Optics and Lasers in Engineering 50 (2012) 280–286

Contents lists available at SciVerse ScienceDirect

Optics and Lasers in Engineering

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

Holovideo: Real-time 3D range video encoding and decoding on GPU

Nikolaus Karpinsky, Song Zhang *

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

ARTICLE INFO

Article history:
Received 13 June 2011
Received in revised form
4 August 2011
Accepted 18 August 2011
Available online 9 September 2011

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

ABSTRACT

We present a 3D video-encoding technique called Holovideo that is capable of encoding high-resolution 3D videos into standard 2D videos, and then decoding the 2D videos back into 3D rapidly without significant loss of quality. Due to the nature of the algorithm, 2D video compression such as JPEG encoding with QuickTime Run Length Encoding (QTRLE) can be applied with little quality loss, resulting in an effective way to store 3D video at very small file sizes. We found that under a compression ratio of 134:1, Holovideo to OBJ file format, the 3D geometry quality drops at a negligible level. Several sets of 3D videos were captured using a structured light scanner, compressed using the Holovideo codec, and then uncompressed and displayed to demonstrate the effectiveness of the codec. With the use of OpenGL Shaders (GLSL), the 3D video codec can encode and decode in realtime. We demonstrated that for a video size of 512×512, the decoding speed is 28 frames per second (FPS) with a laptop computer using an embedded NVIDIA GeForce 9400 m graphics processing unit (GPU). Encoding can be done with this same setup at 18 FPS, making this technology suitable for applications such as interactive 3D video games and 3D video conferencing.

1. Introduction

Advancements in 3D imaging and computational technology have made the acquisition and display of 3D data simpler. Techniques such as structured light, stereovision, and LIDAR have led the way in 3D data acquisition [1]. In recent years, real-time 3D video scanning techniques have been increasingly more feasible [2–10]. Stereoscopic displays have made the display of 3D data a reality. However, as these fields and techniques evolve, a growing problem is being encountered: how can 3D video data be efficiently stored and transmitted?

Storage and transmission of high-resolution 3D video data have become a large problem due to the file sizes associated with 3D geometry. Standard 3D file storage techniques (e.g., STL, OBJ, PLY) do not lend themselves nicely to highly detailed, high frame-rate scenes. Instead, traditional 3D file storage techniques aim to store models and then animated models based on constraints of a few points, typically skeletal points [11]. This does not hold true for 3D scenes captured from 3D scanners as they consist of a large array of 3D points, all of which are animated; this animation is inherently unstructured and unconstrained making typical 3D file storage difficult. Introducing 3D models into 3D scenes only further exacerbates the problem, as both modalities need to be accounted for. How then can such scenes with both types of 3D data be encoded in a unifying way, which provides not only an efficient storage and transmission medium, but also a quick encoding and decoding of high definition (HD) data?

With the advent of structured light scanners comes the possibility to realize real-time 3D video. This makes applications such as 3D video conferencing and 3D presence a possibility. For example, the University of South California (USC) group has demonstrated that 3D video conferencing is viable with powerful hardware system configurations [12]. With these possibilities come the caveat of increased data size through the addition of the third dimension. Previous techniques for 3D video have been proposed [13], but typically involve an intolerance to lossy formats, a lengthy encoding process, or low depth resolution.

Various methods [14–16] have been proposed which allow for amazing compression ratios with lossless quality on 3D point clouds, but these involve lengthy encoding processes. When dealing with real-time 3D video these techniques are undesirable as they do not allow for online encoding of the data, limiting its end-user applications.

The method of compressing a phase map, as indicated in [12], at least in structured light scanner cases, comes to mind but has a problem. The depth map is always a floating-point grayscale image, and thus must be packed into a 24-bit image for most image and video compression. This can easily be done by discarding some of the resolution of the floating-point mantissa bits, dropping the eight least-significant bits, and saving to a lossless format. In this packed format, the phase map can be unpacked and the 3D geometry can be reconstructed with little quality loss.

* Corresponding author. Tel.: +1 515 294 0723; fax: +1 515 294 3261.
E-mail address: song@iastate.edu (S. Zhang).

0143-8166/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.optlaseng.2011.08.002

The problem occurs when using lossy formats, such as most video codecs. Since the most significant bits contain the power bits, any change in them changes the unpacked floating-point number drastically.

One solution to this problem is to use depth mapping to encode 3D scenes consisting of unstructured scanned data and structured models into 2D images, and then to rely on 2D image/video compression and transmission techniques. The benefit of doing this is that decades of research and development in 2D image processing can be leveraged utilizing existing compression and transmission techniques along with existing infrastructure. Existing video services such as Youtube and Vimeo can be used with slight modifications; only the video renderer needs to be modified to decode and display 3D scenes rather than 2D images. The Microsoft Kinect is a great example of a real-time 3D scanner that makes use of a depth map. The simple grayscale depth map allows it to reach real-time speeds and leverage 2D image processing. The caveat of their approach is that the depth is limited by the grayscale image color resolution of 256 different values. Also, lossy encoding typically blurs sharp edges in the geometry, and distorts the final rendered scene. These techniques do have the benefit of being extremely fast, through parallelization and simple storage, along with the ability to use existing 2D compression techniques to compress 3D data. Therefore, when we developed the Holovideo compression we designed a technique which uses these benefits.

