ACHIEVING HIGH-RESOLUTION VIDEO USING SCALABLE CAPTURE, PROCESSING, AND DISPLAY

Donald Tanguay
HP Labs, Palo Alto, CA & Stanford University, Stanford, CA, USA

H. Harlyn Baker, Dan Gelb
Hewlett-Packard Laboratories, 1501 Page Mill Rd, Palo Alto, CA, USA

Keywords: Camera arrays, Chip multiprocessors, GPUs, Mosaicking, Video processing.

Abstract: New video applications are becoming possible with the advent of several enabling technologies: multicamera capture, increased PC bus bandwidth, multicore processors, and advanced graphics cards. We present a commercially-available multicamera system and a software architecture that, coupled with industry trends, create a situation in which video capture, processing, and display are all increasingly scalable in the number of video streams. Leveraging this end-to-end scalability, we introduce a novel method of generating high-resolution, panoramic video. While traditional point-based mosaicking requires significant image overlap, we gain significant advantage by calibrating using shared observations of lines to constrain the placement of images. Two non-overlapping cameras do not share any scene points; however, seeing different parts of the same line does constrain their spatial alignment. Using lines allows us to reduce overlap in the source images, thereby maximizing final mosaic resolution. We show results of synthesizing a 6 megapixel video camera from 18 smaller cameras, all on a single PC and at 30 Hz.

1 INTRODUCTION

Several industry trends are creating new opportunities for video processing on commodity PCs. First, PC bus bandwidth is increasing from 2 Gbit/sec to 40 Gbit/sec, with 80 Gbit/sec planned. Second, graphics cards have become very powerful general-purpose computing platforms that also support high-bandwidth display. Finally, processor manufacturers have begun delivering chip multiprocessors for parallel execution. In particular, they can no longer exclusively rely on smaller transistors and higher clock speeds to improve performance. Due to the economics of their current strategy, manufacturers have turned their attention to multiple-core processors (Gibbs, 2004). For example, Sun's Niagara processor has 32 processing cores (Kongetira, 2005).

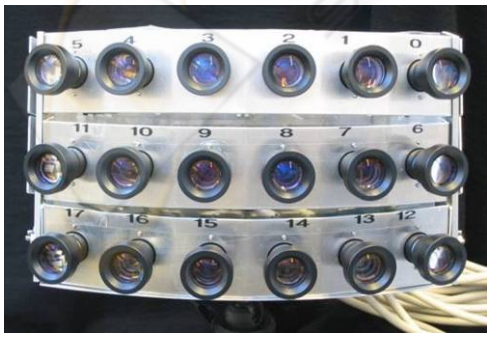


Figure 1: A mosaic camera is one type of new video application enabled by current imaging and computing trends.

Along with these opportunities for video computing come two significant challenges. The first challenge is the difficulty of obtaining good quality video. This is due to a market focus on both low quality webcams and expensive, non-scalable machine vision systems. To address this problem we have designed a capture system that uses the PCI-x bus to stream many video cameras directly into the memory of a single PC. Figure 1 shows an example arrangement of many cameras for video mosaicking.

A second significant challenge is that successful multiprocessing requires multithreaded programming. This is difficult and error prone. In fact, a

Tanguay D., Harlyn Baker H. and Gelb D. (2006).
ACHIEVING HIGH-RESOLUTION VIDEO USING SCALABLE CAPTURE, PROCESSING, AND DISPLAY.
In *Proceedings of the First International Conference on Computer Vision Theory and Applications*, pages 162-169
DOI: 10.5220/0001374301620169

startling result of the multicore trend is that most existing applications will no longer benefit from future processor advances (Gibbs, 2004) unless they are redesigned (McDougall, 2005). While most applications have multiple threads, the vast majority are not performing compute-intensive operations in parallel. While video applications have large computing needs, they can often benefit from multiprocessing. To address this computing challenge, we have developed a cross-platform software framework specifically designed to perform video processing on a multiprocessor.

