High Performance Graphics (2014)
Jonathan Ragan-Kelley and Ingo Wald (Editors)

SegTC: Fast Texture Compression using Image Segmentation

P. Krajcevski[†] and D. Manocha[‡]

The University of North Carolina at Chapel Hill

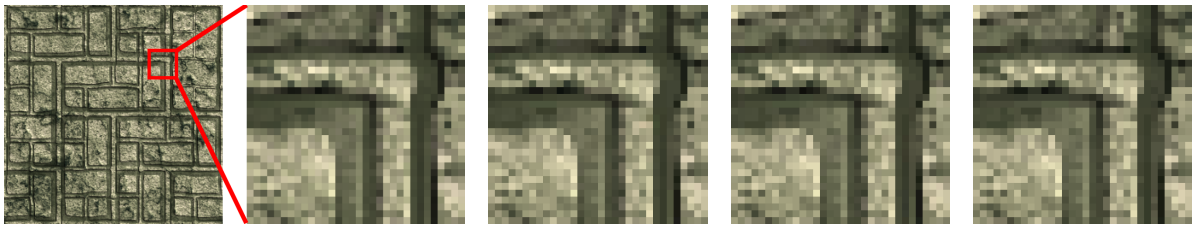


Figure 1: *Our partition-based texture compression algorithm applied to a standard wall texture. The full original texture is shown on the far left, followed by a zoomed in investigation of the region outlined in red. Our method compresses the texture into the BPTC format. The resulting image quality, measured in Figure 4, is comparable to prior methods. This texture is* 256×256 *pixels large and was compressed using an exhaustive method (64 seconds [Don10]), FasTC (567 milliseconds [KLM13]), and our method (143 milliseconds) on an Intel Core i7-4770k 3.80GHz processor using a single core without vector instructions.*

Abstract

Fast, high quality texture compression is becoming increasingly important for interactive applications and mobile GPUs. Modern high-quality compression formats define sets of pre-existing block partitions that allow disjoint subsets of pixels to be compressed independently. Efficient encoding algorithms must choose the best partitioning that fits the data being compressed. In this paper, we describe a new method for selecting the best partition for a given block by segmenting the entire image into superpixels prior to compression. We use the segmentation boundaries to determine a partitioning for each block and then use this partitioning to select the closest matching predefined partitioning. Using our method for BPTC compression results in up to 6x speed-up over prior methods while maintaining comparable visual quality.

1. Introduction

A major issue in the design of current graphics processors is to provide faster access to the visual data for rendering applications, such as geometric data, textures, and animations. These visual assets are often compiled into a format that is amenable to fast CPU to GPU memory transfer. Efficient use of GPU memory is an increasingly important issue in the design of algorithms for interactive applications. The compilation of visual assets is also one of the major bottlenecks in content creation. It is important to design techniques that can result in faster iteration times to support the hardware resources. In many applications, textures account for a large fraction of the visual assets in terms of memory size.

For over a decade, many hardware vendors have incorporated hardware decompression of texture data in GPUs [KSKS96] [TK96]. Compressed textures with small memory footprints have many advantages such as cache coherency during rendering, allowing more textures to be stored in GPU memory, and reduced memory bandwidth for data access. Efficient software texture compression formats are used to fit a large number of textures in memory and

[†] pavel@cs.unc.edu
[‡] dm@cs.unc.edu

DOI: 10.2312/hpg.20141095

increase the hardware utilization. With the recent development of increasingly complex compression formats, such as BPTC [Ope10] and ASTC [NLP*12], we need improved algorithms for fast and high quality texture compression. In practice, texture compression is regarded as one of the most expensive stages of asset compilation.

Compressed texture formats in hardware require the stored data to have support for random access texel lookup. To maintain this random access requirement, compression formats are typically specified on fixed size $N \times M$ texel blocks (commonly $N = M = 4$) [BAC96]. This limitation forces any hardware based compression scheme to be lossy with a fixed compression ratio. For each block, encoders are required to specify compression parameters that will faithfully reconstruct the block in hardware. To mitigate the resulting compression loss, compression algorithms tend to search a large space of parameters for each block [NLP*12].

