# GPU-assisted high-resolution, real-time 3-D shape measurement

**Song Zhang**
*Mathematics Department, Harvard University, One Oxford Street, Cambridge, MA 02138*
szhang77@gmail.com

**Dale Royer**
*Geometric Informatics Inc, Somerville, MA 02143*

**Shing-Tung Yau**
*Mathematics Department, Harvard University, One Oxford Street, Cambridge, MA 02138*

**Abstract:** This paper describes a Graphics Processing Unit (GPU)-assisted real-time three-dimensional shape measurement system. Our experiments demonstrated that the absolute coordinates calculation and rendering speed of a GPU is more than four times faster than that of a dual CPU workstation with the same graphics card. By implementing the GPU into our system, we realized simultaneous absolute coordinate acquisition, reconstruction and display at 30 frames per second with a resolution of approximately 266K points per frame. Moreover, a 2+1 phase-shifting algorithm was employed to alleviate the measurement error caused by motion. Applications of the system include medical imaging, manufacturing, entertainment, and security.

**OCIS codes:** (110.6880) Three-dimensional image acquisition; (120.5800) Scanners; (150.6910) Three-dimensional sensing; (120.6650) Surface measurements, figure; (100.2650) Fringe analysis; (120.3940) Metrology; (120.5050) Phase measurement.

## References and links

1. M. Takeda and K. Mutoh, "Fourier Transform Profilometry for the Automatic Measurement of 3-D Object Shape," Appl. Opt. **22**, 3977–3982 (1983).
2. S. Almazán-Cuéllar and D. Malacara-Hernández, "Two-step Phase-shifting Algorithm," Opt. Eng. **42**, 3524–3531 (2003).
3. C. Quan, C. J. Tay, X. Kang, X. Y. He, and H. M. Shang, "Shape Measurement by Use of Liquid-crystal Display Fringe Projection with Two-step Phase-Shifting," Appl. Opt. **42**, 2329–2335 (2003).
4. P. S. Huang and S. Zhang, "A Fast Three-Step Phase Shifting Algorithm," Appl. Opt. **45** (2006).
5. S. Zhang and P. S. Huang, "High-Resolution, Real-time 3-D Shape Measurement," Opt. Eng. (2006). In Press.
6. S. Zhang and S.-T. Yau, "High-resolution, Real-time 3D Absolute Coordinate Measurement Based on a Phase-shifting Method," Opt. Express **45** (2006).
7. J. R. P. Angel and P. L. Wizinowich, "A Method of Phase Shifting in the Presence of Vibration," in *ESO Proc.*, vol. 30 (1988).
8. P. L. Wizinowich, "Phase Shifting Interferometry in the Presence of Vibration: A New Algorithm and System," Appl. Opt. **29**, 3271–3279 (1990).
9. J. C. Wyant, "Phase Shifting Interferometry," http://www.optics.arizona.edu/jcwyant/Optics513/optics513.htm (1998).
10. M. Ujaldon and J. Saltz, "Exploiting Parallelism on Irregular Applications Using the GPU," in *Intl. Conf. on Paral. Comp.*, pp. 13–16 (2005).
11. B. Khailany, W. Dally, S. Rixner, U. Kapasi, J. Owens, and B. Towles, "Exploring the VLSI Scalability of Stream Processors," in *Proc. 9th Symp. on High Perf. Comp. Arch.*, pp. 153–164 (2003).

#72476 - $15.00 USD Received 30 June 2006; revised 29 August 2006; accepted 30 August 2006

12. E. Lindholm, M. J. Kligard, and H. Moreton, "A user-programmable vertex engine," in *Proc. of SIGGRAPH*, pp. 149–158 (2001).
13. R. Fernando and M. J. Kilgard, *The Cg Tutorial* (Addison-Wesley, Boston, 2003).
14. D. Malacara, ed., *Optical Shop Testing* (John Wiley and Songs, NY, 1992).
15. S. Zhang, "High-Resolution, Real-Time 3-D Shape Measurement," Ph.D. thesis, Stony Brook University, State University of New York (2005).
16. D. C. Ghiglia and M. D. Pritt, *Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software* (John Wiley and Sons, Inc, 1998).
17. S. Zhang, X. Li, and S.-T. Yau, "Multi-level Quality-Guided Phase Unwrapping Algorithm for Real-time 3-D Reconstruction," Appl. Opt. (2006). In Revision.
18. S. Zhang and P. S. Huang, "A Novel Structured Light System Calibration," Opt. Eng. **45** (2006).

