Fast PVRTC Texture Compression using Intensity Dilation

Pavel Krajcevski Dinesh Manocha
University of North Carolina at Chapel Hill

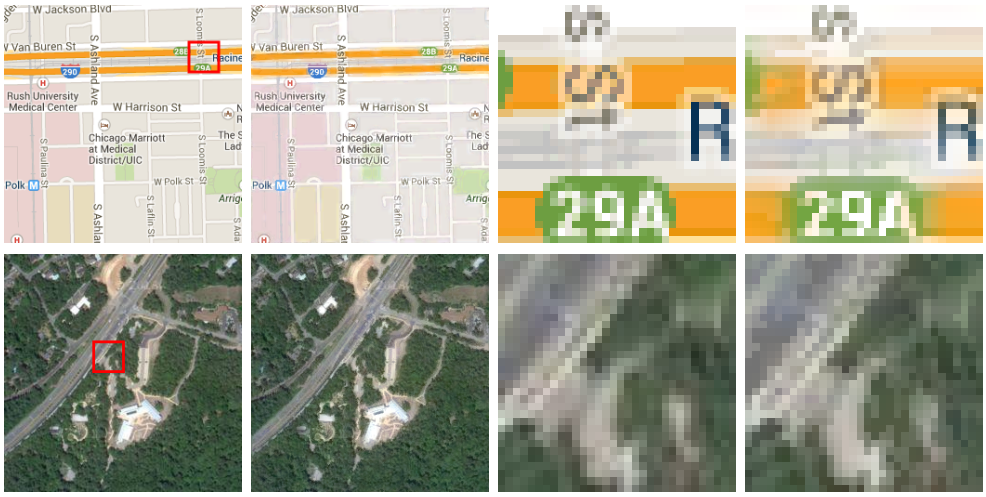


Figure 1. Texture compression using intensity dilation applied to images used in GIS applications such as Google Maps.

Abstract

We present an algorithm for quickly compressing textures into PVRTC, the Imagination PowerVR Texture Compression format. Our formulation is based on intensity dilation and exploits the notion that the most important features of an image are those with high contrast ratios. We present an algorithm that uses morphological operations to distribute the areas of high contrast into the compression parameters. We use our algorithm to compress into PVRTC textures and compare our performance with prior techniques in terms of speed and quality.

http://gamma.cs.unc.edu/FasTC

 ISSN 2331-7418

1 Introduction

Textures are frequently used in computer graphics applications to add realism and detail to scenes. As more applications leverage the GPU and require high-fidelity rendering, the cost for storing textures is rapidly increasing. Within the last decade, texture representations that provide hardware-friendly access to compressed data have become a standard feature of modern graphics processors. Texture compression, first introduced by Beers et al. [1996], exhibits many key differences from standard image compression techniques. Due to the random-access restrictions, compressed texture formats provide lossy compression at a fixed ratio.

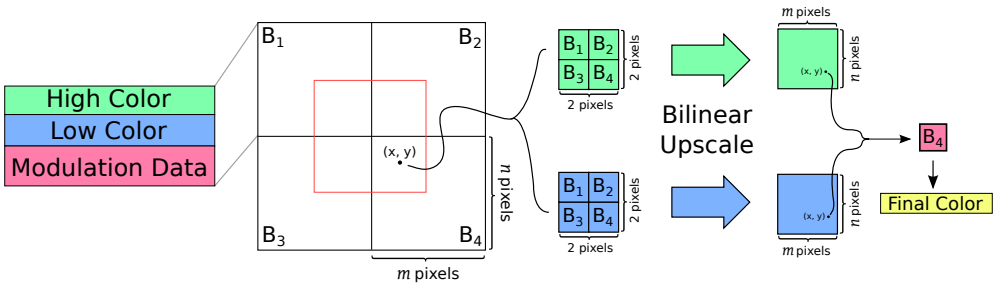
One of the main tenets of a good compression scheme, as introduced by Beers et al. [1996], is the notion that compression can be offloaded to preprocessing stages and real-time decompression is performed in hardware. Increasingly GPUs are being used for applications other than desktop-based 3D games, but continue to make heavy use of textures. These include geographic information systems and mapping tools (e.g. Google Maps) that use textures rendered on-the-fly based on level of detail and other factors, as shown in Figure 1. As a result, it is becoming increasingly important to develop real-time high quality texture compression algorithms.