Block-wide compression parameters, such as those used in PVRTC [Fen03] and S3TC [INH99], are usually too restrictive for high quality rendering. Recent formats, such as ASTC [NLP*12] and BPTC [Ope10], have increased the quality of compressed textures by introducing *partitioning* of blocks so that disjoint subsets of pixels within a block share separate compression parameters. To avoid the exponential increase in the number of partitionings with respect to the block size, these formats choose from a restricted set of common partitionings. The per-block task of an encoding algorithm becomes twofold:

- Select a partitioning out of a predefined set.
- Choose parameters for each subset of the partitioning.

In this paper, we present a new method for choosing predefined partitionings for partition-based compression formats. Our method first computes a segmentation of the image into *superpixels* to identify homogeneous areas based on a given metric. This segmentation defines borders between areas whose pixels share common characteristics. To select predefined partitions, each block uses the segmentation boundaries to determine the best partitioning for that block. Next, we use a *vantage point tree* to find the nearest matching partition based on Hamming distance. Using this selection scheme, we test our technique on low-dynamic range textures. We observe up to a 6x performance increase on existing single-core compression implementations and approach interactive rates for 256×256 sized textures.

The rest of the paper is organized as follows. In Section 2 we briefly survey current compression formats and algorithms. In Section 3 we highlight the criteria used for image segmentation, the metric used to determine the similarity between partitionings, and present our compression algorithm. Finally, we present results in Section 4.

2. Related Work

Modern texture compression formats are based on a technique known as Block Truncation Coding, or BTC, introduced by Delp and Mitchell [DM79]. In this approach, for every 4×4 block of 8-bit pixels, two separate 8-bit grayscale values are chosen by using a single bit per pixel for a total of two bits per pixel (2bpp). Beers et al. [BAC96] introduced the idea of compressing textures using vector quantization while maintaining fast decompression. In particular, they presented a texture compression scheme that preserved random access of pixels, provided a fast decoder, and maintained acceptable compression quality for the ratio. They also claimed that compression speed was not an issue because it could be performed offline. However, with the need for fast iteration times during content creation and multi-platform applications such as Google Maps, compression speed is becoming a major issue in the design of texture compression algorithms.

One of the first commercial uses of texture compression was the S3TC format proposed by Iourcha et al. [INH99] S3TC provided 4bpp for RGB textures by compressing 4×4 blocks of pixels into two RGB565 endpoints and storing two bits per pixel as interpolation values between them. Geldreich improved S3TC by considering macroscopic S3TC blocks [Gel12]. Fenney generalized the S3TC algorithm to exploit the worst-case nearby block access during filtering [Fen03]. Fenney was able to demonstrate acceptable quality with this technique for both 4bpp and 2bpp formats. Strom et al. later introduced a new technique (ETC) that focused on separating the chrominance from luminance in images to compress them separately [SAM05] [SP07]. This 4bpp method provided a significant improvement over S3TC and PVRTC for certain classes of textures.

ETC was the first format to provide partitioning in a given compression block. Compressors could choose between a 2×4 or 4×2 partitioning of a 4×4 block by setting the appropriate bit. More recently, Block Partition Texture Compression (BPTC) was introduced as a high quality compression format that partitions a 4×4 pixel block and compresses each subset separately using the technique from S3TC [Ope10]. Although initial compressors were slow [Don10], there has been considerable work recently on increasing the speed of BPTC compression algorithms [KLM13] [Duf13]. Similarly, Nystad et al. [NLP*12] introduced Adaptive Scalable Texture Compression (ASTC), a diverse new format that supports partitioning similar to BPTC along with ratios from 0.89bpp up to 8bpp. While BPTC stores preselected partitions in two six-bit look up tables, ASTC defines a hashing function that determines the partitioning based on the number of subsets and a ten-bit seed.

