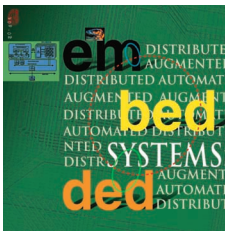
Smart Cameras as Embedded Systems



Smart cameras capture high-level descriptions of a scene and perform real-time analysis of what they see. These low-cost, low-power systems push the design space in many dimensions, making them a leading-edge application for embedded system research.

Wayne Wolf
Burak Ozer
Tiehan Lv
Princeton University

Increasingly powerful integrated circuits are making an entire range of new applications possible. Complementary metal-oxide semiconductor (CMOS) sensors, for example, have made the digital camera a commonplace consumer item. These light-sensitive chips, positioned where film would normally be, capture images as reusable digital files that users can upload to their computer, manipulate with software, and distribute electronically.

Recent technological advances are enabling a new generation of *smart cameras* that represent a quantum leap in sophistication. While today's digital cameras capture images, smart cameras capture high-level descriptions of the scene and analyze what they see. These devices could support a wide variety of applications including human and animal detection, surveillance, motion analysis, and facial identification.

Video processing has an insatiable demand for real-time performance. Fortunately, Moore's law provides an increasing pool of available computing power to apply to real-time analysis. Smart cameras leverage very large-scale integration (VLSI) to provide such analysis in a low-cost, low-power system with substantial memory. Moving well beyond pixel processing and compression, these systems run a wide range of algorithms to extract meaning from streaming video.

Because they push the design space in so many dimensions, smart cameras are a leading-edge application for embedded system research. The Embedded Systems Group in Princeton University's Department of Electrical Engineering (http://www.ee.princeton.edu/~wolf/embedded-group/) has developed a first-generation smart camera system that can detect people and analyze their movement in real time.

DETECTION AND RECOGNITION ALGORITHMS

Although there are many approaches to real-time video analysis, we chose to focus initially on human gesture recognition—identifying whether a subject is walking, standing, waving his arms, and so on. Because much work remains to be done on this problem, we sought to design an embedded system that can incorporate future algorithms as well as use those we created exclusively for this application.

As Figure 1 shows, our algorithms use both low-level and high-level processing. The low-level component identifies different body parts and categorizes their movement in simple terms. The high-level component, which is application-dependent, uses this information to recognize each body part's action and the person's overall activity based on scenario parameters.

Low-level processing

The system captures images from the video input, which can be either uncompressed or compressed (MPEG and motion JPEG), and applies four different algorithms to detect and identify human body parts.

Region extraction. The first algorithm transforms the pixels of an image, like that shown in Figure 2a, into an $M \times N$ bitmap and eliminates the background. It then detects the body part's skin area using a YUV color model with chrominance values downsampled by a factor of two. Next, as Figure 2b illustrates, the algorithm hierarchically segments the frame into skin-tone and non-skin-tone regions by extracting foreground regions adjacent to detected skin areas and combining these segments in a meaningful way.

Contour following. The next step in the process, shown in Figure 2c, involves linking the separate

