Optics and Lasers in Engineering 51 (2013) 620–625

Contents lists available at SciVerse ScienceDirect

Optics and Lasers in Engineering

journal homepage: www.elsevier.com/locate/optlaseng

3D range geometry video compression with the H.264 codec

Nikolaus Karpinsky, Song Zhang *

Mechanical Engineering Department, Iowa State University, Ames, IA 50011, United States

ARTICLE INFO

Article history:
Received 3 October 2012
Received in revised form
5 December 2012
Accepted 31 December 2012
Available online 30 January 2013

Keywords:
3D range data compression
3D video processing
Graphics processing unit (GPU)
Fringe analysis
H.264

ABSTRACT

Advances in three-dimensional (3D) scanning have enabled the real-time capture of high-resolution 3D videos. With these advances brings the challenge of streaming and storing 3D videos in a manner that can be quickly and effectively used. This research addresses this challenge by generalizing the Holovideo technique to video codecs that use the YUV color space such as the H.264 codec. With the H.264 codec, we have achieved a compression ratio of over 6086:1 (Holovideo to OBJ) with a reasonably high quality; utilizing an NVIDIA GeForce 9400 m GPU, we have realized 17 frames per second encoding, and 28 frames per second decoding speed, making it a viable solution for real-time 3D video compression.

1. Introduction

Advances in 3D scanning have enabled the real-time capture of high-resolution 3D video. These advances have brought forth the challenge of streaming and storing these high-resolution 3D video frames in a format that can be quickly and efficiently used. Classical approaches in 3D geometry compression compress the 3D coordinates and their attributes such as normals, UV coordinates, etc., in a model format such as OBJ, PLY, STL. Although these formats work well for static scans or structured meshes with predefined animation, the same does not hold true for high-resolution 3D video frames due to their unstructured animation nature.

To deal with this challenge, different approaches have been taken such as heuristic based encoding of 3D point clouds [1,2] and image based encoding approaches [3–5]. Image based encoding approaches work well, as the 3D geometry can be projected into 2D images and then compressed using 2D image compression techniques. Since 2D image compression is a long studied field, high compression ratios can be achieved with little loss of quality. Later when the 3D geometry is needed, it can be recovered from the 2D images using image based rendering. There are three key steps to effectively compressing the geometry with these techniques: (1) projecting the 3D geometry into 2D images, (2) correctly encoding and decoding the projected images with a 2D codec, (3) recovering the 3D geometry from the 2D images.

This research addresses the second key step, correctly encoding and decoding the projected images with a 2D codec. Typically, 2D video codecs are tailored to natural sinusoidally varying images with color redundancies between frames. The codecs are tailored to natural sinusoidally varying images with the transform that they use, such as the discrete cosine transform or the integer transform. These transforms are applied to image blocks and then small variations are quantized off resulting in slightly lossy encoding at high compression levels. Detecting and encoding changes between frames instead of repeating nearly redundant information leverages color redundancies between frames. With this encoding certain frames are stored (keyframes) with changes being applied to recover frames between the stored frames (interframes) [6].

Previous research has shown that the Holoimage technique [5,7] can be extended to 3D video by modifying the fringe equations and then using OpenGL Shaders and asynchronous direct memory access (DMA) transfers to the graphics processing unit (GPU) [8]. This research used JPEG encoding on the frames and then used Quicktime Run Length Encoding on each of the frames to achieve a compressed 2D representation of the 3D geometry. With this encoding, compression ratios of over 134:1 Holovideo frame to OBJ can be achieved, at 17 frames per second encoding with an NVIDIA GeForce 9400 m GPU. Although good compression is achieved with no noticeable artifacts, it is not optimal as JPEG encoding in the RGB color space with Quicktime Run Length Encoding is not a standard 2D video encoding technique.

This research addresses this by extending the Holovideo technique to the H.264 codec, which is a standard 2D video codec. By applying a conversion to the planar YUV444 and

* Corresponding author. Tel.: +515 294 0723; fax: +515 294 3261.
E-mail addresses: isusong@gmail.com, song@iastate.edu (S. Zhang).

0143-8166/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.optlaseng.2012.12.021

YUV422 color formats, the Holovideo frames can be encoded with the H.264 encoder. With this encoding, compression ratios of over 352:1 when compared to the OBJ file format have been achieved with a mean squared error as low as 0.204%.

