Real-Time Low-Frequency Signal Modulated Texture Compression using Intensity Dilation

Pavel Krajcevski*
University of North Carolina at Chapel Hill

Dinesh Manocha†
University of North Carolina at Chapel Hill

Abstract

We present a real-time algorithm for compressing textures based on low frequency signal modulated (LFSM) texture compression. 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 a simple two pass algorithm for propagating the extremal intensity values that contribute to these contrast ratios into the compressed encoding. We use our algorithm to compress PVRTC textures in real-time and compare our performance with prior techniques in terms of speed and quality.
`http://gamma.cs.unc.edu/FasTC`

CR Categories: I.4.2 [Image Processing and Computer Vision]: Compression (Coding)—Approximate methods I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and Framebuffer Operations I.3.4 [Computer Graphics]: Graphics Utilities—Software Support

Keywords: texture compression, content pipeline

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. The benefits of compressing textures also include power savings by reducing the number of memory accesses required during rendering. These savings have become increasingly important as trends turn towards low-resource platforms such as mobile GPUs.

Texture compression, first introduced by Beers et al. [1996], exhibits many key differences from standard image compression techniques. The GPU has no way of knowing *a priori* which texture pixels, or texels, will be accessed at any given time. This limitation implies that any texture compression scheme must provide random access to the texels. As a result, most compressed texture formats provide lossy compression at a fixed ratio. Usually, a better compression ratio cedes lower quality texture compression. Many of these trade-offs have been studied in recent texture compression formats [Iourcha et al. 1999][Fenney 2003][Ström and Pettersson 2007][Nystad et al. 2012].

*e-mail:pavel@cs.unc.edu
†e-mail:dm@cs.unc.edu

I3D 2014, March 14 – 16, 2014, San Francisco, California, USA.

ACM 978-1-4503-2717-6/14/03 $15.00

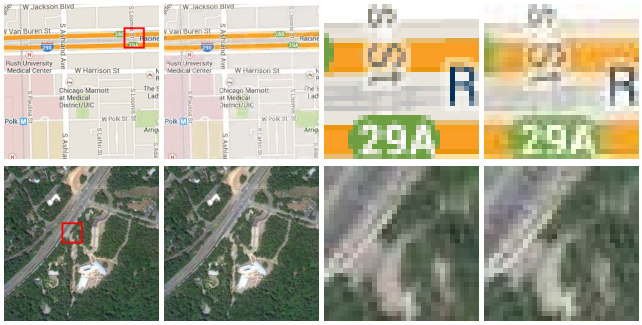


Figure 1: *Real-time texture compression using intensity dilation applied to typical images used in GIS applications such as Google Maps. Each of these 256x256 textures was compressed in about 20ms on a single Intel® Core™ i7-4770 CPU. In each pair of images, the original texture is on the left, and the compressed version is on the right. A zoomed in version of the detailed areas is given on the right. Images retreived from Google Maps.*

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. Cloud-based game services are also emerging where the user experiences high-quality graphics from the lightweight rendering nature of a browser window. Many of these applications require uploading texture data across a network and then to the GPU very frequently. As a result, it is becoming increasingly important to develop real-time high quality texture compression algorithms. Such algorithms also accelerate the content-creation pipeline for 3D games and other real-time applications, allowing developers to better iterate on their products. Finally, another recent trend is to support different texture formats for different GPUs or devices. With fast texture compression schemes, developers only need to store a single texture and compress it on-the-fly based on hardware capabilities of the client, significantly saving on storage space.

In this paper we focus on a widely used texture compression method known as Low-Frequency Signal Modulated Texture Compression (LFSM) [Fenney 2003]. Up until very recently, this texture compression technique was the only technique supported on popular iPhone and iPad devices. 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].

Main Results: We present a novel, fast texture compression algorithm based on intensity dilation for LFSM formats. Our technique evaluates the intensity values of a 32-bit low dynamic range image

to preserve high-contrast areas sensitive to the human visual system during compression [Aydin 2010]. We bleed the high-contrast color information across block boundaries to better represent high frequency regions. We have evaluated our technique on a variety of benchmark images and observe 3 – 3.5X speedup over prior PVRTC algorithms and implementations. We also measure the quality of compression using both raw energy metrics and human perception, and notice considerable benefits over prior schemes.

