RTI Compression for Mobile Devices

Christopher Schuster, Bipeng Zhang, Rajan Vaish, Paulo Gomes, Jacob Thomas, James Davis
University of California, Santa Cruz
{cschuste, bizhang, rvaish, pfontain, jthomas7, davisje}@ucsc.edu

Abstract—**Currently, Reflectance Transformation Imaging (RTI) technology is restricted to desktop and high performance computing devices. In recent years, mobile devices and tablets have become ubiquitous due to their increased performance. However, the size of RTI files (≈100 MB) limits the range of portable devices capable of displaying RTI files. In this paper, we explore different compression techniques, and develop an RTI viewer prototype for both Android and iOS devices. Experiments with JPEG, JPEG2000, lossless compression show the resulting error for compression ratios of 30:1 is comparable to the error of traditional two-dimensional images. For higher compression rates, we present a novel $\alpha\beta$-JPEG algorithm which compresses color and reflectance information individually.**

I. Introduction

Reflectance Transformation Imaging (RTI) describes a file format and a set of methods that extend traditional two-dimentional photography with the ability to dynamically manipulate the lighting condition when displaying an image file.

RTI images are typically created by taking 15 to 60 pictures of the object of interest with a standard digital single-lens reflex camera while varying the position of the light along a notional hemisphere. A black sphere placed adjacent to the object has a highlight cast on its surface by the flash in each instance: a small, bright speck that allows the software to calculate the direction of light for each shot. This data allows the RTI builder to construct a composite image in which each pixel is stored as a set of lighting parameters instead of a static color. This enables the viewer to dynamically relight the image and to enhance 3D features of the surface custom light-dependent filters.

Relightable images expose subtle surface indentations that are hard to distinguish without computer enhancemen and are therefore particularly useful for the study of ancient artifacts and archaeological materials like the Dead Sea Scrolls [1] or stelae from the Bronze Age [4].

The rise in portable device adoption have raised particular interest in their potential application in interactive museum exhibits and for personal uses. However, current tools available at Cultural Heritage Imaging offer no web or mobile device support. The prime issue for RTI on mobile devices is the conflict between bigger and more detailed data sets (100 MB and larger) [5] and the limited memory capacity and network bandwidth. Compression mitigates this problem. However, previous efforts on compression of relightable images did not address the specific challenges on portable devices, particularly for interactive real-time rendering of compressed images.

In this paper, we introduce and evaluate four compression algorithms which are all based on standard image compression techniques and thereby have existing implementations on mobile devices with fast decompression. Our experiments with archaeological RTI data showed that simple independent compression of all light coefficients can be optimized by restricting the light calculation to the luminosity and averaging the color components across all light positions. For high compression rates, this algorithm yields better image quality without adding significant computational overhead for decompression.

Additionally, we present a working prototype implementation for RTI rendering on iOS as well as Android, and outline how the proposed compression algorithms could be integrated.

Overall, our paper's contribution is an evaluation of different RTI compression algorithms, a novel algorithm that outperforms JPEG for high compression rates and RTI viewer implementations for both iOS and Android.

II. Related Work

Malzbender et. al. [13] proposed the first RTI implementation, Polynomial Texture Mapping (PTM) which models the light calculation with a second-order biquadratic polynomial. Other approaches include 4D Light Fields [12], [6] and Dense Photometric Stereo (PST) [28]. In general, these techniques allow more accurate lighting calculations but also require a more complex capturing process or specialized equipment.

Variations on the original PTM method have been explored, including a series of alternative basis including Spherical Harmonics [24], Zonal Harmonics [25], eigen hemispherical harmonics [9], spherical radial basis function [11] and Principal Component Analysis (PCA) [16]. A simple JPEG-Compression for PTM was presented by Motta et. al. [17].

Alternatively, lighting can be computed with a bidirectional texture function (BTF) [3], [14] which requires a different capturing process than RTI. See Haindl et. al. [7] for a survey on different BTF compression methods. Most noteworthy are 3D wavelet compression algorithms. Rodler et. al. [21] originally proposed 3D wavelet compression for fast access to large volume multi-dimensional data. Additionally, wavelets have been used to compress spherical harmonics basis [10], Spherical Radial Basis Functions [11], and Singular Value Decomposition [2].