## 1. Introduction

High-resolution, real-time 3-D measurement is increasingly important with applications in medical imaging, virtual reality, computer vision, computer graphics, etc. With the improvements of the measurement hardware system, the data acquisition speed is faster and faster. Viewing the measurement results interactively is highly desirable.

In general, for real-time measurement, the fewer fringe images used, the faster the speed achieved. Algorithms based on single fringe image [1] or double fringe images [2, 3] can achieve faster data acquisition speed than an algorithm based on three fringe images. However, since the phase cannot be resolved directly from the single or dual fringe images, the measurement normally has strict requirements for the measurement environment or the complexity of the geometric shape of the objects. In contrast, a three-step phase-shifting algorithm is able to solve the phase uniquely from the fringe images. Therefore, for high-speed, accurate measurement of complex geometric shapes, a three-step phase-shifting algorithm is the first choice. By utilizing a fast three-step phase-shifting algorithm [4], Zhang and Huang successfully developed a real-time 3-D shape measurement system that is able to simultaneously realize acquisition, reconstruction and display at a speed of up to 40 fps for relative shape measurement [5]. We recently developed a real-time system that can acquire the absolute coordinates by coding a marker in the projected fringe images [6]. However, it is very difficult for this system to compute the absolute coordinates and display the geometries in real time with an ordinary computer. We found that the computation power of a dual Central Processing Unit (CPU) (Pentium 4, 3.4GHz) Dell Workstation is not sufficient for the mathematically-intensive absolute coordinate computations. To resolve this problem, we employ the Graphics Processing Unit (GPU) to assist with coordinates computation and rendering. The principle of the GPU will be discussed in Sec. 2.

To reduce the measurement errors due to vibration, Angel and Wizinowich proposed a 2+1 phase-shifting algorithm [7, 8]. In this approach, two fringe images having $90^\circ$ phase-shift are rapidly captured and a third, flat image is collected that is the average of two fringe images with a phase shift of $180^\circ$. This algorithm has found limited use because the small number of data frames in this algorithm makes it susceptible to errors resulting from phase-shifter nonlinearity and calibration [9]. In this research, we are trying to measuring dynamically changing objects. The error caused by the motion of the object is similar to those caused by vibration. Moreover, for our system using a digital video projector to create the fringe patterns, the errors resulting from phase shifting are not there due to its digital fringe generation nature. In this research, we used an algorithm similar to the Angel-Wizinowich algorithm, the difference between them is that the third image can be directly captured by projecting a uniform flat image generated by the computer. Hence, we only need to capture three fringe images rather than four. Therefore, by implementing this algorithm into our real-time 3-D shape measurement system, we maintain the measurement speed while reducing the errors caused by motion using a three-step phase-shifting algorithm. The drawback of the 2+1 phase-shifting algorithm, however, we

#72476 - $15.00 USD Received 30 June 2006; revised 29 August 2006; accepted 30 August 2006

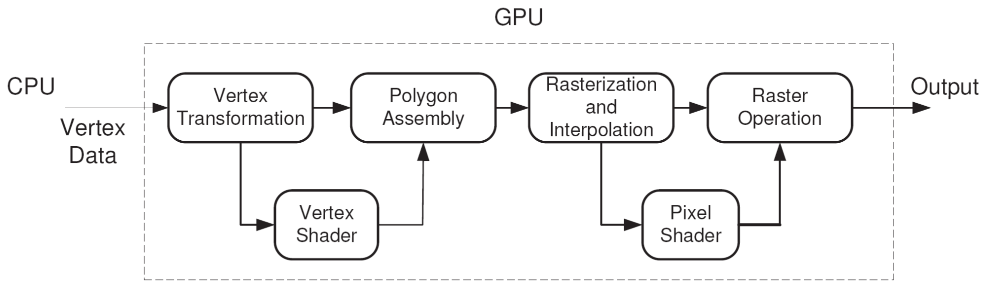


Fig. 1. GPU pipeline. Vertex data including vertex coordinates and vertex normal are sent to the GPU. GPU generates the lighting of each vertex, creates the polygons and rasterizes the pixels, then output the rasterized image to the display screen.

encountered was that the measurement result was noisier than that using the three-step phase-shifting algorithm. By employing a GPU in our real-time system, our experiments show that the acquisition, reconstruction, and display of the near full-resolution absolute coordinates can be simultaneously realized at a frame rate of 30 fps, all in one ordinary computer.

