Computers & Graphics (2023)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

QuickETC2-HQ: Improved ETC2 encoding techniques for real-time, high-quality texture compression

Jae-Ho Nah[a,*]

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

ARTICLE INFO

Article history:
Received September 6, 2023

Keywords: Texture compression, Texture encoding, Texture mapping, ETC1, ETC2

ABSTRACT

ETC2 is a widely-used texture compression format in Android devices and applications, so efficient ETC2 encoding can reduce application development time. We present QuickETC2-HQ, a set of improved ETC2 encoding techniques for real-time, high-quality texture compression. Our modifications to the luma-based approximations used in etcpak 1.0, the state-of-the-art encoder which integrated the QuickETC2 method, allow the execution of additional compression modes and more accurate error comparisons. As a result, the image quality of QuickETC2-HQ is improved compared to that of etcpak 1.0 and is comparable to that of the reference encoder, ETCPACK 2.74, with the fast mode. In terms of performance, QuickETC2-HQ is orders of magnitude faster than ETCPACK, making it practical for real-time application execution and offline production builds of applications.

1. Introduction

Texture mapping is one of the fundamental techniques in computer graphics. To create realistic and stunning visual effects, many games and graphics applications include various types of high-resolution textures. As a result, the number and size of textures have been continuously increasing. Since textures are typically stored in a compressed format to reduce memory bandwidth requirements and storage usage, texture encoding time has also been increasing proportionally.

Texture encoding is often utilized in real-time applications. It is considered an alternative option to standard video coding in latency-critical applications like cloud gaming, extended reality, or machine vision [1]. Some other applications, such as 3D reconstruction [2], web-browsing [3], texture resizing [4], and view synthesis [5], demand the real-time compression of on-the-fly generated textures. In such cases, encoding speed is as important as encoding quality due to the limited time budget.

There are several industry-standard texture compression formats: BC series [6], ETC1/2 [7, 8], ASTC [9], and PVRTC [10]. Among them, ETC1/2 is the most widely used format in the Android platform. According to Google's analysis in September 2020 [11], 99% and 87% of Android devices support ETC1 and ETC2, respectively. Therefore, efficient ETC1/2 encoders can be useful for developing or running Android apps.

The Unity game engine [12] employs three different ETC1/2 encoders to provide flexible speed-quality tradeoffs: etcpak [13], ETCPACK [14], and Etc2Comp [15]. etcpak is a fast encoder, but it results in lower compression quality than the other slower encoders (ETCPACK and Etc2Comp). The reason for this is that its high speed is not only due to highly optimized parallel programming, but also because it restricts encoding modes (up to and including version 0.7) and trials. Nah [16] reported that

*Corresponding author
e-mail: jaeho.nah@smu.ac.kr (Jae-Ho Nah)

etcpak 0.7 was two to three orders of magnitude faster than ETCPACK and Etc2Comp with their fastest settings, respectively, and the PSNR differences between etcpak 0.7 and the others were approximately one to two dB. To increase the compression quality, QuickETC2 [17, 16] adds high-speed ETC2 T/H-mode encoding logic to etcpak 0.7. Using these additional modes can increase encoding time, but the heuristic selector of QuickETC2, which is based on the early compression-mode decision scheme, compensates for this additional encoding cost. As a result, the QuickETC2 technique improves the quality, speed, or both of etcpak 0.7 and has finally been integrated into etcpak 1.0.

However, etcpak 1.0 sometimes produces compression artifacts, such as block artifacts. One of the main reasons for these artifacts is the luma-based approximations used by etcpak. When calculating and comparing block errors, etcpak uses the luma values of candidate colors instead of the separated RGB values. Furthermore, the early compression-mode decision scheme in QuickETC2 also relies on the luma differences of each block to determine further processing modes. Although these approximations can reduce compression costs, they can also lead to incorrect decisions.

To address the above issues, we propose a set of improvements to etcpak 1.0. Firstly, we calculate RGB weights for a block with a dominant color channel and partially use modified luma values to enhance error calculations, mode selections, and clustering. Secondly, we perform compression in the ETC2 modes (T, H, and Planar) additionally for blocks compressed with the ETC1 individual mode, and we standardize the error metrics between the modes. This additional calculation reduces block artifacts caused by 4x2 or 2x4 subblocks. Finally, we make minor modifications to the T/H compression logic in QuickETC2 and the solid-color check function in etcpak to minimize quantization artifacts.

We performed experiments with the 93 test images in different categories (Figures 1, 2, and 3) and compared QuickETC-HQ with other encoders (etcpak 1.0 and ETCPACK 2.74). RGBA images in the test set were compressed using both ETC2 RGB and EAC. The results exhibit that our approach still keep real-time rates of etcpak and improves the compression quality to a level comparable to ETCPACK's fast mode.

The structure and organization of the remainder of this paper are as follows. Section 2 will summarize related papers and open-source projects, including ETC1 [7], ETC2 [8], etcpak [13], and QuickETC2 [17, 16]. Section 3 will describe the new techniques proposed in this paper, along with their implementation details. In Section 4, we will present quantitative and qualitative analyses of our experimental results. Finally, in Section 5, we will conclude this paper.

2. Background

The ETC1 codec began with PACKMAN [18], which compresses a 2x4 block by combining a base color per block and a luminance modifier per pixel. Later, iPACKMAN [7] extended PACKMAN in several ways to improve image quality. The main difference between PACKMAN and iPACKMAN is the block size. Instead of the 2x4 block used in PACKMAN, iPACKMAN uses a 4x4 block that consists of two 2x4 (vertical) or 4x2 (horizontal) subblocks. The base colors of the two sub-blocks can be coded either individually (two RGB444 colors) or differentially (RGB555 base color + RGB333 offset). The intensity (luminance) modifier table of iPACKMAN is also different from that of PACKMAN due to their different block sizes.

Ericsson contributed the iPACKMAN codec to the Khronos group under the name ETC1, which has become the most widely used format for mobile platforms supporting OpenGL ES [20] and Vulkan [21]. ETC2 [8] further improves image quality by adding three new modes: T, H, and Planar. T- and H-modes are suitable for blocks with distinct luma and/or chroma variance, and two base colors are calculated, and two more colors are obtained by modulating them. The number of colors per block is limited to four, but the T- or H-modes can represent different partition patterns, similar to BC7 [6] or ASTC [9]. The Planar mode aims to remove banding artifacts caused by the color quantization of ETC1 by storing three edge colors of a block and interpolating them for the remaining pixels. This mode is best suited for blocks where colors change smoothly.

