ -0.348 \ 0.345 \ -0.329 \ 0.302 \ -0.263 \ 0.215 \ -0.160 \ 0.098 \ -0.033] \\
[S_{7,(0,14)}]_{16} &= [0.129 \ -0.240 \ 0.317 \ -0.348 \ 0.329 \ -0.263 \ 0.160 \ -0.033 \ -0.098 \ 0.215 \ -0.302 \ 0.345 \ -0.338 \ 0.284 \ -0.188 \ 0.066] \\
[S_{7,(0,12)}]_{16} &= [0.240 \ -0.348 \ 0.263 \ -0.033 \ -0.215 \ 0.345 \ -0.284 \ 0.066 \ 0.188 \ -0.338 \ 0.302 \ -0.098 \ -0.160 \ 0.329 \ -0.317 \ 0.129] \\
[S_{7,(0,8)}]_{16} &= [0.348 \ -0.033 \ -0.345 \ 0.066 \ 0.338 \ -0.098 \ -0.329 \ 0.129 \ 0.317 \ -0.160 \ -0.302 \ 0.188 \ 0.284 \ -0.215 \ -0.263 \ 0.240] \\
\\
[S_{7,(1,1)}]_{16} &= [0.098 \ 0.188 \ 0.263 \ 0.317 \ 0.345 \ 0.345 \ 0.317 \ 0.263 \ 0.188 \ 0.098 \ 0 \ -0.098 \ -0.188 \ -0.263 \ -0.317 \ -0.345] \\
[S_{7,(1,13)}]_{16} &= [0.188 \ -0.317 \ 0.345 \ -0.263 \ 0.098 \ 0.098 \ -0.263 \ 0.345 \ -0.317 \ 0.188 \ 0 \ -0.188 \ 0.317 \ -0.345 \ 0.263 \ -0.098] \\
[S_{7,(1,10)}]_{16} &= [0.317 \ -0.263 \ -0.098 \ 0.345 \ -0.188 \ -0.188 \ 0.345 \ -0.098 \ -0.263 \ 0.317 \ 0 \ -0.317 \ 0.263 \ 0.098 \ -0.345 \ 0.188] \\
[S_{7,(1,4)}]_{16} &= [0.263 \ 0.345 \ 0.188 \ -0.098 \ -0.317 \ -0.317 \ -0.098 \ 0.188 \ 0.345 \ 0.263 \ 0 \ -0.263 \ -0.345 \ -0.188 \ 0.098 \ 0.317] \\
[S_{7,(1,7)}]_{16} &= [0.345 \ 0.098 \ -0.317 \ -0.188 \ 0.263 \ 0.263 \ -0.188 \ -0.317 \ 0.098 \ 0.345 \ 0 \ -0.345 \ -0.098 \ 0.317 \ 0.188 \ -0.263] \\
\\
[S_{7,(2,2)}]_{16} &= [0.160 \ 0.284 \ 0.345 \ 0.329 \ 0.240 \ 0.098 \ -0.066 \ -0.215 \ -0.317 \ -0.348 \ -0.302 \ -0.188 \ -0.033 \ 0.129 \ 0.263 \ 0.338] \\
[S_{7,(2,11)}]_{16} &= [0.284 \ -0.329 \ 0.098 \ 0.215 \ -0.348 \ 0.188 \ 0.129 \ -0.338 \ 0.263 \ 0.033 \ -0.302 \ 0.317 \ -0.066 \ -0.240 \ 0.345 \ -0.160] \\
[S_{7,(2,6)}]_{16} &= [0.329 \ 0.215 \ -0.188 \ -0.338 \ -0.033 \ 0.317 \ 0.240 \ -0.160 \ -0.345 \ -0.066 \ 0.302 \ 0.263 \ -0.129 \ -0.348 \ -0.098 \ 0.284] \\
[S_{7,(2,3)}]_{16} &= [0.215 \ 0.338 \ 0.317 \ 0.160 \ -0.066 \ -0.263 \ -0.348 \ -0.284 \ -0.098 \ 0.129 \ 0.302 \ 0.345 \ 0.240 \ 0.033 \ -0.188 \ -0.329] \\
[S_{7,(2,9)}]_{16} &= [0.338 \ -0.160 \ -0.263 \ 0.284 \ 0.129 \ -0.345 \ 0.033 \ 0.329 \ -0.188 \ -0.240 \ 0.302 \ 0.098 \ -0.348 \ 0.066 \ 0.317 \ -0.215] \\
\\
[S_{7,(5,5)}]_{16} &= [0.302 \ 0.302 \ 0 \ -0.302 \ -0.302 \ 0 \ 0.302 \ 0.302 \ 0 \ -0.302 \ -0.302 \ 0 \ 0.302 \ 0.302 \ 0 \ -0.302]
\end{aligned}$$

Figure 8. Floating-point grouped 16-point DST-7 transform kernel elements.

For the derivation of the full-size 16-point DST-7 transform kernel, only four rows from the 16-point DST-7 are selected, that is, $\mathbf{Q} = [S_{7,(0,0)} \ S_{7,(1,1)} \ S_{7,(2,2)} \ S_{7,(5,5)}]^T$ and can be used as base vectors for each group of DST-7 matrix in the proposed method.

2.3. Permutation Matrix (G)

The permutation matrix $\mathbf{G}$ plays a crucial role in the derivation of the DST-7 transform kernel using the proposed grouping algorithm, as mentioned in the previous section. The process of deriving $\mathbf{G}$ in the proposed method is described in Figure 9, which shows the first group of 16-point DST-7 transform kernels represented with floating-point elements, where $S_{7,(0,0)}$ is only stored in matrix $\mathbf{Q}$ in the common sparse unified matrix shown in Figure 1.

$$\begin{aligned}
[S_{7,(0,0)}]_{16} &= [0.033 \ 0.066 \ 0.098 \ 0.129 \ 0.160 \ 0.188 \ 0.215 \ 0.240 \ 0.263 \ 0.284 \ 0.302 \ 0.317 \ 0.329 \ 0.338 \ 0.345 \ 0.348] \\
[S_{7,(0,15)}]_{16} &= [0.066 \ -0.129 \ 0.188 \ -0.240 \ 0.284 \ -0.317 \ 0.338 \ -0.348 \ 0.345 \ -0.329 \ 0.302 \ -0.263 \ 0.215 \ -0.160 \ 0.098 \ -0.033] \\
[S_{7,(0,14)}]_{16} &= [0.129 \ -0.240 \ 0.317 \ -0.348 \ 0.329 \ -0.263 \ 0.160 \ -0.033 \ -0.098 \ 0.215 \ -0.302 \ 0.345 \ -0.338 \ 0.284 \ -0.188 \ 0.066] \\
[S_{7,(0,12)}]_{16} &= [0.240 \ -0.348 \ 0.263 \ -0.033 \ -0.215 \ 0.345 \ -0.284 \ 0.066 \ 0.188 \ -0.338 \ 0.302 \ -0.098 \ -0.160 \ 0.329 \ -0.317 \ 0.129] \\
[S_{7,(0,8)}]_{16} &= [0.348 \ -0.033 \ -0.345 \ 0.066 \ 0.338 \ -0.098 \ -0.329 \ 0.129 \ 0.317 \ -0.160 \ -0.302 \ 0.188 \ 0.284 \ -0.215 \ -0.263 \ 0.240]
\end{aligned}$$

Figure 9. $S_{7,(0,0)}$ dependent grouped 16-point DST-7 transform kernel.

