High-resolution real-time 3D shape measurement on a portable device

Nikolaus Karpinsky
Iowa State University

Morgan Hoke
Iowa State University

Vincent Chen
Iowa State University

Song Zhang
Iowa State University, song@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/me_conf

Part of the Computer-Aided Engineering and Design Commons, and the Graphics and Human Computer Interfaces Commons

Recommended Citation
Karpinsky, Nikolaus; Hoke, Morgan; Chen, Vincent; and Zhang, Song, "High-resolution real-time 3D shape measurement on a portable device" (2013). Mechanical Engineering Conference Presentations, Papers, and Proceedings. 67.
https://lib.dr.iastate.edu/me_conf/67

This Conference Proceeding is brought to you for free and open access by the Mechanical Engineering at Iowa State University Digital Repository. It has been accepted for inclusion in Mechanical Engineering Conference Presentations, Papers, and Proceedings by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
High-resolution real-time 3D shape measurement on a portable device

Abstract
Recent advances in technology have enabled the acquisition of high-resolution 3D models in real-time though the use of structured light scanning techniques. While these advances are impressive, they require large amounts of computing power, thus being limited to using large desktop computers with high end CPUs and sometimes GPUs. This is undesirable in making high-resolution real-time 3D scanners ubiquitous in our mobile lives. To address this issue, this work describes and demonstrates a real-time 3D scanning system that is realized on a mobile device, namely a laptop computer, which can achieve speeds of 20fps 3D at a resolution of 640x480 per frame. By utilizing a graphics processing unit (GPU) as a multipurpose parallel processor, along with a parallel phase shifting technique, we are able to realize the entire 3D processing pipeline in parallel. 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. To demonstrate the effectiveness of the technique, we experiment with the scanner on both static geometry of a statue and dynamic geometry of a deforming material sample in front of the system.

Keywords
Fringe projection, structured light, real-time 3D, high-resolution 3D

Disciplines
Computer-Aided Engineering and Design | Graphics and Human Computer Interfaces

Comments
This is a conference proceeding from Dimensional Optical Metrology and Inspection for Practical Applications II 8839 (2013):1, doi:10.1117/12.2021655. Posted with permission.

Copyright 2013 Society of Photo-Optical Instrumentation Engineers. One print or electronic copy may be made for personal use only. Systematic electronic or print reproduction and distribution, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper are prohibited.
High-resolution, real-time 3D shape measurement on a portable device

Nikolaus Karpinsky, Morgan Hoke, Vincent Chen, and Song Zhang
Iowa State University, Ames IA, USA

ABSTRACT
Recent advances in technology have enabled the acquisition of high-resolution 3D models in real-time through the use of structured light scanning techniques. While these advances are impressive, they require large amounts of computing power, thus being limited to using large desktop computers with high end CPUs and sometimes GPUs. This is undesirable in making high-resolution real-time 3D scanners ubiquitous in our mobile lives. To address this issue, this work describes and demonstrates a real-time 3D scanning system that is realized on a mobile device, namely a laptop computer, which can achieve speeds of 20fps 3D at a resolution of 640x480 per frame. By utilizing a graphics processing unit (GPU) as a multipurpose parallel processor, along with a parallel phase shifting technique, we are able to realize the entire 3D processing pipeline in parallel. 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. To demonstrate the effectiveness of the technique, we experiment with the scanner on both static geometry of a statue and dynamic geometry of a deforming material sample in front of the system.

Keywords: Fringe projection, structured light, real-time 3D, high-resolution 3D

1. INTRODUCTION
Advances in technology have enabled the acquisition of 3D models in real-time with techniques such as stereo vision and structured light scanning. New advancements aim to push the accuracy, resolution, and the speed of the acquisition. Recently, advancements in the speed of structured light based techniques have pushed them into real-time; this growth into real-time has given structured light scanners broad exposure, as can be seen with the introduction of the Microsoft Kinect. 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. 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, 3D information can be triangulated using the established correspondence.