The rest of the paper is organized as follows. In Section 2, we briefly survey existing texture compression formats. In Section 3, we provide background on LFSM and an analysis of the tractability of finding an optimal solution. In Section 4, we describe our intensity dilation technique and its use for LFSM compression. We present various performance and quality metrics used to evaluate the compression scheme in Section 5, and highlight some limitations in Section 6.

2 Related Work

Random-access preserving image compression techniques first emerged with the Block Truncation Coding (BTC) scheme invented by Delp and Mitchell [1979]. This technique was used to compress 8-bit grayscale images by saving two 8-bit values per 4×4 texel blocks, along with a single bit per texel to index into either of the two values. Afterward, this idea has been generalized and improved by others [Nasrabadi et al. 1990] [Fränti et al. 1994]. The most notable improvement was the extension of Campbell et al. [1986] to color values by providing a 256-bit color palette instead of a grayscale image. Due to hardware costs of multiple memory lookups, this method is regarded as expensive, even with hardware support [Knittel et al. 1996].

Although texture compression has been used for almost three decades [Chandler et al. 1986][Economy et al. 1987], one of the first texture compression algorithms based on Vector Quantization (VQ) was presented by Beers et al. [1996]. In this seminal paper, Beers et al. argue for four main tenets of texture compression: fast hardware decoding speed, preservation of random-access, compression rate versus visual quality, and encoding speed. The presented argument claims that encoding speed could be sacrificed for gains in the other three, but the need for fast encoding algorithms was recognized even then, where a Generalized Lloyd's Algorithm [Gersho and Gray 1991] was used to produce fast non-optimal encodings.

Many formats based on VQ have emerged and are used extensively in current GPUs [Iourcha et al. 1999][Fenney 2003][Ström and Akenine-Möller 2004] [Ström and Akenine-Möller 2005] [Ström and Pettersson 2007][OpenGL 2010][Nystad et al. 2012]. DXT1, introduced by Iourcha et al. [1999] is perhaps the most popular format currently in use. It encodes 4×4 texel blocks as two 565 RGB colors with two bit interpolation indices. In ETC1 and ETC2, introduced by Ström et al. [2005][2007], chromatic values are stored using VQ along with a lookup into a table of intensity values. Over the past few years, higher quality VQ formats, such as BPTC [OpenGL 2010] and ASTC [Nystad et al. 2012], have emerged that split fixed blocks into partitions that are encoded separately along with other features. These newer formats also support encoding high dynamic range textures to complement similar schemes developed in recent years [Roimela et al. 2006][Munkberg et al. 2008][Sun et al. 2008]. There are other texture formats based on VQ, but not currently supported in hardware [Krause 2010].

Fast texture compression has been studied comparatively less than new compression methods. J.M.P. van Waveren was the first to develop a real-time compression scheme for DXT1 [Waveren 2006]. His technique compressed a 4×4 texel block by using the endpoints of the axis aligned bounding box diagonal in three dimensional RGB space. This technique not only proved effective, but also opened the doors for similar techniques for normal maps or YCoCg encoded textures [Waveren and Castaño 2007] [Waveren and Castaño 2008]. Because formats such as DXT are easily parallelizable, many compression techniques also leverage GPUs to generate very good results very quickly [Castaño 2007]. Recently, research has also been done on quickly compressing textures into the newer formats such as BPTC and ASTC [Krajcevski et al. 2013].

3 Low Frequency Signal Modulated Texture Compression

In recent years, low frequency signal modulated texture compression has been widely adopted by many mobile devices. Prior to OpenGL ES 3.0, it has been the only technique available on Apple's iPhone and iPad [Apple 2013]. Despite its popularity, there has not been much work in improving the speed of associated compression techniques. In this section, we give an overview of LFSM texture compression.

3.1 LFSM compressed textures

Like other texture compression formats, LFSM compressed textures are stored in a grid of blocks, each containing information for a $n \times m$ 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 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.