Section 2 explains the principle behind the technique, addressing encoding, compressing, and decoding using OpenGL Shaders (GLSL) and the H.264 codec. Section 3 shows experimental results with a unit sphere and short 45 s 3D video, and Section 4 summarizes the paper.

2. Principle

2.1. Fringe projection technique

The fringe projection technique is a structure light method from optical metrology that uses sinusoidally varying structured light patterns. 3D information is recovered from *phase* which is encoded in the sinusoidal pattern. To obtain the phase from the recovered images, a phase-shifting algorithm is typically employed. Phase shifting is used because of its numerous merits, including the capability to achieve pixel-by-pixel spatial resolution during 3D shape recovery. Over the years, a number of phase-shifting algorithms have been developed including three-step, four-step, least-square algorithms, etc. [9].

In a real-world 3D imaging system making use of a fringe projection technique, a three-step phase-shifting algorithm is typically employed due to its ability to help reduce background lighting and noise while using a small number of fringe patterns. Three fringe images with equal phase shift can be described with the following equations:

$$I_1(x,y) = I'(x,y) + I''(x,y)\cos(\phi - 2\pi/3) \tag{1}$$

$$I_2(x,y) = I'(x,y) + I''(x,y)\cos(\phi) \tag{2}$$

$$I_3(x,y) = I'(x,y) + I''(x,y)\cos(\phi + 2\pi/3) \tag{3}$$

where $I'(x,y)$ is the average intensity, $I''(x,y)$ is the intensity modulation, and $\phi(x,y)$ is the phase to be found. Simultaneously solving Eqs. (1)–(3) leads to

$$\phi(x,y) = \tan^{-1}[\sqrt{3}(I_1 - I_3)/(2I_2 - I_1 - I_3)] \tag{4}$$

This equation yields the wrapped phase $\phi(x,y)$ ranging from 0 to 2π with 2π discontinuities. A conventional phase-unwrapping algorithm can be adopted to remove these 2π phase jumps and obtain a continuous phase map [10]. This algorithm simply traverses the wrapped phase map adding integer values of 2π to $\phi(x,y)$, which can be modeled with the following equation:

$$\Phi(x,y) = \phi(x,y) + k \times 2\pi \tag{5}$$

where $\phi(x,y)$ is the wrapped phase, k is the integer number of phase jumps, and $\Phi(x,y)$ is the unwrapped phase. However, all conventional phase-unwrapping algorithms suffer from the limitations that they can neither resolve large step height changes that cause phase changes larger than π nor discontinuous surfaces.

2.2. Holovideo system setup

The Holovideo technique is a specialized fringe projection technique that uses a virtual fringe projection system. This virtual fringe projection system scans 3D scenes into 2D images, compresses and stores them, and then decompresses and recovers the original 3D scenes. Holovideo utilizes the Holoimage technique [5] to depth map 3D scenes into 2D images. Fig. 1 shows a conceptual model of the Holovideo system. In this model, the projector projects fringe images onto the scene and the camera captures the reflected fringe images from another angle. The projector in this conceptual model can be realized as a projective texture implemented through the use of the OpenGL Shading Language (GLSL), and the camera can be realized as the framebuffer. From the camera image, 3D information can be recovered pixel-by-pixel if the geometric relationship between the projector pixel (P) and the camera pixel (C) is known. Since the Holoimage system is mathematically defined using a computer graphics pipeline, both the camera and projector can be orthogonal devices and their geometric relationship can be precisely defined. Thus, converting from phase to 3D coordinates is very simple and can be done in parallel [7].