In this paper we focus on a widely used texture compression method known as Low-Frequency Signal Modulated Texture Compression (LFSM) [Fenney 2003]. LFSM leverages the cache-coherent worst-case scenario of block-based texture compression techniques such as DXT1 and BPTC [Iourcha et al. 1999] [OpenGL 2010]. It has been pointed out that LFSM texture compression formats, such as PVRTC, provide better quality than formats with similar compression ratios (e.g. DXT) on certain classes of textures [Fenney 2003]. However, due to the structure of LFSM formats, fast or real-time compression algorithms are not as available compared to other formats [Schneider 2013].

1.1 Low Frequency Signal Modulated Texture Compression

The most popular texture compression formats, such as DXTn, BPTC, and ASTC, compress textures by operating on 4x4 blocks [Iourcha et al. 1999][OpenGL 2010][Nystad et al. 2012]. For each block, the pixels are reduced to interpolation points along a line segment. Each format saves the endpoints of the line segment at a lower precision than the original image, and it uses predefined interpolation points which provide a savings in bits.

Like other texture compression formats, PVRTC compressed textures are stored in a grid of blocks, each containing information for a 4×4 or 4×8 grid of texels. As shown in Figure 2, each block contains two colors along with per-texel modulation data. Each of these colors, referred to as the *high color* and *low color*, is used in conjunction with neighboring blocks to create two low resolution images: the *high image* and *low image*, respectively. In order to lookup the value for a texel, the high image




```
Color4 Modulate(Color4 low, Color4 high, int modulation) {
  assert(modulation < 4);
  const float lerpVals[4] = {0, 3.0f/8.0f, 5.0f/8.0f, 1};
  return lerp(low, high, lerpVals[modulation]);
}
```

Figure 2. Representation of low-frequency signal modulated texture data. The compressed data is stored in blocks that contain two colors, high and low, along with modulation data for each texel within the block. To decompress a texel at location (x,y), the high and low colors of the surrounding blocks are used to create two upscaled patches. The final color is determined by using the interpolation value of the pixel within the block to interpolate between the corresponding pixels in the generated patches.

and low image are upscaled to the original image size using bilinear interpolation. Once upscaled, modulation data from the block that contains the texel in question is used to compute a final color. This bilinear interpolation avoids the worst-case scenario with respect to memory lookups. By filtering textures across block boundaries, information from four blocks is required to decode any texel value.

The biggest impediment to computing an efficient encoding is the fact that the optimal encoding must be stored within the PVRTC data format. This limitation necessitates discretizing our values, and quantization of the solution from the real numbers to integers does not provide an optimal encoding in general. This means that the aforementioned problem becomes an integer programming problem, which is known to be NP-Complete [von zur Gathen and Sieveking 1978]. As a result, computing the optimal encoding is impractical.

1.2 State of the Art

The only known hardware implementation that uses LFSM texture compression is Imagination's PowerVR architecture, giving the format the name PVRTC (PowerVR Texture Compression) [Imagination 2013]. Currently, PVRTC compressors use a two stage process. In the first stage, they provide an initial approximation of low and high images and modulation data. In the second stage, they continually refine their initial approximation of the low and high images and modulation data until they reach a