Current codecs employ a metric on possible partitionings on a per-block basis. Most BPTC codecs choose a partitioning by estimating the amount of error per partition [KLM13] [Duf13]. The ASTC reference codec computes an ideal partitioining for a given block and then finds the best match to an existing partitioning [ARM12]. We take

a similar approach in our method, but with a few key differences. Namely, the ASTC codec computes an optimal partitioning for every block. Our method computes an optimal global partitioning (segmentation) and then chooses partitions based on the labels covered by compression blocks. The advantage here is that we can use the same segmentation for multiple different block sizes, allowing better performance when compressing textures for a variety of compression ratios. Additionally, we use an acceleration structure to increase the performance of searching through each of the partitionings.

3. Segmenting Images for Texture Compression

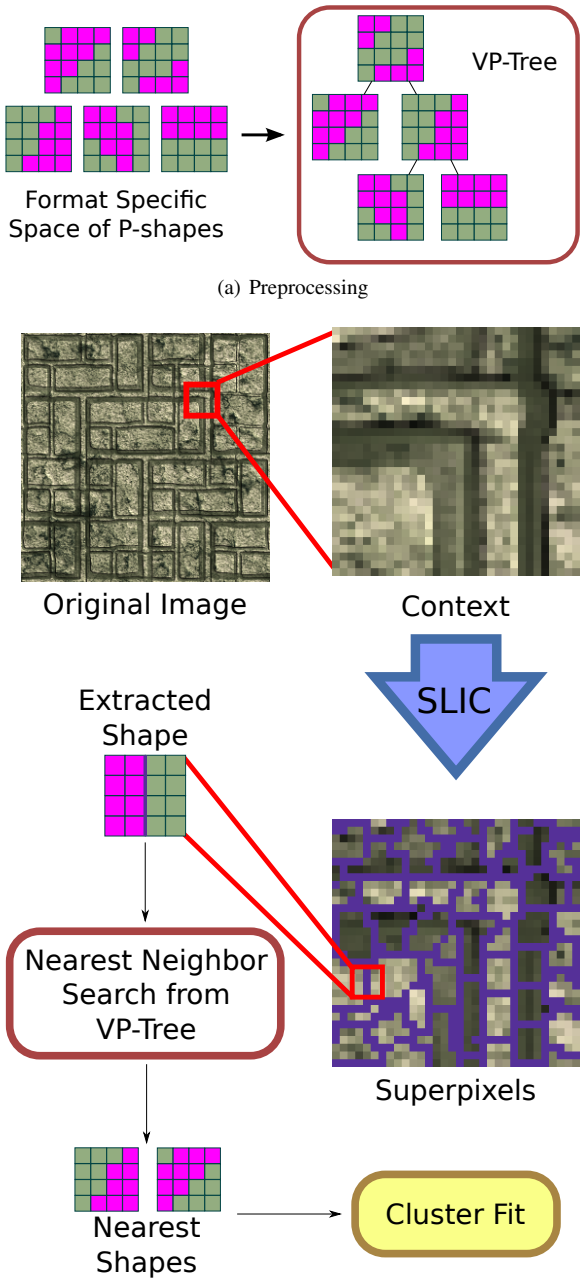
In modern texture compression schemes, *partitioning* has become a prominent technique in increasing the quality of compression formats. Partitioning takes $N \times M$ pixel blocks and assigns a label to each pixel. Pixels that share the same label are compressed independently. Due to the limited number of bits that are allocated for partition selection, compressors must choose from a fixed set of preselected partitionings called *P-shapes*. For example, the BPTC format has 64 separate P-shapes for two and three subset partitionings for a total of 128 [Ope10]. In order to properly select a P-shape, many compressors chose from a partial ordering and then perform a full compression on each P-shape to chose the best one [NLP*12] [Duf13]. For certain compression formats, such as 12×12 ASTC, the P-shape space contains 3123 unique P-shapes [NLP*12]. For large textures, P-shape selection becomes a very expensive part of the algorithm.