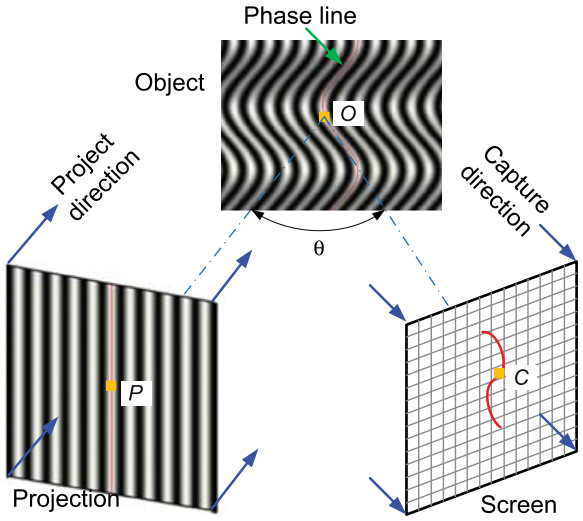

Fig. 1. Holovideo system conceptual model. The virtual projection system projects sinusoidal fringe patterns onto the object, the result is rendered by the graphics pipeline, and then displayed on the screen. The screen view acts as a virtual camera imaging system. Because both the projector and the camera are virtually constructed, they can both be orthogonal devices. The angle between the projection system and the camera imaging system is θ.

2.3. Encoding

To encode the 3D scene, the Holovideo system uses the virtual fringe projection system, which is created through the use of OpenGL Shaders. These shaders color the 3D scene with a structured light pattern defined by the following equations:

$$I_r(x,y) = 0.5 + 0.5\sin(2\pi x/P) \tag{6}$$

$$I_g(x,y) = 0.5 + 0.5\cos(2\pi x/P) \tag{7}$$

$$I_b(x,y) = S \cdot Fl(x/P) + S/2 + (S-2)/2 \cdot \cos[2\pi \cdot Mod(x,P)/P_1] \tag{8}$$

Here P is the fringe pitch, the number of pixels per fringe stripe, $P_1 = P/(K+0.5)$ is the local fringe pitch, K is an integer number, S is the stair height in grayscale intensity value, $Mod(a,b)$ is the modulus operator to get a over b, and $Fl(x)$ is the floor function to get the integer number of x. The depth resolution is then dependent on the pitch P with smaller pitches reducing quantization noise but increasing spiking noise; a discussion of the effects along with optimal values can be found in previous work [11]. Fig. 2 illustrates a typical structure pattern for Holovideo with (a) showing the resulting pattern rendered as an RGB image, and (b) a cross section of the pattern with the intensity of each color channel graphed.

For the Holovideo encoding vertex shader, a model view matrix for the projector and for the camera in the virtual structure light scanner is needed. The model view matrix for the projector

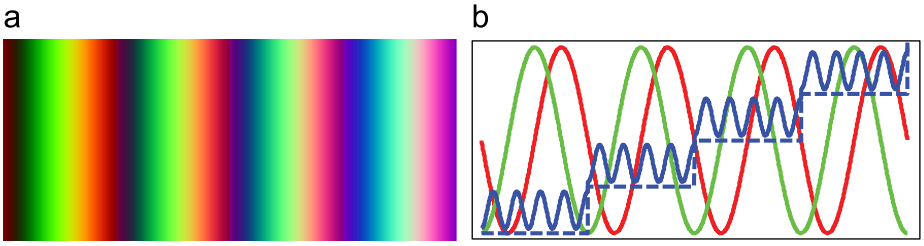

Fig. 2. The encoded structured pattern used by the Holovideo technique. (a) The structured pattern displayed as an RGB image. (b) A graph of one cross section of the structured pattern. Note that all channels use cosine waves to reduce problems associated with lossy encoding.

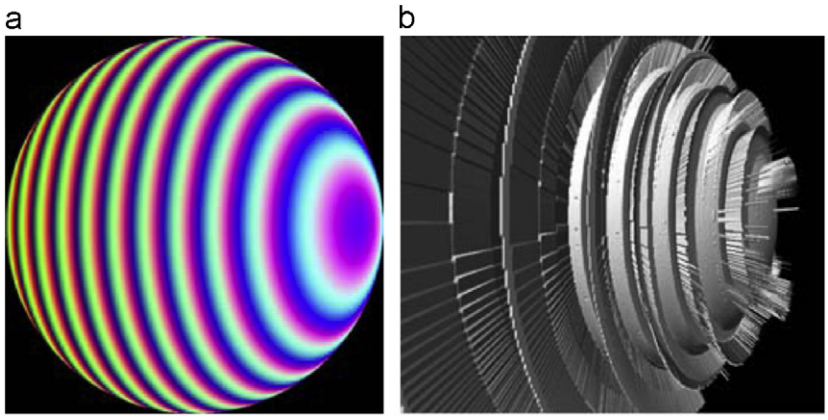

Fig. 3. Holovideo encoded unit sphere with H.264 encoding. (a) RGB Holovideo frame of unit sphere directly encoded by H.264. (b) Reconstructed unit sphere with Holovideo frame from (a).