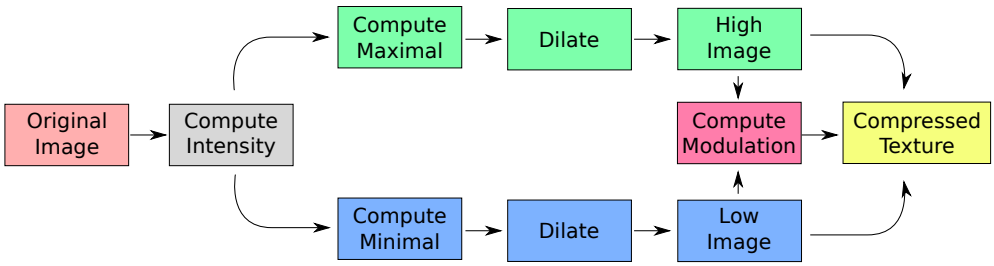


Figure 3. Overview of the compression algorithm: The minimum and maximum of the intensity values are dilated to produce the high and low images. These images are then used to compute the optimal modulation values, and the results are stored in the compressed PVRTC format.

fixed point with respect to improving compression quality against the original image.

The initial approximation of per-block color values is determined by first applying a low-pass filter to the image. Next, the difference between the filtered image and the original is analyzed using principal component analysis. The principal component of the difference vectors, treated as three dimensional vectors in RGB space, are used to generate the high and low block color values for the image. While this scheme provides a good approximation, computing the principal components can be expensive and the low pass filter may remove image details that tend to preserve fidelity.

Once the initial approximation is computed, the values can be used to iteratively refine the solution. First, the modulation values are fixed, and the endpoint values are optimized. Then the endpoint values become fixed, and the modulation values are optimized. The former is done by computing an SVD that solves the problem of finding the optimal endpoints in a 2x2 block window. The latter is done by a brute force search over the four possible interpolation values for each pixel. In this paper we highlight a new way to compute the initial approximation and assume that, if desired, this optimization procedure can be applied.

2 PVRTC compression using Intensity Dilation

The basis for our texture compression approach resides in the well studied foundations of the human visual system's sensitivity to contrast [Aydin 2010]. In particular, our algorithm takes advantage of localized areas of an image that have high contrast ratios. For most textures, these areas are those that contain edges between high intensity and low intensity regions [Krajcevski and Manocha 2014a]. This section describes an overview of the algorithm.

Due to the way that PVRTC compressed textures store the compressed data, as in Figure 2, there is an inherent filtering procedure that takes place during decompression. This filtering procedure blurs sharp edges in textures. By storing nearby

extremal values in the high and low colors, we can recreate sharp edges by using the modulation values. More often than not, the two colors on either side of a visible edge have different intensity values. In order to avoid the filtering during decompression, we store the higher intensity value in the high color of all nearby blocks, and we store the lower intensity value in the lower color of all nearby blocks. During decompression, the per-pixel modulation value can be chosen to be either the high or low color of each block in order to accurately recreate the edge. In many cases, the preservation of these edges is more important than the small gradation in color leading up to them.

2.1 Intensity Labeling

In order to preserve the contrast within textures, the first step in our compression scheme is to determine the high and low intensity values that produce the contrast. We start by using the definition for luminosity derived from the Y value of the CIE XYZ color space due to its speed and simplicity of calculation [on Illumination 2004]:

```
float to_intensity(uint32_t rgba) {
  const float a = (float)((pixel >> 24) & 0xFF) / 255.0f;
  const float r = a * (float)(pixel & 0xFF) / 255.0f;
  const float g = a * (float)((pixel >> 8) & 0xFF) / 255.0f;
  const float b = a * (float)((pixel >> 16) & 0xFF) / 255.0f;

  return r * 0.2126f + g * 0.7152f + b * 0.0722f;
}
```

Other luminance values, such as the L channel of CIE L^*a^*b are also viable alternatives for computing the luminance. For textures with alpha, we premultiply the alpha channel across each color channel before performing the luminance calculation.

There are many ways to determine the local minima and maxima of intensity, including searching for a near-zero magnitude gradient or evaluating the eigenvalues of the Hessian. A simple alternative is to simply look at the intensity value of each of the neighboring pixels. If all of the neighbors have higher intensity values or all of the neighbors have lower intensity values, then the pixel in question is a local minimum or local maximum, respectively. Once we have determined these local minima and local maxima, we can separate them into two images, one representing all local minima, and the other representing all local maxima.