3.1. Overview

We propose a new P-shape selection method based on image segmentation, as described in Figure 2. First, we segment the texture into *superpixels* as described in Section 3.2. As discussed in Section 1, encoders for partition-based compression formats must implement two separate stages. The superpixels computed by the segmentation algorithm are used in the first stage of our approach for fast P-shape selection. The second stage of our approach computes compression parameters for each subset using the cluster-fit algorithm [KLM13]. During image segmentation, each pixel is labeled with a superpixel index. P-shapes are then chosen by considering the labels of the pixels within the block being compressed. As described in Section 3.3, this subset of labels is used as the target for a nearest-neighbor search of the available P-shapes. The P-shapes with the closest matching partitioning are used in the cluster-fit algorithm.

3.2. Segmentation

In this section, we describe the segmentation algorithm that underlies the P-shape selection method. Image segmentation has been heavily studied in computer vision and image processing [ASS*12]. The goal of image segmentation is to apply a labeling to each pixel such that pixels sharing a com-



(a) Preprocessing

(b) Runtime

Figure 2: *An overview of our compression algorithm. (a) The VP-Tree is constructed from format-specific P-shapes as a preprocessing step. (b) For each image, we perform SLIC segmentation. For each block, we extract the corresponding partitioning that matches the superpixel boundaries and find the nearest P-shapes using the VP-Tree. The closest P-shapes are used with the cluster-fit algorithm to produce the final compression parameters.*

Original Context SLIC

Figure 3: *(left) The original image. (center) An investigation of the area highlighted in teal. (right) SLIC superpixels: the image is segmented into small regions that adhere to feature boundaries [ASS*10].*

mon label all share a common property or visual characteristic. This property is often the minimization of an metric used in distinguishing image features. Recently, there has been much work in segmenting the image into large contiguous regions of pixels known as *superpixels* [FH04] [VBM10] [ASS*10]. Such a partitioning provides a classification into areas of pixels that admit certain coherence properties. We find that partitioning blocks by superpixel boundaries is suitable for texture compression.

We use a superpixel segmentation method known as SLIC, or Simple Linear Iterative Clustering [ASS*10]. In order to maintain encoder performance we chose this method for its simplicity and speed versus other methods [ASS*12]. SLIC takes as parameters either the number or desired size of the superpixels. Using this parameter, SLIC uses equally spaced kernels over the texture as the initial cluster centers for a k-means clustering algorithm. Once the clustering is computed, any pixels that are not contiguously connected to their cluster centers then 'push' the superpixel border to connect the components. The error metric chosen to determine the distance between two pixels is a key issue with respect to SLIC. For each pixel p, we calculate the distance from the pixel coordinates (x,y) and the pixel value converted to CIE LAB space (L,a,b) as

$$d(p_1,p_2) = \begin{array}{l} \alpha\|(x_1,y_1)-(x_2,y_2)\|_2 \\ +\,\beta\|(L_1,a_1,b_1)-(L_2,a_2,b_2)\|_2 \end{array}$$

where α and β are values chosen to weigh the relative contribution [oI04]. Although most compression formats operate in RGB space, we segment the image in CIE LAB space in order to leverage the fact that euclidean distance is correlated to perceived difference. Different error metrics may provide better compression values for specific textures because of the high variability of information types stored in textures.

3.3. P-shape Selection

Once the target partitioning has been selected from the segmented image, we proceed by finding the best P-shape defined by our compression format that matches it. The target partitioning rarely matches exactly to any of the predefined P-shapes. In order to properly compare one partitioning to another, we must define an error metric between partitionings. Furthermore, since the P-shape space does not change between textures for a given compression format, we can accelerate the search by using a data structure to perform efficient nearest neighbor lookups using the metric described in Section 3.3.1

Some formats, such as ASTC, use blocks as large as 12x12 pixels, meaning that our partitionings would have up to 144 variables. The high dimensionality per P-shape prevents the use of classical data structures such as k-d trees because they are no better than brute force search. However, many well-studied data structures have been developed for performing nearest neighbor lookups [APPK08] [Yia93] [Ben75]. The major requirement for most data structures is that the metric between two points satisfies the triangle inequality. Provided we can develop an adequate metric for partitionings that satisfies this inequality, we can use an existing data structure that supports high-dimensional queries.

