A Practical Encoding Approach for Texture Compression: Combining Multi-Processing and Multi-Threading

Hyeon-ki Lee
Sangmyung University
Jongno-gu, Republic of Korea
gusrl3204@gmail.com

Jae-Ho Nah[*]
Sangmyung University
Jongno-gu, Republic of Korea
nahjaeho@gmail.com

Abstract

High-resolution textures are critical for delivering immersive graphics. In game development, these textures are typically stored in compressed formats and encoded offline. However, when encoding a large number of textures in parallel, the performance benefits of multi-threading can be limited by bottlenecks, including image loading and decoding (e.g., PNG).

In this paper, we experimentally demonstrate that combining multi-processing techniques with multi-threading can achieve more efficient parallel performance compared to using multi-threading alone in the texture compression pipeline. Based on a dataset of 64 textures, our experimental results show that our approach achieves a speedup of 5.76 × in the best case.

CCS Concepts

• **Computing methodologies** → *Image compression.*

Keywords

Texture Compression, Parallel Computing

ACM Reference Format:
Hyeon-ki Lee and Jae-Ho Nah. 2025. A Practical Encoding Approach for Texture Compression: Combining Multi-Processing and Multi-Threading. In *SIGGRAPH Asia 2025 Posters (SA Posters '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3757374.3771436

1 Introduction

A large number of high-quality textures are essential components in various graphics applications including AAA games, and they are typically stored in a compressed format on the GPU to reduce memory bandwidth usage. Currently, widely used standard texture compression codecs include BCn for desktop platforms and ETC1/2 and ASTC for Android/iOS systems. These formats use lossy compression and support real-time decoding and random access on the GPU, while compression is usually performed offline using software-based encoders. Figure 2 shows an analysis of time distribution across different stages of the texture compression pipeline

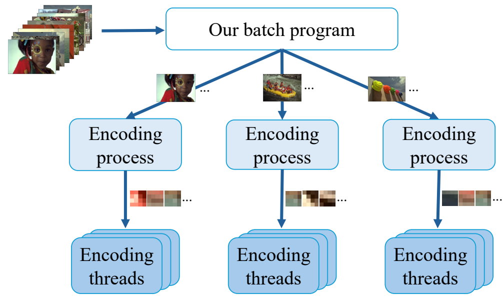

Figure 1: Overview of the proposed parallel texture compression method. Each process handles one texture independently, while multiple encoding threads operate within each process.

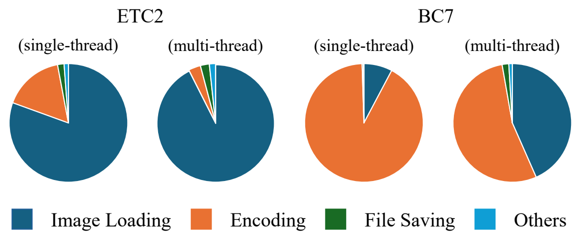

Figure 2: Time ratios for each configuration during ETC2 and BC7 texture compression using etcpak 2.0 were measured over 10 runs, and the median values are reported. Multi-threaded encoding employed 20 threads on an Intel Core i7-12700 CPU. The compression target was "ISCV2_u1_v1.png," an 8K RGB image [Nah 2020].

for the ETC2 and BC7 codecs, using the open-source tool etcpak 2.0 [Taudul 2024], under both single- or multi-threaded environments.

The texture compression process typically consists of three stages: (1) image loading, (2) encoding, and (3) saving the compressed output. Among these, the encoding stage is commonly accelerated using multi-threading, and many studies have presented optimization or acceleration methods for this stage [Lee and Nah 2023; Nah 2020, 2023; Taudul 2024]. As shown in Figure 2, both codecs exhibit relatively low overhead in the file-saving stage. However, for codecs like ETC2, which offer fast encoding speeds, image loading takes up a significant portion of the total processing time. This is because loading textures often involves not only disk I/O but also image decoding steps, depending on the input format; for example, decoding is required when reading data from a PNG file. Such

[*]Corresponding author.

Table 1: The encoding time for ETC2 and BC7 were measured for each combination of process and thread count, where 'Px/Tx' denotes the number of processes (P) and the number of threads per process (T). Each configuration was tested 10 times, and the median encoding time in seconds is reported for three modes: random selection, ascending file size, and descending file size.

