High-resolution, real-time three-dimensional shape measurement on graphics processing unit

Nikolaus Karpinsky
Morgan Hoke
Vincent Chen
Song Zhang

Optical Engineering 53(2), 024105 (February 2014)

High-resolution, real-time three-dimensional shape measurement on graphics processing unit

Nikolaus Karpinsky, Morgan Hoke, Vincent Chen, and Song Zhang*
Iowa State University, Department of Mechanical Engineering, Ames, Iowa 50011

Abstract. A three-dimensional (3-D) shape measurement system that can simultaneously achieve 3-D shape acquisition, reconstruction, and display at 30 frames per second (fps) with 480,000 measurement points per frame is presented. The entire processing pipeline was realized on a graphics processing unit (GPU) without the need of substantial central processing unit (CPU) power, making it achievable on a portable device, namely a laptop computer. Furthermore, the system is extremely inexpensive compared with similar state-of-art systems, making it possible to be accessed by the general public. Specifically, advanced GPU techniques such as multipass rendering and offscreen rendering were used in conjunction with direct memory access to achieve the aforementioned performance. The developed system, implementation details, and experimental results to verify the performance of the proposed technique are presented. *© 2014 Society of Photo-Optical Instrumentation Engineers (SPIE)* [DOI: 10.1117/1.OE.53.2.024105]

Keywords: structured light; fringe projection; multifrequency phase unwrapping; real-time; portable device.

Paper 131615P received Oct. 22, 2013; revised manuscript received Jan. 7, 2014; accepted for publication Jan. 15, 2014; published online Feb. 18, 2014.

1 Introduction

Recent advances in technology have enabled high-resolution, real-time three-dimensional (3-D) shape measurement through the use of structured light techniques.[1,2] (Note that real-time 3-D shape measurement includes 3-D shape acquisition, reconstruction, and display, all in real time.) Although these advances are impressive, they require large amounts of computing power, thus being limited to using large desktop workstations with high-end central processing units (CPUs) and sometimes graphics processing units (GPUs). This is undesirable in making high-resolution, real-time 3-D scanners ubiquitous in our mobile lives due to cost and portability issues. Recently, advancements in the speed of structured light-based techniques have pushed them into real-time without being computationally intensive and are more portable; this growth into real-time has given structured light scanners broad exposure, as can be seen with the introduction of the Microsoft Kinect. Though successful, the Kinect only provides sparse data points with low accuracy. Achieving high-resolution, real-time 3-D shape measurement on a low-end computer, such as a portable laptop, still remains challenging.

Similar to a stereo vision technique, structured light scanning works off of the principle of triangulation. Instead of using two cameras, such as the case of stereo vision, structured light scanning replaces a camera with a projector.[3] The projector projects a set of encoded patterns, which are used to establish a correlation between the projector and camera images, thus circumventing the correlation problem in stereo imaging. Assuming the system is calibrated, 3-D information can be triangulated using the established correspondence.[4]

Decoding patterns and performing triangulation is a computationally intensive task, making it difficult to reach real-time speeds using serial processing methods, as is seen with traditional CPUs. If the processes can be realized using parallel algorithms, parallel compute devices such as a GPU can be used to offload the computationally intensive problem.[5–9] Although the clock speed on a GPU is not as fast as a CPU, anywhere from 1 to 8 times slower, it can process hundreds of threads simultaneously, assuming there is no branching.[10] Gao and Kemao[11] provide a good review on parallel computing devices and their application in optical measurement. If the GPU can be leveraged on a portable device, then it has the potential to reach real-time 3-D shape measurement speeds even with the devices' limited computational power.

To establish the correlation between projector and camera pixels, the encoded patterns must be properly chosen to allow for parallel computation and maximum speed, yet be resilient to noise from things such as ambient light, surface reflections, etc. Many different techniques such as stripe boundary code,[12] binary coded patterns,[13] and phase shifting methods[14] exist. Although codification strategies such as binary coded patterns are parallel in nature, thus being well suited for GPU implementation, they typically require many patterns and are limited to the number of projector pixels. Fringe projection, on the other hand, uses sinusoidally varying fringe images, which no longer limits the measurement to the number of projector pixels, but requires spatial phase unwrapping.[15] Since phase unwrapping is typically a serial operation, it is not well suited to parallel implementation and difficult to achieve on a GPU.[7]

To address this issue, this work describes and demonstrates a real-time 3-D shape measurement system that is realized on a portable device, namely a laptop computer, which achieves a speed of 30 frames per second (fps) at an image resolution of 800×600. This technique is based on a modified two-frequency phase-shifting technique with binary defocusing.[16,17] The two-frequency phase-shifting algorithm enables pixel-by-pixel phase computation

*Address all correspondence to: Song Zhang, E-mail: song@iastate.edu

0091-3286/2014/$25.00 © 2014 SPIE

yet only requires a small number of fringe patterns (only 6); the proposed, modified two-frequency phase-shifting algorithm is also less sensitive to the noise caused by the two-frequency phase-shifting algorithm. To achieve high-resolution, real-time 3-D shape measurement on a portable device, our approach utilizes a GPU as a multipurpose parallel coprocessor. Through the use of the OpenGL Shading Language (GLSL) we have created a structured light processing pipeline that is implemented solely in parallel on the GPU. This reduces the processing power requirements of the device performing 3-D reconstruction, allowing us to realize the system with a portable device, namely a laptop computer. To mitigate high-speed camera transfer problems, which typically require a dedicated frame grabber, we make use of USB 3.0 along with direct memory access (DMA) to transfer camera images to the GPU without the need of synchronization between the CPU and GPU. In addition to low processing power requirements, since the entire system is realized on the GPU, the CPU is nearly ideal, thus freeing it perform other operations in parallel to the 3-D reconstruction, such as 3-D registration or feature extraction. We developed a low-cost system (less than \$3,000 including the laptop computer) that can achieve 30-fps measurement speed with 480,000 points per frame.