Holovideo utilizes the basis of the Holoimage technique [17] to accomplish the task of depth-mapping an entire 3D scene. Utilizing techniques developed in optical metrology, Holoimage creates a virtual fringe projection system which can depth-map an entire 3D scene into a 2D image. The benefits of such a technique include: (1) using existing research in the field of optical metrology; and (2) leveraging existing research in the field of image processing. However, the original implementation has the limitation of encoding *smooth* geometry because of the nature of the encoding, i.e., continuous 3D surfaces without large step changes or discontinuities. The complexity of the decoding process was also difficult and required a powerful computer and graphics card to rapidly recover 3D geometry for real-time applications.

Recently, we proposed a technique to compress arbitrary 3D shapes using Holoimage [18]. For this technique, an arbitrary 3D shape can be encoded as a 24-bit color image with red and green channel as sine and cosine fringe images, while the blue channel as the stair image for phase unwrapping. Because the third channel is used to unwrap the phase map obtained from red and green channels of fringe images point by point, no spatial phase unwrapping is necessary, thus it can be used to recover arbitrary 3D shapes. However, because a stair image is used for blue channel, any information loss will induce problems to correctly recover 3D geometry, thus the whole color image cannot be stored in any lossy format. This problem becomes more significant for videos because most 2D video formats are inherently lossy.

To circumvent this problems, this paper presents a new coding that encode the blue channel with smoothed cosine functions. Because all three channels use smooth cosine functions, lossy image format can be used to restore the original geometry properly. Because a lossy image format can be used, it enables 3D video encoding with standard 2D video formats. This technique is called Holovideo. The Holovideo technique allows existing 2D video codecs such as QuickTime Run Length Encoding (QTRLE) to be used on 3D videos, resulting in compression ratios of over 134:1 Holovideo to OBJ format. We found that under a compression ratio of 134:1, Holovideo to OBJ file format, the 3D geometry quality drops at a negligible level. Several sets of 3D videos were captured using a structured light scanner, compressed using the Holovideo codec, and then uncompressed and displayed to demonstrate the effectiveness of the codec. With the use of OpenGL Shaders (GLSL), the 3D video codec can encode and decode in realtime. We demonstrated that for a video size of 512×512, the decoding speed is 28 frames per second (FPS) with a laptop computer using an embedded NVIDIA GeForce 9400 m graphics processing unit (GPU). Encoding can be done with this same setup at 18 FPS, making this technology suitable for applications such as interactive 3D video games and 3D video conferencing.

Section 2 explains the principle of encoding and decoding using OpenGL Shaders (GLSL). Section 3 shows experimental results. Section 4 discusses the caveats of the proposed technique, and Section 5 summarizes this paper.

2. Principle

2.1. Fringe projection technique

Fringe projection technique is a special structured light method in that it uses sinusoidally varying structured patterns. In a fringe projection system, the 3D information is recovered from *phase* which is encoded naturally into the sinusoidal pattern. To obtain the phase, a phase-shifting algorithm is typically used. Phase shifting is extensively used in optical metrology because of its numerous merits which include 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. [19]. In a real-world 3D imaging system using a fringe projection technique, a three-step phase-shifting algorithm is typically used because of the existence of background lighting and noise. Three fringe images with equal phase shift can be described as

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

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

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

where $I'(x,y)$ is the average intensity, $I''(x,y)$ the intensity modulation, and $\phi(x,y)$ 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 provides the wrapped phase ranging from 0 to 2π with 2π discontinuities. These 2π phase jumps can be removed to obtain the continuous phase map by adopting a phase-unwrapping algorithm [20]. However, all phase-unwrapping algorithms have the common limitations that they can resolve neither large step height changes that cause phase changes larger than π nor discontinuous surfaces.

2.2. Holovideo system setup

The Holovideo technique is devised from the digital fringe projection technique. The idea is to create a virtual fringe projection system, scan scenes into 2D images, compress and store them, and then decompress and recover the original 3D scenes. Holovideo utilizes the basis of the Holoimage technique [17] to accomplish the task of depth-mapping an entire 3D scene. Fig. 1 shows the typical setup of the Holovideo system. The projector projects fringe images onto the object and the camera captures reflected fringe images from another viewing angle. 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. Because Holoimage system is precisely defined by the user, both the camera and the projector

can be orthogonal devices, their geometric relationship is easy to obtain. Thus, the phase to coordinate conversion is very simple [18]. In the virtual fringe projection system, the projector is configured as the projective texture image to project the texture onto the object, the computer screen acts as the camera. The projection angle (θ) is realized by setting the model view matrix of the OpenGL pipeline.

2.3. Encoding on GPU

To speed up the encoding process, the Holovideo system was constructed on GPU. The virtual fringe projection system is created through the use of GLSL Shaders which color the 3D scene with the structured light pattern. The result is rendered to a texture, saved to a video file, and uncompressed later when needed. By using a sinusoidal pattern for the structured light system, lossy compression can be achieved without major loss of quality.