is rotated around the z-axis by some angle ($\theta = 30^\circ$ in our case) from the camera model view matrix. From here the vertex shader passes the x, y values to the fragment shader as a varying variable along with the projector model view matrix, so that x, y values for each pixel can be determined from the projectors perspective. In the fragment shader, each fragment is colored with the Eqs. (6)–(8), and the resulting scene is rendered to a texture yielding a Holovideo encoded frame. It is important to notice that instead of directly using the stair image as proposed in Ref. [7], a cosine function is used to represent this stair image as described by Eq. (8).

Each frame of the 3D video is rendered to a texture in this fashion, and then the resulting texture is pulled from the GPU to the CPU where it can be transformed and then passed to a 2D video codec. To mitigate the bottleneck of transferring the textures from the GPU to the CPU, asynchronous DMA transfers are employed using pixel buffer objects (PBOs).

2.4. Video compression

One of the challenges in directly taking Holovideo frames into H.264 is that most H.264 codecs work in the YUV color space. If the frames are directly passed into the codec, it will convert the RGB Holovideo frame to a planar YUV frame and then compress it. Coming back out, the information is decompressed and converted back into RGB. If this process is done to the Holovideo frame Fig. 3(a), large errors are introduced, shown in Fig. 3(b). One way to address this issue would be to encode three separate movies where in each movie the YUV components would correspond to a specific color channel RGB. This would then allow for the data to be compressed and then later decompressed and reconstructed, but would require three separate synced video streams. Instead, addressing this in a better way, we transform the Holovideo frame directly into the planar YUV color space with the step height channel in the Y component, and the fringe in the U and V, shown by Fig. 4(a). Then the H.264 codec can directly compress these frames with little loss of error shown in Fig. 4(b).

Another challenge associated with H.264 video encoding is downsampling, which occurs with the frames. Since the human eye is less sensitive to color variations (chrominance UV) versus intensity variations (luminance Y), downsampling of the UV components is typically employed. The H.264 codec supports this by downsampling the UV components with YUV422 or YUV420 encoding. In this encoding scheme each pixel has an intensity or Y component, but chrominance UV components are shared between pixels. This reduces the overall bit rate with some lossy error being introduced. Downsampling with YUV422 encoding on a Holovideo frame is shown with Fig. 4(c).

2.5. Decoding on GPU

Decoding a Holovideo frame is a more involved process than encoding, as there are more steps requiring multi-pass rendering, but the process scales with the hardware through subsampling. In the decoding pipeline, shown in Fig. 5, the five major steps that need to be performed are: (1) calculating an unwrapped phase map from the Holovideo frame, (2) filtering the unwrapped phase map, (3) calculating a floating point depth map from the filtered unwrapped phase map, (4) calculating normals from the floating point depth map, (5) performing the final render. To accomplish these five steps, multi-pass rendering can be utilized, saving the results from each pass to a texture which allows neighboring pixel value access during the proceeding steps.

During steps (1)–(4) an orthographic projection with a screen aligned quad and render texture the size of the Holovideo frame is used to perform image processing. Each input image is entered into the shaders through a texture, the vertex shader simply

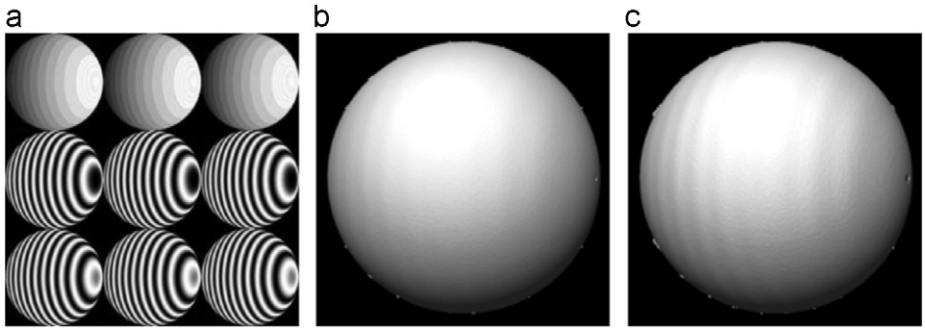

Fig. 4. Transformed Holovideo encoded unit sphere with H.264 encoding. (a) Planar YUV Holovideo frame of a unit sphere encoded by H.264. (b) Reconstructed unit sphere if Holovideo frame is encoded from YUV444 into H.264. (c) Reconstructed unit sphere if Holovideo frame is encoded from YUV422 into H.264.