Such scalable capture and processing systems enable new video applications. For our first effort, we have chosen to develop a high-resolution video camera from many smaller ones (Figure 1). A trend in digital photography is that still cameras can now have upwards of 12 megapixels of resolution, with 5 megapixel cameras being common. Video cameras, however, haven't attained these resolutions. Expensive HDTV cameras have the most resolution, at about 2 megapixels. Our goal is to significantly exceed current video resolutions.

A range of applications could benefit from such a high-resolution video camera. For example, surveillance systems need enough resolution over a large area to identify suspects. As another example, an interactive desk-space may need to image an entire tabletop at high resolution in order to digitize documents placed anywhere on the desk.

2 RELATED WORK

Other systems of high-bandwidth video capture (Kanade 1995, Wilburn 2002) have been developed. Thus far, however, these systems work offline or use lossy compression to satisfy bandwidth constraints. Besides mosaicking, other novel applications of multicamera video being pursued are synthesizing a camera with a very high frame rate (Wilburn, 2004) and making one with an enormous synthetic aperture for selective depth of field (Levoy, 2004).

Most mosaicking methods use point correspondences to constrain image alignment. In digital photography, panoramic mosaics (Peleg 1997, Sawhney 1998, Shum 2000) are derived from the motion of a single hand-held camera. In photogrammetry, aircraft and satellites capture images which are stitched together to produce photographic maps. Having large areas of overlap (typically 20-50%), these solutions are generally not effective for a rigid camera arrangement because this overlap reduces total resolution. They also typically depend on a scene's visual complexity since they require distinguishable features from the content itself. Because our cameras do not move relative to each other, we can calibrate the system beforehand.

Our video processing framework is inspired by early dynamic dataflow computers (Arvind, 1984) which potentially exploit the full parallelism available in a program. In such a computer, each processing node is enabled when tokens with identical tags are present at each of its inputs. Thus, process scheduling is completely determined by the availability of data. Our software framework emulates this behaviour on a multiprocessor. While a dataflow computer can achieve fine-grained parallelism at the instruction level, our framework operates at a much coarser granularity such as a single video frame.

Signal processing software environments (such as Ptolemy (Buck, 1994) and Khoros (Rasure, 1991) have an established history of "visual dataflow programming." Others have presented a thorough review of such systems and their relationship to other dataflow styles (Lee, 1995). The one-dimensional, fine-grained, deterministic nature of audio processing often allows optimal scheduling at compilation time. In video and vision processing, however, the mapping of inputs to outputs is often nondeterministic (*e.g.*, face detection vs. 1-D convolution), and the sample size is larger (*e.g.*, 30 Hz video vs. 44 KHz audio).

Some commercial frameworks are available for multimedia processing, including DirectShow (Microsoft) and the Java Media Framework (Sun). DirectShow is designed for plug-and-play compatibility between third-party developers. For example, a commercial video conferencing application can transparently use any particular video camera if both manufacturers adhere to a common DirectShow interface. Unfortunately, constructing new modules is painfully difficult, and it is a Windows-only system. By encapsulating a simple DirectShow application inside a software module, we leverage DirectShow for its real strength – to access third party devices. The Java Media Framework has cross-platform support, as well as integrated networking support via RTP; however, its video processing performance is not competitive.

Distributed computing extends the dataflow approach from a single machine to a network of machines. The Berkeley Continuous Media Toolkit

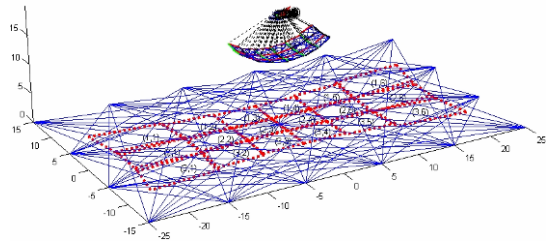


Figure 2: Imaging geometry of the 3x6 mosaic camera. The 18 side-by-side imagers (top) are aligned using common observations of lines on a plane (shown in blue). The individual camera fields of view (outlined in red) illustrate minimal image overlap.