There is ongoing research on solutions for streaming relightable images. Ramanathan et. al. [20] propose a rate-distortion optimized approach for light fields using packet scheduling and Schwartz et. al. [22] proposed a remote rendering system for BTF. In order to achieve real time interaction, a low-latency rendering system needs to be used, whether on a single server [23] or in the cloud [15].

III. Methods and Implementation

The two most successful transformations used in image compression are Discrete Cosine Transform (DCT), which is used by the JPEG algorithm, and Discrete Wavelet Transform (DWT), which is used by JPEG 2000. We applied both procedures to RTI images of archelogical objects which were provided by the Cultural Heritage Imaging (CHI). Based on our observation, we propose and evaluate an optimization to the JPEG algorithm called $\alpha\beta - JPEG$ using the same RTI data sets. As an additional comparator, we also tested the generic lossloss Lempel-Ziv-Welch (LZW) compression on the provided RTI files. Finally, we implemented RTI viewers for both Android and iOS to demonstrate the feasibility of RTI rendering on mobile devices.

A. JPEG2000 and JPEG Compression

For comparing and finding a better compression method, we applied both JPEG and JPEG2000 to each coefficient and color plane.

Polynomial Texture Maps (PTMs) as introduced by Malzbender et al.[13] combine the information of the pixels in corresponding locations (u, v) with intensity dependencies $L(u, v)$ on the light direction (l_u, l_v) by using a second-order biquadratic polynomial as model as shown in Equation 1. We store each pixel (u, v) with the six polynomial coefficients, in addition to its RGB values. Then, we read the six coefficients planes and the three color planes from the original PTM files to compress each plane with JPEG2000 and JPEG method respectively. For the decompression process, each plane is encoded separately and then the PTM file is generated with the corresponding file header.

The implementation was done using MATLAB's `imwrite` function to compress each plane with JPEG2000 and JPEG method respectively. The compressed data can be decoded with the function `imread`. This method is similar to `PTM_FORMAT_JPEG_LRGB` in the PTM File format [17].

$$\begin{aligned} R(u,v) &= L(u,v)R_n(u,v) \\ G(u,v) &= L(u,v)G_n(u,v) \\ B(u,v) &= L(u,v)B_n(u,v) \end{aligned} \tag{1}$$

B. $\alpha\beta$-JPEG Compression

RTI exposes surface structure and geometry in the form of shadowing, which typically involves a change in luminosity and rarely affects the hue of the surface colors. In order to exploit this observation for compression, the image can be encoded with a view and lighting-dependent second order spherical harmonics (SH) equation with 9 coefficients for the luminosity and two additional channels for color information which is averaged to be lighting-independent. Using this encoding, the Discrete Cosine Transform can be applied individually for each channel. The α parameter controls the compression rate of the reflectance coefficients while β controls the compression rate of color information.

A first prototype of the $\alpha\beta$-compression was implemented in Python using numPy [18] to convert RTI files with 9 SH coefficients to compressed CRTI files and vice versa. The compression is shown in Fig. 1 and involves combining all three color channels to get RGB-triples per pixel grouped by coefficient. For 2-order spherical harmonics, this results in 9 RGB images. Each of these images was then converted to the YCbCr color space using the Python Imaging Library (PIL [19]) yielding a triple for each pixel with Y representing the luminosity and Cb and Cr representing the blue and red color channels. The color channels were then averaged across all SH coefficients to yield a single gray-scale image for each of the two color channels. The resulting images were then compressed individually using a Discrete Cosine Transform with α and β controlling the compression rate and packed along size information in a single file. The decompression process works accordingly but in reverse.

