APPROVAL SHEET

Title of Thesis: Quality Guided Variable Bit Rate Texture Compression

Name of Candidate: Wesley Griffin
Doctor of Philosophy,
2016

Thesis and Abstract Approved: _____

Dr. Marc Olano
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Thesis: Quality Guided Variable Bit Rate Texture Compression

Wesley Griffin, Doctor of Philosophy, 2016

Thesis directed by: Dr. Marc Olano, Associate Professor
Department of Computer Science and
Electrical Engineering

The primary goal of computer graphics is to create images by rendering a scene under two constraints: quality, producing the image with as few artifacts as possible, and time, producing the image as fast as possible. Technology advances have both helped to satisfy these constraints, with Graphics Processing Unit (GPU) advances reducing image rendering times, and to exacerbate these constraints, with new HD and virtual reality displays increasing rendering resolutions. To meet both constraints, rendering uses texture mapping which maps 2D textures onto scene objects. Over time, the count and resolution of textures has increased, resulting in dramatic growth of data storage requirements. Compression can help to reduce these storage requirements.

I present a rigorous texture compression evaluation methodology using final rendered images. My method can account for masking effects introduced by the texture mapping process while leveraging the perceptual-rigor of current Image Quality Assessment metrics. Building on this evaluation methodology, I present a demonstration of guided texture compression optimization that minimizes the bitrate of compressed textures while maximizing the quality of final rendered images. Guided texture compression will help with the scalability problem for optimizing texture compression in real-world scenarios.

Quality Guided Variable Bit Rate Texture Compression

Wesley Griffin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

DEDICATION

For Connor.

Anything is possible with dedication.

ACKNOWLEDGEMENTS

I would first like to thank my amazing wife Kim for her constant support and understanding for the last eight years. I could not have completed this without her. I also want to thank my family for their support and encouragement.

My advisor, Dr. Marc Olano, provided the initial idea, on-going direction, and support. Thank you for helping me to see this work completed.

I would also like to thank my dissertation committee for their time spent reading my proposal and dissertation and answering my questions.

There have been many anonymous paper reviewers that took the time to read and provide feedback on the published portions of my dissertation.

Thanks also to my NIST colleagues who also reviewed paper drafts and answered questions and my VANGOGH lab mates.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ASTC Adaptive Scalable Texture Compression. vi, 3, 6, 12, 21, 39, 42, 43, 52, 53

CIELAB CIE $L^*a^*b^*$ color space. vi, 14, 17–19, 36, 40, 41, 55, 57, 59

DMOS Difference Mean Opinion Score. 5, 23, 29–32

FBR Fixed Bit Rate. 2, 8, 9, 64

GPU Graphics Processing Unit. 1–3, 5, 13, 34, 62, 64

IQA Image Quality Assessment. 3, 4, 8, 13, 14, 23, 64, 65

IW-SSIM Information Content Weighted Structural Similarity Index Metric. 17

MS-SSIM Multiscale Structural Similarity Index Metric. 15, 17

MSE Mean Square Error. 5, 33, 65

PSNR Peak Signal-to-Noise Ratio. 3, 5, 33, 65

RMS Root Mean Square. 36, 40, 42, 43, 45

SHAME Spatial Hue Angle Metric. 20

SSIM Structural Similarity Index Metric. 15–17, 36, 40–42, 49, 50, 55, 57

VBR Variable Bit Rate. 2, 8, 9, 13, 64

VIF Visual Information Fidelity. 17

VSNR Visual Signal-to-Noise Ratio. 14

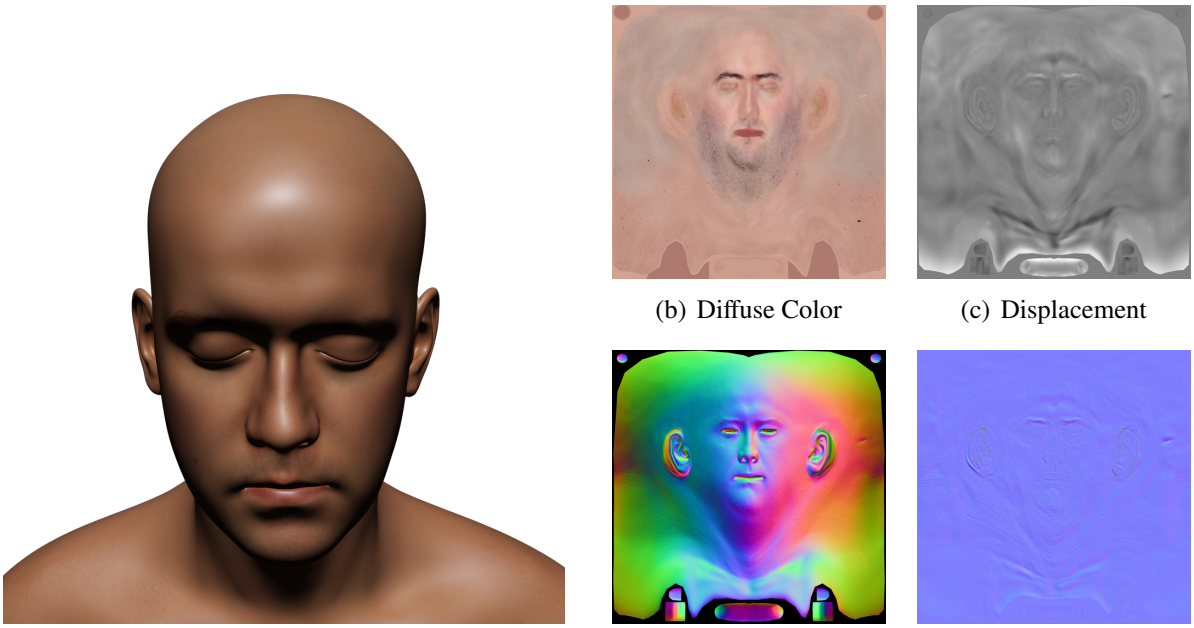
Chapter 1

INTRODUCTION

The primary goal of computer graphics is to create realistic images, or frames, from a description of a scene, a process called rendering. In all rendering a key constraint is the quality of the rendered frame. In real-time rendering, primarily used in video games, there is an additional constraint: time. Real-time systems must render successive frames fast enough to provide the illusion of motion and to maintain interactivity. In most cases, a real-time rendering system has only 16 or 11 milliseconds to render a single frame.

Despite this time constraint, real-time rendering cannot sacrifice quality. Video game consumers expect new games to provide increased graphics realism. These expectations have largely driven the growth in Graphics Processing Unit (GPU) performance over the past decade. Additionally, the growth of high defintion displays in the last few years further increases the quality expectations. Players expect video games to provide interactive and fluid motion at 1920×1080 and higher pixel resolution.

To meet both time and quality constraints real-time rendering uses texture mapping which maps 2D textures onto objects in the scene. Textures store artist-created data that the rendering algorithm uses to render a frame. This data is often color and normal basis vectors for points on models. Figure 1.1 is an example of a head model and the corresponding textures used to render the head. Figure 1.1(b) is the diffuse color texture that provides the base color of the model. Figure 1.1(c) is the displacement texture that adds local geometrical detail by displacing the surface in the normal direction by the displacement amount. Figure 1.1(d) is the world-space normal vector texture that indicates the normal direction at each point on the surface. Figure 1.1(e) is the tangent-space normal vector texture that encodes the normal vector into a basis that is tangent to the surface.



(a) Final Rendered Image (b) Diffuse Color (c) Displacement (d) World Space (e) Tangent Space

Figure 1.1: Final rendered image and the textures used for rendering.

As GPU performance has increased, rendering algorithm complexity has increased to improve graphics realism. While early rendering algorithms stored only object colors in textures, current rendering systems store many different types of data thereby increasing the number of textures required to render one object. Furthermore, increasing pixel resolutions from high definition displays have pushed texture resolutions from 256×256 pixels to 2048×2048 pixels, an increase of 64 times. This increase in both numbers and sizes of textures has resulted in dramatic growth of texture data storage requirements.

To reduce texture mapping memory requirements, GPUs provide Fixed Bit Rate (FBR) compression formats that are efficiently decompressed in hardware. The most common formats have compression ratios of 6:1 or 4:1. However, these formats overcompress some blocks, increasing compression artifacts, and undercompress other blocks, increasing compressed size. We show (Olano et al. 2011) that these FBR formats can no longer meet the storage and quality requirements of current video games, and introduce a Variable Bit Rate (VBR) compression format that achieves a reduction in compressed size of roughly

one-third.

Real-time rendering makes extensive use of texture compression for reducing storage size on the GPU, bus bandwidth for uploading textures, and memory bandwidth for rendering. Current GPU compression algorithms use fixed bit-rates (Iorcha, Nayak, and Hong 1999; OpenGL ARB 2009) and recent research has explored reducing the bit rate (Ström and Pettersson 2007; Khronos Group 2013) as well as using variable bit-rate algorithms (Inada and McCool 2006; Olano et al. 2011).

Recently, Khronos Group (2013) introduced ASTC, which is modeled after the existing block compression algorithms but supports varying the number of texels per block or *block footprint* in a texture to enable granular changes to the bit rate. This varying block footprint feature presents an interesting optimization use-case.

For situations that only use an extremely small number of textures, manually tuning the ASTC block footprint on each texture to maximize compression rate while visually ensuring compressed texture quality might be possible. Modern video games, however, have roughly 5000 textures and neither manual tuning nor visual evaluation will scale. One solution to this problem is to automate the tuning and inspection. The accepted method for this automation is to objectively compare the compressed image to the uncompressed image.

The universal technique for objective image comparison is to compute the Peak Signal-to-Noise Ratio (PSNR) between the uncompressed and compressed images. However, research has led to the realization that PSNR does not correlate with how humans evaluate images (Wang and Bovik 2006). Objective Image Quality Assessment (IQA) research has since thus focused on metrics with strong correlation to human perception. I hypothesize that an objective IQA metric can be used to automatically guide texture compression.

Current top-performing IQA metrics are formulated for luminance (gray-scale) natural images such as real-world photographs or realistically rendered images. Textures, on the other hand, store many types of data, including color, as in Figure 1.1(b) and 3D vectors, as in Figures 1.1(d) and 1.1(e). Thus, it would be a mistake to directly evaluate compressed

texture quality using existing IQA metrics.

The textures, however, are used to render a final realistic (natural) color image. This image could be compared with an undistorted image using an objective IQA metric. By using compressed and uncompressed textures to render the respective images, indirect evaluation of texture compression quality is possible by comparing the final rendered images. Evaluating the final rendered image would solve the problem of textures not being natural images and enable guided texture compression. However, one final problem remains: the final rendered images are color, but the objective IQA metrics have been formulated for luminance images.

For image evaluation, ignoring chrominance distortions is acceptable and the luminance metrics perform very well. For guided texture compression, however, chrominance distortions cannot be ignored. The compression algorithm will recognize that changes in chrominance do noelwood.nist.gov/t affect the quality evaluation, and will increase the compression ratio by eliminating or grossly distorting the color. Furthermore, existing color objective IQA metrics do not perform well compared with the luminance metrics (Ajagamelle, Pedersen, and Simone 2010; Čadík et al. 2012).

1.1 Contributions

I make three specific contributions in this work: 1) a chrominance distortion image database, 2) a rigorous texture compression evaluation methodology, and 3) a demonstration of guided texture compression optimization.

1.1.1 Chrominance Distoration Image Database

To aid research and evaluation of color IQA metrics, I have developed a chrominance distortion image database. This database consists of 24 reference images and 500 distorted images spread across five different chromatic distortions. The chromatic distortions were chosen to model common techniques in image compression as well as other chromatic effects, such as chromatic aberration and chromatic noise. I performed a user study to generate

subjective evaluations of each distorted image as an average Difference Mean Opinion Score (DMOS) value. This database and the user study are discussed further in Chapter 3.

1.1.2 Texture Compression Evaluation

In evaluating texture compression, the most common approach is to use some combination of the Mean Square Error (MSE), PSNR, or visual image inspection on the individual textures. While comparing or inspecting individual textures is straightforward to implement, this method does not properly account for the two ways in which textures are used.

Textures are not single images but instead are 1) mapped onto geometric objects which are rendered from a viewpoint (texture mapping) and 2) combined into texture sets, such as a diffuse and bump map, which are used to evaluate the rendering equation. In both cases, textures are used to render a final image from a specific viewpoint. In a rendered image, there are likely masking effects from geometric distortion or overlapping textures that affect that quality of the image (Ferwerda et al. 1997).

To account for these masking effects, I introduce a texture compression evaluation methodology using final rendered images. This methodology is discussed further in Chapter 4. My results show:

- Geometric and texture set masking occurs and cannot be detected by evaluating individual textures.
- The relative contribution between geometric and texture set masking varies by type of texture.
- Perceptual sensitivity to masking varies with the type of texture, such as diffuse, gloss or bump maps.
- Current GPU compression algorithms are too conservative and bit rates can be reduced while maintaining final rendered image quality.

1.1.3 Texture Compression Optimization

Modern texture compression algorithms (Khronos Group 2013; Skodras, Christopoulos, and Ebrahimi 2001) have many "knobs" for fine-tuning the compression. For example, the ASTC algorithm (Khronos Group 2013) has seven setting variables, five of which are discrete, while the other two are continuous, and all of them having a fixed range. These knobs are used to fine-tune the compression algorithm by trading speed for bitrate. Instead of having end-users pick values for each of these knobs, setting "presets" are created that set specific values for each variable.

Even with presets, the effect of the variables on bitrate and quality can be non-obvious to non-experts. Furthermore, even for experts, manually tuning each variable is a time-consuming process with usually very little payoff and enumerating all possible states of the variables is infeasible since there are more than 10^{12} possible states. Finally, modern video games uses thousands of textures, further compounding the problem.

The large state space created by combining the knobs, however, suggests using a search algorithm for guided optimization. Leveraging my compression evaluation methodology, I introduce a procedure that can optimize an energy function incorporating bitrate and objective image quality. This procedure is described in Chapter 5.

My results present:

- An energy function evaluated by applying perceptually rigorous objective image quality assessment metrics to compare compressed textures to uncompressed textures over a Monte-Carlo sampling of viewpoints.

- A measure to determine if the Monte-Carlo estimate is accurate enough.

- A variance analysis in the Monte-Carlo estimate showing that 2,048 viewpoint samples provides a sufficiently accurate estimate of compress texture quality with a sizable margin of error.

1.2 Significance

Guided texture compression will help with the scalability problem for optimizing texture compression in video games. An automatic approach should maximize both compression ratio and final rendered image quality more quickly and more objectively than a human. By reducing the time to optimize compression, artists can use larger textures and programmers can use more textures to further increase rendering quality. Given these expected results, guided compression could see wide application across the video game industry.

Furthermore, very little research exists on automatic rendering optimization. The closest example is the work done in automatic shader simplification (Section 2.4). It is probable that other parts of the rendering pipeline, beyond texture compression, could be optimized by evaluating the quality of final rendered images. Possible examples include texture filtering to reduce texture sample aliasing, texture encoding of 3D vectors and other floating point data, and different algorithms for modeling light transport. Guided texture compression is a first step that would validate the larger technique of automatic rendering optimization.

Finally, and perhaps most importantly, the field of computer graphics in general has relied on manual visual evaluation of rendering algorithms. Typical research papers present images of different algorithms side-by-side and either highlight the presence of visual differences or note the absence of visual differences. This evaluation is highly subjective based on the person evaluating the images. When objective evaluation does occur, the luminance metrics are very often misused by averaging a per-color-channel quality value. Appropriately applying objective IQA metrics provides more rigorous evaluation technique.

Chapter 2

BACKGROUND

This dissertation explores two areas of existing research, texture compression and image quality assessment, in the context of an optimization framework. Section 2.1 reviews image compression and surveys current research in both FBR and VBR texture compression. Next, Section 2.2 reviews image quality assessment and surveys current top-performing luminance and color objective IQA metrics. Finally, Section 2.4 discusses automatic shader simplification.

2.1 Image Compression

Compression exploits the existence of redundant information in data to reduce the number of bits needed to represent the data. Compression algorithms can be *lossless* or *lossy*. Lossless algorithms reconstruct a decompressed file that exactly matches the original uncompressed file while lossy algorithms reconstruct a file that does not exactly match the original file. The human visual system is not sensitive to several image characteristics, including luminance distortions, chrominance distortions, and high frequency details, so lossy compression is well suited for image compression. The human visual system will perceive a reconstructed image from lossy compression as close, if not identical, to the uncompressed image if the distortions are small.

Vector quantization is a common technique for lossy image compression. A 2D uncompressed image is divided into blocks of pixels and the colors at each pixel are quantized to a set of per-block representative colors, or codebook. Vector quantization is inherently FBR as each block of pixels is encoded using the same number of bits. Since the blocks are the same size, block locations in the compressed file are computed with a simple offset. Further-

more, compressed blocks can be decompressed independently since the blocks do not have any dependencies and can be randomly accessed. These two qualities, known compressed block locations and independent decompression through random access, enables hardware implementation of efficient vector quantization decompression. The drawback, however, is that the fixed block size will cause overcompression in some blocks and undercompression in other blocks. Section 2.1.2 covers relevant FBR algorithms.

A different approach to compression is to model the information entropy in the uncompressed data using a statistical model. The probability model is used to determine how to encode bit patterns: frequently occurring bit patterns are encoded in few bits, while infrequent patterns are encoded in more bits. By transforming the data to increase entropy, encoding efficiency is improved and compression is achieved. These approaches are inherently VBR since blocks are encoded with different numbers of bits.

Figure 2.1 shows how VBR algorithms are a pipeline composed of three steps: transform, quantize, and encode. The transform step improves the probability characteristics of the data and is usually includes a color space transform. Improving the probability characteristics, often in the form of a difference from a prediction, enables more efficient encoding. The two most common transformations are the discrete cosine transform and discrete wavelet transform. The quantize step is non-reversible and only happens for lossy compression. The encode step then encodes the quantized values to create the compressed image. Encoding is performed using a lossless entropy encoding algorithm such as Huffman coding (Huffman 1952), arithmetic coding (Witten, Neal, and Cleary 1987; Howard and Vitter 1994), range coding (Martin 1979), or asymmetric numeral system coding (Duda 2014).

JPEG (Wallace 1991) uses the discrete cosine transform where the low frequency coefficients of the transform contain most of the image detail while the high frequency coefficients should be zero or close to zero. Embedded Zerotree Wavelets (Shapiro 1993) and JPEG-2000 (Skodras, Christopoulos, and Ebrahimi 2001) use the discrete wavelet transform where the highest level of the transform contains most of the image detail while the lower

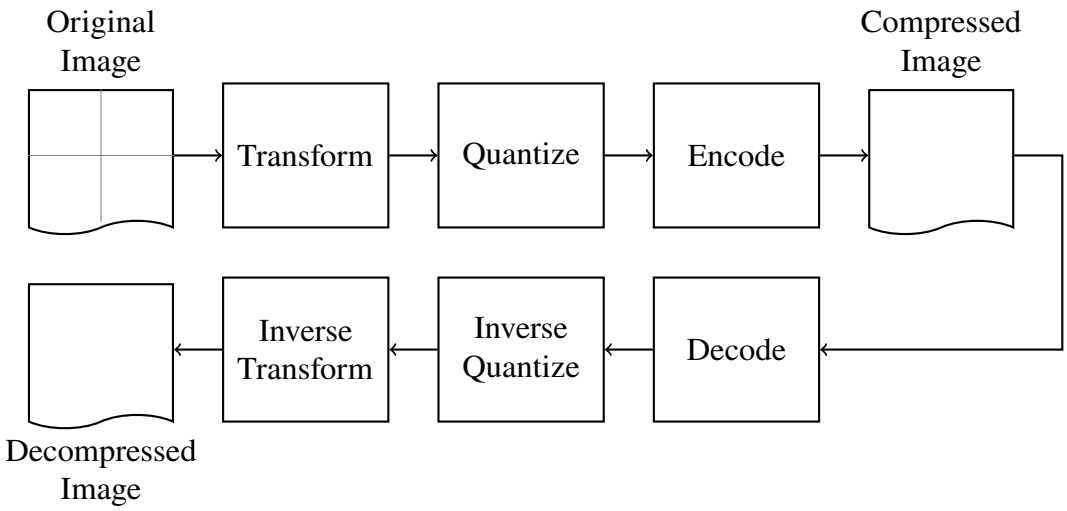


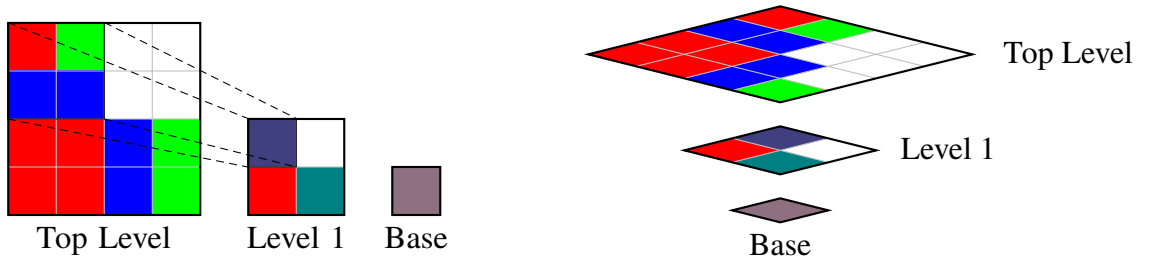
Figure 2.1: The variable bit rate compression and decompression pipeline.

level detail coefficients should be zero or close to zero. In both cases, the probability differential greatly increases encoding efficiency, as values at or near zero are encoded in very few bits. For encoding, JPEG uses Huffman coding while JPEG-2000 uses range coding.