Section 2 of this article explains the principles of the techniques employed by the system. Section 3 breaks the implementation of the system down into stages and discusses how each stage is achieved on the GPU. Section 4 shows the experimental results to demonstrate the success of the proposed techniques. Finally, Sec. 5 summarizes the article.

2 Principle

2.1 *Two-Frequency Phase-Shifting Algorithm*

The structural patterns used by our system are a set of three-step phase-shifted fringe patterns. A phase-shifting method was chosen over other structured light coding techniques due to it only requiring a few patterns for codification resulting in high speed, it not being limited by projector resolution, and its resilience to noise.[15] Three-step phase-shifted patterns can be described by

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

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

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

Here, I' is the average intensity, I'' is the intensity modulation, and ϕ is the encoded phase. Using these equations, the phase ϕ can be solved by

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

This equation yields a phase value ϕ for every pixel, but since the $\tan^{-1}$ only ranges from 0 to 2π, the phase value provided will have 2π phase discontinuities; this phase value is known as wrapped phase. Conventional approaches employ a spatial phase unwrapping algorithm that traverses along the phase map adding multiples of 2π,[18] but this is a serial operation that requires neighboring pixel information thus being undesirable for GPU implementation.

Instead, we adopted a two-frequency phase-shifting algorithm to temporally unwrap the phase pixel by pixel, which is well suited for GPU implementation.[19,20] For the temporal phase unwrapping algorithm, two frequencies are the minimum number needed to unwrap, although more[21] can be utilized for higher-quality phase; to achieve real-time speeds, our implementation chose only two, thus requiring six fringe images. Briefly, the two-frequency phase-shifting algorithm works as follows: we obtain the phase map ϕ_1 from one set of phase-shifted fringe patterns with frequency of f_1 (or fringe period of T_1), and phase map ϕ_2 from the second set of phase-shifted fringe patterns with a different frequency of f_2 (or fringe period of T_2). Instead of using fringe frequencies or fringe periods, conventionally, the "wavelength" was used because such an algorithm was developed for laser interferometers where the wavelength λ of the laser light has a physical meaning, and can uniquely determine the interfered fringe pattern. However, in a digital fringe projection (DFP) system, the fringe patterns are directly generated by a computer, and the wavelength of light used does not have any correlation with the fringe patterns generated. Therefore, in this article, we use fringe period or fringe frequency instead. The fringe period is defined as the number of pixels per period of the fringe.

By utilizing the phase difference of the phases ϕ_1 and ϕ_2 with different fringe frequencies and the modus operation, an equivalent phase map can be obtained

$$\phi_{\mathrm{eq}} = \phi_1 - \phi_2 \bmod 2\pi. \tag{5}$$

This resultant phase map has a different fringe period from T_1 and T_2, which is usually called the equivalent fringe period, T_{eq}, that can be determined by

$$T_{\mathrm{eq}} = \frac{T_1 T_2}{|T_1 - T_2|}. \tag{6}$$

By properly selecting the spatial frequencies of the phases ϕ_1 and ϕ_2, a continuous phase ϕ_{eq}, that spans the entire phase map without 2π discontinuities can be achieved. The equivalent phase map ϕ_{eq} can then be utilized to unwrap the phase ϕ_1 or ϕ_2 point by point by

$$k(x,y) = \mathrm{round}\left[\frac{\phi_{\mathrm{eq}}(x,y) \times T_{\mathrm{eq}}/T_1 - \phi_1(x,y)}{2\pi}\right], \tag{7}$$

$$\Phi(x,y) = \phi_1(x,y) + 2\pi \times k(x,y). \tag{8}$$

This two-frequency phase-shifting approach can obtain an unwrapped absolute phase map Φ per pixel in parallel; thus it is well suited for GPU implementation. Figure 1 shows a set of captured fringe images with two fringe periods, $T_1 = 60$ and $T_2 = 66$ pixels. Their corresponding wrapped and unwrapped phase maps are shown in Figs. 1(c)–1(f).

2.2 *General Purpose GPU*

Recently, in order to accelerate parallel tasks on a computer, general purpose GPU (GPGPU) computation has been