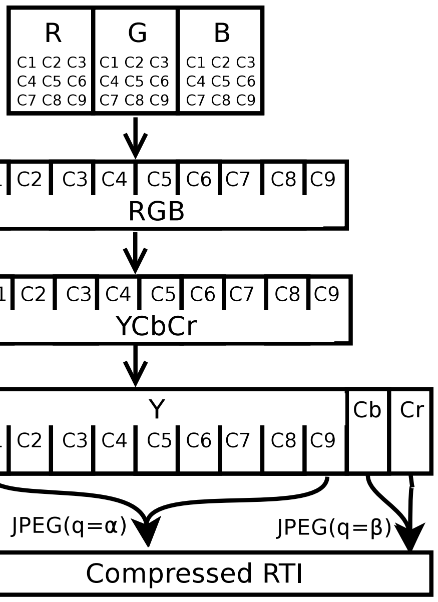


Fig. 1. The compression includes a conversion from RGB to YCbCr color space and a separate compression using α to control the JPEG compression quality of the light-dependent luminosity (Y) and β to control the compression quality of the light-independent color components (Cb and Cr).

The parameters α and β represent a trade-off between compression rate, image quality and geometric detail. Depending on the concrete relightable image, different choices for α and β work better for different use cases. For example, the chromatic information in a picture of an ancient oxidized copper coin might be less interesting to an archelogist than its (color-independent) surface structure which suggests a high α and low β value. In contrast, the image quality of a relightable painting would suffer from too aggressive compression of the color component which indicates that high β values are necessary to maintain image quality.

C. LZW compression

The method we used for lossless compression of RTI images was Lempel-Ziv-Welch (LZW) compression. LZW compression creates a dictionary of long substrings in the file and makes those long substrings available for encoding.

LZW is a lossless compression method and therefore loses no information from the original and adds no new noise between the encoding and decoding stages. We saved the RTI images in the Tagged Image File Format (TIFF) which has built in compatibility with LZW compression. TIFF is a flexible file format that can hold both lossless and lossy images.

In order to implement and test the compression of the RTI images, we loaded the files into MATLAB and encoded them with the `imwrite` function which has built in support of LZW compression for TIFF files. The compression rate was measured using the resulting file size. LZW is a lossless compression method so in contrast to the lossy compression methods like $\alpha\beta$-JPEG and JPEG2000, it was not necessary to evalutate the error.

D. Mobile Devices

We created a RTI Android viewer. The code to load and render the images to an Open GL ES view was developed in C++. The touch interaction was written in Java and for the interface between the two we used Java Native Interface. We also created an RTI Viewer for iOS based on an existing JavaScript viewer. The web content is presented through a *web view*. We used the CORDOVA SDK 2.9 to generate an XCode 5.0.2 project. This SDK allowed us to include all the web and image files in the app itself. When the app is launched, a local http server is started which contains the web content. To load a relightable file, we make an AJAX request to the local server and load it as a Binary Large Object (blob). This blob is then loaded as a binary sequence and rendered.

In both the Android and iOS versions, the user can touch the surface of the device to move the origin of the light. The view is updated in real-time which enables intuitive study and exploration of surface details of the relighted object. We outline how it would be possible in the future to run compression on the mobile devices. To integrate JPEG 2000 compression method with mobile RTI viewers, use the jj2000 library[1] on Android and the Open-JPEG project[2] on iOS. Kivy[3] would make it possible to run our Python implementation of the $\alpha\beta$-JPEG Compression on both iOS and Android.

IV. Experiment

A. Compression algorithms

For the experiment, we consider both lossless model and lossy compression algorithms. The lossless LZW compression was evalutated by comparing the file size of the original images and compressed files for eight different RTI files.

For evaluating lossy compression, we applied both JPEG and JPEG2000 method to each coefficient and RGB plane of three different PTM images using the method described above and measured the compression ratio, Peak Signal to Noise Ratio (PSNR) and Root Mean Squared Error (RMSE) between the original PTM file and the compressed and decompressed PTM file.