Embedded Zerotree Wavelets and JPEG-2000, however, are unsuited to hardware decompression for several reasons. First, since the length of each compressed block varies, the decompression algorithm cannot easily compute the locations of compressed blocks. More importantly, the algorithms order the blocks in the compressed files in such a way to create many global dependencies between the blocks which hinders parallel decompression. Section 2.1.3 discusses the existing research in hardware-based VBR compression.

2.1.1 Texture Compression

While texture compression is simply image compression applied to textures, there is a unique requirement imposed on decompression. To improve sampling quality, textures are pre-filtered into a multiscale pyramid representation called mip-maps (Williams 1983). Figure 2.2(a) shows how this pre-filtering occurs and Figure 2.2(b) shows the pyramid representation. By pre-filtering textures, rendering algorithms avoid expensive filtering at runtime. This efficiency benefit, however, depends on the algorithms being able to use the



(a) Mip-map Creation

(b) Mip-map Pyramid

Figure 2.2: The creation and hierarchical representation of mip-mapped textures.

correct scale, or mip-level, of the multiscale representation. Texture compression, therefore, should enable direct access to the different mip-levels.

2.1.2 Fixed Bit Rate Hardware Formats

Beers, Agrawala, and Chaddha (1996) introduce an early texture compression format using vector quantization that explicitly supports mip-mapped textures. For non-mip-mapped textures, they create a codebook for the texture using 2×2 blocks of pixels. Each compressed block then stores the index of the codeword in an index map. For mip-mapped textures, they exploit the fact that each mip-level is a filtered version of the level above it and encode three levels together with a single codebook and index map. For each group of levels, three codebooks are created, the first from the largest resolution level using 4×4 blocks, the second and third by averaging the prior level codebook. The codewords are concatenated to form a single codebook and all three levels then use one index map.

Iorcha, Nayak, and Hong (1999) extend the Block Truncation Coding technique (Delp and Mitchell 1979) with several formats that have become the de facto standard in GPU texture compression known as S3 Texture Compression. These formats were included in the DirectX and OpenGL rendering APIs and are now called DXT or BC. The currently used variants include BC1 (DXT1), BC3 (DXT5) and BC7 (OpenGL ARB 2009). Van Waveren and Castaño (2007) improve the quality of BC3 compression while maintaining the same compression ratio by using the $\mathrm{Y'C_oC_g}$ color space instead of RGB.

All of the BC/DXT formats compress 4×4 blocks of pixels by quantizing the color at each pixel to a set of per-block representative colors. BC1 can encode fully opaque or fully transparent pixels while BC3 can encode a transparency gradient. For a 24-bit-per-pixel image, BC1 compresses a 384-bit block of pixels to 64 bits for a compression ratio of 6:1. For a 32-bit-per-pixel image, BC3 compresses a 512-bit block of pixels to 128 bits for a compression ratio of 4:1. When the colors in a block are smoothly varying, the BC1 and BC3 single color gradient is adequate, but when a block contains multiple sharp transitions of colors, compressed quality suffers. BC7 can encode multiple color gradients by subdividing the blocks. BC7 still encodes a block in 128 bits for a compression ratio of 4:1.

Fenney (2003) introduces a compression algorithm, PVR, with improved power efficiency for low-power hardware. PVR uses two low-resolution source images combined with a full-resolution modulation image and supports two compression ratios: 8:1 and 16:1. Ström and Akenine-Möller (2004) introduce the PACKMAN algorithm, also designed for low-power hardware. PACKMAN encodes a single chrominance value and a luminance modulation table to store a 2×4 block in 32 bits for a compression ratio of 8:1. Ström and Akenine-Möller (2005) extend PACKMAN with ETC by encoding two 2×4 blocks together. By encoding two blocks in 64 bits, the compression ratio remains 8:1 but the quality is improved. Ström and Pettersson (2007) then use invalid bit combinations in ETC2 for further modes that increase quality within the 8:1 compression ratio.

Recently, Khronos Group (2013) introduced ASTC, which is modeled after the existing block compression algorithms but supports varying the number of texels per block or *block footprint* in a texture to enable granular changes to the bit rate. The compressed block size remains fixed to still allow direct access. ASTC also supports varying the endpoint encoding method per block as well as partitioning the block into multiple endpoints.

2.1.3 Variable Bit Rate Hardware Formats

Inada and McCool (2006) present a VBR lossless format which uses an indexing scheme combined with a B-tree index for random-access and efficient memory usage. They also include a caching architecture for efficient hardware implementation. While the results are good for lossless compression of textures, no hardware has implemented this technique.

We introduce (Olano et al. 2011) a VBR format that is implemented as a GPU program and therefore does not require explicit hardware support. After a color space transform, the compression algorithm performs a mip-differencing transform, where adjacent mip levels are subtracted to form a set of mip-level differences. In most cases, these mip differences are very small or close to zero and are efficiently encoded. A benefit of this transform is that artists can tweak individual mip levels but good probability characteristics are maintained. After transforming, quantization is performed by dropping bits or entire mip levels. The quantized values are then encoded with a range coder. To enable parallel decompression on the GPU, an index map is encoded where each block of the map stores the compressed block location. On decompression, blocks use the index map to determine where to start decoding the compressed stream. We also introduce a fast GPU range decoder.

2.2 Image Quality Assessment

IQA metrics can be classified in two ways: by the type of input or type of model. The input is either luminance or color while the model is either *bottom-up* or *top-down*. Input classification is distinct, that is, metrics work on either luminance or color. Model classification, however, is more of a convenience and most IQA metrics are a mix of both models. Bottom-up metrics attempt to accurately model the human visual system while top-down metrics attempt to model the human visual system input-output characteristics. The human visual system is a complex system and only partially understood so bottom-up models must make simplifying assumptions to create a computationally feasible model. Top-down models, however, treat the human visual system as a black box and try to learn a mapping from input

image to output assessment (Wang and Bovik 2006).

Section 2.2.1 discusses current top-performing luminance metrics. The metrics have been extensively evaluated using several evaluation databases (Le Callet and Autrusseau 2005; Sheikh, Sabir, and Bovik 2006; Chandler and Jemami 2007; Ponomarenko et al. 2009; Larson and Chandler 2010; Horita, Shibata, and Kawayoke 2011). The databases contain color and luminance images evaluated by humans in controlled studies. Metrics are appraised based on how closely they match the human evaluation on the same pairs of images. On the color images, the metrics perform very well, reinforcing the fact that humans are very perceptive to luminance changes and not as perceptive to chrominance changes (Fairchild 2005).

Section 2.2.2 describes the current research in objective color IQA metrics. The color metrics all make use of the CIELAB color space and rigorous perceptual-based color differencing. The existing color metrics attempt to combine CIELAB color differencing with bottom-up models of the human visual system. The results, however, do not correlate well with the evaluation databases (Ajagamelle, Pedersen, and Simone 2010; Čadík et al. 2012) and color image quality assessment remains a difficult problem to solve.

In the following discussion, I use standard image processing practices and notation and treat two-dimensional images as one-dimensional signals denoted by $\mathbf{x}$ and $\mathbf{y}$.

2.2.1 Luminance Metrics

Chandler and Jemami (2007) introduce the Visual Signal-to-Noise Ratio (VSNR) a bottom-up algorithm in two stages. The first stage determines if the distorted image has visible distortions using visual contrast sensitivity. If the distortions are below a detectable threshold, the images are equivalent and the algorithm terminates. If the distortions are visible, the second stage computes the VSNR based on how the contrast distortions disrupt the image.

Wang et al. (2004) hypothesize that local luminance and local contrast changes do not strongly affect perceived image quality and thus define structure as the absence of luminance

and contrast in an image. The Structural Similarity Index Metric (SSIM) is a top-down metric that evaluates the differences in structure between a reference and distorted image. Their results show that structure is an excellent way to determine image quality for luminance natural images. They also introduce a multiscale variant, Multiscale Structural Similarity Index Metric (MS-SSIM), with improved performance over SSIM by evaluating structure differences at many scales (Wang, Simoncelli, and Bovik 2003).

Structural Similarity Index Metric (SSIM)

SSIM (Wang et al. 2004) is a top-down approach to image quality based on structure. By estimating luminance and contrast in a single image and then removing both, structure is then the information left in the image.

For a single image, luminance is estimated as mean intensity,

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2.1}$$

contrast is estimated as the standard deviation,

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2}, \tag{2.2}$$

and structure is then the normalized image signal given by

$$S_x = \frac{\mathbf{x} - \mu_x}{\sigma_x}. \tag{2.3}$$

For two images, **x** and **y**, SSIM is a weighted product of three comparison functions over local image patches given by

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma}, \tag{2.4}$$

where α, β, and γ are the weights for the comparison functions.

Luminance comparison ($l(\mathbf{x}, \mathbf{y})$) is the product of the mean intensities of each image,

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \tag{2.5}$$

contrast comparison ($c(\mathbf{x}, \mathbf{y})$) is the variance of the images,

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{2.6}$$

and structure comparison $s(\mathbf{x}, \mathbf{y})$ is the normalized cross-correlation (Rouse and Hemami 2008) of the images,

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \tag{2.7}$$

where σ_{xy} is the cross-correlation of $\mathbf{x}$ and $\mathbf{y}$ given by

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y). \tag{2.8}$$

In these equations, C_1, C_2, and C_3 are constants used for numerical stability.

Inserting the full comparison functions into Equation 2.4 gives the general SSIM equation between two images:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \left(\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\right)^{\alpha} \cdot \left(\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\right)^{\beta} \cdot \left(\frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\right)^{\gamma}. \tag{2.9}$$

Wang et al. (2004) set the comparison function weights $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$ to create the specific SSIM formula:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \tag{2.10}$$

This equation satisfies three conditions:

1. symmetry: $S(\mathbf{x}, \mathbf{y}) = S(\mathbf{y}, \mathbf{x})$,
2. boundedness: $S(\mathbf{x}, \mathbf{y}) \leq 1$, and
3. unique maximum: $S(\mathbf{x}, \mathbf{y}) = 1$ if and only if $\mathbf{x} = \mathbf{y}$.

To remove any large-scale luminance and contrast changes, the SSIM index is computed over small image patches with the result being an SSIM map of the structural differences between the two images. These differences are pooled across image patches using Minkowski pooling (Wang and Shang 2006) to produce a single value, MSSIM:

$$\mathrm{MSSIM}(\mathbf{X}, \mathbf{Y}) = \frac{1}{M}\sum_{j=1}^{M}\mathrm{SSIM}(\mathbf{x}_j, \mathbf{y}_j), \tag{2.11}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are the images, M is the number of image patches, and $\mathbf{x}_j$ and $\mathbf{y}_j$ are each image patch.

As Wang et al. (2004) demonstrate, this very simple formulation works extremely well in predicting image quality with one drawback: geometric transformations. SSIM does not handle rotation or translation distortions very well.

SSIM and MS-SSIM operate in two stages by first generating a map of local distortion values and then pooling those values into a single distortion value using Minkowski pooling. Wang and Li (2011) introduce a more rigorous pooling method for Information Content Weighted Structural Similarity Index Metric (IW-SSIM) that models local information content using Gaussian Scale Mixture models and then weights the pooling by the amount of information content.

Sheikh and Bovik (2006) introduce the Visual Information Fidelity (VIF) metric. Using natural image statistical modeling (Simoncelli 2005), they hypothesize that human visual processing and image distortion are communication channels and model information content for a local distortion map. By accurately modeling the image source, the image distortion, and human visual processing, mutual information between the reference and distorted images after human visual processing can indicate perceived quality.

2.2.2 Color Metrics

Colorimetry is the science concerned with assigning objective numbers to colors. A *color space* defines a coordinate space on which colors are described by the coordinates. One such color space is CIELAB.

Color Differencing

CIELAB is designed to be a perceptually uniform color space. This means that the Euclidean distance between two colors in the CIELAB coordinate space corresponds to the human perceived difference between those colors. CIELAB uses three components to describe

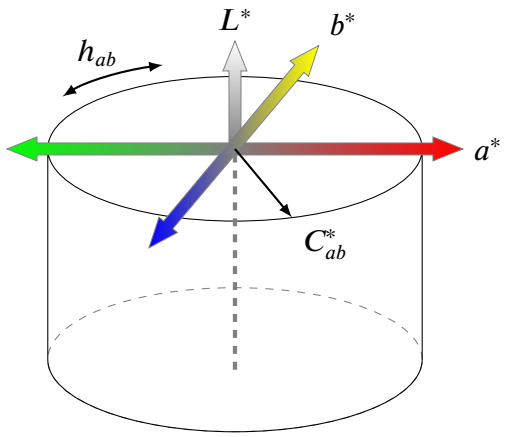


Figure 2.3: The $L^*a^*b^*$ and $L^*C^*_{ab}h_{ab}$ CIELAB coordinate systems.

a color: luminance, red-green chrominance, and yellow-blue chrominance. A color is then defined as a weighted combination of these components. Figure 2.3 shows how the components can be arranged in a Cartesian coordinate system ($L^*a^*b^*$) and or a cylindrical coordinate system ($L^*C^*_{ab}h_{ab}$), where L^* is luminance, or perceived lightness, a^* is red-green chrominance, b^* is blue-yellow chrominance, C^*_{ab} is a linear combination of a^* and b^*, and h_{ab} is hue angle. Luminance has the range $[0, 100]$, while both chrominance values, and hence C^*_{ab}, are unbounded. Hue angle is in degrees and has the range $[0, 360]$.

As CIELAB is perceptually uniform, Euclidean distances can be used for color difference comparison. The International Commission on Illumination has defined ΔE^*_{94} to compute color differences that reasonably match human perceived differences. More recent, but more complex, formulations exist but ΔE^*_{94} is acceptable for just color differences (Fairchild 2005). ΔE^*_{94} is defined at a single pixel and used to compute a color difference map at every pixel between two images.

Figure 2.4 shows an original and distorted image along with the computed (with values scaled for visualization) ΔE^*_{94} color difference map between the two images. Dark areas indicate little color difference (black pixels are exact color match) and increasing brightness indicates increasing color distance. A lowpass filter smooths an image obscuring the high-frequency detail. Thus, the color difference map has very bright areas corresponding to the

(a) Original (b) Distorted (c) ΔE^*_{94} Map

Figure 2.4: Original image, distorted image (32×32 Gaussian lowpass $\sigma = 10.0$), and computed (with values scaled for visualization) ΔE^*_{94} color difference map.

high frequency details (edges) of the reference image.

For two images, $\mathbf{x}$ and $\mathbf{y}$, ΔE^*_{94} is given by

$$\Delta E^*_{94} = \sqrt{(\Delta L^*)^2 + \left(\frac{\Delta C^*_{ab}}{1 + 0.045 \cdot C^*_{ab}(\mathbf{x})}\right)^2 + \left(\frac{\Delta H^*_{ab}}{1 + 0.015 \cdot C^*_{ab}(\mathbf{x})}\right)^2}, \tag{2.12}$$

where

$$\Delta L^* = L^*(\mathbf{x}) - L^*(\mathbf{y}), \tag{2.13}$$

$$\Delta C^*_{ab} = C^*_{ab}(\mathbf{x}) - C^*_{ab}(\mathbf{y}), \tag{2.14}$$

$$\Delta H^*_{ab} = \sqrt{\Delta a^{*2} + \Delta b^{*2} - \Delta C^{*\,2}_{ab}}, \tag{2.15}$$

$$\Delta a^* = a^*(\mathbf{x}) - a^*(\mathbf{y}), \text{ and} \tag{2.16}$$

$$\Delta b^* = b^*(\mathbf{x}) - b^*(\mathbf{y}). \tag{2.17}$$

With this formula a distance of zero indicates two perceptually-identical colors and increasing distances indicate increasing perceptual difference.

Zhang and Wandell (1998) modify the CIELAB color differencing formula to include a simple spatial contrast sensitivity function for the S-CIELAB color metric. Johnson and Fairchild (2001) then modify S-CIELAB with more accurate contrast sensitivity functions. The hue angle algorithm (Hong and Luo 2002) extends CIELAB by varying the weighting

for each pixel based on the hue angle of the color. Recently, Pedersen and Hardeberg (2009) add spatial filtering to the hue angle algorithm (Spatial Hue Angle Metric (SHAME) and SHAME-II). As Ajagamelle, Pedersen, and Simone (2010) show, these metrics do not have strong correlation to the human studies.

2.2.3 Metric Evaluation

Image quality assessment measures are used to evaluate the quality of compressed or rendered images, guide procedural shader simplification or texture compression, and in many applications in computer vision. An assessment algorithm can be used on any image, but the quality of the assessment algorithm is determined by comparing its results to databases of distorted images that have been evaluated by people. There are several existing image distortion databases (Horita, Shibata, and Kawayoke 2011; Larson and Chandler 2010; Le Callet and Autrusseau 2005; Ponomarenko et al. 2009; Sheikh et al. 2010) which are used in the signal processing literature to evaluate image processing algorithms. The databases consist of a set of reference images, a set of distorted images, and a set of human subjective evaluations. The distorted images are generated by applying an image distortion to each reference image. Each database has several different types of distortions, some of which overlap between databases. The subjective evaluations are a score given to each distorted image that indicates how it compares to the associated reference image. The scores are generated by asking humans to evaluate each reference and distorted image pair.

The existing databases contain either luminance-only distortions or chrominance distortions that are also coupled to luminance. This is problematic in cases where the databases are used to evaluate or possibly train image comparison algorithms. Those algorithms will not account for chromatic image distortions that are standalone or uncoupled from luminance distortions. This will be particularly problematic if those image comparison algorithms are used for image compression optimization since those metrics will be blind to chromatic distortions. Realizing this problem, Ponomarenko et al. (2015) present an expanded database

that includes several new types of distortions, specifically related to color, including color quantization and chromatic aberration.

2.3 Optimization and Searching

There is significant work on energy minimization and searching algorithms in the artificial intelligence and machine learning fields. I briefly summarize a specific type of search algorithm below and refer the reader to Russell and Norvig (2003) for further details.

Modern texture compression algorithms have several parameters that are used to fine-tune the algorithm. These parameters are a mix of discrete and continuous variables and are almost always finite. The set of possible values for each parameter form a multi-dimensional state space, where each state is a specific set of values, one for each parameter.

Following Russell and Norvig (2003), I define a problem where the **initial state** is a randomly chosen set of values, one for each parameter, the only possible **action** is to compress the textures with the given set of values, the **path cost** is a function of bitrate and image quality, and the **goal test** is a state with minimal bitrate and maximal quality. Given this problem description, there are many possible ways to solve it.

The largest factor in choosing a method to solve the problem is the size of the state space. The ASTC algorithm (Khronos Group 2013) has seven parameters, five being discrete, two being continuous, and all of fixed range. They combine to form a space of more than 10^{12} possible states. Fully enumerating this state space is impractical and so we must turn to local search algorithms. Gradient descent optimization is also problematic, since it would be difficult to describe the compressor using a differential equation.

Hill-climbing algorithms are a class of local search algorithms that are greedy in that they only look at immediate neighbors and pick the neighbor with the best improvement. Stochastic hill-climbing randomly picks from the better neighbors while first-choice hill-climbing randomly generates neighbors to evaluate until a better neighbor is found.

2.4 Rendering Optimization

There has been little research in optimizing parts of the rendering process. The most directly relevant work is the research in automatic shader simplification. In their seminal work, Olano, Kuehne, and Simmons (2003) automatically create multiple shader levels of detail by reducing the number of texture accesses in shader programs. Pellacini (2005) uses a set of simplification rules on abstract syntax trees generated from shaders. The search for a more simple syntax tree was guided by the $\mathbf{L}^2$ distance between the original shader and simplified shader. Sitthi-Amorn et al. (2011) employ genetic programming to more thoroughly search the state space of simplified shaders and also use $\mathbf{L}^2$ distance for evaluation. More recent work by He et al. (2015) attempts to simplify across shader stages and not just within a single shader. Additionally, there has been some work in automating texture compression in the asset build pipeline. Mitchell (2006, 2015) allowed producers to force textures to a specific size for final game builds based on observed usage.

Chapter 3

CHROMINANCE DISTORTION DATABASE

To aid research and evaluation of color IQA metrics, I have developed a chrominance distortion image database. This database consists of 24 reference images and 500 distorted images spread across five different chromatic distortions. The chromatic distortions were chosen to model common techniques in image compression as well as other chromatic effects, such as chromatic aberration and chromatic noise. I performed a user study to generate subjective evaluations of each distorted image as an average DMOS (see Section 3.1). The database is available at http://bitbucket.org/wgriffin/cdid. I provide the reference images, distorted images, mean DMOS scores and the source code used to create the distorted images.

A set of 24 reference images were taken from the Kodak Lossless True Color Image Suite (Franzen 2010) and then cropped to 512x512. The images are all color RGB images. Five different types of chrominance-only distortions were applied to the set of reference images. The distortions were chosen to model common techniques in image compression as well as other chromatic effects, such as chromatic aberration and chromatic noise. The types are listed below and are referred to by the acronyms in the rest of the document.

- **QAB**: Quantizing the A and B channels of the CIELAB image.
- **CD**: Geometric rescaling the G and B channels of the RGB image.
- **DSAB**: Down-sampling of the A and B channels of the CIELAB image.
- **HR**: Random rotation of the H channel of the HSV image.
- **GBAB**: Gaussian blurring of the A and B channels of the CIELAB image.