As stated before, the Holovideo encoding shader colors the scene with the structured light pattern. To accomplish this, a model view matrix for the projector in the virtual structured light scanner is needed. This model view matrix is rotated around the z axis by some angle ($\theta = 18^\circ$ in our case) from the camera matrix. From here the Vertex Shader can pass the x,y values to the Fragment Shader as a varying variable along with the projector model view, which can then be used to find the x, y values for each pixel from the projectors perspective. At this point, each fragment is colored with Eqs. (5)–(7), and the resulting scene is rendered to a texture giving a Holo-encoded scene:

$$I_r(x,y) = 0.5 + 0.5\sin(2\pi x/P), \tag{5}$$

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

$$I_b(x,y) = S \cdot Fl(x/P) + S/2 + (S-2)/2 \cdot \cos[2\pi \cdot Mod(x,P)/P_1], \tag{7}$$

here P is the fringe pitch, the number of pixels per fringe stripe, $P_1 = P/(K+0.5)$ is the local fringe pitch and 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 to get the integer number of x by removing the decimals. Compared to previous work [18], Eq. (7) has been modified to vary sinusoidally. This is done so that all three color channels can be stored in a lossy format, which makes it feasible to encode 3D video data into lossy 3D video format. In addition, since the processing can be performed in parallel, we have implemented the processing on GPU. All details will be addressed later during the experimental results discussion. Fig. 2 illustrates a typical structure pattern for Holovideo.

After each render, which renders to a texture, we pull the texture from the GPU and save it as a frame in the current movie file. The two main bottlenecks are transferring all of the geometry to the graphics card to be encoded, and copying the resulting texture from the graphics card to the movie file in the computer memory. Since we already have to transfer the geometry to the GPU there is nothing we can do about the former bottleneck. The latter bottleneck, however, can be mitigated by accessing textures from the GPU through DMA using pixel buffer objects, resulting in asynchronous transfers.

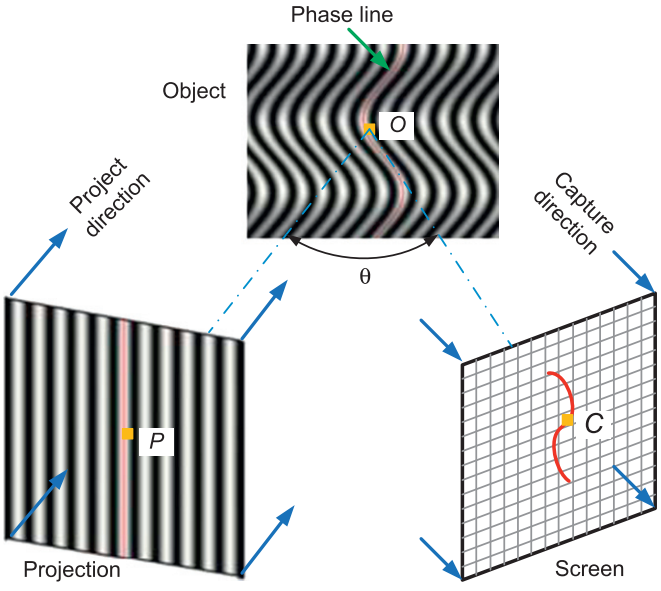


Fig. 1. Holovideo system setup. The virtual projection system projects sinusoidal fringe patterns onto the object and rendered by the graphics pipeline, and then displayed on the screen. The screen view acts as a virtual camera imaging system. Because both the projector and the camera are virtually constructed,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

2.4. Decoding on GPU

Decoding the resulting Holovideo is more involved than encoding, as there are more steps, but it can be scaled to the hardware by simply subsampling. In decoding, four major steps need to be accomplished: (1) calculating the phase map from the Holovideo frame, (2) filtering the phase map, (3) calculating normals from the phase map, and (4) performing the final render. To accomplish these four steps, we utilized multipass rendering, saving results from the intermediate steps to a texture, which allowed us to access neighboring pixel values in proceeding steps.

To calculate the phase map, we set up the rendering with an orthographic projection and a render texture and then rendered a screen-aligned quad. With this setup, we can perform image processing using GLSL. From here, the phase-calculating shader took each pixel value and applied Eq. (8) below, saving the result to a floating-point texture for the next step in the pipeline. Eqs. (5)–(7) provide the phase uniquely for each point:

$$\Phi(x,y) = 2\pi \times Fl[(I_b - S/2)/S] + \tan^{-1}[(I_r - 0.5)/(I_g - 0.5)]. \tag{8}$$

Unlike the phase obtained in Eq. (4) with 2π discontinuities, the phase obtained here is already unwrapped naturally without the common limitations of conventional phase unwrapping algorithms.