The elements between the consecutive rows in the group are observed with specific patterns that appear based on the characteristics of DST-7 kernels. Figure 10 shows the patterns of the elements in consecutive rows. As shown in Figure 10, elements in the odd column (1st, 3rd, ...) of the 0th row ($[S_{7,(0,0)}]_{32}$) are repeated in the left-half column in the next row. The elements in the right-half column are copied from the elements of even columns (0th, 2nd, ...) in the previous row vector in reverse order.

$$\begin{aligned}
[S_{7,(0,0)}]_{16} &= [0.033 \ 0.066 \ 0.098 \ 0.129 \ 0.160 \ 0.188 \ 0.215 \ 0.240 \ 0.263 \ 0.284 \ 0.302 \ 0.317 \ 0.329 \ 0.338 \ 0.345 \ 0.348] \\
[S_{7,(0,15)}]_{16} &= [0.066 \ -0.129 \ 0.188 \ -0.240 \ 0.284 \ -0.317 \ 0.338 \ -0.348 \ 0.345 \ -0.329 \ 0.302 \ -0.263 \ 0.215 \ -0.160 \ 0.098 \ -0.033] \\
\\
[S_{7,(0,0)}]_{16} &= [0.033 \ 0.066 \ 0.098 \ 0.129 \ 0.160 \ 0.188 \ 0.215 \ 0.240 \ 0.263 \ 0.284 \ 0.302 \ 0.317 \ 0.329 \ 0.338 \ 0.345 \ 0.348] \\
[S_{7,(0,15)}]_{16} &= [0.066 \ -0.129 \ 0.188 \ -0.240 \ 0.284 \ -0.317 \ 0.338 \ -0.348 \ 0.345 \ -0.329 \ 0.302 \ -0.263 \ 0.215 \ -0.160 \ 0.098 \ -0.033]
\end{aligned}$$

Figure 10. An example of patterns observed between two consecutive row vectors of 16-point DST-7 transform kernel.

Thus, this relation can be described in Equation (21).

$$[S_{7,(i,1)}]_N = [(S_{7,(i,1)})_{N/2}]_L' [(S_{7,(i,1)})_{N/2}]_R' \quad (21)$$

Based on Figure 7, (21) is the next row vector, composed of two parts, $([S_{7,(i,1)}]_{N/2})_L$ and $([S_{7,(i,1)}]_{N/2})_R$. $([S_{7,(i,1)}]_{N/2})_L$ and $([S_{7,(i,1)}]_{N/2})_R$ represent the left and right halves of $[S_{7,(i,1)}]_N$.

$([S_{7,(i,1)}]_{N/2})_L$ is derived using (22), where elements in the odd column (1st, 3rd, ...) of the 0th row (previous row) are selected. $([S_{7,(i,1)}]_{N/2})_R$ is derived using (23), where elements in the even column (0th, 2nd, ...) of the 0th row (previous row) are selected and substituted in a reverse way. Note that $[S_{7,(i,j),k}]_{N/2}$ denotes the element in the k -th column and j -th row vector of the i -th group for N -point DST-7.

$$([S_{7,(i,1)}]_{N/2})_L = [S_{7,(i,0),2n-1}]_{N/2} \quad (22)$$

$$([S_{7,(i,1)}]_{N/2})_R = [S_{7,(i,0),2n-2}]_{N/2} \quad (23)$$

where n is the integer value, $N/2 < n > 0$.

Hence, a change of one element in the row vectors in a group may affect the other elements of the other vector of the same group. Figure 10 shows an example of patterns observed between two consecutive row vectors of the 16-point DST-7 transform kernel. From the patterns, $\mathbf{G}$ comprising -1 , 0 , and 1 can be derived. Figure 11 shows the $\mathbf{G}$ matrix based on the pattern in Figure 10 for deriving the DST-7 transform kernel.

Figure 11. An example of 32-point permutation matrix ($\mathbf{G}$).

As shown in Figure 11, a smaller $\mathbf{G}$ can be embedded into a larger $\mathbf{G}$ with identical patterns of element values. Figure 12 shows an algorithm for obtaining the matrix. Even if the memory size of $\mathbf{G}$ in Figure 11 accounts for 256 bytes, there can be a case to derive $\mathbf{G}$ on the fly using the algorithm shown in Figure 12, not using static memory to save $\mathbf{G}$.

Figure 12. Algorithm for the generation of the permutation matrix G .

As shown in Figure 12, the algorithm comprises two parts for the left column (S0) and right column part (S1) of G . First, variables used for the algorithm are initialized and the size (N) of the kernel matrix to derive is set. Then, variables a and b are initially set to 1 and 0, which will be used as initial values of the left column (S0) part of G . i_r and i_c represent the row and column indices of G , respectively. The index value of i_r was initialized to 0. First, the condition is checked if the i_r index value is less than the size N . If this is true, then the process is divided into two columns, the left column (S0) and right column (S1); then, the i_c index value is initialized to 0 else, the process terminates.

Thus, the i_c index value is checked if it is less than half of the value of N . If the condition defined is satisfied, then i_r and i_c corresponding to the respective a and b variables are checked. Variables a and b are set only for the S0 of the G matrix. The value of the permutation matrix at the respective location $[G_N]_{i_r, i_c}$ is set to 0 if the checked condition is false, i_c is incremented by 1, and looped back to the condition, where the i_c index value is checked for whether it is less than half of N . If the checked condition holds true (T), c_1 is checked for whether it is true (T) or false (F). If c_1 is false, then the value $[G_N]_{i_r, i_c}$ is set to -1 , and c_1 is set to true. Otherwise, the value of $[G_N]_{i_r, i_c}$ is set to 1, and c_1 is set to false. Subsequently, the value of a is incremented by 2, and the value of b is incremented by 1. After setting the values of a and b , i_c is incremented by 1 and finally looped back to the condition where the i_c index value is checked whether it is less than half of the value of N .

For the right column (S1) part of G , variables m and n are used, whose initial values are set to 0 and $N-1$, respectively. If the condition, where i_c index value is checked to be less than half of N is false, then the value of i_c is set as half of N . Further, the i_c index is checked to determine whether it is less than N . If the condition is false (F), then the i_r index value is incremented by 1 and looped back to the condition, where i_r is less than N or is not

checked. However, if the condition exists (T), then the i_r index equals m and the i_c index, which equals n , is checked. If it is false, the value of $[G_N]_{i_r, i_c}$ is set to 0. Otherwise, the c_2 condition is further checked for whether it is false. If c_2 is false, the value $[G_N]_{i_r, i_c}$ is set to 1, and c_2 is set as false; thus, the i_c index is incremented by 1 and looped back to the condition where i_c is checked to determine whether it is less than N . Otherwise, the value $[G_N]_{i_r, i_c}$ is set as -1 and c_2 is set as true, the value of m is incremented by 2, and that of n is decremented by 1. Finally, the value of the i_c index is incremented by 1 and looped back to the condition where the i_c index is checked to determine whether it is less than N .

An example of the splitting of S_0 and S_1 in the G matrix for $N = 8$ is shown in Figure 13.

S0				S1			
0	0	0	0	0	0	0	-1
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	-1	0	0	0	0	0	0
0	0	0	0	0	-1	0	0
0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	-1	0	0	0	0

Figure 13. An example of resulting G matrix for $N = 8$ by Figure 12.

2.4. Integer Kernels with VVC Standard

Generally, the floating-point elements in the kernels are scaled up to integer elements to avoid a mismatch between the encoder and decoder in most video coding standards. Transform and quantization using resulting integer kernel matrices are implemented with integer arithmetic, which benefits the complexity of implementations and is one of the primary concerns in the standardization of VVC. Despite the benefits of integer kernels, several disadvantages exist. Orthogonality in the kernel matrix can be lost when scaled up to integer values and the magnitudes of the matrix elements should be as small as possible under the consideration of dynamic ranges for the process of transform and quantization [26]. Nevertheless, it is known that the kernel similarity between the derived kernel and original kernels is more important than orthogonality when quantization is applied to transform the coefficients. Thus, it is unnecessary to maintain kernel orthogonality at the expense of kernel similarity. By reflecting the requirements of transform kernels for video codec, integer kernels for every transform block are proposed and finally adopted [15]. In this section, we show how our proposed kernel derivation method is applied for transformation with no deviation, which means the kernels derived by the proposed method are perfectly matched with the standard kernels.

Figure 14 shows a 32-point integer DST-7 transform kernel adopted in VTM-3.0 [27] as the standard. In Figure 14, each row vector is shown in the order of the group. The element values in the row vectors were identical to those of the standard kernel.

As described in Figure 7 of Section 2.2, the absolute value of e_f in the current row should be identical to the absolute value of e_l of the next row, which is obtained by multiplying the current row vector by G . The multiplication can be repeated until e_l of the base (first) row is equal to e_f of the row multiplied by G . For the derivation of all sizes of DST-7, the multiplications are repeated until the end of the last vectors of the group, except for a few cases for 32-point DST-7, which are observed in the 3rd group for 32-point DST-7. Figure 15 shows the case shown in the 3rd group for the 32-point DST-7.