0018-9162/02/$17.00 © 2002 IEEE

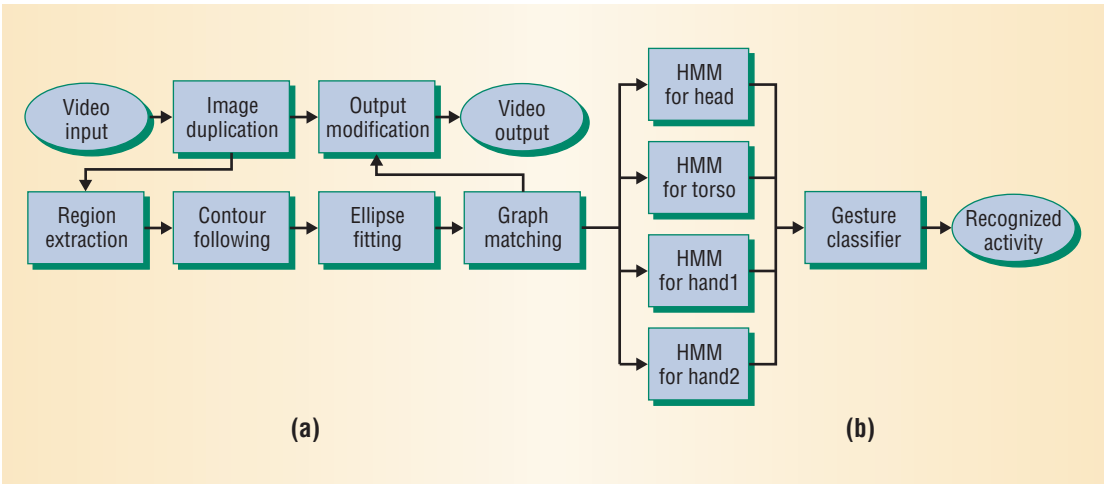


Figure 1. Human detection and activity recognition algorithms. (a) Low-level processing algorithms identify body parts and categorize their movements. (b) High-level processing algorithms use hidden Markov models (HMMs) and a gesture classifier to evaluate overall activity.