[1]https://code.google.com/p/jj2000/
[2]http://www.openjpeg.org/
[3]http://kivy.org/

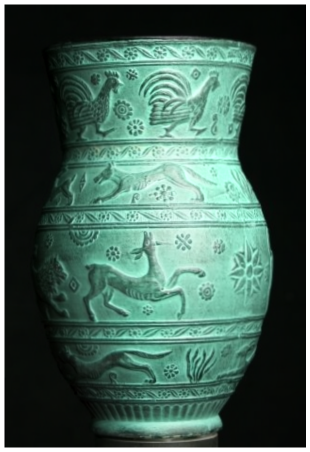
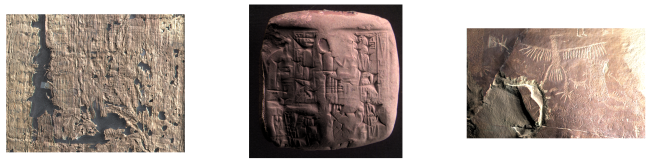


Fig. 2. The relightable image `vase.rti` was used to evaluate both lossy and lossless compression techniques.



a) `Papyrus` b) `Tablet` c) `Bird`

Fig. 3. The original images shown are based on three uncompressed PTM files with varying lighting direction.

The two methods use different parameters controlling the final compression result. For JPEG2000 compression, 'Compression Ratio' was used directly while the compression ratio of JPEG compression depends on the 'Quality' parameter.

Additionally, $\alpha\beta$-JPEG compression was evaluated by compressing the RTI file `vase.rti` (see Fig. 2) using different values for α and β and decompressing the compressed file. Both the original RTI file and the decompressed RTI file were then rendered with different light positions. The difference between the rendered images was measured in terms of the Mean Structural Similarity Index (SSIM) [27] using a Python MSSIM library [26], and in terms of the Peak Signal to Noise Ratio (PSNR) and Root Mean Squared Error (RMSE) using ImageMagick [8].

B. Mobile Devices

The web version developed by Wei Lao and the Android viewer were compared by loading 3 RTI files one at a time. We analyzed file loading time and lighting recalculation time in response to touch and mouse-move events.

The test device for the Android version was a Galaxy Nexus with Android 4.3, a dual-core 1.2 GHz (ARM) Cortex-A9 processor; PowerVR SGX540 GPU; 16 GB storage; 1 GB RAM and a 1280 x 720 pixel display. The JavaScript Viewer was tested on a PC with Mozilla Firefox 27.0.1 on Microsoft Windows 7 Professional SP1, an Intel Core i3 M 330 @ 2.13GHz dual core processor; 1GB ATI Mobility Radeon 5650 GPU; 8 GB RAM and a 1920 x 1080 pixels display.

Table I
COMPRESSION RATIOS USING LZW COMPRESSION.

File name	Compression Ratio
`Boat.rti`	1.759
`vase.rti`	1.053
`cuniform.rti`	1.437
`_Boat2.rti`	2.345
`Wallet.rti`	1.673
`Antiques.rti`	1.298
`Clay.rti`	1.530
`watch.rti`	1.869

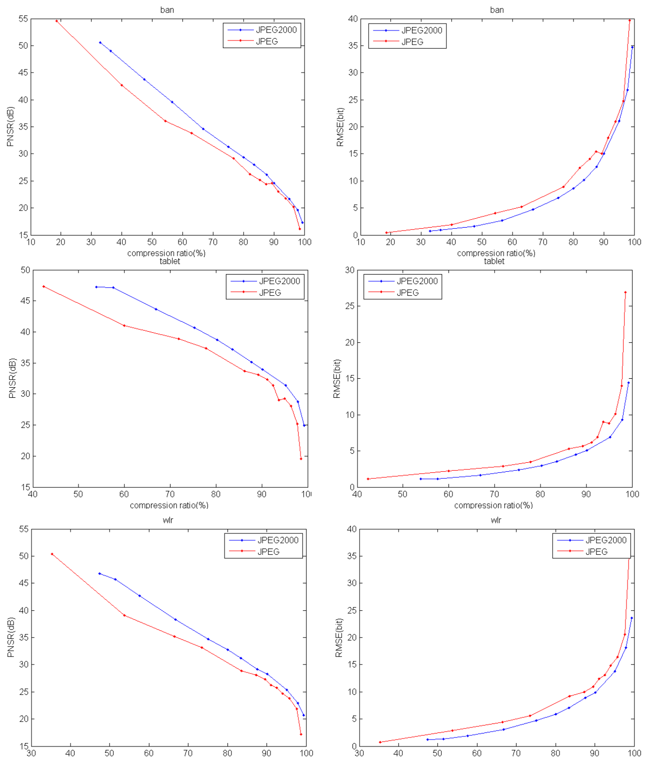


Fig. 4. For the original images in Fig. 3, the relationship between PSNR, RMSE and Compression Ratio respectively with the two methods are shown.