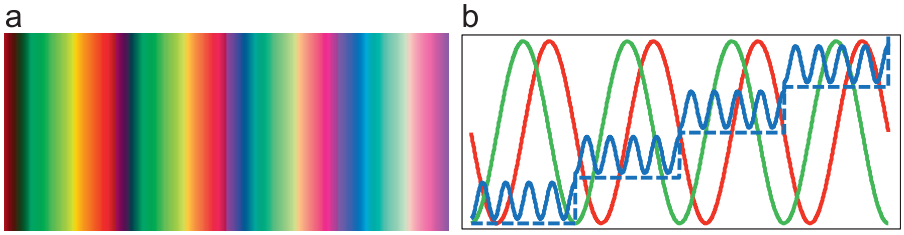


Fig. 2. The encoded structured pattern. (a) The structured pattern, whose three channels are all encoded with cosine functions. (b) One cross section of the structured pattern. Note that all channels use cosine waves to reduce problems associated with lossy encoding.

Therefore, it can be used to encode an arbitrary 3D scene scanned by a 3D scanner even with step height variations. It is important to notice that under the virtual fringe projection system all lighting can be controlled or eliminated, thus the phase can be obtained by two-channel fringe patterns with $\pi/2$ phase shift. This allows for the third channel to be used for phase unwrapping.

Since the phase is calculated point by point, it allows for leveraging the parallelism of the GPU for the decoding process. It is also important to notice that instead of directly using the stair image as proposed in Ref. [18], we use a cosine function to represent this stair image as described by Eq. (7). If the image is stored in a lossy format, the smooth cosine function causes less problems than the straight stair function with sharp edges.

From the unwrapped phase $\Phi(x,y)$ obtained in Eq. (8), the normalized coordinates (x^n,y^n,z^n) can be decoded as [18]

$$x^n = j/W, \tag{9}$$

$$y^n = i/W, \tag{10}$$

$$z^n = \frac{P\Phi(x,y) - 2\pi i \cos(\theta)}{2\pi W \sin\theta}. \tag{11}$$

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 ($\theta = 18^\circ$ 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 recovered point by point

$$x = x^n \times S_c + C_x, \tag{12}$$

$$y = y^n \times S_c + C_y, \tag{13}$$

$$z = z^n \times S_c + C_z. \tag{14}$$

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

Because of the subpixel sampling error, we found that some areas of the phase $\Phi(x,y)$ have one-pixel jumps along the edge of the stair image on I_b. This problem can be easily filtered out since it is only one pixel wide. The filter that we perform on the phase map is a median filter which removes spiking noise in the phase map. We used McGuire's method, allowing for a fast and efficient median filter in a GLSL Shader [21].

Normal calculation is done by calculating surface normals with adjacent polygons, and then averaging them together to form a normal map. Again, this uses the same setup as above with the orthogonal projection, render texture, and screen-aligned quad.

At last we have the final render step. Before we perform this step, we switch to a perspective projection, although an orthogonal projection could be used. We also bind the back screen buffer as the main render buffer, bind the final render shader, and then render a plane of pixels. With the plane of pixels, we can reduce the number of vertices by some divisor of the width and height of the Holovideo. This allows us to easily subsample the Holovideo, reducing the detail of the final rendering but also reducing the computational load. This is what allows the Holovideo to scale from devices with small graphics cards to those with large workstation cards.

2.5. 3D video compression

Because each frame is encoded with cosine functions, lossy image formats can be used. Therefore, lossy compression results in little loss of quality if the codec is properly selected. Most codecs use some transform that approximates the Karhunen–Loève Transform (KLT) such as the cosine or integer transform. These transforms work the best on the so-called natural images where there are no sharp discontinuities in the color space of the local block that the transform is applied to. Since the Holovideo uses cosine waves, the discontinuities are minimized and the transform yields highly compressed blocks which can then be quantized and encoded.

3. Experimental results

To verify the performance of the proposed Holovideo encoding system, we first encode one single 3D frame with rich features. Fig. 3 shows the results. For this example, the Holovideo system is configured as follows: the image resolution is 512 (W) × 512 (H); the angle between the projection and the camera $\theta = 18^\circ$; the fringe pitch P=32 pixels; the high-frequency modulation pitch P=6 pixels; and the stair height is S=16. Fig. 3(a) shows the original 3D geometry that is compressed into a single color Holoimage as shown in Fig. 3(b). The red and green channels are encoded as sine and cosine fringe patterns as shown in Fig. 3(c) and (d), respectively. Fig. 3(e) shows the blue channel image that is compose of stair with high-frequency cosine functions following Eq. (7). From the red and green channels, the phase can be wrapped with a value ranging from $-\pi$ to $+\pi$ as shown in Fig. 3(f). From the third channel, a stair image shown in Fig. 3(g) can be obtained, which can be used to unwrap the phase. Fig. 3(h) shows the unwrapped raw phase map $\Phi(x,y)^r$. Because of the sub-pixel sampling issue, the perfectly designed stair image and the wrapped phase may have misalignment on edges. This figure shows some artifacts (white does) on the raw unwrapped phase map.