$[S_{7,(0,0)}]_{32} = [4 \ 9 \ 13 \ 17 \ 21 \ 26 \ 30 \ 34 \ 38 \ 42 \ 45 \ 50 \ 53 \ 56 \ 60 \ 63 \ 66 \ 68 \ 72 \ 74 \ 77 \ 78 \ 80 \ 82 \ 84 \ 85 \ 86 \ 88 \ 88 \ 89 \ 90 \ 90]$
 $[S_{7,(0,31)}]_{32} = [9 \ -17 \ 26 \ -34 \ 42 \ -50 \ 56 \ -63 \ 68 \ -74 \ 78 \ -82 \ 85 \ -88 \ 89 \ -90 \ 90 \ -88 \ 86 \ -84 \ 80 \ -77 \ 72 \ -66 \ 60 \ -53 \ 45 \ -38 \ 30 \ -21 \ 13 \ -4]$
 $[S_{7,(0,30)}]_{32} = [17 \ -34 \ 50 \ -63 \ 74 \ -82 \ 88 \ -90 \ 88 \ -84 \ 77 \ -66 \ 53 \ -38 \ 21 \ -4 \ 13 \ 30 \ -45 \ 60 \ -72 \ 80 \ -86 \ 90 \ -89 \ 85 \ -78 \ 68 \ -56 \ 42 \ -26 \ 9]$
 $[S_{7,(0,28)}]_{32} = [34 \ -63 \ 82 \ -90 \ 84 \ -66 \ 38 \ -4 \ 30 \ 60 \ -80 \ 90 \ -85 \ 68 \ -42 \ 9 \ 26 \ -56 \ 78 \ -89 \ 86 \ -72 \ 45 \ -13 \ 21 \ 53 \ -77 \ 88 \ -88 \ 74 \ -50 \ 17]$
 $[S_{7,(0,24)}]_{32} = [63 \ -90 \ 66 \ -4 \ 60 \ 90 \ -68 \ 9 \ 56 \ -89 \ 72 \ -13 \ 53 \ 88 \ -74 \ 17 \ 50 \ -88 \ 77 \ -21 \ 45 \ 86 \ -78 \ 26 \ 42 \ -85 \ 80 \ -30 \ 38 \ 84 \ -82 \ 34]$
 $[S_{7,(0,16)}]_{32} = [90 \ -4 \ 90 \ 9 \ 89 \ -13 \ 88 \ 17 \ 88 \ -21 \ 86 \ 26 \ 85 \ -30 \ 84 \ 34 \ 82 \ -38 \ 80 \ 42 \ 78 \ -45 \ 77 \ 50 \ 74 \ -53 \ 72 \ 56 \ 68 \ -60 \ 66 \ 63]$

 $[S_{7,(1,1)}]_{32} = [13 \ 26 \ 38 \ 50 \ 60 \ 68 \ 77 \ 82 \ 86 \ 89 \ 90 \ 88 \ 85 \ 80 \ 74 \ 66 \ 56 \ 45 \ 34 \ 21 \ 9 \ -4 \ -17 \ -30 \ -42 \ -53 \ -63 \ -72 \ -78 \ -84 \ -88 \ -90]$
 $[S_{7,(1,29)}]_{32} = [26 \ -50 \ 68 \ -82 \ 89 \ -88 \ 80 \ -66 \ 45 \ -21 \ -4 \ 30 \ -53 \ 72 \ -84 \ 90 \ -88 \ 78 \ -63 \ 42 \ -17 \ -9 \ 34 \ -56 \ 74 \ -85 \ 90 \ -86 \ 77 \ -60 \ 38 \ -13]$
 $[S_{7,(1,26)}]_{32} = [50 \ -82 \ 88 \ -66 \ 21 \ 30 \ -72 \ 90 \ -78 \ 42 \ 9 \ 56 \ 85 \ -86 \ 60 \ -13 \ 38 \ 77 \ -90 \ 74 \ -34 \ -17 \ 63 \ -88 \ 84 \ -53 \ 4 \ 45 \ -80 \ 89 \ -68 \ 26]$
 $[S_{7,(1,20)}]_{32} = [82 \ -66 \ 30 \ 90 \ -42 \ 56 \ 86 \ -13 \ 77 \ 74 \ 17 \ -88 \ 53 \ 45 \ -89 \ 26 \ 68 \ -80 \ -4 \ 84 \ -63 \ 34 \ 90 \ -38 \ -60 \ 85 \ -9 \ 78 \ 72 \ 21 \ -88 \ 50]$
 $[S_{7,(1,8)}]_{32} = [66 \ 90 \ 56 \ -13 \ 74 \ -88 \ 45 \ 26 \ 80 \ 84 \ 34 \ -38 \ 85 \ 78 \ -21 \ 50 \ 88 \ 72 \ 9 \ -60 \ 90 \ -63 \ 4 \ 68 \ 89 \ 53 \ -17 \ 77 \ -86 \ 42 \ 30 \ 82]$
 $[S_{7,(1,15)}]_{32} = [90 \ 13 \ -88 \ -26 \ 84 \ 38 \ -78 \ 50 \ 72 \ 60 \ -63 \ 68 \ 53 \ 77 \ -42 \ 82 \ 30 \ 86 \ -17 \ 89 \ 4 \ 90 \ 9 \ -88 \ -21 \ 85 \ 34 \ -80 \ 45 \ 74 \ 56 \ -66]$

 $[S_{7,(2,2)}]_{32} = [21 \ 42 \ 60 \ 74 \ 84 \ 89 \ 89 \ 84 \ 74 \ 60 \ 42 \ 21 \ 0 \ -21 \ -42 \ -60 \ -74 \ -84 \ -89 \ -89 \ -84 \ -74 \ -60 \ 42 \ 21 \ 0 \ 21 \ 42 \ 60 \ 74 \ 84 \ 89]$
 $[S_{7,(2,27)}]_{32} = [42 \ -74 \ 89 \ -84 \ 60 \ -21 \ 21 \ 60 \ -84 \ 89 \ -74 \ 42 \ 0 \ -42 \ 74 \ -89 \ 84 \ 60 \ 21 \ 21 \ -60 \ 84 \ -89 \ 74 \ 42 \ 0 \ 42 \ -74 \ 89 \ -84 \ 60 \ -21]$
 $[S_{7,(2,22)}]_{32} = [74 \ -84 \ 21 \ 60 \ -89 \ 42 \ 42 \ -89 \ 60 \ 21 \ -84 \ 74 \ 0 \ -74 \ 84 \ 21 \ -60 \ 89 \ -42 \ 42 \ 89 \ -60 \ 21 \ 84 \ 74 \ 0 \ 74 \ -84 \ 21 \ 60 \ -89 \ 42]$
 $[S_{7,(2,12)}]_{32} = [84 \ 60 \ -42 \ 89 \ -21 \ 74 \ 74 \ -21 \ 89 \ 42 \ 60 \ 84 \ 0 \ -84 \ 60 \ 42 \ 89 \ 21 \ -74 \ 74 \ 21 \ 89 \ 42 \ -60 \ 84 \ 0 \ 84 \ 60 \ -42 \ 89 \ -21 \ 74]$
 $[S_{7,(2,7)}]_{32} = [60 \ 89 \ 74 \ 21 \ -42 \ 84 \ 84 \ 42 \ 21 \ 74 \ 89 \ 60 \ 0 \ -60 \ 89 \ 74 \ 21 \ 42 \ 84 \ 84 \ 42 \ -21 \ 74 \ 89 \ 60 \ 0 \ 60 \ 89 \ 74 \ 21 \ -42 \ 84]$
 $[S_{7,(2,17)}]_{32} = [89 \ -21 \ 84 \ 42 \ 74 \ -60 \ 60 \ 74 \ 42 \ -84 \ 21 \ 89 \ 0 \ -89 \ 21 \ 84 \ 42 \ 74 \ 60 \ 74 \ 42 \ 84 \ 21 \ 89 \ 0 \ 89 \ -21 \ 84 \ 42 \ 74 \ -60]$