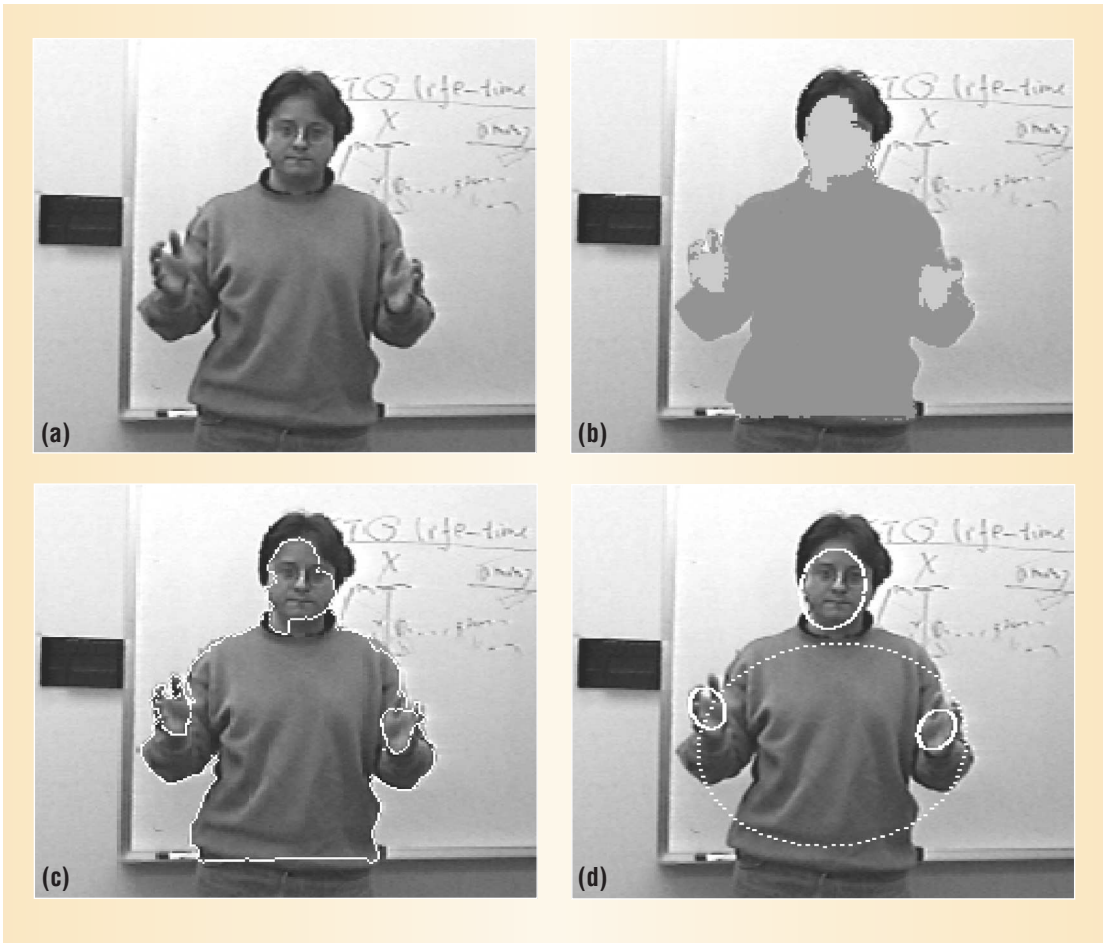


Figure 2. Initial steps in gesture recognition: (a) original image, (b) region extraction, (c) contour following, and (d) ellipse fitting.

groups of pixels into contours that geometrically define the regions. This algorithm uses a 3×3 filter to follow the edge of the component in any of eight different directions.

Ellipse fitting. To correct for deformations in image processing caused by clothing, objects in the frame, or some body parts blocking others, an algorithm fits ellipses to the pixel regions as Figure 2d shows to provide simplified part attributes. The algorithm uses these parametric surface approximations to compute geometric descriptors for segments such as area, compactness (circularity), weak perspective invariants, and spatial relationships.

Graph matching. Each extracted region modeled with ellipses corresponds to a node in a graphical representation of the human body. A piecewise quadratic Bayesian classifier uses the ellipses parameters to compute feature vectors consisting of binary and unary attributes. It then matches these attributes to feature vectors of body parts or meaningful combinations of parts that are computed offline. To expedite the branching process, the algorithm

Motion-Detection and Gesture-Recognition Systems

The research efforts focusing on human motion detection and gesture-recognition systems include Leonard,[1] a single-camera system that classifies simple motion events such as picking up an object using force dynamics. Mark Lucente, Gert-Jan Zwart, and Andrew D. George[2] have implemented a multimodal input system that also relies on one camera to let subjects manipulate virtual objects using gestures and voice commands. This system processes approximately 10 frames per second with a latency of 0.2 seconds.

The University of Maryland's Keck Laboratory for the Analysis of Visual Motion (http://www.umiacs.umd.edu/users/lsd/kecklab.html) employs a multicamera system to construct dynamic graphical representations of human movement and object manipulation. Digital cameras simultaneously capture some activity—such as a technician repairing a mechanism—from multiple viewpoints, and a suite of networked computers integrates this data with other sensor information into a 3D model for analysis using advanced computer graphics. Thomas B. Moeslund and Erik Granum discuss related work in their survey.[3]

Mircea Nicolescu and Gérard G. Medioni[4] have developed algorithms to electronically pan, tilt, and zoom through images supplied by an array of cameras. Their qualitative criteria for evaluating video input have confirmed that pan-tilt-zoom systems outperform wide-angle-lens cameras. Jonathan Foote and Don Kimber[5] have built a computationally and materially inexpensive panoramic camera system that also uses multiple cameras.

The MIT Media Lab (http://www.media.mit.edu/) is developing technology that can track people's actions, interpret gestures, and recognize facial expressions in environments ranging from the home and workplace to car interiors.[6-9] Smart rooms, smart desks, and wearable computers use context-sensing and communication devices to unobtrusively help people carry out everyday functions.

Scott Stillman and Irfan Essa[10] also have proposed a near-real-time system consisting of various types of sensors spread throughout an environment to track persons, detect faces, and recognize speech signals.

Other research efforts focus on various aspects of multiprocessor systems for video processing. Sek M. Chai and colleagues, for example, have developed an architecture for pixel-level processing in the imaging array.[11] In addition, John A. Watlington and V. Michael Bove are building a dynamically scheduled dataflow system using a distributed

begins with the face, which is generally easiest to detect.