Section 2 introduces the fundamentals of the GPU. Section 3 explains the principle. Section 4 shows experimental results. Section 5 concludes the paper.

## 2. Fundamentals of the GPU

A Graphics Processing Unit or GPU is a dedicated graphics rendering device for a personal computer or game console. Modern GPUs are very efficient at manipulating and displaying computer graphics, and their highly parallel structure makes them more effective than typical CPUs for a range of complex algorithms (http://en.wikipedia.org/wiki/Graphics_processing_unit). Modern CPUs have been increasing their performance over the last decades. However, they encountered severe boundaries for progressing since such increments were mostly achieved by increasing clock frequency and improving the manufacturing process. In contrast, GPUs boost their performance rapidly relying on memory latency rather than on raw speed. By setting a streaming execution model, which reverses the bottleneck inherent to memory access, they achieved the improvements. Since there are no memory hierarchy nor data dependencies in the streaming model, the pipeline maximizes throughput without being stalled. Therefore, whenever the GPU is consistently fed by input data, performance boosts, leading to an extraordinarily scalable architecture [10]. By taking advantage of this streaming processing model, modern GPUs are outperforming their CPU counterparts in some general-purpose applications, and the difference is expected to increase in the future [11].

Figure 1 shows the GPU pipeline. CPU sends the vertex data including the vertex position coordinates and vertex normal to GPU which generates the lighting of each vertex, creates the polygons and rasterizes the pixels, then output the rasterized image to the display screen. Modern GPUs allow user specified code to execute within both the vertex and pixel sections of the pipeline which are called vertex shader and pixel shader, respectively. Vertex shaders are applied for each vertex and are run on a programmable vertex processor. Vertex shaders define a method to compute vector space transformations and other linearizable computations, such as the calculations of the clip-space coordinate and the color of each vertex, etc. Figure 2 shows the diagram of the vertex shader. The vertex position coordinates and normal are usually sent to the GPU. The input position is in the homogeneous coordinates form of $(x,y,z,w)$.

Received 30 June 2006; revised 29 August 2006; accepted 30 August 2006

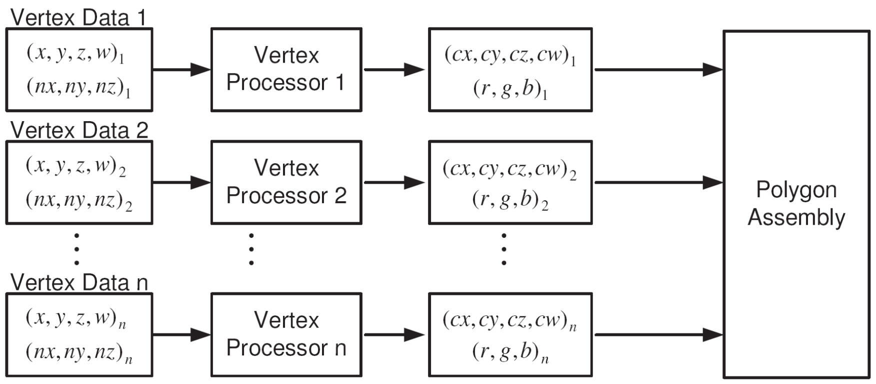


Fig. 2. Vertex shader. The input vertex data including vertex homogeneous position coordinates and normals are sent to the vertex shader, the vertex shader generates the clip space coordinates and the color of every single isolated vertex data. The GPU assembles the polygons based on the order of the streaming data which are then put in the next pipeline.

The vertex shader generates the clip space coordinates $(cx,cy,cz,cw)$ by multiplying the input position coordinates with the model-view matrix. The color for each vertex is also needed to be assigned for each vertex in order to visualize it. The vertex shader takes the input vertex normal $(nx,ny,nz)$ from CPU and computes the vertex color from the lighting condition in a form, $(r,g,b)$, for example. The vertex data is streamed into the GPU where the polygon vertices are processed and assembled based on the order of the incoming data. The GPU handles the transfer of streaming data to parallel computation automatically. Although the clock rate of a GPU is significantly slower than that of a CPU (e.g. our system's 425MHz Quadro FX 3450 GPU compared to our system's 3.4GHz Pentium 4 CPU), it has multiple vertex processors (8) acting in parallel, therefore, the throughput of the GPU can exceed that of the CPU. As GPUs increase in complexity the number of vertex processors increase leading to great improvements in performance. Pixel shaders are mostly used to compute color properties of each pixel. Pixel shaders are applied for each pixel and are run on a pixel processor. They usually have much more processing power than its vertex-oriented counterpart.