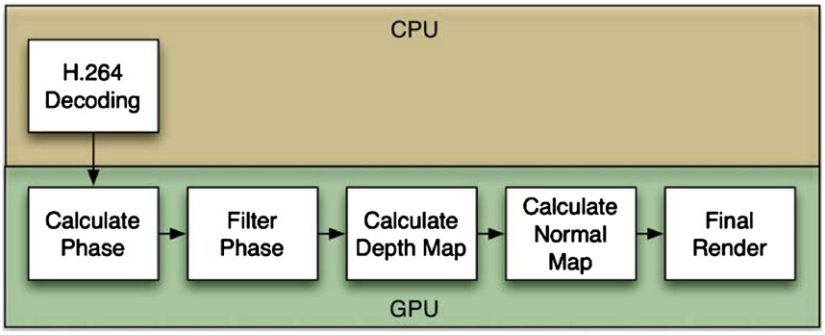

Fig. 5. Holovideo decoding pipeline. First, frames are decoded using the H.264 decoder on the CPU side, and frames are passed to the GPU as textures. Next from the Holovideo frame, the unwrapped phase map is calculated and filtered, followed by calculating a depth map and normal map. Finally, final rendering is performed with the depth map and normal map, resulting in the reconstructed scene.

passes the four vertices through, and then the fragment shader performs the pixel wise image processing.

To calculate the unwrapped phase map, step (1), we input the Holovideo frame and apply Eq. (9) below, saving the resulting unwrapped phase map to a floating point texture for the next step in the pipeline. Eqs. (6)–(8) provide the phase uniquely for each point

$$\Phi(x,y) = 2\pi \times Fl[(I_b - S/2)/S] + \tan^{-1}[(I_r - 0.5)/(I_g - 0.5)] \tag{9}$$

Unlike the phase obtained in Eq. (4) with 2π discontinuities, the phase obtained here is already unwrapped without the limitations of conventional phase unwrapping algorithms. Therefore, scenes with large height variations can be encoded which is not true when using conventional phase unwrapping algorithms. It is also important to notice that under the virtual fringe projection system, all lighting is eliminated, thus the phase can be obtained by using only two fringe patterns with $\pi/2$ phase shift. This allows for the third channel to be used for phase unwrapping.

For step (2) we used a modified median filter similar to the one proposed by McGuire [12]. The reason that median filtering needs to be applied is due to sub-pixel sampling and quantization errors during 2D image compression. Some areas of the phase $\Phi(x,y)$ have one-pixel jumps which result in large spikes in the decoded geometry known as spiking noise. We found this problem can be filtered out by a small 3×3 median filter. Instead of directly applying the median filter, we inspect the median and if it is different than the original value, we either add or subtract 2π from the value which removes the spiking noise that is introduced by having the stair image in a lossy color channel. For this step, the unwrapped phase map is passed to the shaders and the filtered unwrapped phase map is returned.

From the filtered unwrapped phase map obtained in step (2), the normalized coordinates (x^n, y^n, z^n) can be calculated, as [7]

$$x^n = j/W \tag{10}$$

$$y^n = i/W \tag{11}$$

$$z^n = \frac{P\Phi(x,y) - 2\pi i \cos(\theta)}{2\pi W \sin\theta} \tag{12}$$

This yields a value z^n in terms of P which is the fringe pitch, i, the index of the pixel being decoded in the Holovideo frame, θ, the angle between the capture plane and the projection plane ($\theta = 30^\circ$ for our case), and W, the number of pixels horizontally.

From the normalized coordinates (x^n, y^n, z^n), the original 3D coordinates can be recovered point by point forming a floating point depth map which is step (3) in the decoding process

$$x = x^n \times S_c + C_x \tag{13}$$

$$y = y^n \times S_c + C_y \tag{14}$$

$$z = z^n \times S_c + C_z \tag{15}$$

Here S_c is the scaling factor to normalize the 3D geometry and (C_x, C_y, C_z) are the center coordinates of the original 3D geometry.