(a) Reference (b) Distorted

Figure 3.1: The **QAB** chrominance distortion uses quantization to down-sample the chroma components. The distortion results in blocks of a single chroma value replacing smoothly varying areas of color.

For each type of distortion, a parameter was varied to create multiple distorted images. Additionally, that parameter was fixed for 10% of the distorted images for each distortion type such that the "distorted" image is just the reference image, i.e., with no distortion applied, and I call those *check images*. I have a total of 500 images across the five distortion types.

The **QAB** distortion models the down-sampling technique of **DSAB** but using quantization instead of down-sampling blocks. In Figure 3.1, the quantization is seen as blocks of a single chroma value replacing smoothly varying areas of color. I generated 100 distorted images by quantizing the A and B channels of the CIELAB image. A *nbins* value is selected from the set $[4, 6, 8, 10]$. The quantization uses a number of bins equal to the *nbins* value. There are ten *check images*.

The **CD** distortion simulates chromatic aberration which is a distortion effect introduced by optical lenses. In Figure 3.2, the chromatic aberration is seen as a general blurring of edges along with green and magenta fringing along edges. I generated 50 distorted images

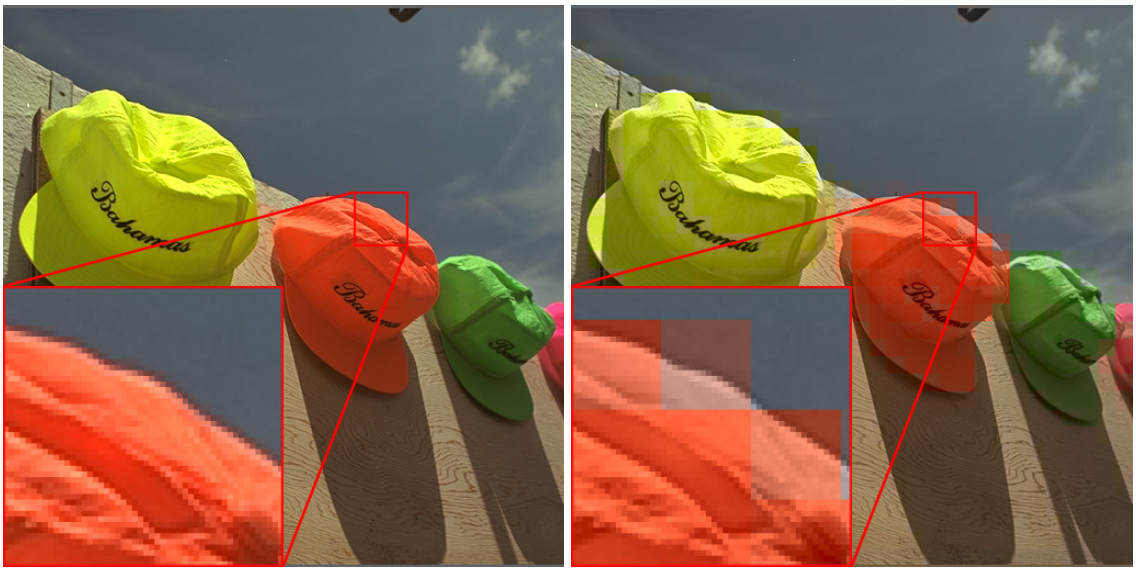
(a) Reference (b) Distorted

Figure 3.2: The **CD** chrominance distortion simulating chromatic aberration. The distortion results in general blurring of edges and green and magenta fringing along edges.

by geometrically scaling and resampling the pixel positions of the G and B channels of the RGB image. For each distorted image, a *scale factor* was randomly selected from the set $[2, 3, 4]$. The G channel was shifted by the *scale factor* from the image center, and the B channel was shifted by twice the *scale factor*. There are five *check images*.

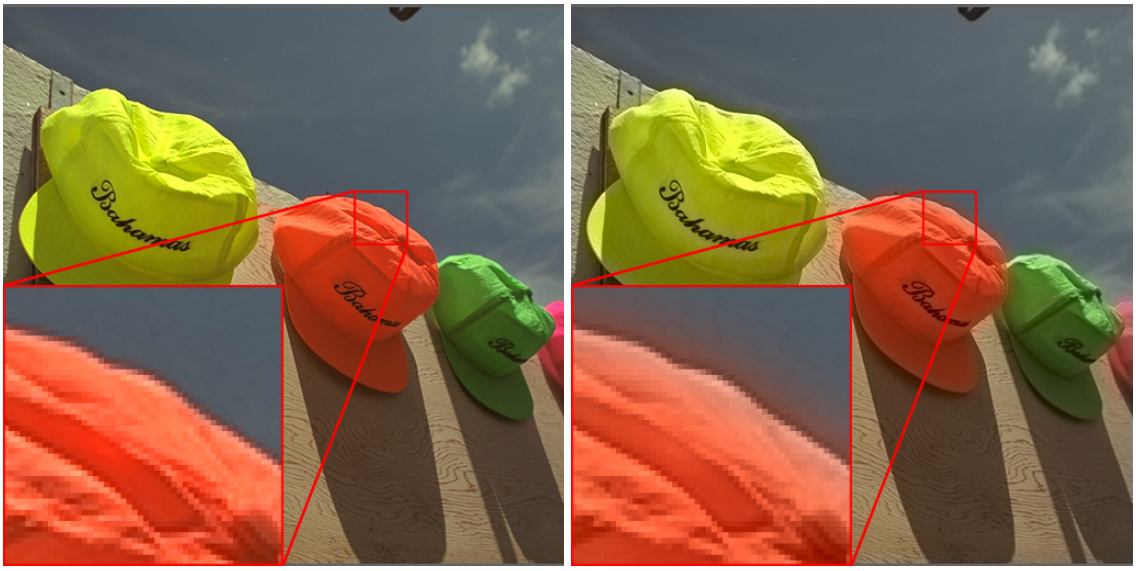
The **DSAB** distortion models the common approach in image compression of down-sampling the chroma components before compressing, which exploits the fact that the human visual system is more perceptive to errors in luminance than errors in chrominance (Hao and Shi 2000). In Figure 3.3, the down-sampling is seen as blocks of chroma. I generated 100 distorted images by down-sampling the A and B channels of the CIELAB image. For each distorted image, a *blocksize* was randomly selected from the set $[2, 4, 8, 16]$ and corresponds to the size in pixels of each block. The A and B channels were both down-sampled by the *blocksize*. There are ten *check images*.

The **GBAB** distortion roughly approximates down-sampling of the chroma channels modeled by **DSAB** by using a Gaussian blur instead of down-sampling blocks. In Figure 3.4, the blurring is seen as smooth color bleeding across edges. I generated 125 distorted images



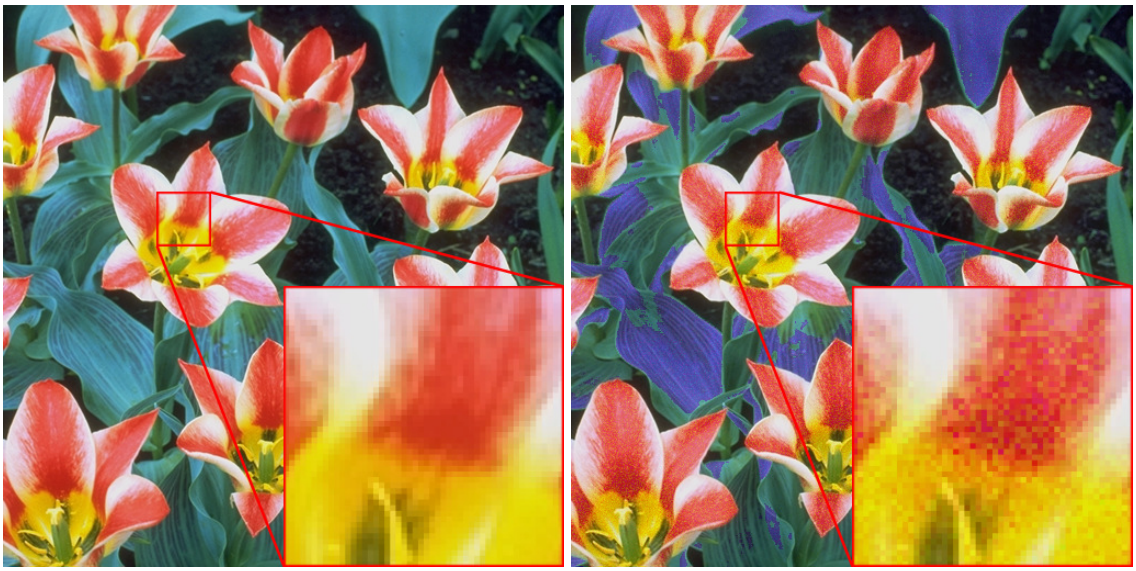
(a) Reference (b) Distorted

Figure 3.3: The **DSAB** chrominance distortion modeling chroma component down-sampling. The distortion results in blocks of chroma based on the down-sampled block size.



(a) Reference (b) Distorted

Figure 3.4: The **GBAB** chrominance distortion roughly approximating chroma component down-sampling. The distortion results in smooth color bleeding across edges.



(a) Reference (b) Distorted

Figure 3.5: The **HR** chrominance distortion simulating random color noise. The distortion results in large blocks of color and color speckles.

by applying a Gaussian blur to the A and B channels of the CIELAB image. A *sigma* value is selected from a normal distribution ($\mu = 7.5$, $\sigma = 2.5$). The Gaussian blur kernel has $\mu = 0$ and $\sigma = sigma$. There are 13 *check images*.

The **HR** distortion simulates random color noise. In Figure 3.5, the noise can be seen as both large blocks of color and color speckles. I generated 125 distorted images by randomly rotating the angle of the H channel of the HSV image. A *kappa* value is selected from a normal distribution ($\mu = 100$, $\sigma = 64$). The amount of angular rotation applied is selected from the von Mises distribution with the given *kappa*. There are 13 *check images*.

3.1 Evaluation

To generate the subjective evaluations, I used Amazon Mechanical Turk (AMT). AMT is a large-scale crowd-sourced task completion system. Requesters create Human Intelligence Tasks (HITs) which are small tasks that typically cannot be automated. Users, also called "Turkers", complete tasks for micro-payments. Amazon manages the infrastructure for

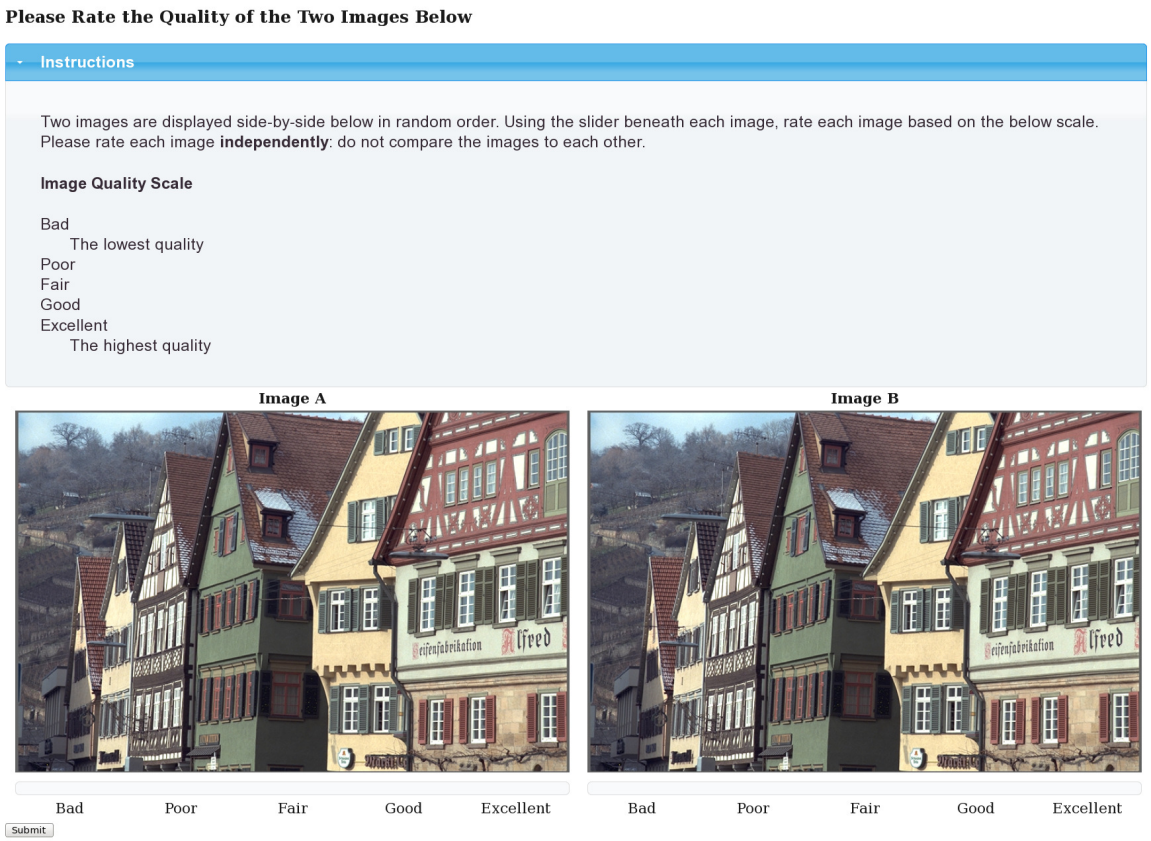


Figure 3.6: Sample HIT for evaluation on Mechanical Turk.

Requesters to submit HITs and Turkers to complete HITs and receive payment. Two key reasons AMT is useful for conducting research studies are 1) the large number of Turkers available and 2) AMT maintains anonymity of the Turkers to the Requesters. The user study was conducted under UMBC IRB Protocol Y13MO37200.

That Turkers remain anonymous, however, is also a drawback. As Kittur, Chi, and Suh (2008) and Downs et al. (2010) point out, there is an incentive to maximize profit while minimizing effort. This incentive manifests as Turkers who attempt to complete HITs as quickly as possible without regard for correctness of the work. These Turkers attempting to game the system show up as noise and outliers in the results of research studies and reduce the usefulness of AMT.

I evaluated the distorted images using the Double-Stimulus Continuous Quality Scale (DSCQS) (ITU-R 2012; Video Quality Experts Group 2003) method on AMT. The DSCQS

method presents the distorted image and reference image side-by-side and asks the participant to evaluate both images independently using a sliding scale from 0–100 with five equally spaced labels: Bad, Poor, Fair, Good, and Excellent. Figure 3.6 shows a sample HIT. The slider is "continuous" and the labels are only for reference. The participant is not told which image is the reference image and the order of the images is randomized within a single test, i.e. the reference image is randomly chosen to be either on the left or right. The slider control was initialized to a value of -1, which is impossible to select on the slider and used to verify responses.

To remain within acceptable time limits, the 500 images were randomly divided into batches of 50 images, consisting of ten images from each distortion type. Each DSCQS image evaluation task was a single HIT and thus each batch had 50 HITs (1 HIT per image in each batch). Both **HR** and **GBAB** have 125 distorted images. A total of 13 batches (where each batch has 50 images) were necessary to ensure all 125 images were evaluated. With 13 batches, there was some overlap in the set of images within a batch. I used this overlap to verify responses.

I used three methods for verifying responses: 1) any response where one rated image had a value of -1 was excluded, 2) any participants with more than 6% of their responses having a value of -1 was excluded, and 3) any participant who rated the reference image of repeated image pairs within a batch differently by more than 20%. For the first measure, 16 individual responses were excluded where the either rated image had a score of -1. For the second measure, one participant was excluded where that participant had rated more than three images with a score of -1. For the final measure, six participants were excluded. In all, I had a total of 50 participants generate 10,484 subjective evaluations across the 500 distorted images.

The DSCQS method provides a DMOS for each participant on each HIT by subtracting the distorted image rating from the reference image rating. The Video Quality Experts Group (VQEG) recommends normalizing the DMOS values per subject to reduce variability

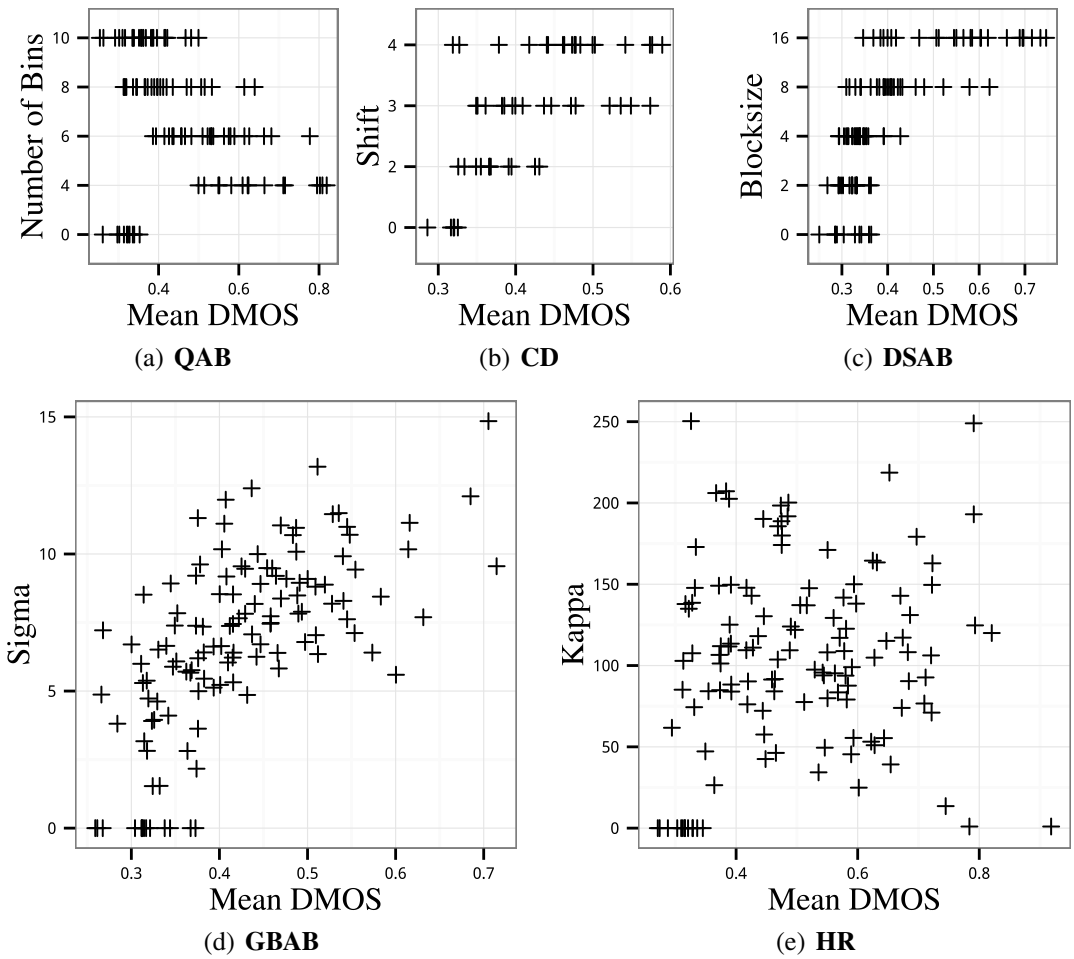
within a single set of ratings by a subject. Following this procedure, a normalized DMOS value of zero indicates the subject rated the reference and distorted images identical, while increasingly positive DMOS values indicate the subject rated the reference image higher in quality than the distorted image and increasingly negative DMOS values indicate the subject rated the distorted image higher in quality than the reference image.

3.2 Analysis

As mentioned, lower DMOS scores indicate the subjects rated the reference image with better quality. As such, when the expected distorted image quality is lower (i.e. the distortion is more apparent and degrading) I expect the DMOS scores to also decrease, showing that the subjects find the reference images have better quality. Figure 3.7 plots the mean DMOS scores broken out by each distortion type. These plots clearly show the trend that as the expected distorted image quality decreased, the mean DMOS score also decreased.

For the **QAB** distortion, Figure 3.7(a) plots the mean DMOS for each distorted image with the y axis being the number of bins used for quantization. A lower number of bins should result in worse image quality while a higher number of bins should results in better quality. The zero value indicates no distortion was applied (the distorted and reference images were identical). As expected, when no quantization is performed, the mean DMOS is close to zero with little variance. This shows that the subjects rated the distorted and reference images close to equal in quality. Next, starting with four quantization bins (which I expect to have the worst quality) and increasing the sixteen bins (which I expect to have the best quality), the mean DMOS scores start high and decrease towards zero. This confirms my expectations that the subjects found using few quantization bins produced images with bad quality while increasing the number of bins progressively improved the distorted image quality.

For the rest of the cases, **CD**, **DSAB**, **GBAB**, and **HR**, the y-axis is reversed: decreasing distortion parameters should results in worse image quality. The zero value still has the same



(a) **QAB** (b) **CD** (c) **DSAB**

(d) **GBAB** (e) **HR**

Figure 3.7: Mean DMOS by Distortion Parameter. See Section 3.2 for detailed discussion.

meaning – no distortion was applied.

For the **CD** distortion, Figure 3.7(b) plots the mean DMOS for each distorted image with the y axis being the amount of shift applied to each image. As mentioned, a lower shift value should result in better image quality while a higher shift value should results in worse quality. Again, the zero value indicates identical reference and distorted images. As expected, when the shift value is zero, the mean DMOS is close to zero and the variance is small showing the subjects rated the reference and distorted images close to equal in quality. As the shift values increase, which should result in worse distorted image quality, the mean DMOS increases indicating the subjects did indeed rate the distorted image with worse quality.