 $[S_{7,(3,3)}]_{32} = [30 \ 56 \ 77 \ 88 \ 89 \ 80 \ 63 \ 38 \ 9 \ -21 \ -50 \ -72 \ -85 \ -90 \ -84 \ -68 \ -45 \ 17 \ 13 \ 42 \ 66 \ 82 \ 90 \ 86 \ 74 \ 53 \ 26 \ -4 \ -34 \ -60 \ -78 \ -88]$
 $[S_{7,(3,25)}]_{32} = [56 \ -88 \ 80 \ -38 \ -21 \ 72 \ -90 \ 68 \ -17 \ 42 \ 82 \ -86 \ 53 \ 4 \ -60 \ 88 \ -78 \ 34 \ 26 \ -74 \ 90 \ -66 \ 13 \ 45 \ -84 \ 85 \ -50 \ -9 \ 63 \ -89 \ 77 \ -30]$
 $[S_{7,(3,18)}]_{32} = [88 \ -38 \ 72 \ 68 \ 42 \ -86 \ -4 \ 88 \ -34 \ 74 \ 66 \ 45 \ -85 \ -9 \ 89 \ -30 \ 77 \ 63 \ 50 \ -84 \ 13 \ 90 \ -26 \ 78 \ 60 \ 53 \ -82 \ 17 \ 90 \ -21 \ 80 \ 56]$
 $[S_{7,(3,4)}]_{32} = [38 \ 68 \ 86 \ 88 \ 74 \ 45 \ 9 \ -30 \ -63 \ 84 \ -90 \ 78 \ -53 \ 17 \ 21 \ 56 \ 80 \ 90 \ 82 \ 60 \ 26 \ -13 \ 50 \ 77 \ 89 \ -85 \ 66 \ 34 \ 4 \ 42 \ 72 \ 88]$
 $[S_{7,(3,23)}]_{32} = [68 \ -88 \ 45 \ 30 \ -84 \ 78 \ -17 \ 56 \ 90 \ -60 \ 13 \ 77 \ -85 \ 34 \ 42 \ -88 \ 72 \ -4 \ 66 \ 89 \ -50 \ 26 \ 82 \ 80 \ 21 \ 53 \ -90 \ 63 \ 9 \ 74 \ 86 \ -38]$
 $[S_{7,(3,14)}]_{32} = [88 \ 30 \ 78 \ -56 \ 60 \ 77 \ -34 \ 88 \ 4 \ 89 \ 26 \ -80 \ 53 \ 63 \ 74 \ -38 \ 86 \ 9 \ 90 \ 21 \ -82 \ 50 \ 66 \ 72 \ 42 \ -85 \ 13 \ 90 \ 17 \ -84 \ 45 \ 68]$

 $[S_{7,(5,5)}]_{32} = [45 \ 78 \ 90 \ 77 \ 42 \ -4 \ -50 \ -80 \ 90 \ 74 \ 38 \ 9 \ 53 \ 82 \ 89 \ 72 \ 34 \ 13 \ 56 \ 84 \ 88 \ 68 \ 30 \ 17 \ 60 \ 85 \ 88 \ 66 \ 26 \ -21 \ 63 \ 86]$
 $[S_{7,(5,21)}]_{32} = [78 \ -77 \ -4 \ 80 \ 74 \ -9 \ 82 \ -72 \ 13 \ 84 \ 68 \ 17 \ 85 \ 66 \ 21 \ 86 \ 63 \ 26 \ 88 \ 60 \ 30 \ 88 \ 56 \ 34 \ 89 \ 53 \ 38 \ 90 \ 50 \ 42 \ 90 \ 45]$
 $[S_{7,(5,10)}]_{32} = [77 \ 80 \ 9 \ 72 \ 84 \ 17 \ 66 \ 86 \ 26 \ -60 \ 88 \ 34 \ 53 \ 90 \ 42 \ 45 \ 90 \ 50 \ 38 \ 89 \ 56 \ 30 \ 88 \ 63 \ 21 \ 85 \ 68 \ 13 \ 82 \ 74 \ 4 \ 78]$
 $[S_{7,(5,11)}]_{32} = [80 \ 72 \ 17 \ 86 \ 60 \ 34 \ 90 \ 45 \ 50 \ 89 \ 30 \ 63 \ 85 \ 13 \ 74 \ 78 \ 4 \ 82 \ 68 \ 21 \ 88 \ 56 \ 38 \ 90 \ 42 \ 53 \ 88 \ 26 \ 66 \ 84 \ 9 \ 77]$
 $[S_{7,(5,9)}]_{32} = [72 \ 86 \ 34 \ 45 \ 89 \ 63 \ 13 \ 78 \ 82 \ 21 \ 56 \ 90 \ 53 \ 26 \ 84 \ 77 \ 9 \ 66 \ 88 \ 42 \ 38 \ 88 \ 68 \ -4 \ 74 \ 85 \ 30 \ 50 \ 90 \ 60 \ 17 \ 80]$
 $[S_{7,(5,13)}]_{32} = [86 \ 45 \ 63 \ 78 \ 21 \ 90 \ 26 \ 77 \ 66 \ 42 \ 88 \ 4 \ 85 \ 50 \ 60 \ 80 \ 17 \ 90 \ 30 \ 74 \ 68 \ 38 \ 88 \ -9 \ 84 \ 53 \ 56 \ 82 \ 13 \ 89 \ 34 \ 72]$

 $[S_{7,(6,6)}]_{32} = [53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85 \ 85 \ 53 \ 0 \ 53 \ 85]$
 $[S_{7,(6,19)}]_{32} = [85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53 \ 53 \ 85 \ 0 \ 85 \ 53]$

Figure 14. Grouped 32-point DST-7 transform kernel elements.

$[S_{7,(3,3)}]_{32} = [30 \ 56 \ 77 \ 88 \ 89 \ 80 \ 63 \ 38 \ 9 \ -21 \ -50 \ -72 \ -85 \ -90 \ -84 \ -68 \ -45 \ 17 \ 13 \ 42 \ 66 \ 82 \ 90 \ 86 \ 74 \ 53 \ 26 \ -4 \ -34 \ -60 \ -78 \ -88]$
 $[S_{7,(3,25)}]_{32} = [56 \ -88 \ 80 \ -38 \ -21 \ 72 \ -90 \ 68 \ -17 \ 42 \ 82 \ -86 \ 53 \ 4 \ -60 \ 88 \ -78 \ 34 \ 26 \ -74 \ 90 \ -66 \ 13 \ 45 \ -84 \ 85 \ -50 \ -9 \ 63 \ -89 \ 77 \ -30]$
 $[S_{7,(3,18)}]_{32} = [88 \ -38 \ 72 \ 68 \ 42 \ -86 \ -4 \ 88 \ -34 \ 74 \ 66 \ 45 \ -85 \ -9 \ 89 \ -30 \ 77 \ 63 \ 50 \ -84 \ 13 \ 90 \ -26 \ 78 \ 60 \ 53 \ -82 \ 17 \ 90 \ -21 \ 80 \ 56]$
 $[S_{7,(3,4)}]_{32} = [38 \ 68 \ 86 \ 88 \ 74 \ 45 \ 9 \ -30 \ -63 \ 84 \ -90 \ 78 \ -53 \ 17 \ 21 \ 56 \ 80 \ 90 \ 82 \ 60 \ 26 \ -13 \ 50 \ 77 \ 89 \ -85 \ 66 \ 34 \ 4 \ 42 \ 72 \ 88]$
 $[S_{7,(3,23)}]_{32} = [68 \ -88 \ 45 \ 30 \ -84 \ 78 \ -17 \ 56 \ 90 \ -60 \ 13 \ 77 \ -85 \ 34 \ 42 \ -88 \ 72 \ -4 \ 66 \ 89 \ -50 \ 26 \ 82 \ 80 \ 21 \ 53 \ -90 \ 63 \ 9 \ 74 \ 86 \ -38]$
 $[S_{7,(3,14)}]_{32} = [88 \ 30 \ 78 \ -56 \ 60 \ 77 \ -34 \ 88 \ 4 \ 89 \ 26 \ -80 \ 53 \ 63 \ 74 \ -38 \ 86 \ 9 \ 90 \ 21 \ -82 \ 50 \ 66 \ 72 \ 42 \ -85 \ 13 \ 90 \ 17 \ -84 \ 45 \ 68]$

Figure 15. An example of the case not deriving the DST-7 kernel.