Because the artifacts are single pixel in width, they could be removed by applying a median filter to obtain smoothed unwrapped phase $\Phi^s(x,y)$. However, applying a single median filter will make the phase on those artifact point incorrect. Fortunately, because the phase changes must be multiples $n(x,y)$ of 2π for those artifact points, we only need to determine the integer number $n(x,y)$ to correct those points. In this research, $n(x,y)$ was determined as

$$n(x,y) = \text{round}\left[\frac{\Phi^s(x,y) - \Phi^r(x,y)}{2\pi}\right], \tag{15}$$

and the correctly unwrapped phase map can be obtained by

$$\Phi(x,y) = \Phi(x,y)^r + n(x,y) \times 2\pi. \tag{16}$$

Fig. 3 (i) shows the unwrapped phase map after properly removing those artifacts using the aforementioned procedures. The normal map can be calculated once the (x,y,z) coordinates could be calculated using Eqs. (12)–(14). Fig. 3(j) shows the computed normal map. Finally, 3D geometry could be rendered on GPU as shown in Fig. 3(k). To compare the precision of the reconstructed 3D geometry and the original 3D geometry, they are rendered in the same scene as shown in Fig. 3(l), here gold color geometry represents the original 3D geometry and the gray color represents the recovered 3D geometry. It clearly shows that they are well aligned together, which conforms to our previous finding: the error is negligible for the Holoimage to represent a 3D geometry [22].

To demonstrate the potential of compressing Holovideo with lossy formats, we compressed a single frame with varying levels of JPEG compression under Photoshop 10.0. Fig. 4 shows the results of compressing the 3D frame shown Fig. 3. Fig. 4(a)–(d) shows compressed JPEG images with the quality levels of 12, 10, 8, 6, respectively. From these encoded images, the 3D shape can be recovered, as shown in Fig. 4(e)–(h). It can been that the image can be stored as high quality lossy JPEG files without bringing obvious problems to recovered 3D geometry. With more compressed images being used, the recovered 3D geometry quality reduces (i.e., losing details), and some artifacts (spikes)

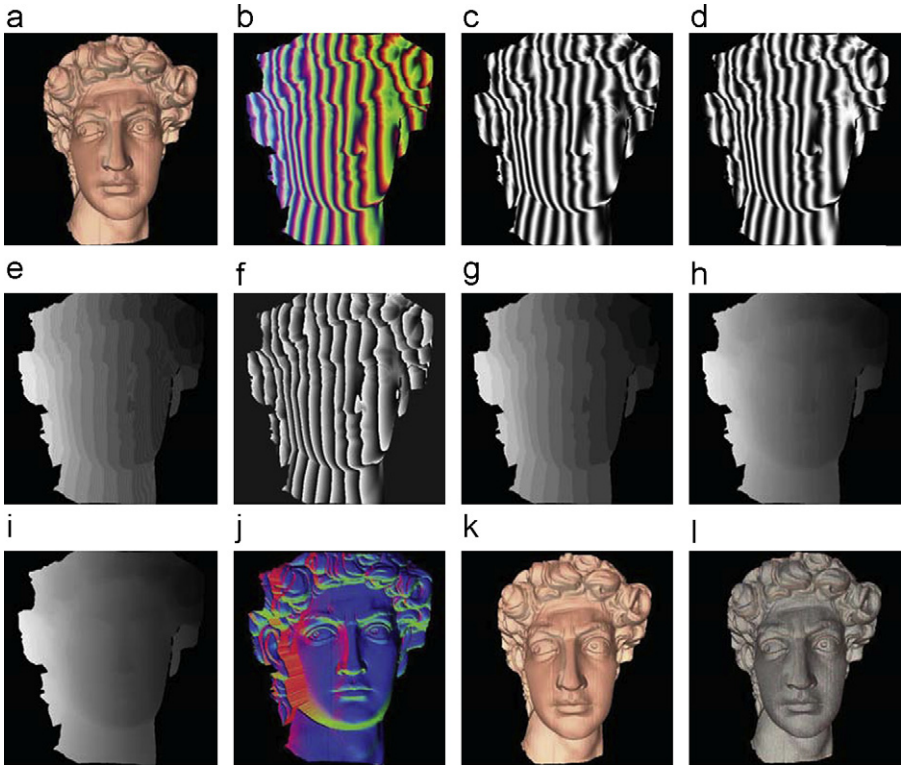


Fig. 3. Example of the Holovideo codec encoding a single frame from 3D to 2D and then decoding back to 3D. (a) The original scanned 3D geometry by a structured light scanner; (b) the encoded 3D frame into 2D image; (c)–(e) three color channels of the encoded 2D image frame; (f) wrapped phase from red and green channel fringe patterns; (g) image codec used for unwrapping the wrapped phase point by point; (h) the unwrapped phase map using Eq. (8); (i) the unwrapped phase after filtering; (j) the normal map; (k) the final 3D recovered geometry; (l) overlapping the original 3D geometry with the recovered one. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