(Mayer-Patel, 1997) is multi-platform and uses Tcl/TK. The open-source Network-Integrated Multimedia Middleware project (Lohse, 2002) has demonstrated applications with set-top boxes and wireless handhelds. Unlike our framework, these approaches do not have single-machine performance features, such as a dataflow scheduler. However, we believe a distributed computing approach can be very complementary to our framework.

Finally, the power of graphics cards is being used in many other applications. For example, GPUs are being used to perform a real-time plane-sweep stereo computation (Yang, 2004). Furthermore, the GPGPU organization (GPGPU) promotes general purpose computing on graphics cards.

3 MOSAIC CAMERA

A mosaic camera is a composite of many other cameras. Calibration is the process of determining the parameters that map these cameras into a seamless image. We describe the calibration problem, formulate the solution as a global minimization, and describe how to calibrate in practice.

3.1 Problem Statement

The goal of our video application is to produce high resolution video from many cameras. We have narrowed our focus with two requirements. First, to produce resolution that scales with the number of cameras, we fully utilize native resolution by "tiling" the cameras. In other words, we require that camera images overlap only to ensure spatial continuity in the final mosaic. Super-resolution techniques, on the other hand, require complete overlap and face theoretical limits on resolution (Robinson, 2003).

Second, we facilitate an initial solution by using a linear homographic model for mapping each camera into a common imaging plane to produce the final mosaic. This homographic model supports two types of scenario: those where the imagers have a common centre of projection (*i.e.*, pure camera rotation), and those where the scene is basically planar (*e.g.*, imaging a white board). For the latter scenario, we can increase the range of valid depths by increasing the distance from the cameras to the scene, or by reducing the separation between camera centres. In fact, continued reduction in camera sizes improves the validity of the homographic model.

Our calibration methodology is as follows: (1) Place cameras in an arrangement so that they cover the desired field of view, while minimizing overlap. (2) Select one camera C_0 as the reference camera (typically, a central one), with its image plane becoming the imaging plane of the final mosaic. (3) Estimate a homographic mapping ${}^0_i\mathbf{H}$ for each camera C_i that maps its image into camera C_0 to produce a single, coherent mosaic image.

3.2 Line-Based Solution

We calibrate the cameras using line correspondences. Using lines has two major benefits. First, a line can be estimated more accurately than a point. It can be localized across the gradient to subpixel accuracy (Canny, 1986), and the large spatial extent allows estimation from many observations along the line. Second, two cameras can observe the same line even without any overlapping imagery. This profoundly increases the number of constraints on a solution because non-neighbouring cameras can have common observations.

We have developed a bundle adjustment formulation to minimize the geometric error in the mosaic image by simultaneously estimating both the homographies of each camera and the line models behind the observations. The nonlinear least-squares formulation is

$$\arg\min_{{}^0_i\hat{\mathbf{H}},\,{}^0\hat{\mathbf{l}}_j,\,i\neq 0} \sum_{i,j} d\left({}^i\hat{\mathbf{l}}_j, {}^i\tilde{\mathbf{l}}_j\right)^2, \qquad (1)$$

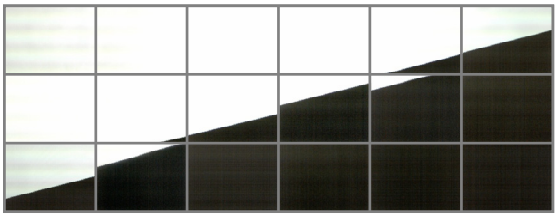


Figure 3: A single calibration projector image, as seen from all 18 cameras.