As shown in Figure 15, $[S_{7,(3,18)}]_{32}$ is obtained after the multiplication of $[S_{7,(3,25)}]_{32}$ and $\mathbf{G}$. Because e_l of $[S_{7,(3,3)}]_{32}$ equals e_f of $[S_{7,(3,18)}]_{32}$, the algorithm for the current group is complete, as shown in Figure 7. Thus, $[S_{7,(3,4)}]_{32}$, $[S_{7,(3,23)}]_{32}$ and $[S_{7,(3,14)}]_{32}$ for the full-size DST-7 kernel cannot be obtained using the proposed algorithm. This only happens when the element values are scaled up to make the element values of the kernel integer. To address this problem for integer kernels of DST-7, especially for the 32-point kernel, two approaches are proposed in our method. The first method is to tune e_l of $[S_{7,(3,3)}]_{32}$ from 88 to 89 without considering the sign value, and by tuning the value, the case of stopping the algorithm in $[S_{7,(3,18)}]_{32}$ obtained after the multiplication of $[S_{7,(3,25)}]_{32}$ and $\mathbf{G}$ can be removed, resulting in a full-size 32-point DST-7 transform kernel. Figure 16a shows this procedure. Similarly, the second method is to tune the 3rd element of the value of $[S_{7,(3,3)}]_{32}$ from 88 to 89 without considering the sign value, as shown in Figure 16b. By tuning the values, the related values in the other vectors are changed by the proposed algorithm, accordingly.

$$\begin{aligned}
[S_{7,(3,3)}]_{32} &= [30\ 56\ 77\ 88\ 89\ 80\ 63\ 38\ 9\ -21\ -50\ -72\ -85\ -90\ -84\ -68\ -45\ -17\ 13\ 42\ 66\ 82\ 90\ 86\ 74\ 53\ 26\ -4\ -34\ -60\ -78\ -89] \\
[S_{7,(3,25)}]_{32} &= [56\ -88\ 80\ -38\ -21\ 72\ -90\ 68\ -17\ -42\ 82\ -86\ 53\ 4\ -60\ 89\ -78\ 34\ 26\ -74\ 90\ -66\ 13\ 45\ -84\ 85\ -50\ -9\ 63\ -89\ 77\ -30] \\
[S_{7,(3,18)}]_{32} &= [88\ -38\ -72\ 68\ 42\ -86\ -4\ 89\ -34\ -74\ 66\ 45\ -85\ -9\ 89\ -30\ -77\ 63\ 50\ -84\ -13\ 90\ -26\ -78\ 60\ 53\ -82\ -17\ 90\ -21\ -80\ 56] \\
[S_{7,(3,4)}]_{32} &= [38\ 68\ 86\ 89\ 74\ 45\ 9\ -30\ -63\ -84\ -90\ -78\ -53\ -17\ 21\ 56\ 80\ 90\ 82\ 60\ 26\ -13\ -50\ -77\ -89\ -85\ -66\ -34\ 4\ 42\ 72\ 88] \\
[S_{7,(3,23)}]_{32} &= [68\ -89\ 45\ 30\ -84\ 78\ -17\ -56\ 90\ -60\ -13\ 77\ -85\ 34\ 42\ -88\ 72\ -4\ -66\ 89\ -50\ -26\ 82\ -80\ 21\ 53\ -90\ 63\ 9\ -74\ 86\ -38] \\
[S_{7,(3,14)}]_{32} &= [89\ 30\ -78\ -56\ 60\ 77\ -34\ -88\ 4\ 89\ 26\ -80\ -53\ 63\ 74\ -38\ -86\ 9\ 90\ 21\ -82\ -50\ 66\ 72\ -42\ -85\ 13\ 90\ 17\ -84\ -45\ 68]
\end{aligned}$$

(a)

$$\begin{aligned}
[S_{7,(3,3)}]_{32} &= [30\ 56\ 77\ 89\ 89\ 80\ 63\ 38\ 9\ -21\ -50\ -72\ -85\ -90\ -84\ -68\ -45\ -17\ 13\ 42\ 66\ 82\ 90\ 86\ 74\ 53\ 26\ -4\ -34\ -60\ -78\ -88] \\
[S_{7,(3,25)}]_{32} &= [56\ -89\ 80\ -38\ -21\ 72\ -90\ 68\ -17\ -42\ 82\ -86\ 53\ 4\ -60\ 88\ -78\ 34\ 26\ -74\ 90\ -66\ 13\ 45\ -84\ 85\ -50\ -9\ 63\ -89\ 77\ -30] \\
[S_{7,(3,18)}]_{32} &= [89\ -38\ -72\ 68\ 42\ -86\ -4\ 88\ -34\ -74\ 66\ 45\ -85\ -9\ 89\ -30\ -77\ 63\ 50\ -84\ -13\ 90\ -26\ -78\ 60\ 53\ -82\ -17\ 90\ -21\ -80\ 56] \\
[S_{7,(3,4)}]_{32} &= [38\ 68\ 86\ 88\ 74\ 45\ 9\ -30\ -63\ -84\ -90\ -78\ -53\ -17\ 21\ 56\ 80\ 90\ 82\ 60\ 26\ -13\ -50\ -77\ -89\ -85\ -66\ -34\ 4\ 42\ 72\ 89] \\
[S_{7,(3,23)}]_{32} &= [68\ -88\ 45\ 30\ -84\ 78\ -17\ -56\ 90\ -60\ -13\ 77\ -85\ 34\ 42\ -89\ 72\ -4\ -66\ 89\ -50\ -26\ 82\ -80\ 21\ 53\ -90\ 63\ 9\ -74\ 86\ -38] \\
[S_{7,(3,14)}]_{32} &= [88\ 30\ -78\ -56\ 60\ 77\ -34\ -89\ 4\ 89\ 26\ -80\ -53\ 63\ 74\ -38\ -86\ 9\ 90\ 21\ -82\ -50\ 66\ 72\ -42\ -85\ 13\ 90\ 17\ -84\ -45\ 68]
\end{aligned}$$

(b)

Figure 16. Removing the case not obtaining full kernel matrix by tuning values highlighted. (a) Method 1; (b) Method 2.

Finally, once all sizes of DST-7 kernels are derived, DCT-7 and all DCT-8 transform kernels can be easily derived using relation Equation (24) as

$$\mathbf{C}_8 = \mathbf{S} \times \mathbf{S}_7 \times \mathbf{F} \quad (24)$$

where $\mathbf{C}_8$ and $\mathbf{S}_7$ are the DCT-8 and DST-7 transform kernels, respectively, and $\mathbf{S}$ and $\mathbf{F}$ are the sign-changing and flipping matrices, defined in (7) and (8), respectively.

2.5. Application of the Proposed Method to the DST-7/DCT-8 Fast Algorithm in VTM

In Equation (23), a fast DST-7/DCT-8 algorithm with dual support was introduced and adopted in VTM-3.0. The fast algorithm is based on three different features for integer N -point DST-7/DCT-8 transform kernels shown in Figure 17, which can be defined as Figure 17.

```

{ a b c d e f g h i j k l m n o p }
{ c f i l o o l i f c 0 -c -f -i -l -o }
{ e j o m h c -b -g -l -p -k -f -a d i n }
{ g n l e -b -i -p -j -c d k o h a -f -m }
{ i o f -c -l -l -c f o i 0 -i -o -f c l }
{ k k 0 -k -k 0 k k 0 -k -k 0 k k 0 -k }
{ m g -f -n -a l h -e -o -b k i -d -p -c j }
{ o c -l -f i i -f -l c o 0 -o -c l f -i }
{ p -a -o b n -c -m d l -e -k f j -g -i h }
{ n -e -i j d -o a m -f -h k c -p b l -g }
{ l -i -c o -f -f o -c -i l 0 -l i c -o f }
{ j -m c g -p f d -n i a -k l -b -h o -e }
{ h -p i -a -g o -j b f -n k -c -e m -l d }
{ f -l o -i c c -i o -l f 0 -f l -o i -c }
{ d -h l -p m -i e -a -c g -k o -n j -f b }
{ b -d f -h j -l n -p o -m k -i g -e c -a }

```

Figure 17. Illustration of 16-point DST-7 kernel unique integer numbers [28].

Where $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p\}$ are the N -point unique integer numbers for the 16-point DST-7 kernel. Assume that the output vector $\mathbf{y} = [y_0, y_1, \dots, y_{N-1}]^T$ is

obtained by multiplying the input vector $\mathbf{x} = [x_0, x_1, \dots, x_{N-1}]^T$ with the transform kernel $\mathbf{C}$, that is, $(\mathbf{y} = \mathbf{x}\mathbf{C})$.