3.3.1. Block partitioning metric

The main idea behind the metric is to determine whether or not a given partitioning is sufficiently different from any other. Partitionings are represented by a per-pixel labeling, but each label's compression parameters are computed independently. Hence, for a given block of $N \times M$ pixels, each partitioning can define labels $l_i \in \mathbf{N}$ with $i \in [0, NM-1]$. Two partitionings with labels p and q are identical if

$$p_i = p_j \iff q_i = q_j \;\forall\, i,j. \tag{1}$$

To determine the difference between two P-shapes, we consider their labeling as strings of length $k = NM$. Given labelings $p = p_0 p_1 ... p_k$ and $q = q_0 q_1 ... q_k$ we must first find a relabeling R from unique labels in p to unique labels in q such that the Hamming distance between strings $R(p) = R(p_0)R(p_1)...R(p_k)$ and q is minimized [Ham50]. This distance is used as our P-shape metric.

The relabeling R is not necessarily bijective: one P-shape may have more unique labels than the other. We can define a subset of a P-shape p to be a set of all identical labels p_j. If two subsets of a P-shape p independently fulfill Property 1 with respect to a single subset of a P-shape q, then the compression parameters for the subsets of p may be duplicated for both subsets. However, in practice the number of subsets in a P-shape limits the number of bits that are allowed for compression parameters. Forcing R to be one-to-one, but not necessarily surjective, enforces the bit allocation constraint. Hence, we compute the optimal relabeling as a bipartite matching problem which can be done in polynomial time. For performance, we approximate the optimal solution by relabelling each partitioning such that the pixel in the top left has label zero, and the labels increase from left to right. Using this method we observe a negligible decrease in com-

pression quality (< 0.1 db) over full bipartite matching while observing a 2X performance increase.

3.3.2. Vantage Point Trees

In order to perform nearest neighbor lookups we use a *vantage-point tree* or VP-Tree [Yia93]. We use the VP-Tree because of its ability to handle high dimensional queries along with its $O(\log N)$ query complexity. The quality and speed of the VP-Tree ultimately depend on the breadth of P-shapes available for the compression format. The VP-tree is a binary tree where each node is a P-shape p, and the left child p_l contains all of the P-shapes within a radius r of p while the right child p_r contains all of the P-shapes outside of this radius. At each node, we can prune half of the remaining nodes if the candidate P-shape falls within the associated radius of that node. In practice, the discrete nature of the search space precludes asymptotic $O(\log n)$ behavior, but we still observe better performance than brute force search.

4. Results

In order to test our algorithm, we compare it against existing encoders. Due to availability and maturity of tools, we compare our results in terms of the BPTC format. We compare it against all software implementations that can be run on a single core irrespective of specialized hardware, although implementations that use GPUs or vector instructions achieve faster compression speeds. However, the presence of such features is not guaranteed on all platforms, such as embedded devices. We test our algorithm against the existing reference encoder that performs an exhaustive search of the compression parameters and against FasTC [KLM13] [Don10].

As Griffin et al. have shown, it is difficult to choose a single quality metric for compressed textures [GO14]. Even the classical *peak signal-to-noise ratio* (PSNR) is an unreliable metric [GO14] [WBSS04]. However, for reproducibility and comparison with prior work, we present comparisons using both PSNR and the structural similarity image metric (SSIM) in Figure 4 [WBSS04]. As SSIM is only a single channel metric, we first convert the textures to grayscale prior to using the reference implementation. PSNR is calculated using the same formula as FasTC [KLM13].