where the estimated parameters are the ideal point homographies ${}^{0}_{i}\hat{\mathbf{H}}$ from the cameras C_i to the reference camera C_0 and the ideal lines ${}^{0}\hat{\mathbf{l}}_j$, expressed in the coordinates of the reference camera. The function $d()$ measures the difference between ${}^{i}\tilde{\mathbf{l}}_j$, the observation of line $\mathbf{l}_j$ in camera C_i, and its corresponding estimate ${}^{i}\hat{\mathbf{l}}_j$. This measure can be any distance metric between two lines. We chose the perpendicular distance between the ideal line and the two endpoints of the measured line segments. This is a meaningful error metric in the mosaic space, and it is computationally simple. More specifically, we have selected $d()$ so that the bundle adjustment formulation of Equation 1 becomes

$$\underset{{}^{0}_{i}\hat{\mathbf{H}},\,{}^{0}\hat{\mathbf{l}}_j,\,i\neq 0}{\arg\min} \sum_{i,j} \left({}^{0}\hat{\mathbf{l}}_j^{T} \cdot {}^{0}_{i}\hat{\mathbf{H}} \cdot {}^{i}\mathbf{p}_j\right)^2 + \left({}^{0}\hat{\mathbf{l}}_j^{T} \cdot {}^{0}_{i}\hat{\mathbf{H}} \cdot {}^{i}\mathbf{q}_j\right)^2 \quad (2)$$

where ${}^{i}\mathbf{p}_j$ and ${}^{i}\mathbf{q}_j$ are the intersection points of the measured line with the boundary of the original source image.

The domain of the error function (Equation 2) has $D = 9(N_c - 1) + 3L$ dimensions, where N_c is the number of cameras and L is the number of lines. Our current setup has 18 cameras and 250 lines, or 903 parameters. We use Levenberg-Marquardt minimization to find a solution to Equation 2, and we use a novel linear method to initialize the parameters (Authors).

3.3 Calibration Method

We use a digital projector as a calibration instrument. The projector displays a series of lines one-by-one. Cameras that see a line at the same instant j are actually viewing different parts of the same line $\mathbf{l}_j$ and therefore have a shared observation. Image analysis determines line equations ${}^{i}\tilde{\mathbf{l}}_j$, and this is intersected with the image boundary to find the points ${}^{i}\mathbf{p}_j$ and ${}^{i}\mathbf{q}_j$. After all observations are collected, the solution of Equation 2 is found, paying careful attention to parameter normalization.

Figure 4: Our capture hardware currently supports 22 Bayer-format VGA cameras at 30 Hz, without compression.

Image analysis easily dominates the calibration time. A simple approach is to sample the space of lines "evenly" by enumerating all pairs of points evenly distributed around the perimeter of the projector image. For the 3x6 mosaic camera, a 1024x768 projector, and points spaced 150 pixels apart, calibration presents 253 calibration images and processes 4554 total images. Reducing the number of line images will significantly reduce calibration time.

We have devised an alternate approach that adapts to the number of constraints accumulated thus far. Addressing horizontal (vertical) lines separately, we perform a breadth-first search in a binary tree in which each node defines an interval, and the two children define the top and bottom (left and right) of the parent's interval. Starting at the root, which contains the entire image, each visited node represents a line drawn across (down) the middle of the interval. The sequence becomes: one line, two lines offset a quarter from opposite edges, *etc*. The search is terminated when every camera has at least two lines in common with another camera. This coarse to fine method reduces the number of calibration image to 60, so that the calibration processes 1080 total images, a 76% reduction.

4 SCALABLE CAPTURE

The camera system (Figure 4) streams synchronized video to PC memory through a tree structure with three layers: camera, concentrator, and frame grabber.

The camera package is a 30 mm cube consisting of three small boards: a color VGA CMOS sensor, a