High-level processing

The high-level processing component, which can be adapted to different applications, compares the motion pattern of each body part—described as a spatiotemporal sequence of feature vectors—in a set of frames to the patterns of known postures and gestures and then uses several hidden Markov models in parallel to evaluate the body's overall activity. We use discrete HMMs that can generate eight directional code words that check the up, down, left, right, and circular movement of each body part.

Human actions often involve a complex series of movements. We therefore combine each body part's motion pattern with the one immediately following it to generate a new pattern. Using dynamic programming, we calculate the probabilities for the original and combined patterns to identify what the person is doing. Gaps between gestures help indicate the beginning and end of discrete actions.

A quadratic Mahalanobis distance classifier combines HMM output with different weights to generate reference models for various gestures. For example, a pointing gesture could be recognized as a command to "go to the next slide" in a smart meeting room or "open the window" in a smart car, whereas a smart security camera might interpret the gesture as suspicious or threatening.

To help compensate for occlusion and other image-processing problems, we use two cameras set at a 90-degree angle to each other to capture the best view of the face and other key body parts. We can use high-level information acquired through one view to switch cameras to activate the recognition algorithms using the second camera. Certain actions, such as turning to face another direction or executing a predefined gesture, can also trigger the system to change views.

TOWARD AN EMBEDDED SYSTEM

As the "Motion-Detection and Gesture-Recognition Systems" sidebar describes, a number of researchers are working on human motion detection and gesture-recognition systems.

We initially used Matlab (http://www.mathworks.com/) to develop our algorithms. This technical computation and visualization programming environment runs orders of magnitude more slowly than embedded platform implementations, a speed difference that becomes critical when processing video in real time. We therefore ported our Matlab implementation to C code running on a very long instruction word (VLIW) video processor, which

resource manager for compact, relatively inexpensive media-processing applications.[12]

References

1. J.M. Siskind, "Visual Event Classification via Force Dynamics," *Proc. 17th Nat'l Conf. Artificial Intelligence* and *12th Conf. Innovative Applications of Artificial Intelligence* (AAAI/IAAI 00), AAAI Press/MIT Press, Menlo Park, Calif., 2000, pp. 149-155.
2. M. Lucente, G-J. Zwart, and A.D. George, "Visualization Space: A Testbed for Deviceless Multimodal User Interface," *Computer Graphics,* vol. 31, no. 2, 1997; http://www.lucente.biz/pubs/pubs.html.
3. T.B. Moeslund and E. Granum, "A Survey of Computer Vision-Based Human Motion Capture," *Computer Vision and Image Understanding,* vol. 81, no. 3, 2001, pp. 231-268.
4. M. Nicolescu and G.G. Medioni, "Electronic Pan-Tilt-Zoom: A Solution for Intelligent Room Systems," *Proc. IEEE Int'l Conf. Multimedia and Expo* (ICME 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 1581-1584.
5. J. Foote and D. Kimber, "FlyCam: Practical Panoramic Video and Automatic Camera Control," *Proc. IEEE Int'l Conf. Multimedia and Expo* (ICME 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 1419-1422.
6. A. Pentland and T. Choudhury, "Face Recognition for Smart Environments," *Computer*, Feb. 2000, pp. 50-55.
7. A.D. Wilson and A.F. Bobick, "Realtime Online Adaptive Gesture Recognition," *Proc. 15th Int'l Conf. Pattern Recognition* (ICPR 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 270-275.
8. A. Pentland, "Smart Rooms, Smart Clothes," *Proc. 14th Int'l Conf. Pattern Recognition* (ICPR 98), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 949-953.
9. A. Pentland, "Looking at People: Sensing for Ubiquitous and Wearable Computing," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, 2000, pp. 107-119.
10. S. Stillman and I. Essa, "Towards Reliable Multimodal Sensing in Aware Environments," *Proc. Workshop on Perceptive User Interfaces* (PUI 01), ACM Press, New York, 2001; http://www.cs.ucsb.edu/PUI/PUIWorkshop/.
11. S.M. Chai et al., "Focal Plane Processing Architectures for Real-Time Hyperspectral Image Processing," *J. Applied Optics: Special Issue on Information Processing,* vol. 39, no. 5, 2000, pp. 835-849.
12. J. Watlington and V.M. Bove Jr., "A System for Parallel Media Processing," *Parallel Computing,* vol. 23, no. 12, 1997, pp. 1793-1809.

let us make many architectural measurements on the application and make the necessary optimizations to architect a custom VLSI smart camera.