For the **DSAB** distortion, Figure 3.7(c) plots the mean DMOS with the y axis being the down-sampling block size. A lower blocksize results in smaller sized blocks (i.e. there are fewer pixels per block and hence more total blocks in the distorted image) and should result in better image quality. For this case as well, when the blocksize is zero the mean DMOS is again close to zero, and as the blocksize increases the mean DMOS increases, showing the subjects consider the higher blocksizes to have worse image quality.

For the **GBAB** distortion, Figure 3.7(d) plots the mean DMOS with the y axis being the sigma parameter of the Gaussian kernel (with $\mu = 0$). As before, a lower sigma should result in better image quality with increasing sigma values worsening the image quality. While these results aren't as clear in this case, there is still a general trend of higher distortion having higher DMOS scores which corresponds with my expectations that the subjects find increasing blurriness reduces the quality of the distorted image.

For the **HR** distortion, Figure 3.7(e) plots the mean DMOS against the kappa parameter. A lower kappa value results in less angular rotation in the hue angle and thus better image quality. For these results, however, there is no clear trend in the mean DMOS scores. A likely explanation is that the human visual system perceives similar distortion amounts at different hues differently. Another possible explanation is that the speckles (Figure 3.5) do not introduce enough perceptual distortion to affect any one image.

Chapter 4

TEXTURE COMPRESSION EVALUATION

In evaluating texture compression, the most common approach is to use some combination of the MSE, PSNR, or visual image inspection on the individual textures. While comparing or inspecting individual textures is straightforward to implement, this method does not properly account for the two ways in which textures are used.

Textures are not single images but are instead used in two specific ways. First, they are mapped onto geometric objects and then those texture-mapped objects are rendered from some viewpoint. Second, multiple textures are frequently used together, such as a diffuse and bump map, or a diffuse, gloss, and bump map and these sets of textures are used to evaluate the rendering equation.

In both cases, textures are used to render a final image from a specific viewpoint. In a rendered image, there are likely masking effects from geometric distortion or overlapping textures that affect that quality of the image (Ferwerda et al. 1997). I define *geometric* masking to be the effects from texture mapping, that is wrapping a texture onto a model. Geometric masking effects include parts of a texture not visible from a viewpoint (e.g. back of a model) or undersampling the texture due to model pose (e.g. stretching or creases). I define *texture set* masking to be the effects from multiple textures. Texture set masking effects include diffuse texture artifacts masked by high frequency bump maps. Both of these types of masking effects cannot be accounted for when evaluating individual textures, they only appear when rendering a final image from a viewpoint.

To account for these masking effects, I introduce a texture compression evaluation methodology using final rendered images. However, rendering a scene from a single viewpoint is effectively the same as evaluating individual textures, since the masking effects

will vary between viewpoints. Since the set of possible viewpoints in a scene is infinite but discrete, my method samples this viewpoint space and evaluates the final rendered images at each sampled viewpoint.

For every sampled viewpoint, I render the scene using variations of compressed textures along with a ground truth image using uncompressed textures. Each compressed variation is compared to the uncompressed ground truth by computing two perceptually rigorous objective image quality assessment metrics at each viewpoint.

The following results show:

- Geometric and texture set masking occurs and cannot be detected by evaluating individual textures.
- The relative contribution between geometric and texture set masking varies by type of texture.
- Perceptual sensitivity to masking varies with the type of texture, such as diffuse, gloss or bump maps.
- Current GPU compression algorithms are too conservative and bit rates can be reduced while maintaining final rendered image quality.

4.1 Approach

In order to perform the evaluations, I need to render the scene from a set of viewpoints. I choose to sample the infinite, but discrete, space of viewpoints and then render a final image at each viewpoint using variations of compressed textures. After rendering, I then compare each final rendered image to a render at the same viewpoint with uncompressed textures.

4.1.1 Viewpoint Sampling

My evaluation method cannot use just a single viewpoint, as that would not account for the possible differences in masking effects across viewpoints. Since the viewpoint space

is infinite, I must sample it to make my approach computationally tractable. I sample the viewpoint space uniformly. Sampling techniques such as importance sampling based on expected viewer locations could improve the evaluation, but I leave these for future work. Viewpoint sampling has been used in other areas of computer graphics, such as image-based mesh simplification (Lindstrom and Turk 2000), but not, to my knowledge, for texture compression.

For the single-object models, Fire Hydrant, Taurus, and Urban Guy, I restrict the set of viewpoints to a bounding sphere. I use the quasi-random Sobol sequence generator (Joe and Kuo 2003) to sample this sphere of viewpoints. To ensure quality viewpoint samples, I constrain the sphere to a multiple of the object's bounding sphere. I also reject any viewpoint samples closer than a multiple of the bounding sphere radius, to prevent the viewpoint samples from being inside the model. The multipliers were chosen such that for each model, the final rendered images ranged from having 0.8 % to 91 % pixels filled. Figure 4.2 shows the closest, median, and furthest viewpoint.

The set of possible viewpoints is frequently restricted to some subset of space in the scene. For example, in many games, the viewpoints are restricted to a character's point of view or to a path of camera movements that follow a character. With an instrumented rendering engine, a trace of viewpoints could be captured during development and testing of a game. These traces would become the space of viewpoints to sample. To demonstrate this, a set of key viewpoints for the Japanese Castle model were captured while navigating around the model. The key viewpoints were interpolated to form a set of evaluation viewpoints.

4.1.2 Rendered Image Comparisons

At each viewpoint, I render a ground truth image using uncompressed textures. I then render the same viewpoint using variations of compressed textures. After all the images are rendered at a viewpoint, I compare each compressed variation final rendered image to the uncompressed ground truth image. I extend the method introduced by Beers, Agrawala, and

Chaddha (1996) with perceptually rigorous objective image quality assessment metrics.

I evaluate three metrics: the Root Mean Square (RMS) Color Error as a baseline reference, the CIELAB ΔE^*_{94} color differencing metric to evaluate chroma error, and the Structural Similarity Index Metric (SSIM) to evaluate structural error. Since SSIM captures error in the structure of an image, it is well suited to capturing compression artifacts in the bump and gloss maps.

RMS Color Error (Equation 4.1) has no perceptual foundation, but is widely used and I include it for reference. Note that we are still measuring the influence of geometric and texture set masking on final rendered image quality and not individual texture error.

$$\mathrm{RMS}_{\mathrm{Color}} = \sqrt{\frac{1}{N}\sum_{1}^{N}(\Delta r^2 + \Delta g^2 + \Delta b^2)} \tag{4.1}$$

Each metric is averaged across the set of viewpoints to compute a mean metric score for each compression algorithm. To further isolate just the compression artifacts, I modified each metric to compute the error over just the pixels that were rendered. This was done by using an RGBA framebuffer for the rendering, and then weighting the metric at each pixel by the alpha value.

SSIM is formulated for luminance-only natural images. While some have applied SSIM to the color channels of an RGB image, SSIM is only a valid perceptual metric when applied to luminance. For my evaluation, I transform the RGB images into the CIELAB color space and compute SSIM on the L^* component. I also compute the CIELAB ΔE^*_{94} color difference metric on the entire CIELAB image as my full-color perceptual metric.

Finally, for some results, I report the RMS Angular Error (Equation 4.2) computed on individual bump textures. This metric is used to evaluate the individual bump textures to compare against our evaluation method.

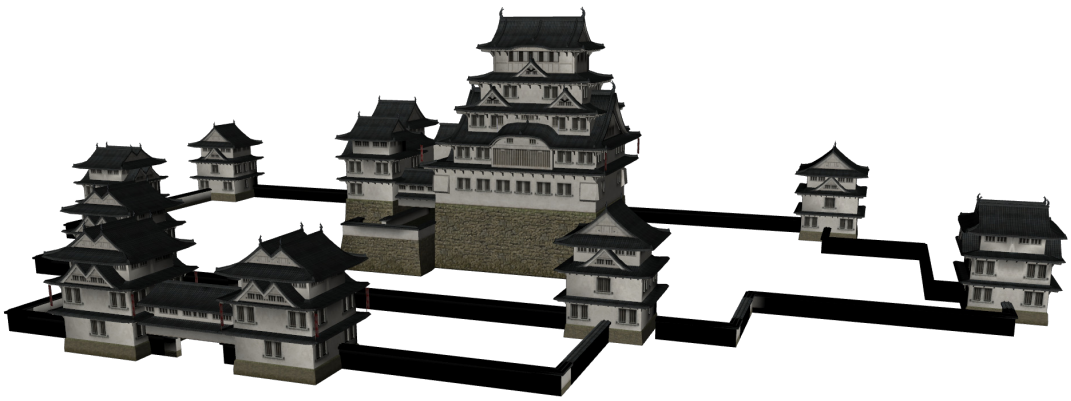
$$\mathrm{RMS}_{\mathrm{Angular}} = \sqrt{\frac{1}{N}\sum_{1}^{N} arccos(n_0 \cdot n_1)^2} \tag{4.2}$$

where n_0 and n_1 are the reconstructed unit normals.

(a) Fire Hydrant

(b) Taurus

(c) Urban Guy



(d) Japanese Castle

Figure 4.1: The models used for evaluation.

4.2 Results

The following results are from applying my evaluation methodology to a set of models and compression algorithms. The results show evidence of geometric and texture set masking. These masking effects cannot be accounted for when evaluating individual textures and are only present when rendering from a viewpoint. Additionally, the masking effects vary based on the viewpoint and so evaluating a single viewpoint is not sufficient. My approach of sampling the viewpoint space and comparing final rendered images accounts for both the masking effects and the viewpoint variation. By using final rendered images, the evaluation can use perceptually rigorous objective image quality assessment metrics which match how

(a) Closest (b) Median (c) Furthest

Figure 4.2: Coverage range of sampled viewpoints.

humans perceive image distortions. For space, I only present a small set of the data here. Plots and tables of all of the data are provided in Griffin and Olano (2014 and 2015).

4.2.1 Models

I use four models shown in Figure 4.1: Fire Hydrant, Taurus, Urban Guy, and Japanese Castle. Each model has at least one texture atlas set which consists of a bump map, gloss map, and diffuse map, all using a single parameterization. The Fire Hydrant model (Figure 4.1(a)) uses one texture atlas set at a resolution of 2048×2048. The Taurus model (Figure 4.1(b)) uses one texture atlas set at a resolution of 4096×4096. The Urban Guy model (Figure 4.1(c)) uses four texture atlas sets, one for the skin and clothes at 2048×2048, one for the jacket at 1024×1024, and two for the accessories (sunglasses and watch) at 512×512. The Japanese Castle model (Figure 4.1(d)) has 21 texture atlas sets for the various buildings and props, one set at 2048×2048, four at 1024×1024 and the rest at lower resolutions (mostly 512×512).

I use the Cook-Torrance shading model with true Fresnel assuming unpolarized light. The mipmaps for each texture were generated with a Kaiser filter. The bump and gloss maps were transformed into the second moments of variance for proper linear filtering (Olano and Baker 2010), then converted back into gloss for texture storage. The bump maps are stored in the dual-paraboloid encoding (Heidrich and Seidel 1998) and the gloss maps encode the

Cook-Torrance variance, which is reconstructed in the pixel shader using: $\sigma^2 = 2/(2^{10r+1}+2)$, where r is the gloss value.

4.2.2 Compression Algorithms

I use three formats for Direct3D Block Compression: BC5 (8 bits/pixel compression rate) for the bump maps, BC4 (4 bits/pixel compression rate) for the gloss maps and BC3 (8 bits/pixel compression rate) for the diffuse maps. The NVIDIA Texture Tools Library (NVIDIA 2013) was used for compression, with the quality set to 'normal'.

I use three different block sizes for Adaptive Scalable Texture Compression: 4×4 (8 bits/pixel compression rate), 8×8 (4 bits/pixel compression rate), and 12×12 (0.89 bits/pixel compression rate) with RGBA textures for all three maps. The bump maps are compressed in the RG channels, while the gloss maps remain RGBA. The ARM Mali ASTC Evaluation Codec (ARM 2013) was used for compression, with the quality set to 'thorough'.

To explore extreme compression rates, I "compressed" the textures by dropping from one to seven bits from each color plane. In addition to compressing all of the textures, I also chose to compress just a single class of texture, such as diffuse or gloss textures, leaving the other textures uncompressed to determine which class had the most effect on the final image quality.

4.2.3 Discussion

I first explain my notation and then discuss my results and make several conclusions. In the plots and tables, *All* refers to compressing all of the textures, while *Diffuse*, *Gloss*, and *Bump* refer to compressing just that respective texture. *NVTT/BC* refers to Direct3D Block Compression using the NVIDIA Texture Tools Library and *ASTC* refers to Adaptive Scalable Texture Compression using the ARM Mali ASTC Evaluation Codec. Each *ASTC* label includes the block size. Finally, *Bits Dropped* refers to the textures with 1–7 bits dropped from each color plane.

For the RMS Color Error, CIELAB ΔE^*_{94} Difference, and RMS Angular Error metrics, lower is better and 0.0 is a perfect match. For the SSIM metric, higher is better and 1.0 is a perfect match. All of the mean results are over the entire set of 2500 sampled viewpoints. In all histogram plots, the y axis in each plot is the number of viewpoints with a specific metric score and the x axis is the metric score, with values to the right indicating better quality.

In the following sections I present three test cases:

1. **Wrapped Texture Set**: is the full set of textures for each model wrapped onto the model and rendered with lighting. This is the standard texture mapping use case.

2. **Wrapped Single Texture**: is just a single texture class (diffuse, gloss, or normal) wrapped onto the model and rendered without lighting. This case attempts to measure the contribution of texture set masking.

3. **Unwrapped Texture Set**: is the full set of textures for each model rendered with lighting, but unwrapped in texture space. This case attempts to measure the contribution of geometric masking.

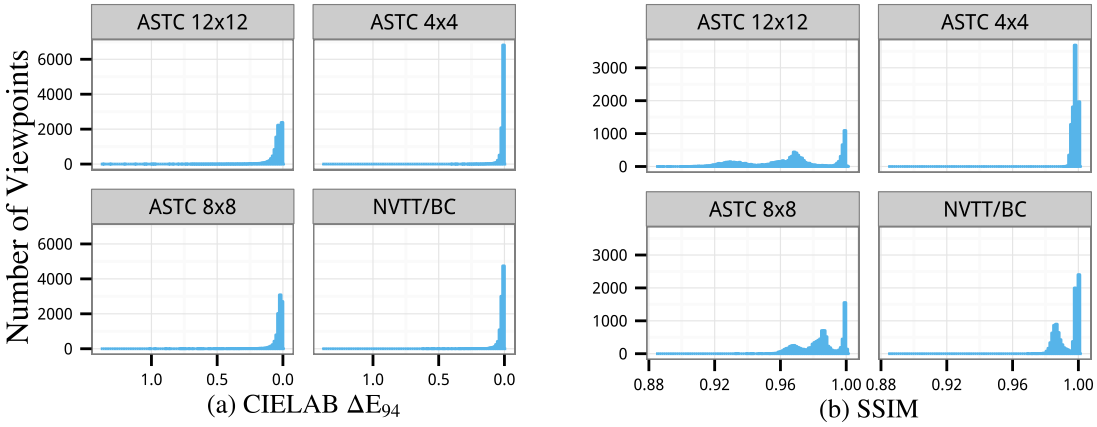


Figure 4.3: Fire Hydrant **Wrapped Texture Set** metric histograms. Values to the right are better. The variance by viewpoint implies geometric and texture set masking occurs.

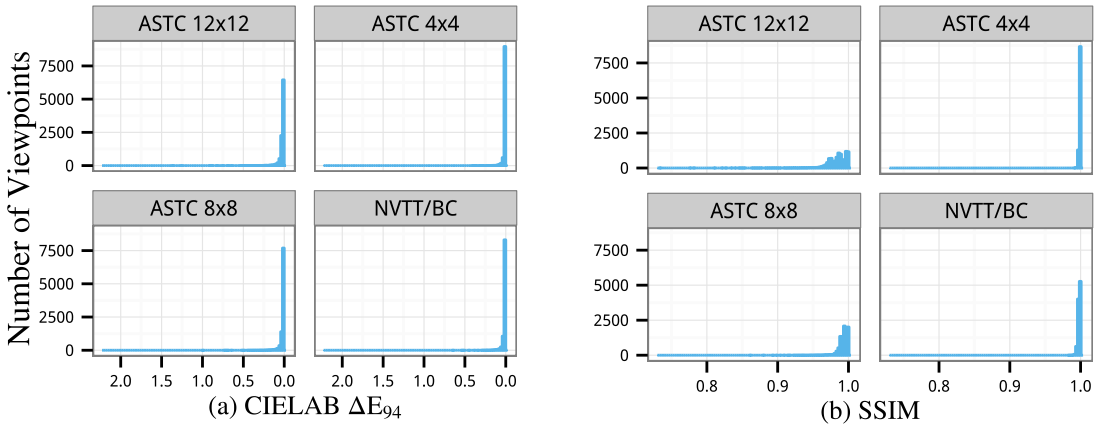


Figure 4.4: Urban Guy **Wrapped Texture Set** metric histograms. Values to the right are better. The variance by viewpoint implies geometric and texture set masking occurs.

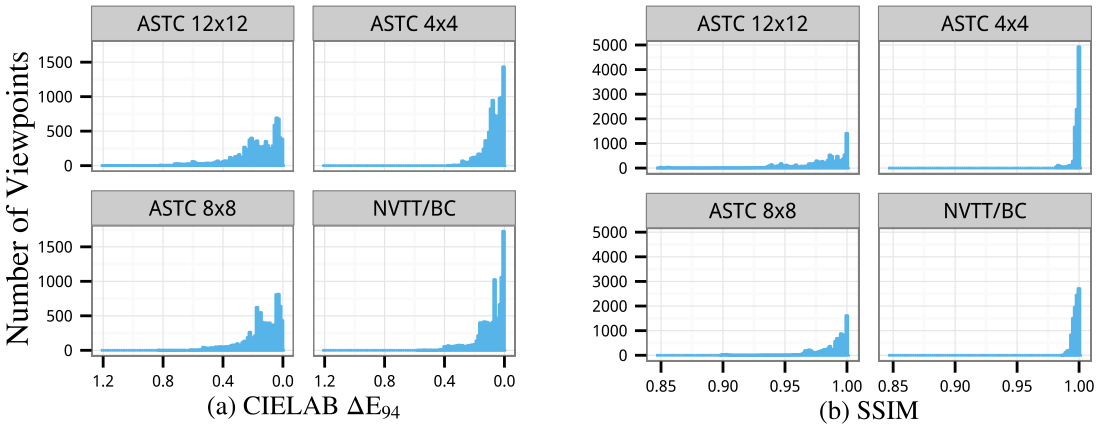
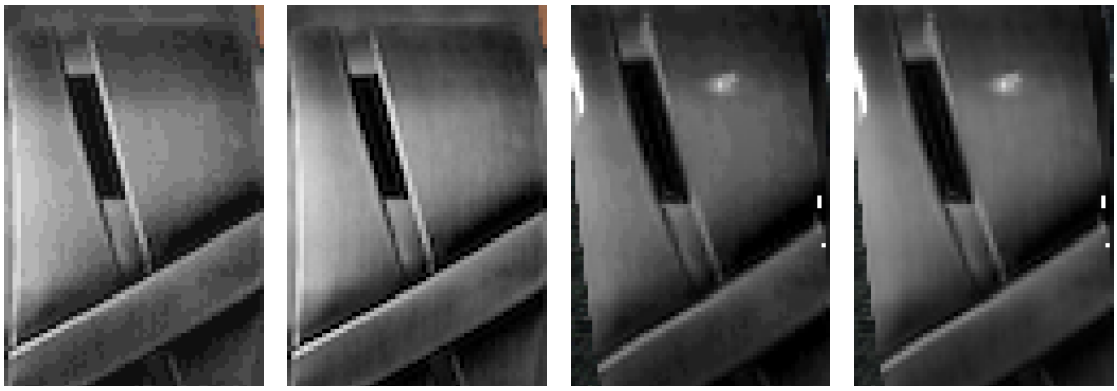


Figure 4.5: Japanese Castle **Wrapped Texture Set** metric histograms. Values to the right are better. The variance by viewpoint implies geometric and texture set masking occurs.

Comparing final rendered images is appropriate.

Figures 4.3, 4.4, and 4.5 shows the histograms of the **Wrapped Texture Set** test case for the CIELAB ΔE^*_{94} and SSIM metrics for each model. As the histograms illustrate, the error metrics do vary with the viewpoint. I can conclude that these differences are due to geometric and texture set masking and can only be accounted for by evaluating final rendered images. The variation also indicates that a single viewpoint cannot be used for the evaluation and that the viewpoint space must be sampled. However, the relatively small variance indicated in the histograms implies that a summary statistic can be used to represent the error metric.