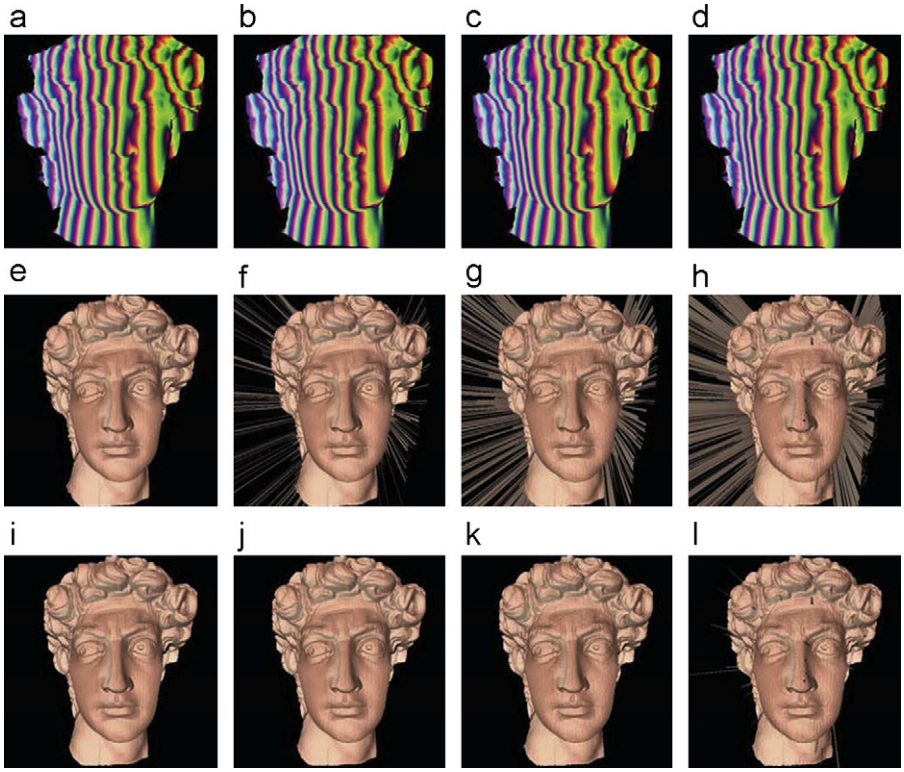


Fig. 4. The effect of compressing an individual frame with a lossy JPEG file format. (a)–(d) The Holovideo encoded compressed frame with different compression ratios; (e)–(h) The corresponding decoded 3D geometry from the above images. (i)–(l) The 3D geometry of above row after boundary cleaning. The images in (a)–(d) respectively show the compression ratios of 104:1, 174:1, 237:1, and 310:1 when compared against the OBJ file format.

start appearing. However, most of problematic points occur around boundary regions which are caused by sharp intensity changes of the image. The boundary problems can be significantly reduced if a few pixels are dropped out. Fig. 4(i)–(l) show the corresponding results after removing the boundary. This experiment clearly indicates that the proposed encoding method allows the use of the lossy imaging format.

OBJ file format is widely used to store 3D mesh data. If the original 3D data is stored in OBJ format without normal information, the file size is 20,935 kB. In comparison with the OBJ file format, we could reach 174:1 with slight quality dropping. When the compression ratio reaches 310:1, the quality of 3D geometry is noticeability reduced, but the overall 3D geometry is still well preserved.

As a comparison, if we use the encoding method discussed in Ref. [18] with blue channels as a straight stair function. The encoded image is shown in Fig. 5(a). If the image is stored as lossless format (e.g., bitmap), the 3D geometry can be accurately recovered as illustrated in Fig. 5(b). Fig. 5(c) shows that even the image is stored as the highest quality level (12) JPEG format, the 3D recovered result appears to be problematic. If the image quality is reduced to quality level 10, the 3D shape cannot be recovered, as show in Fig. 5(d). It is important to note that the boundaries of 3D recovered results were cleaned using the same aforementioned approach. This clearly demonstrate that the previously proposed encoding method cannot be used if a lossy image format is needed.

To show that the proposed method can be used to encode and decode 3D videos, we captured a short 45-s clip of an actress using a structured light scanner [23] running at 30 FPS with an image resolution of 640×480 per frame. Saving each frame out in the OBJ format, we end up with over 42 GB worth of data. Then we took the data, ran it through the Holovideo encoder, and saved the raw lossless 24-bit Bitmap data to an AVI file with a resolution of 512×512, which resulted in a file that was 1 GB. This is already a compression of over 42:1 Holovideo to OBJ. Next we JPEG-encoded each frame and ran it through the QTRLE codec, which resulted in a file that was 314.3 MB, achieving a ratio of over 134:1 Holovideo to OBJ. Media 1 associated with Fig. 6 shows the original scanned 3D video (the video on the left), the encoded Holovideo (the video in the middle), and the decoded 3D video (the video on the right). The resulting video had no noticeable artifacts from the compression, and it could be further compressed with some loss of quality. All of the encoding and decoding processes are performed on the GPU in real-time (28 FPS decoding and 18 FPS encoding) with a simple graphics card (NVIDIA GeForce 9400 m) and the resulting compression ratio is over 134:1 in comparison with the 3D data stored in OBJ file format.

4. Discussion

One caveat to note is that a lot of video codecs are tailored to the human eye and reduce information in color spaces that humans typically do not notice. An example of this is the H.264 codec which converts source input into the YUV color space. The human eye has a higher spatial sensitivity to luma (brightness) than chrominance (color). Knowing this, bandwidth can be saved by reducing the sampling accuracy of the chrominance channels with little impact on human perception of the resulting video.

