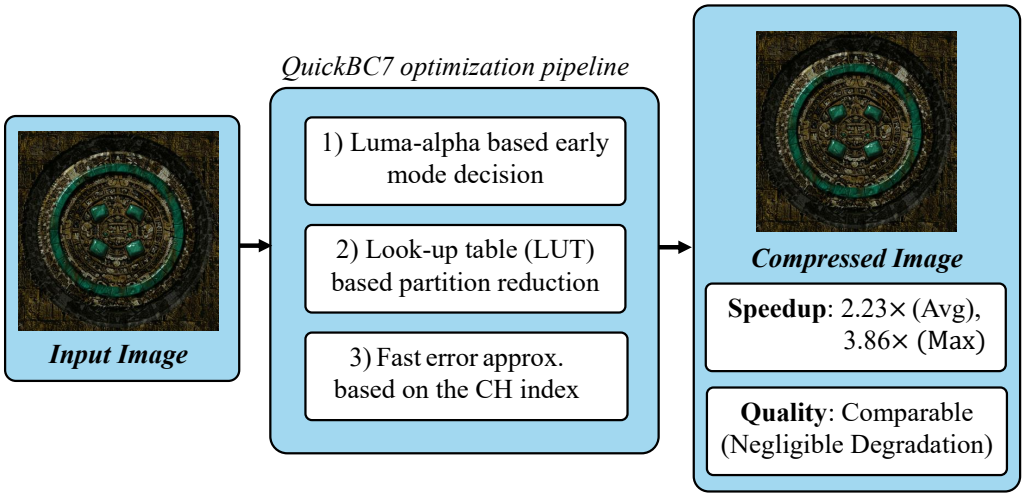
Graphical Abstract

QuickBC7: Fast BC7 Texture Compression Heuristics

Hyeon-ki Lee, Jae-Ho Nah



Highlights

QuickBC7: Fast BC7 Texture Compression Heuristics

Hyeon-ki Lee, Jae-Ho Nah

- Statistical analysis of luma and alpha channels prunes the mode search space.
- Look-up table based on structural correlations accelerates partition selection.
- Calinski-Harabasz (CH) index-based error approximation maximizes encoding speed.
- Proposed method achieves 2.23× average (up to 3.86 ×) speedup with negligible quality loss.

QuickBC7: Fast BC7 Texture Compression Heuristics

Hyeon-ki Lee[a], Jae-Ho Nah[a,*]

[a]*Department of Computer Science, Sangmyung University, 20 Hongjimun 2-gil, Jongno-gu, 03016, Seoul, Republic of Korea*

Abstract

Texture compression is a key technique in real-time rendering for alleviating GPU memory bandwidth bottlenecks. In this paper, we present QuickBC7, a high-speed encoding algorithm for the BC7 format, which is widely adopted as a standard for high-quality texture compression. The proposed method exploits the statistical characteristics of linear luminance (luma) and alpha distributions to reduce the mode search space and to limit partition candidates through a look-up table. In addition, we applied a fast error approximation method based on the Calinski–Harabasz (CH) index to accelerate the partition selection process. Experimental results show that QuickBC7 achieves an average encoding speedup of 2.23× (up to 3.86×) compared to the high-speed encoder etcpak 2.0, while maintaining comparable visual quality with minimal degradation.

Keywords: Texture, Texture Compression, Block Compression, BC7

1. Introduction

Recent advances in virtual reality (VR), high-resolution displays, and real-time rendering applications have led to a steady increase in the use of high-quality textures. To address this demand, texture compression techniques such as BC*n* [1], ETC1/2 [2, 3], and ASTC [4] have become essential components in modern graphics pipelines by reducing memory usage and bandwidth requirements [5]. Among them, the BC7 format is widely adopted in major engines such as Unreal Engine [6] and Unity [7] due to its ability to maintain high visual quality.

*Corresponding author(s).

Texture compression typically involves a trade-off between encoding speed and compression quality. As a high-quality codec compared to other alternatives, BC7 tends to require a relatively longer encoding time. It supports a total of eight modes, from Mode 0 to Mode 7, each with different characteristics in terms of color channels, precision, and subset structures, for representing a single 4×4 pixel block. In addition, each mode employs different partition patterns, endpoint configurations, and bit layouts for compression (details are discussed in Section 2.2). As a result, while BC7 encoders can achieve high visual quality, they must explore a large search space, which significantly increases the computational complexity of the encoding process.

To alleviate this computational bottleneck, various high-speed BC7 encoders, such as *bc7enc* [8] and *etcpak 2.0* [9], have been proposed. These methods improve encoding performance by applying various optimization techniques, including search space reduction and parallel processing, and are widely used in practical texture compression. Similar efforts to accelerate encoding have also been explored in other texture compression codecs. For example, heuristic-based approaches such as QuickETC2 [10, 11] and QuickETC2-HQ [12] have been introduced for ETC2 to reduce unnecessary mode evaluations during encoding. However, these approaches are not directly applicable to BC7 due to its more complex mode structure and RGBA support. As a result, BC7 still exhibits relatively slow encoding speed compared to other codecs, and further optimization is required. Moreover, unlike ETC2, which can distinguish modes using linear luminance (luma), BC7 requires consideration of multiple factors. Therefore, new optimization techniques that can effectively handle these characteristics are needed.

In this paper, we propose a set of lightweight optimization techniques that reduce the computational cost of BC7 encoding by leveraging the statistical and structural characteristics of pixel blocks. Unlike conventional single-channel heuristic approaches, our method introduces a multi-channel decision framework that considers the complex mode and partition structures of BC7. Specifically, we propose a luma–alpha-based early mode decision method that jointly analyzes the variations of the luma and alpha channels to reflect block-level statistical characteristics. This approach effectively removes unnecessary mode evaluations and selects suitable candidates at an early stage. In addition, we design a lookup table (LUT)-based partition candidate reduction method by exploiting structural correlations among BC7 partition patterns. This method replaces computationally expensive partition searches with efficient table lookup operations, thereby significantly

reducing the overall search space. Finally, we introduce a lightweight error approximation method based on the Calinski–Harabasz (CH) index [13], which enables fast partition selection while incurring only a minor quality degradation.

Experimental results show that the proposed techniques achieve effective improvements in encoding speed compared to other encoders (*bc7enc*, *etcpak 2.0*), while the quality loss is limited to a small increase in ꟻLIP [14] values of approximately 0.002–0.003. This increase corresponds to a negligible degradation in visual quality. As a result, the proposed methods achieve an effective trade-off by significantly improving encoding efficiency while preserving visual quality.

2. Related works

2.1. Overview of block compression (BC) format

Block compression (BC) is a standard texture compression technique widely used in modern graphics pipelines, particularly in DirectX and OpenGL, to reduce texture memory bandwidth. BC formats operate by compressing fixed 4×4 pixel blocks and are categorized into seven variants, from BC1 to BC7, according to data characteristics and precision requirements [5].

Basic RGB and alpha formats (BC1–BC3) BC1 (aka. S3TC [15]) represents a block using two 16-bit RGB565 endpoints and a 2-bit index table (4 bpp). Although it supports 1-bit punch-through alpha, the resulting reduction in color precision makes it unsuitable for RGBA888 textures. To overcome this limitation, BC2 and BC3 (8 bpp) allocate an additional 64 bits for alpha compression. BC2 stores explicit 4-bit alpha values, whereas BC3 applies endpoint interpolation to the alpha channel, providing higher quality for alpha textures with smooth gradients.

Special-purpose formats (BC4–BC5) BC4 and BC5 are designed for the efficient storage of single- and two-channel data. BC4 compresses single-channel data at 4 bpp by reusing the alpha block compression scheme of BC3. BC5 combines two independent BC4 blocks to store two-channel data at 8 bpp. This format is commonly used for normal map compression, as it avoids the channel interference that can occur when storing two channels in BC1 and thereby improves visual quality.

High-quality and HDR formats (BC6H–BC7) BC6H and BC7 (aka. BPTC [16]) maintain an 8 bpp compression rate while reducing compression artifacts. BC6H supports half-precision floating-point data and is specialized

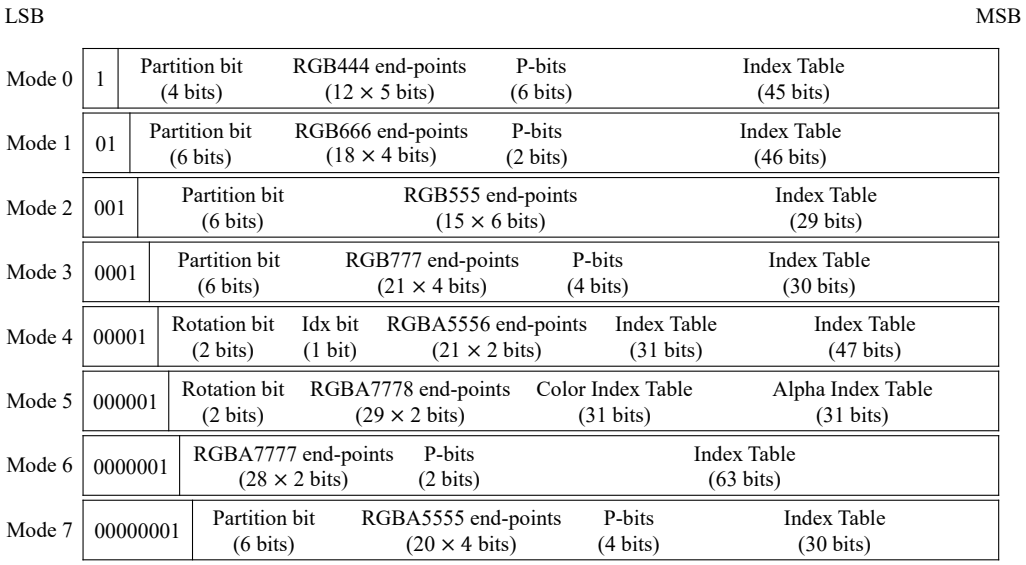


Figure 1: Bitstream layout of the eight BC7 modes (Modes 0–7) [17]. The diagram shows the allocation of bits within a fixed 128-bit block, including partition information, endpoints, P-bits, and index tables for each mode.

for High Dynamic Range (HDR) texture compression. In contrast, BC7 is designed for Low Dynamic Range (LDR) RGB/RGBA data and provides the flexibility to partition a block into multiple subsets and to select modes according to the color distribution within the block. BC7 supports eight modes, whereas BC6H supports fourteen modes; this increased mode complexity leads to higher encoding cost but enables improved endpoint precision and local palette representation.

2.2. BC7 modes and partitioning structure

BC7 encodes a 4×4 pixel block (16 pixels) using a fixed 128-bit structure. Depending on the characteristics of the input data, the encoder selects an appropriate mode from eight distinct modes (Modes 0–7). Each mode is defined by the number of subsets, partition patterns, endpoint color precision, and the configuration of index bits (Figure 1). While this structural flexibility enables high compression quality, it also expands the search space that the encoder needs to explore identify an optimal configuration.

A defining characteristic of BC7 is its ability to partition a pixel block into two or three subsets, each associated with independent endpoints. BC7

defines a global set of 128 partition patterns, categorized by the number of subsets: 64 patterns are assigned to two-subset modes, and the remaining 64 are assigned to three-subset modes. Accordingly, a given partitioned mode evaluates only the 64 patterns corresponding to its subset configuration.

Modes 0 and 2 partition a block into three subsets to represent complex color transitions. However, because these modes require encoding partition indices and three pairs of endpoints, the available bit budget for endpoints or pixel indices is reduced, which can limit color precision. Modes 1 and 3 partition the block into two subsets. In particular, Mode 1 selects one of the 64 partition patterns available for two-subset configurations and employs a shared P-bit for each subset to improve endpoint precision, offering a balanced trade-off between quality and efficiency for RGB data. Mode 7 also supports two subsets while encoding all RGBA channels and is used to represent complex blocks containing both transparency and color boundaries. However, because Modes 1 and 7 require error evaluation over all 64 applicable partition patterns to determine an optimal configuration, they incur substantial computational overhead during encoding.