Decoding patterns and 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 entire process can be realized utilizing parallel algorithms, parallel compute devices such as graphics processing units (GPUs) can be used to offload the computationally intensive problem. Although the processing speed on a GPU is not as fast as a CPU, anywhere from 1-8 times slower, it can process hundreds of threads simultaneously, assuming there is no branching. If the GPU can be leveraged on a portable device, then processing is not limited by the devices lower computational power.

To establish the correlation between projector and camera pixels, the encoded patterns must be properly chosen to allow for parallel computation, maximum speed, yet being resilient to noise from things such as ambient light, surface reflections, etc. Many different techniques such as stripe boundary code, binary coded patterns, and phase shifting methods 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...

Further author information:
Nikolaus Karpinsky: E-mail: karpinsn@iastate.edu
Song Zhang: E-mail: song@iastate.edu
Recently, a multi-wavelength fringe projection technique for high speed measurement using projector defocusing has been proposed by Wang and Zhang,\textsuperscript{9} which uses multiple wavelength fringe to assist in phase unwrapping. If the wavelengths of the fringe are properly chosen, phase unwrapping becomes a parallel operation, circumventing problems with traditional phase unwrapping algorithms such as the inability to measure discontinuous surfaces. Although the technique assists in phase unwrapping, it involves a costly arctangent calculation for each wavelength at each pixel. This makes it difficult to achieve real-time speeds with the technique on a CPU, but since it is a parallel algorithm it is well suited for GPU implementation.

To achieve high-resolution real-time 3D shape measurement on a portable device, our approach utilizes a GPU as a multipurpose parallel co-processor. 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 3D reconstruction, allowing us to realize the system with a portable device, namely a laptop computer. In addition to low processing power requirements, since the entire system is realized on the GPU, the CPU is free to perform other operations in parallel to the 3D reconstruction, such as 3D registration or feature extraction.

Section 2 of this paper addresses 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. Section 4 shows the experiments performed to demonstrate the system and discusses the findings. Finally, Section 5 concludes the paper and talks about venues for future work.

2. PRINCIPLE

2.1 Multi-Wavelength Phase Shifting Algorithm

The structural patterns used by our system are a set of three-step phase shifting patterns. Phase shifting 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. Three-step phase shifting patterns can be described by the following.

\begin{align*}
I_1(x, y) &= I'(x, y) + I''(x, y) \cos[\phi(x, y) - 2\pi/3], \\
I_2(x, y) &= I'(x, y) + I''(x, y) \cos[\phi(x, y)], \\
I_3(x, y) &= I'(x, y) + I''(x, y) \cos[\phi(x, y) + 2\pi/3].
\end{align*}

$I'$ is the average intensity, $I''$ is the intensity modulation, and $\phi$ is the encoded phase. Using these equations, the phase $\phi$ can be solved for via:

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

This will yield a phase value $\phi$ for every pixel, but since the $\tan^{-1}$ only ranges from 0 to $2\pi$, the phase value provided will have $2\pi$ phase discontinuities; this phase value is known as wrapped phase. Conventional approaches employ a phase unwrapping algorithm that traverses along the phase map adding multiples of $2\pi$,\textsuperscript{10} but this is a serial operation that requires neighboring pixel information thus being undesirable for GPU implementation. Instead a second set of three-step phase shifting patterns are used with a slightly different phase, which will result in another wrapped phase $\phi_2$. By utilizing the phase difference of the phases $\phi_1$ and $\phi_2$ a larger equivalent phase can be calculated.
\[ \lambda_{12} = \frac{\lambda_1 \times \lambda_2}{|\lambda_1 - \lambda_2|} \]  
\[ \phi_{12} = \phi_1 - \phi_2 \mod 2\pi. \] (5) (6)

By properly selecting the phases \( \phi_1 \) and \( \phi_2 \), a large equivalent phase that spans the entire phase map can be achieved and then utilized in unwrapping the smaller phase \( \phi_1 \). This process is modeled by Equations 7-8, where \( \lambda \) is the wavelength of the corresponding phase and \( k \) is the integer number of phase jumps.