A good LFSM data compression algorithm needs to determine for each block $\mathbf{b}$ both the best high color, $\mathbf{b}_H$, and low color $\mathbf{b}_L$ and the modulation data w for each texel. These values must be chosen such that when decoding a texel $\widetilde{\mathbf{p}}$ surrounded by blocks $\mathbf{b}_A$, $\mathbf{b}_B$, $\mathbf{b}_C$, $\mathbf{b}_D$, the resulting color given by

$$\begin{aligned}\widetilde{\mathbf{p}} = \quad & w\left(l_A\mathbf{b}_{H,A} + l_B\mathbf{b}_{H,B} + l_C\mathbf{b}_{H,C} + l_D\mathbf{b}_{H,D}\right) \\ & +(1-w)\left(l_A\mathbf{b}_{L,A} + l_B\mathbf{b}_{L,B} + l_C\mathbf{b}_{L,C} + l_D\mathbf{b}_{L,D}\right)\end{aligned}$$

best matches the original texel $\mathbf{p}$, where l_k is the appropriate bilinear interpolation weight and $0 \leq w_i \leq 1$.

In order to better understand the problem of compressing into LFSM formats, we present an analytic formulation of the global optimization computation needed for this compression algorithm. Given a source RGBA image of dimensions $n_s \times m_s$ and pixels $\mathbf{p} = \{\mathbf{p}_R, \mathbf{p}_G, \mathbf{p}_B, \mathbf{p}_A\}$, we need to generate a compressed image with dimensions $n_c \times m_c$ where $(n_c, m_c) = (n_s/r_n, m_s/r_m)$ for some $r_n, r_m \in \mathbb{N}$ (typical values are $r_n = 4$ and $r_m = 4$ or 8). The high and low colors $\mathbf{b}_H$ and $\mathbf{b}_L$ of the compressed image will be treated as $1 \times n_c m_c$ vectors. We must also determine modulation values w that correspond to the interpolation weights between the two bilinearly interpolated endpoints. Each channel of an image reconstructed from compressed data can be described by a matrix equation as follows:

$$\widetilde{\mathbf{p}_k} = WQ\mathbf{b}_{H,k} + (\mathbf{I} - W)Q\mathbf{b}_{L,k}, \tag{1}$$

where Q is the $n_s m_s \times n_c m_c$ matrix corresponding to the bilinear

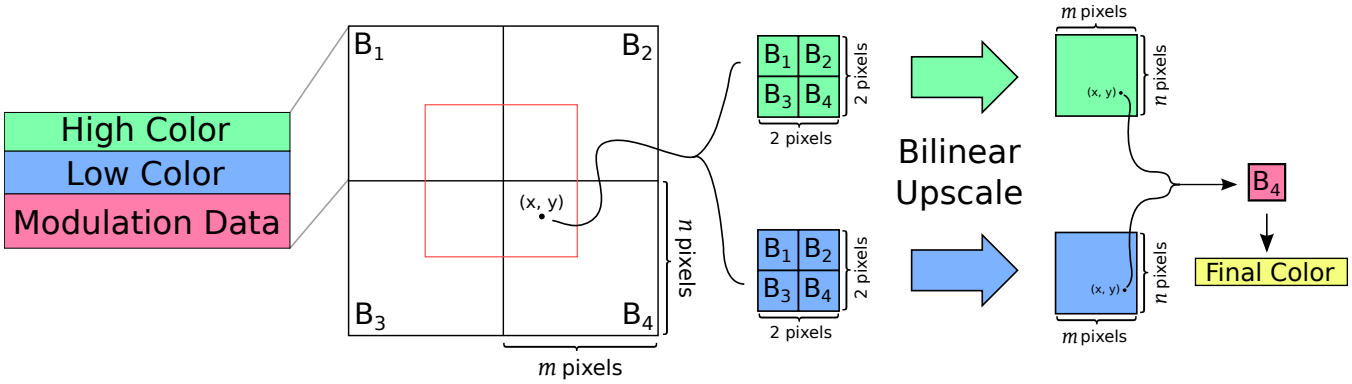


Figure 2: *Representation of low-frequency signal modulated texture data. The compressed data is stored in blocks that contain two colors, high and low, along with modulation data for each texel within the block. When looking up the color value for texel at location (x, y), information is used from the blocks whose centers are the four corners of a rectangle that encompass the texel. The high and low colors are separately used to generate two block sized images using bilinear interpolation, and then the modulation value of the texel's corresponding block is used in conjunction with these upscaled images in order to produce the final color.*