	ETC2						BC7					
Modes	**P1/T20**	**P2/T10**	**P4/T5**	**P5/T4**	**P10/T2**	**P20/T1**	**P1/T20**	**P2/T10**	**P4/T5**	**P5/T4**	**P10/T2**	**P20/T1**
Mode 1	5.54	3.05	1.73	1.54	**1.05**	1.17	9.27	6.37	**5.78**	5.87	6.18	9.38
Mode 2	5.56	3.20	1.85	1.64	**1.13**	1.22	9.23	6.50	6.07	**6.06**	6.61	9.92
Mode 3	5.53	2.88	1.54	1.32	**0.96**	1.11	9.22	6.17	5.45	**5.37**	6.16	9.45

overhead becomes more noticeable when the encoding time is short, making the loading stage a potential bottleneck in the pipeline. In contrast, for more complex codecs like BC7, the encoding stage dominates the overall processing time and becomes the main factor affecting throughput. While multi-threading can achieve high performance in single-texture compression, codecs like ETC2—with relatively low overhead in the encoding stage—tend to shift the performance bottleneck to the image loading stage. As a result, idle resources can increase processing time and lower productivity.

2 Proposed method

Modern CPUs have typically multiple cores, enabling flexible parallel execution. From our perspective, based on analysis and these features, combining multi-processing with multi-threading can further improve performance when compressing multiple textures, depending on the file access strategy. In multi-threaded environments, thread synchronization within a single process and variations in per-texture encoding time can cause task imbalance in later pipeline stages; we expect that sorting files in descending order of size reduces this issue by allowing larger files to be loaded and encoded first while smaller files are processed later, resulting in a more balanced workload and reduced idle computational resources.

Based on these observations, we propose a practical parallel encoding approach that leverages both multi-threading and multi-processing to reduce bottlenecks. To this end, we implemented a batch program that compresses multiple files simultaneously, enabling efficient multi-process execution in addition to multi-threading. This is possible due to the multi-input, multi-output characteristics of the texture encoding process. Furthermore, we analyze file access patterns and offer guidelines for efficient large-scale texture processing.

3 Results

Experiments were conducted on a system equipped with an Intel Core i7-12700 CPU (14 cores, 20 threads), 32 GB of RAM, a 2 TB SSD, and Windows 11. Scripts were implemented in Python 3, and texture compression was performed using the open-source encoder etcpak 2.0 [Taudul 2024], which supports BC7 and ETC2 formats. The dataset consisted of 64 textures (.PNG) obtained from QuickETC2 [Nah 2020], with varying resolutions (256×256–8192×8192) and channels (RGB/RGBA). We evaluated six process/thread configurations, ranging from P1/T20 to P20/T1, and measured the total encoding time under three file access modes: (1) random, (2) ascending order by file size, and (3) descending order by file size.

As shown in Table 1, for the ETC2 codec—which has a short encoding time—the P10/T2 configuration achieved the best performance. This is because a large portion of total processing time is spent on image loading rather than encoding, making multi-process configurations that parallelize the loading phase more effective. Regarding file access patterns, P10/T2 achieved speedups of 5.27 $\times$ in Mode 1, 4.91 $\times$ in Mode 2, and 5.76 $\times$ in Mode 3 relative to the baseline P1/T20 configuration. As discussed in Section 2, these results confirm that the image loading order has a significant impact on performance. Mode 3 consistently produced the best results, while Mode 2 generally showed the lowest performance.

In contrast, for the BC7 codec—which involves more complex encoding computations—the P4/T5 and P5/T4 configurations achieved the best performance. This suggests that for computationally intensive codecs, both the number of processes and the number of threads per process are critical for optimal performance. Similar to ETC2, BC7 showed the best results under Mode 3, achieving speedups of 1.69 $\times$ for P4/T5 and 1.72 $\times$ for P5/T4 relative to the baseline P1/T20. These findings indicate that a balanced parallelization strategy that considers the distribution of work between encoding and loading is essential for high-complexity codecs like BC7.

4 Conclusion and Future Work

This study demonstrates that the integration of multi-processing, multi-threading, and input file ordering enhances performance compared to multi-threading alone in large-scale texture compression. Future work will explore optimal configurations and parallelization strategies for tasks with diverse I/O patterns, such as multi-input single-output compression and single-input multi-output decompression.

Acknowledgments

We thank the etcpak 2.0 authors for their public code release. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-00521436).

References

Hyeon-ki Lee and Jae-Ho Nah. 2023. H-ETC2: Design of a CPU-GPU Hybrid ETC2 Encoder. *Computer Graphics Forum* (2023). doi:10.1111/cgf.14969

Jae-Ho Nah. 2020. QuickETC2: Fast ETC2 texture compression using Luma differences. *ACM Trans. Graph.* 39, 6, Article 270 (Nov. 2020), 10 pages. doi:10.1145/3414685.3417787

Jae-Ho Nah. 2023. QuickETC2-HQ: Improved ETC2 encoding techniques for real-time, high-quality texture compression. *Computers & Graphics* 116 (2023), 308–316. doi:10.1016/j.cag.2023.08.032

Bartosz Taudul. 2024. *etcpak 2.0 : The fastest ETC compressor on the planet.* https://github.com/wolfpld/etcpak