The advancement of the GPU has lead to a programmable graphics pipeline. The pipeline can be split into two phases: polygon assembly and polygon rasterization [12]. During polygon assembly the vertex position coordinates, normals, and lighting properties are specified as well as their order within the drawn polygon. During rasterization, the polygon is drawn pixel by pixel on the scene. Currently, the GPU is mainly utilized for rasterization of 3-D primitives, which are often strenuous on the CPU. However, as more complex computations are evaluated on the graphics hardware, a significant increase in computer performance can be achieved by developing GPU-based algorithms. With the introduction of the Cg (C for Graphics) language, programming the pipeline is simplified [13]. Moreover, the GPU isolates vertex data and pixel data so that they can be processed in parallel. This parallel computation structure increases the performance dramatically. In this research, we take advantage of the fast parallel computation and rendering power of GPUs to assist our high-resolution, real-time 3-D shape measurement.

## 3. Principle

Phase-shifting-based methods for 3-D shape measurement are widely used for optical metrology. For these methods, a number of fringe images are recorded (normally larger than or equal to three) and the phase is extracted from the fringes. In general, the more fringe images used, the better the accuracy achieved. Various phase-shifting methods have been developed, including three-step, four-step, and five-step algorithms [14]. For real-time, accurate, 3-D shape measurement, a three-step phase-shifting algorithm is normally employed [15].

### *3.1. Three-step phase shifting algorithm*

The intensity of the fringe images for a three-step phase shifting algorithm with a phase shift of $2\pi/3$ can be written as,

$$\begin{aligned} I_1(x,y) &= I'(x,y)+I''(x,y)\cos(\phi(x,y)-2\pi/3), && (1)\\ I_2(x,y) &= I'(x,y)+I''(x,y)\cos(\phi(x,y)), && (2)\\ I_3(x,y) &= I'(x,y)+I''(x,y)\cos(\phi(x,y)+2\pi/3), && (3) \end{aligned}$$

where $I'(x,y)$ represents the average intensity, $I''(x,y)$ the intensity modulation, and $\phi(x,y)$ the phase to be resolved. Solving these equations simultaneously, we can obtain the phase

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

and the data modulation

$$\gamma(x,y)=\frac{I''(x,y)}{I'(x,y)}=\frac{\sqrt{3(I_1-I_3)^2+(2I_2-I_1-I_3)^2}}{I_1+I_2+I_3}. \tag{5}$$

### *3.2. 2+1 phase-shifting algorithm*

Although the three-step phase-shifting algorithm has the advantages of its symmetry, the measurement error is sensitive to any fringe image errors caused by various sources, such as motion blur. To alleviate this problem, in this research, we replace the third image by a uniform flat image, which is the 2+1 phase shifting algorithm. The intensity of the fringe images therefore becomes,

$$\begin{aligned} I_1 &= I'(x,y)+I''(x,y)\sin(\phi(x,y)), && (6)\\ I_2 &= I'(x,y)+I''(x,y)\cos(\phi(x,y)), && (7)\\ I_3 &= I'(x,y). && (8) \end{aligned}$$

Solving these equations simultaneously, the phase and the data modulation are,

$$\phi(x,y)=\tan^{-1}\frac{I_1-I_3}{I_2-I_3}, \tag{9}$$

and

$$\gamma(x,y)=\frac{I''(x,y)}{I'(x,y)}=\frac{\sqrt{(I_1-I_3)^2+(I_2-I_3)^2}}{I_3}, \tag{10}$$

respectively. Where phase $\phi(x,y)$ in Eq. (4) or Eq. (9) is the so-called modulo $2\pi$ at each pixel, whose value ranges from 0 to $2\pi$. If the fringe patterns have multiple fringes, phase unwrapping is necessary to remove the sawtooth-like discontinuities and obtain a continuous phase map [16]. In this research, we employed a multi-level quality-guided phase unwrapping algorithm that is robust and fast. The detailed algorithm is in a paper under revision at current stage [17]. Once the continuous phase map is obtained, the phase at each pixel can be converted to *xyz* coordinates of the corresponding point on the object surface through calibration [18].