3 Intensity Dilation

The local minima and maxima of an image give a sparse respresentation of the pixels that represent the contrast of an image in those neighborhoods. We treat the local minima and maxima as two separate images that will eventually be used to calculate the high and low colors of the image. In order to expand the influence of the extremal values, for each image we use a technique from mathematical morphology known

as *dilation* [Serra 1983]. Usually applied to binary images, dilation is the use of a small kernel shape, such as a 3×3 pixel box, to expand a region of pixels. Dilation considers this kernel centered on each pixel in the image, and if the pixel is to be dilated, it is copied into each pixel in the kernel. For PVRTC, the input textures have at least three 8-bit channels that must be dilated. We can consider pixels that have not yet been dilated into 'empty', such that prior to dilation, all of the minimal and maximal pixels are non-empty while every other pixel is empty. When an empty pixel is adjacent to two or more non-empty pixels there must be a strategy for how to perform the dilation. In our method, we have chosen to average adjacent pixel values in order to preserve the color range that corresponds to a block. This reduces the amount that noise affects our choice of block colors. One alternative is to take the texel with the higher or lower intensity based on the image being dilated, but this causes problems with noisy images.

In order to capture the important features of a texture, the extremal pixels of the image (Section 2.1) must be dilated until they influence neighboring block values. In PVRTC, blocks cover 4×4 pixel regions. This implies that any pixel $\mathbf{p}$ at location (p_x, p_y) affected by a block $\mathbf{b}$ centered at (b_x, b_y) is at most 3 units away, where the distance d is defined as

$$d = \sup_{p_x, p_y} \{||\mathbf{b} - \mathbf{p}||_1 : |b_x - p_x| < 4 \text{ and } |b_y - p_y| < 4\}.$$

In order to properly influence the colors of a block that covers a given labeled pixel, we must dilate each of the extrema 3 times.

Once dilated, each block will represent the major local influences of either low or high intensity depending on the image. The resulting block color will be the average of the intensities within the block boundaries. Certain areas, such as color gradients, contain very few local minima or maxima and may not have any dilated texels. In order to prevent these areas from being influenced by texels relatively far from the block center, we fill empty texel values with the enclosing block's corresponding extrema color. Once we have the high and low colors corresponding to a given block, we are free to compute optimal modulation values to match our original pixel colors. We compute the modulation values by locally decompressing the blocks into full resolution high and low images, and then selecting the optimal modulation value via brute force search.

3.1 Parallelization

The dilation steps are inherently parallelizable since they operate locally on pixels. Once the extrema values are extracted from the original image, each image can be dilated in parallel. Since each extremum value needs to be dilated three times, we can compute the value of each pixel in the high and low images by running a 7x7 kernel over each pixel, and choosing the value that is closest spatially to this pixel. Since

PSNR: ∞ PSNR: 26.547 PSNR: 26.550 PSNR: 26.609
SSIM: 1.0 SSIM: 0.9884 SSIM: 0.6940 SSIM: 0.6640

Figure 4. Problems with using PSNR as the only metric. Each image above has a similar PSNR to the original image on the far left. Images courtesy of Zhou Wang [2004].

each pixel is independent, this operation can be performed very efficiently using a GPU.

Similarly, to save on GPU compute cycles, each pixel can be dilated by a 3x3 kernel. However, since this only expands the original image by one pixel in every direction, the 3x3 dilation step would need to be run three times to achieve the same results as the 7x7 dilation mentioned above. The results are identical but can be tailored to your particular architecture. For example, dilating a 3x3 kernel can be separated into two passes: one pass to expand the pixels along the horizontal axis, and one to expand the pixels along the vertical axis. However, all of the results in Section 4 report benchmarks on single-core implementations.

4 Results

The only PVRTC texture compressor known to the authors is Imagination's PVRTexTool [2013], which we use to compare the speed and quality of our algorithm. It incorporates the two stage compression technique described by Fenney et al. [2003] and reviewed in Section 1.2. The following comparisons all use the fastest setting for the compressor and are not focused on quality compression. They do not represent the best possible quality achievable by PVRTC. Also, our results focus on the 4bpp version of PVRTC, but similar methods should be useful for both 2bpp and future iterations of PVRTC. Although the compressor is closed source, the decompressor provided with the SDK was used to verify the results [Imagination 2013].