In order to test the iOS viewer, we used XCode 4 iOS Simulator on iPhone Retina mode. It ran on a OS X laptop with a 2.8 GHz Intel Core 2 Duo processor, 4 GB RAM, and a NVIDIA GeForce 9400M 256 MB graphics card. We tested the system with `vase.rti` (see Fig. 2) and `tombstone512.ptm`.

V. RESULTS

A. Compression algorithms

1) Lempel-Ziv-Welch(LZW): The LZW compression scheme is lossless, therefore it perfectly reconstructs the original images. Table I shows the size of eight original images and the corresponding compressed images. The resulting compression rate depends on the actual file but does not exceed 3:1 which limits the effect of compression.

2) JPEG-2000: The results of comparing JPEG and JPEG-2000 for each of the three original PTM images a), b) and c) in Fig. 3 is shown Fig. 4.

The results above indicate that for all three examples both PTM-JPEG and PTM-JPEG2000 achieve similar PSNR or RMSE than traditional two-dimensional JPEG compression for a given compression ratio. JPEG2000 method outperforms JPEG method when the compressed images have the same compression ratio. Also, when the compressed images have the same RMSE or PSNR, JPEG2000 method has greater compression ratio than JPEG method does.

In addition to the measurement results, a visual comparison between JPEG and JPEG2000 method for very high compression ratios ($\approx$ 30) is given in Fig. 5. We chose the light direction when the pixel point (350,500) has the largest luminance, which ensures the images look not too dim to show details. For the JPEG compression, the 'block effect' of the Discreate Cosine Transform is visible while the Wavelet based compression JPEG2000 causes the rendered images to appear blurry. However, the image quality is still comparable to traditional compression of two-dimensional images, therefore both of the two methods could be capable for compressing PTM files. We know the size of images encoding by JPEG2000 is smaller than those encoding by JPEG. Therefore, under the similar compression ratio, JPEG2000 method achieves a better

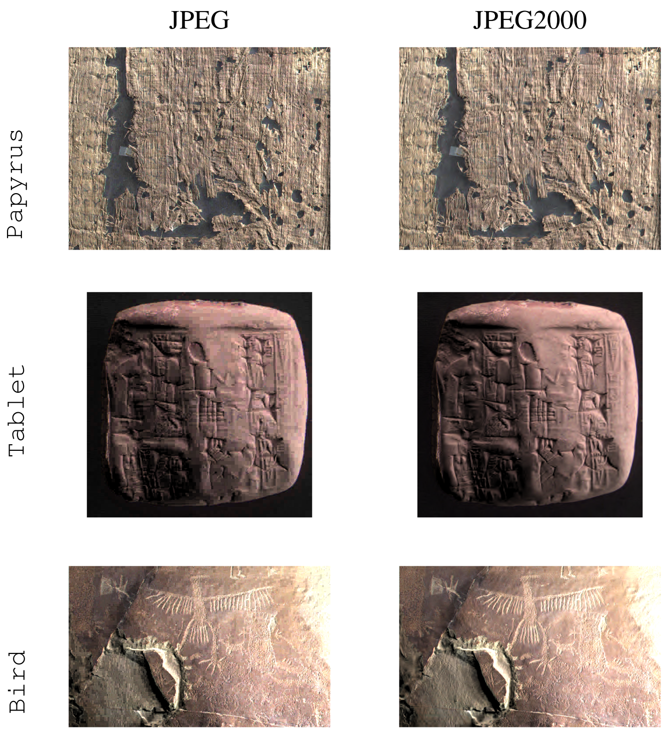


Fig. 5. For each example, the corresponding compressed images resulting from JPEG and JPEG2000 are shown with a compression rate of 96%-98%. The decompressed images from the JPEG method exhibit a 'block effect'. This corresponds to the increasing PNSR and RMSE under high compression rates. The decompressed images resulting from JPEG2000 compression do not have blocks but appear more blurry instead.

