ACCELERATED MULTI-VIEW STEREO USING PARALLEL PROCESSING CAPABABILITIES OF THE GPUS

O. Moslah[*], *A. Valles-Such, V. Guitteny, S. Couvet*

THALES Security Solutions and Services
1 Rue du General de Gaulle,
95523 Cergy-Pontoise, France.

S. Philipp-Foliguet

ETIS - UMR CNRS 8051, ENSEA
6 Avenue du Ponceau,
95014 Cergy-Pontoise, France.

ABSTRACT

This paper presents an accelerated implementation of a multi-view stereo pipeline using parallel processing capabilities of the GPUs. Our system takes as input a set of calibrated photographs and produces a textured 3d mesh of the scene. The pipeline is divided into three parts: dense stereo matching, multi-view correspondence linking and 3d model generation. First, we use a combined vertical aggregation and dynamic programming (DP) scheme to produce disparity maps between pairs of photographs. Then, the depth maps are computed using a multi-view correspondence linking algorithm. Finally, we use a Delaunay triangulation algorithm and texture mapping to produce the 3d model of the scene.

1. INTRODUCTION

We present in this paper an accelerated implementation of a multi-view stereo pipeline that takes advantage of the parallel processing capabilities of actual GPUs. Stereovision is a very challenging research topic in computer vision. The research community has devoted a lot of effort to developp algorithms and techniques for an automatic 3D reconstruction of a scene from a set of uncalibrated photographs [6, 7]. Scharstein and Szeliski [1] have established a classification of the different dense stereo matching algorithms into local and global methods. Local methods use the intensity of a pixel and of its neighbours to compute the disparity. Global methods make use of optimization techniques (dynamic programming, graph cuts, ...) for the computation of the disparities. Acceleration using graphics hardware to estimate depth was first explored by Yang et al. [2] using a plane sweep approach. Cornells and Van Gool [5] combined this with the iterative refinement from Zach et al. [8] to generate quality depth maps for fine 3D structures.

[*]Email: oussama.moslah@thalesgroup.com

1.1. GPU pipeline and GPGPU

Acceleration using graphics hardware has been for a long time restricted to purely graphical processing. With the constant evolution of graphics hardware and the emerging GPGPU techniques and technologies such Cg [12] and CUDA [13] researchers start to re-design their algorithms to benefit from the parallel capabilities of modern GPUs. GPGPU stands for General Purpose Graphics Processing Unit [11]. Stating it briefly, GPGPU is a combination between hardware components and software that allows the use of a traditional GPU to perform computing tasks that are extremely demanding in terms of processing power. The graphics pipeline is designed to allow hardware implementations to maintain high computation rates through parallel execution. The pipeline is divided into several stages; all geometric primitives pass through every stage. In hardware, each stage is implemented as a separate piece of hardware on the GPU in what is termed a task-parallel machine organization. Figure 1 shows the pipeline stages in current GPUs. This pipeline is described in more detail in the OpenGL Programming Guide [9].

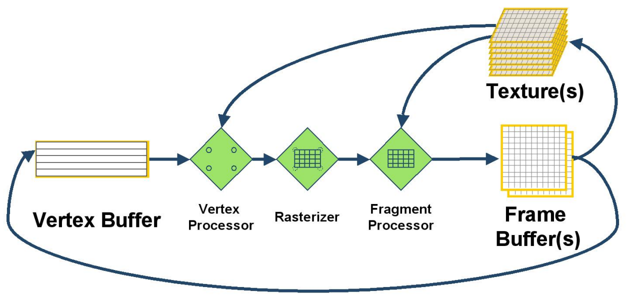

Fig. 1. *The modern graphics hardware pipeline: the vertex and fragment processor stages are both programmable by the user.*

1.2. System overview

The 3d reconstruction pipeline presented in this paper consists of three parts. (1) Dense Stereo Matching: match all the pixels between consecutive image pairs and produces the disparity maps. We use two different dense stereo matching methods: the winner takes all approach, that runs fully in GPU and a dynamic programming approcah that runs in both GPU and CPU. (2) Multi-View Correspondence Linking: produces the depth maps using multiple view triangulation of all correspondences. This process fully runs in GPU and form the major contribution of this paper. (3) 3D Mesh Generation and Texture Mapping: produces the final 3D model using a Delaunay triangulation and texture mapping.

