USING GRAPHICS DEVICES IN REVERSE: GPU-BASED IMAGE PROCESSING AND COMPUTER VISION

James Fung[*]

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, California, USA
jfung@nvidia.com

Steve Mann

University of Toronto
Dept. of Electrical and Computer Engineering
10 King's College Road, Mailstop. B540
Toronto, Ontario, Canada
mann@eecg.toronto.edu

ABSTRACT

Graphics and vision are approximate inverses of each other: ordinarily Graphics Processing Units (GPUs) are used to convert "numbers into pictures" (i.e. computer graphics). In this paper, we discus the use of GPUs in approximately the reverse way: to assist in "converting pictures into numbers" (i.e. computer vision). For graphical operations, GPUs currently provide many hundreds of gigaflops of processing power. This paper discusses how this processing power is being harnessed for Image Processing and Computer Vision, thereby providing dramatic speedups on commodity, readily available graphics hardware. A brief review of algorithms mapped to the GPU by using the graphics API for vision is presented. The recent NVIDIA CUDA programming model is then introduced as a way of expressing program parallelism without the need for graphics expertise.

Index Terms— GPU, Graphics Processing Unit, Computer Vision, Image Processing

1. INTRODUCTION

It has often been said that Computer Vision and Computer Graphics are inverses of one another: "Image Synthesis" vs. "Image Analysis". Graphics can be viewed, at least at a high level, as an act of expanding input data. For example, a set of vertices and lighting specifications is *expanded* into megapixels worth of raster data whereas in vision, camera motion may be determined from pairs of images, taking megapixels of data and expressing the result in less than a dozen parameters.

Graphics hardware now incorporates a specialized processor called the Graphics Processing Unit, or "GPU". The GPU is a highly parallel architecture for performing the operations of computer graphics fast. As graphics algorithms became more sophisticated, there was a need to develop more flexible hardware and programming environments which led to the development of user-programmable hardware. The new flexibility combined with floating point capability and specialized performance over contemporaneous CPUs, led to work in *General Purpose GPU (GPGPU)* processing: using the graphics hardware to perform computations for tasks other than graphics.

This trend towards greater programmability has continued. New GPU architectures and programming tools have created a parallel programming environment this is no longer based on the graphics processing pipeline, but still exploit the parallel architecture of the GPU. This new approach to GPU programming for non-graphics tasks has been termed *GPU Computing*. To efficiently exploit this level of parallelism, mappings of algorithms onto the GPU must be found. This paper examines how GPU resources have evolved, along with mappings for vision, and analyzes the trend to GPU computing.

2. WHY USE THE GPU?

Graphics card based processing architectures benefit from "same day" architecting since the components are commodity priced and distributed. Additionally, once a vision algorithm is mapped onto the GPU, it will continue to benefit new GPU hardware improvements, typically released every 6 months to a year.

Highly Parallel Floating Point Computation Current GPUs have up to 128 processors capable of concurrent floating point operation which can accelerate computation many times faster than the CPU alone is capable of. [1]

Increased Memory Bandwidth. GPUs can be useful for applications such as pattern matching where a set of reference

[*]Both authors would like to thank NSERC, SSHRC, Canada Council for the Arts, Ontario Arts Council, Toronto Arts Council, and Ontario Graduate Scholarships, and Nikon Canada for support. Thanks to NVIDIA and Viewcast for equipment donations. Also thank you to all our past/present contributors and users.

[1]Many parallel floating point architectures have small deviations in floating point implementation. For example, architectures employ different handling of denormals. See for example the CUDA reference [1], or similar notes on rounding in the Cell processor's SPE units or floating point modes on the Altivec processor. Section 4 shows an application achieving speedup using single precision arithmetic while maintaining useful accuracy even compared to double precision.

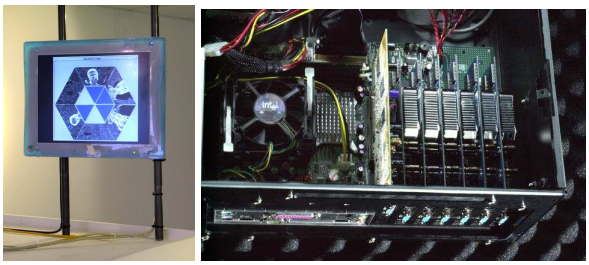

Fig. 1: Parallel GPU Computer Vision Machine. A computer vision application (2002) using the graphics hardware to display 6 video streams in a hexagonal form (upper left). The upper right shows the vision machine with 6 video capture cards, whose video is processed and then displayed with the hexagonal geometry using OpenGL. The lower image shows a computer vision machine with 6 PCI graphics cards, and 1 AGP graphics card. Each PCI card has a GeForce FX 5200 GPU which runs pattern recognition and computer vision tasks in parallel, creating a cheap, powerful, and easily constructible parallel architecture well suited for pattern recognition and computer vision.