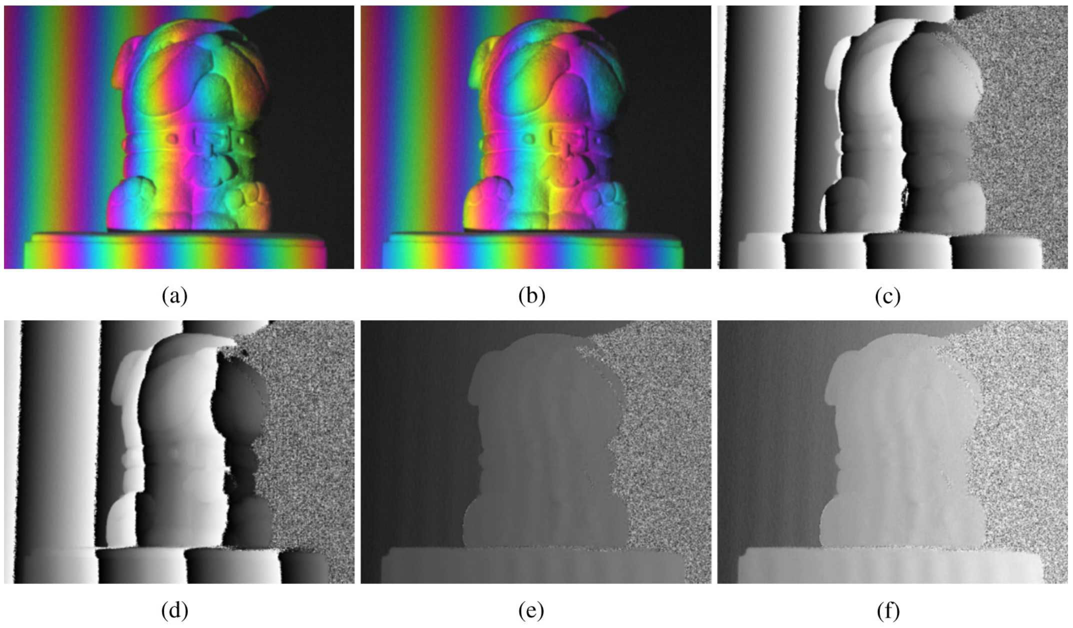


Fig. 1 Two-frequency phase unwrapping process. (a) Three fringe images of smaller phase; (b) three fringe images of larger phase; (c) wrapped phase ϕ_1; (d) wrapped phase ϕ_2; (e) equivalent phase ϕ_{eq}; (f) unwrapped phase Φ for ϕ_1.

leveraged. The main goal of GPGPU computation is to free the CPU of a computer from parallel intensive tasks by leveraging the GPU as a parallel coprocessor.[7] Although not having nearly as high a clock speed as modern CPUs, GPUs have many more processing cores, typically on the scale of hundreds to thousands. To leverage this technology in applications such as 3-D reconstruction, different programming interfaces can be used, such as NVIDIA CUDA,[22] OpenCL,[23] or the GLSL.[24]

While GPGPU programming interfaces such as CUDA and OpenCL offer lower level hardware access to features such as shared memory, they are only beginning to be supported in portable and mobile devices with platforms such as the NVIDIA Tegra. Conversely, GLSL is supported on nearly all modern graphics devices, is part of the OpenGL ES specification,[25] and is better for interoperability with a graphics application programming interface.[26] In terms of performance, CUDA and OpenCL have been shown to be marginally faster than GLSL assuming efficient memory access.[27,28] Due to interoperability and only minimal performance loss, GLSL was the chosen GPGPU programming interface for our implementation.

In order to use GLSL for GPGPU computation versus traditional computer graphics applications, certain techniques need to be leveraged: offscreen rendering, DMA, and multipass rendering. Offscreen rendering allows OpenGL scenes to be rendered into buffers other than the standard frame buffer or screen. This is done by creating a frame buffer object (FBO) and binding its output to the desired output buffer, such as an OpenGL texture. When geometry is rendered through the pipeline, it will output into the texture versus the screen. By rendering a screen aligned quad with the FBO bound, a GLSL fragment program can be run for every pixel in the output buffer, allowing per-pixel computation.[29]

In order to get input into the GLSL program, buffers such as textures are bound that the program can access. When using GPUs, one of the major bottlenecks is transfer of data to and from the GPU buffers. To alleviate this bottleneck, we use DMA, which allows specifically allocated parts of memory to be used in transfers.[30] Transfers through DMA do not require the CPU and GPU to synchronize, and the GPU can transfer data while simultaneously processing its pipeline. Thus, utilizing a DMA approach mitigates the bottleneck of transfers.

Lastly, multipass rendering is leveraged to run different GLSL programs multiple times on different buffers, achieving synchronization of threads between multiple stages in a pipeline. By clearing or not using depth buffering, the OpenGL driver will not cull any geometry from the processing pipeline. Transforms on the data can be utilized by binding different input and output buffers as well as different GLSL programs in between rendering screen aligned quads. This allows previously computed data to be utilized in future stages as well as compounding effects and is known as multipass rendering, since multiple rendering passes are used to render a single scene.

3 Implementation

In the 3-D decoding pipeline for our system, there are seven discrete stages: phase wrapping, phase filtering, phase unwrapping, phase filtering, depth map calculation, normal map calculation, and final rendering. Figure 2 illustrates the decoding pipeline, showing the data at each step. This section will look into the implementation of each step.

3.1 *Step 1: Phase Wrapping*

Phase wrapping is the first step in the overall pipeline that takes incoming fringe data from the camera and wraps it into wrapped phase maps. Each set of three step fringe images are passed in as a texture, with the three images in the red, green, and blue color channel. Next, Eq. (4) is applied to each image resulting in two-wrapped phase