Feature #1: It is observed that the sum of several (2 or 3) numbers is equal to the sum of several (1 or 2) numbers. The elements l, m, n, o , and p are obtained by $(a + j)$, $(b + i)$, $(c + h)$, $(d + g)$, and $(e + f)$, respectively.

To calculate y_0 , instead of performing vector-by-vector multiplication that requires 16 multiplications, the following alternative implementation is computed as shown in Equation (25):

$$y_0 = a \cdot (x_0 + x_{11}) + b \cdot (x_1 + x_{12}) + \dots + j \cdot (x_9 + x_{11}) + k \cdot x_{10} \quad (25)$$

which requires 10 multiplications.

Feature #2: This is based on the feature that some elements in a row vector of DST-7 are symmetrically mirrored with each other. For example, the second vector $\{c, f, i, l, o, o, l, i, f, c, o, -c, -f, -i, -l, -o\}$ in Figure xx with only c, f, i, l , and o elements.

To calculate y_1 , instead of performing vector-by-vector multiplication that requires 16 multiplications, the following alternative implementation is computed, as shown in Equation (26):

$$y_1 = c \cdot (x_0 + x_9 - x_{11}) + f \cdot (x_1 + x_8 - x_{12}) + i \cdot (x_2 + x_7 - x_{13}) + l \cdot (x_3 + x_6 - x_{14}) + o \cdot (x_4 + x_5 - x_{15}) \quad (26)$$

which requires five multiplications.

Feature #3: This is based on the basic vectors that contain very few numbers (1 or 2) of distinct values without considering sign values $\{k, k, 0, -k, -k, 0, k, k, 0, -k, -k, 0, k, k, 0, -k\}$. To calculate y_5 , instead of performing vector-by-vector multiplication that requires 16 multiplications, the following alternative implementation is computed as shown in Equation (27):

$$y_5 = k \cdot (x_0 + x_1 - \dots - x_{15}) \quad (27)$$

which requires one multiplication.

The major drawback of the fast algorithm [28] for DST-7 transform in VTM3.0 is that one feature out of the mentioned three features has to be applied in each row. However, once a feature is selected for the basic vector such as $[S_{7,(0,0)}]_{16}$, $[S_{7,(1,1)}]_{16}$, $[S_{7,(2,2)}]_{16}$ and $[S_{7,(5,5)}]_{16}$, the same feature is applied to vectors belonging to the same group. Hence, the proposed method follows “one group one feature” and removes the matching of a single feature among the three features in each row.

For example, in the first row, a vector of DST-7, Feature #1 is applied, whereas in the second-row vector, Feature #2 is applied. Similarly, for the other rows, a single feature out of three features was applied. Initially, before applying the feature in each row vector, we were unaware of the feature to be applied in the respective row vector. Consequently, we need to check if the respective row vector follows Feature #1, #2, or #3 such that the single matched feature is applied. For a larger matrix, the number of row vectors is large; therefore, for each row, we have to manually check if each row follows Feature #1, #2, or #3, which is time-consuming.

Based on the proposed grouping method, only for the base row vector, a feature out of three features is selected manually, and the row vectors that depend on the base row, that is, each grouped row vector, follow the same feature that is applied to the based row vector. Figure 18 shows the application of the proposed 16-point grouped DST-7 transform to the fast algorithm in VVC. Feature 1 can be applied to the 0th and 2nd groups. The elements in the l, m, n, o , and p columns are obtained by the summation of the column elements such as $(a + j)$, $(b + i)$, $(c + h)$, $(d + g)$, and $(e + f)$, respectively, of the 0th and 2nd groups. Feature #2 is applied to the 1-st group where column elements in f, g, h, i , and j are mirrored values

of column elements e, d, c, b , and a , respectively. Similarly, elements in l, m, n, o , and p are mirrored in the column elements j, i, h, g , and f with flipped signed values, respectively. Similarly, Feature #3 is applied to $[S_{7,(5,5)}]_{16}$ as it contains only a single distinct element, that is, 77. This is possible with the help of the proposed grouping method.

Feature #1															
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p
$[S_{7,(0,0)}]_{16} =$	9	17	25	33	41	49	56	62	66	72	77	81	83	87	90
$[S_{7,(0,15)}]_{16} =$	17	-33	49	-62	72	-81	87	-90	89	-83	77	-66	56	-41	25
$[S_{7,(0,14)}]_{16} =$	33	-62	81	-90	83	-66	41	-9	-25	56	-77	89	-87	72	-49
$[S_{7,(0,12)}]_{16} =$	62	-90	66	-9	-56	89	-72	17	49	-87	77	-25	-41	83	-33
$[S_{7,(0,8)}]_{16} =$	90	-9	-89	17	87	-25	-83	33	81	-41	-77	49	72	-56	-62

Feature #2															
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p
$[S_{7,(1,1)}]_{16} =$	25	49	66	81	89	89	81	66	49	25	0	-25	-49	-66	-81
$[S_{7,(1,13)}]_{16} =$	49	-81	89	-66	25	25	-66	89	-81	49	0	-49	81	-89	66
$[S_{7,(1,10)}]_{16} =$	81	-66	-25	89	-49	-49	89	-25	-66	81	0	-81	66	25	-89
$[S_{7,(1,4)}]_{16} =$	66	89	49	-25	-81	-81	-25	49	89	66	0	-66	-89	-49	25
$[S_{7,(1,7)}]_{16} =$	89	25	-81	-49	66	66	-49	-81	25	89	0	-89	-25	81	-49

Feature #1															
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p
$[S_{7,(2,2)}]_{16} =$	41	72	89	83	62	25	-17	-56	-81	-90	-77	-49	-9	33	66
$[S_{7,(2,11)}]_{16} =$	72	-83	25	56	-90	49	33	-87	66	9	-77	81	-17	-62	89
$[S_{7,(2,6)}]_{16} =$	83	56	-49	-87	-9	81	62	-41	-89	-17	77	66	-33	-90	-25
$[S_{7,(2,3)}]_{16} =$	56	87	81	41	-17	-66	-90	-72	-25	33	77	89	62	9	-49
$[S_{7,(2,9)}]_{16} =$	87	-41	-66	72	33	-89	9	83	-49	-62	77	25	-90	17	81

Feature #3															
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p
$[S_{7,(5,5)}]_{16} =$	77	77	0	-77	-77	0	77	77	0	-77	-77	0	77	77	0

Figure 18. Application of the proposed 16-point grouped DST-7 transform to a fast algorithm in VVC.

3. Experimental Results and Analysis

To verify the proposed method for the transform kernel derivation for VVC, the similarities between the proposed kernels and the original kernels in VTM-3.0 [27] were investigated by plotting the kernels in a graphical representation. Figure 19 shows the comparison of the transform kernels. It can be seen that the derived transform kernels are almost perfectly matched with the original transform kernels. As described in Section 3, only a few integer elements in the 32-point DST-7 kernel are tuned from the original kernels to obtain full transform kernels. Thus, it can be inferred that the proposed transform kernels are more likely to have identical coding gains.

Experiment for verification of the proposed method were performed on the VVC reference software, VTM-3.0 [27], using CTC conditions [29]. The PC for simulation was Centos 7 OS with an Intel® Xeon® Silver 4114 CPU @ 2.20 GHz processor with 200 cores and 156 GB of RAM. Parallel processing is turned off to precisely measure the encoding and decoding complexity in the execution time. The coding performance was measured in BDBR (%) [30].

Table 2 shows the experimental results of the proposed method. As shown in Table 2, the coding performances of the proposed method are almost identical to the original VTM3.0 for all configurations. This is because the similarity of the proposed transform kernel with the original transform kernel is almost perfect.

(a) Original DCT-2 vs proposed DCT-2 (16-point)

(b) Original DST-7 vs proposed DST-7 (32-point)

(c) Original DCT-8 vs proposed DCT-8 (32-point)

Figure 19. Comparison between the proposed and original transform kernels.

Table 2. Simulation results.