4.1 PSNR vs SSIM

Classically, the quality of texture compression techniques have always been measured with *Peak Signal to Noise Ratio* (PSNR) [Fenney 2003][Ström and Pettersson 2007][Nystad et al. 2012][Krajcevski et al. 2013]. This metric originates from signal processing and corresponds to the amount of absolute difference between pixel values. When compressing textures, such a metric can be useful, such as when we need

to encode a 2D function as a texture. However, in LFSM compressed textures, decompression focuses on a filtered representation of the compressed data and is mostly designed for textures that will be consumed visually. As shown in Figure 4, PSNR does not correlate with visual fidelity.

For this reason, we also include the Structural Similarity Image Metric (SSIM), a metric developed by Wang et al. [2004] that captures differences of two images as perceived by the human visual system. The metric is defined as

$$SSIM(I_x, I_y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

where μ is the mean intensity and σ is the standard deviation, and σ_{xy} is defined as

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y).$$

C_1 and C_2 are application-defined constants to avoid numerical instability. One limitation of SSIM is that it only measures a single channel. In the subsequent comparisons, we measure SSIM by first converting both the original and compressed image to grayscale.

4.2 Compression Speed

The main benefits of using intensity dilation over previous techniques is in compression speed. Looking at Table 1, we observe a 3.1x speedup over the previous fastest implementations. Similar to other texture compression algorithms, we optimize away areas of homogeneous pixels with precomputed lookup tables [Waveren 2006][Krajcevski et al. 2013]. Furthermore, textures that contain a lot of homogeneity such as the 'streets' texture in Table 1 gain a small benefit from the instruction cache since intensity calculations will reuse texel values. However, as we will see in Section 4.3, we suffer from aggressive averaging artifacts in these areas. Most images do not have large homogeneous areas, and consequently compression speed grows linearly with the number of pixels in the texture.

4.3 Compression Quality

Compressing textures using intensity dilation, we observe an increase in the SSIM index for a majority of textures and maintain similar results in PSNR. Most notably, we can see that certain low frequency features are retained in the compressed versions of many textures with high entropy. Due to intensity dilation, the averaging during dilation around the edges of the roof prevents compression artifacts from arising due to local extrema. This is noticeable across all images that have low frequency features, such as photographs or common billboard textures.

Although our technique is useful for this class of textures, we also observe a class of textures that perform poorly with intensity dilation. These textures correspond to

streets	gradient	satellite	mountains	bricks	gametex	lorikeet
256×256	256×256	256×256	256×256	256×256	512×512	512×512