For blocks with smooth color transitions or when high precision is required in a specific channel, modes without subset partitioning are often preferred. Mode 6 encodes the RGB and alpha channels using a single subset. By providing high bit precision together with P-bits, it is well suited for representing smooth gradients. Mode 5 is advantageous when high precision is needed for a particular channel among the RGBA channels, as it allows distinct bit allocation to a selected channel, making it preferable to Mode 6 for textures in which single-channel precision is critical. Mode 4 is designed for regions with low spatial frequency. This mode prioritizes endpoint precision at the expense of index precision, thereby reducing color banding in flat regions.

Consequently, compressing a single input block in BC7, in principle, involves evaluating all modes along with their corresponding partition configurations. An ideal encoder would compute optimal endpoints for each applicable partition pattern and select the configuration that produces the minimum error. However, such a brute-force approach incurs high computational cost. As a result, many existing encoders reduce this complexity by employing heuristic strategies, such as restricting the set of evaluated modes or limiting the partition search space.

2.3. Previous approaches to texture compression optimization

Various studies and open-source encoders have been proposed to improve texture encoding speed. These approaches can be broadly categorized into three groups: hardware acceleration and heuristic-based optimization, methods leveraging structural information or neural networks, and luminance-based optimization techniques.

Hardware acceleration and heuristic-based BC7 optimization *bc7enc* [8] is a representative open-source BC7 encoder designed with a focus on high-speed encoding. Instead of performing an exhaustive brute-force search over all modes, it selectively evaluates a restricted set of modes and applies a frequency-based sorting strategy derived from offline statistical analysis of large-scale texture datasets (e.g., RGB: Modes 1 and 6; RGBA: Modes 5, 6, and 7). This strategy prioritizes partition patterns that are empirically more likely to be selected.

For two-subset modes, *bc7enc* initially evaluates only the 14 most frequently used partition patterns. Among these candidates, the partition that produces the lowest approximate error is selected as the *key partition*, which serves as a representative structural pattern for the input block. Additional partition candidates are considered only when a predefined predictor indicates structural compatibility with the selected key partition. If no compatible partition is found within the first 34 ranked patterns, the search is terminated early. This heuristic approach reduces encoding time while incurring only a small quality loss compared to an ideal BC7 encoder.

Subsequently, *etcpak 2.0* [9], which builds upon the algorithms of *bc7enc*, achieves up to 2.6× faster encoding speed by increasing hardware-level parallelism through extensive use of multi-threading and SIMD (Single Instruction Multiple Data) intrinsics. In addition, *bc7e* [18] and *ISPCTexTool* [19] employ the Intel SPMD Program Compiler (ISPC) [20] to implement parallel encoding, effectively utilizing the CPU's vector processing capabilities.

Structural information and neural network-based approaches Research aimed at improving compression efficiency by analyzing image structural information or by employing neural networks (NNs) has also been explored. To estimate the optimal partitioning for BC7, FasTC [21] introduced an optimization technique that approximates the color distribution of texel blocks using an axis-aligned bounding box (AABB) and determines endpoints through a generalized cluster fitting process. Alternatively, SegTC [22] proposed an approach that accelerates the search process by extracting

structural boundaries via superpixel-based image analysis and incorporating this information into partition selection.

Similarly, TexNN [23] reformulated the search stage—one of the primary bottlenecks in conventional encoding algorithms—as a classification problem. By employing trained neural networks to rapidly approximate an appropriate encoding configuration for a given texture, TexNN improves computational efficiency of ASTC [4] encoding. More recently, neural network–based compression methods emerging from industrial research have been proposed [24, 25, 26, 27, 28]. However, most of these methods rely on GPU software-based decoding. The associated decoding overhead, resulting from the absence of dedicated hardware support, limits their applicability in real-time rendering pipelines and motivates further research to address these constraints.

Linear luminance utilization and heterogeneous computing The approach most closely related to this study leverages the linear luminance characteristics of pixel blocks. QuickETC2 [10, 11] achieved a substantial performance improvement by preselecting ETC2 modes based on the luma values of pixels within each block and introducing a min–max-based heuristic clustering strategy. However, a limitation of purely luma-based approaches is the potential introduction of visual artifacts due to the loss of color information during computation.

To address this issue, QuickETC2-HQ [12] was proposed to improve encoding quality by analyzing the dominant color channel within a block. This method recalculates RGB weights accordingly and incorporates the adjusted luma values into both error computation and mode decision. Beyond algorithmic optimizations, system-level approaches have also been explored. For example, H-ETC2 [29] improved encoding performance by distributing the computational workload through the simultaneous utilization of CPU and GPU resources.

3. Proposed optimization methods

3.1. Overview of proposed BC7 encoder

We propose a high-speed BC7 encoder built upon *etcpak 2.0* that alleviates the primary computational bottlenecks of the BC7 encoding pipeline while preserving visual quality suitable for real-time applications. The high computational complexity of the BC7 format mainly arises from its large mode search space and the costly decision process associated with partition

pattern selection. To address these challenges, we introduce a three-stage optimization strategy that exploits the statistical characteristics and structural information of texture blocks. Our method is designed under the assumption of standard RGB/A color textures.

First, we employ a luma–alpha-based early mode decision technique that analyzes the luminance and alpha distributions of pixel blocks to prune unnecessary search costs by preselecting a small set of promising mode candidates (Section 3.2). Second, we propose a look-up table (LUT)-based partition selection method that rapidly infers likely partition candidates based on pixel correlation patterns, thereby avoiding exhaustive evaluation of all 64 partition patterns (Section 3.3). Third, we selectively utilized a Calinski–Harabasz (CH) index-based error approximation to further improve computational efficiency by replacing iterative endpoint error calculations with a clustering evaluation metric (Section 3.4). The principles and implementation details of each proposed technique are described in the following subsections.

3.2. Luma-alpha based early mode decision

The core principle of the proposed technique is to predict the optimal mode in advance by analyzing the statistical characteristics of pixel blocks, thereby reducing computational complexity. QuickETC2 [11] demonstrated the effectiveness of pre-determining modes based on the luma range of a pixel block to improve encoding speed. However, because BC7 supports the RGBA format including the alpha channel, variability in the alpha channel also has a substantial impact on optimal mode selection. Therefore, we extend the luma-based approach and propose a novel early mode decision technique that considers the variability of both luma and alpha channels.

We first analyzed the statistical distribution of blocks for which specific BC7 modes were selected as optimal. Figure 2 presents histograms showing the range (maximum–minimum difference) of luma and alpha values for each mode, using the *Android Jellybean* image as an example. This analysis highlights distinct characteristics that differentiate the modes.

Modes 5 and 6 are predominantly selected in regions with small luma differences, indicating that they are favored in relatively flat areas with low-frequency variations. In contrast, Modes 1 and 7 display a wider distribution of luma differences. These two-subset modes are better suited for representing complex edges or regions with high-frequency details.

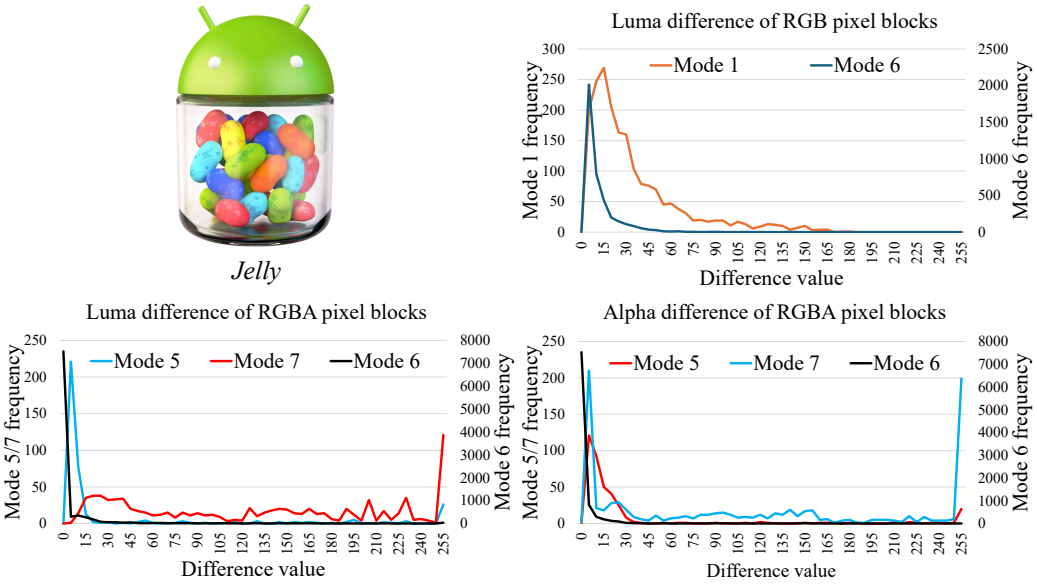


Figure 2: Histograms showing the distributions of luma and alpha difference values for blocks encoded with optimal BC7 modes on the *Android Jellybean* image. Two-subset modes (Modes 1 and 7) are predominantly selected in regions with large luma variations. For RGBA texture encoding, although Modes 5 and 6 are both favored for blocks with low luma differences, Mode 5 is more frequently selected for blocks exhibiting high alpha variability, distinguishing it from Mode 6.

The distinction between Modes 5 and 6 is further evident in the alpha channel distribution. Mode 6 is concentrated in ranges with small alpha variations, whereas Mode 5 spans a broader range, including blocks with large alpha variations. This behavior reflects the design characteristics of the BC7 format. As discussed in Section 2.2, Mode 6 encodes color and alpha components jointly, prioritizing overall compression efficiency for the pixel block. In contrast, Mode 5 allows separate bit allocation to a specific channel, enabling more precise representation of variations within that channel.

Based on this statistical analysis, we designed a decision algorithm that selects the optimal BC7 mode by leveraging the characteristics of pixel blocks. Figure 3 compares the mode selection process of the existing *etcpak 2.0* with that of the proposed method. To quantify block complexity, we define D_L and D_A as follows:

$$D_L = \max(Y) - \min(Y), \quad D_A = \max(A) - \min(A) \tag{1}$$

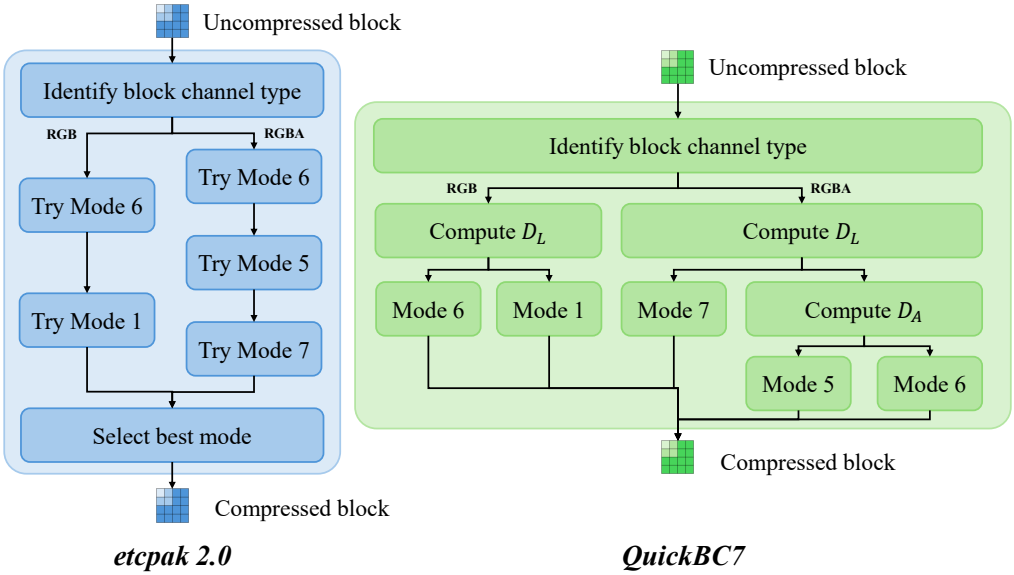


Figure 3: Comparison of the mode decision process between the baseline, *etcpak 2.0*, and QuickBC7. *etcpak 2.0* (left) employs a fixed search strategy that sequentially evaluates a predetermined set of candidate modes. In contrast, QuickBC7 (right) introduces an adaptive decision tree based on luma and alpha difference values (D_L, D_A) to prune unnecessary mode searches and branch directly to a single optimal mode.