weights of each pixel given by

$$Q_{i,j} = \frac{1}{r_n r_m} \begin{cases} 0 & \text{if } |x_i - r_n x_j| \geq r_n \text{ or} \\ & \text{if } |y_i - r_m y_j| \geq r_m, \\ (r_n - l_x)(r_m - l_y) & \text{if } r_n x_j \leq x_i \text{ and } r_m y_j \leq y_i, \\ l_x (r_m - l_y) & \text{if } r_n x_j > x_i \text{ and } r_m y_j \leq y_i, \\ l_y (r_n - l_x) & \text{if } r_n x_j \leq x_i \text{ and } r_m y_j > y_i, \\ l_x l_y & \text{if } r_n x_j > x_i \text{ and } r_m y_j > y_i, \end{cases}$$

$$x_i = i \bmod n_s \qquad y_i = \left\lfloor \frac{i}{n_s} \right\rfloor$$

$$x_j = j \bmod n_c \qquad y_j = \left\lfloor \frac{i}{n_c} \right\rfloor$$

$$l_x = x_i \bmod r_n \qquad l_y = y_i \bmod r_m.$$

W is the unknown diagonal matrix of dimensions $n_s m_s \times n_s m_s$ containing the modulation values w_i. The product WQ corresponds to both applying modulation weights and bilinear interpolation between the values $\mathbf{b}_H$. The values $\mathbf{b}_L$ are similarly weighted with the product $(\mathbf{I} - W)Q$. We have four of these systems, one corresponding to each channel $k \in \{R, G, B, A\}$ that are all coupled by W. This formulation gives us $n_s m_s + 8 n_c m_c$ unknowns (one for each w_i and one for each channel in each color $\mathbf{b}_H$, $\mathbf{b}_L$) and $4 n_s m_s$ equations. Since $(n_s, m_s) = (r_n n_c, r_m m_c)$, we will have as many equations as unknowns when $4 r_n r_m \geq r_n r_m + 8$. For any reasonable compression format, $r_n, r_m > 2$, so a solution exists in the continuous domain at least.

3.2 High Complexity

The first thing to notice about Equation (1) is that it is a non-linear system, which is typically solved using iterative solvers. Furthermore, for n_s, $m_s = 256$ and $r_n, r_m = 4$, the size of our solution vector will have dimension

$$n_s m_s + 8 n_c m_c = 2^8 2^8 + 2^3 2^6 2^6 > 2^{16}.$$

This is too large for any non-linear solver to find a solution to, especially in real-time.

We can reduce the complexity of the system by first using an approximation for W. In this case, we can bundle up the problem in Equation (1) by combining W and Q to get an equation of the form:

$$\mathbf{p} = A \begin{bmatrix} \mathbf{b}_H \\ \mathbf{b}_L \end{bmatrix}. \tag{2}$$

This becomes a linear system where the resulting matrix A is large, non-square, and high rank, since each row has $8 n_c m_c$ elements and thirty-two non-zero elements. The non-zero elements in row i are the elements corresponding to the channels in the high and low colors of the blocks that affect pixel i.

Finally, the biggest impediment to using this formulation to compute an efficient solution is the fact that the solution to our problem must be stored within an LFSM data format, which necessitates discretizing our values. Quantization of the solution to the $Ax = b$ problem from the real numbers to integers does not provide an optimal solution 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 solution is impractical.

3.3 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 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.

The refinement steps take the approximation and generate an initial value for the W matrix from Section 3.2. This initial value of W is used to look at a smaller version of the linear system given in Equation (2) that corresponds to 2×2 blocks and the pixels that they influence while treating the blocks outside of these four as fixed. This constrains the problem to a 121×8 sized linear system, which is solved using singular value decomposition. Fenney mentions that 2-4 iterations of this refinement is enough for good quality results [Fenney 2003].