5. Conclusions, Future Work, and Limitations

Conclusions: We have presented a new algorithm for selecting P-shapes for partition based texture compression formats. We use image segmentation to designate superpixels of an image and use them to select the ideal partitioning for each block. We expect this algorithm to be the basis for future research in fast P-shape selection methods. Efficient representations of images that quickly convert to GPU-based formats open up an entire area of research devoted to efficient GPU-oriented image representations.

Future Work: The segmentation algorithm at the core of our method is crucial to providing fast compression speeds. The SLIC algorithm used in our method performs k-means clustering to group pixels based on both spatial and perceptual proximity. However, compression formats and partitionings do not meet these specific constraints in general. We believe that there is valuable future work to be done in terms of using segmentation algorithms that can group pixels amenable to compression formats. Similarly, the choice of error metric can provide better segmentations based on what the texture is used for. For example, a normal map may be segmented such that pixels that share a label reconstruct to a similar unit normal. Furthermore, it should be possible to store segmented images along with per-label compression parameters more efficiently than bare GPU-specific formats. This research would greatly benefit applications that leverage on-the-fly compression such as mobile devices.

Limitations: Our algorithm also suffers from a few problems due to the limitations of the underlying segmentation algorithm. Most notably, it may not work well compressing textures with alpha. Also, it is very sensitive to the parameters used to perform the segmentation. The parameter that chooses the size of the superpixels must be small enough to capture fine details but not large enough such that we lose the benefits of the segmentation. Finally, the formats that support multiple subsets per block also support high-quality single-subset compression. For these formats, the acceleration gained from spending less time calculating an optimal P-shape pushes the multi-subset encodings of a single block behind the single-subset encodings with respect to image quality. This limitation ultimately diminishes the benefits of formats that support block partitoning. For this reason, calculating optimal P-shapes remains an active area of research.

Overall, the approach presented in this paper provides a good basis for developing specialized and faster texture compression algorithms for modern texture compression formats.

Acknowledgements: This work was supported in part by ARO Contracts W911NF-10-1-0506, W911NF-12-1-0430, and NSF awards 100057 and Intel. We would also like to thank the reviewers for their valuable feedback.

References

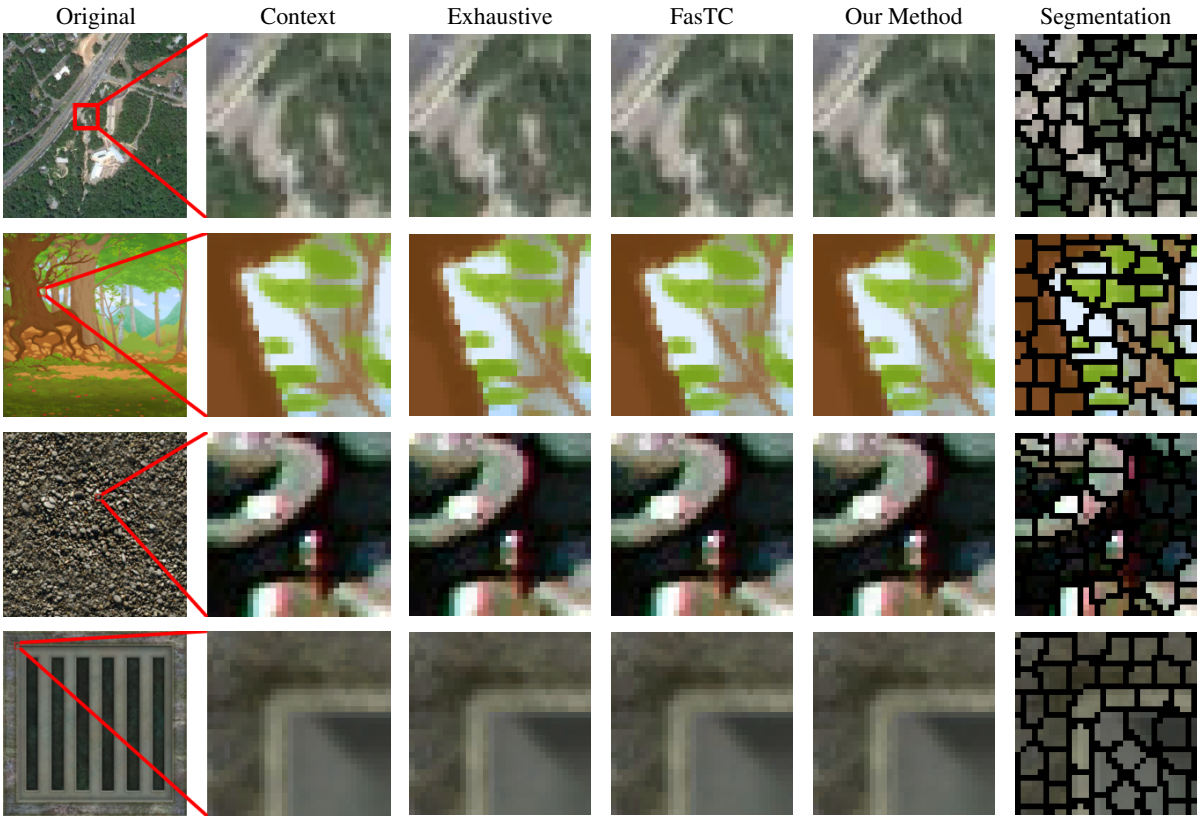
[APPK08] ATHITSOS V., POTAMIAS M., PAPAPETROU P., KOLLIOS G.: Nearest neighbor retrieval using distance-based hashing. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on* (April 2008), pp. 327–336. 4