data can be stored in the memory of each graphics card. On board graphics RAM ranges in size from 128MB to 1.5GB and is typically accessed by the GPU faster than the system RAM is accessed by the CPU. Current graphics card use GDDR3 or GDDR4 (Graphics Double Data Rate) RAM. The GeForce 8 GPU architecture achieves up to 70GB/sec. memory transfers on some applications as compared to an 800 MHz front side bus which peaks at 6.4GB/sec. Since each GPU is capable of accessing its own graphics memory independently of CPU intervention, each additional graphics card adds memory bandwidth to the overall system.

Multi-GPU Computation. Multiple graphics cards for processing computer graphics took on an early embodiment in the form of the "scan line interleaving". Fung and Mann presented a computer system using 6 PCI graphics cards in a single workstation working as a parallel video processing machine, exhibited at the "Deconference" exhibition [2] and later used to accelerate an eigenspace image recognition algorithm [3]. This architecture is shown in figure 1. With each card concurrently reading different sets of reference image data, the overall aggregate bandwidth of the system could be utilized. Multi-GPU programs explicitly send work to each card, viewing each GPU as a single processing resource and inter-card communication is achieved by reading back results to main memory before sharing them with other GPUs. Cluster based parallel graphics computations have also been explored by Fan et al. [4] and Horn et al. [5], in the Folding@Home application [6], and by Göddeke et al. [7].

In contrast to processing on the CPU, processing on the GPU requires that data be transferred to and from main memory via the system bus, termed "download" and "readback" respectively. However, many vision algorithms apply multiple iterations, or repeated processing stages to produce a final result. The GPU can render results of processing stages to off-screen memory on the card and thereby keep the processing local to the graphics hardware. Bus transfer overhead is then only incurred to transfer the initial image and final results. When bus communication is necessary, the graphics hardware typically is found on the fastest bus available. Currently these include the PCI Express bus, (4.0GB/s in each direction) and the PCI-E, Generation 2 bus (potentially doubling performance).

3. GPU COMPUTER VISION

GPUs have developed from fixed function architectures to programmable, multi-core architecture, leading to new applications (see figure 2). Owens et al. [8] present a broad survey of GPU computing. Here, we present a brief, but focussed, subset of work done in GPU vision and imaging. Under the old, fixed function hardware pipeline, only some operations were possible, and often only partial vision algorithms could be accelerated, leading to significant overhead when reading back at PCI speeds. For example, projective transforms could be performed by displaying texture mapped planes with the appropriate viewing geometry [9]. A method of tracking pose of a 3D object uses the GPU for dot products [10]. While useful for visualization, the 8-bit output induced error in computations, such as with wavelet analysis [11]. 3D Depth estimation and scene reconstruction is also presented in [13] on a GeForce 4 creating depth maps quickly by utilizing the bilinear and trilinear interpolation in texture hardware.

Graphics card manufacturers began producing hardware with programmable shading operations. The operation of texturing/colouring fragments to be displayed could now be controlled by short, user written programs. Additionally, floating point arithmetic became available. By using programmable shaders to conduct summations across images, entirety of a projective image registration algorithm [9] was mapped onto graphics hardware [14]. This was later used to create applications in Mediated Reality [15] where multiple graphics cards were used in parallel to run multiple algorithms concurrently and maintain interactive framerates.

Strzodka et. al [16] implement a motion estimation algorithm that provides dense estimates from optical flow. They achieve a 2.8 times speedup on a GeForce 5800 Ultra GPU over a optimized Pentium 4 CPU implementation.

Images can be mapped efficiently into transform spaces using the drawing functionality and pixel counting hardware implementing the Hough transform for lines [17] and circles [18]. Feature extraction and description have also been mapped to provide interactive framerates [19, 20].

The combination of display and processing provided by graphics hardware was applied to a projector-camera system [21]. Feeding back images locally on the card allowed the application to perform image processing and display the results across two heads, without the requirement to read back data. Similarly, Sherbondy et al. [22] present both volume segmentation and visualization on graphics hardware.