2. DENSE STEREO MATCHING

Dense stereo matching consists in matching all the pixels between two consecutive images in order to produce disparity maps. Our algorithm has three major steps: matching cost computation, cost aggregation and disparity selection. To compute the cost volume, we draw a rectangle aligned with the two input rectified images stored as textures, and one of them shifted by d pixels. We adopt the following matching cost criterion:

$$|p(x, y) - q(x + d, y)| \tag{1}$$

where p and q are the corresponding matched pixels and d is the hypothesized disparity value. We use a fragment program to calculate the color absolute difference and output it to a texture. After doing this process over all the disparity hypothesis d we have the entire cost volume separated in different textures. One 4-channel texture for each 4 disparity hypothesis. The matching cost computation is done entirely in the graphics processing unit. We compute the weight masks in a similar way using the following equation:

$$w(p, q) = \exp\left(-(\frac{\Delta c_{pq}}{\gamma_c} + \frac{\Delta g_{pq}}{\gamma_g})\right) \tag{2}$$

where p and q are pixels, Δc_{pq} their color difference, Δg_{pq} their Euclidean distance and γ_c and γ_g are weighting constants determined empirically. The cost aggregation is implemented as a pixel shader that can index between both cost and weight textures to produce a new set of textures that will conform the aggregated cost volume. The aggregated cost is computed as a weighted sum of the per-pixel cost [3] and is implemented as a pixel shader. The number of rendering passes needed to do this process is $\lceil \frac{N*H}{16} \rceil$ where N is the number of disparity hypothesis and H is the size of the vertical window. To select the best disparity for each pixel in the image we use both Winner Takes All (WTA) approach and a Dynamic Programming (DP) scheme [10]. WTA approach simply selects the disparity that has the lower cost while DP selects the disparity trying to minimize a global energy function [3]. The WTA algorithm can be fully implemented in the GPU. The DP algorithm can be implemented also in GPU, but as reported in [3, 4], a GPU-based DP program is actually slower than it's CPU counterpart. Figure 2 shows the differences between the disparity maps calculated with the two different methods: WTA and DP. Table 1 illustrates the real-time performances of the dense stereo matching algorithm.

Image Size	DR	VA + DP			WTA
		GPU	CPU	Total	GPU
640x480	16	0.024	0.382	0.407	0.026
	32	0.051	0.731	0.783	0.054
	48	0.076	1.082	1.158	0.081
320x240	16	0.007	0.098	0.105	0.007
	32	0.013	0.182	0.196	0.014
	48	0.019	0.271	0.290	0.020

Table 1. *Real-time Performance in seconds. The test system is a 3.0Ghz with a NVIDIA GForce 8800 GTS 512 Mb graphics card. VA+DP denotes Vertical Aggregation + Dynamic Programming, WTA denotes Winner Takes All and DR the Disparity Range.*

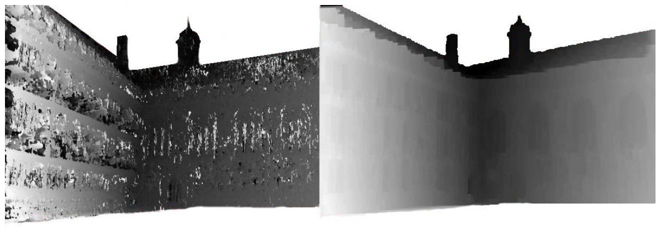

Fig. 2. *Resulting disparity maps using WTA and DP methods.*

3. MULTI-VIEW CORRESPONDENCE LINKING

The pairwise disparity estimation allows to compute correspondences between adjacent rectified image pairs and independent depth estimates for each camera viewpoint. In order to fuse those separate depth estimates into a common 3D model we use the multi-view correspondence linking algorithm described in [7]. For each image point m_k we create two chains of correspondence links one up m_{k+1} and one down m_{k-1} as follows :