ETCPACK [14] is the reference encoder for ETC1/2. When compressing an ETC2 block, this encoder generates blocks compressed with all the ETC1/2 modes and selects the best block among them by comparing the errors

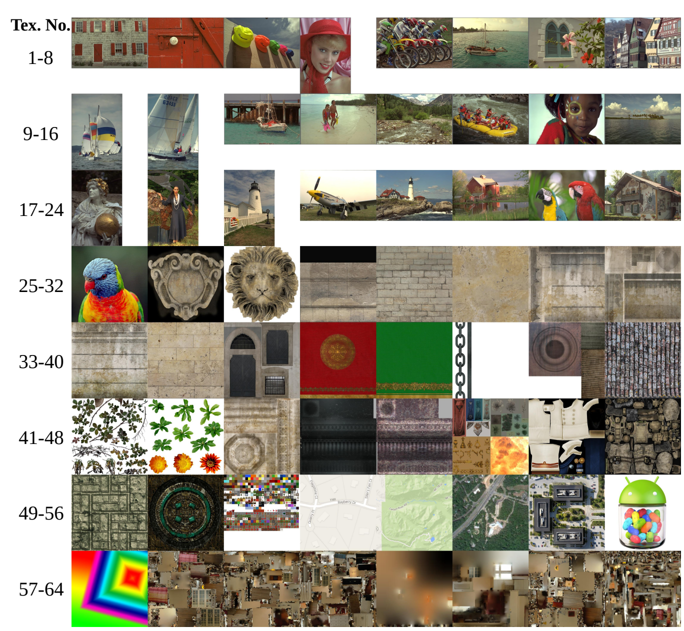

Fig. 1. 64 test images used in both QuickETC2 and our experiments. This set includes various types of textures: photos (No. 1-25), game textures (No. 26-51), GIS map data (No. 52-55), synthesized images (No. 56-57), and captured images from the real world (No. 58-64). If you need detailed descriptions of the images, please refer to the supplemental document of QuickETC2 [16].

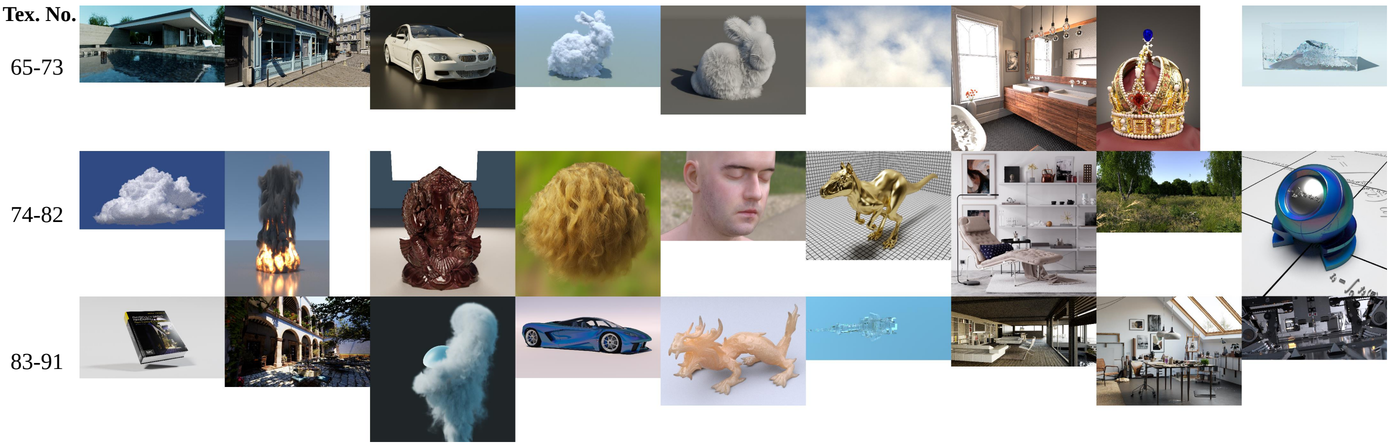

Fig. 2. 27 test images rendered by pbrt-v4 [19] to evaluate the performance of a texture codec in streaming rendered content. For convenience, we will refer to these images as No. 65 to No. 91.

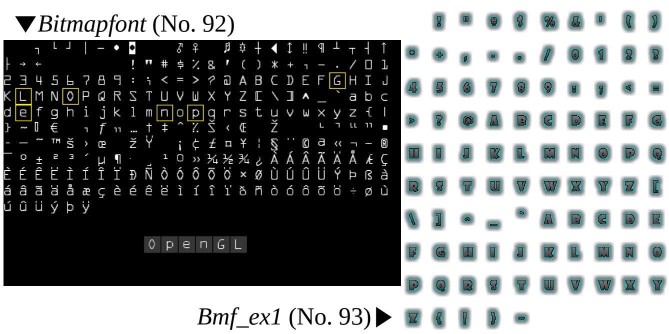

Fig. 3. Two test textures for font rendering. Each contains white text on a black background (No. 92) and colored text on a white background (No. 93), respectively.

of the blocks. Therefore, ETC2 compression takes longer than ETC1 compression. McAnlis [22] reported that the ETC2 encoding time using the Mali Texture Compression Tool [23], which internally includes ETCPACK, was several minutes to 1.3 hours per texture when the slow mode was selected.

The etcpak [13] encoder was released to address the long encoding time of ETC2 compression. Its primary goal is a high speed, achieving several orders of magnitude speed-up compared to other encoders. This speed-up is possible due to efficient work distribution and highly optimized compression parts using SSE and AVX2 intrinsics. However, earlier versions of etcpak (up to and including version 0.7) did not support the T- and H-modes, which are more complex than the ETC1 and Planar modes. Additionally, this encoder restricts the number of ETC1 encoding trials, only utilizing the average color of sub-blocks. Both the limited mode support and the limited encoding trials can degrade the image quality.

QuickETC2 [17, 16] addresses the quality degradation of etcpak 0.7. QuickETC2 supports high-speed T-/H-block compression to reduce block artifacts. Additionally, it optimizes encoding performance by deciding the compression mode(s) early based on the luma differences in each block, thereby avoiding unnecessary tests. QuickETC2 achieves up to a 3 × speed-up and up to a 1 dB higher PSNR compared to etcpak 0.7. Recently, a QuickETC2 patch has been integrated into etcpak 1.0.

3. Our Approach

3.1. Analysis of etcpak's compression artifacts

Even though the additional modes in QuickETC2 efficiently reduce many block artifacts in etcpak 0.7, there is still a quality gap between etcpak 1.0 and other high-quality encoders (e.g., ETCPACK, Etc2Comp, etc.) with their fast modes. Therefore, we need to analyze the compression artifacts of etcpak first. For the analysis, we executed etcpak 1.0 with 64 test images in the QuickETC2 paper (Figure 1), 27 test images rendered by pbrt-v4 [19] (Figure 2), and two images containing pure text (Figure 3). Using the second test set representing various rendering effects can be helpful to judge compression quality in that streaming case [1]. Rendered images similar to Figure 2 are not mapped into 3D objects; instead, they can be compressed and decompressed using a texture codec for video streaming.

According to our analysis, etcpak 1.0 exhibited several artifact patterns, as shown in Figure 4. We will analyze the reasons for each artifact type in the following descriptions. To better understand the following description, please refer to the flow chart of etcpak's ETC2 compression, illustrated in Figure 5.