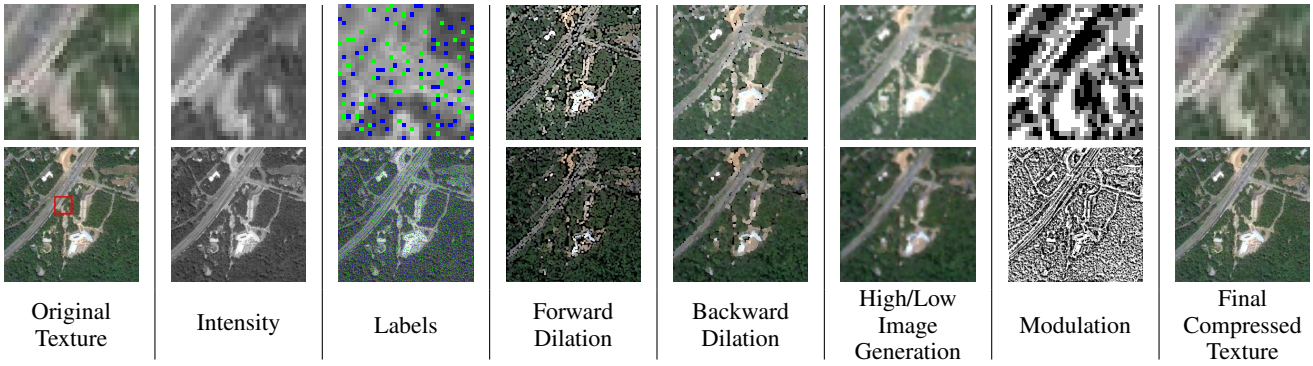


Figure 3: *The different stages of the algorithm. Original texture: the texture we are compressing explicitly marked with an area of interest which is depicted in the zoomed in versions. Intensity: original image and zoomed in region in grayscale. Labels: labeled image and zoomed in region of texels with intensity values larger than their neighbors (green) and lower than their neighbors (blue). Forward dilation: after the first pass of the algorithm, both the high image containing local intensity maxima (top) and the low image containing local intensity minima (bottom) have been dilated forward. Backward dilation: after the second pass of the algorithm, both of the images have been completely dilated. High/Low image generation: Downscaled images that resulted from averaging all of the texels in a block of the dilated images. Modulation: computed optimal modulation values for the original image and the zoomed in region, given the computed high and low images. Final compressed texture: The resulting compressed texture and the corresponding zoomed in region. Original image retreived from Google Maps.*

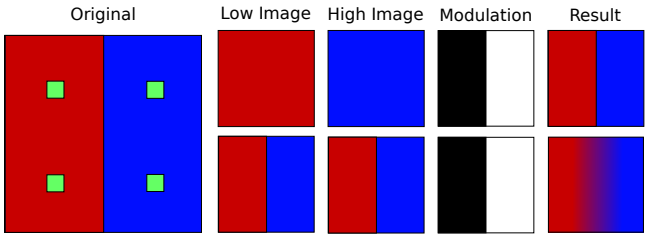


Figure 4: *A red and blue texture is compressed using LFSM compression. Green positions are block centers. Since the border between blue and red areas aligns with block borders, the optimal compression is to store one color as the high color of each block, and another color as the low color of each block. The modulation data is used to reconstruct the original image.*

4 LFSM compression using Intensity Dilation

Given the high complexity of computing the optimal solution of Equation (1), we present an alternative technique for real-time texture compression. The basis for our approach resides in the well studied foundations of the human visual system's sensitivity to contrast [Aydin 2010] [Thompson et al. 2011]. 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.

Due to the way that LFSM compressed textures store the compressed data, as in Figure 2, there is an inherent filtering procedure that takes place during retrieval of texel data. During the bilinear upscale of the colors stored per block, adjacent blocks must maintain the extreme values in order to preserve the edges. In Figure 4, the optimal compression scheme to preserve the edge would be to store each color, red and blue, in all four blocks that cover the edge. The modulation can be used to choose the appropriate pixels from either of the two images. If any of the blocks have either red or blue as both high and low colors, then the result would never be able to fully encode the edge because the edge pixels would be blurred from the bilinear upscale. In order to fully encode areas of high contrast, such as edges across very different colors, the high and low intensity texels must be represented in all blocks that influence that region during decompression. In this section, we cover the basic principles behind the full intensity dilation algorithm, and present a two-pass approximation algorithm that performs the encoding.

4.1 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 [Smith and Guild 1931]

$$I(x) = R_x * 0.2126 + G_x * 0.7152 + B_x * 0.0722.$$

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.