Table II
JAVASCRIPT AND ANDROID LOADING AND RELIGHTING TIMES

Name	Size	Resolution	Loading		Relighting	
			JS	Android	JS	Android
`coin`	6239kb	600x400	$\tilde{1}$s	4s	1s	$t < 0.5s$
`vase`	3966kb	320x470	1s	3s	1s	$t < 0.5s$
`watch`	9113kb	720x480	2s	6s	1s	$t < 0.5s$

quality than JPEG compression.

3) $\alpha\beta$-JPEG: The results of the $\alpha\beta$-JPEG compression are shown in Fig. 6. The heatmap indicates that both α and β determine the resulting error. Finding the right values for α and β depends on the given relightable image and is potential topic for future work. The direct comparison between a standard JPEG compression for each of the 9 planes and the $\alpha\beta$-JPEG compression shows that an error which is independent of the compression ratio. This error is caused by averaging the color information over all coefficients which is done by the $\alpha\beta$-JPEG compression and causes information loss independent of the actual parameters α and β. This error causes $\alpha\beta$-JPEG to yield worse results than JPEG for relatively low compression ratios. However, for high compression ratios, $\alpha\beta$-JPEG outperforms standard JPEG in SSIM, PSNR and RMSE. SSIM values close to 1.0 indicate a high structural similarity which is generally better for evaluating perceived image quality than PSNR or RMSE [27]. The measured SSIM values show that $\alpha\beta$-JPEG is preferable to JPEG for compression ratios of 50 and above.

B. Mobile Devices

We were able to render relightable images encoded in both RTI and PTM formats, on iOS and Android environments. Fig. 7 shows the data set image `vase.rti` in RTI format rendered on the iOS Simulator, while Fig. 7 shows the same image (different light direction) rendered in the Android environment.

The results regarding the Android/Web comparison are summarized in Table II. The light recalculation time was significantly shorter in the Android device, however the initial image loading was faster in the Webviewer. On average the Web Viewer took $\sim$ 1 second to recalculate the rendered image under different light conditions, while in Android the delay was hardly noticeable. We believe this is due to the use of OpenGL ES 2.0 on the Android version leading to graphics hardware acceleration. In order to extend the comparison to the iOS version, we would need to test it in an actual device. Finally, the drawn images on iOS, Android, and Web Viewer did not present noticeable artifacts in the tested resolutions.

VI. CONCLUSION AND FUTURE WORK

This paper presented four different compression algorithms which are based on standard image compression and therefore can be implemented sufficiently fast for real-time rendering on current mobile devices.