With the programmable features, more complex depth map computations could be conducted. Brunton et al. [23] use the GPU to compute depth or disparity maps using belief propagation methods. Woetzel and Koch [24] present 3D reconstruction with special attention given to depth discontinuities. Computations can benefit from being split between

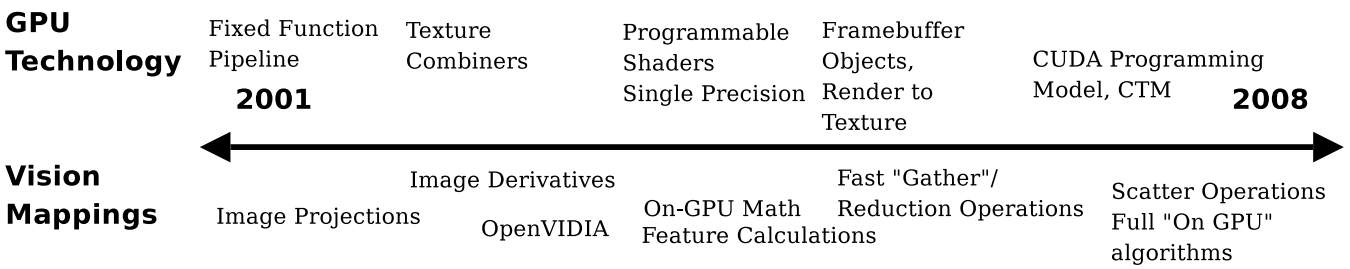

Fig. 2: GPU Technology & Computer Vision. GPU hardware and software architectures have developed to provide increased flexibility allowing more vision and imaging algorithms to be mapped on the hardware. This shows a rough timeline of functionality and a some techniques and applications they enabled.

the CPU and GPU such as in a graph cut algorithm [25], dynamic programming [26], or a GPU Adaboost implementation [27].

While fragment shaders are designed to operate on all output fragments, there are also hardware methods for restricting computation to pixels of interest. Hadwiger et al. [28] present a review of GPU image segmentation in medical imaging applications for data visualization. where un-necessary computations are avoided by use of an early z-cull.

Libraries of computer vision and image processing have been developed in the OpenVIDIA [17] project and GPUCV [29]. This has aided development of higher level algorithms. OpenVIDIA provides a framework for video input, display, and programming GPU processing, as well as implementations of feature detection and tracking, skin tone tracking, and projective panoramas. Real-time blink detection [30] is accelerated with OpenVIDIA. GPUCV is designed to provide seamless acceleration with the familiar OpenCV interfaces.

4. NEW GPU ARCHITECTURAL FEATURES FOR VISION AND IMAGING

More recently, a new architecture and programming model has exposed more flexibility on GPU hardware. The Compute Unified Device Architecture (CUDA) [1] allows programmers to write C programs which no longer require knowledge or dependence on the graphics pipeline. For instance, to move data to and from the GPU, familiar *malloc* and *memcpy* APIs are provided and processing occurs in *kernels*, which, written in C, perform calculations similar to the *for* loop bodies. Pointer arithmetic, scatter and gather memory accesses allows frequency space transforms [31] and processing, feature point processing [20] and histogram processing - previously difficult to map to or inefficient on GPUs. Faster memory shared between groups of processors can act as a programmable cache, accelerating spatially local operations. A high level programmers view of the CUDA memory architecture is shown in figure 3, and mapping techniques are discussed in [32]. Optical flow has been implemented on CUDA [33]. Figure 4 shows a GPU implementation of Lucy–Richardson deconvolution compared with a CPU version.

Spatially coherent memory accesses, common to vision and imaging, are are cached locally, in 2-D to accelerate reads. When dealing with camera data in the form of byte elements, the GPU can perform hardware data conversion from 8 or 16 bit integer values to floating point during the fetch. When fetching, the architecture can sample with bilinear interpolation, useful in stereo matching. Writing code to handle variable sized images is made easier via texture normalized texture addresses $\in [0, 1]$. Convolution operations at image edges can be made easier by wrap and clamping modes of operation at texture borders.