The first artifact type is posterization, which appeared in Atlas. When all the color values in a block are the same, etcpak's solid-color check function quickly determines the block color through RGB555 quantization and skips further processing. This technique had been used for the ETC1-only mode in etcpak 0.7, but etcpak 1.0 forces this early filtering for both ETC1 and ETC2 encodings. This filtering accelerates the compression of background parts, but its simple quantization can bring out unnecessary background boundaries. If most blocks in a texture

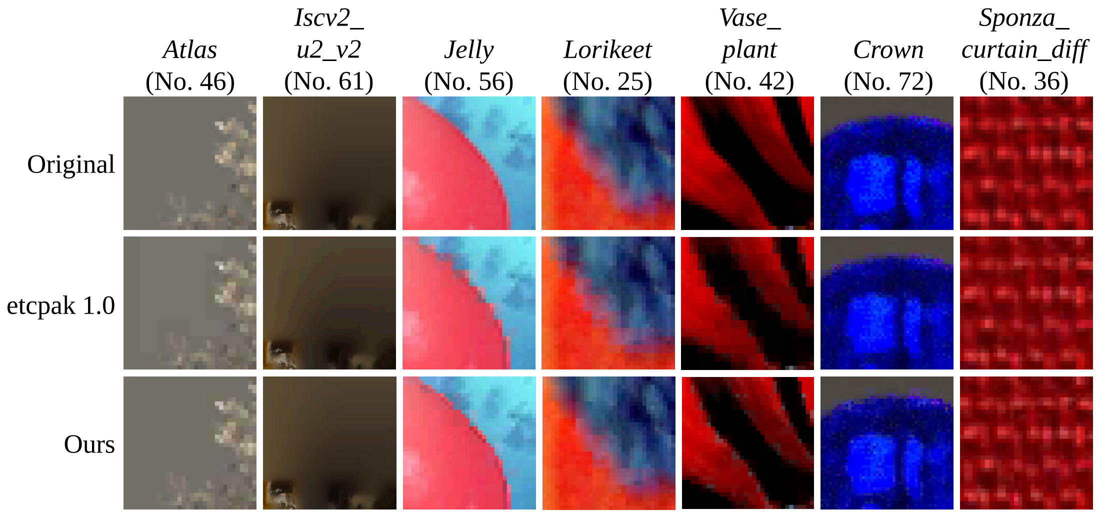

Fig. 4. Artifact patterns appeared in the images compressed by etcpak 1.0: Posterization, banding, block artifacts, and blurring. These artifacts could be alleviated after applying our approaches as shown in the third row.

are filled with solid colors, this posterization can get worse and visible banding can appear, as shown in Iscv2_u2_v2.

The second type is block artifacts, which are related to the conversion from RGB to luma in Equation 1 [24].

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

etcpak 1.0 extensively utilizes the above luma-conversion equation for error calculations, early compression-mode decisions, and clustering in the T-/H-mode compression, thereby reducing computational overhead. However, this strategy can sometimes generate several artifacts with different aspects and reasons, as shown in Jelly, Lorikeet, and Vase_plant.

In Jelly, etcpak 1.0 shows blocky edges with color bleeding, which can be handled efficiently by the T- or H-modes in ETCPACK. However, etcpak 1.0 selects the ETC1 mode instead of T- or H-modes, resulting in the artifacts. The reason is that QuickETC2's early compression-mode decisions and its clustering during the T-/H-mode compression utilize luma values of each texel. The luma values of light pink and light blue colors in the Jelly texture are similar, so the luma differences of each block are not high enough for T- or H-modes to be selected either in the early-compression mode decisions (due to the luma differences being less than T_3=0.38 in QuickETC2 [16]) or in the encode selector (due to low-quality T-/H-mode compression).

In Lorikeet, the ideal mode for the boundary region is Planar to express soft gradients. However, the early-compression mode decision scheme does not consider the Planar mode because the blocks' luma differences are not less than the predefined threshold value (T_2=0.09 in QuickETC2 [16]). As a result, etcpak 1.0 selects the ETC1 mode for the region, and the softness of the region was lost.

The block artifacts in Vase_plant are more visible than in the previous cases. The red flower consists of high-contrast black and red colors, and due to the absence of green and blue colors in the region, the luma values of each pixel are always low. As a result, luma-based approximations fail to maintain compression quality in those cases. For example, the ETC1 mode used in the boundaries results in block artifacts because of improper calculation of distance errors. Additionally, the Planar mode used in the inner part of the petals also amplifies block artifacts. The blue jewel in Crown has very low luma contrasts, so we can see blocky boundaries for a similar reason to Vase_plant.

The fourth artifact type is blurring. As shown in Sponza_curtain_diff, compressing a block with a dominant color channel (red or blue) could result in a loss of detail. The reason for the blurring in this region is the low luma values, similar to the case of Vase_plant. However, Sponza_curtain_diff and Vase_plant show different artifact types because the blurred part in the former has low contrast and no diagonal edges.

3.2. Our modifications to minimize the compression artifacts

The artifacts described in Section 3.1 are mainly related to luma-based approximations and encoding-mode skipping. To solve these problems, we modify all stages in etcpak 1.0, as illustrated in Figure 5. However, we have not considered extending search spaces to find base colors in each compression mode, as described in THUMB [25] and ETCPACK [14], because this strategy can exponentially increase compression time.

First, we use the Planar mode to compress solid-color blocks and avoid posterization in the background. When encoding solid-color blocks, the check-solid-color function uses RGB555 quantization and skips further processes (shown by the gray arrow in Figure 5), reducing compression costs. However, RGB676 encoding in the Planar mode preserves color much better in these solid-color cases. It is no surprise that in such situations, 9 bits for dRdGdB in an ETC1 sub-block are set to zero, while the Planar mode

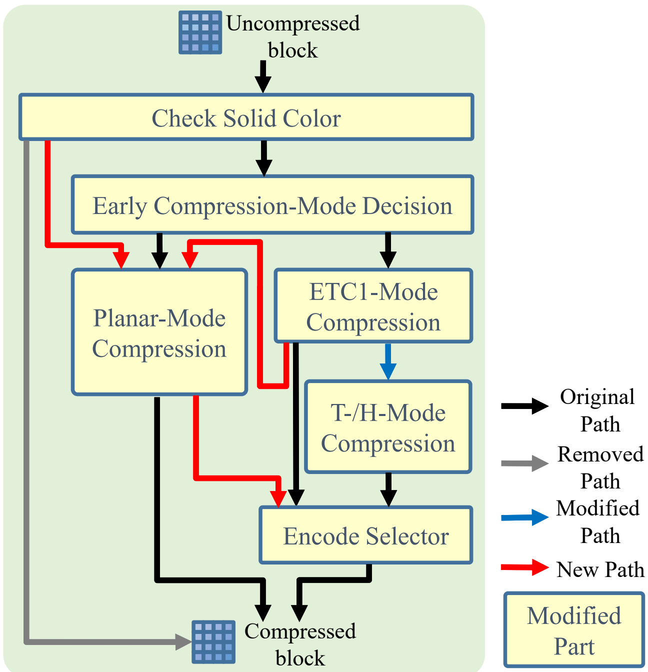

Fig. 5. A flow chart of the ETC2 RGB compression in etcpak 1.0 (ProcessRGB_ETC2()). We have modified all the sub-functions in the process.

can allocate more bits for RGB quantization compared to ETC1.