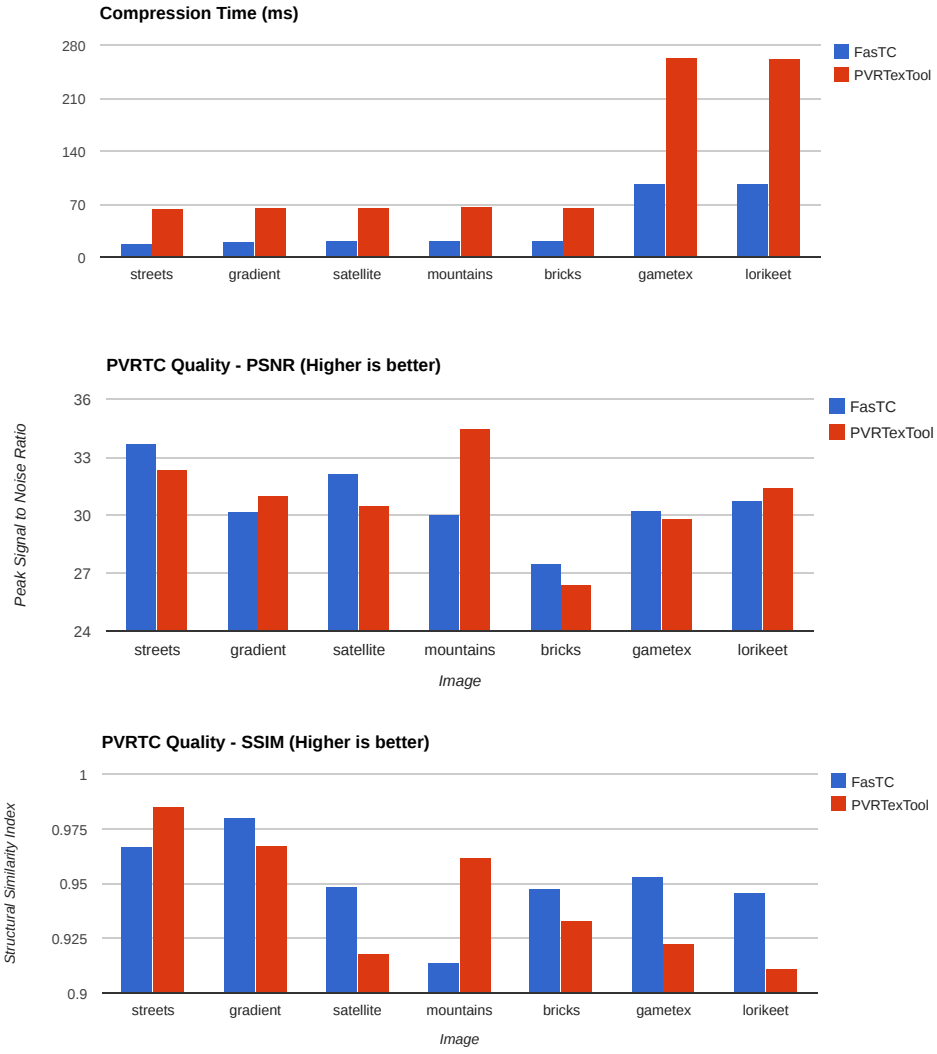
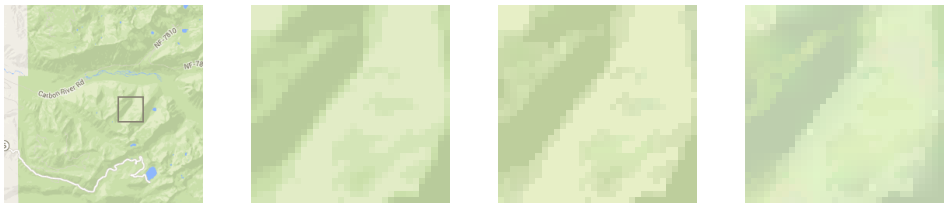


Table 1. Various metrics of comparison for LFSM compressed textures using intensity dilation versus the existing state of the art tools. All comparisons were performed using the fastest quality settings of the February 21st 2013 release of the PVRTexTool [Imagination 2013]. For both metrics, higher numbers indicate better quality. The above results were generated on a single 3.40GHz Intel® Core™ i7-4770 CPU running Ubuntu Linux 12.04. Images courtesy of Google Maps, Simon Fenney, and `http://www.spiralgraphics.biz/`



Context Original PVRTexTool Our Method

Figure 5. Detailed investigation of areas with high pixel homogeneity. We notice that the texture compressed using intensity dilation suffers from artifacts arising from aggressive averaging of nearby intensity values, while the PCA based approach has relatively good quality compression results. Original image retreived from Google Maps.

the relatively low entropy texture 'mountains' (Table 1) generated from vector graphics and used in some modern day geomapping applications. We measure entropy using the common formula from 8-bit intensity values [Shannon 1948]:

$$E = -\sum p_i \log p_i,$$

where p_i is the number of pixels with intensity i divided by the total number of texels. This is not a steadfast metric of when our algorithm performs poorly due to the metric's lack of spatial coherence, but it does provide a good intuition for when intensity dilation may not produce favorable results. Many spatially correlated areas of moderate homogeneity result in overaggressive extrema labeling. The problem arises from the fact that in homogeneous regions of pixels, there is no maximum or minimum. In these instances, either no maximum or minimum exist, and the high and low images will take the maximum and minimum intensity pixel, which is the same value, or every pixel is a maximum and a minimum, so the dilation aggressively eliminates small scale image features. In the worst case, this problem occurs when there are very few colors in a block's region: on the order of two or three. Then every pixel becomes labeled as both a maximum and a minimum, and blurring occurs which removes image detail, as shown in Figure 5.