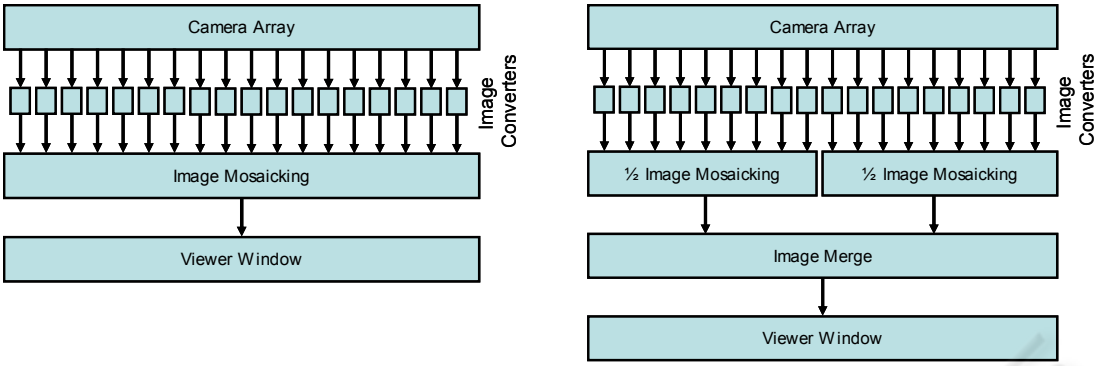


Figure 5: Leveraging chip multiprocessors with a flexible software design. The data flow structure of an application reveals opportunities for parallel computation. The straightforward mosaic application (*left*) has task parallelism across the image converters. Data parallelism can concurrently execute the monolithic mosaicking module, possibly increasing latency. Alternatively, a developer can divide the mosaic module (right) into multiple sections and therefore increase task parallelism.

co-processor, and a low-voltage differential-signalling (LVDS) chip interfacing to the rest of the system. It is connected to a concentrator board through a 10-wire UTP connector over an LVDS bus that carries captured video data, control signals, and power. An FPGA is available for truly scalable on-camera processing (not exploited in this work).

The concentrators and frame grabber use Altera Flex FPGAs, and the frame grabber is built on a QuickLogic 5064 64-bit DMA engine designed for a 64-bit PCIx bus. The DMA engine delivers 24 Bayer-format VGA video streams directly into RAM at 30 Hz, without causing any load on the CPU. Our interface card currently operates at 33MHz on this bus, with theoretic utilization of about 0.25GB/sec. We plan extensions to PCI-express which will provide 4.4 GB/sec, or 350 VGA streams at 30 Hz.

Our hardware collaborator is now commercial-izing the system. More details are available (Authors).

5 SCALABLE PROCESSING

Video processing performance becomes more scalable if multiprocessing can take advantage of inherent parallelism in an application. We have built a cross-platform software framework (Authors) to build multimedia applications. We now describe how to use such an architecture to achieve computational scalability in video applications.

5.1 Software Dataflow

A dataflow architecture naturally models the streaming nature of real-time rich media applications. Simply stated, a dataflow architecture is one based on sequential transformations of data. By observing the lifecycle of signals in the application (from production through transformation to consumption), one can define a pipeline of distinct processing stages. These stages can be expressed as a directed graph of processing modules. The application, then, is a connected graph of functional modules linked together by directed arcs. This modular graph structure reveals opportunities for parallel computing. Figure 5 shows two dataflow graphs for the mosaic camera.

Our software framework uses the dataflow paradigm to isolate the algorithms (*e.g.*, video processing or analysis) from the runtime system (*e.g.*, multithreading, synchronization, performance measures). The developer concentrates on the algorithmic processing specific to the application at hand, while at the same time leveraging the framework to address performance issues. More specifically, our architecture provides a framework for (1) decomposing an application's processing into task dependencies, and (2) automating the distribution and execution of those tasks on a multiprocessor machine to maximize performance.

5.2 Parallelism

Our architecture has facilities to take advantage of both task and data parallelism. In *task parallelism*, each processing module is treated as *sequential*, that is, that each input must be processed in order of arrival. Sequential modules have internal memory, and so the output may depend both on the current input and previous inputs. An example sequential module is one that tracks coordinates of a hand, where the location in the previous frame initializes the search for the hand location in the current frame. At most one execution thread may be resident in a sequential module at any given instant. A computer with processors equal to the number of modules will reach the limit of usable task parallelism. Each module essentially has its own processor, and additional processors can not improve performance. In fact, the application throughput is limited by the latency of the slowest module. Figure 6 (*middle*) illustrates execution using task parallelism.