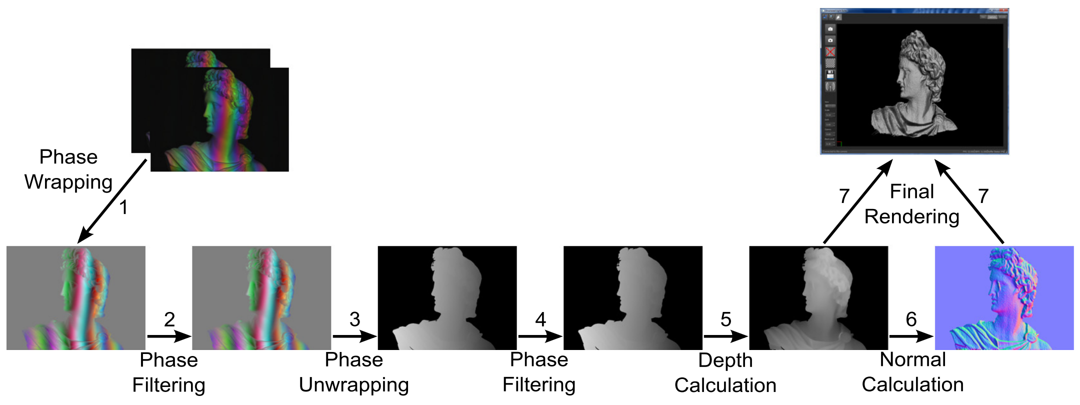


Fig. 2 Real-time three-dimensional (3-D) scanning pipeline. The pipeline starts with the fringe images (packed into the RGB color channels of the fringe images presented) and streams them to the GPU via direct memory access transfers. Next, phase wrapping is performed, followed by Gaussian phase filtering, phase unwrapping, median phase filtering, depth calculation, and normal calculation. Finally, final rendering is performed using the depth map and normal map producing the 3-D scan.

maps, ϕ_1 and ϕ_2, one for each frequency. At this point, the sine and cosine components of the phases are taken out, and each component is rendered out to the output texture color channels (r, g, b, a) as

$$r = \sin(\phi_1), \tag{9}$$

$$g = \cos(\phi_1), \tag{10}$$

$$b = \sin(\phi_2), \tag{11}$$

$$a = \cos(\phi_2). \tag{12}$$

The components are extracted so that during phase filtering, errors are not introduced into the wrapped phase map.

3.2 *Step 2: Gaussian Phase Filtering*

Phase filtering is the only stage of the pipeline that can have a variable number of steps, since it depends on how the unwrapped phase should be filtered. In our experimental pipeline, we performed one pass of a separable 11×11 Gaussian filter; this is done to smooth out high-frequency phase noise. If larger kernels are needed, multiple passes of a small kernel can be utilized, but in our analysis, it was faster to perform just a single 11×11 kernel. The separable Gaussian filter requires two rendering passes, one for the vertical pass and one for the horizontal pass of the Gaussian kernel. By using a separable Gaussian filter, only 22 texture lookups are required, 11 for horizontal and 11 for vertical. If a nonseparable kernel were used, 121 texture lookups would be required, substantially slowing down filtering.

3.3 *Step 3: Phase Unwrapping*

The phase unwrapping stage involves taking the filtered wrapped phase components and combining them into an unwrapped phase map. To recover ϕ_1 and ϕ_2, the $\tan^{-1}$ is applied to the filtered sine and cosine components.

$$\phi = \tan^{-1}\left[\frac{\sin\,\phi}{\cos\,\phi}\right]. \tag{13}$$

At this point, the equivalent phase ϕ_{eq} can be calculated using Eq. (5). If the fringe periods T_1 and T_2 are properly chosen, such that the equivalent fringe period T_{eq} is large enough that the single fringe spans the entire image, ϕ_1 can be unwrapped with Eq. (8). Choosing these optimal fringe periods is typically not easy,[31] since if these two fringe periods are too close, the equivalent phase map will be very noisy, making it difficult to resolve the phase steps.[32]

The two-frequency phase-shifting algorithm works well for high-end systems, where the sensor noise is small. To mitigate the noise-induced phase unwrapping problems for our low-end system, we chose fringe periods that result in the equivalent phase only spanning half the image (i.e., it is ~50% less sensitive to noise effects). This selection, of course, will introduce a 2π phase jump. However, unlike the convention phase map where there are many 2π jumps per line perpendicular to the fringe direction, the proposed method only has a single phase jump at its maximum. Therefore, it is not necessary to adopt a complex spatial phase unwrapping algorithm to unwrap the phase map. Instead, to correct for the phase jump, we employ a parallel phase unwrapping method that unwraps the phase based on a phase value and its location. Specifically, we start with capturing the phase maps of a flat plane at two extreme depth locations, Z_{min} and Z_{max}, the minimum and maximum depth values respectively, and then plotting a cross-section yielding Fig. 3(a). As can be seen, there is a gap between the min and max phase jump that can be best separated by a line. The equation of this line is

$$y = \frac{2\pi}{P} \times x + b_\phi, \tag{14}$$

where P is the number of camera pixels per period of the fringe, and b_ϕ is the y intercept found though fitting the line between the phase of Z_{min} and Z_{max}. Using this equation, phase values below this line should have 2π added to them to correct for the jump, and phase values above do not. The resulting unwrapping is shown with Fig. 3(b). By adopting this proposed phase unwrapping algorithm, fringe