Data modulation $\gamma(x,y)$ in Eq. (5) or Eq. (10) has a value between 0 and 1 and can be used to determine the quality of the phase data at each pixel with 1 being the best.

Since our research focuses on real-time 3-D shape measurement automatically, determining the phase automatically from three fringe images is very important. Due to the symmetry of the three step phase shifting algorithm [14], or the the fast three step phase shifting algorithm [4], the three fringe images do not need to be identified to resolve the phase. However, for the 2+1 phase shifting algorithm, the three fringe images have to be uniquely identified in order to correctly wrap the phase. In this research, the fringe images are identified by analyzing the three fringe images automatically. In order to detect the three fringe images, the flat fringe image has to be identified first. We define a functional

$$\Delta_i = \sum_x \left|\frac{\partial I_i(x,y)}{\partial x}\right| + \sum_y \left|\frac{\partial I_i(x,y)}{\partial y}\right| \tag{11}$$

to find the flat image. The fringe image with the minimum value is the flat image since this image has less intensity variations than the other two fringe images have. Because three images are projected and captured sequentially and repeatedly, the other two images are uniquely determined once the flat image is known.

### 3.3. *Absolute phase to absolute coordinate conversion*

In this research, we used the system calibration method discussed in Ref. ([18]). Once the system is calibrated, we obtain the camera parameter matrix $A_c$ and projector matrix $A_p$,

$$A_c = \begin{bmatrix} a^c_{11}, & a^c_{12}, & a^c_{13}, & a^c_{14} \\ a^c_{21}, & a^c_{22}, & a^c_{23}, & a^c_{24} \\ a^c_{31}, & a^c_{32}, & a^c_{33}, & a^c_{34} \end{bmatrix}, \tag{12}$$

$$A_p = \begin{bmatrix} a^p_{11}, & a^p_{12}, & a^p_{13}, & a^p_{14} \\ a^p_{21}, & a^p_{22}, & a^p_{23}, & a^p_{24} \\ a^p_{31}, & a^p_{32}, & a^p_{33}, & a^p_{34} \end{bmatrix}. \tag{13}$$

These matrices include the translation and rotation matrices from the camera or projector coordinates to the world coordinates. The relationships between the world coordinates $(x,y,z)$ (or the absolute coordinates) and the camera pixel coordinates $(u^c, v^c)$ and the projector pixel coordinates $(u^p, v^p)$ are

$$s^c \begin{bmatrix} u^c, & v^c, & 1 \end{bmatrix}^T = A^c \begin{bmatrix} x, & y, & z, & 1 \end{bmatrix}^T, \tag{14}$$

$$s^p \begin{bmatrix} u^p, & v^p, & 1 \end{bmatrix}^T = A^p \begin{bmatrix} x, & y, & z, & 1 \end{bmatrix}^T, \tag{15}$$

where $s^c, s^p$ are camera and projector scaling factor, respectively.

Once the absolute phase $\phi_a$ is obtained, the relationship between the camera coordinates and the projector coordinates can be established,

$$\phi_a(u^c, v^c) = \phi_a^p(u^p) = \phi_a. \tag{16}$$

Since the projector fringe image is composed of uniform stripes, assume the fringe image has a fringe pitch $P$, the number of pixels per fringe period, and the total number of pixels in the $x$ direction being $W^p$,

$$u^p = \phi_a^c \times P/(2\pi) + W^p/2. \tag{17}$$

Term $W^p/2$ in this equation is used to convert the center of the projected fringe image to be absolute phase 0.

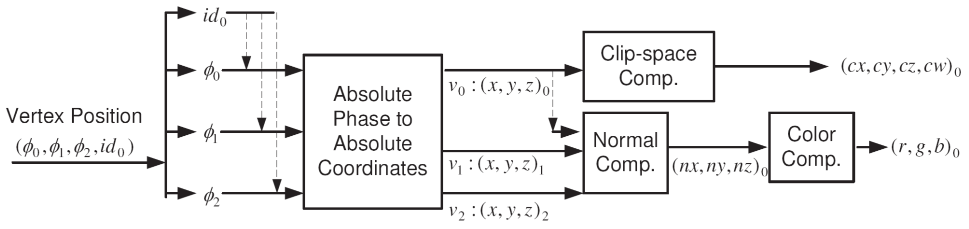


Fig. 3. Our vertex shader. Three absolute phases of one triangle and one vertex index are sent to the GPU. The GPU computes the absolute coordinates for each vertex. The normal for one vertex can then be computed on the GPU.