5 Limitations and Future Work

Limitations: Although intensity dilation provides good results at 3.1 times the speed of conventional LFSM compression techniques, there are still some problems to contend with. Recently, trends in mobile devices are supporting multiple compression formats, such as DXT1 and ETC, where much faster, higher quality texture compression techniques may be available, as shown in Table 2.

Using intensity dilation for LFSM formats should be used to focus on devices that exclusively support LFSM texture compression, such as Apple's iPhone and iPad.

Image	Speed (ms)			Quality (PSNR)		
	DXT1	PVRTC	ETC1	DXT1	PVRTC	ETC1
satellite	0.5	20.9	21.7	32.1	30.4	33.9
mountains	0.5	21.8	18.3	33.3	30.0	36.6
gametex	2.1	97.3	90.3	31.2	30.2	33.2

Table 2. Fastest available compression speeds (including our intensity dilation for PVRTC) for a variety of formats with similar compression ratios.

However, we have bridged the gap between fast texture compression techniques for certain formats, such as PVRTC and ETC1 [Geldreich 2013]. These times do not reflect any multi-threaded or GPU based techniques. Devoting an entire GPU to compress a texture will likely have certain benefits, but will also likely consume more power on mobile devices, which is ultimately undesirable.

Future Work: Although intensity dilation is a good technique for fast PVRTC compression, it does not try to optimize the amount of compression quality afforded by LFSM formats. For example, additional investigation is required to determine the effects of gamma-corrected images versus raw RGB. Furthermore, in most compression techniques, an initial approximation is refined to gain better quality, as described in Section 1.2. We believe that intensity dilation serves as a better initial approximation to these refinement techniques than the previous state of the art and that developing fast techniques for refinement is a ripe area of research. Additionally, we can use the multi-pass formulation of intensity dilation, as introduced in Sections 2.1 and 3, to come up with a parallelizable algorithm that exploit both SIMD and multiple cores, such as consumer GPUs.

Additional compression can be applied to already compressed PVRTC images. Due to the high amount of spatial correlation in the high and low images, they can easily be further compressed using something similar to JPEG compression, similar to the work done by Ström and Wennersten [Strom and Wennersten 2011]. Techniques that combine the compression parameters of PVRTC and other compression formats can potentially be used in conjunction with image segmentation to increase the compression even further [Krajcevski and Manocha 2014b].

In this paper, we have presented a new technique, *intensity dilation* for compressing textures into LFSM formats. This allows real-time graphics applications that require fast access to on-the-fly generated textures to benefit from texture compression on devices that support LFSM. We believe that, with the rise of distributed graphics applications, compressing textures on the fly will help decrease server side storage costs, and provide overall greater flexibility for developers.

Acknowledgements: This work was supported in part by ARO Contracts W911NF-10-1-0506, W911NF-12-1-0430, and NSF awards 100057 and Intel.

References

AYDIN, T. O. 2010. *Human visual system models in computer graphics*. Doctoral dissertation, Universität des Saarlandes, Saarbrücken. 135

BEERS, A. C., AGRAWALA, M., AND CHADDHA, N. 1996. Rendering from compressed textures. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '96, 373–378. URL: http://doi.acm.org/10.1145/237170.237276, doi:10.1145/237170.237276. 133

FENNEY, S. 2003. Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, HWWS '03, 84–91. URL: http://dl.acm.org/citation.cfm?id=844174.844187. 133, 138

GELDREICH, R., 2013. Fast, high quality ETC1 (ericsson texture compression) block packer/unpacker. http://code.google.com/p/rg-etc1/. 142