Compression codecs that use the YUV color space currently do not work with the Holovideo compression technique as they result in large blocking artifacts. Thus we used the QTRLE codec which is a lossless run length encoding video codec. To achieve lossy video compression, we JPEG-encoded the source images in the RGB color space and then passed them to the video encoder. This allows us to achieve a high compression ratio at a controllable quality level.

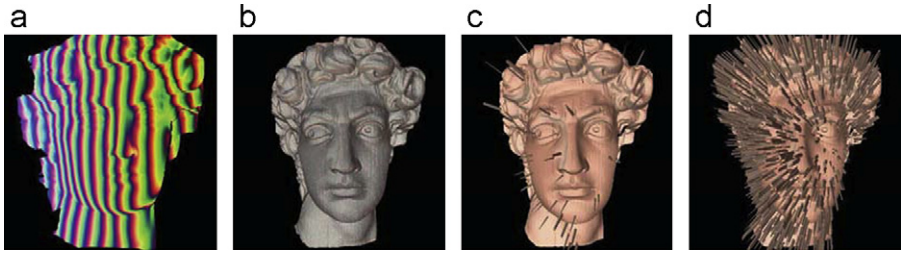


Fig. 5. Comparing results of storing prior encoded image with a lossy JPEG file format. (a) The encoded Holovideo frame using the method introduced in Ref. [18]; (b) overlap the original 3D scanned data with the recovered 3D geometry from the lossless bitmap file; (c) 3D recovered shape when the image was stored as lossy JPEG format with quality level 12; (d) 3D recovered shape when the image was stored as lossy JPEG format with quality level 10.

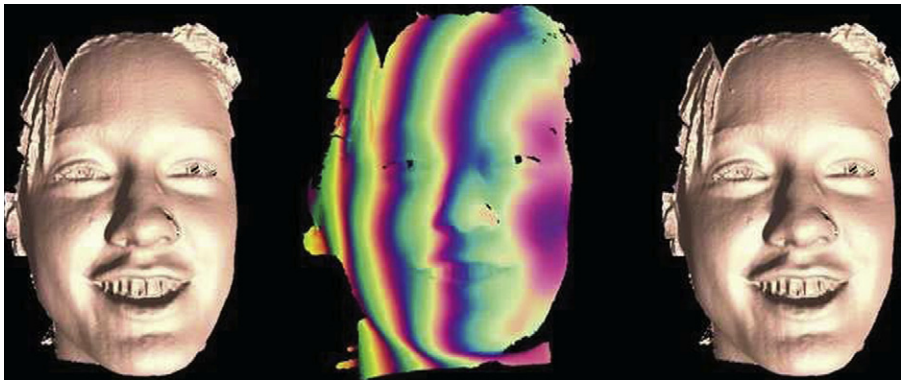
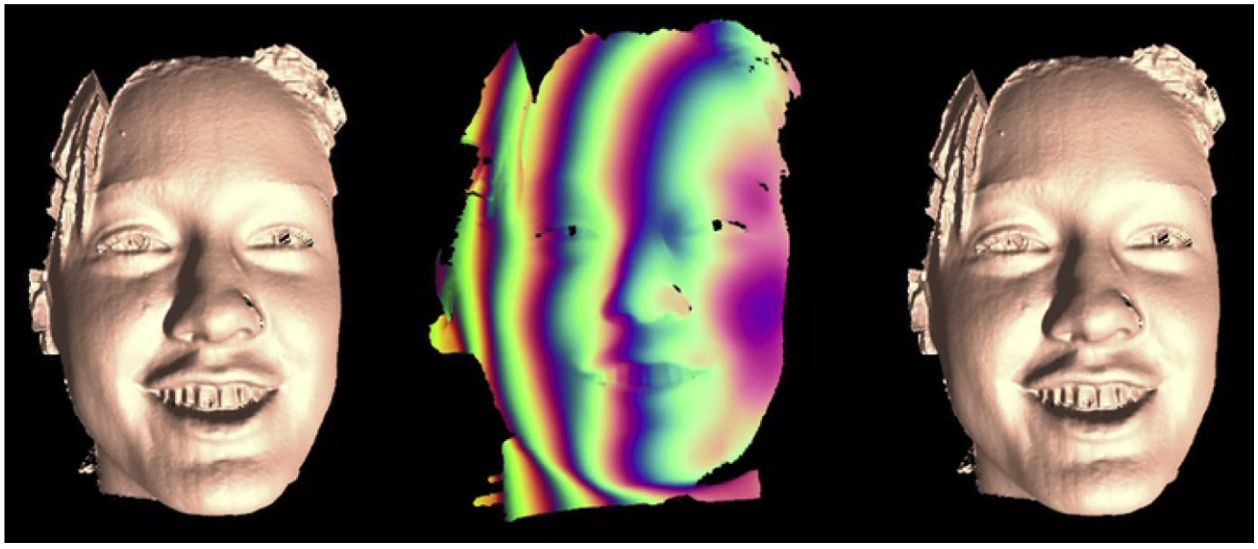


Fig. 6. 3D video compression result using the proposed Holovide technique. The left video shows the original scanned 3D video, the middle video shows the encoded Holovideo, and the right video shows the decoded 3D video.