$$m_{k+1} = (H^k_{k+1})^{-1} D_{(k,k+1)} [H^{k+1}_k m_k] \tag{3}$$

$$m_{k-1} = (H^k_{k-1})^{-1} D_{(k,k-1)} [H^{k-1}_k m_k] \tag{4}$$

This linking process is repeated along the image sequence for a reference view i to create a chain of correspondences upwards $(i, i+1, .., n)$ and downwards (i,i-1,...,1) . Every correspondence link requires 2 mappings and 1 disparity lookup. The Disparity map $D_{(k,k-1)}$ holds the downward correspondences from image I_k to I_{k-1} while the map $D_{(k,k+1)}$ contains the upward correspondences from I_k to I_{k+1} . Rectification of image points for a stereoscopic pair of images (I_k, I_{k+1}) is done using transformation matrices H_k^{k+1} and H_{k+1}^{k}. Once we have located the same pixel in a sequence of images, we can proceed to triangulate them to get a set of depths. And last, we use a Kalman filter to estimate the final depth. We also have to detect when a depth estimation falls out of our previous mean estimation. Outliers can be detected comparing the depth estimation of the new point with the previous filtered mean and taking into account the error in the current step. If we find an outlier, the correspendance chain ends and we have to assure that this pixel will not be triangulated in further steps. Figure 3 illustrates the principle of depth estimation and outlier detection using a chain of correspondence link.

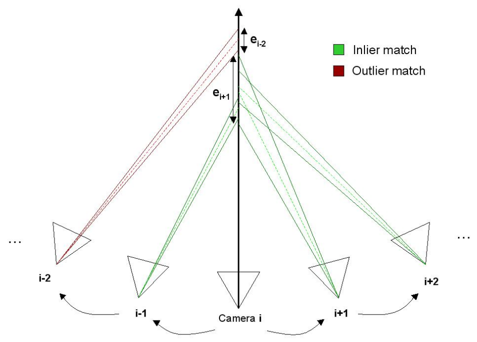

Fig. 3. *Depth fusion, uncertainty estimation and outlier detection from correspondence linking.*

We have implemented this algorithm in the GPU by performing the linking in an incremental way. We select a camera for which we want to compute the depth map then we step forward and backward to triangulate all the correspondences. At each step we triangulate correspondent image points and estimate the pixels depth values. For each step and for each image point we save the following informations in textures: the coordinates of correspondant point in the next image, the filtered depth estimation and the number of estimations. By this way the algorithm is suitable to run in GPU and generates the depth maps in a very fast way. The CPU implementation takes about 112 s to calculate the depth maps of the Wadham College example, our GPU implementation takes less than 0.5 s. We can see an example of a computed depth map using this algorithm in figure 4.

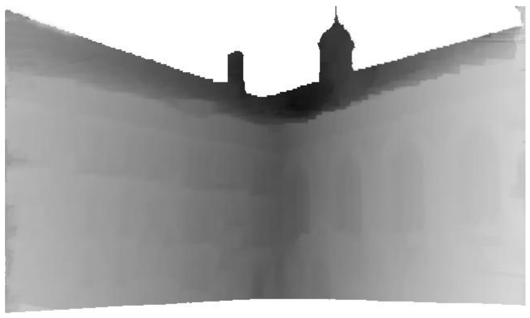

Fig. 4. *Resulting depth map using the multi-view correspondence linking algorithm.*

4. 3D MESH GENERATION AND TEXTURE MAPPING

The 3d mesh generation is performed using a 2D Delaunay triangulation of a selected depth map followed by a inverse projective transformation. We compute normal vectors of each triangle and extract textures from the original images depending on the camera viewpoint. The visualisation and rendering of the final 3d model is then obtained using the OpenSceneGraph library [14]. Figure 5 shows some final 3d models generated with our system and Table 2 illustrates the computation times.

5. CONCLUSION

This paper presented algorithms and techniques to accelerate a multi-view stereo pipeline using parallel processing capababilities of the GPUs. Most previous work focus on the accelerlation of two-frame dense stereo matching algorithms. Our GPU implementation of a multi-view correspondence linking algorithm allows 3d reconstruction from multiple images. This form the major contribution of this paper.

6. ACKNOWLEDGEMENTS

We wish to acknowledge the Cap Digital Business Cluster Terra Numerica project for sponsoring the research reported in this paper.

7. REFERENCES

[1] D. Scharstein and R. Szeliski, A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision (IJCV)*, 47(1):7-42, May 2002.

