MVA'90 IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

MULTIRESOLUTION PYRAMID ARCHITECTURES For REAL-TIME MOTION ANALYSIS

Peter J. Burt

David Sarnoff Research Center
Subsidiary of SRI International
Princeton NJ, 08543-5300

ABSTRACT

Multiresolution pyramid techniques can improve the efficiency of basic vision algorithms by orders of magnitude. They will be key to developing practical vision systems to perform challenging tasks in real time. However, these efficiencies cannot be fully realized when pyramid-based algorithms are implemented on machines with conventional SIMD mesh and pipeline architectures. We describe a *segmented pipeline* architecture that can support pyramid processing. We illustrate this architecture with an application to image motion analysis for vehicle guidance.

INTRODUCTION

Pyramid techniques have become a standard element of algorithms for such computationally demanding vision tasks as stereo and motion analysis. They provide important mechanisms for achieving efficiency, and for isolating signals from noise and background clutter. When implemented in software, on general purpose computers, these techniques have improved speed dramatically, often by two, three, or more orders of magnitude.

However the same techniques are not readily transferred to the parallel computing machines, such as SIMD mesh and pipeline designs, that are under consideration for real-time vision applications. Difficulties arise in keeping processing elements busy when sample rates change within the pyramid, and analysis is restricted to regions of 'focal attention.'

We are developing a modified pipeline design, the *segmented pipeline*, to overcome the limitations of current designs [1]. In this paper we review the segmented pipeline concept, and show its application to image motion analysis for vehicle guidance.

PYRAMID-BASED ANALYSIS

Image pyramids are commonly used as a basis for efficient coarse-fine search. Less well know, but equally important, pyramid techniques have been developed in recent years for isolating signal components in complex imagery, and for implementing focal analysis techniques. All of these techniques can be implemented within a common framework to serve image motion analysis.

Coarse-fine search has been described for motion analysis by numerous researchers, e.g, [2],[3]. Motion analysis for a given pair of image frames begins at a low resolution pyramid level. There an initial rough estimate of image motion can be obtained at minimal computational cost because data arrays and frame-to-frame displacements are small. The estimate is then refined in steps as analysis moves to progressively higher resolution pyramid levels. The computation cost remains low because these refinements involve only low-complexity local computations.

The pyramid also provides a powerful means for separating multiple motion components without explicit image segmentation. For example, motion tuned 'channels' are obtained by applying motion analysis separately at each level of the Laplacian pyramid representations of an image pair [5]: fast motions are detected at low resolution levels, while slow motions are detected at high resolution levels. An ability to separate motions with very similar velocities can be obtained by taking advantage of this selection property in coarse-fine search [6]. In this way image regions containing multiple superimposed moving patterns, such as moving shadows or transparent objects, can be analyzed.

Integration pyramids provide a means for systematically controlling the size of the local regions in which individual motion estimates are obtained, as well as the image resolutions at which computations are performed. The best choice of region size depends on image content: small windows are required near motion boundaries, while large windows are best in areas of uniform motion [7].

Finally, the pyramid provides an ideal framework for implementing focal analysis strategies that are analogous to eye movements in humans: details of a scene are sensed only within a focal region of the scene, while the scene as a whole is sensed at much reduced resolution [8]. Focal analysis can be implemented as a sequence of *focal probes*, as suggested in Figure 1. Each probe examines a different region of the scene at a different resolution. In this driving scene, probes at low resolution are used first to identify the road near the vehicle, then probes at progressively higher resolution are used to follow the road into the distance. Other probes are used to locate and analyze oncoming vehicles and road signs. Image data processed in successive probes is obtained from selected analysis windows within a pyramid representation of the image, as shown on the right in Figure 1.

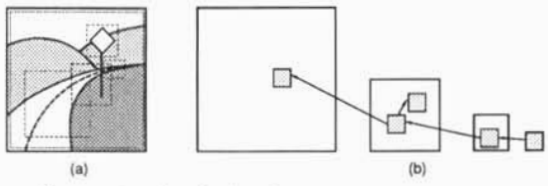

Figure 1. Analysis of a road scene as a sequence of focal probes (a) based on data from a multiresolution image pyramid (b).

In this paper we examine parallel implementations of an algorithm that incorporates all four of these pyramid techniques. Details of the algorithm are not required here, but can be found in [7]. Figure 2 illustrates an application of the algorithm to a vehicle guidance task. A vision system on board a moving vehicle is used to detect objects moving in or near a road, such as the postmen in this example. The system must discriminate between image motion that is due to the observer's own motion, and that which is due to object motion. At the particular point in time shown we assume the system has selected a region of the scene for focal analysis that contains the road as it recedes into the distance, Fig. 2*b*. Figure 2*c* shows the difference between two successive images of the sequence within this focal region. Image motion is dominated by camera rotation, as is evident in the significant misalignment in this difference image. Coarse-fine motion analysis is performed within the focal region to determine displacement due to camera motion. Successive images are then aligned based on this estimate, and a difference image is formed, Fig. 3*d*. The postman is now isolated from the background due to his relative motion.