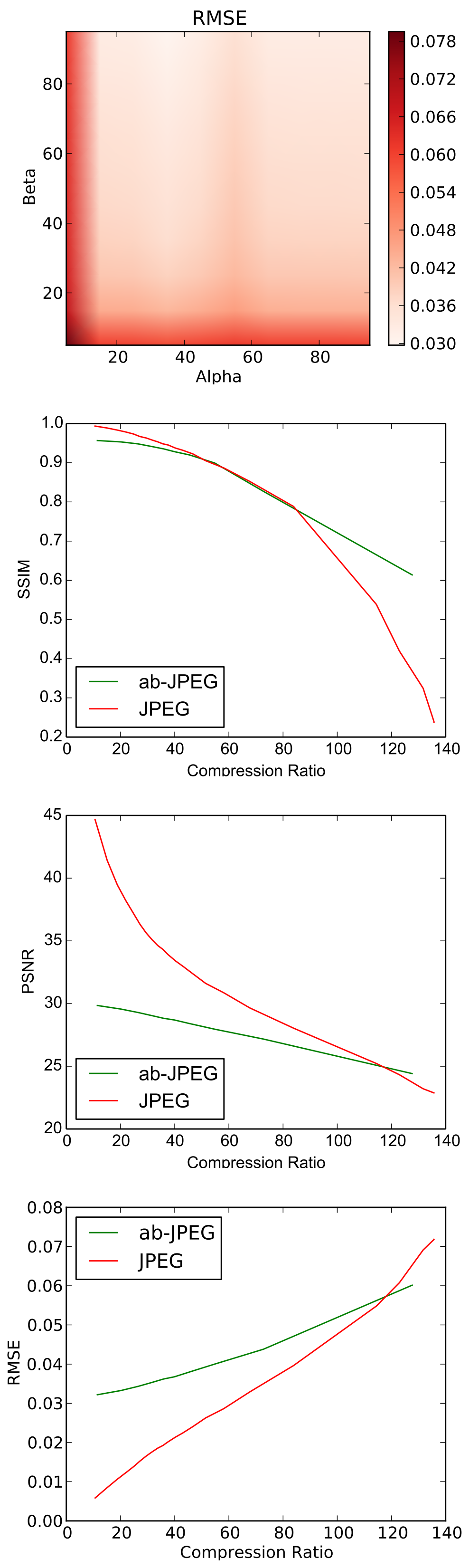


Fig. 6. The heatmap shows the resulting Root Mean Squared Error (RMSE) for different combinations of α and β values. The plots show the Structural Similarity Index (SSIM), Peak Signal to Noise Ratio (PSNR) and RMSE of $\alpha\beta$-JPEG and standard JPEG for varying compression ratios.

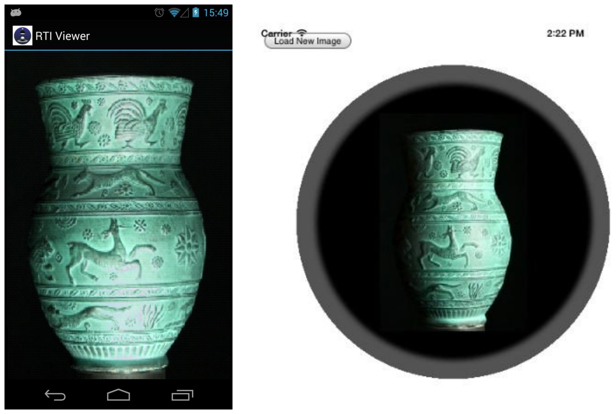


Fig. 7. `vase.rti` sample image rendered on Android 4.3 (left) and on iOS Simulator (right).

The results shown in the previous section indicate that simple JPEG compression of all coefficients can be optimized by treating the color components as light-independent which results in a better image quality as measured by PSNR for high compression rates.

By using the standard image compression implementations which are well-supported for mobile devices, we avoid potential computational overhead for decompression. Additionally, the implementation of RTI viewers for both iOS and Android show that the approach can be practically used on current mobile devices. The implementation of RTI compression for mobile devices is especially useful for museum exhibits of archelogical objects which often involve detailed and large datasets that also benefit from the intuitive touch interface.

In order to support large high-definition datasets which do not fit into main memory of these devices, tiling, streaming or hybrid solutions are necessary. Combining the techniques for compressing RTI data presented in this paper with streaming or tiling is a promising area of future research.

References

[1] Moshe Caine and Michael Magen. Pixels and Parchment: The Application of RTI and Infrared Imaging to the Dead Sea Scrolls. *EVA London*, 2011.

[2] I-Cheng Chang, Hsin-Yo Chou, and Chung-Lin Huang. Texture compression and relighting of 3-d color objects using singular value decomposition. *Journal of Information Science & Engineering*, 26(2), 2010.

[3] Kristin J Dana, Bram Van Ginneken, Shree K Nayar, and Jan J Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics (TOG)*, 18(1):1–34, 1999.

[4] Marta Diaz-Guardamino and David Wheatley. Rock art and digital technologies: the application of Reflectance Transformation Imaging (RTI) and 3D laser scanning to the study of Late Bronze Age Iberian stelae. *MENGA. Journal of Andalusian Prehistory*, 4, 2014.