4.2 Intensity Dilation

In order to capture the contrast features of an image, we propose the use of 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. Figure 5 shows how a star can be dilated by using a small disk to create a larger star with rounded corners. In LFSM texture compression, the input textures have at least three 8-bit channels that must be dilated. 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, as shown in Figure 5, 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

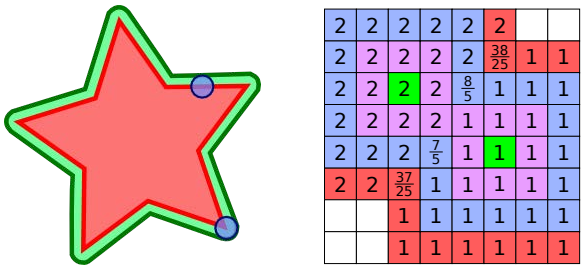


Figure 5: *Examples of dilation. Left: a red star is dilated by a smaller circle into the green star with rounded corners. Right: Two pixels, denoted in green, are dilated three times using a 3×3 pixel box. If an empty pixel $\mathbf{p}$ is to be filled during dilation from multiple pixels $\mathbf{q}_i$ of different values, then the value stored for $\mathbf{p}$ will be the average of the $\mathbf{q}_i$. The picture is labeled with the values that the pixels would take after dilation of the initial pixels. The pixels that have fractional labels denote the value that they would have taken between labels one and two.*

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 completely capture the important features of a texture, the intensity labels of the image (Section 4.1) must be dilated until they influence neighboring block values. In LFSM, blocks cover $r_n \times r_m$ 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 d units away, where d is defined as

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

In order to properly influence the colors of a block that covers a given labeled pixel, we must dilate each of the extrema d 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 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 high and low colors and bilinearly upscaling them to get the proper interpolation extremes. We then choose the optimal interpolation weight based on the restrictions imposed by the format. For a complete overview of the algorithm, see Figure 3.

4.3 Two Pass Algorithm

In the previous sections, we present an approach which requires eight stages with a simple implementation. One to convert the image to intensity values, one to label the maxima/minima, and three to perform the dilation for each image containing intensity minima and maxima. In this section, we propose a scheme to approximate this pipeline using two passes: a forwards and a backwards pass. For each pixel $\mathbf{p}$, we store a per-pixel cache that lazily stores an intensity value, a high label, and a low label. In other words, we do not compute the intensity value until it is needed, at which point we store it for future use. Each label has a distance value $dist(\mathbf{p})$, and a list of indices into the pixels that correspond to the maxima or minima that the current pixel is dilated from.

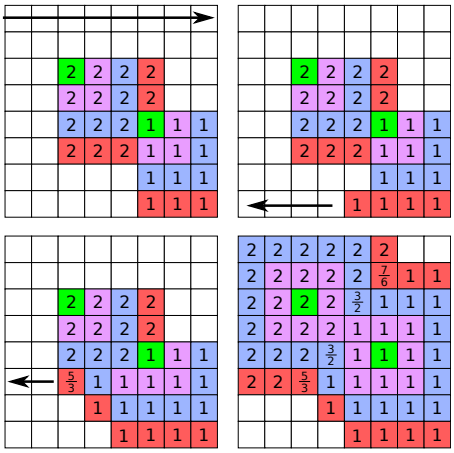


Figure 6: *Our fast approximate dilation strategy. We perform the extrema calculation and dilation in two passes. Top Left: First pass, traverse the pixels from left to right, top to bottom labeling and dilating extrema in the order of traversal as we encounter them. Top right, bottom left, bottom right: Second pass, traverse the pixels from right to left, bottom to top. At each pixel, assign the label corresponding to the average of the pixels with the lowest distance to their respective labels.*

Forward Pass: We traverse pixels from left to right starting at the top-left corner of the image. At each pixel $\mathbf{p}$, we determine whether or not it is an extrema by looking at neighboring intensity values. Whenever an intensity value is computed, it is subsequently cached to avoid further computation. If the pixel is a local maximum or local minimum, we set $dist(\mathbf{p}) = 0$, and continue. If the pixel is not a local extrema, we investigate the values to the left and above the pixel to determine its distance from a local extrema. We need not look at any other neighbors due to the direction of this iteration. We also assume the painter's algorithm, so that if two or more local extrema conflict, they will be overwritten (see Figure 6).

Backward Pass: After we have labeled the pixels with their local extrema in one direction, we may proceed to dilate the pixels in the opposite direction. We dilate backwards by starting in the bottom right corner and proceed from right to left. At each pixel $\mathbf{p}$, both labeled and unlabeled, we find the set of neighbors of $\mathbf{p}$, $\{\mathbf{c}\}$, that have the least value $d = dist(\mathbf{c})$. If d is already the maximum number of dilations or $dist(\mathbf{p}) < d + 1$, then we ignore this texel. If $dist(\mathbf{p}) = d + 1$, then we concatenate $\{\mathbf{c}\}$ to the list for $\mathbf{p}$. Otherwise, we simply change the list for $\mathbf{p}$ to be $\{\mathbf{c}\}$ and set $dist(\mathbf{p}) = d + 1$.