Here, Y and A denote the sets of luma and alpha values for the 16 pixels within a block, respectively. Similar to QuickETC2 [11], SIMD intrinsics are used to compute the luma values (Y) of the 16 pixels in parallel. The computed D_L and D_A are compared against predefined thresholds T_L and T_A, which serve as criteria for selecting the optimal mode.

For RGB blocks, if $D_L \leq T_L$, the block is considered relatively flat, and Mode 6 is prioritized. Conversely, if $D_L > T_L$, the block is deemed to contain high-frequency details or significant color variations, and Mode 1, which supports partitioning, is selected to preserve detail.

For RGBA blocks, the decision process accounts for alpha variability in addition to luma. If both D_L and D_A are below their respective thresholds $(D_L \leq T_L$ and $D_A \leq T_A)$, Mode 6 is prioritized. If RGB variability is low but alpha variation is high $(D_L \leq T_L$ and $D_A > T_A)$, Mode 5 is selected to maintain precision in the alpha channel, as alpha details have a stronger impact on visual quality in such cases. Finally, if $D_L > T_L$, the block is considered to contain edges or significant color transitions, and Mode 7 is

prioritized to preserve details. The thresholds T_L and T_A were determined empirically (Section 4.3).

However, binary classification based solely on hard thresholds (T_L, T_A) may reduce accuracy near decision boundaries. As shown in Figure 2, the distributions of optimal modes overlap within certain ranges of D_L and D_A, indicating that in these transitional intervals, a mode other than the pre-selected one could yield a lower error. To address this issue and improve encoding stability, we utilized a *gray-zone* strategy. This zone is defined by a margin (d) around the thresholds $(T \pm d)$. When a block falls within this interval, the encoder evaluates additional candidate modes rather than enforcing a single-mode decision. For example, if an RGB block's luma difference lies within the gray zone, both Mode 1 and Mode 6 are evaluated to identify the true optimum. Although this approach incurs a modest computational overhead, it ensures encoding robustness and consistent visual quality.

3.3. Partition-aware candidate reduction using look-up table (LUT)

Selecting the optimal partition pattern within the BC7 encoding pipeline is a key step in balancing compression quality and encoding speed. Modes 1 and 7 (two-subsets) support 64 partition patterns, and performing an exhaustive search over all patterns introduces a substantial computational bottleneck. To alleviate this issue, the baseline encoder, *etcpak 2.0* inspired by *bc7enc* [8], adopts early termination strategies. Partition patterns are first sorted by usage frequency based on offline statistical analysis of large-scale texture datasets, and the search is performed by prioritizing the most frequently selected patterns. This heuristic significantly reduces encoding time while maintaining acceptable quality.

However, this approach inherently excludes less frequently used patterns (e.g., ranks 35–64), which can be essential for accurately representing specific edges or geometric details in complex textures. Ignoring these patterns may result in detail loss or visual artifacts. To overcome this limitation, we propose a look-up table (LUT)-based partition selection method that directly analyzes the structural characteristics of a pixel block to rapidly identify the most suitable candidate group from all 64 patterns.

The LUT encodes correlations between pixel block features—such as principal color channel indices or spatial distribution characteristics—and partition pattern groups. During encoding, for Mode 1 or Mode 7, instead of exhaustively evaluating all 64 patterns, the encoder performs a simple table lookup to select a small set of promising candidates, effectively reducing

Figure 4: Frequency analysis of partition pattern indices and selection of discriminant pixels for each channel type. (Left) A heatmap illustrating the frequency with which each pixel is assigned to Subset 0 across all 64 partition patterns. Darker shades denote higher frequencies, revealing a distinct statistical bias in the top-left region. (Center) For RGB blocks, indices [0, 1, 4, 5] are selected as discriminant pixels to effectively narrow down partition candidates by exploiting this statistical bias. (Right) For RGBA blocks, corner pixels (indices [0, 3, 12, 15]) are selected to capture texture edges and minimize visual artifacts arising from alpha channel discrepancies.

computational cost. This method ensures that critical partition patterns are not overlooked, while maintaining high encoding efficiency and preserving texture details in complex regions.

The core idea of the proposed LUT-based partition selection framework is to reduce the computational cost of BC7 partition search by using a small set of discriminant pixels, rather than exhaustively evaluating all 4×4 pixel locations. A BC7 partition pattern is represented as a binary mask that divides a 4×4 block into two subsets. To identify informative pixel locations, we first analyze the frequency with which each pixel is assigned to Subset 0 across all 64 partition patterns. The resulting distribution shows a clear spatial bias, as visualized in the heatmap (Figure 4).

Based on this observation, discriminant pixel selection is formulated as a structural approximation problem, where a small subset of pixels is used to represent the dominant partition tendency of the full block. This formulation is inspired by the *selective compression method* of H-ETC2 [29]. Since BC7 blocks show different characteristics depending on channel composition, we adopt a channel-aware form of this framework.

For RGB channels, pixels in the top-left region are most frequently assigned to Subset 0, indicating that these locations capture the dominant

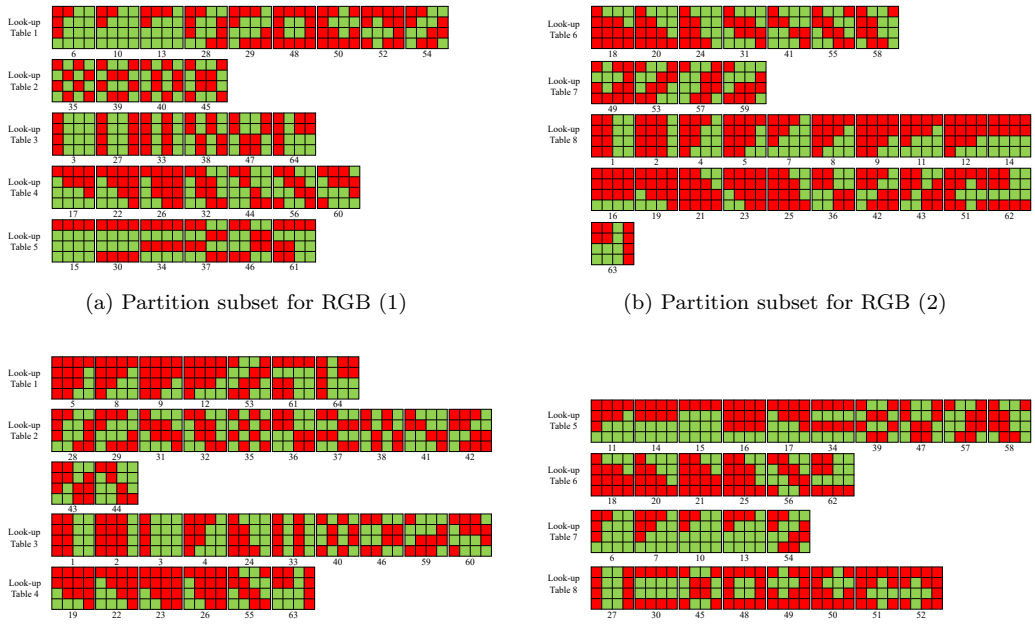
partition tendency. Therefore, we select indices [0, 1, 4, 5] as discriminant pixels. These pixels form a compact 2×2 spatial region that preserves local structural consistency while reflecting the global partition bias observed in the heatmap. Using these pixels, we construct a LUT and observe that partition patterns with similar structures are grouped together according to the generated match mask (Figures 5a and 5b). This shows that a small number of pixels is sufficient to capture structural similarities among partition patterns, allowing a meaningful candidate set to be formed without exhaustive search. Since multiple candidate partitions are evaluated instead of relying on a single prediction, potential suboptimality in pixel selection is naturally reduced.

For RGBA-supporting modes such as Mode 7, the same framework is adapted to focus on geometric structure rather than color distribution. These modes are mainly used in regions with transparency variations or complex boundaries, where small changes in the alpha channel can produce visible artifacts even if RGB errors are small. Therefore, capturing the global structure of the block is more important than relying on channel-wise statistical bias.

Accordingly, for RGBA blocks, we select the four corner pixels (indices 0, 3, 12, 15) as discriminant pixels. These pixels define the spatial boundary of the block and provide a minimal but sufficient representation for inferring global structure such as transparency and opacity. This is consistent with the idea that boundary samples can approximate internal structure. Although other sampling strategies may provide more uniform coverage, the corner-based selection offers a simple and computationally efficient approximation that better reflects global alpha distribution and boundary orientation. In particular, two regions in a block separated by an alpha transition can be effectively characterized based on the relationships among corner pixels. Finally, similar to the RGB case, the constructed LUT for RGBA blocks is shown in Figures 5c and 5d.

Figure 6 illustrates the partition selection process for the proposed RGB block. To efficiently extract the structural features of the block, we adopted the concept of the *dominant color channel* proposed in QuickETC2-HQ [12]. The algorithm proceeds as follows.

First, for the four previously selected discriminant pixels, the channel with the highest intensity among the RGB components is identified as the dominant color channel for that pixel. This serves as a simplified metric representing the pixel's primary color and the direction of its variation. Second,



(a) Partition subset for RGB (1)

(b) Partition subset for RGB (2)

(c) Partition subset for RGBA (1)

(d) Partition subset for RGBA (2)

Figure 5: Visualization of the look-up tables (LUTs) used for partition selection. Each table represents the mapping between match masks and candidate partitions. (a)–(b) correspond to RGB cases, while (c)–(d) correspond to RGBA cases.

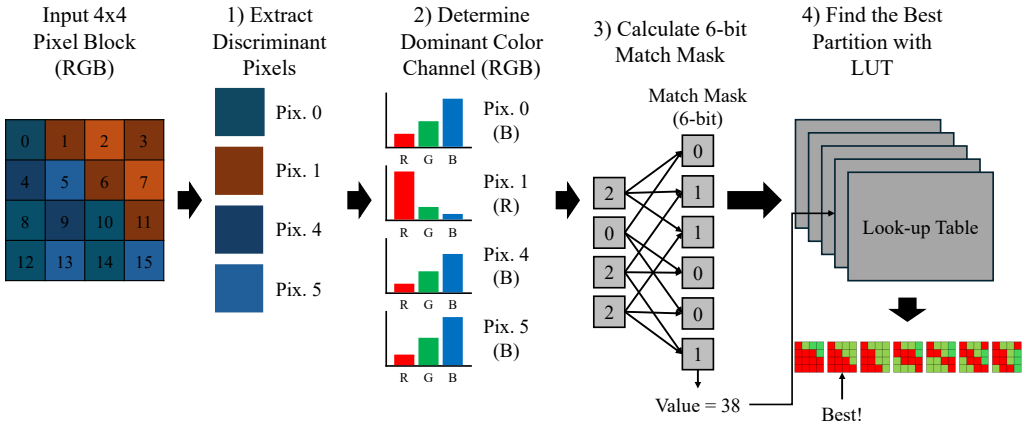


Figure 6: Pipeline of the proposed LUT-based partition pattern selection for RGB blocks. This process comprises four distinct steps: 1) Color information is extracted using the four previously selected discriminant pixels. 2) The dominant color channel, corresponding to the maximum intensity among the RGB components, is determined for each pixel. 3) A 6-bit match mask is generated by comparing the consistency of dominant channels between pairs of discriminant pixels. 4) The generated mask serves as an index to query the Look-up table (LUT) to retrieve a set of partition candidates, from which the final optimal partition pattern is identified.

a 6-bit match mask is generated by evaluating the consistency of dominant channels among all pairs of discriminant pixels. Specifically, a bit is set to 1 if the dominant channels of a pixel pair match, and 0 otherwise. By considering all combinations ($_4C_2$) of the four pixels, a 6-bit index is produced. This mask serves as a unique key representing the color distribution pattern within the block.