From Eqs. (12)-(17), we can obtain

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a^c_{11} - u^c a^c_{31} & a^c_{12} - u^c a^c_{32} & a^c_{13} - u^c a^c_{33} \\ a^c_{21} - v^c a^c_{31} & a^c_{22} - v^c a^c_{32} & a^c_{23} - v^c a^c_{33} \\ a^p_{11} - u^p a^p_{31} & a^p_{12} - u^p a^p_{32} & a^p_{13} - u^p a^p_{33} \end{bmatrix}^{-1} \begin{bmatrix} u^c a^c_{34} - a^c_{14} \\ v^c a^c_{34} - a^c_{24} \\ u^p a^p_{34} - a^p_{14} \end{bmatrix}. \qquad (18)$$

It can be seen that the coordinate calculations involve mathematically intensive matrix computations. It will put burden on the CPU if all the computations are done by CPU, which makes the real-time reconstruction and display difficult for an ordinary computer. On the contrary, a GPU can perform these computations efficiently, which will be discussed in the next section.

### *3.4. GPU aided computation and rendering*

In this research, we use the vertex shader of the GPU to compute the coordinates, the normal and the color from the absolute phase. The absolute phase of each vertex and its index are encoded as the position coordinates of the vertex data and are sent to GPU. Since the input position coordinate of the vertex shader has four components, the absolute phase of three vertices of a triangle and one vertex index are put into the four elements of the vertex position coordinates and are sent at a time. For example, we assume that three vertices of one triangle $v_0$, $v_1$ and $v_2$ of one triangle have absolute phase $\phi_0$, $\phi_1$ and $\phi_2$, respectively, and $v_0$ has index $id_0$ in the phase image. The position coordinate input of the vertex data is $(\phi_0, \phi_1, \phi_2, id_0)$. Since the relationship between the three vertices is known, the indices of the other two vertices are uniquely determined from the index of $v_0$. Therefore, the absolute coordinates for each vertex can be computed using Eq. (18). Once the absolute coordinates are known, the vertex normal $(nx, ny, nz)_0$ for vertex $v_0$ can be computed on the GPU from the three vertices of the triangle. Once the vertex normal is known, the color $(r, g, b)_0$ for this vertex can be calculated from the light condition setup. Vertex data for vertex $v_0$ will be used for the following pipeline, while the other two vertex data are discarded from this vertex shader. Figure 3 shows the diagram of our vertex shader. Once the coordinates and the normal for each vertex are known, the GPU then generates the clip-space coordinates and lighting. Finally, the streaming points are assembled into polygon sets and rendered by the GPU.

Table 1 shows the comparison between the computation and rendering results using the CPU and the GPU. In this experiments, all the programs are written in C++ under Microsoft Visual C++.net 2005 environment, and exactly the same algorithm is implemented both in CPU and GPU. From this table we can see that the GPU boosts the computation and rendering speed dramatically. For the full resolution image ($532 \times 500$), our NVidia Quadro FX 3450 graphics card can do 25.56 fps while the 3.4GHz CPU can only do 6 fps with the same graphics card. The speed of the GPU is more than 4 times faster than that of the CPU. However, in real

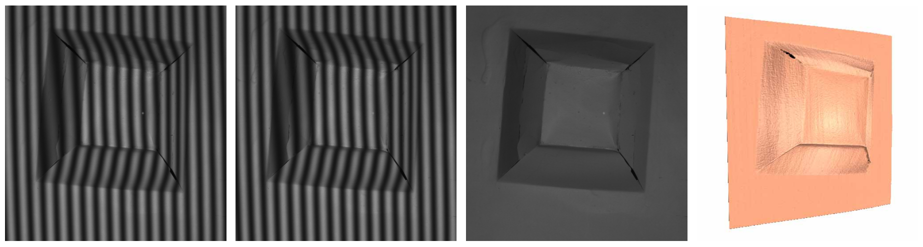

(a) $I_1$. (b) $I_2$. (c) $I_3$. (d) 3-D geometry.