\[ k(x, y) = \text{round} \left( \frac{\phi_{12}(x, y) \times \frac{\lambda_{12}}{\lambda_1} - \phi_1(x, y)}{2\pi} \right), \] (7)
\[ \Phi(x, y) = \phi_1(x, y) + 2\pi \times k(x, y). \] (8)

Using this multi-wavelength approach to phase unwrapping an unwrapped phase map \( \Phi \) can be achieved per pixel in parallel, thus being well suited to GPU implementation. Figure 1 shows a set of captured fringe images with \( \lambda_1 = 60 \) and \( \lambda_2 = 66 \), and their corresponding wrapped and unwrapped phase.

![Multi-wavelength phase unwrapping process](image)

Figure 1. Multi-wavelength phase unwrapping process. (a) three fringe images of smaller phase (b) three fringe images of larger phase (c) wrapped phase \( \phi_1 \) (d) wrapped phase \( \phi_2 \) (e) equivalent phase \( \phi_{12} \) (f) unwrapped phase \( \Phi \).

### 2.2 General purpose GPU

Recently, in order to accelerate parallel tasks on a computer, general purpose GPU (GPGPU) computation has been 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 co-processor. 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 3D reconstruction, different programming interfaces can be used such as NVIDIA CUDA\textsuperscript{11} or the OpenGL Shading Language (GLSL).\textsuperscript{12} GLSL is widely used for this task due to its speed and ability to work on nearly any modern graphics card.
In order to use GLSL for GPGPU computation versus traditional computer graphics applications certain techniques need to be leveraged: offscreen rendering, direct memory access, 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.

In order to get input into the GLSL program, buffers can be bound that the program can access, such as OpenGL textures. When using GPUs, one of the major bottlenecks are transfers to and from the GPU. To alleviate this bottleneck, direct memory access (DMA) can be used, which allows specifically allocated parts of memory to be used in transfers. Transfers through DMA do not require the CPU and GPU to synchronize, and the GPU can transfer data while simultaneously processing its pipeline; this allows the bottleneck of transfers to be mitigated.

Lastly, multipass rendering can be utilized to run different GLSL programs multiple times on different buffers to achieve multiple stages in a pipeline. By clearing or not using depth buffering, the OpenGL driver will not remove any geometry from the processing pipeline. By binding different input and output buffers as well as different GLSL programs in between rendering screen aligned quads, transforms on the data can be utilized in stages. 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 utilized to render the final scene.

3. IMPLEMENTATION

In the 3D decoding pipeline for our system, there are five discrete stages: phase wrapping, 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.

![Real-time 3D 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. Next phase unwrapping is performed, followed by phase filtering, depth calculation, and normal calculation. Finally, final rendering is performed using the depth map and normal map producing the 3D scan.](image-url)
3.1 Phase Calculation

Phase calculation 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, Equation 4 is applied to each image resulting in two wrapped phase maps, \( \phi_1 \) and \( \phi_2 \), one for each wavelength. 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 \((r,g,b,a)\), illustrated by Equations 9-12. The components are extracted so that during phase filtering, errors are not introduced into the wrapped phase map.

\[
\begin{align*}
r &= \sin(\phi_1), & (9) \\
g &= \cos(\phi_1), & (10) \\
b &= \sin(\phi_2), & (11) \\
a &= \cos(\phi_2). & (12)
\end{align*}
\]

3.2 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 \( \times \) 11 Gaussian filter, a phase wrapping pass, and then one pass of a specialized median filter. \(^15\)

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 non separable kernel was used, 121 texture lookups would be required, substantially slowing down filtering.

Next a phase wrapping pass is employed that takes the filtered phase components, and wraps it into a wrapped phase map \( \Phi \) using Equations 6-8. Finally a pass of our specialized 5x5 median filter is applied. Instead of applying a direct median filter, the specialized median filter calculates the median and then sets the number of phase jumps \( k \) of the original value to that of the median, removing spiking noise from phase wrapping.