Finally, the generated mask is used to query a pre-constructed Look-up table (LUT), retrieving a set of promising partition candidates that best match the structural characteristics of the current block. The final partition pattern is determined by calculating errors only for these selected candidates. This approach replaces iterative and computationally expensive error calculations with a simple table lookup, enabling rapid inference of the optimal partition based solely on the block's color-space structural features.

Figure 7 illustrates the partition selection process for RGBA blocks. Unlike RGB blocks, the alpha channel plays a critical role in defining texture transparency and geometric boundaries, requiring a strategy distinct from that used for RGB.

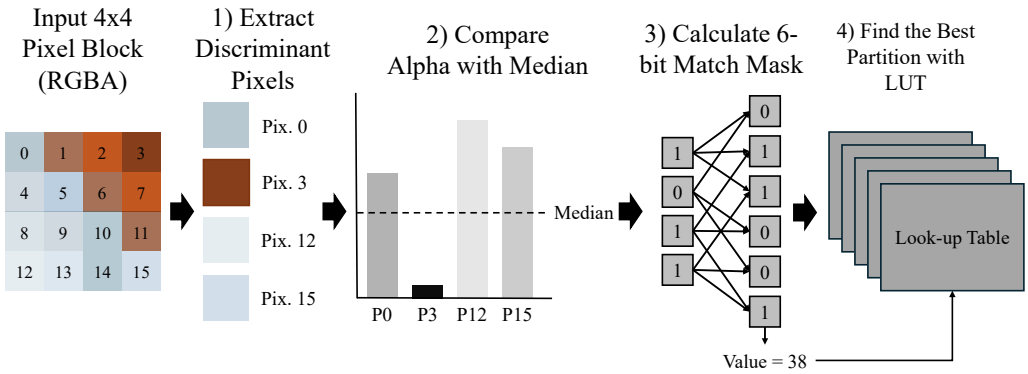


Figure 7: Pipeline of the LUT-based partition selection for RGBA blocks. Unlike the RGB approach, this process prioritizes the analysis of the alpha channel's geometric distribution. The specific steps are as follows: 1) Alpha values are extracted from discriminant pixel indices [0, 3, 12, 15], which represent the geometric boundaries of the block. 2) The extracted values undergo binarization using the median as a threshold. 3) A 6-bit match mask is generated by cross-comparing the consistency of these binarized values across all pixel pairs, similar to the process used for RGB. 4) Finally, the generated 6-bit mask serves as an index to query a dedicated RGBA LUT, retrieving the optimal set of partition candidates.

To process RGBA blocks, we utilize the previously selected discriminant pixels (indices 0, 3, 12, 15). Unlike the dominant color approach for RGB, we binarize the alpha values of these pixels based on the median. Each discriminant pixel is assigned a value of 1 if its alpha exceeds the median, and 0 otherwise. These binary values are cross-compared to generate a match mask reflecting consistency among the pixels. The resulting mask is then used to query a dedicated RGBA LUT, retrieving a set of promising partition candidates. Subsequent error calculation and final partition selection follow the same procedure as for RGB blocks.

The LUTs used in these pipelines are constructed in an offline pre-processing step. For each of the 64 partition patterns, a corresponding key is generated by evaluating the consistency of subset indices among the discriminant pixels. The LUT is then compiled by mapping each key to its associated partition patterns.

In summary, the proposed LUT-based approach substantially improves encoding speed by eliminating the need to exhaustively search all 64 partition patterns, instead evaluating only a small set of candidates that align with the block's inherent structural characteristics. Furthermore, it can be observed

lorikeet

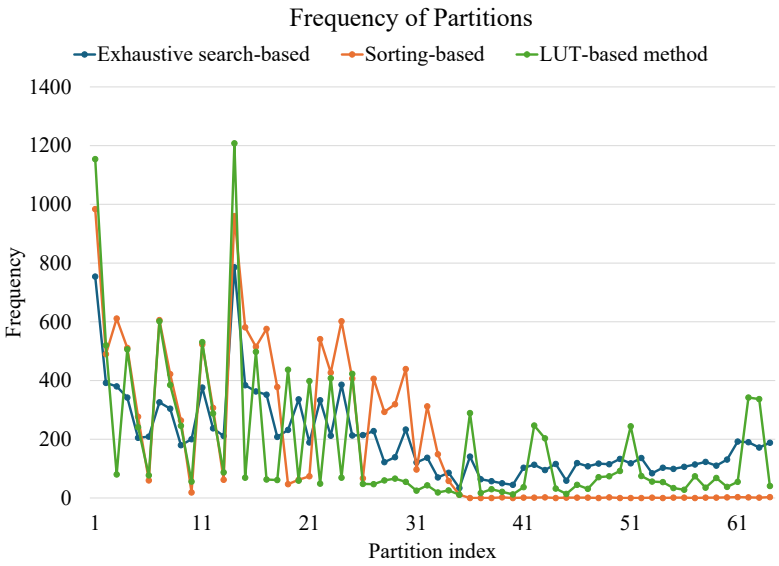


Figure 8: Partition usage distribution across different selection methods for the *lorikeet* image. The results show that the proposed LUT-based method (green) effectively utilizes a wider range of the 64 partition patterns, including less frequently selected ones, overcoming the search space limitations of conventional sorting-based heuristics employed in *bc7enc* (orange).

that the LUT-based method effectively explores a broader set of relevant partition configurations, as illustrated in Figure 8.

3.4. CH index–based fast error evaluation

Although the previously proposed early mode decision and LUT-based techniques effectively reduce the search space, calculating the actual encoding error (e.g., MSE) for the selected partition candidates still requires pixel-wise endpoint optimization, which incurs significant computational cost. To further accelerate encoding, we adopt an additional strategy: a fast error approximation technique using the Calinski-Harabasz (CH) index [13].

The BC7 partitioning process can be interpreted as a clustering problem, in which a 4×4 pixel block is divided into subsets to facilitate linear approximation. An optimal partition corresponds to a configuration where the variance within each subset is minimized (intra-subset compactness) and the distance between subsets is maximized (inter-subset separation). This concept is analogous to ETC2, which employs k-means clustering for endpoint

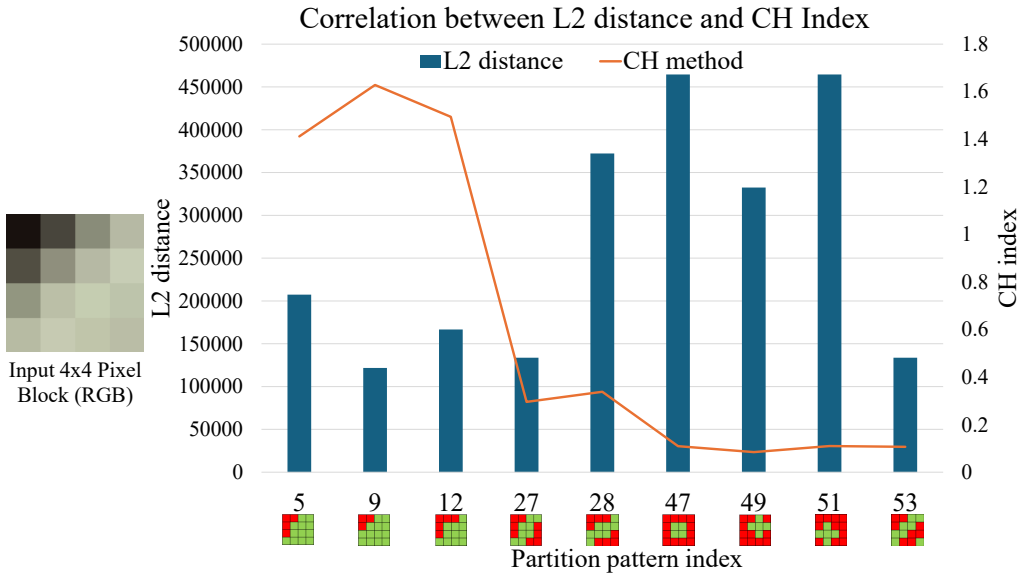


Figure 9: Correlation between L2 distance error and CH index. The graph compares the L2 error (blue bars, left axis) and CH index (orange line, right axis) for nine partition candidates extracted via the LUT described in the previous section. As observed, there is a distinct inverse relationship where partitions with higher CH Indices exhibit lower L2 errors. This suggests that the optimal partition can be effectively identified using only the CH Index, thereby bypassing complex computations.

determination in T- and H-modes [3]. To optimize the computationally expensive endpoint operations, we apply the CH index—a metric for evaluating clustering quality—to rapidly assess partition suitability.

The CH index is defined as the ratio of inter-cluster dispersion to intra-cluster dispersion:

$$CH = \left[\frac{\sum_{k=1}^{K} n_k \|c_k - c\|_2^2}{K-1} \right] / \left[\frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \|x_i^k - c_k\|_2^2}{n-K} \right] \tag{2}$$

Here, x_i^k denotes the i-th data point in cluster k, K is the number of clusters (i.e., subsets), c_k is the centroid of cluster k, n_k is the number of points in cluster k, and c is the global centroid of the dataset. According to this formulation, the CH index increases when inter-cluster distances grow larger and intra-cluster points become more compact, providing a rapid measure of partition quality.

Conventional methods exhibit low computational efficiency because they

rely on exhaustive search strategies that perform endpoint fitting and pixel-wise error evaluation for all partition candidates. In contrast, the proposed approach models the two subsets defined by a partition as clusters with $K = 2$, reflecting the two-subset structure used in *etcpak 2.0* to enable fast encoding.

Instead of performing complex endpoint fitting, we directly measure inter-subset separation using the CH index with $K = 2$. A higher CH index indicates that the two subsets are more distinctly separated in color space. To maximize throughput, these operations are vectorized using SIMD intrinsics. Pixel values stored as 8-bit unsigned integers are first unpacked and converted to floating-point vectors. Centroid computation is performed in parallel across RGBA channels. The squared Euclidean distance for inter-cluster separation is then computed using the SIMD dot product instruction. The dot product mask is dynamically adjusted—`0xF1` for Mode 7 to include the alpha channel and `0x71` for the remaining modes to restrict computation to RGB—ensuring correctness across different encoding configurations without introducing branching overhead.

This approach leverages the CH index to accelerate error evaluation, rapidly identifying the optimal partition while incurring only negligible quality loss. Furthermore, across the full dataset, the proposed CH index achieves an 80.75% agreement rate with the L2-optimal partition selection, demonstrating strong consistency as a surrogate objective for L2-based partition evaluation (see an example in **??**. To prevent visual artifacts in regions with extreme alpha variations, this optimization is selectively disabled when the alpha-channel spread exceeds a predefined safety threshold (e.g., $T_S > 240$; the threshold setting is discussed in Section 4.3).

4. Experiments

4.1. Experiments setting

In this section, we quantitatively and qualitatively assess the impact of the proposed optimization techniques on BC7 encoding efficiency. We selected *etcpak 2.0*, the open-source BC7 encoder described in Section 2, as the baseline for our comparative experiments. In addition, we include *bc7enc* (u0–u4) [8] as another representative encoder with different optimization levels (uX), enabling a more comprehensive comparison across varying rate-distortion trade-offs. This choice ensures stable performance and high portability across platforms, as it is implemented in standard C++ without dependencies on

specific hardware architectures or compilers. To validate a variety of texture characteristics and channel configurations (RGB and RGBA), we constructed an experimental dataset comprising 100 textures, incorporating data from QuickETC2 [11] and a subset of MatSynth [30, 31], the recent PBR Material dataset (Figures A.14 and A.15).