More significantly, our architecture can also use *data parallelism*, which has the potential of increasing performance linearly with the number of processors. The scheduler uses data parallelism on modules specified as *combinational*, or stateless. In other words, these modules do not have any internal history of previous executions. An example combinational module is one that compresses images to JPEG. Because each conversion is independent of any other, the code of the converter can be executed simultaneously by multiple threads on different data. Being combinational is particularly key for a module that is a performance bottleneck (*i.e.*, one that has the largest latency).

Figure 5 shows two versions of the CPU-based

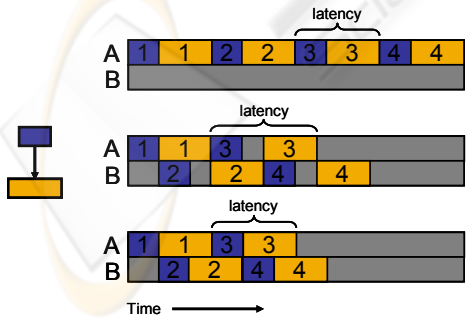


Figure 6: Parallel execution. On a dual processor system, a program can be executed sequentially (*top*), with task parallelism (*middle*), or with data parallelism (*bottom*), changing the latency and throughput of the application.

mosaicking application. The straightforward implementation (*left*) exhibits significant parallelism among the converters. Because the mosaic module is combinational, multiple instances of it can be executed concurrently via data parallelism. Alternatively, the large mosaic module can be divided spatially so that two modules each work on one half of the mosaic (Figure 5, *right*). A separate lightweight module is introduced to combine the results.

5.3 Performance Tuning

Because the application graph is a pipeline architecture, it exhibits the usual latency/throughput trade-offs. At the extremes, the single processor machine will have lower latency and throughput, and a multiprocessor with data parallelism may have higher latency but similar or better throughput. Constraints on particular applications determine the appropriate latency/throughput trade-off. Our architecture provides simple mechanisms to change the execution behaviour to explore these performance trade-offs.

Our framework assumes data and processing have coarse granularity. That is, media samples generally are at least a kilobyte in size and have a duration of at least a millisecond. Our framework is not designed for implementing integer addition, for example, because the fine granularity reveals the multiprocessing overhead (*e.g.*, data transfer, threading, and synchronization).

6 SCALABLE DISPLAY

Of the three key technologies enabling new video applications, graphics cards are the most accessible. Driven by the demands of the gaming market, a modern graphics card has both a high-bandwidth connection to main memory and impressive processing capabilities.

The high-bandwidth connection to the host CPU's main memory is either AGP or PCI, both delivering 5-16 Gbits/sec. This easily supports up to hundreds of uncompressed colour VGA video streams. Drawing video on a graphics card entails sending it the latest frame, storing this as a texture, and drawing the texture through the normal graphics transformation pipeline. Amazingly, the latest graphics processors have 200 gigaflops of

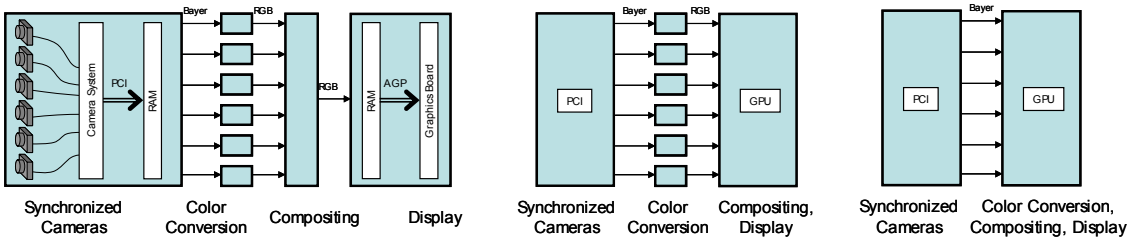
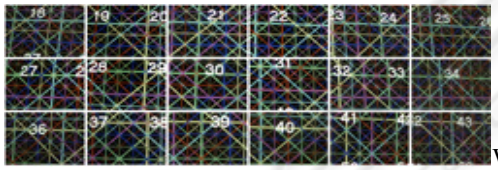


Figure 7: GPUs and CPUs. With an appropriate formulation, graphics processors can assist GPUs. For limited situations of the mosaic camera, the graphics card can display video (*left*), compute the composite image (*middle*), and convert image formats (*right*).

computing power. These GPUs have been heavily optimized for frame-based computing. To leverage the full capabilities, one must formulate a problem as a graphics rendering (GPGPU).