Therefore, we exclusively use the Planar mode in the cases to minimize quantization artifacts shown in Figure 4. We bypass the early compression-mode decision stage (indicated by the leftmost red arrow in Figure 5) in this case because there is no need to select a different mode in such instances.

Second, we apply variable color weights to blocks with a dominant color channel when calculating the luma values of the blocks. Equation 1 specifies these color weights as 0.299, 0.587, and 0.114, chosen based on the human eye's sensitivity to green light. Ström and Akenine-Möller [7] demonstrated that the perceptual error metric using these weights more accurately represents edges between different color areas than the error metric with equal RGB weights.

However, the fixed color weights for all blocks may not be suitable for all cases. In cases where a block has a dominant color channel, the block's colors may appear reddish, greenish, or bluish. A block with a high level of green may not exhibit any issues after compression with the color weights specified in Equation 1 due to the high value assigned to the green channel (0.587). However, the weights can produce suboptimal results for predominantly reddish or bluish blocks, as explained in Section 3.1.

To identify cases that require changing the color weights, we have added an RGB weight calculation function before the luma calculation in the early compression-mode decision stage. The function calculates new color weights through the following steps:

1. For each color channel, we calculate 'bgrRange' by subtracting the minimum value from the maximum value among 16 pixels in a block.

2. We then compare 'bgrRange' values for each channel and identify the maximum value. The corresponding channel for this maximum 'bgrRange' is stored in 'maxBgrCh,' while the maximum 'bgrRange' value itself is stored in 'maxBgrRange.' These two values can be used for additional T-/H-mode compression later.

3. To obtain 'totalBgr' for each channel, we sum up the color values of 16 pixels in that channel. Additionally, we calculate 'sumOfTotalBgr' by adding up 'totalBgr' values for all channels.

4. We evaluate the values of 'bgrRange' in each channel and the ratios of 'totalBgr' to 'sumOfTotalBgr.' The former represents contrast in each channel, while the latter indicates the proportion of each channel's contribution to the overall color information.

4.1. If the 'bgrRange' of the red or blue channel exceeds a threshold value (currently set at 1/5), and the 'totalBgr' of the red or blue channel accounts for a significant proportion (currently higher than 2/3) of 'sumOfTotalBgr,' we recalculate the weights. We mix the original weights of each channel from Equation 1 with the ratios of 'totalBgr' for each channel to 'sumOfTotalBgr' using a 1:1 ratio.

4.2. Otherwise, if the conditions in 4.1 are not met, we retain the original color weights as stated in Equation 1.

The unmodified or modified weights are passed to the stages for further ETC2 encoding, allowing us to select appropriate compression modes and color candidates for reddish or bluish blocks. Figure 6 illustrates an example.

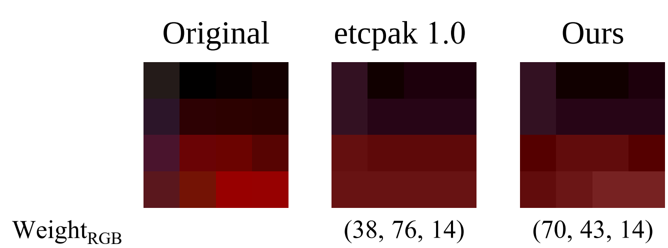

Fig. 6. RGB weights of a 4x4 block calculated using etcpak 1.0 and ours. In etcpak, the weight range is [0, 128], so the RGB weights in Equation 1 (0.299, 0.587, and 0.114) are represented as 38, 76, and 14, respectively. However, because the RGB difference of the uncompressed block in the above image is (151, 27, 45), our approach identifies this block's dominant color as red and increases the weight of the red channel from 38 to 70. As a result, our approach can better preserve the reddish shape thanks to more accurate per-pixel distance values.

Third, we additionally conduct compression using the T-/H-/Planar modes to an ETC1 block compressed by the individual mode. Unlike the differential mode, the individual mode uses two different RGB444 base colors. As a result, blocks compressed in this mode may exhibit 4x2 or 2x4 block artifacts, particularly along the boundaries between different color areas. The additional compression modes can alleviate this issue. The red and blue arrows started from 'ETC1-Mode Compression in Figure 5 indicate the newly added or modified paths for this additional process.

We perform the additional T-/H-mode compression using a modified clustering method only when the luma differences in a block are not high since luma-based clustering in QuickETC2 may not efficiently generate appropriate clusters in the case. Instead, we use values of the dominant color channel for clustering to avoid that case. If the difference in the dominant color channel of a block is low (less than T_2=0.09 in QuickETC2 [16]), we do not execute the additional T-/H-mode compression. The reason is the expected quality improvement for a low-contrast block may be minimal.

In contrast to the original implementation in etcpak 0.7, QuickETC2 omits error calculations in the Planar mode because this mode is selected only in low-contrast regions. However, this assumption does not hold in the case of multiple modes. Therefore, we need to compare the compression results from the Planar, ETC1, and T-/H-modes in the encode selector, as shown in the red paths in Figure 5. We calculate the block error only if the Planar-mode compression function is additionally called after ETC1 compression. Otherwise, we skip the error calculation and directly encode the block as implemented in QuickETC2.

Fourth, to ensure a fair comparison between the compression results from the ETC1 and T-/H-modes, we perform additional error calculations in the encode selector. QuickETC2 used a slightly different error calculation method for its T-/H-mode compression compared to the ETC1 mode in etcpak 1.0. In the ETC1 mode, the luma values of the candidates are calculated first and then used for error comparisons. There is only one base color per subblock, and the distance modifier per pixel only affects the brightness of each pixel. This is a clever strategy to reduce overhead of error calculations. In the T-/H-mode compression, differences between the original and compressed pixels are calculated for each color channel and combined according to color weights for luma conversion to obtain an error value. Because a difference can be a negative number, the T- and H-modes use the absolute value in contrast to the ETC1 mode. As a result, the errors calculated in the ETC1 mode can be lower than those in the T-/H-modes. This inconsistency can be solved by changing the ETC1 compression logic, but it leads to increased overhead. Therefore, we enable the additional error correction logic in the encode selector only if the current block contains a result from the T-/H-mode. If a variable color weight is applied to the current block, we disable this additional error calculation to prevent leaning too much toward the T- or H-mode.