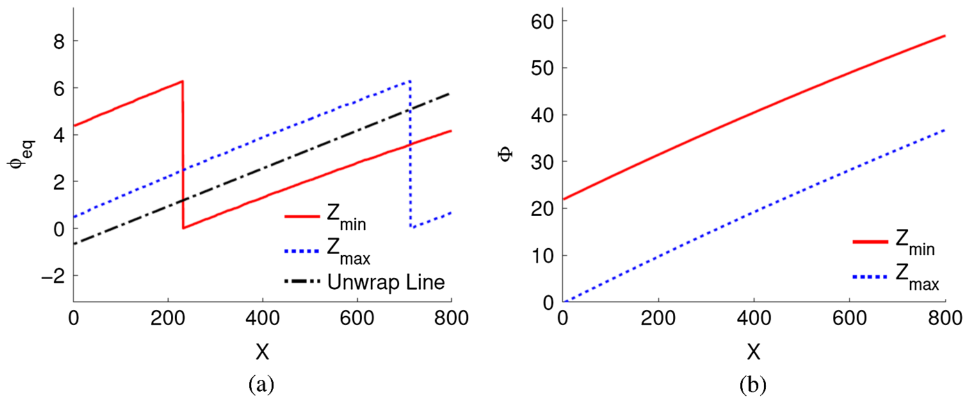


Fig. 3 Cross-section of captured phase before and after phase unwrapping. (a) Cross-section of wrapped phase ϕ_{eq}; red denotes the phase at Z_{min}, blue the phase at Z_{max}, and the black line is the phase unwrapping line; (b) cross-section of unwrapped phase after unwrapping ϕ_{eq} with the phase unwrapping line, and using Φ_{eq} to unwrap ϕ_1.

periods with a larger difference can be selected, reducing the overall noise.

3.4 *Step 4: Median Phase Filtering*

Again, similar to Step 2 (phase filtering), this stage can have a variable number of steps since it depends on how many passes of the filter are needed for the implementation. In our implementation, we performed a single pass of a specialized median filter[33] that removes one or two pixels spiking noise due to incorrect phase unwrapping that could be caused by motion or system noise. In this research, we adopted the median filter with a size of 1×5 that operated in the direction of the phase gradient, reducing the number of comparisons and branches required. Once the median for a pixel is calculated, the delta between the median phase ϕ_m and actual phase ϕ_a is taken and rounded after dividing by 2π. This results in an integer number k that can be determined by

$$k = \text{round}\left[\frac{\phi_a - \phi_m}{2\pi}\right]. \tag{15}$$

The nonzero k indicates a spiking point that can be corrected by subtracting the k number of 2π. Our research found that this filtering stage effectively removes spiking noise yet will not introduce artifacts caused by standard median filtering.

3.5 *Step 5: Depth Calculation*

Depth map calculation involves calculating depth values for each unwrapped phase value. There are a number of approaches[4,34–43] to do this based on the chosen calibration, and in our method, we chose to perform a very simple reference-plane-based approach detailed by Xu et al.[44] By capturing the phase of a reference plane Φ_R where $z = 0$, a phase difference between the captured phase Φ_C and Φ_R can be calculated. This phase difference will be proportional to the depth z by a scaling value. To calculate this in the fragment shader, a texture containing Φ_R is read in along with a texture containing the filtered phase Φ_C. Subtracting the two phases and scaling, based on a scaling factor c determined through calibration, yields the depth value z; this depth value is then rendered out, yielding the depth map.

3.6 *Step 6: Normal Map Calculation*

During the reconstruction, point normals for the geometry are calculated so that Phong lighting may be applied. The normal map is calculated by calculating all adjacent surface normals and then averaging them together, resulting in a point normal. Adjacent surface normals are calculated by taking the vectors between the current coordinate and two neighboring coordinates, moving sequentially counterclockwise in a 3×3 neighborhood and calculating the cross product. This yields a surface normal for the polygon composed of these three points. After normalizing and averaging all these surface normals, the result is the point normal for the coordinate. This is rendered out to the normal map texture, yielding a normal map for the scanned data.

3.7 *Step 7: Final Rendering*

The last stage in the 3-D scanning pipeline is final rendering. Before final rendering can take place, the frame buffer needs to be switched back from the FBO to the screen so that the result is rendered to the screen. After doing so, the depth map and normal map are passed to the final render shader, and a plane of points is rendered with uniformly varying texture coordinates. At this point, the final geometry can be downsampled, if needed, by rendering a reduced number of points in the plane of points. In the vertex shader, the depth for the point is read from the depth map and the vertex z attribute is modified accordingly. In the fragment shader, the point normal is read from the normal map, and then per-fragment Phong shading is applied to each point. At this point, the reconstructed geometry is rendered onto the screen.

4 Experimental Results and Discussion

To test the effectiveness of the system, we performed different experiments including measuring a flat surface, measuring static sculptures, and measuring dynamically moving objects. In each of the experiments the hardware stayed consistent, a Point Grey Research Flea3 camera with a Computar 12-mm lens, a Texas Instruments (Dallas, Texas) digital light processing (DLP) LightCrafter projector, an Arduino Uno for timing signal generation, and a IBM Lenovo laptop with a Intel i5 3320M 2.6-GHz CPU and NVIDIA (Santa Clara, California) Quadro NVS5400M GPU. The DLP Lightcrafter can project and switch binary structured patterns