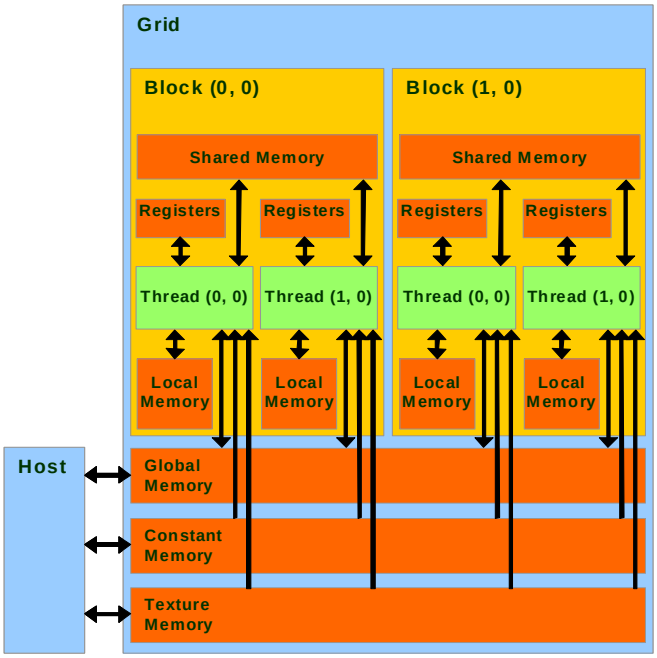

Fig. 3: CUDA Memory Programming Model. This memory hierarchy allows concurrent threads to share information and GPU memory can allocated, read from, and written to using familiar *malloc* and *memcpy* commands, and accessed as an array.

Current GPUs provide two features which ameliorate download and readback time overhead. Firstly, *pinned memory* can be allocated which is non-pageable, and can then be transferred using hardware DMA. Secondly, processing times and transfer times can be overlapped by the use of *streams*. Similarly GPU processing can by launched asynchronously allowing the CPU to perform other computation in parallel with GPU processing. GPU Computing architecture and software no longer require knowledge/overhead of the graphics processing pipeline. Rather the challenge now lies in

Original Blurred CPU GPU

Lucy-Richardson Deconvolution Config.	Steps	CPU Time (s)	GPU Time (s)	Speedup
No damping/weighting	10	40.4	4.14	$9.8\times$
With damping/weighting	20	102	4.87	$21\times$

Fig. 4: CUDA Implementation of Lucy Richardson Deconvolution. Processing times are shown for a 1024×1024 color image, run for reference on an Intel Xeon E5230, 1.86 GHz in Matlab (no IPP acceleration) at double precision. An NVIDIA Quadro 5600 (single precision only) was used for the GPU. In all cases the GPU was accurate to the CPU version within ± 1 grey level in any channel. The work was done with the CUDA FFT package and custom kernels.

expressing algorithms in a highly parallel fashion. Naive implementations typically can result in a $2-3\times$ speedup, however, further optimizations can easily result in speedups beyond $10\times$. There is still, then, much room for exploration in mapping algorithms onto a massively parallel architecture.

5. REFERENCES

[1] NVIDIA Corporation, "NVIDIA CUDA compute unified device architecture programming guide," Jan. 2007, `http://developer.nvidia.com/cuda`.

[2] "Taking liberties," *Azure Magazine*, pp. 32–33, November–December 2002.

[3] James Fung and Steve Mann, "Using multiple graphics cards as a general purpose parallel computer : Applications to computer vision," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR2004)*, Cambridge, United Kingdom, August 23–26 2004, vol. 1, pp. 805–808.

[4] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover, "GPU cluster for high performance computing," in *Proc. ACM / IEEE Supercomputing Conference 2004*, Pittsburgh, PA, 2004.

[5] Daniel Reiter Horn, Mike Houston, and Pat Hanrahan, "ClawHMMER: A streaming HMMER-search implementation," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov. 12-18 2005, pp. 11–11.

[6] Vijay Pande and Stanford University, "Folding@Home on ATI GPUs," 2006, `http://folding.stanford.edu/FAQ-ATI.html`.

[7] Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Sven H.M. Buijssen, Matthias Grajewski, and Stefan Turek, "Exploring weak scalability for FEM calculations on a GPU-enhanced cluster," *Parallel Computing*, vol. 33, no. 10–11, pp. 685–699, 2007.

[8] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[9] James Fung, Felix Tang, and Steve Mann, "Mediated reality using computer graphics hardware for computer vision," in *Proceedings of the International Symposium on Wearable Computing 2002 (ISWC2002)*, Seattle, Washington, USA, Oct. 7 – 10 2002, pp. 83–89.

[10] W. Michael, H. Ron, and K. Paul, "Alignment and tracking using graphics hardware," in *In Image Understanding Workshop, pages 837–842. DARPA*, 1996.

[11] M. Hopf and T. Ertl, "Hardware-Based Wavelet Transformations," in *Workshop of Vision, Modelling, and Visualization (VMV '99)*. 1999, pp. 317–328, infix.