Fifth, we recommend some minor updates to the T-/H-mode compression in QuickETC2. The first update involves using the colors in four vertices of a block (1st, 4th, 13th, and 16th pixels) for base color calculation, in addition to the colors in pixels with the minimum and maximum luma values. This change is intended to alleviate block artifacts that become more visible when the colors of pixels at the block's vertices are substantially different from those in neighboring blocks. Figure 7 demonstrates the advantages of this vertex-weighted base-color calculation. The second update involves forcing the T-mode if the number of pixels in the smaller cluster is less than four. This change enables a cluster of more than 13 pixels to be expressed with three palette colors instead of one or two. Although these methods may not reduce errors, they can help reduce visible block artifacts. Finally, when selecting the H-mode or the additional T-mode with the ETC1 individual mode, we opt for a more conservative start distance index for error calculations, two steps earlier than in QuickETC2. This approach slightly reduces performance but enhances quality.

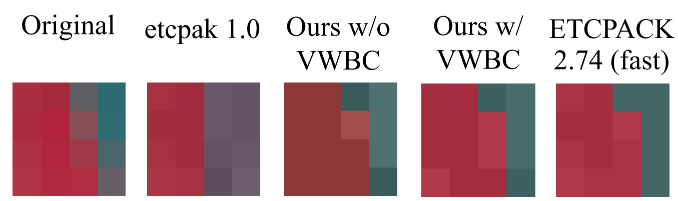

Fig. 7. A case where our vertex-weighted base-color (VWBC) calculation is advantageous. The H-mode in our method preserves the shape of the block compared to the ETC1 mode selected by etcpak 1.0. However, without VWBC, the colors after compression are less vibrant than the original colors. The VWBC approach resolves this issue, resulting in compressed colors that are closer to the original.

4. Experimental Results

In this section, we will analyze how a combination of the modifications introduced in Section 3.2 affects quality and performance. Our approaches are built on etcpak 1.0 [13] integrated the QuickETC2 patch [16], and we will compare our QuickETC2-HQ method to etcpak 1.0 and ETCPACK 2.74 [14]. When executing ETCPACK, we used its default setting, the fast mode with the perceptual error metric.

4.1. Experiment setup

For our experiments, we utilized the test image set used in QuickETC2 [17, 16] (Figure 1), the set rendered by pbrt-v4 [19] (Figure 2), and the two text textures in Figure 3. The quality-comparison metrics we used are ꟻLIP [26], PSNR and the standard luma-based SSIM [27]. The ꟻLIP values range from 0 to 1. We used ꟻLIP version 1.2 with a default value of p=67 pixels per degree for our experiments. We obtained PSNR and SSIM values from opencv-python 4.6.0.66 and ImageMagick 7.1.1-8, respectively.

For the performance comparison, we used a desktop computer equipped with an Intel Core i7 12700 CPU (8 performance-cores, 4 efficient-cores, and total 20 threads), 32GB DDR4-3200 RAM, a 2TB SSD, and Windows 11. We measured the execution time using each encoder's timing function. The time spent on the other operations, such as file I/O and PNG decompression, was not included in the timing values.

4.2. Comparison analysis

Table 1 summarizes the experimental results. Our QuickETC2-HQ method bridges the quality gap between etcpak 1.0 and ETCPACK. Compared to the baseline, etcpak 1.0, our approach achieved 9.5% lower ꟻLIP and increased PSNR and SSIM values. The average ꟻLIP value of QuickETC2-HQ is now comparable to that of ETCPACK, with only a 0.0023 difference between them. However, QuickETC2-HQ is two to three orders of magnitude faster than ETCPACK (the speed-up factor depends on the number of threads). Thus, QuickETC2-HQ can be practically used to reduce texture encoding time in a production build.

Table 1. Experimental results. All the values in this table were averaged over the 93 test textures. Upwards (↑) and downwards (↓) arrows indicate that lower and higher values are better, respectively.

Quality comparison			
Encoder	ꟻLIP↓ (mean)	PSNR↑ (dB)	SSIM↑
etcpak 1.0	0.0476	36.98	0.954
Ours	0.0431	37.61	0.961
ETCPACK 2.74 (fast)	0.0408	38.56	0.968

Performance comparison (MPixels/s↑)		
Encoder	single-threading	multi-threading
etcpak 1.0	345	2266
Ours	212	1390
ETCPACK 2.74 (fast)	0.94	0.94

The reason why ꟻLIP is more favorable to our approach than PSNR and SSIM is as follows. First of all, a PSNR value with equal weighting factors for the three RGB channels does not properly reflect the advantage of our variable color weights mentioned in Section 3. For example, our approach decreases the mean squared error (MSE) value in the red channel in Sponza_curtain_diff and provides sharper details of the red curtain. When calculating the PSNR value, the increased MSE values in the green and blue channels offset the decreased MSE value in the red channel, even though the red curtain makes up most of the image. Similarly, although our approach resulted in a 2% lower SSIM value than etcpak 1.0 in the test image, this result is erroneous as with Nilsson and Akenine-Möller's analysis [28]. Moreover, we found that SSIM often failed to capture the reduction in block artifacts, as described in the QuickETC2 paper [16]. In contrast, ꟻLIP measures both color and feature differences and can accurately detect visual enhancements resulting from our additions, including variable color weights. Therefore, we will use ꟻLIP values in further in-depth analysis.

Figure 8 presents a detailed quality comparison of our approach with etcpak 1.0 and ETCPACK. In Kodim07, our additional Planar-mode compression improves the