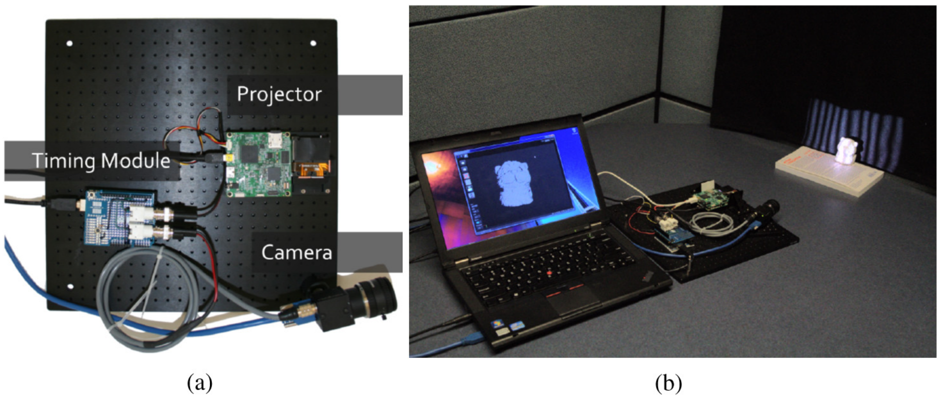


Fig. 4 Picture of the developed system. (a) Overview of the system with labeled components; (b) system scanning static sculpture.

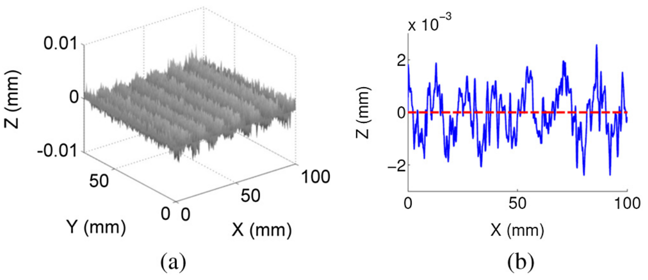


Fig. 5 Measurement result of a flat surface. The measured area is $\sim 100 \times 75$ mm^2, and the resulting root-mean-square error is 0.00081 mm. (a) 3-D plot of the surface; (B) example cross-section of the surface.

at 4 kHz with full resolution, and the Point Grey camera can acquire images up to 180 Hz with an image resolution of 800×600 and exposure time of <2 ms operating under the external triggering mode. Figure 4 shows an overview of the system with labeled components as well as the system scanning a sculpture.

Since under the external triggering mode, the maximum exposure time of the camera can use is 2 ms at 180-Hz capturing rate, the conventional sinusoidal fringe projection technique does work due to the rigorous timing requirement (i.e., the camera exposure time must be precisely $1000/180$ ms). Therefore, the binary defocusing technique[16] was used to circumvent this problem. Using defocused binary patterns, each pixel of the DMD is either on or off and no time modulation is used; thus the camera only needs to capture a specific slice of the projector exposure reducing the rigid timing constraints. Furthermore, to alleviate the short depth range problem caused by square binary defocusing technique, we adopted the error-diffusion dithering method to generate all six desired sinusoidal fringe patterns[17] and the modified two-frequency phase-shifting algorithm to unwrap the phase pixel by pixel. For such a system, since it requires six fringe images to recover one 3-D shape, the 3-D shape measurement rate is 30 fps.

To test the system noise, we first captured a flat surface. In an ideal system, the surface should be perfectly flat, but due to sensor and environment noise there are small variations. Figure 5(a) shows the results of the capture, and Fig. 5(b) shows a horizontal cross-section at the 300th row. The variations in the surface height results in a root-mean-square error of 0.00081 mm, for a measurement area of 100×75 mm^2 with a resolution of 800×600.

To test the system's capabilities of measuring more complex 3-D objects, we performed measurements on a static sculpture. Figure 6(a) shows the statue we captured, and the surface geometry is pretty complex. The live reconstructed 3-D results on the computer screen are shown in Figs. 6(b) and 6(c). These results clearly show there are no phase jumps, verifying that the proposed phase unwrapping algorithm works properly. These images are also very clean without spiking noise, common to a multifrequency phase-shifting algorithm, meaning that the proposed filtering methods can effectively remove spiking noise.

To further demonstrate the speed of the proposed system, we measured some dynamically changing objects. Figure 7

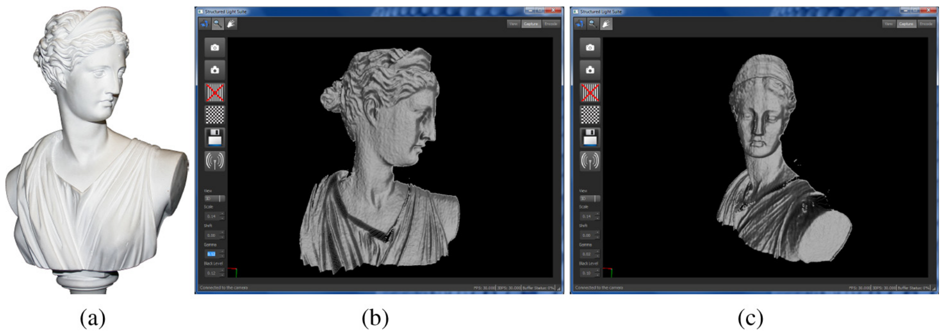
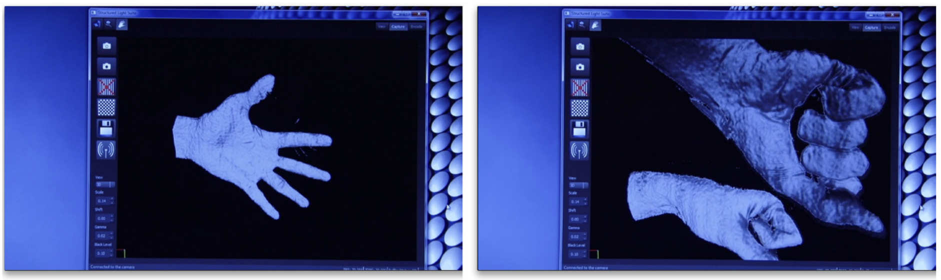


Fig. 6 Capture of static statues. (a) Two-dimensional photo of statue; (b) and (c) two screen shots of the 3-D reconstructions of the statue.