Developers now have the option of processing on either the host PC or a graphics card. In our mosaic camera, for example, we have two ways of using the graphics card. First, ignoring lens distortion and other nonlinear effects, the homographic mapping may have the same parameterization as drawing textures onto a quadrilateral. Figure 7 shows a reduced software pipeline using the GPU to mosaic the images. Second, because our single-sensor cameras deliver the original Bayer colour image, the GPU shader can be programmed to perform the colour sampling. Figure 7 depicts the further reduced software pipeline, in which the CPU does nothing at all. Note that these types of optimizations are not always possible.

A display could also be scalable in the number of pixels. In a similar fashion to the mosaic camera, a tiled projector display would dramatically increase the display resolution and allow presentation of a mosaic video. We have begun efforts on this problem.



We have built an end-to-end system and now show initial results. Figure 8 shows the mosaic results of the cameras looking at a patterned wall. The pattern is an arrangement of coloured lines with white numbers as landmarks. The input images are shown in Figure 8 (*left*), where they are tiled in the same 3x6 arrangement as the cameras. Figure 8 (*right*) shows the final result. The solution has a maximum error of 1.2 pixels, or 0.04% of the mosaic width of 3600.

Remarkably, the six left cameras are perfectly aligned with the rest of the mosaic in spite of the gap between these two sets of images. Although the two sets of cameras do not share any 3-D scene points, they do have enough information from observing different parts of the same lines.

The best performance for the mosaic camera came with the computing configuration of Figure 7 (*right*). We achieved 30 Hz mosaic video using an NVIDIA FX3400 graphics card on a dual processor machine. Multicore results are expected by press time.

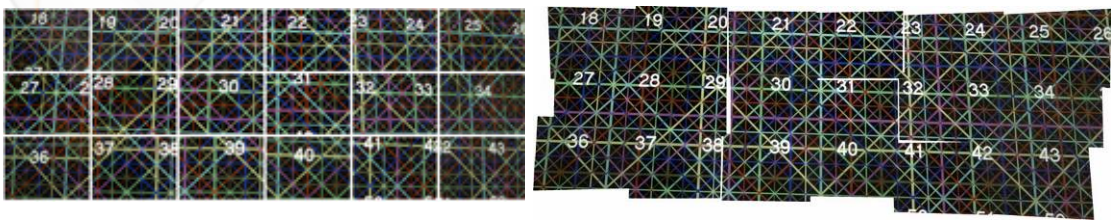


Figure 8: Results from a 3x6 mosaic camera. 18 VGA images (*left*) are combined into one 3600x1400 mosaic (*right*). The scene is a projected image with coloured lines and numbered landmarks. Notice that all the lines are straight in the result.

8 CONCLUSIONS

We are entering an exciting period in which capture, processing, and display are increasingly scalable in the number of video streams supported. In support of this view, we capture 24 uncompressed VGA streams in real-time on a single PC and leverage multiprocessors for video processing. Coupling these developments with recent commercial hardware advances brings forth end-to-end scalability that enables new applications of camera arrays and other multi-video sources. We selected a mosaic camera as our first application in this space.

We presented several innovations in this paper. First, we described how to use a digital projector as a calibration instrument for a mosaic camera. We also presented an adaptive technique for reducing the number of calibration images using the solution constraints accumulated thus far. We described how the mosaic camera is built using our camera array, and how our dataflow architecture combined with chip multiprocessors and graphics cards present new abilities and options for mosaic processing. Finally, we showed initial results for 18 cameras with small error and real-time performance.

REFERENCES

Authors. References withheld to preserve anonymity during review process.

Arvind, D. Culler, R. Iannucci, V. Kathail, K. Pingali, R. Thomas, 1984. The tagged token dataflow architecture. *Technical report, MIT Laboratory for Computer Science*.