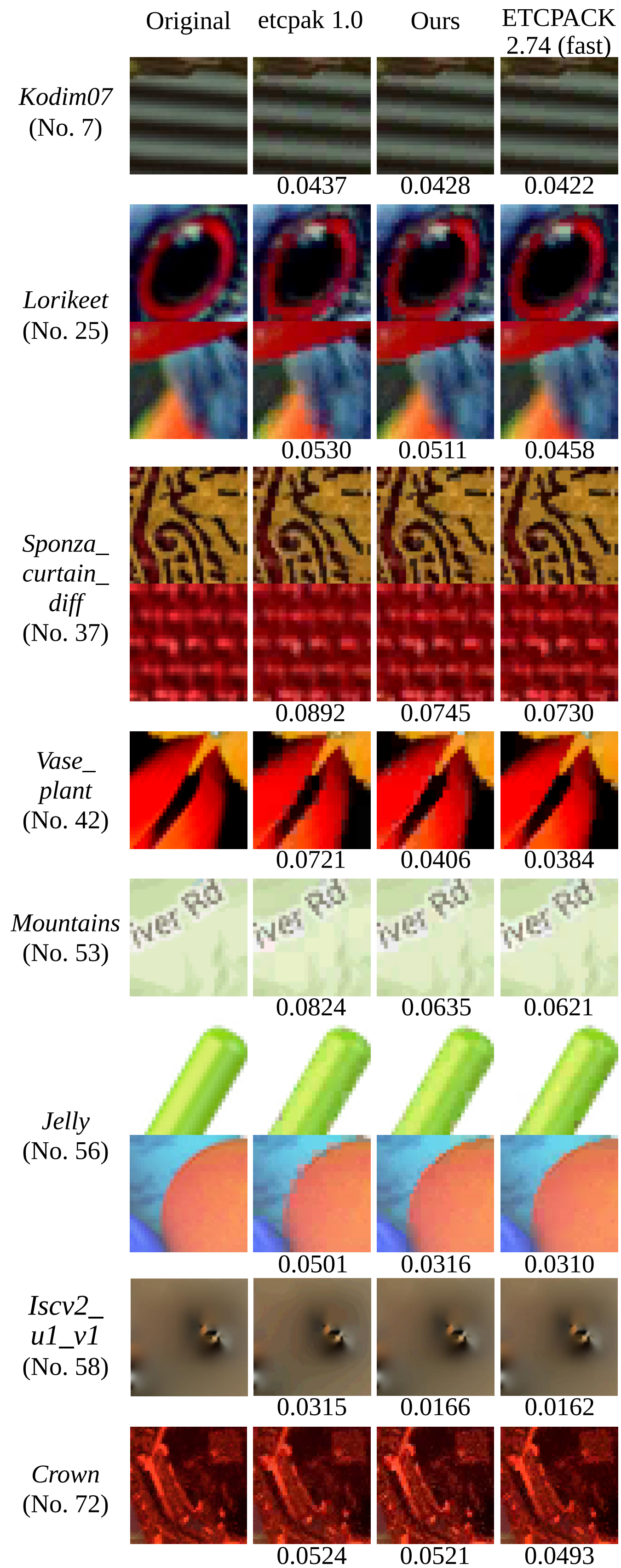

Fig. 8. Quality comparison. The number below each image is the ꟻLIP value (lower is better). QuickETC2-HQ alleviates various issues remained in etcpak 1.0, such as aliasing, color shifts, shape distortion, color bleeding, block artifacts, blurring, posterization, etc. Please zoom in the image to distinguish differences.

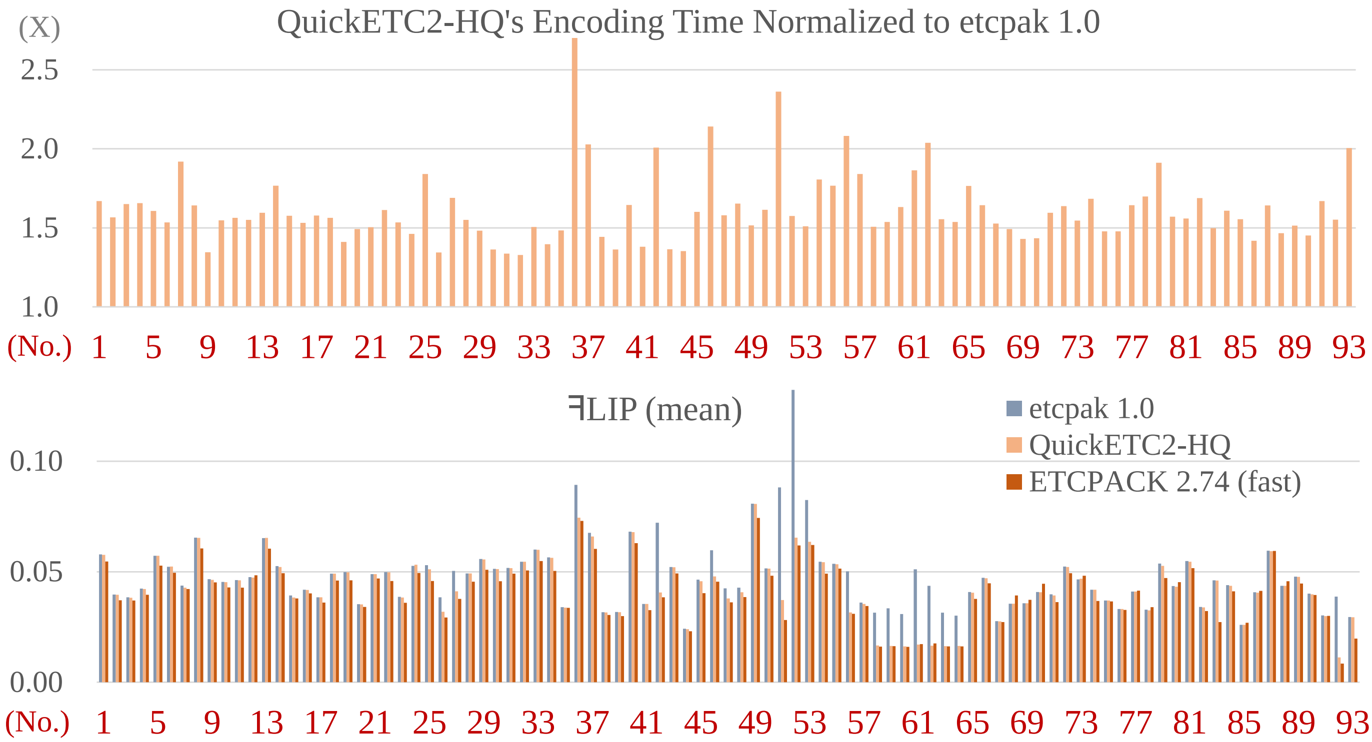

Fig. 9. Comparison results in each texture. Top: Performance comparison using the encoding-time overhead of QuickETC2-HQ compared to etcpak 1.0. Bottom: Quality comparison using ꟻLIP values between etcpak, QuickETC2-HQ, and ETCPACK. QuickETC2-HQ requires 1.6 × overhead over etcpak on average, but it suppresses etcpak's increases in ꟻLIP values.

quality by smoothing aliased edges and preventing color shifts that appeared in ETC1 blocks, as ETCPACK does. In Lorikeet, QuickETC2-HQ preserves the eye's ellipse shape, removes the color-bled edge on the bill, and softens blocky boundaries. In Sponza_curtain_diff, etcpak's results show greenish-gold colors and blurred weaving patterns, but our approach restores them to the original. ETCPACK sharpens the original image. In Vase_plant, QuickETC2-HQ removes a significant amount of block artifacts between the red and black regions, although it does not outperform ETCPACK. In Mountains, the posterization artifact presented in etcpak 1.0 no longer appears in our approach, and the text is more visible due to the preservation of the white text borders. In Jelly, our method dramatically improves the compression quality. In contrast to the others, ours prevents the black color in the transparent background from penetrating the pea-green antenna. Furthermore, our additional T-/H-mode compression removes many block artifacts with color bleeding in etcpak. ETCPACK generally handles the boundary between two different colors better than our approach. In Iscv2_u1_v1, ours smooths banding artifacts in etcpak, similar to ETCPACK. In Crown, etcpak generates blocky boundaries and slightly faded colors, and ours alleviates these artifacts well. The results between ours and ETCPACK are similar in terms of shape preservation, but ETCPACK expresses vivid colors in the original images better.

Figure 9 presents the performance and quality comparison in each texture. The encoding time of QuickETC2-HQ normalized to that of etcpak 1.0 ranges from 1.3 × to 2.7 ×, with an average of 1.6 ×. Although the additional computations in our approach increases the encoding overhead, it also improves the compression quality, as indicated by the ꟻLIP values. When the quality difference between etcpak 1.0 and ETCPACK 2.74 with the fast mode is small, the impact of QuickETC2-HQ is not significant. However, when etcpak 1.0 exhibits visible artifacts, our approach corrects most of them, resulting in lower ꟻLIP values that are closer to those of ETCPACK.