CURRENT LIMITATIONS

Because pyramid techniques will play an important role in real-time applications, it is important that vision machines support pyramid-based computations effectively. In particular, the architectures for a real-time vision system must (a) accommodate a wide variation in sample densities, (b) accommodate a wide variation in the size and position of the focal analysis region, and (c) accommodate rapid changes in these characteristics in the course of highly dynamic analysis.

In many respects conventional SIMD mesh and pipeline architectures are well suited for motion analysis. The operations performed are local and homogeneous: computations at a given point are based only on sample values in a restricted neighborhood of that point, and the same operations are repeated at each sample point over extended regions of the image. However, when multiresolution focal techniques are used, both the mesh and pipeline designs. If a separate processing element is assigned to each image pixel, as in a 'fine grained' SIMD mesh, reductions in sample density or in the size of the focal analysis region only result in most processors being left idle. Since active processors are widely separated, communication between processors is slow. If a processing element is assigned to a block of image pixels, as in a 'coarse-grained' mesh, then there is less parallelism but more flexibility for the system to adapt to changes in sample density. But restrictions in the focal analysis region still leave many processors idle.

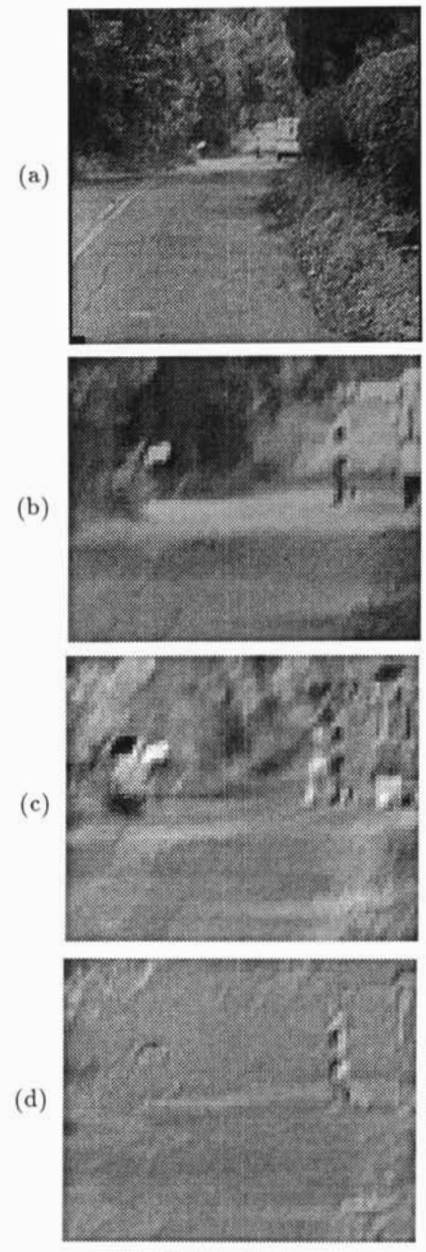

Figure 2. (a) Postman crossing road. (b) Region selected for motion analysis. (c) Difference between successive image frames showing motion due to the camera and postmen. (d) Difference after detecting and compensating for surface motion within the analysis region.

A pipeline machine has some advantages over a mesh. It can process subsampled, region-of-interest data if these data are read selectively from an input memory buffer: only samples that need to be processed enter the data stream, and the transmission rate is matched to the capacity of the processing elements.

Still, if further subsampling or windowing takes place in the course of analysis, then subsequent processing elements in the pipeline run below their capacity.

SEGMENTED PIPELINES

To resolve these difficulties we proposed a modified pipeline system. In this *segmented pipeline* the flow of data through the processing steps is interrupted periodically and data are returned to a memory buffer. Data transfer rates can then be readjusted to compensate for changes in sample densities and the size of the analysis region.

The computation within a segmented pipeline can be specified with an *image flow diagram*. This represents each processing step with an appropriate symbol and indicates the paths followed by data as arrows between symbols, Figure 3.

Each pipeline segment is a simple pipeline. A pipeline segment consists of an input buffer, a sequence of processing elements, and an output buffer, as shown in Figure 3*a*. Image data are organized into blocks, each representing an array of image samples. Image processing is performed as a data block is transferred from the input buffer, through the processing elements, into the output buffer. Samples are transferred sequentially, in a prescribed order (e.g., raster scan).

Resampling steps can be included within a pipeline segment, as shown in Figure 3*b*. While resampling by arbitrary factors is possible, a factor of two is most common and will be assumed here. In this case the data stream that is output from a down-sampling operation, $\downarrow$, consists of every other sample and every other row of the input data stream, while other samples are discarded. Up-sampling, $\uparrow$, means that a new sample (with value zero) is inserted between each pair of input samples and a row of new samples is inserted between each pair of input rows.