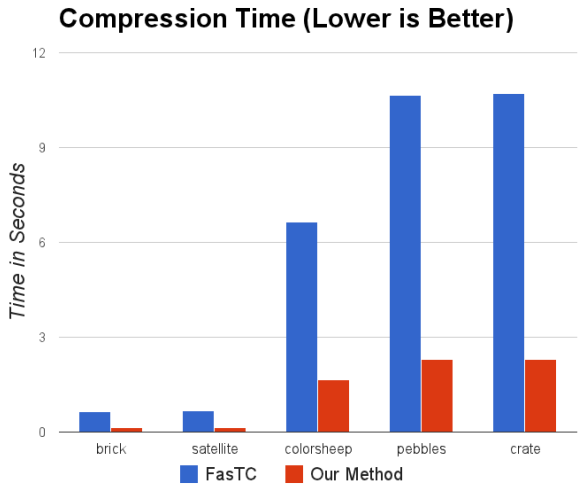
[ARM12] ARM: Astc evaluation codec. http://malideveloper.arm.com/develop-for-mali/tools/astc-evaluation-codec/, 2012. 2

[ASS*10] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SÃIJSSTRUNK S.: SLIC Superpixels, 2010. 4

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SÃIJSSTRUNK S.: SLIC Superpixels Compared to State-of-the-art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 11 (2012). 3, 4

[BAC96] BEERS A. C., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *Proceedings of the 23rd*





Compression Quality

Peak Signal to Noise Ratio (PSNR)

	brick	satelite	colorsheep	pebbles	crate
Ex.	43.14	45.30	49.50	39.07	49.25
FasTC	42.49	44.61	47.30	38.14	47.99
SegTC	41.60	43.55	44.75	36.00	47.09

Structural Similarity Image Metric (SSIM)

	brick	satelite	colorsheep	pebbles	crate
Ex.	.9987	.9975	.9976	.9970	.9969
FasTC	.9985	.9970	.9953	.9963	.9956
SegTC	.9981	.9966	.9925	.9947	.9954