4.3. Ablation study

For the ablation study, we selected six representative images in the test set and measured the ꟻLIP values and performance on different settings in each image. The purpose of this study is to determine how each method introduced in Section 3.2 affects compression quality and performance. We tabulate the results in Table 2.

Firstly, the Planar mode for solid-color blocks (a) only affects results if some part of the texture contains solid-color blocks. Thus, the results of Kodim07, Lorikeet, Sponza_curtain_diff remained the same before and after applying the method to the baseline. While this method could reduce the ꟻLIP values of RGBA textures such as Vase_plant and Jelly, there was no noticeable visual improvement because errors in the transparent regions with

Table 2. Ablation study. (a), (b), (c), (d), and (e) in this table refer an addition of each method introduced in Section 3.2, respectively: (a) forcing the Planar mode for solid-color blocks, (b) variable color weights, (c) additional T-/H-/Planar mode compression, (d) additional error calculations, and (e) the updates for the T-/H-mode compression. The baseline is etcpak 1.0.

	ꟻLIP (mean) ↓						Encoding time (ms) ↓					
Image	Baseline	+(a)	+(b)	+(c)	+(d)	+(e)	Baseline	+(a)	+(b)	+(c)	+(d)	+(e)
Kodim07	0.0437	0.0437	0.0433	0.0433	0.0428	0.0428	0.35	0.35	0.43	0.60	0.64	0.68
Lorikeet	0.0530	0.0530	0.0523	0.0521	0.0511	0.0511	0.28	0.28	0.34	0.46	0.49	0.52
Sponza_curtain_diff	0.0892	0.0892	0.0753	0.0753	0.0744	0.0745	2.3	2.3	3.6	5.9	6.0	6.4
Vase_plant	0.0721	0.0412	0.0407	0.0407	0.0406	0.0406	0.40	0.47	0.62	0.75	0.78	0.80
Jelly	0.0501	0.0333	0.0333	0.0332	0.0318	0.0316	0.16	0.19	0.28	0.31	0.33	0.34
Iscv2_u1_v1	0.0315	0.0167	0.0167	0.0166	0.0166	0.0166	14.1	15.0	21.0	21.9	22.1	22.3

the alpha value of zero were mainly reduced. On the other hand, the method is highly efficient for removing posterization and banding artifacts in RGB textures, as demonstrated by the results for Iscv2_u1_v1.

Secondly, the variable color weights (b) always affects the encoding speed because the weights need to be recalculated for all blocks. This approach is especially effective for reddish or blueish textures, such as Sponza_curtain_diff.

Thirdly, both additional T-/H-/Planar mode compression (c) and additional error calculations (d) need to be used together to be effective. This is because the encode selector can only choose one block among several candidate blocks compressed on different modes. The encoding overhead depends on the ratio of ETC1 to ETC2 blocks.

Finally, the updates for T-/H-mode compression (e) can improve compression quality on some blocky diagonal edges. As a result, this method reduced the ꟻLIP value in Jelly, the most problematic texture in the QuickETC2 test set. However, if there are no noticeable block artifacts in a texture, the method has little to no effect on the compression result.

4.4. Failure cases

Although QuickETC2-HQ solves a number of quality issues appearing in real-time ETC2 encoding, this method sometimes fails to find optimal results. First, the error-based mode selection does not always guarantee the perceptually best quality. As shown in the first row of Figure 10, the ETC1 blocks selected by etcpak 1.0 express smoothness in the pink region better than the T- or H-blocks selected by the other encoders. If a novel error-judgment method is utilized during the encoding process instead of MSE, we think this problem may be alleviated. Of course, this can be a fundamental modification in a block-based codec and can lead to some unexpected compression results. Thus, we would like to leave this investigation as a future research topic.

Second, some difficult cases need an exhaustive search to get high-quality ETC2 compression. In the second row of Figure 10, the twinkling gems in Crown generate spatially-incoherent colors, and some of the shiny pixels are discolored after ETC2 compression, except for ETCPACK with the slow mode. The third row of Figure 10 shows a similar example. While ETCPACK with the slow mode made a flawless result, the others generated color bleeding in and out the black letter borders. Even additional T-/H-blocks in our approach worsened the color bleeding because inner grey texels and outer bright emerald-blue texels were grouped into the same cluster due to their luma similarity. Using the slow mode in ETCPACK can solve these color distortion and bleeding, but this is possible at the expense of compression speed; our encoder can compress each texture in Figure 10 within 1 ms, but ETCPACK with the slow mode requires about 1 minute per texture.

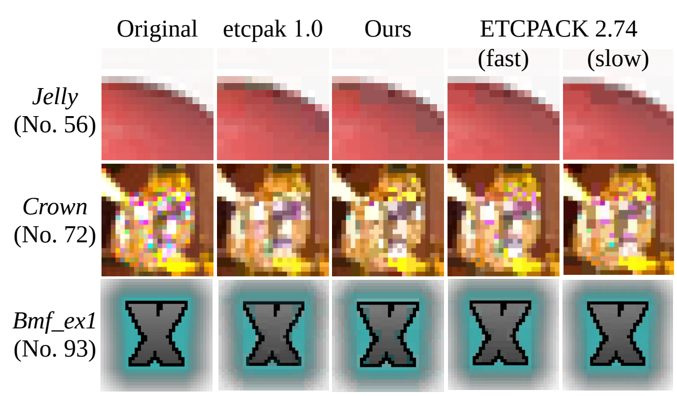

Fig. 10. Failure cases. Our approach may exacerbate block artifacts (Jelly), distort colors (Crown), or bleed colors (Bmf_ex1) during the additional ETC2 encoding process.

5. Conclusions and Future Work

In this paper, we introduced QuickETC2-HQ, a set of improvements to etcpak 1.0, a state-of-the-art texture encoder. QuickETC2-HQ includes several techniques, such as Planar-mode compression of solid-color blocks, variable color weights, selective execution of additional T-/H-/Planar mode compression, vertex-weighted base-color calculations, and more accurate error comparisons. Combining these techniques reduces compression artifacts, such as blurring, posterization, and block artifacts. QuickETC2-HQ built on etcpak 1.0 provides a significant speed advantage over ETCPACK 2.74 with the fast mode while delivering comparable compression quality to that. Therefore, we believe QuickETC2-HQ can be an excellent option for mobile app developers who need high compression quality and performance.

There are many avenues for future work. First, QuickETC2-HQ has a few parameters, and more optimal parameter values for QuickETC2-HQ may exist for better tradeoffs between quality and performance. With additional consideration of masking effects [29], we will be able to apply different parameters to different texture types. Second, if we consider not only the luma (Y) space but also the CbCr or HSV space for compression, we will be able to expect more quality improvements. Third, we would like to extend our work to other texture encoders in different platforms, such as the Betsy GPU texture encoder [30] and the H-ETC2 CPU-GPU hybrid encoder [31]. We would also like to explore how to apply some of our methods to other texture formats, such as ASTC [9] and BC7 [6]. Finally, the concept of variable color weights can be used for other image-processing applications. Because global color weights for luma conversion have been widely used in the image-processing community, we expect our locally variable approach may lead to new research topics.