Requirements

At the development stage, we evaluated the algorithms according to accuracy and other familiar standards. However, an embedded system has additional real-time requirements:

- *Frame rate*. The system must process a certain amount of frames per second to properly analyze motion and provide useful results. The algorithms we use as well as the platform's computational power determine the achievable frame rate, which can be extremely high in some systems.
- *Latency.* The amount of time it takes to produce a result for a frame is also important because smart cameras will likely be used in closed-loop control systems, where high latency makes it difficult to initiate events in a timely fashion based on action in the video field.

Moving to an embedded platform also meant that we had to conserve memory. Looking ahead to highly integrated smart cameras, we wanted to incorporate as little memory in the system as possible to save on both chip area and power consumption. Gratuitous use of memory also often points to inefficient implementation.

Components

Our development strategy called for leveraging off-the-shelf components to process video from a standard source in real time, debug algorithms and programs, and connect multiple smart cameras in a networked system. We use the 100-MHz Philips TriMedia TM-1300 as our video processor. This 32-bit fixed- and floating-point VLIW processor features a dedicated image coprocessor, a variable length decoder, an optimizing C/C++ compiler, integrated peripherals for concurrent real-time input/output, and a rich set of application library functions including MPEG, motion JPEG, and 2D text and graphics.

Our testbed architecture, shown in Figure 3, uses two TriMedia boards attached to a host PC for programming support. Each PCI bus board is connected to a Hi8 camera that provides NTSC composite video. Several boards can be plugged into a single computer for simultaneous video operations. The shared memory interface offers higher performance than the networks likely to be used in VLSI cameras, but they let us functionally implement and debug multiple-camera systems with real video data.

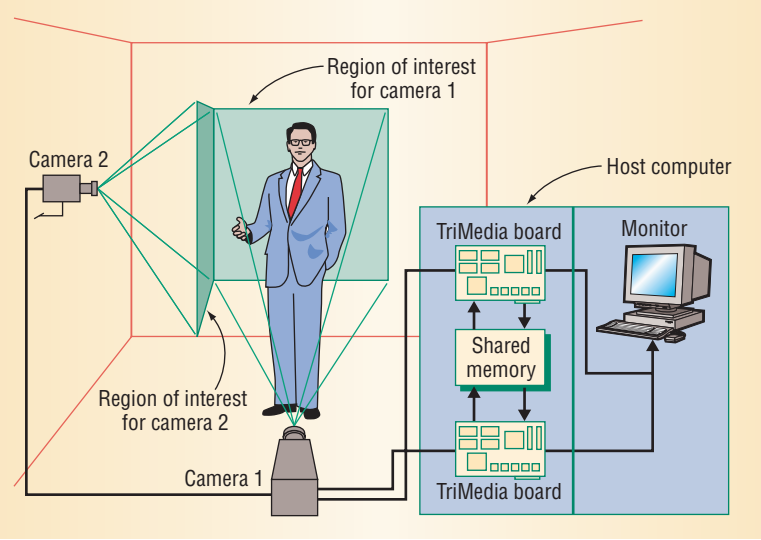


Figure 3. Smart camera test room and testbed architecture. Principal components include video processors on standard PCI bus cards, a shared memory interface, and a host PC for programming support.

EXPERIMENTS AND OPTIMIZATIONS

After converting the original Matlab implementation into C, we performed some experiments to gauge the smart camera system's effectiveness and evaluate bottlenecks. The unoptimized code took, on average, 20.4 million cycles to process one input frame, equal to a rate of 5 frames per second.