Buck, J., S. Ha, E. Lee, and D. Messerschmitt, 1994. Ptolemy: A framework for simulating and prototyping heterogeneous systems. In *International Journal of Computer Simulation*, April 1994.

Canny, J., 1986. A computational approach to edge detection. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-698.

Gibbs, W. Wayt, 2004. A split at the core. In *Scientific American*, Nov. 2004.

Gortler, S.J., R. Grzeszczuk, R. Szeliski, M.F. Cohen, 1996. The Lumigraph, In *Proc. ACM SIGGRAPH*, New Orleans, USA.

GPGPU. General-purpose computation using graphics hardware. http://www.gpgpu.org.

Hartley, R., A. Zisserman, 2000. *Multiple view geometry in computer vision*, Cambridge University Press.

Kanade T., P.J. Narayanan, P.W. Rander, 1995. Virtualized reality: Concepts and early results. In *Proc. IEEE Workshop on Representation of Visual Scenes*.

Kongetira, P., K. Aingaran, K Olukotun, 2005. Niagara: A 32-way multithreaded SPARC processor. In *IEEE Micro*, vol. 25, no. 2, pp. 21-29.

Lee, E., T. Parks, 1995. Dataflow Process Networks. In *Proceedings of the IEEE*, May 1995.

Levoy, M., B. Chen, V. Vaish, M. Horowitz, L. McDowall, M. Bolas, 2004. Synthetic aperture confocal imaging. In *ACM Trans on Graphics (SIGGRAPH 2004)*.

Levoy, M., P. Hanrahan, 1996. Light field rendering. In *Proc. ACM SIGGRAPH*, New Orleans, USA.

Lohse, M., M. Repplinger, P. Slusallek, 2002. An open middleware architecture for network-integrated multimedia. In *Protocols and Systems for Interactive Distributed Multimedia Systems, Proceedings of IDMS/PROMS'2002 Joint International Workshops on Interactive Distributed Multimedia Systems / Protocols for Multimedia Systems*, Coimbra, Portugal, November 26-29, 2002.

Mayer-Patel, K., Rowe, L., 1997. Design and performance of the Berkeley Continuous Media Toolkit. In *Multimedia Computing and Networking, Proc. SPIE 3020*, pp. 194-206.

McDougall, R., 2005. Extreme software scaling. In *ACM Queue*, vol. 3, no.7, pp. 36-46.

Microsoft Corp., DirectShow. http://msdn.microsoft.com.

Peleg, S., J. Herman, 1997. Panoramic mosaicing with VideoBrush. In *DARPA Image Understanding Workshop*, May 1997, pp.261-264.

Rasure, J., C. Williams, 1991. An integrated visual language and software development environment. In *Journal of Visual Languages and Computing*, vol. 2, pp. 217-246.

Robinson, D., P. Milanfar, 2003. Statistical performance and analysis of super-resolution image reconstruction. In *Proceedings of Intl. Conf. on Image Processing*.

Sawhney, H.S., S. Hsu, R. Kumar, 1998. Robust video mosaicing through topology inference and local to global alignment. In *Proc. of the 5th European Conference on Computer Vision*, vol. II, 1998, pp. 103-119.

Shum, H.-Y., R. Szeliski, 2000. Construction of panoramic mosaics with global and local alignment. In *International Journal of Computer Vision*, February 2000, vol. 36, no.2, pp. 101-130.

Sun Corp., Java Media Framework. http://java.sun.com/products/java-media/jmf.

Wilburn, B., N. Joshi, V. Vaish, M. Levoy, M. Horowitz, 2004. High speed video using a dense camera array. In *Proc. Computer Vision Pattern Recognition*.

Wilburn, B., M. Smulski, H-H. Kelin Lee, M. Horowitz, 2002. The light field video camera. In *Proc. Media Processors, SPIE Electronic Imaging*, vol. 4674, 29-36.

Yang, R., M. Pollefeys, S. Li, 2004. Improved real-time stereo on commodity graphics hardware. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04)*.