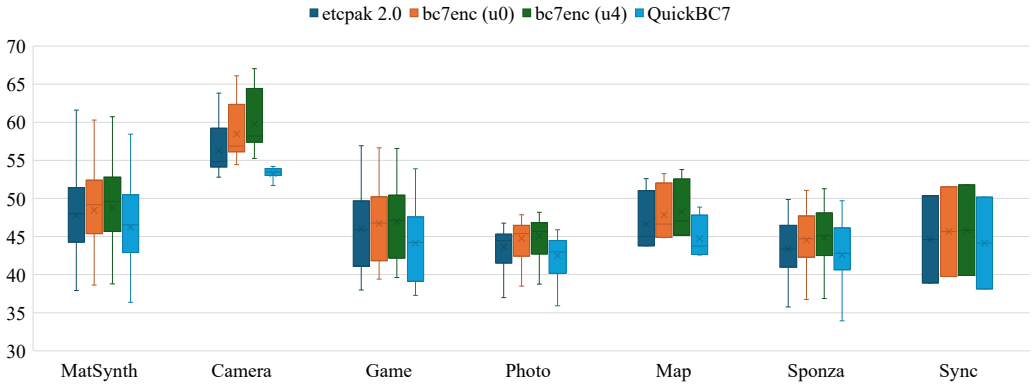
All evaluations were performed on a Windows 11 system. The primary assessment—including encoding speed, quality analysis, and ablation studies—was conducted on a desktop equipped with an Intel Core Ultra 9 285K CPU and 64 GB of RAM. For quantitative quality evaluation, we employed the ꟻLIP metric [14], where lower values indicate better perceptual fidelity. ꟻLIP accounts for both color and feature differences and enables accurate detection of perceptual improvements, including those arising from variable color weighting. It is therefore used as the primary quality metric for in-depth analysis. Encoding speed was measured in MPixels/s (million pixels per second; higher is better), representing the number of pixels processed per unit time. To isolate the intrinsic performance of the algorithm, overheads associated with file I/O and image decoding were excluded from the measurement.

4.2. Encoding quality and speed

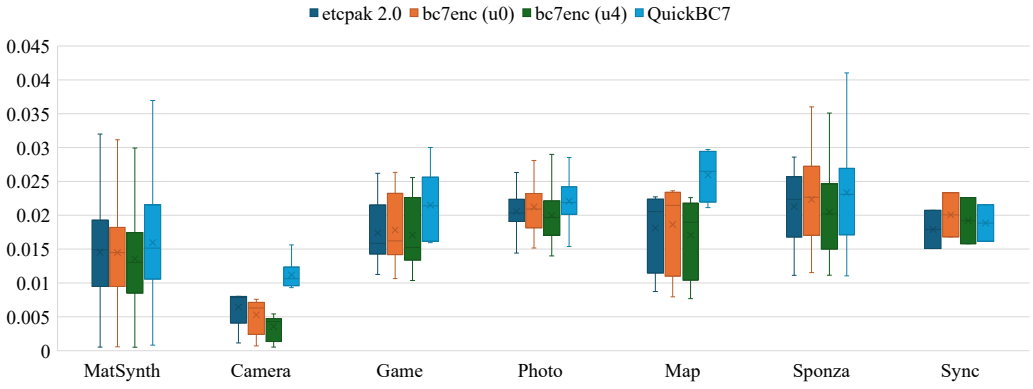
The experimental results demonstrate the effectiveness of the proposed method across multiple representative BC7 encoders, including *etcpak 2.0*, *bc7enc* (u0, u4), and QuickBC7 (Figure 10). In general, texture compression algorithms involve a trade-off between encoding speed and compression quality, where increases in computational speed often result in reduced visual fidelity. The proposed optimization techniques, however, successfully mitigate this trade-off (see Table B.3 for details).

Compared to *bc7enc* (u0, fastest mode) and *etcpak 2.0*, the proposed method exhibits a slight increase in the average ꟻLIP value (approximately 0.001–0.002), while demonstrating a significant improvement in encoding throughput. In particular, on the MatSynth dataset, QuickBC7 achieves approximately 2.4× higher throughput than *etcpak 2.0*, and up to 5.52× speedup compared to *bc7enc* (u0). These results indicate that our method accelerates computation without significant quality loss.

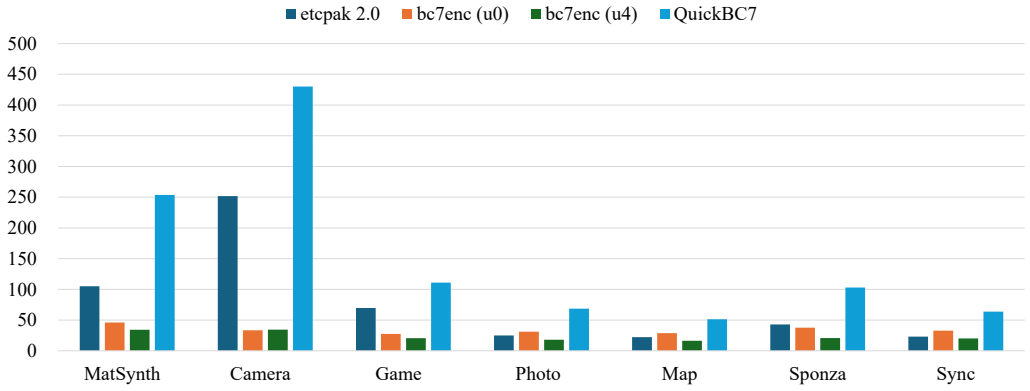
Although small differences are observed in the quantitative ꟻLIP metric, as shown in Figure 11, these differences are visually comparable. Figure 11 presents both qualitative and quantitative comparisons between the proposed



(a) PSNR (↑)



(b) ꟻLIP (↓)



(c) MPixels/s (↑)

Figure 10: Comparison of BC7 encoders across multiple datasets. (a) PSNR (higher is better), (b) ꟻLIP (lower is better), and (c) encoding throughput in MPixels/s. The evaluated methods include *etcpak 2.0*, *bc7enc* (u0), *bc7enc* (u4), and the proposed QuickBC7. The x-axis represents the dataset categories used for evaluation.

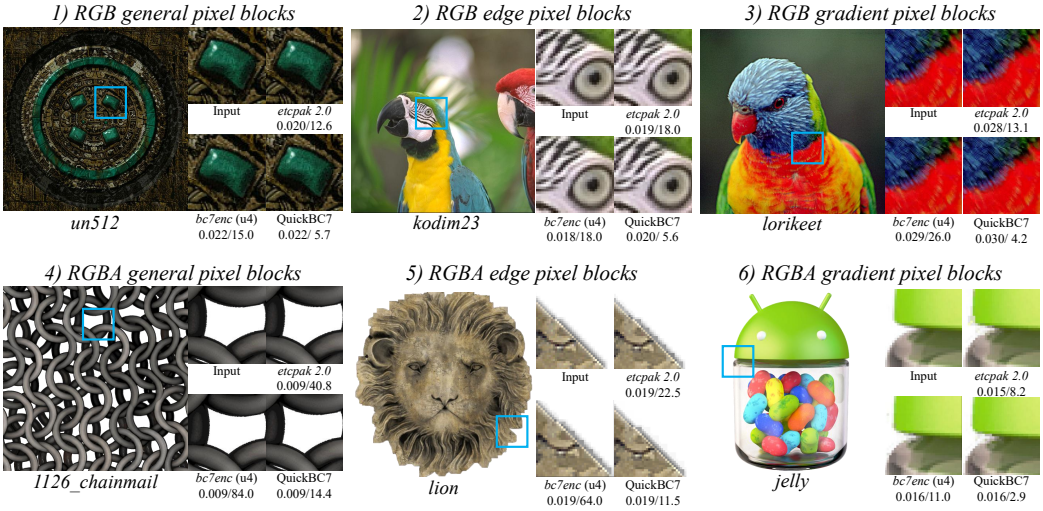


Figure 11: Visual quality comparison of RGB and RGBA textures. The numerical values displayed below each encoder represent the ꟻLIP metric (lower is better) and encoding time in milliseconds (lower is better), formatted as (ꟻLIP / ms). The results demonstrate that the proposed method achieves performance improvements in encoding speed with visually imperceptible quality degradation compared to the existing method.

method and the baselines, *etcpak 2.0* and *bc7enc* (u4), across three texture categories: general, edge, and gradient patterns.

First, for RGB/RGBA general patterns (Cases 1 and 4), the proposed method demonstrates high encoding efficiency in regions containing general texture details or relatively flat areas. In particular, for Case 4, which involves simple alpha masking, the method achieves a ꟻLIP score comparable to that of *etcpak 2.0* while improving encoding speed by approximately 2.83×. This result indicates that the early mode decision technique effectively accelerates encoding by selecting an appropriate mode in advance, thereby eliminating unnecessary mode searches.

Second, RGB/RGBA edge patterns (Cases 2 and 5) correspond to high-frequency regions, such as sharp edges, which are particularly susceptible to visual artifacts caused by abrupt variations in color and alpha values. If Mode 6, which relies on a single subset, is selected in these regions, it is inherently limited in accurately representing color boundaries and may lead to visible artifacts. This issue becomes more pronounced in areas with strong contrast—for example, around the eyes in Case 2 (*kodim23*). In contrast, the proposed algorithm preferentially selects Mode 1 (RGB) and Mode 7

Table 1: Encoding time comparison on the *1126_chainmail* (2048×2048 RGBA) image across different BC7 encoders.

Encoder	**ms**
QuickBC7	14.4
bc7e (u0)	14.8
bc7e (u6)	59.5
ISPC TexTool (ufast)	18.1
ISPC TexTool (vfast)	60.0
ISPC TexTool (fast)	107.3
ISPC TexTool (basic)	208.2
ISPC TexTool (slow)	712.1
NVTT (fast, GPU)	257.1
NVTT (highest, CPU)	40936.2

(RGBA) in such regions, resulting in more faithful preservation of the original boundaries. Furthermore, in Case 2, the LUT-based partition search maintains edge quality while improving encoding speed by approximately 3.2× compared to *etcpak 2.0*.

Finally, the proposed method maintains stable visual quality in regions with smooth color transitions, such as RGB/RGBA gradient patterns (Cases 3 and 6). Although high-speed encoding methods are generally prone to banding artifacts in gradient regions, magnified results for Cases 3 and 6 exhibit visually comparable quality. Notably, in RGBA blocks, the mode decision logic based on alpha distribution differences selects appropriate modes, enabling efficient encoding without compromising visual fidelity. Consequently, despite a slight increase in ꟻLIP values, the proposed method improves encoding speed while preserving gradient quality at a level visually equivalent to *etcpak 2.0*. In some instances (Cases 5 and 6, RGBA), the method even provides better visual quality than *etcpak 2.0*.

Table 1 provides a comparison of encoding time in milliseconds across several existing BC7 encoders on the *1126_chainmail* image, corresponding to Case 4 in Figure 11. Compared to the fastest configurations of the other encoders, the proposed method achieves comparable or lower encoding times. This suggests that the proposed optimization strategy is effective on its own and could also be applied to other BC7 encoders to further improve their encoding efficiency. It is worth noting that slower configurations of these encoders may support a broader set of BC7 modes and thus potentially achieve

higher visual quality, which is not the focus of this comparison.

4.3. Threshold determination

In this study, we conducted experiments on a dataset of 100 textures to determine the optimal thresholds for the proposed techniques, including early mode decision, gray-zone, and CH-index-based optimization described in Sections 3.2 and 3.4. To prevent interference between variables, the thresholds were determined sequentially. First, the luma threshold (T_L), which reflects the complexity of RGB channels, was established. Then, with T_L fixed, the alpha threshold (T_A) was determined. All experiments were performed on a Windows 11 system equipped with an Intel Core i5-12400 CPU and 32 GB of RAM.

Figure 12a shows the performance trends as T_L varies. As T_L increases, the frequency of Mode 6 selection rises, resulting in a nearly linear improvement in encoding throughput. Compression quality remains stable up to $T_L = 19$, after which it begins to degrade. Accordingly, $T_L = 19$ was selected as the optimal value. A similar trend was observed for the alpha channel, where quality degradation starts at $T_A = 21$ (Figure 12b). Therefore, $T_L = 19$ and $T_A = 21$ were selected as the final thresholds.