Test Sequences		AI (All Intra)			RA (Random Access)			LDB (Low Delay B)		
		Y	U	V	Y	U	V	Y	U	V
Class A1 (4K)	<i>Tango2</i>	−0.01	−0.31	0.08	0.04	0.14	0.01	-	-	-
	<i>FoodMarket4</i>	0.01	−0.01	0.04	0.00	0.11	0.17	-	-	-
	<i>Campfire</i>	−0.01	0.08	−0.10	0.00	−0.04	0.01	-	-	-
Class A2 (4K)	<i>CatRobot1</i>	−0.01	−0.01	0.01	−0.04	0.10	0.17	-	-	-
	<i>DaylightRoad2</i>	0.00	−0.10	0.02	−0.04	0.19	0.07	-	-	-
	<i>ParkRunning3</i>	0.01	0.00	−0.03	−0.03	−0.06	0.01	-	-	-
Class B (1080p)	<i>MarketPlace</i>	0.02	−0.13	0.00	−0.02	0.28	−0.13	0.05	−0.04	−0.35
	<i>RitualDance</i>	0.00	0.07	0.10	−0.01	−0.02	0.00	0.03	−0.11	−0.27
	<i>Cactus</i>	0.01	0.01	0.05	−0.03	−0.04	0.14	−0.10	−0.45	−0.06
Class C (WVGA)	<i>BasketballDrive</i>	−0.01	−0.02	0.07	0.05	−0.01	−0.14	0.01	−0.16	−0.13
	<i>BQTerrace</i>	0.00	0.15	0.09	0.01	0.69	0.17	−0.05	−0.68	−1.28
	<i>BasketballDrill</i>	0.01	−0.18	−0.12	−0.01	0.13	0.15	0.04	0.55	0.10
Class D (WQVGA)	<i>BQMall</i>	−0.01	0.15	−0.06	−0.02	−0.12	0.18	0.16	0.61	0.11
	<i>PartyScene</i>	0.00	−0.04	−0.10	0.02	0.18	−0.18	0.04	−0.22	−0.08
	<i>RaceHorses</i>	0.01	−0.09	0.00	0.00	−0.15	0.24	−0.01	−0.10	0.28
Class E (720p)	<i>BasketballPass</i>	0.01	−0.17	−0.08	−0.03	−0.09	−0.17	−0.09	0.36	−0.43
	<i>BQSquare</i>	0.00	0.23	0.28	−0.07	−0.95	−0.24	−0.02	0.55	0.59
	<i>BlowingBubbles</i>	0.00	−0.03	0.03	−0.01	0.01	0.30	0.00	−0.43	−0.83
Class E (720p)	<i>RaceHorses</i>	0.00	−0.03	0.18	0.08	0.23	−0.47	−0.02	−0.16	−0.17
	<i>FourPeople</i>	0.00	−0.05	−0.01	-	-	-	0.17	−0.77	−0.30
	<i>Johnny</i>	−0.02	0.18	−0.07	-	-	-	−0.22	2.81	−0.22
Class E (720p)	<i>KristenAndSara</i>	0.02	−0.02	0.08	-	-	-	0.00	−0.43	−1.00
Overall		0.00	−0.02	0.00	−0.01	0.09	0.06	0.01	0.08	−0.27

Unit: %

Tables 3–5 show a comparison of the results of the proposed method for the AI, RA, and LDB configurations [31], respectively.

Although COT [21] provides better coding gain, it fails to provide a 64-point DCT-2 transform kernel, as some rows of 64-point DCT-2 transform kernels are removed and replaced with selected row values of DST-7/DCT-8 transform kernels [21].

In addition, [21] used five transform kernels, that is, DCT-2, DST-7, DCT-8, DST-4, and DCT-4. Similarly, the unified matrix [23] presented excellent coding gain; however, the higher resolution class sequences, Classes A1 and A2 have higher coding loss. This results from the deviation of the proposed mathematically derived transform kernels with the original transform kernels. In [24,25], owing to the mismatch in the proposed transform kernels with original transform kernels, coding losses can be seen. In our proposed method, no coding loss is observed in higher-resolution class sequences. The overall result showed a minor gain of 0.02% in the U-chroma. This is because the proposed transform kernels were in good agreement with the original transform kernels, as shown in Figure 19.

Table 4 shows the overall experimental results of RA using the VTM-3.0 reference software under the CTC conditions. Coding loss can be seen in [23–25] but [21] showed less significant luma gain. Similarly, the proposed method showed no significant loss compared to the other methods.

Table 5 shows the overall experimental results of LDB using the VTM-3.0 reference software under the CTC conditions. [21,23,25] showed higher coding loss, whereas the proposed method showed a 0.27% coding gain in V-chroma and negligible loss in U-chroma.

Table 3. Comparison of the overall simulation results for the derivation of transform kernels with the proposed method (AI).

	COT [18]					Unified Matrix [20]					AMT [21]					TAF [22]					Proposed Method				
	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT
Class A1	−0.007	−0.014	0.092	101	100	0.200	0.162	0.299	101	100	0.061	0.184	0.187	101	97	0.101	0.115	0.125	95	84	0.000	−0.080	0.000	100	100
Class A2	−0.050	−0.051	−0.012	102	100	0.152	0.089	0.085	101	100	0.063	0.027	0.105	102	97	0.139	0.076	0.030	97	88	0.000	−0.041	0.001	100	100
Class B	−0.136	−0.131	−0.125	102	98	−0.024	−0.147	−0.140	101	99	0.031	0.093	0.135	102	98	0.075	0.029	0.097	96	88	0.001	0.021	0.059	100	100
Class C	−0.289	−0.214	−0.310	101	99	−0.304	−0.354	−0.415	102	100	0.009	−0.005	0.113	105	105	0.015	0.065	−0.032	97	93	0.000	−0.040	−0.071	100	100
Class D	−0.314	−0.324	−0.307	104	104	−0.349	−0.356	−0.474	103	102	0.007	0.052	−0.009	107	109	0.042	−0.114	0.085	97	96	0.000	0.000	0.100	102	100
Class E	−0.335	−0.277	−0.132	98	98	−0.214	−0.350	−0.289	100	98	0.025	−0.063	0.107	109	106	0.063	0.044	0.110	96	89	0.001	0.041	0.001	100	100
Overall	−0.167	−0.141	−0.112	101	99	−0.058	−0.136	−0.115	101	99	0.033	0.054	0.107	104	101	0.075	0.060	0.064	96	88	0.000	−0.020	0.001	100	100
Unit: %																									

Unit: %

Table 4. Comparison of the overall simulation results for the derivation of transform kernels with the proposed method (RA).

Seq.	COT [18]					Unified Matrix [20]					AMT [21]					TAF [22]					Proposed Method				
	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT	Y	U	V	EncT	DecT
Class A1	0.002	0.021	0.048	101	99	0.175	0.259	0.324	101	101	0.032	0.059	0.139	101	99	0.090	0.106	0.079	99	98	0.011	0.070	0.060	101	100
Class A2	−0.021	0.168	0.093	99	99	0.140	0.190	0.311	99	99	0.040	0.118	0.215	100	100	0.055	0.195	0.239	99	98	−0.040	0.080	0.080	101	100
Class B	−0.069	0.060	0.012	101	99	0.067	0.172	−0.104	100	100	0.048	−0.128	0.052	101	99	0.047	0.248	0.117	99	98	0.002	0.178	0.011	100	100
Class C	−0.111	−0.046	0.037	101	100	−0.069	−0.043	0.049	98	96	0.000	0.041	0.027	99	98	0.056	−0.124	0.041	99	100	0.001	0.010	0.100	101	100
Class D	−0.136	−0.438	−0.341	101	101	−0.145	−0.445	−0.259	98	96	0.055	0.256	0.238	100	100	−0.017	−0.447	−0.511	99	100	−0.006	−0.201	−0.149	101	100
Overall	−0.057	0.045	0.042	101	99	0.067	0.136	0.105	99	99	0.035	0.060	0.130	100	99	0.060	0.110	0.114	99	99	−0.010	0.086	0.061	101	100
Unit: %																									

Unit: %

Table 5. Comparison of the overall simulation results for the derivation of transform kernels with the proposed method (LDB).