Acknowledgement

This research was supported by a 2021, 2022 Research Grant from Sangmyung University (2021-A000-0156, 2022-A000-0255). We appreciate the reviewers for their thorough reviews to improve our manuscript, and we also thank the authors of etcpak who have opened their source code to the public. Image courtesy of Kodak, Simon Fenney, Crytek, UNC GAMMA Lab, Spiral Graphics, Vokselia Spawn, Cesium, Google, fhernand of Sketchfab, Hamza Cheggour, Fred C. M'ule Jr, Duc Nguyen, Ron Fedkiw, Mareck, Martin Lubich, Yining Karl Li, Wenzel Jakob, Cem Yuksel, Infinite Realities, USC-ICT light probe image gallery, headus/Rezard, Yasutoshi Mori, Beeple, Karl Li, Guillermo M. Leal Llaguno, Theodore Kim, Stanford Computer Graphics Laboratory, Bernhard Vogl, Florent Boyer, Nolan Goodnight, Joey de Vries, and Warriors of the Cucumber. We plan to release the QuickETC2-HQ patch soon at the following link: https://nahjaeho.github.io

References

[1] Žádník, J, Mäkitalo, M, Vanne, J, Jääskeläinen, P. Image and video coding techniques for ultra-low latency. ACM Computing Surveys 2022;54(11s). doi:10.1145/3512342.

[2] Meerits, S. Real-time 3D reconstruction of dynamic scenes using moving least squares. Ph.D. thesis; Keio University; 2018. URL: https://core.ac.uk/download/pdf/161842143.pdf.

[3] Oom, D. Real-time adaptive scalable texture compression for the web. Master's thesis; Chalmers University of Technology; 2016. URL: https://www.cse.chalmers.se/~uffe/xjobb/Daniel%20Oom.pdf.

[4] Nah, JH, Choi, B, Lim, Y. Classified texture resizing for mobile devices. In: ACM SIGGRAPH 2018 Talks. 2018,doi:10.1145/3214745.3214763.

[5] Kwak, S, Yun, J, Jeong, JY, Kim, Y, Ihm, I, Cheong, WS, et al. View synthesis with sparse light field for 6DoF immersive video. ETRI Journal 2022;44(1):24–37. doi:10.4218/etrij.2021-0205.

[6] Microsoft, . Texture block compression in Direct3D 11. 2018. URL: https://docs.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11.

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

[8] Ström, J, Pettersson, M. ETC 2: texture compression using invalid combinations. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware. 2007, p. 49–54. doi:10.2312/EGGH/EGGH07/049-054.

[9] Nystad, J, Lassen, A, Pomianowski, A, Ellis, S, Olson, T. Adaptive scalable texture compression. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on High-Performance Graphics. 2012, p. 105–114. doi:10.2312/EGGH/HPG12/105-114.

[10] Fenney, S. Texture compression using low-frequency signal modulation. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware. 2003, p. 84–91. URL: https://dl.acm.org/doi/10.5555/844174.844187.

[11] Google, . Target texture compression formats in Android app bundles. 2021. URL: https://developer.android.com/guide/app-bundle/asset-delivery/texture-compression.

[12] Unity Technologies, . Unity user manual (2022.2). 2022. URL: https://docs.unity3d.com/2022.2/Documentation/Manual/class-EditorManager.html.

[13] Taudul, B. etcpak:the fastest ETC compressor on the planet. 2022. URL: https://github.com/wolfpld/etcpak.

[14] Ericsson, . ETCPACK. 2018. URL: https://github.com/Ericsson/ETCPACK.

[15] Google Inc., , Blue Shift Inc., . Etc2Comp - texture to ETC2 compressor. 2017. URL: https://github.com/google/etc2comp.

[16] Nah, JH. QuickETC2: Fast ETC2 texture compression using luma differences. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2020) 2020;39(6). doi:10.1145/3414685.3417787.

[17] Nah, JH. QuickETC2: How to finish ETC2 compression within 1 ms. In: ACM SIGGRAPH 2020 Talks. 2020,doi:10.1145/3388767.3407373.

[18] Ström, J, Akenine-Möller, T. PACKMAN: Texture compression for mobile phones. In: ACM SIGGRAPH 2004 Sketches. 2004, p. 66. doi:10.1145/1186223.1186306.

[19] Pharr, M, Jakob, W, Humphreys, G. Physically Based Rendering, fourth edition: From Theory to Implementation. The MIT Press; 2023. URL: https://pbrt.org/.

[20] Leech, J, Lipcha, B. OpenGL© ES Version 3.0.6 (November 1, 2019). 2019. URL: https://www.khronos.org/registry/OpenGL/specs/es/3.0/es_spec_3.0.pdf.

[21] The Khronos® Vulkan Working Group, . Vulkan® 1.3.250 - a specification (with all registered vulkan extensions). 2023. URL: https://registry.khronos.org/vulkan/specs/1.3-extensions/pdf/vkspec.pdf.

[22] McAnlis, C. Building a blazing fast ETC2 compressor. 2016. URL: https://medium.com/@duhroach/building-a-blazing-fast-etc2-compressor-307f3e9aad99#.acqks0pzct.

[23] Arm Limited, . Mali texture compression tool. 2016. URL: https://developer.arm.com/tools-and-software/graphics-and-gaming/mali-texture-compression-tool/downloads/4-3.

[24] ITU, . BT.601 : Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios. 2011. URL: https://www.itu.int/rec/R-REC-BT.601-7-201103-I/en.

[25] Pettersson, M, Ström, J. Texture compression: THUMB: Two hues using modified brightess. In: SIGRAD 2005 The Annual SIGRAD Conference Special Theme-Mobile Graphics. 016;

Linköping University Electronic Press; 2005, p. 7–12. URL: https://ep.liu.se/ecp/016/002/ecp01602.pdf.

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

[27] Wang, Z, Bovik, AC, Sheikh, HR, Simoncelli, EP. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing 2004;13(4):600–612. doi:10.1109/TIP.2003.819861.

[28] Nilsson, J, Akenine-Möller, T. Understanding SSIM. ArXiv e-prints 2020;doi:10.48550/arXiv.2006.13846].

[29] Griffin, W, Olano, M. Evaluating texture compression masking effects using objective image quality assessment metrics. IEEE Transactions on Visualization and Computer Graphics 2015;21(8):970–979. doi:10.1109/TVCG.2015.2429576.

[30] Goldberg, MN. Betsy GPU compressor. 2022. URL: https://github.com/darksylinc/betsy.

[31] Lee, H, Nah, J. H-ETC2: Design of a CPU-GPU hybrid ETC2 encoder. Computer Graphics Forum (Pacific Graphics 2023) 2023;42(7).