To mitigate instability near the threshold boundaries, the gray-zone logic was additionally applied. This range was defined based on both the threshold values (T_L, T_A) and the crossover regions in the speed–quality trade-off. Specifically, the gray-zone was set to $19 < D_L \leq 48$ for luma and $10 < D_A \leq 21$ for alpha. Within these ranges, additional neighboring candidate modes are evaluated instead of enforcing a single-mode decision. This strategy improves encoding stability by preventing abrupt mode switching near the decision boundary, where small variations in pixel statistics may otherwise lead to inconsistent mode assignments, while maintaining computational efficiency by limiting additional evaluations to a narrow range.

Finally, to determine the threshold T_S described in Section 3.4, we conducted additional experiments analyzing the trade-off between quality and performance (Figure 12c). The results show that PSNR degradation becomes more pronounced as the threshold increases, particularly at higher values (e.g., 249). Therefore, $T_S = 240$ was selected as the final threshold, as it provides a balanced trade-off between minimizing quality loss and maintaining stable performance. The additional generalization analysis of all selected thresholds with another 80 texture images is provided in Appendix C.

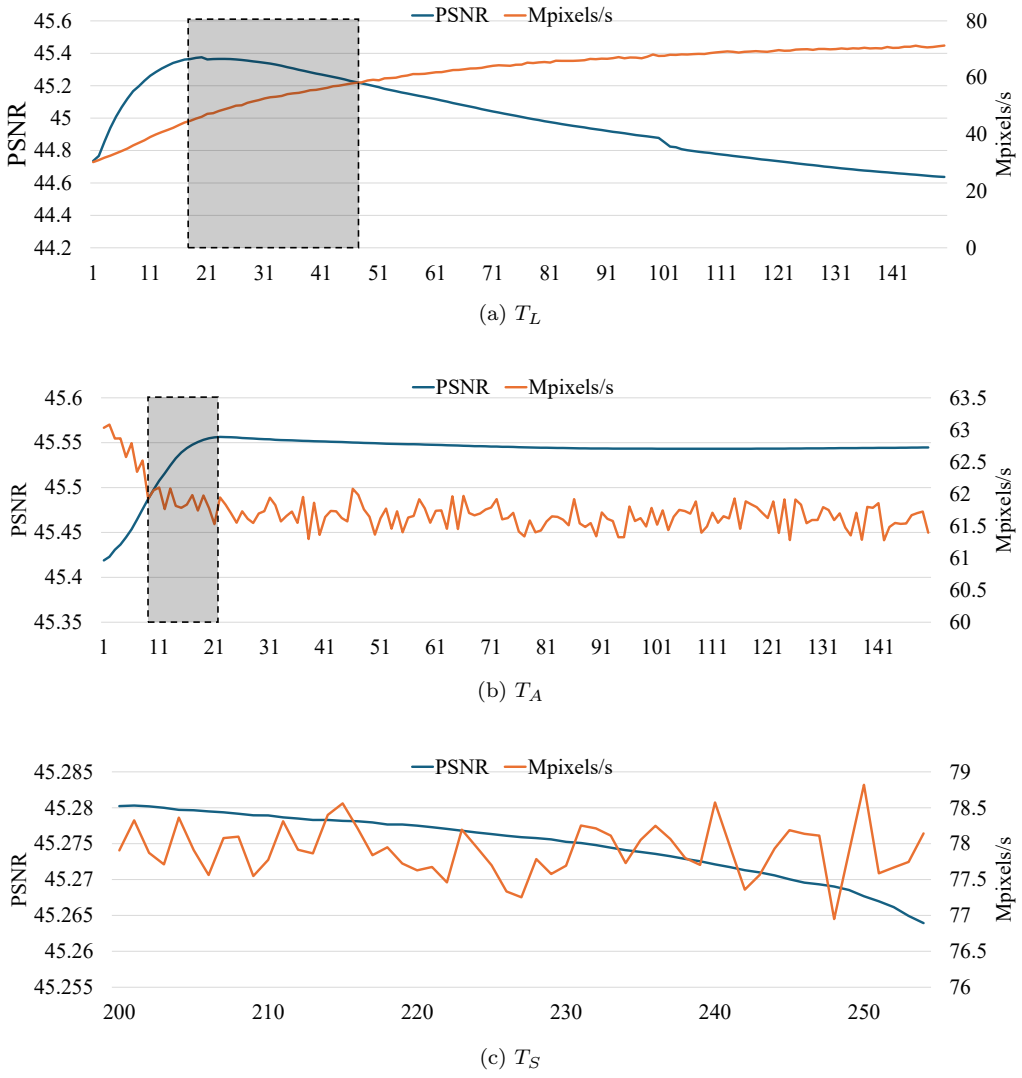


Figure 12: Threshold analysis of PSNR and MPixels/s with respect to (T_L, T_A, T_S).

Table 2: Ablation study analyzing the impact of individual and combined optimization methods. We evaluated the trade-off between performance and quality by progressively applying the proposed techniques to the *etcpak 2.0* baseline. (a), (b), (c), and (d) denote early decision mode, LUT-based partition selection, gray-zone, and CH-index based error calculation optimization, respectively.

Method	**ꟻLIP (↓)**	**MPixels/s (↑)**
Baseline (*etcpak 2.0*)	0.0172	75.75
(a)	0.0194	136.16
(b)	0.0173	88.77
(d)	0.0176	102.88
(a) + (c)	0.0187	131.38
(a) + (b)	0.0197	151.74
(a) + (b) + (c)	0.0189	145.25
(a) + (b) + (d)	0.0202	180.32
(a) + (b) + (c) + (d)	**0.0194**	**169.14**

4.4. Ablation study

To evaluate the impact of each proposed optimization technique on compression quality and encoding performance, we adopt *etcpak 2.0* as the baseline and conduct an ablation study by applying each module individually as well as in incremental combinations. The quantitative results are summarized in Table 2.

First, the early decision mode method (a) provides the most substantial improvement in encoding speed, highlighting the mode selection stage as a major computational bottleneck in BC7 encoding. Combining (a) with other optimization techniques proves particularly effective for further accelerating the encoding process.

Second, the Look-up table (LUT)-based partitioning method (b) yields a notable speed improvement while maintaining compression quality comparable to the baseline. Since exhaustive partition search is one of the most com-

putationally expensive steps in BC7 encoding, reducing the number of evaluated candidates significantly lowers the overall cost. Importantly, method (b) incurs only a marginal increase in the ꟻLIP metric while delivering a clear performance gain. When combined with (a), this approach achieves an average encoding speedup of 2.00× relative to the baseline.

Third, the gray-zone technique (c) primarily contributes to improving compression quality. Although encoding speed is slightly reduced due to the additional evaluation of adjacent modes near decision boundaries, this trade-off results in a measurable improvement in visual fidelity.

Finally, the CH-index-based error calculation optimization method (d) further enhances encoding throughput by simplifying the computationally intensive error metric evaluation. Incorporating (d) into the combination of (a) and (b) achieves the highest observed throughput of 180.32 MPixels/s. While this approximation introduces a minor increase in ꟻLIP values, the trade-off is justified by the substantial gain in processing speed. In the final configuration, integrating (d) effectively offsets the computational overhead introduced by the gray-zone logic, resulting in a balanced throughput of 169.14 MPixels/s while preserving high visual quality.

4.5. Failure cases

While the proposed QuickBC7 achieves substantially higher encoding speed than *etcpak 2.0* while maintaining comparable overall visual quality, it exhibits limitations in certain challenging texture patterns due to the approximate strategies employed for high-throughput processing. Figure 13 presents representative failure cases observed in our experiments.

First, as illustrated by the *Vase_plant* and *Jelly* examples (top and middle rows), boundary regions containing diagonal or gently curved structures may exhibit mild aliasing and block artifacts due to the LUT-based partition selection strategy, which accelerates encoding by evaluating only a limited set of statistically favorable partition candidates. Although this approach performs well for most texture patterns, it can occasionally exclude partition configurations that more accurately align with oblique or smoothly varying boundaries. Consequently, the selected partition may provide only a coarse approximation of the true boundary at the block level, resulting in visible discontinuities along diagonal or curved edges. This limitation could be addressed in future work by reinforcing the edge detection logic or performing additional partition searches.

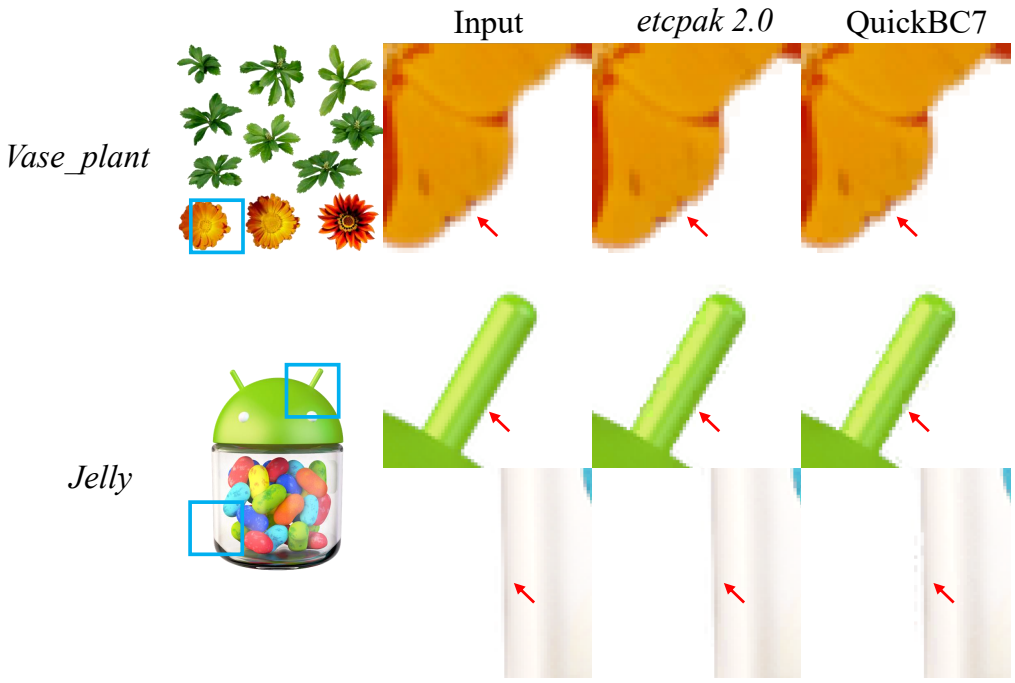


Figure 13: Visual comparison of failure cases. (Left) Original input, (Center) *etcpak 2.0*, and (Right) Proposed method (Ours). While the proposed method maintains overall high quality, it exhibits artifacts in certain challenging regions.

Second, as shown in the *Jelly* example (bottom row, white-highlighted region), banding artifacts can appear in areas with low overall luminance variation combined with subtle color or intensity gradients. In the proposed method, the encoding mode is pre-selected based on the distribution of luma and alpha values within each block to accelerate processing. When the luma variation falls below a predefined threshold, the block may be classified as a near-flat region, favoring a single-subset mode (e.g., Mode 6). In these cases, smooth highlights or gentle transitions that would benefit from a two-subset representation (e.g., Mode 1) may be oversimplified, resulting in visible block boundaries. This issue could potentially be mitigated by enlarging the gray-zone search range or by incorporating chroma-based statistics into the mode decision process.

5. Conclusions

In this paper, we proposed a set of optimization techniques for BC7 encoding that reduce computational cost by exploiting structural properties of pixel blocks. The proposed method includes an early mode decision scheme based on luma and alpha variance, a LUT-based partition selection strategy that reduces the search space using inter-pixel correlations, and a fast error approximation method based on the CH index. In addition, a gray-zone strategy is introduced to improve robustness near decision boundaries and to maintain stable visual quality. Experimental results demonstrate that the proposed method achieves a favorable trade-off between encoding speed and visual quality, making it suitable for large-scale texture processing.

6. Limitation & Future Work

Based on QuickBC7, several future research directions can be considered to further improve performance and broaden the applicability of the proposed method.