Fig. 4. Measurement results of a static object using the 2+1 phase-shifting algorithm.

measurement of a human face, only about half of the pixels need to be rendered since the background is not displayed. In this case, the GPU can do more than 30 fps. It should be noted that even with a fairly cheap ATI 9800 XT graphics card, computing and rendering one quarter of the points can be as fast as 53.37 fps, which can keep up with our data acquisition speed. It should be noted that all the coordinates and color data are remain in GPU. Of course, the 3-D coordinates data can be transferred back to CPU but in a very slow manner due the current architecture design of the graphics card.

Table 1. Comparison between the computation and rendering with GPU and CPU

| | CPU (fps) | | GPU (fps) | | |
|---|---|---|---|---|---|
| Image Size | 3.2GHz | 3.4GHz | ATI 9800 XT | Quadro FX 1400 | Quadro FX 3450 |
| $532 \times 500$ | 5.50 | 6.00 | 13.56 | 20.35 | 25.56 |
| $266 \times 250$ | 21.28 | 23.15 | 53.37 | 78.05 | 102.81 |
| $177 \times 167$ | 46.50 | 51.06 | 111.29 | 157.71 | 222.56 |
| $133 \times 125$ | 79.50 | 89.93 | 179.45 | 246.15 | 372.82 |

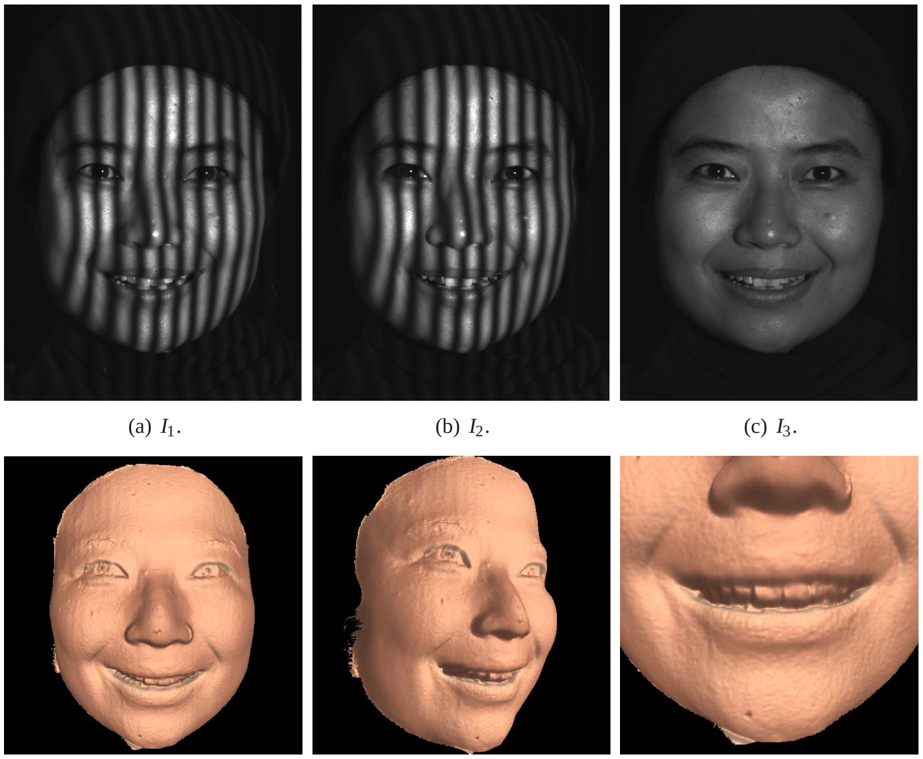
Note: Quadro FX 1400 and Quadro FX 3450 are NVidia graphics cards. The computers are Dell Precision 670 Workstations with Dual CPUs, Pentium 4, 3.2GHz and 3.4GHz respectively.

## 4. Experiments

To verify the performance of the 2+1 phase-shifting algorithm, we implemented this algorithm into our pre-developed real-time 3-D shape measurement system by replacing the three-step phase-shifting algorithm [6]. Figure 4 shows one measurement of a stationary object. Figure 4(a)-4(c) shows three fringe images captured by the system, and Fig. 4(d) shows the reconstructed 3-D result. It clearly shows that this algorithm can measure a stationery object satisfactorily. It should be noted that we use a $5 \times 5$ Gaussian filter to smooth the geometry in order to reduce the random measurement noise and all 3-D data presented in this paper are smoothed by the same Gaussian filter.