Now that the depth map has been calculated, step (4) normal calculation can be performed. This is done by calculating normalized surface normals with adjacent polygons on the depth map, and then normalizing the sum of these adjacent surface normals to get a normalized point normal. These values are passed out of the shader as a texture which forms the normal map.

Finally, the final rendering pass can be performed, step (5). Before this step is performed, the projection is switched back to a perspective projection, and the back screen buffer is bound as the draw buffer. Then the final render shaders are bound and a plane

of pixels is rendered out. In the vertex shader, the vertex is modified according to the depth map. In the fragment shader, per pixel lighting is applied using the normal map calculated during step (4). To subsample the geometry, the number of pixels rendered out in this stage can be reduced by some divisor of the width and height of the Holovideo frame. This allows for a simple subsampling mechanism, since the points will not get calculated during the shader passes, reducing the level of detail and computational load. This is what allows the Holovideo technique to scale to different devices with various graphics capabilities.

3. Experimental results

In all of our experiments we used a Holovideo system configured as follows: $512(W) \times 512(H)$ image resolution, $\theta = 30^{\circ}$ for the angle between the projection and capture planes, fringe pitch $P=42$, and high-frequency modulation pitch $P=4$. Previous work has compared the Holovideo technique to other techniques [8], in this discussion we will focus on the results of applying Holovideo with H.264 encoding. To verify the performance of the proposed encoding system, we first encoded a unit sphere which represents smooth changing geometry. Fig. 6 shows the results. Fig. 6(a) shows the

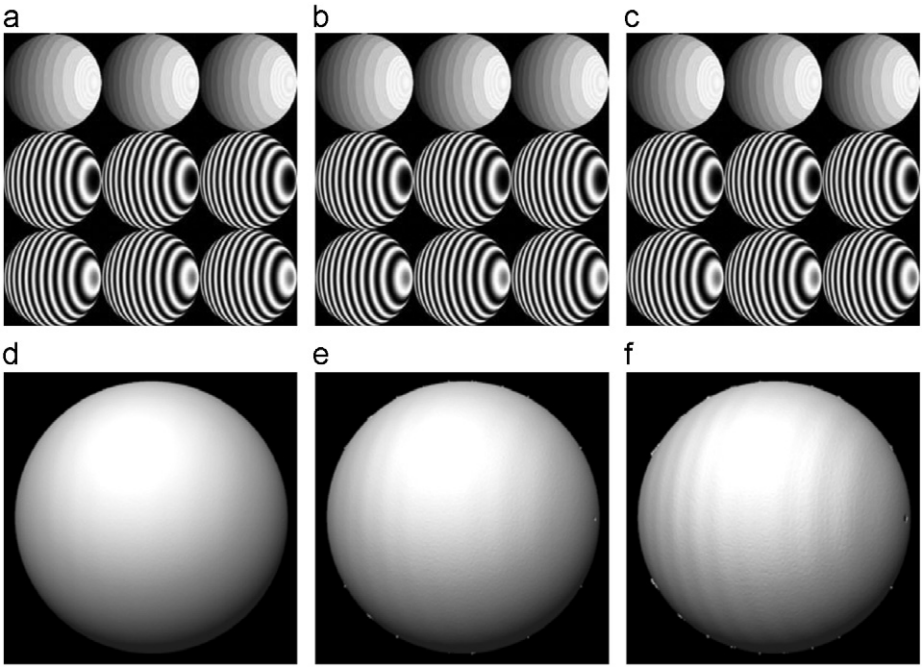

Fig. 6. The effect of downsampling and compressing the unit sphere with the H.264 codec. (a) Original planar YUV Holovideo frame, (b) recovered planar YUV444 Holovideo frame, (c) recovered planar YUV422 Holovideo frame, and (d)–(f) corresponding reconstructed unit sphere.