First, the current QuickBC7 supports only a subset of frequently used modes (Modes 1, 5, 6, and 7). Since BC7 provides a total of eight modes, extending the proposed framework to the remaining modes (Modes 0, 2, 3, and 4) remains an important direction for future work. This extension may benefit from incorporating richer structural cues, such as combined luma–alpha match mask relationships or more expressive partition-aware representations. In a related direction, recent analytical approaches such as *bc7f* [32] demonstrate that BC7 mode selection can also be formulated as a covariance-based statistical prediction problem over the full mode space. While such methods focus on global error estimation using closed-form modeling, the proposed approach instead emphasizes lightweight structural heuristics for efficient candidate reduction. Exploring how these complementary perspectives can be integrated may further improve robustness and completeness in mode selection across the full BC7 mode space.

Second, extending the proposed methods to other parallel environments remains an important direction. While this work targets a multi-core CPU implementation, the block-independent nature of BC7 encoding makes it well-suited for GPU-based parallelization (e.g., CUDA or compute shaders). Such extensions could further amplify performance gains, particularly in latency-critical scenarios such as real-time texture streaming and load-time

transcoding. Evaluating GPU implementations would provide insights into scalability in massively parallel settings. In addition, integrating some of the proposed methods into other high-performance frameworks (e.g., ISPC-based encoders such as *bc7e* [18] or GPU-oriented toolchains such as NVIDIA Texture Tools 3 [33]) would provide a better trade-off between encoding speed and quality.

Lastly, QuickBC7 is designed for RGB/A textures; accordingly, non-color maps such as normal maps and motion vector maps are not included in the evaluation, as they can be efficiently compressed using BC4 (1-channel) or BC5 (2-channel). However, when multiple independent maps are compressed into a single BC7 texture, the proposed method may not be suitable, since the luma–alpha correlation may not hold and structural similarity across maps may not be sufficiently captured. Therefore, alternative feature representations, such as pixel-wise similarity measures and channel-wise weighting strategies, are likely to be required for such cases.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to greatly appreciate the anonymonous reviewers who proivde valuable, constructive reviews that helped increasing the quality of our manuscript. We also thank the authors of *etcpak 2.0*, *bc7enc*, *bc7e*, ISPCTexTool and NVTT samples who have opened their source code to the public. This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korean Government Ministry of Science and ICT (MSIT)] under Grant RS-2025-00521436.

Data availability

The datasets used for the experiments and the source code can be accessed through the links in the paper.

- QuickETC2 dataset link: https://nahjaeho.github.io

- MatSynth dataset link: https://huggingface.co/datasets/gvecchio/MatSynth
- Source code link: https://github.com/gusrlLee/etcpak

Declaration of generative AI and AI-assisted technologies

During the preparation of this work the authors used Gemini and ChatGPT in order to improve English grammar. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Appendix A. Dataset

The datasets used in all experiments are described as follows. A total of 100 images (Figures A.14 and A.15) were used to evaluate compression quality and encoding speed (MPixels/s), while an additional 80 images (Figure A.16) were used to assess the generalization of the threshold.

Appendix B. Detailed Results

Table B.3 reports a detailed comparison of average ꟻLIP, PSNR, and encoding speed (MPixels/s) across different BC7 encoders. For each metric, values are presented as Mean ± Standard Deviation, with the corresponding minimum and maximum values provided in parentheses.

Appendix C. Threshold Generalization Evaluation

This section presents additional experiments to validate the generalizability of the selected thresholds. We conducted experiments on 80 non-overlapping texture images (Figure A.16) sampled from the MatSynth dataset [30, 31]. The same thresholds ($T_L = 19$, $T_A = 21$, and $T_S = 240$), along with the gray-zone strategy, were applied without modification. The results indicate that the proposed method maintains consistent performance on unseen data, suggesting good generalization capability rather than overfitting to the original dataset (Table C.4).

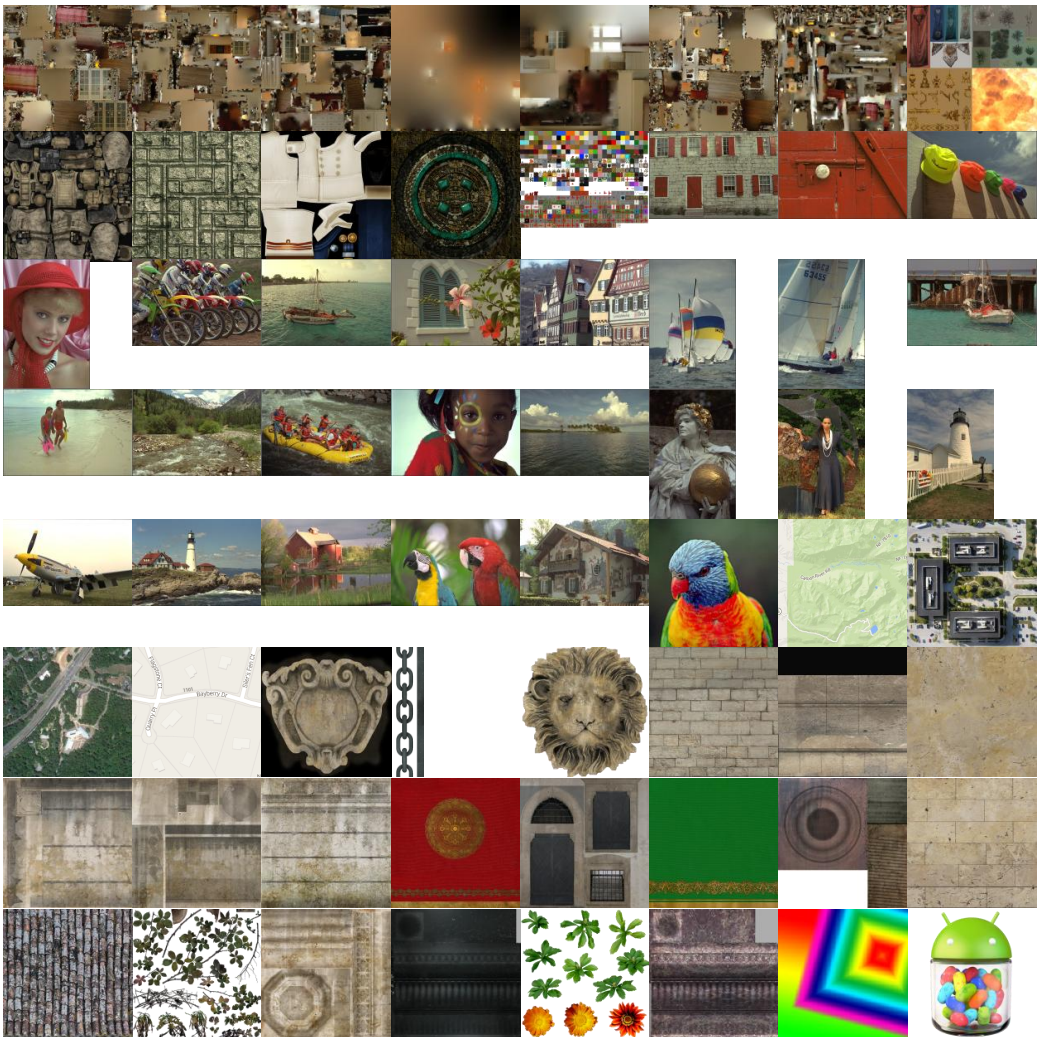


Figure A.14: Overview of the QuickETC2 [11] texture dataset used for evaluation. The dataset comprises 64 images in both RGB and RGBA formats.

Figure A.15: Overview of selected textures from the MatSynth [30, 31] dataset used for evaluation. A total of 36 RGBA images were sampled from the full dataset.

Figure A.16: A subset of 80 RGBA PBR textures extracted from the MatSynth [30, 31] dataset, explicitly selected to avoid overlap with the original dataset. The images exhibit diverse material and texture characteristics, supporting the generalization of the proposed threshold setting.

Table B.3: Comparison of average ꟻLIP, PSNR values and MPixels/s between *etcpak 2.0*, *bc7enc* (u0–u4 presets), and QuickBC7 (Ours). Values are reported as Mean ± Standard Deviation. The values in parentheses below indicate the (Min / Max) bounds. Note that for ꟻLIP, lower values indicate better quality, while for PSNR and MPixels/s, higher values indicate better quality and performance.