(a) (b)

Fig. 7 Example frames from capturing dynamically moving objects. (a) A single hand motion (Media 1, 4.9 MB) [URL: http://dx.doi.org/10.1117/1.OE.53.2.024105.1]; (b) two hands motion (Media 2, 3.0 MB) [URL: http://dx.doi.org/10.1117/1.OE.53.2.024105.2].

and the associated media show two examples, one single hand motion capture, and simultaneous two-hand motion capture. The videos were filmed from the computer screen using an high definition (HD) video recorder (Sony, Tokyo, Japan: HDR-AX2000) so as not to affect the reconstruction frame rate. The computer screen was the Lenovo laptop screen that demonstrates the live-reconstructed 3-D results of the object being measured. Regardless of the geometry complexity, this laptop can constantly reconstruct and display 3-D geometry at 30 fps with an image resolution of 800×600. These experiments once again confirm that the proposed GPU phase-shifting algorithm and portable system can deliver high-resolution, real-time 3-D shape measurement of dynamically deformable objects with arbitrary shapes.

5 Conclusion

This article has presented a technique for achieving high-resolution, real-time 3-D shape measurement on a portable device by implementing the entire processing pipeline of a modified two-frequency phase-shifting algorithm on a GPU. We have demonstrated the principles behind the techniques leveraged by the system, as well as giving a description of the GPU implementation. By utilizing a GPU for the entire 3-D processing and display process, the processing power requirements of CPU have been drastically reduced, allowing the system to be realized with a portable device. Through experiments, we have shown that 3-D shape acquisition, reconstruction, and display can reach 30 fps on a Lenovo laptop at an image resolution of 800×600. Utilizing the binary defocusing technique, the USB 3.0 camera, and the GPU implementation, the whole system is quite inexpensive, making such a system potentially accessible to the general public.

Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) Graduate Research Fellowship under Grant No. DGE1247194, and the NSF under grant numbers CMMI-1150711 and CMMI-1300376. Any opinion, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the NSF.

References

1. S. Zhang, D. Royer, and S.-T. Yau, "GPU-assisted high-resolution, real-time 3-D shape measurement," *Opt. Express* **14**(20), 9120–9129 (2006).
2. K. Liu et al., "Dual-frequency pattern scheme for high-speed 3-D shape measurement," *Opt. Express* **18**(5), 5229–5244 (2010).
3. J. Salvi, J. Pages, and J. Batlle, "Pattern codification strategies in structured light systems," *Pattern Recognit.* **37**(4), 827–849 (2004).
4. S. Zhang and P. S. Huang, "Novel method for structured light system calibration," *Opt. Eng.* **45**(8), 083601 (2006).
5. L. Ahrenberg et al., "Using commodity graphics hardware for real-time digital hologram view-reconstruction," *J. Disp. Technol.* **5**(4), 111–119 (2009).
6. A. Espinosa-Romero and R. Legarda-Saenz, "GPU based real-time quadrature transform method for 3-D surface measurement and visualization," *Opt. Express* **19**(13), 12125–12130 (2011).
7. W. Gao et al., "Real-time pipelined heterogeneous system for windowed fourier filtering and quality guided phase unwrapping algorithm using graphic processing unit," in *AIP Conf. Proc.*, Vol. 1236, pp. 129–134 (2010).
8. J. Carpenter and T. D. Wilkinson, "Graphics processing unit–accelerated holography by simulated annealing," *Opt. Eng.* **49**(9), 095801 (2010).
9. H. Kang et al., "Acceleration method of computing a compensated phase-added stereogram on a graphic processing unit," *Appl. Opt.* **47**(31), 5784–5789 (2008).
10. S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *Int. Conf. Field Programmable Logic Appl.*, pp. 126–131, IEEE, Prague, Czech Republic (2009).
11. W. Gao and Q. Kemao, "Parallel computing in experimental mechanics and optical measurement: a review," *Opt. Lasers Eng.* **50**(4), 608–617 (2012).
12. S. Rusinkiewicz, O. Hall-Holt, and M. Levoy, "Real-time 3D model acquisition," in *SIGGRAPH '02 Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 438–446, ACM, San Antonio, TX (2002).
13. I. Ishii et al., "High-speed 3D image acquisition using coded structured light projection," in *IEEE/RSJ Int. Conf. Intell. Robots Syst IROS 2007*, pp. 925–930, IEEE, San Diego, CA (2007).
14. S. Zhang and S.-T. Yau, "High-resolution, real-time 3D absolute coordinate measurement based on a phase-shifting method," *Opt. Express* **14**, 2644–2649 (2006).
15. S. Zhang, "Recent progresses on real-time 3D shape measurement using digital fringe projection techniques," *Opt. Lasers Eng.* **48**(2), 149–158 (2010).
16. S. Lei and S. Zhang, "Flexible 3-D shape measurement using projector defocusing," *Opt. Lett.* **34**(20), 3080–3082 (2009).
17. Y. Wang and S. Zhang, "Superfast multifrequency phase-shifting technique with optimal pulse width modulation," *Opt. Express* **19**, 5149–5155 (2011).
18. D. C. Ghiglia and M. D. Pritt, *Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software*, Wiley, New York (1998).
19. K. Creath, "Phase-measurement interferometry techniques," *Prog. Opt.* **26**(26), 349–393 (1988).
20. C. Joenathan, "Phase-measuring interferometry: new methods and error analysis," *Appl. Opt.* **33**(19), 4147–4155 (1994).
21. Y.-Y. Cheng and J. C. Wyant, "Multiple-wavelength phase shifting interferometry," *Appl. Opt.* **24**(6), 804–807 (1985).
22. NVIDIA CUDA Compute Unified Device Architecture—Programming Guide, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (2007).