We first measured the CPU times of each low-level processing step to determine where the cycles were being spent. Microsoft Visual C++ is more suitable for this purpose than the TriMedia compiler because it can collect the running time of each function as well as its subfunctions' times.

Figure 4a shows the processing-time distribution of the four body-part-detection algorithms. Figure 4b shows the memory characteristics of each low-level processing stage.

As data representation becomes more abstract, input/output data volume decreases. The change in required memory size, however, is less predictable given the complex relationships that can form between abstract data. For example, using six single-precision, floating-point parameters to describe 100 ellipses requires only 2.4 Kbytes of memory, but it takes 10 Kbytes to store information about two adjoining ellipses.

Based on these early experiments, we optimized our smart camera implementation by applying techniques to speed up video operations such as substituting new algorithms better suited to real-time processing and using TriMedia library routines to replace C-level code.

Algorithmic changes

We originally fit superellipses (generalized ellipses) to contour points, and this was the most time-consuming step. Rather than trying to optimize the code, we decided to use a different algorithm. By replacing the original method developed from principal component analysis with moment-based initialization, we reduced the Levenberg-Marquardt fitting procedure, thus decreasing the execution time. Also, to accelerate processing during the graph-matching stage, we modified the algorithm to determine different regions' adjacency.

Library functions

During the region-extraction stage, the system processes each pixel in the input frame independently. Absolute value and threshold calculations result in branching, which limits instruction-level parallelism (ILP). One possible solution to this problem is to split the frame into several pieces and process these pieces on a multiprocessor or simultaneous multithreading platform. However, we opted to reduce the number of branches in the program.

The TriMedia processor provides an INONZERO operation that takes two input operands. If the first is not zero, the destination is set to the value of the second operand; otherwise, it is set to zero. Another special operation, IABS, can provide absolute values. These operations are visible in C code as they are packed into functions, and together they remove most of the branches.

We also used loop unrolling to extend basic block size. This optimization increased the processing speed of the region-extraction step by a factor of 2.3.

Control-to-data transformation

Increasing the processor's issue width can exploit the high degree of parallelism that region extraction offers. Using a processor with more functional units could thus reduce processing time during this stage. However, contour following, which converts pixels to abstract forms such as lines and ellipses, consumes even more time. The algorithm also operates serially: It finds a region's boundary by looking at a small window of pixels and sequentially moving around the contour; at each clockwise step it must evaluate where to locate the contour's next pixel. While this approach is correct and intuitive, it provides limited ILP.

We evaluated all possible directions in parallel and combined the true/false results into a byte, which served as an index to look up the boundary pixel in a table. We also manipulated the algorithm's control-flow structure to further increase ILP. These optimizations doubled the contour-following stage's running speed.

Optimization results

The combination of these methods radically improved CPU performance for the application. Optimization boosted the program's frame rate from 5 to 31 frames per second. In addition, latency

decreased from about 340 to 40-60 milliseconds per frame. We have since added HMMs and other high-level processing parts, and the program now runs at about 25 frames per second.

Our board-level system is a critical first step in the design of a highly integrated smart camera. Although the current system is directly useful for some applications, including security and medicine, a VLSI system will enable the development of high-volume, embedded computing products.

Because the digital processors and memory use advanced small-feature fabrication and the sensor requires relatively large pixels to efficiently collect light, it makes sense to design the system as two chips and house them in a multichip module. Separating the sensor and the processor also makes sense at the architectural level given the well-understood and simple interface between the sensor and the computation engine.

We believe that embedding single-instruction multiple-data (SIMD) processors into the sensor is not critical to achieve real-time performance. The advantages of leveraging existing sensor technology far outweigh any benefits of using pixel-plane processors until they become more plentiful. However, attaching special-purpose SIMD processors to the multiprocessor can be useful for boundary analysis and other operations. Such accelerators can also save power, which is important given the cost and effort required to deploy multiple cameras, especially in an outdoor setting. High-frame-rate cameras, which are useful for applications ranging from vibration analysis to machinery design, will likely require many specialized processing elements that are fast as well as area efficient, allowing the inclusion of more parallel units.