[2] R. Yang, G. Welch, and G. Bishop, Real-Time Consensus-Based Scene Reconstruction Using Commodity Graphics Hardware. In *Proceedings of Pacific*

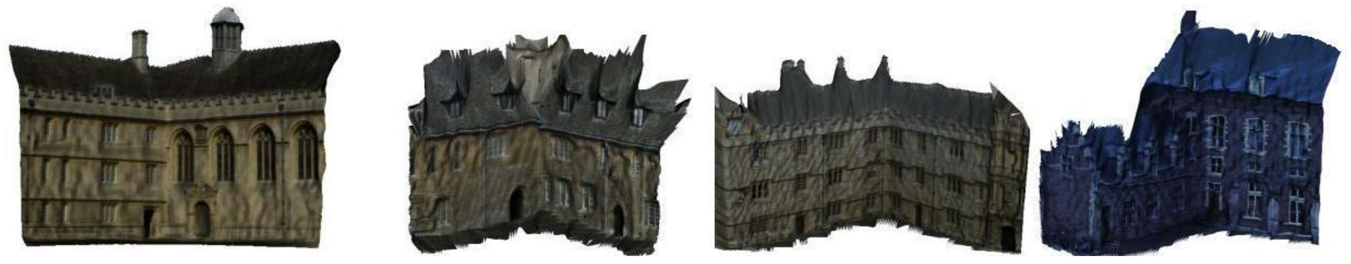

Fig. 5. *Four views of the reconstructed models. From left to right: Wadham College, Merton College I, Merton College II [15] and Arenberg Castle [16].*

Image Sequence	Maximum Disparity Range	DSM	MVCL	3DMG&TM	Total
Wadham College (5 images 640x480)	[-34,94]	40.987	0.414	1.122	42.623
Arenberg Castle (22 images 768x576)	[-8,52]	89.301	3.053	1.774	94.128
Merton College I (3 images 640x480)	[-248,90]	418.214	2.942	0.969	422.125
Merton College II (2 images 1024x768)	[-65,106]	244.812	2.370	2.430	249.612

Table 2. *Computation times in seconds. The test system is a 3.0Ghz with a NVIDIA GForce 8800 GTS 512 Mb graphics card. DSM, MVCL and 3DMG&TM respectively denotes for Dense Stereo Matching, Multi-View Correpondence Linking and 3D Mesh Generation and Texture Mapping.*

Graphics 2002, pages 225-234, Beijing, China, October 2002.

[3] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, High Quality Real-time Stereo using Adaptive Cost Aggregation and Dynamic Programming. *Third International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT), 2006*

[4] M. Gong and Y.-H. Yang, Near real-time reliable stereo matching using programmable graphics hardware. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 924-931, 2005.

[5] N. Cornells and L. V. Gool, Real-time connectivity constrained depth map computation using programmable graphics hardware. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1099-1104, 2005.

[6] R. Hartley and A. Zisserman, *Multiple View Geometry in computer vision.* Cambridge University Press, Second Edition, 2003.

[7] M. Pollefeys, Visual Modeling With A Hand-Held Camera, *International Journal of Computer Vision (IJCV)*, 59(3), 207-232, 2004.

[8] C. Zach, A. Klaus, and K. Karner, Accurate Dense Stereo Reconstruction using Graphics Hardware. In *Eurographics 2003*, pages 227-234, 2003.

[9] D. Shreiner , M. Woo, J. Neider and T. Davis, *OpenGL Programming Guide: The Official Guide to Learning OpenGL.* Addison-Wesley, 2003.

[10] S. Forstmann, J. Ohya, Y. Kanou, A. Schmitt, and S. Thuering, Realtime stereo by using dynamic programming. In *Proc. of CVPR Workshop on Real-time 3D Sensors and Their Use*, 2004.

[11] GPGPU, General-Purpose Computation Using Graphics Hardware homepage 2008. http://www.gpgpu.org/

[12] NVidia Cg programming language homepage, 2008. http://developer.nvidia.com/page/cg_main.html

[13] NVidia Cuda API technology homepage, 2008. http://www.nvidia.com/cuda

[14] OpenSceneGraph: an open source high performance 3D graphics toolkit, 2008. http://www.openscenegraph.org/

[15] Visual Geometry Group Home Page, University of Oxford, 2008. www.robots.ox.ac.uk/˜vgg/

[16] Marc Pollefeys' homepage, University of North Carolina at Chapel Hill, 2008. www.cs.unc.edu/˜marc/