[5] G. Earl, P. J. Basford, A. S. Bischoff, A. Bowman, C. Crowther, J. Dahl, M. Hodgson, K. Martinez, L. Isaksen, H. Pagi, K. E. Piquette, and E. Kotoula. Reflectance transformation imaging systems for ancient documentary artefacts. In Jonathan P. Bowen, Stuart Dunn, and Kia Ng, editors, *EVA London 2011: Electronic Visualisation and the Arts*. BCS, 2011.

[6] Todor Georgiev, Ke Colin Zheng, Brian Curless, David Salesin, Shree Nayar, and Chintan Intwala. Spatio-angular resolution tradeoffs in integral photography. *Rendering Techniques*, 2006:263–272, 2006.

[7] Michal Haindl and Jiri Filip. Extreme compression and modeling of bidirectional texture function. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(10):1859–1865, 2007.

[8] ImageMagick. Imagemagick website. Accessed 2013-05-07, 2013.

[9] P-M Lam, C-S Leung, and T-T Wong. Noise-resistant hemispherical basis for image-based relighting. *Image Processing, IET*, 6(1):72–86, 2012.

[10] Ping-Man Lam, Chi-Sing Leung, and Tien-Tsin Wong. Noise-resistant fitting for spherical harmonics. *Visualization and Computer Graphics, IEEE Transactions on*, 12(2):254–265, 2006.

[11] Chi-Sing Leung, Tien-Tsin Wong, Ping-Man Lam, and Kwok-Hung Choy. An rbf-based compression method for image-based relighting. *Image Processing, IEEE Transactions on*, 15(4):1031–1041, 2006.

[12] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM, 1996.

[13] Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 519–528. ACM, 2001.

[14] David K McAllister, Anselmo Lastra, and Wolfgang Heidrich. Efficient rendering of spatial bi-directional reflectance distribution functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 79–88. Eurographics Association, 2002.

[15] Dan Miao, Wenwu Zhu, and Chang Wen Chen. Low-delay cloud-based rendering of free viewpoint video for mobile devices. In *SPIE Optical Engineering+ Applications*, pages 88560C–88560C. International Society for Optics and Photonics, 2013.

[16] Gero Miiller, Jan Meseth, and Reinhard Klein. Compression and real-time rendering of measured btfs using local pca. In *Vision, Modeling, and Visualization: Proceedings*, page 271. AKA, 2003.

[17] Giovanni Motta and Marcelo J Weinberger. Compression of polynomial texture maps. Technical report, Technical Report, Information Theory Research Group, HP Laboratories Palo Alto, HPL-2000-143 (R2), 2001.

[18] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[19] PythonWare. Python Imaging Library (PIL).

[20] Prashant Ramanathan, Mark Kalman, and Bernd Girod. Rate-distortion optimized interactive light field streaming. *Multimedia, IEEE Transactions on*, 9(4):813–825, 2007.

[21] Flemming Friche Rodler. Wavelet based 3d compression with fast random access for very large volume data. In *Computer Graphics and Applications, 1999. Proceedings. Seventh Pacific Conference on*, pages 108–117. IEEE, 1999.

[22] Christopher Schwartz, Roland Ruiters, and Reinhard Klein. Level-of-detail streaming and rendering using bidirectional sparse virtual texture functions. In *Computer Graphics Forum*, volume 32, pages 345–354. Wiley Online Library, 2013.

[23] Shu Shi. *A low latency remote rendering system for interactive mobile graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

[24] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536. ACM, 2002.

[25] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, deformable precomputed radiance transfer. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1216–1224. ACM, 2005.

[26] Antoine Vacavant, Adelaide Albouy-Kissi, Pierre-Yves Menguy, and Justin Solomon. Fast smoothed shock filtering. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 182–185. IEEE, 2012.

[27] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[28] Tai-Pang Wu, Kam-Lun Tang, Chi-Keung Tang, and Tien-Tsin Wong. Dense photometric stereo: A markov random field approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(11):1830–1846, 2006.