(a) (b) (c) (d)

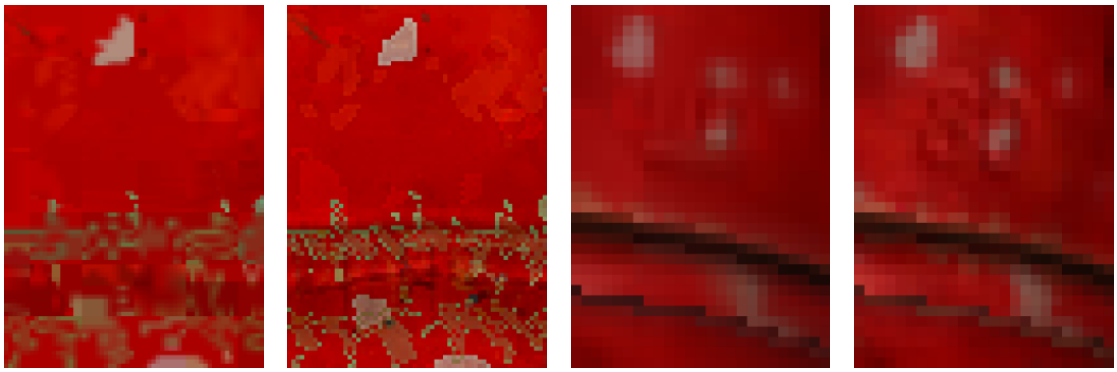
Figure 4.6: Visual example of color banding artifacts being masked by a high frequency bump map on the Urban Guy model. The compressed texture (a) shows color banding compared to uncompressed (b). When rendering with the compressed texture (c), the color banding artifacts are masked compared to rendering with the uncompressed texture (d).

I choose to use the mean error value over all of the viewpoints. Other possibilities include some percentile of the distribution, for example, the median.

Geometric and texture set masking effects exist.

Figures 4.6 and 4.7 are visual examples of masking in the diffuse texture for the Urban Guy model and Fire Hydrant model respectively. In Figure 4.6, notice the color banding caused by dropping four bits in Figure 4.6(a) is masked in Figure 4.6(c) by the high frequency bump map. In Figure 4.7, notice the severe block artifacts caused by compression in Figure 4.7(a) are masked in Figure 4.7(c) by the high frequency bump map.

Table 4.1 lists mean color (ΔE^*_{94}) and structure (**SSIM**) error for normal only compression next to mean RMS Angular Error (**RMSa**) over the MIP levels of the largest sized texture set for the Urban Guy model. By evaluating just the individual texture RMS Angular error, Table 4.1 indicates that ASTC 12×12 at 2 bits/pixel has better quality than NVTT/BC at 8 bits/pixel. This result is clearly non-intuitive since it does not take into account masking effects. By evaluating error over the sampled viewpoints, the ΔE^*_{94} and SSIM metrics indicate that NVTT/BC results in higher rendered image quality than ASTC 12×12.



(a) (b) (c) (d)

Figure 4.7: Visual example of compression block artifacts being masked by a high frequency bump map on the Fire Hydrant model. The compressed texture (a) shows severe block artifacts compared to uncompressed (b). When rendering with the compressed texture (c), the block artifacts are masked as compared to rendering with the uncompressed texture (d).

Algorithm	ΔE^*_{94}	**SSIM**	**RMSa**
NVTT/BC	0.0093	0.9990	1.7788
ASTC 12×12	0.0305	0.9830	1.2883

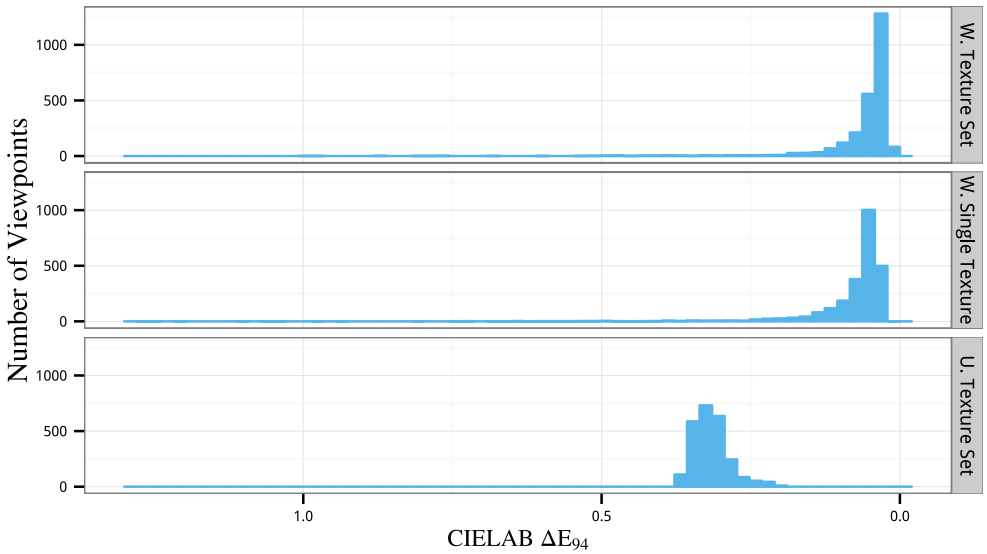
Table 4.1: Mean color (ΔE^*_{94}) and structure (**SSIM**) error for normal compressed textures and the mean RMS Angular Error (**RMSa**) over all of the MIP levels of the largest sized texture set for the Urban Guy model.

These results clearly indicate masking is occurring. However, since I render final images using diffuse, gloss, and bump maps, it is unclear whether this masking is caused by geometric masking or texture set masking.

Relative masking effects vary by texture class.

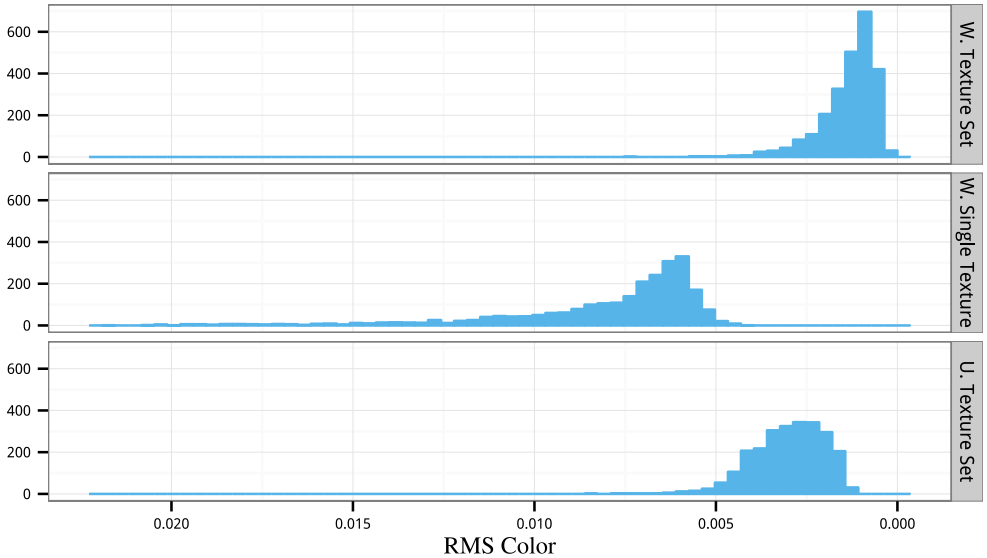
Figures 4.8, 4.9, and 4.10 plot the ASTC 12×12 metric histograms and statistics for Diffuse, Gloss, and Normal only compression for the Fire Hydrant and Taurus models respectively. Normal RMS Angular histograms are clipped at x=0.05. **W.** is Wrapped and **U.** is Unwrapped. These plots attempt to classify the relative contributions between geometric masking and texture set masking.

Looking at the Diffuse Only plots for both models, the **W. Single Texture** plots show



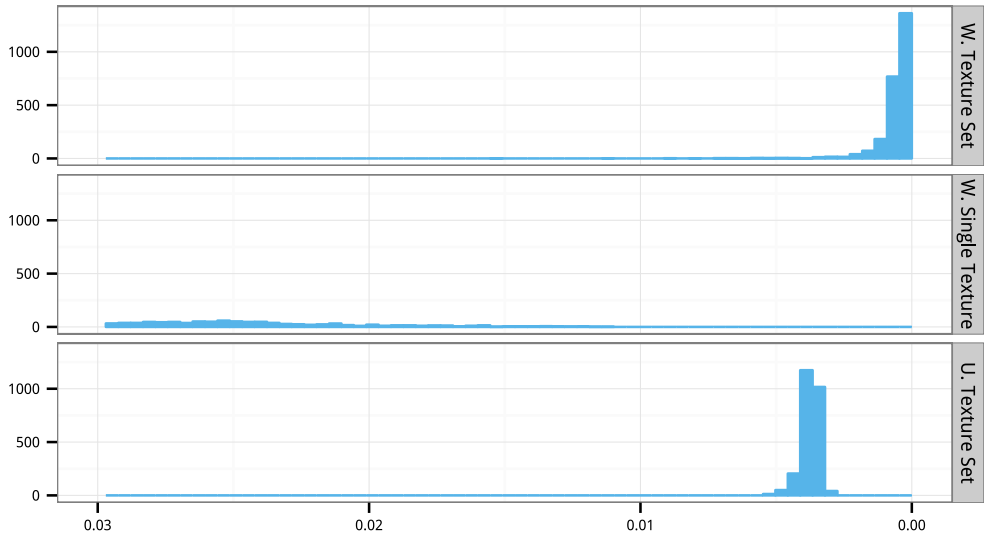
Test	Min	Max	Mean	Variance
W. Texture Set	0.0125	0.9991	0.0608	0.00555998
W. Single Texture	0.0210	1.2717	0.0836	0.00952325
U. Texture Set	0.2039	0.3694	0.3164	0.00087361

Figure 4.8: ASTC 12×12 metric histograms and statistics for Diffuse only compression on the Fire Hydrant model. **W.** is Wrapped and **U.** is Unwrapped. Values to the right are better.



Test	Min	Max	Mean	Variance
W. Texture Set	0.0003	0.0075	0.0013	0.00000059
W. Single Texture	0.0043	0.0218	0.0080	0.00000785
U. Texture Set	0.0011	0.0083	0.0030	0.00000095

Figure 4.9: ASTC 12×12 metric histograms and statistics for Gloss only compression on the Fire Hydrant model. **W.** is Wrapped and **U.** is Unwrapped. Values to the right are better.



Test	Min	Max	Mean	Variance
W. Texture Set	0.0002	0.0155	0.0007	0.00000071
W. Single Texture	0.0112	0.9360	0.0565	0.00593989
U. Texture Set	0.0031	0.0052	0.0038	0.00000011

Figure 4.10: ASTC 12×12 metric histograms and statistics for Normal only compression on the Fire Hydrant model. This histogram is clipped on the left. **W.** is Wrapped and **U.** is Unwrapped. Values to the right are better.

similar ΔE^*_{94} color quality to **W. Texture Set**, while the **U. Texture Set** plots show worse quality. Since quality gets worse when the texture set is unwrapped, geometric masking has a greater affect on diffuse texture compression artifacts.

Looking at the Gloss Only plots for both models, the relative masking effects are reversed. The **W. Single Texture** plots show worse RMS Color quality, while the **U. Texture Set** plots show similar quality to **W. Texture Set**. Since quality gets worse when a single texture is used, texture set masking has a greater affect on gloss texture compression artifacts.

Finally, for the Normal Only plots, the results are similar to the Gloss Only plots. The main difference is that masking effects from normal map compression are significantly less dependent on the viewpoint. This wide variance in the texture set masking effects is illustrated in Figures 4.11 and 4.12 which plot the sampled viewpoints coloring each viewpoint by its RMS Angular Error metric score.

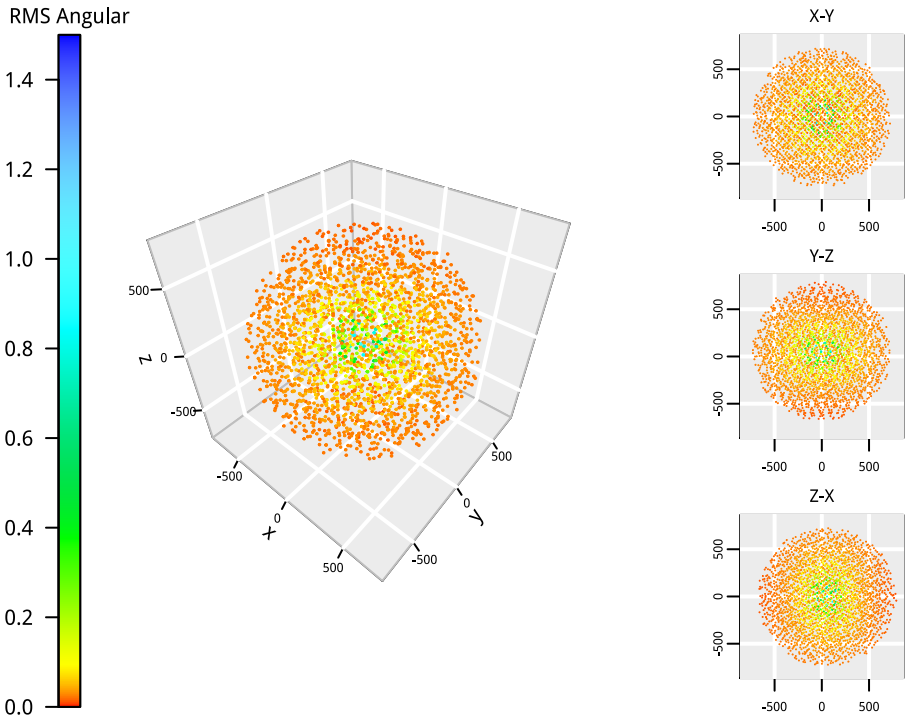


Figure 4.11: Viewpoints for the Fire Hydrant model colored by RMS Angular Error with only normal textures compressed using ASTC 12×12.

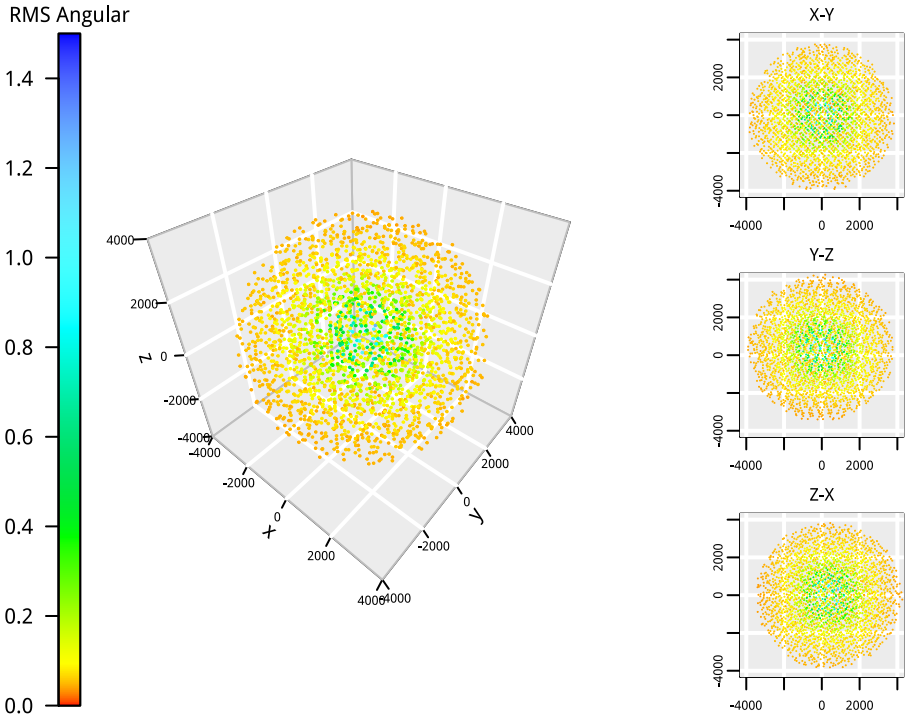


Figure 4.12: Viewpoints for the Taurus model colored by RMS Angular Error with only normal textures compressed using ASTC 12×12.

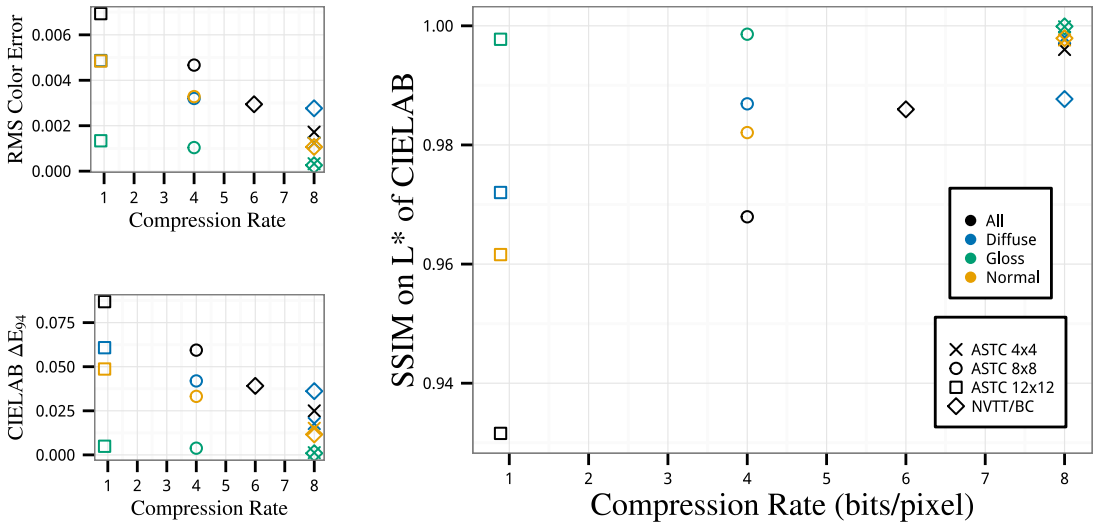


Figure 4.13: **Wrapped Texture Set** mean error for the Fire Hydrant model. The left two plots are color error against compression rate and the right plot is SSIM. For RMS Color Error and CIELAB ΔE^*_{94}, lower is better and 0.0 is perfect. For SSIM, higher is better and 1.0 is perfect. The vertical error scales **are different** across all plots.

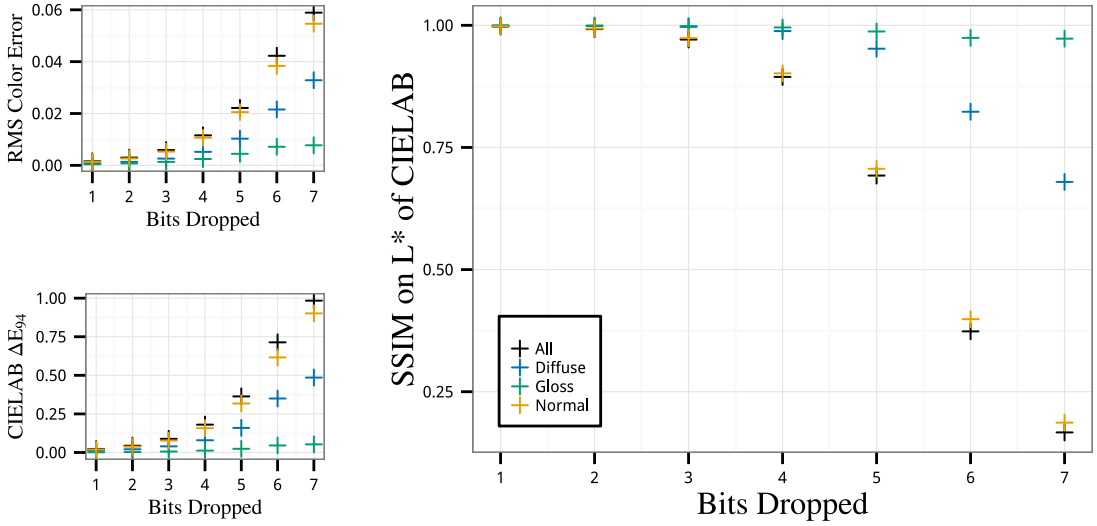


Figure 4.14: **Wrapped Texture Set** mean error for the Fire Hydrant model. The left two plots are color error against compression rate and the right plot is SSIM. For RMS Color Error and CIELAB ΔE^*_{94}, lower is better and 0.0 is perfect. For SSIM, higher is better and 1.0 is perfect. The vertical error scales **are different** across all plots.

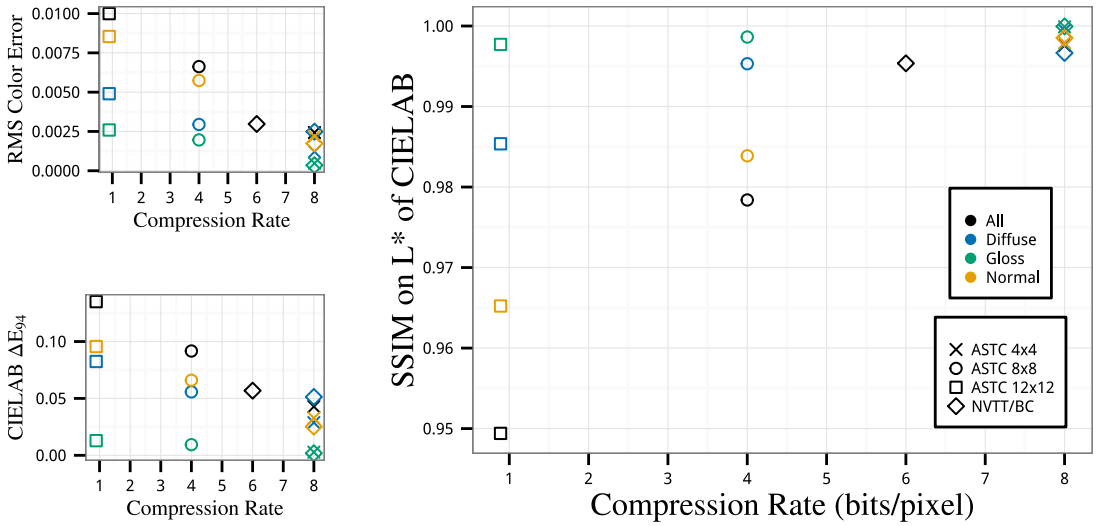


Figure 4.15: **Wrapped Texture Set** mean error for the Taurus model. The left two plots are color error against compression rate and the right plot is SSIM. For RMS Color Error and CIELAB ΔE^*_{94}, lower is better and 0.0 is perfect. For SSIM, higher is better and 1.0 is perfect. The vertical error scales **are different** across all plots.