[12] Ruigang Yang and M. Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE*, vol. 1, pp. 211–217, 2003.

[13] Ruigang Yang, Marc Pollefeys, Hua Yang, and Greg Welch, "A unified approach to real-time, multi-resolution, multi-baseline 2D view synthesis and 3D depth estimation using commodity graphics hardware," *International Journal of Image and Graphics*, vol. 4, no. 4, pp. 627–651, 2004.

[14] James Fung and Steve Mann, "Computer vision signal processing on graphics processing units," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, Montreal, Quebec, Canada, May 17–21 2004, pp. 83–89.

[15] Rosco Hill, James Fung, and Steve Mann, "Reality window manager: A user interface for mediated reality," in *Proceedings of the 2004 IEEE International Conference on Image Processing (ICIP2004)*, Singapore, October 24-27 2004.

[16] Robert Strzodka and Christoph Garbe, "Real-time motion estimation and visualization on graphics cards," in *Proceedings IEEE Visualization 2004*, 2004, pp. 545–552.

[17] James Fung, Steve Mann, and Chris Aimone, "OpenVIDIA: Parallel GPU computer vision," in *Proceedings of the ACM Multimedia 2005*, Singapore, Nov. 6-11 2005, pp. 849–852.

[18] M. Ujaldn, A. Ruiz, and N. Guil, "On the computation of the circle hough transform by a GPU rasterizer," *Pattern Recognition Letters*, vol. 29, no. 3, pp. 309–318, 2008.

[19] James Fung, "Chapter 40: Computer vision on the GPU," in *GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation*, pp. 649–665. Addison-Wesley, 2005.

[20] Sudipta N. Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, 2007.

[21] Matthew Flagg, Jay Summet, and James M. Rehg, "Improving the speed of virtual rear projection: A GPU-centric architecture," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, 2005, vol. 3, p. 105.

[22] Anthony Sherbondy, Mike Houston, and Sandy Napel, "Fast volume segmentation with simultaneous visualization using programmable graphics hardware," in *IEEE Visualization 2003*, Oct. 2003, pp. 171–176.

[23] Alan Brunton, Chang Shu, and Gerhard Roth, "Belief propagation on the GPU for stereo vision," in *3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, 2006, vol. 0, p. 76.

[24] Jan Woetzel and Reinhard Koch, "Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware," in *Proc. IEEE 17th International Conference on Pattern Recognition (ICPR 2004)*, Cambridge, United Kingdom, August 2004.

[25] I. Geys, T.P. Koninckx, and L. Van Gool, "Fast interpolated cameras by combining a GPU based plane sweep with a max-flow regularisation algorithm," in *Proceedings of second international symposium on 3D Data Processing Visualization & Transmission - 3DPVT04*, Thessaloniki, Greece, 2004, pp. 534–541.

[26] Minglun Gong and Yee-Hong Yang, "Near real-time reliable stereo matching using programmable graphics hardware," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 1, pp. 924–931.

[27] Hicham Ghorayeb, Bruno Steux, and Claude Laurgeau, "Boosted algorithms for visual object detection on graphics processing units," *Lecture Notes in Computer Science, ACCV 2006*, vol. 3852/2006, pp. 254–263, 2006.

[28] Marksu Hadwiger, Caroline Langer, Henning Scharsach, and Katja Bhler, "State of the art report 2004 on GPU-based segmentation," Technical Report TR-VRVis-2004-017, VRVis Research Center, Vienna, Austria, 2004.

[29] Farrugia J.-P., Horain P., E. Guehenneux, and Y. Alusse, "Gpucv: A framework for image processing acceleration with graphics processors," *Multimedia and Expo, 2006 IEEE International Conference on*, pp. 585–588, July 9-12 2006.

[30] Lalonde, Byrns, Gagnon, Teasdale, and Laurendeau, "Real-time eye blink detection with gpu-based sift tracking," *CRV*, vol. 00, pp. 481–487, 2007.

[31] NVIDIA Corporation, "CUDA CUFFT library programming guide," Oct. 2007, `http://www.nvidia.com/object/cuda_develop.html`.

[32] Mark Harris, "Mapping computational concepts to GPUs," in *GPUGems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Matt Pharr, Ed., chapter 31, pp. 493–508. Addison-Wesley, 2005.

[33] Yoshiki Mizukami and Katsumi Tadamura, "Optical flow computation on compute unified device architecture," *ICIAP*, vol. 0, pp. 179–184, 2007.