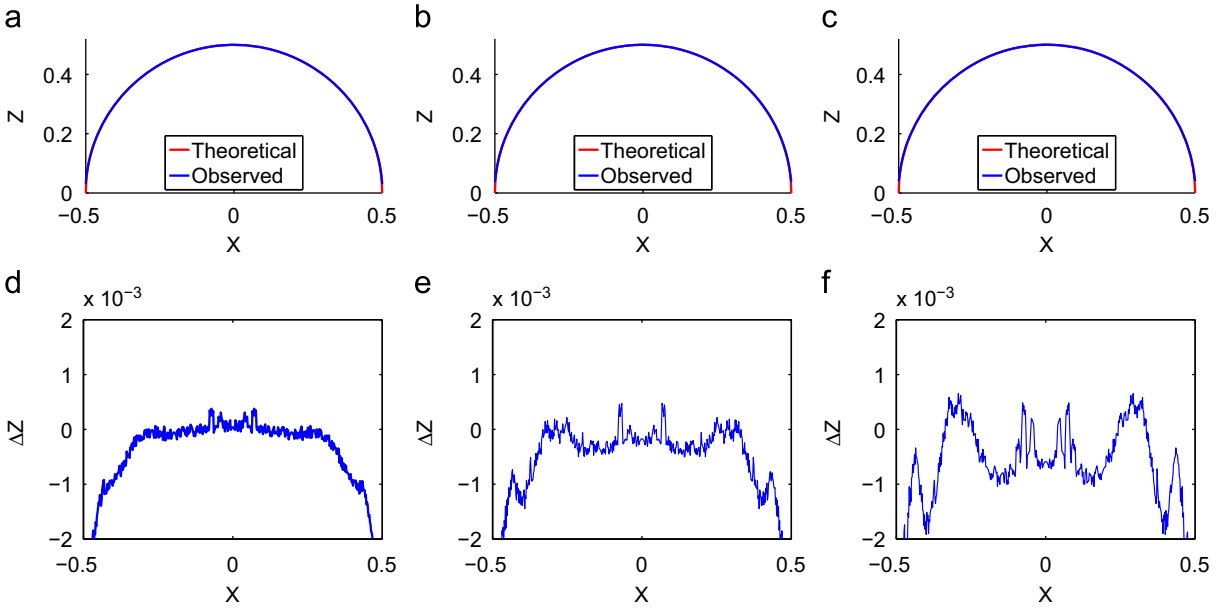

Fig. 7. The effect of downsampling and compressing the unit sphere with the H.264 codec. (a)–(c) Cross section of reconstructed unit sphere from Fig. 6. (d)–(f) ΔZ between theoretical and observed value.

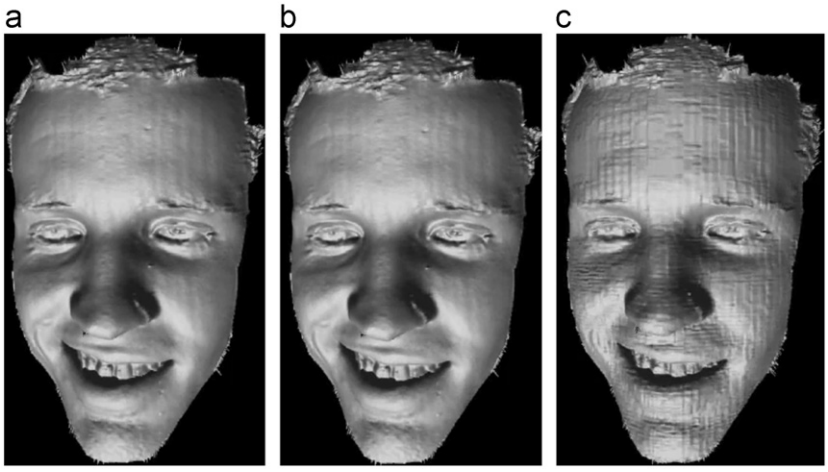

Fig. 8. Sample frame from a short 45 s 3D video clip captured using a real-time structured light scanner running at 30 frames per second. (a) The original rendered data (Video 1), (b) the data with H.264 Holovideo encoding using the FFMPEG *lossless_max* preset (Video 2), (c) the data with H.264 Holovideo encoding using the FFMPEG *baseline* settings (Video 3). FFMPEG *lossless_max* settings in the YUV444 color format give a compression ratio of over 352:1, and FFMPEG *baseline* settings in the YUV422 color format give a compression ratio of over 6086:1.

original planar YUV Holovideo frame, Fig. 6(b) shows the recovered planar YUV444 Holovideo frame, and Fig. 6(c) shows the recovered planar YUV422 Holovideo frame. Fig. 6(d)–(f) shows the reconstructed unit spheres with their respective encoding, Fig. 7(a)–(c) shows cross sections of the sphere, and Fig. 7(d)–(f) shows plots of the ΔZ. When downsampling on the color channels is used, bands corresponding to the fringe patterns show up on the geometry as noise, but the mean squared error remains at 0.204% and 0.415%, respectively.