IMAGINATION, 2013. PowerVR Insider SDK and Utilities. http://www.imgtec.com/powervr/insider. 134, 138, 140

IOURCHA, K. I., NAYAK, K. S., AND HONG, Z., 1999. System and method for fixed-rate block-based image compression with inferred pixel values. U. S. Patent 5956431. 133

KRAJCEVSKI, P., AND MANOCHA, D. 2014. Real-time low-frequency signal modulated texture compression using intensity dilation. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '14, 127–134. URL: http://doi.acm.org/10.1145/2556700.2556719, doi:10.1145/2556700.2556719. 135

KRAJCEVSKI, P., AND MANOCHA, D. 2014. SegTC: Fast Texture Compression using Image Segmentation. Eurographics Association, Lyon, France, I. Wald and J. Ragan-Kelley, Eds., 71–77. URL: http://diglib.eg.org/EG/DL/WS/EGGH/HPG14/071-077.pdf, doi:10.2312/hpg.20141095. 142

KRAJCEVSKI, P., LAKE, A., AND MANOCHA, D. 2013. FasTC: accelerated fixed-rate texture encoding. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, I3D '13, 137–144. URL: http://doi.acm.org/10.1145/2448196.2448218, doi:10.1145/2448196.2448218. 138, 139

NYSTAD, J., LASSEN, A., POMIANOWSKI, A., ELLIS, S., AND OLSON, T. 2012. Adaptive scalable texture compression. In *Proceedings of the ACM*

SIGGRAPH/EUROGRAPHICS conference on High Performance Graphics, Eurographics Association, HPG '12, 105–114. 133, 138

ON ILLUMINATION, I. C. 2004. *Colorimetry*. CIE technical report. Commission internationale de l'Eclairage, CIE Central Bureau. URL: http://books.google.com/books?id=P1NkAAAACAAJ. 136

OPENGL, A. R. B., 2010. ARB_texture_compression_bptc. http://www.opengl.org/registry/specs/ARB/texture_compression_bptc.txt. 133

SCHNEIDER, J. 2013. GPU-friendly data compression. Presentation at GPU Technology Conference. 133

SERRA, J. 1983. *Image Analysis and Mathematical Morphology*. Academic Press, Inc. 137

SHANNON, C. E. 1948. A mathematical theory of communication. *The Bell System Technical Journal 27* (July, October), 379–423, 623–656. URL: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf. 141

STRÖM, J., AND PETTERSSON, M. 2007. ETC2: texture compression using invalid combinations. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, Eurographics Association, GH '07, 49–54. URL: http://dl.acm.org/citation.cfm?id=1280094.1280102. 138

STROM, J., AND WENNERSTEN, P. 2011. Lossless compression of already compressed textures. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, HPG '11, 177–182. URL: http://doi.acm.org/10.1145/2018323.2018351, doi:10.1145/2018323.2018351. 142

VON ZUR GATHEN, J., AND SIEVEKING, M. 1978. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society 72*, 1, pp. 155–158. URL: http://www.jstor.org/stable/2042554. 134

WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. 2004. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (april), 600–612. doi:10.1109/TIP.2003.819861. 138, 139

WAVEREN, J. M. P. V. 2006. Real-time DXT Compression. *Intel Software Network*. 139

Author Contact Information

Pavel Krajcevski and Dinesh Manocha
Department of Computer Science
201 South Columbia Street, Sitterson Hall
UNC-Chapel Hill
Chapel Hill, NC 27599-3175
pavel@cs.unc.edu dm@cs.unc.edu
http://www.cs.unc.edu/ dm

Pavel Krajcevski and Dinesh Manocha, Fast PVRTC Compression using Intensity Dilation, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 4, 132–145, 2014
http://jcgt.org/published/0003/04/07/

Received:	2013-10-22		
Recommended:	2013-12-09	Corresponding Editor:	Padraic Hennessy
Published:	2014-12-19	Editor-in-Chief:	Morgan McGuire