Seq.	Y	U	COT [18]			Y	Unified Matrix [20]				Y	U	TAF [22]			Y	Proposed Method			
			V	EncT	DecT		U	V	EncT	DecT			V	EncT	DecT		U	V	EncT	DecT
Class B	−0.015	−0.263	0.078	100	100	0.168	0.016	0.326	98	96	0.048	−0.169	0.118	100	101	−0.010	−0.291	−0.420	100	100
Class C	0.015	0.437	0.174	99	96	0.148	0.341	0.388	98	98	0.013	−0.009	0.214	100	100	0.060	0.211	0.100	101	100
Class D	0.016	0.202	0.078	104	102	0.057	0.054	−0.170	98	97	−0.035	0.206	−0.171	100	100	−0.030	0.081	−0.210	101	100
Class E	0.098	1.063	0.158	102	98	0.186	0.812	−0.355	96	88	0.019	1.000	0.162	100	99	−0.020	0.540	−0.510	101	100
Overall	0.023	0.302	0.130	100	98	0.166	0.323	0.177	97	95	0.030	0.177	0.161	100	100	0.000	0.079	−0.270	101	100

Unit: %

4. Conclusions

This study proposed an analytical kernel derivation method for the VVC video codec. Memory consumption to store different block size transform kernels is regarded as a major issue in video coding standardization. To address this scenario, the proposed method attempts to save static memory by storing only fewer elements of the transform kernels. Full-required transform kernels are derived using the stored transform kernels and generated unit-element matrices and a permutation matrix. In the proposed method, all the different block size transform kernels are generated dynamically during runtime. The static memory required is only for 1648 elements instead of 8180 elements, each with 8-bit precision, which means that approximately 80% of the static memory can be saved. The fast algorithm of DCT-2 and DST-7 can be implemented in the proposed method as the proposed signal closely overlaps with the original transform kernels, that is, DCT-2, DCT-8, and DST-7. Similarly, the proposed grouping concept of the DST-7 transform kernel shows the effect of tuning on one-row elements on the remaining rows that depend on the tuned row element using the permutation matrix.

Author Contributions: Conceptualization, A.K. and B.L.; methodology, B.L.; software, A.K.; validation, A.K. and B.L.; formal analysis, B.L.; investigation, B.L.; writing—original draft preparation, A.K.; writing—review and editing, B.L.; visualization, B.L.; supervision, B.L.; project administration, B.L.; funding acquisition, B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korea government (No. NRF-2019R1I1A3A01058959).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vetrivel, S.; Suba, K.; Thisha, G.A. An Overview of H.26x Series And its Applications. *Int. J. Eng. Sci. Technol.* **2010**, *2*, 4622–4631.
2. Wiegand, T.; Sullivan, G.J.; Bjontegaard, G.; Luthra, A. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Technol.* **2003**, *13*, 560–576. [[CrossRef](#)]
3. Sullivan, G.J.; Ohm, J.; Han, W.; Wiegand, T. *Overview of the High Efficiency Video Coding (HEVC) Standard*; IEEE: Piscataway, NJ, USA, 2012; pp. 1649–1668.
4. Chen, J.; Ye, Y.; Kim, S.H. Algorithm description for Versatile Video Coding and Test Model 3 (VTM 3), [JVET-L1002]. In Proceedings of the JVET of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 12th Meeting, Macao, China, 3–12 October 2018.
5. Rao, K.R.; Yip, P. Discrete Cosine Transform. In *IEEE Transactions on Computers*; Academic Press: Boston, MA, USA, 1990.
6. Cham, W.K. Development of Integer Cosine Transforms by the Principle of Dyadic Symmetry. *Proc. Inst. Elect. Eng.* **1989**, *136*, 276–282. [[CrossRef](#)]
7. Strang, G. The Discrete Cosine Transform. *SIAM Rev.* **1999**, *41*, 135–147. [[CrossRef](#)]
8. Abedi, M.; Sun, B.; Zheng, Z. A Sinusoidal-Hyperbolic Family of Transforms with Potential Applications in Compressive Sensing. *IEEE Trans. Image Process.* **2019**, *28*, 3571–3583. [[CrossRef](#)] [[PubMed](#)]
9. Jain, A.K. A Sinusoidal Family of Unitary Transforms. *IEEE Trans. Pattern Anal. Mach. Intell.* **1979**, *1*, 356–365. [[CrossRef](#)] [[PubMed](#)]
10. Sullivan, G.J.; Ohm, J. Meeting Report of the 13th Meeting of the Joint Video Experts Team (JVET). In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
11. Zhao, X.; Chen, J.; Karczewicz, M.; Zhang, L.; Li, X.; Jung, W. Enhanced Multiple Transform for Video Coding. In Proceedings of the Data Compression Conference, Snowbird, UT, USA, 29 March–1 April 2016.
12. Philippe, P.; Biatek, T.; Lorc, V. Improvement of HEVC Inter-coding Mode Using Multiple Transforms. In Proceedings of the 25th European Signal Processing Conference (EUSIPCO), Kos Island, Greece, 28 August–2 September 2017.
13. Abe, K.; Toma, T.; Ikeda, M.; Naser, K.; Leanne, F.L.; Francois, E. CE6: JVET-L0262: Replacing all DST-7/DCT-8 by DST-4/DCT-4 used in MTS (test 6.1.1e). In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.

14. Zhao, Z.; Kim, S.; Zhao, Y.; Egilmez, H.; Koo, M.; Liu, S.; Lainema, J.; Karczewicz, M. Transform coding in VVC standard. *IEEE Trans. Circuits Syst. Video Technol.* **2021**. early access. [[CrossRef](#)]
15. Bross, B.; Chen, J.; Liu, S. Versatile Video Coding (Draft 4) JVET-M1001. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
16. JVET VTM Software—JVET. Available online: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/tree/VTM-4.0 (accessed on February 2019).
17. Zhao, X.; Li, X.; Luo, Y.; Liu, S. CE6: Fast DST-7/DCT-8 with dual implementation support JVET-M0497. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
18. Luxán-Hernández, S.D.; George, V.; Ma, J.; Nguyen, T.; Schwarz, H.; Marpe, D.; Wiegand, T. CE3: Intra Sub-Partitions Coding Mode (Tests 1.1.1 and 1.1.2). In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
19. Zhao, Y.; Gao, H.; Yang, H.; Chen, J. CE6: Sub-block transform for inter blocks (CE6.4.1). In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
20. Budagavi, M.; Fuldseth, A.; Bjøntegaard, G.; Sze, V.; Sadafale, M. Core Transform Design in the High Efficiency Video Coding (HEVC) Standard. *IEEE J. Sel. Top. Signal Process.* **2013**, *7*, 1029–1041. [[CrossRef](#)]
21. Zhao, X.; Li, X.; Liu, S. CE6: Compound Orthonormal Transform JVET-M0200. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
22. Feig, E.; Winograd, S. Fast algorithms for the discrete cosine transform. *IEEE Trans. Signal Process.* **1992**, *40*, 2174–2193. [[CrossRef](#)]
23. Choi, K.; Park, M.; Park, M.W.; Choi, W. CE6: Unified matrix for transform JVET-M0200. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
24. Said, A.; Egilmez, H.; Seregin, V.; Karczewicz, M. Complexity Reduction for Adaptive Multiple Transforms (AMTs) using Adjustment Stages JVET-J0066. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 10th Meeting, San Diego, CA, USA, 10–20 April 2018.
25. Philippe, P. CE6: MTS simplification with TAF. In Proceedings of the Joint Video Experts Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.
26. Ochoa-Dominguez, H.; Rao, K.R. *Discrete Cosine Transform*, 2nd ed.; CRC Press: Boca Raton, FL, USA, 2019; pp. 133–186.
27. JVET VTM Software—JVET. Available online: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/tree/VTM-3.0 (accessed on November 2018).
28. Zhang, Z.; Zhao, X.; Li, X.; Li, L.; Luo, Y.; Liu, S.; Li, Z. Fast DST-VII/DCT-VIII with Dual Implementation Support for Versatile Video Codec. *IEEE Trans. Circuits Syst. Video Technol.* **2021**, *31*, 355–371. [[CrossRef](#)]
29. Hanhart, P.; Boyce, J.; Choi, K.; Lin, J.-L. JVET common test conditions and evaluation procedures for 360° video. In Proceedings of the JVET of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 12th Meeting, Macao, China, 3–12 October 2018.
30. Lu, G.; Ouyang, W.; Xu, D.; Zhang, X.; Cai, C.; Gao, Z. DVC: An End-to-end Deep Video Compression Framework. In Proceedings of the CVPR, Long Beach, CA, USA, 16–20 June 2019.
31. Bossen, F.; Boyce, J.; Li, X.; Seregin, V.; Sühring, K. JVET common test conditions and software reference configurations for SDR video. In Proceedings of the Joint Video Exploration Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, MA, USA, 9–18 January 2019.