	etcpak 2.0	bc7enc (u0)	bc7enc (u1)	bc7enc (u2)	bc7enc (u3)	bc7enc (u4)	QuickBC7 (ours)
				PSNR (dB) (↑)			
MatSynth	47.76 ± 5.53 (37.91 / 61.6)	48.46 ± 5.21 (38.63 / 60.3)	48.67 ± 5.28 (38.7 / 60.73)	48.74 ± 5.26 (38.77 / 60.73)	48.74 ± 5.26 (38.77 / 60.73)	48.76 ± 5.26 (38.79 / 60.73)	46.24 ± 5.40 (36.36 / 58.43)
Camera	56.30 ± 3.87 (52.79 / 63.82)	58.48 ± 4.16 (54.45 / 66.11)	59.70 ± 4.27 (55.16 / 66.93)	59.75 ± 4.27 (55.24 / 67.01)	59.75 ± 4.27 (55.24 / 67.01)	59.77 ± 4.28 (55.27 / 67.05)	53.29 ± 0.81 (51.70 / 54.21)
Game	46.03 ± 6.34 (37.99 / 56.93)	46.72 ± 5.86 (39.42 / 56.65)	46.84 ± 5.80 (39.49 / 56.58)	46.94 ± 5.76 (39.58 / 56.58)	46.94 ± 5.76 (39.58 / 56.58)	46.97 ± 5.75 (39.61 / 56.58)	44.15 ± 5.74 (37.28 / 53.91)
Photo	43.70 ± 2.35 (36.98 / 46.76)	44.75 ± 2.30 (38.49 / 47.87)	44.92 ± 2.32 (38.62 / 48.07)	45.03 ± 2.32 (38.71 / 48.16)	45.03 ± 2.32 (38.71 / 48.16)	45.06 ± 2.32 (38.75 / 48.19)	42.49 ± 2.41 (35.92 / 45.89)
Map	46.62 ± 4.16 (43.77 / 52.61)	47.85 ± 3.96 (44.86 / 53.28)	48.09 ± 4.01 (45.01 / 53.51)	48.22 ± 4.04 (45.14 / 53.68)	48.22 ± 4.04 (45.14 / 53.68)	48.27 ± 4.07 (45.17 / 53.79)	44.74 ± 2.90 (42.57 / 48.86)
Sponza	43.35 ± 3.77 (35.75 / 49.86)	44.54 ± 3.76 (36.74 / 51.06)	44.70 ± 3.80 (36.81 / 51.24)	44.81 ± 3.81 (36.85 / 51.27)	44.81 ± 3.81 (36.85 / 51.27)	44.85 ± 3.81 (36.86 / 51.28)	42.60 ± 4.21 (33.94 / 49.70)
Sync	44.61 ± 8.08 (38.90 / 50.32)	45.65 ± 8.33 (39.76 / 51.54)	45.79 ± 8.45 (39.82 / 51.76)	45.83 ± 8.42 (39.87 / 51.78)	45.83 ± 8.42 (39.87 / 51.78)	45.84 ± 8.42 (39.88 / 51.79)	44.15 ± 8.52 (38.13 / 50.17)
Avg.	**46.25**	**47.27**	**47.52**	**47.61**	**47.61**	**47.64**	**44.84**
				ꟻLIP (↓)			
MatSynth	0.015 ± 0.008 (0.001 / 0.032)	0.014 ± 0.008 (0.001 / 0.033)	0.014 ± 0.007 (0.001 / 0.033)	0.014 ± 0.007 (0.001 / 0.033)	0.014 ± 0.007 (0.001 / 0.033)	0.014 ± 0.007 (0.001 / 0.033)	0.016 ± 0.008 (0.001 / 0.037)
Camera	0.006 ± 0.003 (0.001 / 0.008)	0.005 ± 0.003 (0.001 / 0.008)	0.004 ± 0.002 (0.001 / 0.005)	0.004 ± 0.002 (0.001 / 0.005)	0.004 ± 0.002 (0.001 / 0.005)	0.004 ± 0.002 (0.001 / 0.005)	0.011 ± 0.002 (0.009 / 0.016)
Game	0.017 ± 0.005 (0.011 / 0.026)	0.018 ± 0.006 (0.011 / 0.026)	0.017 ± 0.006 (0.010 / 0.026)	0.017 ± 0.006 (0.010 / 0.026)	0.017 ± 0.006 (0.010 / 0.026)	0.017 ± 0.006 (0.010 / 0.026)	0.022 ± 0.005 (0.016 / 0.030)
Photo	0.021 ± 0.003 (0.014 / 0.028)	0.021 ± 0.004 (0.015 / 0.032)	0.020 ± 0.004 (0.014 / 0.030)	0.020 ± 0.003 (0.014 / 0.029)	0.020 ± 0.003 (0.014 / 0.029)	0.020 ± 0.003 (0.014 / 0.029)	0.022 ± 0.004 (0.015 / 0.030)
Map	0.018 ± 0.006 (0.009 / 0.023)	0.019 ± 0.007 (0.008 / 0.024)	0.017 ± 0.007 (0.008 / 0.023)	0.017 ± 0.007 (0.008 / 0.023)	0.017 ± 0.007 (0.008 / 0.023)	0.017 ± 0.006 (0.008 / 0.023)	0.026 ± 0.004 (0.021 / 0.030)
Sponza	0.021 ± 0.005 (0.011 / 0.029)	0.022 ± 0.006 (0.012 / 0.036)	0.021 ± 0.006 (0.011 / 0.036)	0.021 ± 0.006 (0.011 / 0.035)	0.021 ± 0.006 (0.011 / 0.035)	0.020 ± 0.006 (0.011 / 0.035)	0.023 ± 0.008 (0.011 / 0.041)
Sync	0.018 ± 0.004 (0.015 / 0.021)	0.020 ± 0.005 (0.017 / 0.023)	0.019 ± 0.005 (0.016 / 0.023)	0.019 ± 0.005 (0.016 / 0.023)	0.019 ± 0.005 (0.016 / 0.023)	0.019 ± 0.005 (0.016 / 0.023)	0.019 ± 0.004 (0.016 / 0.022)
Avg.	**0.017**	**0.018**	**0.017**	**0.016**	**0.016**	**0.016**	**0.019**
				MPixels/s (↑)			
MatSynth	105.05 ± 52.26 (70.39 / 335.25)	45.98 ± 13.45 (24.11 / 65.54)	45.06 ± 7.14 (25.97 / 58.66)	38.95 ± 9.83 (23.30 / 57.65)	37.13 ± 9.23 (23.93 / 59.71)	34.17 ± 10.10 (20.39 / 62.84)	253.69 ± 57.96 (172.92 / 382.06)
Camera	251.76 ± 82.01 (171.42 / 374.42)	33.44 ± 1.81 (30.27 / 35.56)	33.16 ± 3.87 (25.57 / 37.81)	29.05 ± 5.34 (23.07 / 35.04)	31.59 ± 3.94 (23.34 / 34.93)	34.53 ± 2.07 (31.34 / 37.22)	430.08 ± 29.37 (384.18 / 469.31)
Game	69.61 ± 106.28 (10.56 / 285.04)	27.43 ± 10.49 (16.38 / 45.59)	25.11 ± 6.92 (13.11 / 33.83)	23.09 ± 8.60 (10.92 / 35.25)	22.73 ± 9.68 (9.36 / 36.47)	20.50 ± 8.42 (9.36 / 34.66)	110.99 ± 122.13 (24.21 / 353.32)
Photo	24.88 ± 5.39 (19.98 / 38.79)	31.11 ± 3.46 (21.85 / 39.32)	26.09 ± 2.42 (21.85 / 30.25)	19.65 ± 4.42 (13.80 / 32.77)	20.03 ± 3.40 (12.48 / 28.09)	18.02 ± 3.73 (10.08 / 28.09)	68.61 ± 23.63 (43.00 / 125.03)
Map	22.28 ± 14.18 (10.75 / 42.71)	28.67 ± 12.11 (16.38 / 43.69)	19.66 ± 8.87 (13.11 / 32.77)	15.68 ± 6.68 (10.92 / 25.58)	16.92 ± 7.62 (13.11 / 28.34)	16.32 ± 4.46 (10.92 / 21.85)	51.22 ± 38.16 (21.70 / 107.24)
Sponza	42.82 ± 8.93 (17.62 / 66.59)	37.65 ± 6.72 (29.13 / 55.19)	32.59 ± 5.17 (21.85 / 43.69)	24.65 ± 6.19 (13.80 / 41.94)	25.37 ± 6.17 (14.56 / 38.84)	20.72 ± 5.53 (12.48 / 32.77)	102.85 ± 27.05 (47.11 / 156.71)
Sync	23.07 ± 12.33 (14.34 / 31.79)	32.77 ± 0.00 (32.77 / 32.77)	27.31 ± 7.72 (21.85 / 32.77)	20.11 ± 5.27 (16.38 / 23.83)	18.47 ± 7.58 (13.11 / 23.83)	20.11 ± 5.27 (16.38 / 23.83)	63.76 ± 33.49 (40.08 / 87.44)
Avg.	**75.75**	**37.65**	**34.42**	**28.32**	**28.07**	**25.65**	**169.14**

Table C.4: Generalization performance on a held-out subset of 80 texture images sampled from the MatSynth [30, 31].

Metric	*etcpak 2.0*	QuickBC7
PSNR (dB)	49.26	47.39
ꟻLIP	0.011	0.014
MPixels/s	144.03	290.07

References

[1] Microsoft, Texture block compression in Direct3D 11 (2018).
URL https://docs.microsoft.com/en-us/windows/win32/direct3d11/texture-block-c

[2] J. Ström, T. Akenine-Möller, iPACKMAN: High-quality, low-complexity texture compression for mobile phones, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware, 2005, pp. 63–70. doi:https://doi.org/10.1145/1071866.1071877.

[3] J. Ström, M. Pettersson, ETC 2: texture compression using invalid combinations, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware, 2007, pp. 49–54. doi:https://dl.acm.org/doi/10.5555/1280094.1280102.

[4] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, T. Olson, Adaptive scalable texture compression, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on High-Performance Graphics, 2012, pp. 105–114. doi:https://dl.acm.org/doi/10.5555/2383795.2383812.

[5] T. Paltashev, I. Perminov, Texture compression techniques, Scientific Visualization 6 (1) (2014) 106–146.

[6] Epic Games, Unreal Engine 5.7 Documentation (2026).
URL https://dev.epicgames.com/documentation/en-us/unreal-engine/texture-forma

[7] Unity Technologies, Unity 6.3 (6000.3) user manual (2026).
URL https://docs.unity3d.com/6000.3/Documentation/Manual/texture-formats-refe

[8] Rich Geldreich, bc7enc - Fast, single source file BC1-5 and BC7/BPTC GPU texture encoders (2020).
URL https://github.com/richgel999/bc7enc

[9] B. Taudul, etcpak 2.0: The fastest ETC compressor on the planet (2024). URL https://github.com/wolfpld/etcpak

[10] J.-H. Nah, QuickETC2: How to finish ETC2 compression within 1 ms, in: ACM SIGGRAPH 2020 Talks, 2020. doi:https://doi.org/10.1145/3388767.3407373.

[11] J.-H. Nah, QuickETC2: Fast ETC2 texture compression using luma differences, ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2020) 39 (6) (2020). doi:https://doi.org/10.1145/3414685.3417787.

[12] J.-H. Nah, QuickETC2-HQ: Improved ETC2 encoding techniques for real-time, high-quality texture compression, Comput. Graph. 116 (C) (2023) 308–316. doi:10.1016/j.cag.2023.08.032.

[13] T. Caliński, J. Harabasz, A dendrite method for cluster analysis, Communications in Statistics-theory and Methods 3 (1) (1974) 1–27.

[14] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Å. M. D. Fairchild, ꟻLIP: A difference evaluator for alternating images, Proceedings of the ACM on Computer Graphics and Interactive Techniques (HPG 2020) 3 (2) (2020). doi:https://doi.org/10.1145/3406183.

[15] K. I. Iourcha, K. S. Nayak, Z. Hong, System and method for fixed-rate block-based image compression with inferred pixel values, uS Patent 5,956,431 (Sep. 21 1999).

[16] OpenGL Architecture Review Board, ARB_texture_compression_bptc (2010).
URL https://registry.khronos.org/OpenGL/extensions/ARB/ARB_texture_compression

[17] Microsoft, BC7 Format Mode Reference (2021).
URL https://learn.microsoft.com/en-us/windows/win32/direct3d11/bc7-format-mod

[18] Rich Geldreich, bc7e - Basis SIMD BC7 Texture Encoder v1.18 (2020).
URL https://github.com/BinomialLLC/bc7e

[19] Intel Corp., Fast ISPC texture compressor (2019).
URL https://github.com/GameTechDev/ISPCTextureCompressor

[20] M. F. Dufresne, How to create a high quality, fast texture compressor using ISPC, in: Game Developer Conference 2015, 2015.
URL https://software.intel.com/sites/default/files/managed/4a/38/High-Quality

[21] P. Krajcevski, D. Manocha, Accelerated texture compression (2014).
URL http://gamma.cs.unc.edu/FasTC/

[22] P. Krajcevski, D. Manocha, SegTC: Fast Texture Compression using Image Segmentation, in: I. Wald, J. Ragan-Kelley (Eds.), Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics, The Eurographics Association, 2014. doi:10.2312/hpg.20141095.

[23] S. Pratapa, T. Olson, A. Chalfin, D. Manocha, TexNN: Fast Texture Encoding Using Neural Networks, Computer Graphics Forum 38 (1) (2019) 328–339. doi:https://doi.org/10.1111/cgf.13534.

[24] L. Belcour, A. Benyoub, Hardware Accelerated Neural Block Texture Compression with Cooperative Vectors, in: A. Knoll, C. Peters (Eds.), High-Performance Graphics - Symposium Papers, The Eurographics Association, 2025. doi:10.2312/hpg.20251173.

[25] F. Farhadzadeh, Q. Hou, H. Le, A. Said, R. Rauwendaal, A. Bourd, F. Porikli, Neural Graphics Texture Compression Supporting Random Access, in: A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, G. Varol (Eds.), Computer Vision – ECCV 2024, Springer Nature Switzerland, Cham, 2025, pp. 412–429. doi:https://doi.org/10.1007/978-3-031-72913-3_23.

[26] K. Vaidyanathan, M. Salvi, B. Wronski, T. Akenine-Möller, P. Ebelin, A. Lefohn, Random-Access Neural Compression of Material Textures, ACM Transactions on Graphics 42 (4) (Jul. 2023). doi:10.1145/3592407.

[27] C. Weinreich, L. de Oliveira, A. Houdard, G. Nader, Real-Time Neural Materials using Block-Compressed Features, Computer Graphics Forum 43 (2) (2024) e15013. doi:https://doi.org/10.1111/cgf.15013.

[28] S. Fujieda, T. Harada, Neural Texture Block Compression, in: Workshop on Material Appearance Modeling, 2024.

[29] H.-k. Lee, J.-H. Nah, H-ETC2: Design of a CPU-GPU Hybrid ETC2 Encoder, Computer Graphics Forum (2023). doi:10.1111/cgf.14969.

[30] G. Vecchio, V. Deschaintre, MatSynth: A Modern PBR Materials Dataset, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.

[31] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, A. Bousseau, Single-image SVBRDF capture with a rendering-aware deep network, ACM Transactions on Graphics (ToG) 37 (4) (2018) 1–15. doi:https://doi.org/10.1145/3197517.3201378.

[32] Richard Geldreich, bc7f: A New Real-Time Analytical BC7 Encoder (2026).
URL https://richg42.blogspot.com/2026/01/bc7f-new-real-time-analytical-bc7.ht

[33] NVIDIA, NVIDIA texture tools 3 (2020).
URL https://developer.nvidia.com/gpu-accelerated-texture-compression