3.3 Depth Map Calculation

Depth map calculation involves calculating depth values for each unwrapped phase value. There are a number of approaches to do this based on the chosen calibration, and in our method we chose to perform a reference plane based approach detailed by Xu et al.\(^16\) By capturing the phase of a reference plane \( \Phi_R \) where \( z = 0 \), a phase difference between the captured phase \( \Phi_C \) and \( \Phi_R \) can be calculated. This phase difference will be proportional to the depth \( z \) by a scaling value, as shown by Figure 3. To calculate this in the fragment shader, a texture containing \( \Phi_R \) is read in along with a texture containing the filtered phase \( \Phi_C \). Subtracting the two phases and scaling, based on a scaling factor \( c \) determined through calibration, yields the depth value \( z \), as illustrated by Equation 13. This depth value is then rendered out as the color, yielding the depth map.

\[
z = c \times [\Phi_C - \Phi_R] \quad (13)
\]

3.4 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 point and two neighboring points, moving sequentially counterclockwise in a 3x3 neighborhood, and calculating the cross product. This yields a surface normal for the polygon comprised 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.
3.5 Final Rendering

The last stage in the 3D 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 this, 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 subsampled, 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 render 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 a static sculpture, and measuring a dynamically moving object. In each of the experiments the hardware stayed consistent, a Point Grey Research Flea3 camera with a Computar 12mm lens, a Texas Instruments Light Crafter DLP projector, and a IBM Lenovo laptop with a Intel i5 3320M 2.6GHz CPU and NVIDIA Quadro NVS5400M.

To test the measurement noise of the system we first captured a flat surface. In an ideal system, the surface should be perfectly flat, but due to sensor and environment noise their will be small variations. Figure 4 (a) shows the results of the capture, and Figure 4 (b) shows a horizontal cross section at the 300th scan line. The variations in the surface height results in an RMSE error of .219\%, for a measurement area of 100\( \text{mm} \times 75\text{mm} \) with a resolution of 640 \( \times \) 480.

To further test the system’s capabilities we performed measurements on a static sculpture. Figure 5 (a) shows the sculpture being scanned, and Figure 5 (b) shows the resulting reconstruction. The system was able to reconstruct the 3D geometry of the statue at 20 FPS 3D on the portable device, with a resolution of 640 \( \times \) 480. Finally, to test the system even further, we captured a dynamically moving object. Figure 6 shows an example of 4 different frames of the capture. Again the system was able to capture and reconstruct the 3D geometry of the object at 20 FPS 3D at a resolution of 640 \( \times \) 480 on the portable device.

Overall proposed real-time 3D scanner has several advantages over previous methods:

1. High resolution: Since the fringe are generated using binary defocussing, the spatial resolution is limited only by the camera using for capture.
Figure 4. Measurement result of a flat surface. The measured area is 100mm × 75mm, and the resulting RMSE is .219%.
(a) 3D plot of the surface, (B) Example cross section of the surface.

Figure 5. Capture of static statue. (a) Real-time 3D capture system and portable device, (b) resulting reconstruction from capture.

Figure 6. Frames from the 3D capture of a dynamically moving object.
2. High decoding speed: Since the entire pipeline is implemented in parallel on a GPU, decoding is extremely fast and can easily reach real-time speeds.

3. Reduced CPU load: Again since the entire pipeline is implemented in parallel on the GPU, the only load on the CPU is transferring images. This frees up the CPU for other tasks, such as 3D image processing, object detection, etc.

5. CONCLUSION

This paper has presented a technique for achieving high-resolution, real-time 3D imaging on a portable device. We have demonstrated the principles behind the techniques leveraged by the system, as well as giving a description of the implementation. By utilizing a GPU for the entire 3D reconstruction and display process, processing power requirements have been drastically reduced, allowing the system to be realized with a portable device. Through experiments, we have shown that speeds of 20 FPS can be achieved on the portable device at a resolution of 640 × 480. Future work could include ways of utilizing the nearly idle CPU for additional tasks, such as 3D image processing, object detection, etc.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE1247194. 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 National Science Foundation.

REFERENCES

1. Corp., M. “Microsoft kinect.”