After both passes, the list of indices stored at each pixel are approximately those pixels that would contribute to the final color of the pixel during a decoupled dilation of the extrema. This approximation can be seen in the difference between the final labels in Figures 5 and 6. During backwards dilation of a pixel $\mathbf{p}$, we do not have the proper information yet about whether or not pixels have dilated to the left above $\mathbf{p}$. This problem is most noticeable by the missing pixels in the bottom row of the bottom-right image in Figure 6. This can be mitigated by handling the special case whenever we place a non-maximally distant label above a label with a larger distance.

Once both passes are complete, at each pixel we have stored the intensity, and the closest minimum and maximum intensity pixels. When averaging the pixels in each block of the high and low images, we can simultaneously find the minimum or maximum intensity for the block. For a $n \times m$ block with N non-labeled pixels, we store as block colors the sum of each averaged label weighted with $\frac{1}{nm}$ and the pixel corresponding to the minimum or maximum

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

Figure 7: *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].*

intensity weighted with $\frac{N}{nm}$.

5 Results

The only LFSM 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 3.3. 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 or LFSM in general. 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].

5.1 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 7, PSNR does not correlate with visual fidelity.

For this reason, we also include 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.

5.2 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 5.3, we suffer from aggressive averaging artifacts in these areas. Most images do not have large homogeneous areas, and consequently compression speed is tightly correlated with the size of the texture.

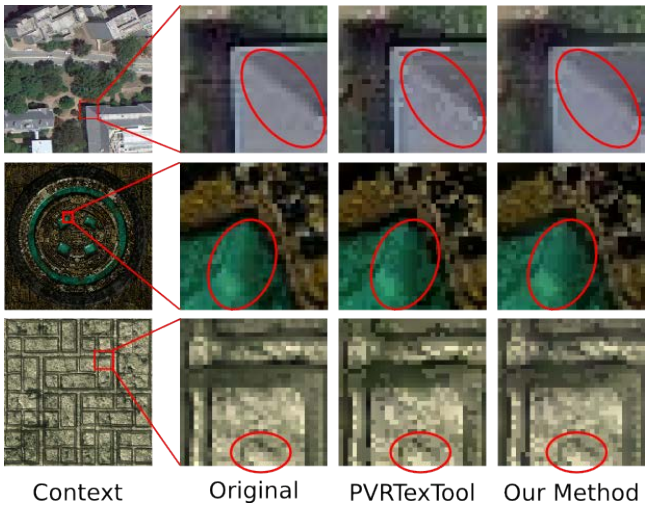


Figure 8: *Investigation of areas with high detail in some common mobile graphics images. We notice that the texture compressed using intensity dilation maintains the smoothness of many image features, while the original PCA based approach leaves blocky streaks. Images courtesy of Google Maps and Simon Fenney.*

A majority of the speedups in our method come from minimizing the number of times that we traverse the entire texture. In doing so, we minimize the number of penalizing cache misses. Furthermore, during the optimized dilation step described in Section 4.3, the per-pixel cache that stores list of *indices* to pixels means that we are not averaging pixels until the very end. This also has the benefit of being cache friendly by avoiding costly memory lookups during the dilation process.

5.3 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. In Figure 8, the differences between the two methods are noticeable. 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 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 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]:

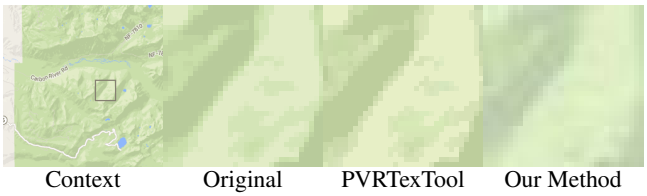
$$E = -\sum p_i \log p_i,$$

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,

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

Image	Compression Speed (ms)		SSIM		Peak Signal to Noise Ratio		Entropy
	Our Method	PVRTexTool	Our Method	PVRTexTool	Our Method	PVRTexTool	
streets	17.76	63.74	0.9666	0.9850	33.63	32.35	1.546
gradient	20.30	65.11	0.9797	0.9673	30.17	30.97	7.528
satellite	20.93	64.92	0.9488	0.9180	32.09	30.43	6.963
mountains	21.77	66.63	0.9138	0.9620	30.01	34.47	3.960
bricks	21.91	65.45	0.9476	0.9331	27.44	26.35	7.468
gametex	97.25	264.68	0.9531	0.9225	30.20	29.80	6.552
lorikeet	97.39	263.14	0.9455	0.9111	30.75	31.37	7.386

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 9: *Detailed investigation of areas with high pixel homogeneity. Unlike the images in Figure 8, 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.*

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 9.