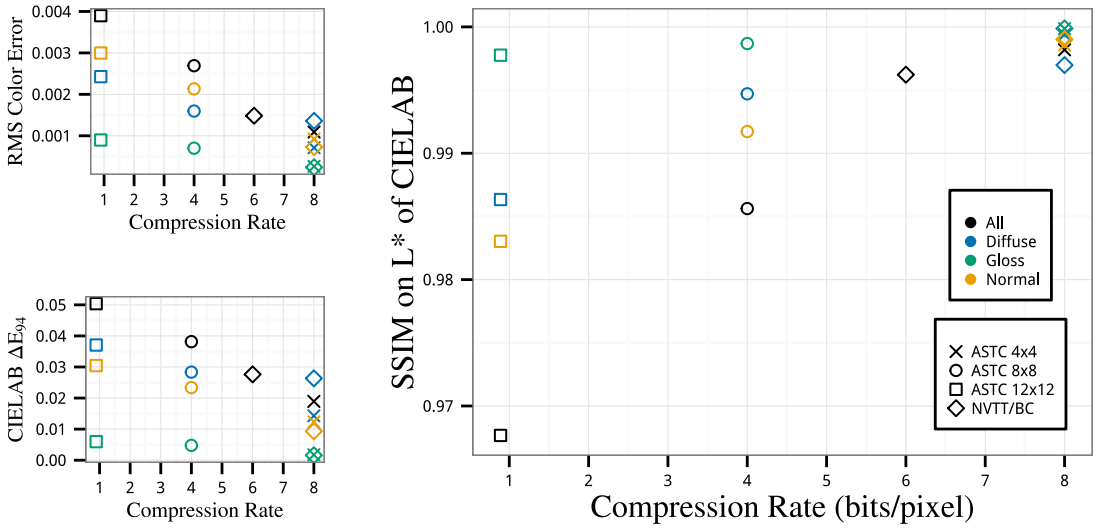


Figure 4.16: **Wrapped Texture Set** mean error for the Urban Guy model. The left two plots are color error against compression rate and the right plot is SSIM. For RMS Color Error and CIELAB ΔE^*_{94}, lower is better and 0.0 is perfect. For SSIM, higher is better and 1.0 is perfect. The vertical error scales **are different** across all plots.

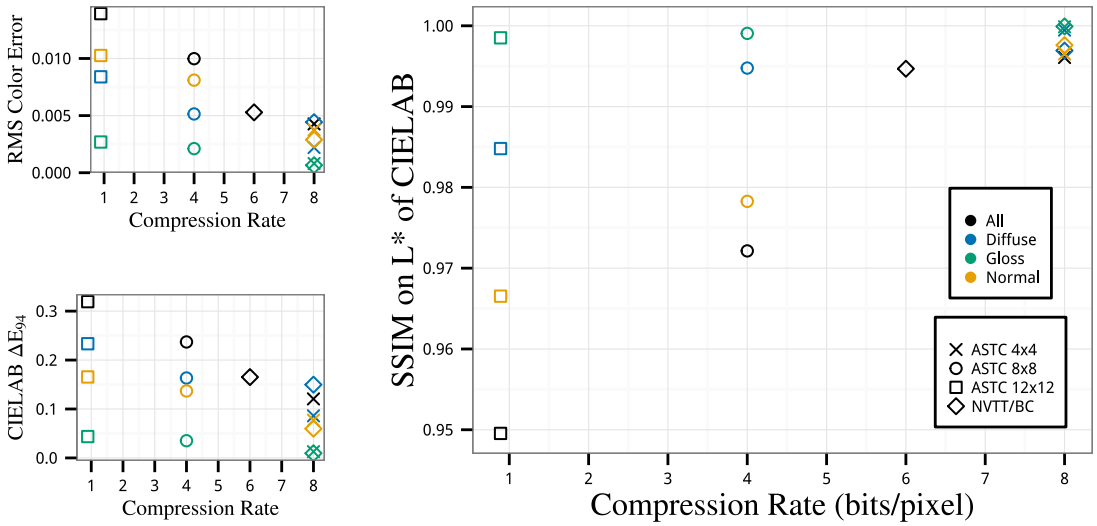


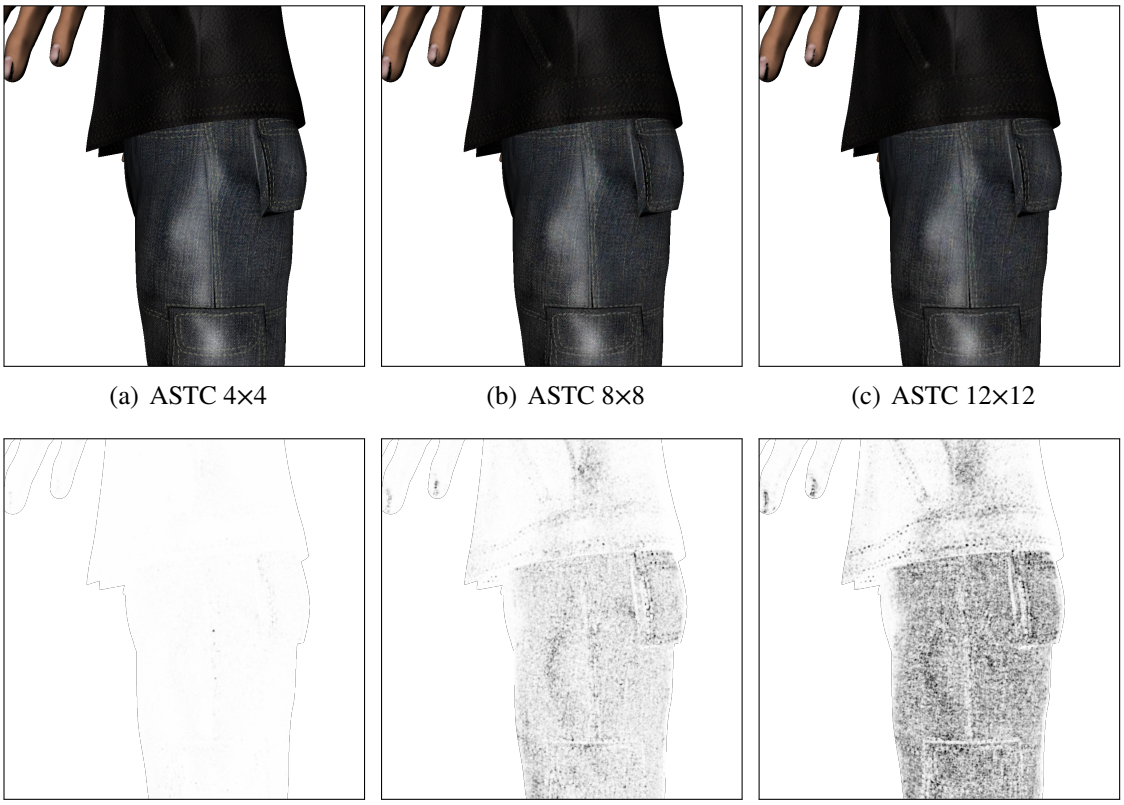
Figure 4.17: **Wrapped Texture Set** mean error for the Japanese Castle model. The left two plots are color error against compression rate and the right plot is SSIM. For RMS Color Error and CIELAB ΔE^*_{94}, lower is better and 0.0 is perfect. For SSIM, higher is better and 1.0 is perfect. The vertical error scales **are different** across all plots.

Perceptual sensitivity varies by texture class.

Figures 4.13, 4.14, 4.15, 4.16, and 4.17 plot the mean of the error metrics for the **Wrapped Texture Set** test case for each model. The shape of the points corresponds to compression algorithm and the color to variations in compressed textures. For NVTT/BC compression, the All points are average bit rate at each pixel. The right plots are SSIM and color error against dropping bits from each color plane and the color corresponds to the variations in "compressed" textures. The vertical error scales **are different** across all plots. These figures reveal that perceptual sensitivity to compression artifacts and masking effects varies based on the class of texture. The Gloss results show very little reduction in quality as both the compression rate in bits/pixel is reduced and increasing numbers of bits are dropped. This implies that texture classes can be compressed differently based on perceptual sensitivity.

GPU compression is too conservative.

Figures 4.13, 4.14, 4.15, 4.16, and 4.17 plot the mean of the error metrics for the **Wrapped Texture Set** test case for each model. Figures 4.18 and 4.19 show several individual



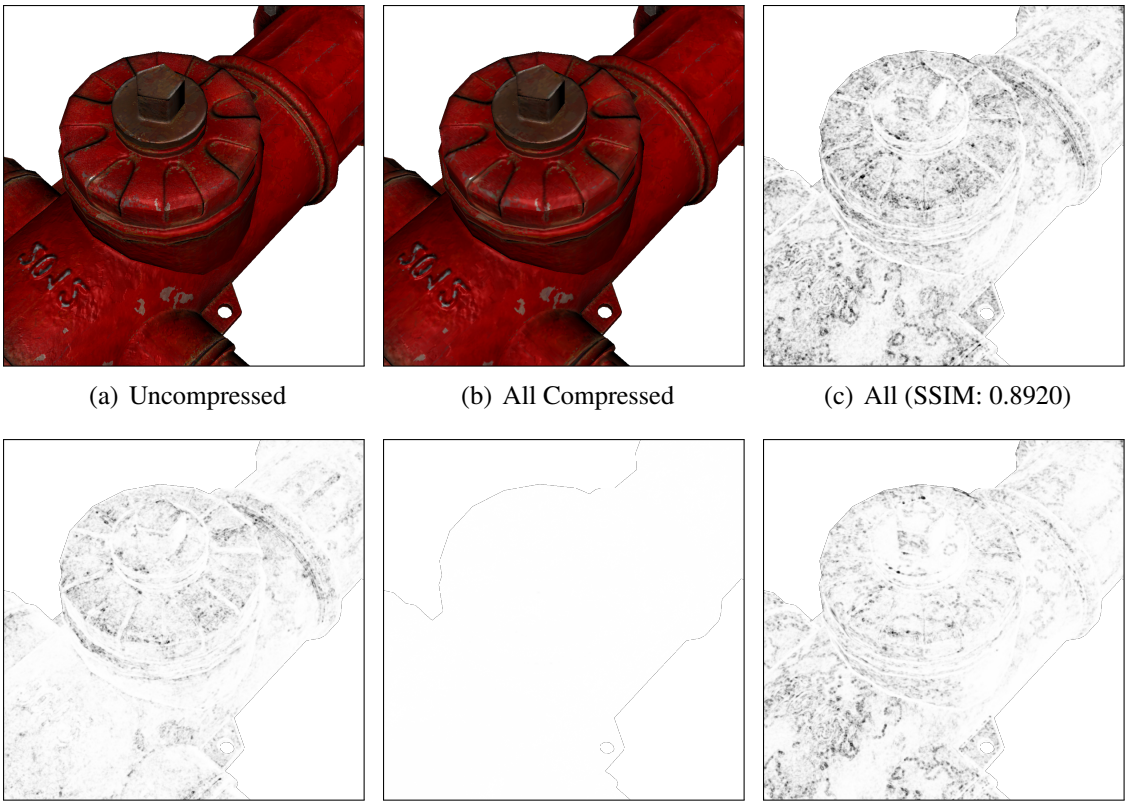
(a) ASTC 4×4 (b) ASTC 8×8 (c) ASTC 12×12

(d) SSIM: 0.9958 (e) SSIM: 0.9241 (f) SSIM: 0.8210

Figure 4.18: Visual comparison of a single viewpoint of the Urban Guy model with each ASTC variant. Figures (d)–(f) are the SSIM maps.

viewpoints of the Urban Guy and Fire Hydrant models respectively with various compression algorithms and corresponding SSIM maps. As these plots and figures illustrate, geometric and texture set masking effects can be leveraged to reduce the bit rate of compressed textures while still maintaining good final rendered image quality. In particular, Figure 4.19 implies SSIM values as low as 0.9 still result in good final rendered image quality.

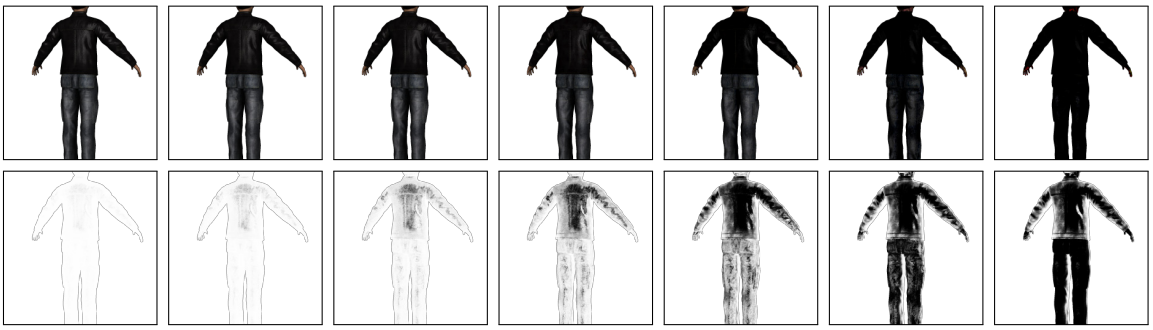
To explore how far compression can be pushed, Figure 4.20 visually shows the effects of dropping from 1 to 7 bits from each color plane for all textures at a single viewpoint and the SSIM map and mean value. Interestingly, dropping three bits of color from each plane still results in an SSIM mean value higher than 0.9 indicating good rendered image quality.



(a) Uncompressed (b) All Compressed (c) All (SSIM: 0.8920)

(d) Diffuse (SSIM: 0.9511) (e) Gloss (SSIM: 0.9999) (f) Bump (SSIM: 0.9427)

Figure 4.19: Visual comparison of a single viewpoint of the Fire Hydrant model and the ASTC 12×12 compression algorithm. Figures (c)–(f) are SSIM maps.



Drop 1 bit	Drop 2 bits	Drop 3 bits	Drop 4 bits	Drop 5 bits	Drop 6 bits	Drop 7 bits
SSIM 0.9942	SSIM 0.9772	SSIM 0.9171	SSIM 0.7764	SSIM 0.5359	SSIM 0.3207	SSIM 0.2450

Figure 4.20: Visual comparison of dropping from 1 to 7 bits on all textures. The bottom row is the corresponding SSIM maps with the mean SSIM from this viewpoint.

Chapter 5

TEXTURE COMPRESSION OPTIMIZATION

Modern texture compression algorithms (Khronos Group 2013; Skodras, Christopoulos, and Ebrahimi 2001) have many "knobs" for fine-tuning the compression. For example, the ASTC algorithm (Khronos Group 2013) has seven setting variables, five of which are discrete, while the other two are continuous, and all of them having a fixed range. These knobs are used to fine-tune the compression algorithm by trading speed for bitrate.

Even with just seven parameters, the compression algorithms provide setting "presets" that set specific values for each variable. These presets provide end-users with a way to quickly choose a desired trade-off between speed and bitrate without having to understand the various knobs. For ASTC, the presets are named *veryfast*, *fast*, *medium*, *thorough*, and *exhaustive*. As expected, the *veryfast* preset will compress sacrifice bitrate to decrease the compression time, while *exhaustive* will sacrifice compression time to decrease the bitrate.

Even with presets, the effect of the variables on bitrate and quality can be non-obvious to non-experts. Furthermore, even for experts, manually tuning each variable is a time-consuming process with usually very little payoff. For ASTC, the seven setting variables are listed in Table 5.1. Assuming 1000 discrete steps for *Optimization Limit* and 100 discrete steps for *Correlation Cutoff*, those knobs combine to form a search space of $6^2 \cdot 1020 \cdot 1000 \cdot 100^2 \cdot 4 \geq 10^{12}$ possible states. Enumerating that many states is clearly infeasible for just a single image and modern video games uses thousands of textures, further compounding the problem.

The large state space created by combining the knobs, however, suggests using a search algorithm for guided optimization. Leveraging my compression evaluation methodology, I introduce a procedure that can optimize an energy function incorporating bitrate and

Name	Min	Max	Step
X Blocksize	1	6	1
Y Blocksize	1	6	1
Partitions Limit	2	1024	1
Optimization Limit	1.0	1000.0	1†
Correlation Cutoff	0.5	0.99	0.0049†
Block Modes Cutoff	0	100	1
Max Iterations	1	4	1

Table 5.1: ASTC Compression Settings. The † for **Step** indicates a continuous value variable.

objective image quality.

The following results present:

- An energy function evaluated by applying perceptually rigorous objective image quality assessment metrics to compare compressed textures to uncompressed textures over a Monte-Carlo sampling of viewpoints.
- A measure to determine if the Monte-Carlo estimate is accurate enough.
- A variance analysis in the Monte-Carlo estimate showing that 2,048 viewpoint samples provides a sufficiently accurate estimate of compress texture quality with a sizable margin of error.

5.1 Approach

My optimization algorithm consists of three stages: 1) compress, 2) render, and 3) compare. Each stage is performed once for each iteration of the algorithm. After the compare stage, an energy function is evaluated and with the goal of minimizing the energy. I discuss each stage in detail below.

To reduce the complexity of the search space, I chose to restrict the search space to just the *X Blocksize* and *Y Blocksize* variables. Even after ignoring the other variables, that still leaves twelve possible states for each texture. With three textures per texture atlas set (Section 4.2.1), there are $\binom{18}{6} = 18{,}564$ possible states for the Fire Hydrant Model. The

Urban Guy model is more representative of real-world texture usage and has 12 total textures for $\binom{72}{6} = 156{,}238{,}908$ possible states. So even after reducing the complexity, exhaustive enumeration of the state space is infeasible.

As described in Section 2.3, I define a problem where the **initial state** is a randomly chosen set of values, one for each parameter, the only possible **action** is to compress the textures with the given set of values, the **path cost** is a function of bitrate and image quality, and the **goal test** is a state with minimal bitrate and maximal quality.

As discussed, the ASTC algorithm parameters combine to form a state space with more than 10^{12} possible states. Fully enumerating this state space is impractical and constructing a differential equation for gradient descent is also problematic. As such I have chosen to use hill-climbing to locally search the state space.

Hill-climbing algorithms are a class of local search algorithms that are greedy in that they only look at immediate neighbors and pick the neighbor with the best improvement. Stochastic hill-climbing randomly picks from the better neighbors while first-choice hill-climbing randomly generates neighbors to evaluate until a better neighbor is found.

Even after reducing the number of variables to just the X and Y blocksizes, I have to track the compressor state for each texture on a model. Given the large dimensions of the search space, I use first-choice hill-climbing, where, from the current state, a neighbor state is randomly chosen and evaluated. If that new state is better, the algorithm accepts it, otherwise the algorithm evaluates a different random neighbor state.

Compress Each texture on the model has a unique compression state. For each texture, a random neighbor in the state space for that texture is chosen and then each texture is compressed with that candidate state.

Render After compressing, the model is rendered using a set of sampled viewpoints. The viewpoint sampling process is described in detail in Section 4.1.1.

Compare After rendering, the rendered images with compressed textures are compared with rendered ground-truth images using uncompressed textures. This comparison is fully discussed in Section 4.1.2. For guided optimization, I only evaluate the CIELAB ΔE_{94} and SSIM metrics.

Energy Computation The result from the compress stage is an overall bitrate and the result from the compare stage is a mean ΔE^*_{94} and SSIM value. To combine these into an energy function for minimization, I use a simple average:

$$\frac{(1 - \text{SSIM}) + \Delta E^*_{94} + \text{Bitrate}}{3} \tag{5.1}$$

where SSIM has a range of 0.0 to 1.0 and 1.0 is a perfect match, lower ΔE^*_{94} values are better and 0.0 is a perfect match and lower bitrate values are better. Since I am minimizing energy, reversing SSIM $(1 - \text{SSIM})$ aligns it with ΔE^*_{94} and Bitrate.

Once the energy is computed, the compressor is updated with the computed energy. At this point, if the energy is lower than the current best energy, the state is "accepted" as the new best state and a new iteration is started. If the computed energy is not lower than the current best energy, then a new neighbor state is randomly chosen.

5.2 Results

I present results from the Fire Hydrant (Figure 4.1(a) and Urban Guy (Figure 4.1(c)) models. These models are described more fully in Section 4.2.1. I first discuss the convergence of the optimization framework, followed by an analysis of the variance in the components of the energy function, and finally present some performance data.

For both models, I am able to sample 12,288 maximum viewpoints. With 1000×1000 pixel rendered images, this is the most viewpoints that would fit within 128 Gb of RAM. As such, I treat 12,288 viewpoints as a "ground truth" data set. In the following results, an "iteration" refers to when the optimization algorithm accepts a new state with lowest energy. The number of neighbor states that are evaluated each iteration are referred to as "steps."