Figure 4: *From top to bottom we compare encodings of 'satellite', 'colorsheep', 'pebbles', and 'crate'. In the graphs, 'brick' refers to the texture displayed in Figure 1. The exhaustive algorithm is not displayed in the performance graph because it is two orders of magnitude slower than FasTC. To complement our segmentation algorithm we have chosen a representative sample of textures that are meant to be consumed visually, and report errors using both peak signal-to-noise ratio and the structural similarity image metric. We observe an increase in encoding speed over existing implementations while maintaining a similar quality level. Additionally, we notice that the choice of segmentation is very important because we lose some detail in parts of 'colorsheep' where the segmentation is too large to catch fine details. To contrast, we maintain the visual detail of 'pebbles' and 'satellite' very well. All timings are performed on a single core Intel Core i7-4770 CPU 3.40GHz without vector instructions. The texture 'colorsheep' is provided courtesy of Trinket Studios, Inc. The texture 'satellite' is provided courtesy of Google, Inc. The remaining textures are public domain from* www.opengameart.org

annual conference on Computer graphics and interactive techniques (1996), SIGGRAPH '96, ACM, pp. 373–378. 2

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509–517. 4

[DM79] DELP E., MITCHELL O.: Image compression using block truncation coding. *Communications, IEEE Transactions on 27*, 9 (sep 1979), 1335–1342. 2

[Don10] DONOVAN W.: Bc7 export, distrubuted with nvidia texture tools. http://code.google.com/p/nvidia-texture-tools/, 2010. 1, 2, 5

[Duf13] DUFRESNE M. F.: Fast ispc texture compressor. http://software.intel.com/en-us/articles/fast-ispc-texture-compressor, 2013. 2, 3

[Fen03] FENNEY S.: Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), HWWS '03, Eurographics Association, pp. 84–91. 2

[FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: Efficient graph-based image segmentation. *Int. J. Comput. Vision 59*, 2 (Sept. 2004), 167–181. 4

[Gel12] GELDREICH R.: Crunch. http://code.google.com/p/crunch/, 2012. 2

[GO14] GRIFFIN W., OLANO M.: Objective image quality assessment of texture compression. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D '14, ACM, pp. 119–126. 5

[Ham50] HAMMING R. W.: Error detecting and error correcting codes. *Bell System Tech. J. 29* (1950), 147–160. 4

[INH99] IOURCHA K. I., NAYAK K. S., HONG Z.: System and method for fixed-rate block-based image compression with inferred pixel values. U. S. Patent 5956431, 1999. 2

[KLM13] KRAJCEVSKI P., LAKE A., MANOCHA D.: FasTC: accelerated fixed-rate texture encoding. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2013), I3D '13, ACM, pp. 137–144. 1, 2, 3, 5

[KSKS96] KNITTEL G., SCHILLING A., KUGLER A., STRAÃ§ER W.: Hardware for superior texture performance. *Computers & Graphics 20*, 4 (1996), 475–481. 1

[NLP*12] 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), HPG '12, Eurographics Association, pp. 105–114. 2, 3

[oI04] ON ILLUMINATION I. C.: *Colorimetry*. CIE technical report. Commission internationale de l'Eclairage, CIE Central Bureau, 2004. 4

[Ope10] OPENGL A. R. B.: ARB_texture_compression_bptc. http://www.opengl.org/registry/specs/ARB/texture_compression_bptc.txt, 2010. 2, 3

[SAM05] 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), HWWS '05, ACM, pp. 63–70. 2

[SP07] STRÖM J., PETTERSSON M.: ETC2: texture compression using invalid combinations. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), GH '07, Eurographics Association, pp. 49–54. 2

[TK96] TORBORG J., KAJIYA J. T.: Talisman: Commodity real-time 3d graphics for the pc. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 353–363. 1

[VBM10] VEKSLER O., BOYKOV Y., MEHRANI P.: Superpixels and supervoxels in an energy optimization framework. In *ECCV 2010*, Daniilidis K., Maragos P., Paragios N., (Eds.), vol. 6315 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 211–224. 4

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (april 2004), 600–612. 5

[Yia93] YIANILOS P. N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1993), SODA '93, Society for Industrial and Applied Mathematics, pp. 311–321. 4, 5