6 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. 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.

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.*

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 3.3. 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 4.1 and 4.2, to come up with a parallelizable algorithm that exploit both SIMD and multiple cores, such as consumer GPUs.

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

APPLE, 2013. Best practices for working with texture data. https://developer.apple.com/library/ios/.

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

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.

CAMPBELL, G., DEFANTI, T. A., FREDERIKSEN, J., JOYCE, S. A., AND LESKE, L. A. 1986. Two bit/pixel full color encoding. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '86, 215–223.

CASTAÑO, I. 2007. High Quality DXT Compression using CUDA. *NVIDIA Developer Network*.

CHANDLER, J. E., BUNKER, W. M., ECONOMY, R., FADDEN, J. R. G., AND NELSON, M. P., 1986. Advanced video object generator. U. S. Patent 4727365.

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

ECONOMY, R., FADDEN, J. R. G., AND STEINER, W. R., 1987. Yiq based color cell texture. U. S. Patent 4965745.

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.

FRÄNTI, P., NEVALAINEN, O., AND KAUKORANTA, T. 1994. Compression of digital images by block truncation coding: A survey. *The Computer Journal 37*, 4, 308–332.

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

GERSHO, A., AND GRAY, R. M. 1991. *Vector quantization and signal compression*. Kluwer Academic Publishers.

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

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.

KNITTEL, G., SCHILLING, A., KUGLER, A., AND STRAER, W. 1996. Hardware for superior texture performance. *Computers & Graphics 20*, 4, 475–481.

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.

KRAUSE, P. K. 2010. ftc–floating precision texture compression. *Computers Graphics 34*, 5, 594–601. CAD/GRAPHICS 2009, Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference, Vision, Modeling & Visualization.

MUNKBERG, J., CLARBERG, P., HASSELGREN, J., AND AKENINE-MLLER, T. 2008. Practical HDR Texture Compression. *Computer Graphics Forum 27*, 6, 1664–1676.

NASRABADI, N. M., CHOO, C. Y., HARRIES, T., AND SMALLCOMB, J. 1990. Hierarchical block truncation coding of digital HDTV images. *IEEE Trans. on Consum. Electron. 36*, 3 (Aug.), 254–261.

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.

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

ROIMELA, K., AARNIO, T., AND ITÄRANTA, J. 2006. High dynamic range texture compression. In *ACM SIGGRAPH 2006 Papers*, ACM, SIGGRAPH '06, 707–712.

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

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

SHANNON, C. E. 1948. A mathematical theory of communication. *The Bell System Technical Journal 27* (July, October), 379–423, 623–656.

SMITH, T., AND GUILD, J. 1931. The C.I.E. colorimetric standards and their use. *Transactions of the Optical Society 33*, 3.

STRÖM, J., AND AKENINE-MÖLLER, T. 2004. PACKMAN: texture compression for mobile phones. In *ACM SIGGRAPH 2004 Sketches*, ACM, SIGGRAPH '04, 66–.

STRÖM, J., AND AKENINE-MÖLLER, T. 2005. iPACKMAN: high-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, HWWS '05, 63–70.

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.

SUN, W., LU, Y., WU, F., AND LI, S. 2008. DHTC: an effective DXTC-based HDR texture compression scheme. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, Eurographics Association, GH '08, 85–94.

THOMPSON, W., FLEMING, R., CREEM-REGEHR, S., AND STEFANUCCI, J. K. 2011. *Visual Perception from a Computer Graphics Perspective*, 1st ed. A. K. Peters, Ltd.

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.

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.

WAVEREN, J. M. P. V., AND CASTAÑO, I. 2007. Real-time YCoCg-DXT Compression. *NVIDIA Developer Network*.

WAVEREN, J. M. P. V., AND CASTAÑO, I. 2008. Real-time Normal Map DXT Compression. *NVIDIA Developer Network*.

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