We are still in the early stages of determining the type of network best suited to a multicamera system. Distributed processing is an important means of reducing power consumption in such systems—sending raw pixels over the network is less efficient than sending the results of intermediate analysis. However, as algorithms for real-time multicamera analysis continue to develop, bandwidth requirements may change. ■

Acknowledgments

This work was funded by the New Jersey Center for Pervasive Information Technology; the MARCO/DARPA Center for Circuits, Systems, Software; and the National Science Foundation. We also thank Kees Vissers, Flaviu Turean, and Sebastian Mirolo at TriMedia, which generously provided us with processor boards, for their constructive suggestions.

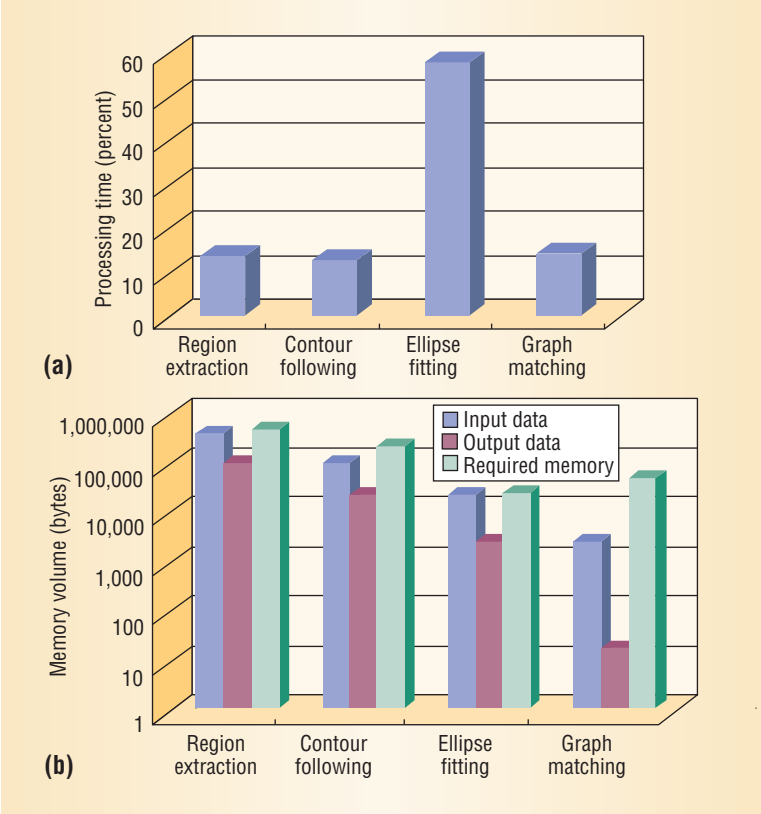


Figure 4. (a) Processing-time distribution of the original algorithms, implemented in C, before optimization. (b) Memory volume of low-level processing stages.

Wayne Wolf *is a professor in Princeton University's Department of Electrical Engineering, where he heads the Embedded Systems Group, and an associated faculty member in the Department of Computer Science. He received a PhD in electrical engineering from Stanford University. He is a Fellow of the IEEE and the ACM. Contact him at wolf@princeton.edu.*

Burak Ozer *is a research staff member and member of the Embedded Systems Group in the Department of Computer Engineering at Princeton University. He received a PhD in electrical engineering from the New Jersey Institute of Technology. He is a member of the IEEE. Contact him at iozer@ee.princeton.edu.*

Tiehan Lv *is a graduate student and member of the Embedded Systems Group in the Department of Electrical Engineering at Princeton University. He received an MAE in electrical engineering from Peking University, China. Contact him at lv@ee.princeton.edu.*