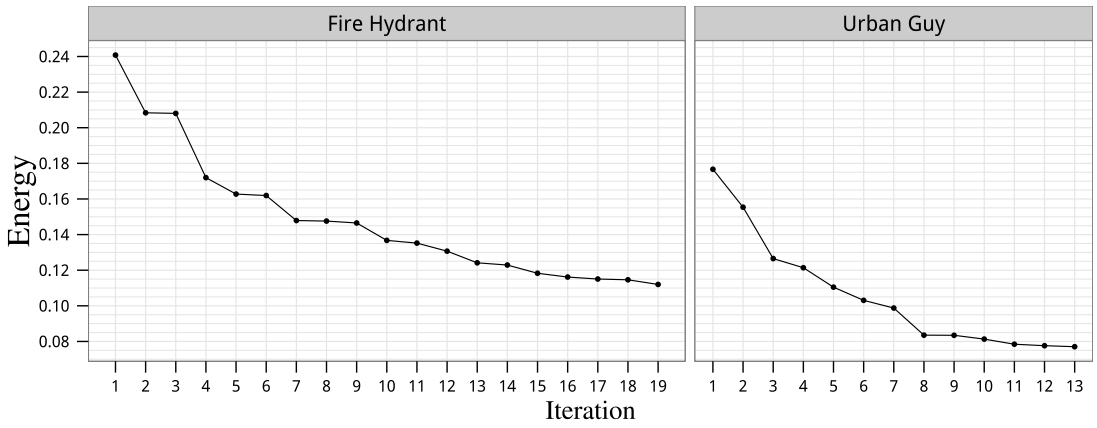


Figure 5.1: Energy by iteration for the Fire Hydrant and Urban Guy models with 12,288 viewpoint samples. For both models, energy is minimized over time.

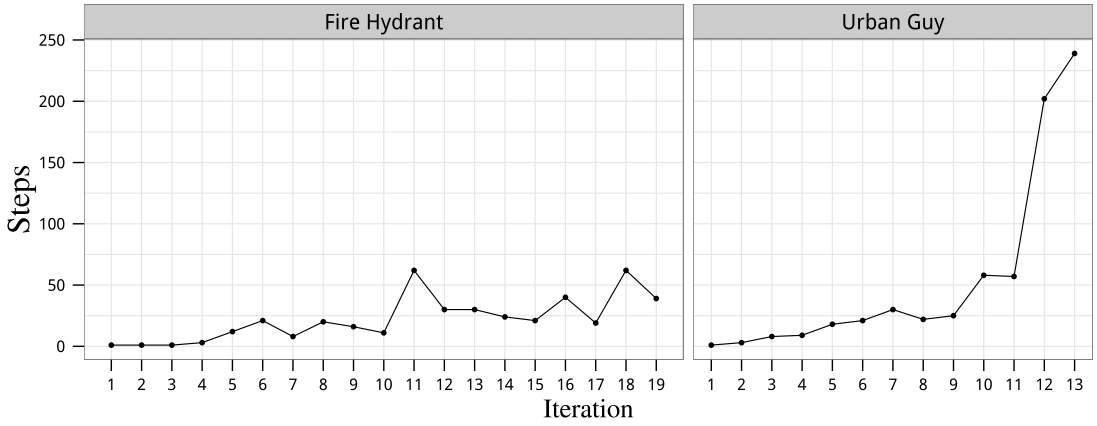


Figure 5.2: Steps by iteration for the Fire Hydrant and Urban Guy models with 12,288 viewpoint samples. For both models, the number of steps taken at each iteration increases. This shows the algorithm is spending more time looking a better neighbor state.

5.2.1 Convergence

Figure 5.1 plots the energy function per iteration for the Fire Hydrant and Urban Guy models with 12,288 viewpoint samples. For both models, energy is minimized over time as expected. The different starting energies are due to the random selection of initial state. Figure 5.2 plots the steps taken at each iteration for the Fire Hydrant and Urban Guy models with 12,288 viewpoint samples. Given decreasing energy over time, this shows that the algorithm is spending more time looking for a better neighbor state.

Figures 5.1 and 5.2 initially show that guided optimization is working. However, when

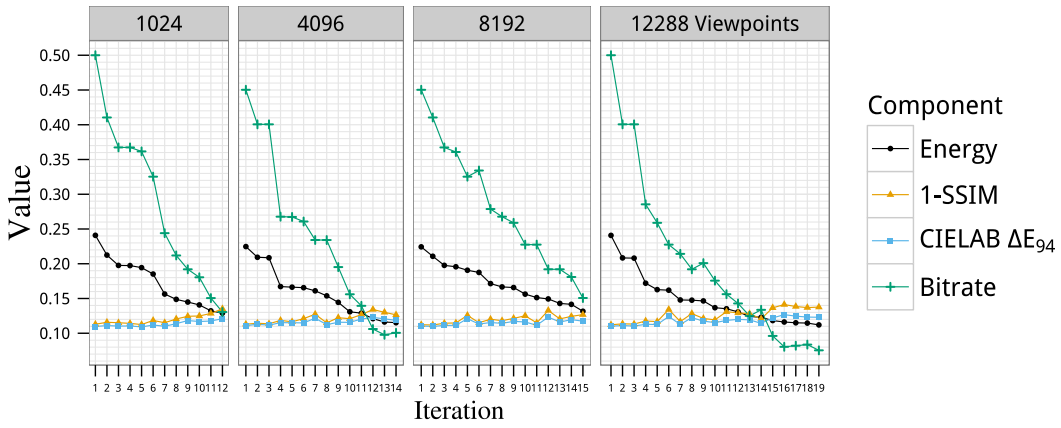
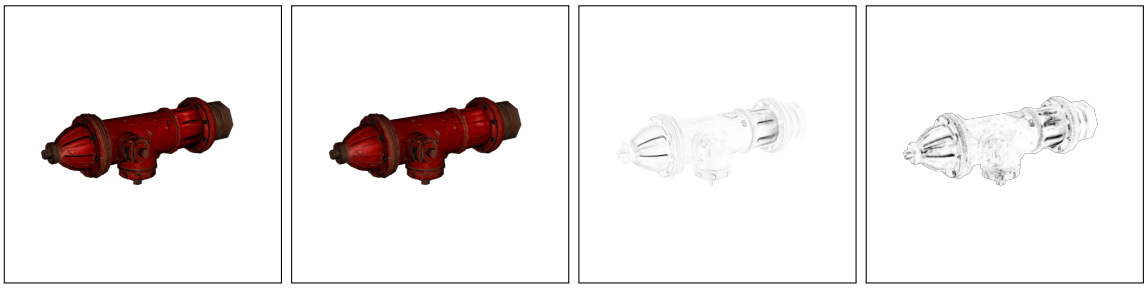


Figure 5.3: Energy components by iteration for the Fire Hydrant model with various viewpoint samples. Bitrate is being over-optimized at the expense of rendered image quality.

the components of the energy function are plotted, as in Figure 5.3, we see that the algorithm is over-optimizing bitrate at the expense of rendered image quality. In Figure 5.3, we would expect to see the components converge, with energy remaining in between the bitrate and image quality components. However, the algorithm is over-optimizing as the bitrate is pushed well below the error metrics. Certainly for 12,288 viewpoint samples, we can see the image quality metrics getting worse as the bitrate is pushed lower. This is counter-intuitive, as the algorithm should be minimizing all functions and implies that the image quality metrics should be weighted somehow in the energy function.

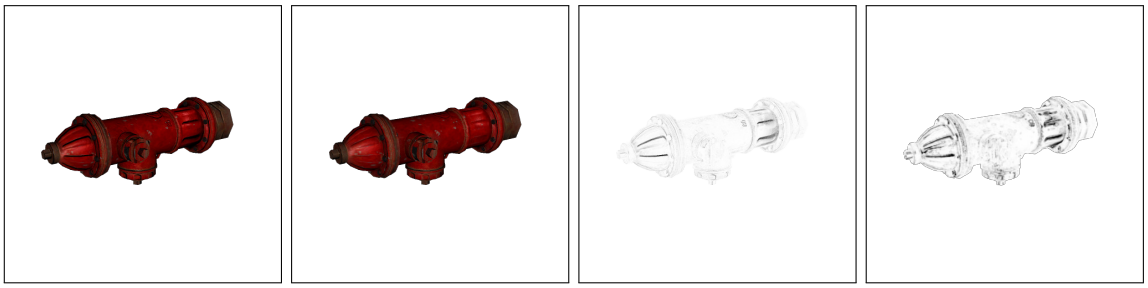
Looking at iteration 12, Figure 5.4 shows the uncompressed (Figure 5.4(a)), compressed (Figure 5.4(b)), 1 - CIELAB ΔE^*_{94} error (Figure 5.4(c)), and SSIM error (Figure 5.4(d)). Figure 5.5 shows the corresponding images for iteration 19. For easier comprehension, the error metrics are shown such that darker pixels indicate worse quality and lighter pixels indicate better quality. This is opposite to how the metrics are used in the energy function, where a lower metric value is desired as it reflects better quality.

Figures 5.4 and 5.5 show that the image quality is not very different between iteration 12 and 19 for this specific viewpoint. Table 5.2 also shows that the overall 1-SSIM and ΔE^*_{94} values are similar, while the bitrate has been cut in half. These results imply that the error metrics should likely be weighted in the energy function, which I leave for future work.



(a) Uncompressed (b) Compressed (c) 1 - ΔE^*_{94} (0.94629) (d) SSIM (0.81699)

Figure 5.4: Iteration 12 of the Fire Hydrant model with 12,288 viewpoint samples. There is some visual difference between the Uncompressed (a) and Compressed (b) images and this is reflected in the 1 - CIELAB ΔE^*_{94} (c) and SSIM (d) maps. For both (c) and (d), darker pixels indicate worse quality. This is opposite to how the image quality metrics are used in the energy function.



(a) Uncompressed (b) Compressed (c) 1 - ΔE^*_{94} (0.94402) (d) SSIM (0.79813)

Figure 5.5: Iteration 19 of the Fire Hydrant model with 12,288 viewpoint samples. There is some visual difference between the Uncompressed (a) and Compressed (b) images and this is reflected in the 1 - CIELAB ΔE^*_{94} (c) and SSIM (d) maps. For both (c) and (d), darker pixels indicate worse quality. This is opposite to how the image quality metrics are used in the energy function.

Looking at the energy components in Figures 5.3 and 5.6, we see that changes in bitrate are very large compared to changes in ΔE^*_{94} or 1-SSIM.

Figure 5.6 plots the energy components for the Urban Guy model. These results also show the over-optimization of bitrate at the expense of rendered image quality. In future work I will be performing a systematic exploration of weighting terms for the energy function components to develop an energy function that does not over-optimize.

Iteration	Bitrate	1-SSIM	ΔE^*_{94}	Energy
12	0.14286	0.87073	0.11996	0.13069
19	0.07540	0.86236	0.12298	0.11201

Table 5.2: The components and energy at iterations 12 and 19 for the Fire Hydrant model with 12,288 viewpoint samples. Even though the bitrate has been cut in half between iterations 12 and 19, the 1-SSIM and CIELAB ΔE^*_{94} values are very similar.

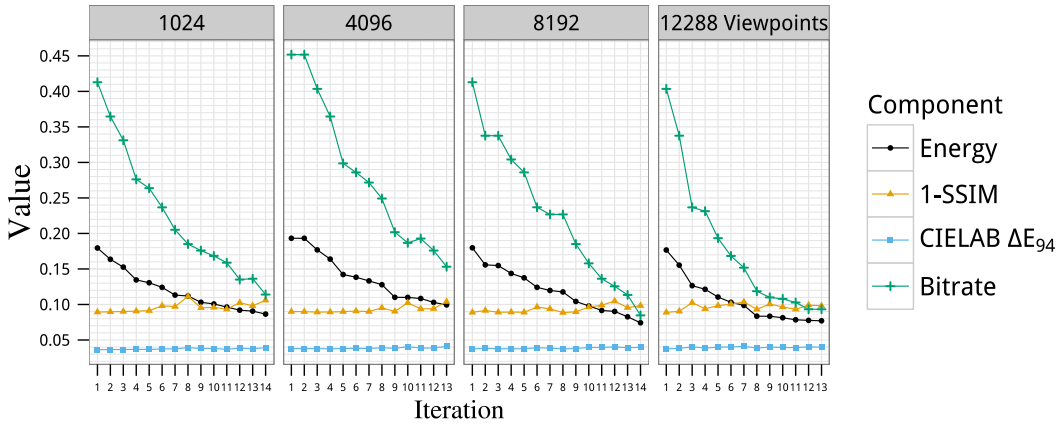


Figure 5.6: Energy components by iteration for the Urban Guy model with various viewpoint samples. Bitrate is being over-optimized at the expense of rendered image quality.

5.2.2 Variance Analysis

It is possible that 12,288 viewpoint samples are not enough to fully capture the errors introduced by the compression. That is, if the error in estimating the energy function is higher than the changes in the actual energy function from step to step, then the optimization algorithm will accept objectively bad states due solely to the error in the estimated energy. Since the viewpoint samples are quasi-randomly generated in a sphere surrounding the model, the image comparison metrics computed at each viewpoint might not be "seeing" important compression artifacts. To explore this situation, I analyze the variance of the energy components for both models.

Using all 12,288 viewpoint samples, I calculate the variance of a subsampled estimate. That is, for the set of viewpoint samples $V = (16, 32, 48, \ldots, 6128, 6144)$, I divide the 12,288 viewpoint samples into an estimate: $E = (1, x)$, where $x \in V$, and an actual:

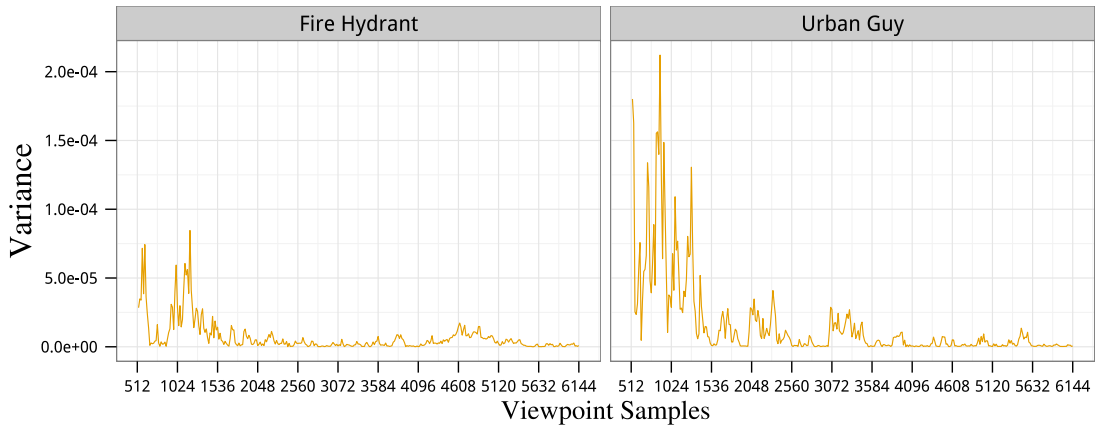


Figure 5.7: Variance analysis for the 1-SSIM component. The variance is significantly smaller than the 1-SSIM delta in Figures 5.3 and 5.6, which is on the order of 0.1, implying that even 2,048 viewpoint samples is sufficient. Section 5.2.2 describes how the variance was calculated.

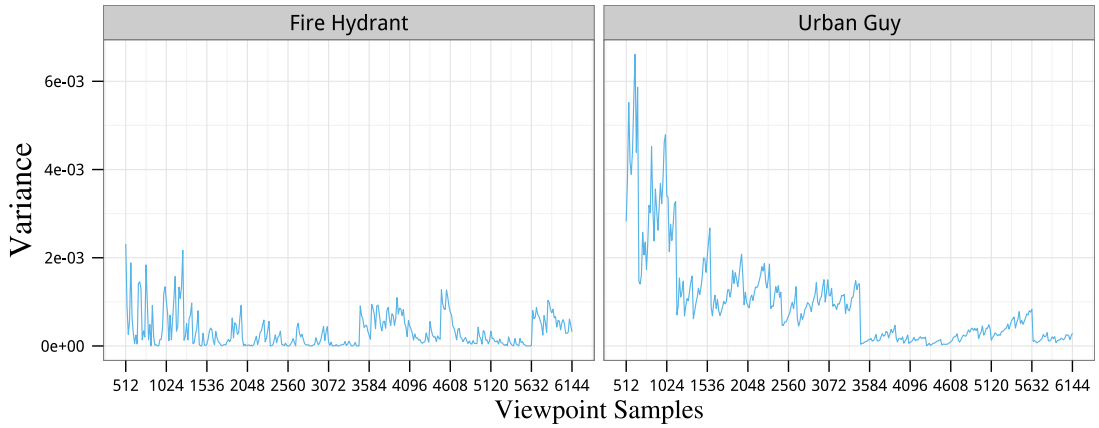


Figure 5.8: Variance analysis for the CIELAB ΔE^*_{94} component. The variance is significantly smaller than the ΔE^*_{94} delta in Figures 5.3 and 5.6, which is on the order of 0.3–0.4, implying that even 2,048 viewpoint samples is sufficient. Section 5.2.2 describes how the variance was calculated.

$A = (x + 1, 12288)$. I then compute the variance between the estimate and the actual: $(E - A)^2$. Figures 5.7, 5.8, and 5.9 plot these variances for the two image quality metrics and energy.

Comparing the variance plots to the 12,288 viewpoint sample cases in Figures 5.3 and 5.6, we can see that the variance is significantly smaller than the change in each component, even with as few as 2,048 viewpoint samples. First, this indicates that the over-

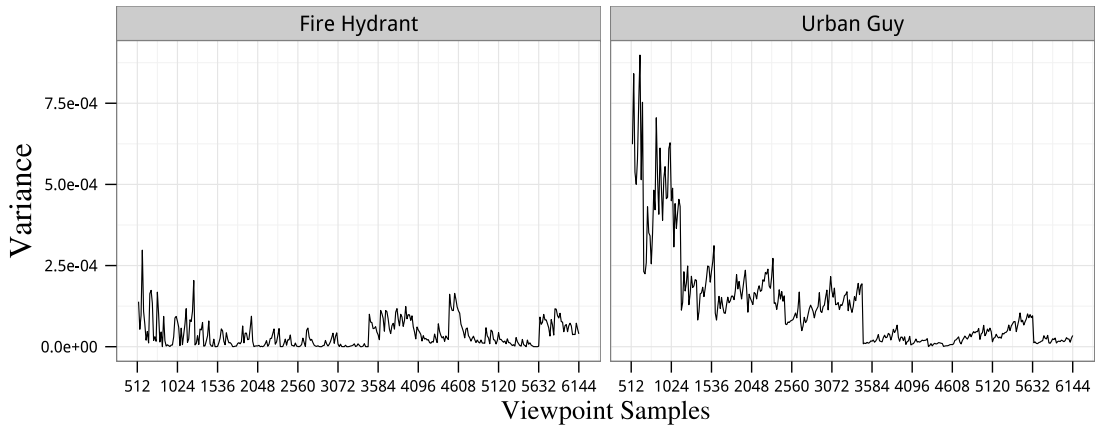


Figure 5.9: Variance analysis for energy. The variance is significantly smaller than the energy delta in Figures 5.3 and 5.6 implying that even 2,048 viewpoint samples is sufficient. Section 5.2.2 describes how the variance was calculated.

CPU	**Chipset**	**Cores† P/H**	**Clock (Ghz)**	**RAM (Gb)**
Dual Xeon E5-2687W v2	Sandy Bridge	16/32	3.4	128
Xeon E5-2687W	Sandy Bridge	8/16	3.1	32
Core i7-5960X	Haswell	8/16	3.0	128
Xeon W5590	Nehalem	4/4	3.3	12

Table 5.3: System configurations. †The Cores are listed as physical / hyperthreaded, where the total number of available cores is the **H** column.

optimization is not due to insufficient viewpoint samples. Second, these results also show that 2,048 viewpoint samples may be sufficient for optimizing these two models resulting in a significantly faster optimization process as I explain in the following section.

5.2.3 Runtime Performance

To understand the runtime performance of the optimization algorithm, I ran several different viewpoint sample cases for the Fire Hydrant model on identical hardware. I also ran the 12,288 viewpoint sample case for the Urban Guy on the same hardware. Overall I used four different hardware configurations which are outlined in Table 5.3 to generate all of the results presented here.

Figure 5.10 plots the mean walltimes for each stage of the optimization algorithm for

	Time (s)			
Model	**Compare**	**Render**	**Compress**	**Total**
Fire Hydrant	759.8	49.7	155.4	964.9
Urban Guy	761.2	283.2	188.6	1233.0

Table 5.4: Mean walltimes for the different stages of the optimization algorithm for the Fire Hydrant and Urban Guy models with 12,288 viewpoints.