Video S1. A video clip is available online. Supplementary material related to this article can be found online at doi:10.1016/j.optlaseng.2011.08.002.

Future work will entail researching different fringe patterns which fit into the YUV color space.

5. Conclusion

We have presented a technique which can encode and decode high-resolution 3D data in real-time, thus achieving 3D video. Decoding was performed at 28 FPS and encoding was performed at 17 FPS on an NVIDIA GeForce 9400 m GPU. Due to the design of the algorithm, standard 2D video codecs can be applied so long as they can encode in the RGB color space. Our results showed that a compression ratio of over 134:1 can be achieved in comparison with the OBJ file format. By using 2D video codecs to compress the geometry, existing research and infrastructure in 2D video can be leveraged in 3D.

Acknowledgments

Thank you to Amy Dixon for staring in the video, Ying Xu for performing scans, Laura Ekstrand for critiquing the writing, and the other members of the 3D Machine Vision Lab at Iowa State University for their help and suggestions.

Appendix A. Supplementary material

The following are the supplementary data to this article:
Video S1.

References

[1] Gorthi S, Rastogi P. Fringe projection techniques: whither we are? Opt Laser Eng 2010;48:133–40.
[2] Rusinkiewicz S, Hall-Holt O, Levoy M. Real-time 3D model acquisition. In: SIGGRAPH '02; 2002. p. 438–46.
[3] Zhang L, Snavely N, Curless B, Seitz SM. Spacetime faces: high resolution capture for modeling and animation. In: SIGGRAPH '04; 2004.
[4] Davis J, Ramamoorthi R, Rusinkiewicz S. Spacetime stereo: a unifying framework for depth from triangulation. IEEE PAMI 2005;27(2):1–7.
[5] Zhang S, Huang PS. High-resolution real-time three-dimensional shape measurement. Opt Eng 2006;45(12):123,601.
[6] Zhang S, Royer D, Yau S-T. GPU-assisted high-resolution, real-time 3-D shape measurement. Opt Express 2006;14(20):9120–9. (Selected for November 13, 2006 issue of The *Virtual Journal for Biomedical Optics*).
[7] Li Y, Zhao C, Qian Y, Wang H, Jin H. High-speed and dense three-dimensional surface acquisition using defocused binary patterns for spatially isolated objects. Opt Express 2010;18:21,628–35.
[8] Liu K, Wang Y, Lau DL, Hao Q, Hassebrook LG. Dual-frequency pattern scheme for high-speed 3-D shape measurement. Opt Express 2010;18:5229–44.
[9] Zhang S, van der Weide D, Olvier J. Superfast phase-shifting method for 3-D shape measurement. Opt Express 2010;18(9):9684–9. (Selected for July 6, 2010 issue of The *Virtual Journal for Biomedical Optics*).
[10] Wang Y, Zhang S. Superfast multifrequency phase-shifting technique with optimal pulse width modulation. Opt Express 2011;19(6):5143–8.
[11] Forstmann S, Ohya J, Krohn-Grimberghe A, McDougall R. Deformation styles for spline-based skeletal animation. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation; 2007. p. 141–50.
[12] Jones A, Lang M, Fyffe G, Yu X, Busch J, McDowall I, Bolas M, Debevec P. Achieving eye contact in a one-to-many 3D video teleconferencing system. In: SIGGRAPH '09; 2009.
[13] Kauff P, Atzpadin N, Fehn C, Muller M, Schreer O, Smolic A, Tanger R. Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability. Signal Proc: Image Commun 2007; 22(2):217–34.
[14] Gumhold S, Kami Z, Isenburg M, Seidel H-P. Predictive point-cloud compression. In: ACM SIGGRAPH 2005 Sketches; 2005. p. 137.
[15] Merry B, Marais P, Gain J. Compression of dense and regular point clouds. Comput Graphics Forum 2006;25(4):709–16.
[16] Schnabel R, Klein R. Octree-based point-cloud compression. In: Eurographics Symposium on Point-Based Graphics; 2006.
[17] Gu X, Zhang S, Zhang L, Huang P, Martin R, Yau S-T. Holoimages. In: ACM Solid and Physical Modeling, UK, 2006. p. 129–38.
[18] Karpinsky N, Zhang S. Composite phase-shifting algorithm for 3-D shape compression. Opt Eng 2010;49(6):063,604.
[19] Schreiber H, Bruning JH. Optical shop testing.3rd ed. New York, NY: John Willey & Sons; 2007. p. 547–666 [chapter 14].
[20] Ghiglia DC, Pritt MD. Two-dimensional phase unwrapping: theory, algorithms, and software. New York, NY: John Wiley and Sons, Inc; 1998.
[21] McGuire M. A fast, small-radius GPU median filter. In: ShaderX6; 2008.
[22] Zhang S, Yau S-T. Three-dimensional data merging using Holoimage. Opt Eng 2008;47(3):033,608. (Cover feature).
[23] Zhang S, Yau S-T. High-resolution, real-time 3-D absolute coordinate measurement based on a phase-shifting method. Opt Express 2006;14(7): 2644–9.