23. A. Munshi et al., "The openCL specification," Khronos OpenCL Working Group, pp. 11–15, https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf (2009).
24. R. J. Rost, J. M. Kessenich, and B. Lichtenbelt, *Open GL: Shading Language*, Addison-Wesley Professional (2004).
25. A. Munshi, *Opengl es Common Profile Specification 2.0*, Khronos Group, http://www.khronos.org/registry/gles/specs/2.0/es_cm_spec_2.0.24.pdf (2007).
26. J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of cuda and opencl," in *Int. Conf. Parallel Proces. (ICPP)*, pp. 216–225, IEEE, Taipei, Taiwan (2011).
27. R. Amorim et al., "Comparing cuda and opengl implementations for a Jacobi iteration," in *Int. Conf. High Perform. Comput. Simul. HPCS'09*, pp. 22–32, IEEE, Leipzig, Germany (2009).
28. T. I. Vassilev, "Comparison of several parallel api for cloth modelling on modern GPUs," in *Proc. 11th Int. Conf. Comput. Syst. Technol. Workshop PhD Students Comput. Int. Conf. Comput. Syst. Technol.*, pp. 131–136, ACM, Sofia, Bulgaria (2010).
29. J. Fung and S. Mann, "Computer vision signal processing on graphics processing units," in *IEEE Int. Conf. Acoust., Speech, Signal Proces. Proc. (ICASSP'04).*, Vol. 5, pp. V-93, IEEE, Montreal, Canada (2004).
30. D. Shreiner et al., *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Addison-Wesley Professional, Boston, MA (2009).
31. C. E. Towers, D. P. Towers, and J. D. Jones, "Optimum frequency selection in multifrequency interferometry," *Opt. Lett.* **28**(11), 887–889 (2003).
32. K. Creath, "Step height measurement using two-wavelength phase-shifting interferometry," *Appl. Opt.* **26**(14), 2810–2816 (1987).
33. M. McGuire, "A fast, small-radius GPU median filter," in ShaderX6, http://www.amazon.com/ShaderX6-Rendering-Techniques-Wolfgang-Engel/dp/1584505443 (2008).
34. Y. Xiao, Y. Cao, and Y. Wu, "Improved algorithm for phase-to-height mapping in phase measuring profilometry," *Appl. Opt.* **51**(8), 1149–1155 (2012).
35. Y. Villa et al., "Transformation of phase to (x,y,z)-coordinates for the calibration of a fringe projection profilometer," *Opt. Laser Eng.* **50**(2), 256–261 (2012).
36. Y. Wen et al., "Universal calculation formula and calibration method in fourier transform profilometry," *Appl. Opt.* **49**(34), 6563–6569 (2010).
37. R. Legarda-Sáenz, T. Bothe, and W. P. Jüptner, "Accurate procedure for the calibration of a structured light system," *Opt. Eng.* **43**(2), 464–471 (2004).
38. F. J. Cuevas et al., "Multi-layer neural networks applied to phase and depth recovery from fringe patterns," *Opt. Commun.* **181**(4), 239–259 (2000).
39. Z. Li et al., "Accurate calibration method for a structured light system," *Opt. Eng.* **47**(5), 053604 (2008).
40. W. Gao, L. Wang, and Z. Hu, "Flexible method for structured light system calibration," *Opt. Eng.* **47**(8), 083602 (2008).
41. R. Yang, S. Cheng, and Y. Chen, "Flexible and accurate implementation of a binocular structured light system," *Opt. Lasers Eng.* **46**(5), 373–379 (2008).
42. Q. Hu et al., "Calibration of a 3-D shape measurement system," *Opt. Eng.* **42**(2), 487–493 (2003).
43. L. Huang, P. S. K. Chua, and A. Asundi, "Least-squares calibration method for fringe projection profilometry considering camera lens distorsion," *Appl. Opt.* **49**(9), 1539–1548 (2010).
44. Y. Xu et al., "Phase error compensation for three-dimensional shape measurement with projector defocusing," *Appl. Opt.* **50**(17), 2572–2581 (2011).

Nikolaus Karpinsky is a PhD student in human computer interaction (HCI) and computer engineering at Iowa State University. He received his BS in software engineering from the Milwaukee School of Engineering in 2009 and his MS in HCI and computer engineering from Iowa State University in 2011. His current research interests include parallel computing, high-speed 3-D scanning, 3-D video compression, and computer vision.

Song Zhang is the William and Virginia Binger assistant professor of mechanical engineering at Iowa State University. He received his PhD degree from Stony Brook University in 2005. His major research interests include high-speed 3-D optical metrology, 3-D information processing, 3-D biophotonics imaging, and human computer interaction. He is the recipient of NSF CAREER award in 2012, and a fellow of SPIE.

Biographies of the other authors are not available.