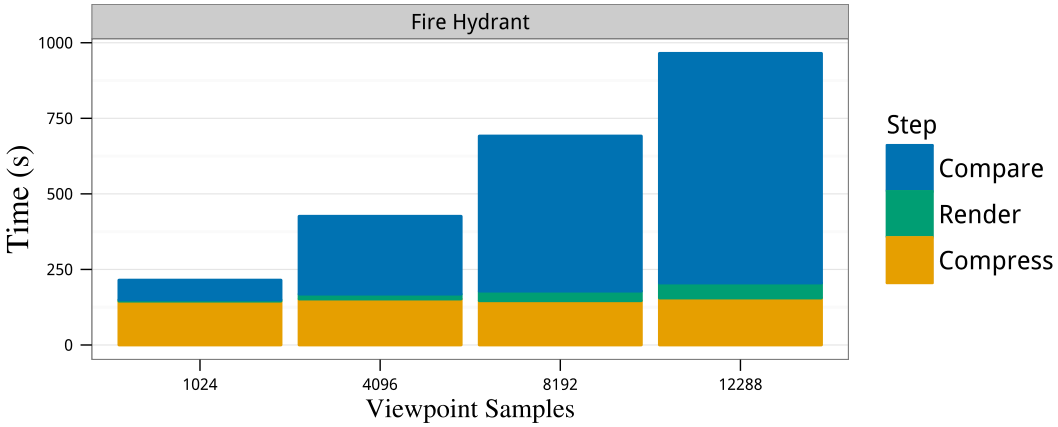


Figure 5.10: Mean walltimes for the different stages of the optimization algorithm for the Fire Hydrant model with various viewpoint samples. As expected, the overall runtime of the algorithm increases proportionally with the number of viewpoint samples.

several different viewpoint sample cases. These times are all from the Dual Xeon E5-2687W v2 system with 128 Gb of RAM. As expected, as the number of viewpoint samples increases, the total running time of the algorithm increases as well. Furthermore, the **Compare** stage of the algorithm clearly dominates the running time. Since I am computing two image comparison metrics for each viewpoint sample, this makes sense. My implementation currently runs only on the CPU, and while the **Compare** stage does utilize all available CPU cores, the image comparison metrics would run much faster with a GPU implementation.

The Fire Hydrant model is relatively simple compared to the Urban Guy model (one texture atlas set compared to four texture atlas sets as described in Section 4.2.1.) Table 5.4 presents the mean walltimes for each stage of the algorithm for both models with 12,288 viewpoint samples. As expected, the **Compare** stage takes the same amount of time for both models. Surprisingly, the **Compress** times are relatively close given that the optimization

algorithm is only compressing three textures for the Fire Hydrant model but twelve textures for the Urban Guy model.

Chapter 6

CONCLUSION

Real-time rendering systems have two competing constraints: rendered frame time and rendered frame quality. To ensure interactivity and provide the illusion of fluid motion, real-time rendering must render frames in 33 milliseconds or less. Furthermore, consumers demand increasingly realistic graphics and video games cannot sacrifice quality while meeting the time constraint. These two constraints have largely driven the performance gains in GPUs, which has, in turn driven the push for improved graphics realism.

Real-time rendering uses texture mapping to meet both time and quality constraints. Current rendering algorithms make use of several different textures per object, each storing artist-created data, including color and 3D vectors. Additionally, high definition displays are driving texture pixel resolutions higher, causing an increase of 64 times in the texture sizes. The increase in texture count combined with the increase in texture resolution has caused dramatic growth in texture storage data requirements.

GPUs offer a texture compression format with efficient hardware decompression but the quality of this FBR format no longer meets the storage and quality requirements of video games. The FBR algorithms have been extended to try and address this problem (Khronos Group 2013) and recent texture compression research has focused on VBR algorithms (Olano et al. 2011). These newer algorithms have many different "knobs" for adjusting the compression algorithm and manually adjusting the parameters for textures is not feasible for real-world scenarios.

Research has shown that the peak signal-to-noise ratio is not a useful metric for optimizing image processing tasks. The field of objective IQA has since explored metrics for comparing images that correlate with human perception. This research has mostly focused

on natural luminance images. Unfortunately, textures are neither natural nor luminance. The textures are used, however, to render a final frame which is a color natural image.

The existing color IQA metrics are not competitive with the current luminance metrics and developing a competitive color IQA metric remains an open question. One issue with the current evaluation methods for IQA metrics is that most of the databases do not contain chrominance distortions. In Chapter 3, I presented a new chrominance distortion database with human subjective evaluations of the distorted images.

In evaluating texture compression, the most common approach is to use some combination of the MSE, PSNR, or visual image inspection on the individual textures. While comparing or inspecting individual textures is straightforward to implement, this method does not properly account for the masking effects introduced by the texture mapping process. In Chapter 4, I introduced a texture compression evaluation methodology that accounts for these masking effects.

The number of tuning parameters of modern texture compression algorithms combined with the number of textures used in real-time rendering poses a serious scalability solution to optimizing texture compression. These parameters, however, do create a multi-dimensional space that is a good match for guided optimization. In Chapter 5, I showed a demonstration of guided texture compression optimization building on my evaluation methodology.

Little research exists on automatic rendering optimization. Texture compression is one part of the entire rendering pipeline. It is probable that other parts of the pipeline could be optimized by evaluating the quality of final rendered images. Guided texture compression is a first step that would validate the larger technique of automatic rendering optimization.

Most importantly, the field of computer graphics has traditionally relied on manual visual evaluation of rendering algorithms. This evaluation is highly subjective and when objective evaluation does occur, the luminance metrics are very often misused by averaging a per-color-channel quality value. Appropriately applying objective IQA metrics provides more rigorous evaluation technique.

6.1 Future Work

There are three main areas for future work: 1) sampling and estimation, 2) energy function and searching, and 3) additional types of masking effects.

For both evaluation and optimization, I sample the viewpoint space uniformly. Sampling techniques such as importance sampling based on expected viewer locations could improve the evaluation and this is likely a fruitful area of research. Another possibility for future work is further analysis in the Monte-Carlo estimate that would enable a refinement schedule to vary the number of viewpoint samples over time as the energy function converges.

A systematic exploration of weighting terms for the energy function components would ensure a good energy function for optimization. Even with a weighted energy function, it is likely there are many local minima in the search space and hill climbing methods are well known to have problems with local minima. As Burke and Kendall (2014) discuss, there are several different *diversification* techniques that can be employed to help search algorithms explore large regions of the state space. Even with diversification, incorporating metaheuristic techniques, such as tabu search (Glover 1989, 1990), will likely improve the optimization framework.

All of my results are with a single model at a small range of visible scales. Real-world use cases are going to have multiple models arranged in a scene. Just having multiple models likely introduces *occlusion* masking where one object is obscured by another. Also, with multiple objects, there will probably be large scale differences between the objects which will also have an effect on final rendered image quality.

6.2 Model Credits

The Taurus Model is by Thinking On Pause and the Urban Guy model is by Inventerion Productions, both from Turbo Squid. The Fire Hydrant model is by Trap Door and the Japanese Castle scene is by JD Creative Machine, both from the Unity Store.

REFERENCES

Ajagamelle, Sebastien Akli, Marius Pedersen, and Gabriele Simone. 2010. "Analysis of the Difference of Gaussians Model in Image Difference Metrics." In *Proc. 5th European Conf. Colour in Graphics, Imaging, and Vision (CGIV),* 489–496. Society for Imaging Science / Technology.

ARM. 2013. *Encoder and Decoder Tool for Evaluation of ARM Adaptive Scalable Texture Compression (ASTC).* http://malideveloper.arm.com/develop-for-mali/tools/astc-evaluation-codec/. Accessed Aug. 2013.

Beers, Andrew C., Maneesh Agrawala, and Navin Chaddha. 1996. "Rendering from compressed textures." In *Proc. SIGGRAPH 1996,* 373–378. Computer Graphics Proc., Annual Conf. Series, ACM. ACM Press / ACM SIGGRAPH. doi:10.1145/237170.237276.

Burke, Edmund K., and Graham Kendall, eds. 2014. *Search Methodologies.* 2nd ed. New York: Springer Press. isbn: 978-1-4614-6940-7. doi:10.1007/978-1-4614-6940-7.

Čadík, Martin, Robert Herzog, Karol Mantiuk Rafałand Myszkowski, and Hans-Peter Seidel. 2012. "New Measurements Reveal Weaknesses of Image Quality Metrics in Evaluating Graphics Artifacts." *ACM Trans. Graph.* (New York, USA) 31, no. 6 (November): 147:1–147:10. doi:10.1145/2366145.2366166.

Chandler, Damon M., and Sheila S. Jemami. 2007. "VSNR: A Wavelet-Based Visual Signal-to-Noise Ratio for Natural Images." *Image Processing, IEEE Trans.* 16 (9): 2284–2298. doi:10.1109/TIP.2007.901820.

Delp, Edward J., and O. Robert Mitchell. 1979. “Image Compression Using Block Truncation Coding.” *Communications, IEEE Trans.* 27 (9): 1335–1342. doi:10.1109/TCOM.1979.1094560.

Downs, Julie S., Mandy B. Holbrook, Steve Sheng, and Lorrie Faith Cranor. 2010. “Are your participants gaming the system?: screening mechanical turk workers.” In *Proc. 28th International Conf. Human Factors in Computing Systems (CHI),* 2399–2402. New York: ACM. doi:10.1145/1753326.1753688.

Duda, Jarek. 2014. “Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding.” arXiv: 1311.2540v2 [cs.IT].

Fairchild, Mark D. 2005. *Color Appearance Models.* 2nd ed. West Sussex, England: John Wiley & Sons. isbn: 978-0470012161.

Fenney, Simon. 2003. “Texture compression using low-frequency signal modulation.” In *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graphics Hardware,* 84–91. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Ferwerda, James A., Peter Shirley, Sumanta N. Pattanaik, and Donald P. Greenberg. 1997. “A Model of Visual Masking for Computer Graphics.” In *Proc. SIGGRAPH 97, Annual Conference Series,* 143–152. ACM. doi:10.1145/258734.258818.

Franzen, Rich. 2010. *Kodak Lossless True Color Image Suite.* http://r0k.us/graphics/kodak/index.html. Accessed Aug. 2010.

Glover, Fred. 1989. “Tabu Search — Part I.” *ORSA Journal on Computing* 1 (3): 190–206. doi:10.1287/ijoc.1.3.190.

———. 1990. “Tabu Search — Part II.” *ORSA Journal on Computing* 2 (1): 4–32. doi:10.1287/ijoc.2.1.4.

Griffin, Wesley, and Marc Olano. 2014. “Objective Image Quality Assessment of Texture Compression.” In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games,* 119–126. ACM. doi:10.1145/2556700.2556711.

———. 2015. “Evaluating Texture Compression Masking Effects Using Objective Image Quality Assessment Metrics.” *IEEE Transactions on Visualization and Computer Graphics* 21 (8): 970–979. doi:10.1109/TVCG.2015.2429576.

Hao, Pengwei, and Qingyun Shi. 2000. “Comparative Study of Color Transforms for Image Coding and Derivation of Integer Reversible Color Transform.” In *Proceedings of the 15th International Conference on Pattern Recognition.*

He, Yong, Tim Foley, Natalya Tatarchuck, and Kayvon Fatahalian. 2015. “A System for Rapid, Automatic Shader Level-of-Detail.” *ACM Transactions on Graphics* 34 (6): 187:1–187:12. doi:10.1145/2816795.2818104.

Heidrich, Wolfgang, and Hans-Peter Seidel. 1998. “View-independent environment maps.” In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware,* 39–45. ACM. doi:10.1145/285305.285310.

Hong, Guowei, and Ming R. Luo. 2002. “Perceptually-based color difference for complex images.” *Proc. SPIE* 4421:618–621. doi:10.1117/12.464761.

Horita, Yuukou, Keiji Shibata, and Yoshikazu Kawayoke. 2011. *MICT Image Quality Evaluation Database.* http://mict.eng.u-toyama.ac.jp/mictdb.html. Accessed Apr. 2011.

Howard, Paul G., and Jeffrey Scott Vitter. 1994. “Arithmetic coding for data compression.” *Proc. IEEE* 82 (6): 857–865. doi:10.1109/5.286189.

Huffman, David A. 1952. “A Method for the Construction of Minimum-Redundancy Codes.” *Proc. of the IRE* 40 (9): 1098–1101. doi:10.1109/JRPROC.1952.273898.

Inada, Tetsugo, and Michael D. McCool. 2006. “Compressed lossless texture representation and caching.” In *Proc. ACM SIGGRAPH/EUROGRAPHICS Symp. Graphics Hardware,* edited by Marc Olano and Philipp Slussalek, 111–120. New York: ACM.

Iorcha, Konstantine I., Krishna S. Nayak, and Zhou Hong. 1999. System and Method for Fixed-rate Block-based Image Compression with Inferred Pixel Values 5956431 (US), filed 1999.

ITU-R. 2012. *Recommendation ITU-R BT.500-13: Methodology for the subjective assessment of the quality of television pictures.* http://www.itu.int/rec/R-REC-BT.500-13-201201-I, January.

Joe, Stephen, and Frances Y. Kuo. 2003. “Remark on algorithm 659: Implementing Sobol’s quasirandom sequence generator.” *ACM Trans. Math. Softw.* 29 (1): 49–57. doi:10.1145/641876.641879.

Johnson, Garrett M., and Mark D. Fairchild. 2001. “Darwinism of Color Image Difference Models.” In *Proc. 9th Color Imaging Conf.* 108–112. Society for Imaging Science / Technology.

Khronos Group. 2013. *Extension #118: KHR_texture_compression_astc_hdr.* http://khronos.org/registry/gles/extensions/KHR/texture_compression_astc_hdr.txt. Accessed Oct. 2013.

Kittur, Aniket, Ed H. Chi, and Bongwon Suh. 2008. “Crowdsourcing user studies with Mechanical Turk.” In *Proc. 26th Conf. Human Factors in Computing Systems (CHI),* 453–456. New York: ACM. doi:10.1145/1357054.1357127.

Larson, Eric C., and Damon M. Chandler. 2010. “Most apparent distortion: full-reference image quality assessment and the role of strategy.” *J. Electron. Imaging* 19 (011006). doi:10.1117/1.3267105.

Le Callet, Patrick, and Florent Autrusseau. 2005. *Subjective quality assessment IRCCyN / IVC database*. http://www.irccyn.ec-nantes.fr/ivcdb/. Accessed Apr. 2011.

Lindstrom, Peter, and Greg Turk. 2000. "Image-drive Simplification." *ACM Trans. Graph.* 19, no. 3 (July): 204–241. doi:10.1145/353981.353995.

Martin, G. 1979. "Range encoding: An algorithm for removing redundancy from a digitised message." In *Proc. Video and Data Recording Conf. (Southampton, UK, July 24–27).*

Mitchell, Kenny. 2006. "Next Generation Game Development." In *Proc. 3rd European Conf. on Visual Media and Production (CVMP).* Invited Talk. London, UK: IET.

———. 2015. personal communication.

NVIDIA. 2013. *NVIDIA Texture Tools*. http://code.google.com/p/nvidia-texture-tools. Accessed Aug. 2013.

Olano, Marc, and Dan Baker. 2010. "LEAN Mapping." In *Proc. of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games,* 181–188. ACM. doi:10.1145/1730804.1730834.

Olano, Marc, Dan Baker, Wesley Griffin, and Joshua Barczak. 2011. "Variable Bit Rate GPU Texture Compression." *Computer Graphics Forum* 30 (4): 1299–1308. doi:10.1111/j.1467-8659.2011.01989.x.

Olano, Marc, Bob Kuehne, and Maryann Simmons. 2003. "Automatic Shader Level of Detail." In *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graphics Hardware,* 7–14. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

OpenGL ARB. 2009. *Extension #77: ARB_texture_compression_bptc*. http://www.opengl.org/registry/specs/ARB/texture_compression_bptc.txt. Accessed Aug. 2010.

Pedersen, Marius, and Jon Hardeberg. 2009. “A New Spatial Hue Angle Metric for Perceptual Image Difference.” In *Computational Color Imaging,* edited by Alain Trémeau, Raimondo Schettini, and Shoji Tominaga, 5646:81–90. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. doi:10.1007/978-3-642-03265-3_9.

Pellacini, Fabio. 2005. “User-configurable automatic shader simplification.” *ACM Trans. Graphics* 24 (3): 445–452. doi:10.1145/1073204.1073212.

Ponomarenko, Nikolay, Lina Jin, Oleg Ieremeiev, Vladimir Lukin, Karen Egiazarian, Jaakko Astola, Benoit Vozel, et al. 2015. “Image database TID2013: Peculiarities, results and perspectives.” *Signal Processing: Image Communication* 30:57–77. doi:10.1016/j.image.2014.10.009.

Ponomarenko, Nikolay, Vladimir Lukin, Alexander Zelensky, Karen Egiazarian, Jaakko Astola, Marko Carli, and Federica Battisti. 2009. “TID2008 - A database for evaluation of full-reference visual quality assessment metrics.” *Advances of Modern Radioelectronics* 10:30–45. http://www.ponomarenko.info/tid2008.htm.

Rouse, David M., and Sheila S. Hemami. 2008. “Understanding and simplifying the structural similarity metric.” In *Proc. 15th IEEE Int’l Conf. Image Processing (ICIP),* 1188–1191. IEEE. doi:10.1109/ICIP.2008.4711973.

Russell, Stuart, and Peter Norvig. 2003. *Artificial Intelligence, A Modern Approach.* 2nd. Upper Saddle River, New Jersey, US: Pearson Education. isbn: 0-13-790395-2.

Shapiro, Jerome M. 1993. “Embedded image coding using zerotrees of wavelet coefficients.” *Signal Processing, IEEE Trans.* 41 (12): 3445–3462. doi:10.1109/78.258085.

Sheikh, Hamid Rahim, and Alan C Bovik. 2006. “Image information and visual quality.” *Image Processing, IEEE Trans.* 15 (2): 430–444. doi:10.1109/TIP.2005.859378.

Sheikh, Hamid Rahim, Muhammad Farooq Sabir, and Alan C Bovik. 2006. “A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms.” *Image Processing, IEEE Trans.* 15 (11): 3440–3451. doi:10.1109/TIP.2006.881959.

Sheikh, Hamid Rahim, Z. Wang, L. Cormack, and A.C. Bovik. 2010. *LIVE Image Quality Assessment Database Release 2*. http://live.ece.utexas.edu/research/quality. Accessed Sep. 2010.

Simoncelli, Eero P. 2005. “Statistical Modeling of Photographic Images.” In *Handbook of Image and Video Processing,* edited by Alan C Bovik, 431–441. Elsevier/Academic Press.

Sitthi-Amorn, Pitchaya, Nicholas Modly, Westley Weimer, and Jason Lawrence. 2011. “Genetic programming for shader simplification.” *ACM Trans. Graphics (Proc. SIGGRAPH Asia’11)* 30 (6): 152:1–152:12. doi:10.1145/2070781.2024186.

Skodras, Athanassios, Charilaos Christopoulos, and Touradj Ebrahimi. 2001. “The JPEG 2000 still image compression standard.” *IEEE Signal Processing Magazine* 18 (5): 36–58. doi:10.1109/79.952804.

Ström, Jacob, and Tomas Akenine-Möller. 2004. “PACKMAN: texture compression for mobile phones.” In *ACM SIGGRAPH 2004 Sketches,* 66–66. ACM. doi:10.1145/1186223.1186306.

———. 2005. “iPACKMAN: high-quality, low-complexity texture compression for mobile phones.” In *Proc. 20th ACM SIGGRAPH/EUROGRAPHICS Symp. Graphics Hardware,* 63–70. New York: ACM. doi:10.1145/1071866.1071877.

Ström, Jacob, and Martin Pettersson. 2007. “ETC2: texture compression using invalid combinations.” In *Proc. 22nd ACM SIGGRAPH/EUROGRAPHICS Symp. Graphics Hardware,* 49–54. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Van Waveren, J.M.P., and Ignacio Castaño. 2007. *Real-Time YCoCg-DXT Compression.* Technical report. iD Software, September. http://www.nvidia.com/object/real-time-ycocg-dxt-compression.html.

Video Quality Experts Group. 2003. *Final Report From the Video Quality Experts Group on the Validation of Objective Models of Video Quality Assessment, Phase II.* http://www.vqeg.org.

Wallace, Gregory K. 1991. "The JPEG still picture compression standard." *Comm. ACM* 34 (4): 30–44. doi:10.1145/103085.103089.

Wang, Zhou, and Alan C Bovik. 2006. *Modern Image Quality Assessment.* San Rafael, CA, US: Morgan & Claypool Publishers.

Wang, Zhou, Alan C Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. 2004. "Image quality assessment: from error visibility to structural similarity." *Image Processing, IEEE Trans.* 13 (4): 600–612. doi:10.1109/TIP.2003.819861.

Wang, Zhou, and Qiang Li. 2011. "IW-SSIM: Information Content Weighted Structural Similarity Index for Image Quality Assessment." *Image Processing, IEEE Trans.* 20 (5): 1186–1198.

Wang, Zhou, and Xinli Shang. 2006. "Spatial Pooling Strategies for Perceptual Image Quality Assessment." In *Proc. 2006 Int'l. Conf. Image Processing (ICIP).* IEEE. doi:10.1109/ICIP.2006.313136.

Wang, Zhou, Eero P. Simoncelli, and Alan C. Bovik. 2003. "Multi-scale structural similarity for image quality assessment." In *IEEE Asilomar Conf. Signals, Systems, and Computers.* November.

Williams, Lance. 1983. "Pyramidal parametrics." In *Computer Graphics (Proc. of SIGGRAPH 83),* 1–11. New York: ACM. doi:10.1145/800059.801126.

Witten, Ian H., Radford M. Neal, and John G. Cleary. 1987. “Arithmetic coding for data compression.” *Comm. ACM* 30 (6): 520–540. doi:10.1145/214762.214771.

Zhang, Xuemei, and Brian A. Wandell. 1998. “Color image fidelity metrics evaluated using image distortion maps.” *Signal Processing* 70 (3): 201–214. doi:10.1016/S0165-1684(98)00125-X.