To further test the performance of the proposed encoding system, we used it on a short 45 s 3D video clip captured using a structured light scanner with an image resolution of $640(H) \times 480(W)$ running at 30 frames per second [13]. The 3D frames were Holovideo encoded and run through the H.264 encoder using the FFMPEG *lossless_max* preset in a YUV444 color format, and using the FFMPEG *baseline* preset in a YUV422 color format. Fig. 8(a) shows a sample frame from the scanner (associated Video 1), (b) the *lossless_max* preset Holoencoded frame (associated Video 2), and (c) the *baseline* preset Holoencoded frame (associated Video 3). The original size of the video in the OBJ format was 42 GB. In the *lossless_max* preset Holovideo format, the resulting video size is 119 MB resulting in a compression ratio of over 352:1. Using the *baseline* preset Holovideo format, the resulting video size is 6.9 MB resulting in a compression ratio of over 6086:1.

4. Conclusion

We have presented a way to utilize the Holovideo technique with 2D video codecs that use the YUV color space, specifically the H.264 codec. Example frames of a unit sphere and a recorded data set were encoded, decoded, and presented. A mean squared error of only 0.204% was achieved when using the planar YUV444 color space and 0.415% when using planar YUV422. Applying the technique to a short 45 s 3D video clip captured using a real-time structured light scanner, a compression ratio of over 352:1 was achieved when compared to the OBJ file format. Currently, decoding at 28 frames per second with an NVIDIA GeForce 9400 m GPU can be achieved, with encoding at 17 frames per second. Future work for this research includes ways of minimizing interframe changes to optimize the H.264 codec's keyframe and interframe coding to maximize compression, along with optimizing encoding parameters to achieve high compression with little loss of quality.

References

[1] Merry B, Marais P, Gain J. Compression of dense and regular point clouds. Comput Graph Forum 2006;25(4):709–16.
[2] Gumhold S, Kami Z, Isenburg M, Seidel H-P. Predictive point-cloud compression. In: ACM SIGGRAPH 2005 sketches on SIGGRAPH '05; 2005. p. 137.
[3] Gu X, Gortler SJ, Hoppe H. Geometry images. ACM Trans Graph 2002;21(3).
[4] Krishnamurthy R, Chai B, Tao H. Compression and transmission of depth maps for image-based rendering. Image Process 2002;1(c).
[5] Gu X, Zhang S, Zhang L, Huang P, Martin R, Yau S-T. Holoimages, ACM solid and physical modeling, ACM New York, NY, USA 2006. p. 129–38.
[6] Richardson IE. The H.264 advanced video compression standard electronic version. Chichester: John Wiley & Sons Ltd; 2010.
[7] Karpinsky N, Zhang S. Composite phase-shifting algorithm for 3-D shape compression. Opt Eng 2010;49(6) 063,604.
[8] Karpinsky N, Zhang S. Holovideo: real-time 3D video encoding and decoding on GPU. Opt Laser Eng 2012;50(2):280–6.
[9] Schreiber H, Bruning JH. Optical shop testing. third ed. New York, NY: John Willey & Sons; 2007 p. 547–666 [chapter 14].
[10] Ghiglia DC, Pritt MD. Two-dimensional phase unwrapping: theory, algorithms, and software. New York, NY: John Wiley and Sons, Inc; 1998.
[11] Karpinsky N. 3D geometry compression with Holoimage. Iowa State University; 2011.
[12] McGuire M. A fast, small-radius GPU median filter. ShaderX6; 2008.
[13] Zhang S, Yau S-T, High-resolution. Real-time 3D absolute coordinate measurement based on a phase-shifting method. Opt Express 2006;14(7): 2644–9.