We also measured a human face with expressions, Fig. 5 shows one typical frame. During the experiment, the subject was asked to do various facial expressions continuously. It can be seen that the face can be clearly captured with details.

We then implemented our GPU aided computation and rendering program into our system. In this experiment, we used an NVidia Quadro FX 3450 graphics card. The graphics card has 256MB of memory clocked at 500 MHz. The GPU is clocked at 425 MHz. It has 8 vertex shader and 24 pixel shader units. The computer we used was a Dell Precision Workstation 670 with Dual Pentium 4, 3.4GHz CPUs and 3.0GB memory. The video in Fig. 6 shows the experiment. It can be seen from this video that the data acquisition, reconstruction and display


(a) $I_1$. (b) $I_2$. (c) $I_3$.

(d) 3-D geometry. (e) Another view angle. (f) Zoom-in view.

Fig. 5. Measurement result of human face.

are simultaneously realized at 30 fps. The rendered 3-D geometry can keep up with the subject movements well. This video demonstrated that the absolute coordinates of the object can be computed and reconstructed in real-time.

## 5. Conclusions and Future Works

This paper presented GPU-assisted real-time 3-D shape measurement using a 2+1 phase shifting algorithm. By taking advantage of the processing and rendering power of a NVidia FX 3450 Graphics Card, the absolute coordinates of the objects, sampled at $532 \times 500$ points per frame, can be computed and rendered at 25.56 fps. Otherwise, if the coordinates computation is done in a Dell Precision 670 Workstation with a Dual Pentium 4, 3.4GHz CPU and then displayed with the same graphics card, the speed would reduce to approximately 6 fps excluding the phase wrapping and unwrapping cost. By putting the CPU's coordinate computation burden on the GPU, we realized a simultaneous acquisition, reconstruction, and display at 30 fps with almost full resolution for absolute coordinates, all in an ordinary computer. We replaced the three-step phase-shifting algorithm of our previous system with the 2+1 phase-shifting algorithm. The measurement errors caused by motion were alleviated significantly. With this new algorithm, the ordinary expression of a human face can be captured with better quality than our previous system. Moreover, the 2-D texture image quality is much higher since the texture image can be captured when the projector is projecting the flat images. For a three-step phase-shifting algorithm, the flat 2-D image can be computed, however, if one of the three fringe images are not ideally sinusoidal with the exact phase shift used, the flat image quality will drop significantly.

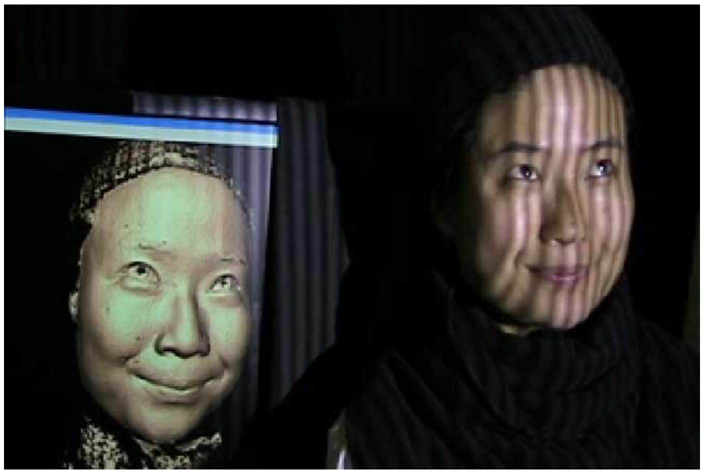


Fig. 6. (2.32MB) Real-time acquired, reconstructed, and displayed 3-D absolute coordinates at 30 fps.

Even though this system was able to acquire the 3-D geometric shape in real-time at 30 fps and the 2+1 phase-shifting algorithm can alleviate the errors caused by motion, we still encounter significant measurement errors of faster motion such as facial geometry changes when speaking. Our future work is to improve the data acquisition speed so that the errors caused by motion will be reduced.

**Acknowledgement**

We thanks Xiaomei Hao for serving as model to test our system, thanks the members of Geometric Informatics Inc for their cooperation for this research. This work was done in Geometric Informatics Inc, funded by the Advanced Technology Program (ATP) of the National Institute of Standards and Technology (NIST).