Down-sampling in two dimensions results in a reduction in data rate by a factor of $\frac{1}{4}$, while up-sampling results in an increase by 4. In the figure, data rates are indicated as numbers under the transfer arrows. Here the full data rate is shown as a '1', while reduced rates are fractions: $\frac{1}{4}$, $\frac{1}{16}$, etc.

Down-sampling and up-sampling operations are shown in these diagrams as distinct processing steps. In practice such operations are performed in conjunction with other processing steps and with buffer I/O. Down-sampling is generally performed after a processing step and before data is stored in a memory buffer, while up-sampling is performed before a processing step and after data are read from a buffer. Resampling incurs no cost or delay.

A windowing operation is performed at a given pipeline segment when a subarray of data are read from the input buffer and transferred through the processing elements. We show this as a shaded rectangle of data within the buffer symbol, Figure 3*c*.

EFFICIENCY ANALYSIS

A pipeline computation, and hence a pipeline segment, may consist of multiple data pathways running

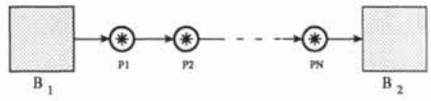

Figure 3a. A basic pipeline segment.

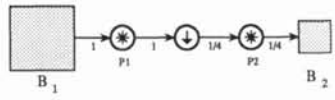

Figure 3b. A pipeline segment with resampling.

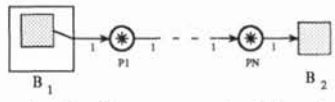

Figure 3c. A pipeline segment with windowing

in parallel. A simple case is shown in Figure 4*a*. Here a single input path diverges to form two paths, with identical copies of the source data flowing on both. The pathways merge at processing element $P4$. This could be any image operation that requires two inputs, such as sample-by-sample addition or multiplication.

Figure 4*b* shows an alternative implementation of the same computation as that in Figure 4*a*, but now organized as two pipeline segments. Data processed in the parallel pathways is stored in buffers $B2$ and $B3$ after the down-sampling step. These become the input buffers for the second segment. Note that processing elements $P2$ and $P4$ run at $\frac{1}{4}$ rate in the single segment implementation, but at full rate in the two segment implementation, thus providing a potential for more efficient use of these processing elements.

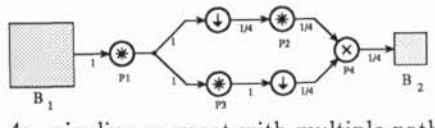

Figure 4a. pipeline segment with multiple paths.

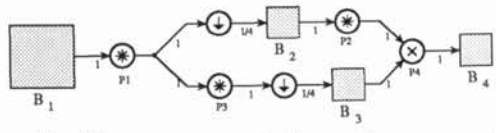

Figure 4b. The same computation as two segments.

The efficiency of these two implementations can be analyzed as shown in Figure 5. Time utilization of each processing element in the pipeline is shown over the course of the computation. Assume the initial data block has unit size, and that the nominal processing rate of all processing elements is one unit of data per unit of time. (There are added buffer delays within individual processing elements required to implement neighborhood operations, but these are insignificant for the present analysis.) The computation in the first configuration takes one unit of time, and all four processing elements are active during this entire period. However, processing elements P2 and P4 run at only 1/4 capacity because they follow the subsampling steps.

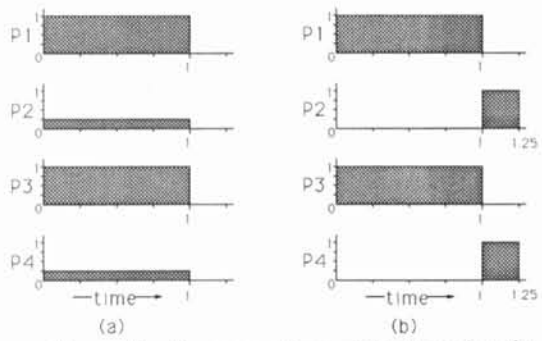

Figure 5. Processor time utilization for the pipelines in Figures 4a and 4b.

The diagram for the second configuration shows that two processing elements are busy for one time unit, then two other processors are busy for 1/4 time unit. In this case each processing element runs at full capacity when it is busy. The overall computation takes 25% longer than the computation in the single segment configuration. However, processing resources can be more efficiently used in this case, by assigning individual elements to other processing tasks when they are not required for the computation shown.

We define efficiency as the ratio of the processing performed by all the processing elements over the time course of the computation to the processing that could have been performed by these same elements if they ran at full capacity throughout the period they are assigned to the computations. By this definition the efficiency of the first configuration is .625 while that of the second is 1. The benefit of the segmented implementation are modest here, but can be very significant in other, more complex computations.

APPLICATIONS TO MOTION ANALYSIS

The motion analysis technique outlined, and illustrated in Figure 2, can be implemented within a segmented pipeline. The basic structure is shown in Figure 6.

Motion analysis is performed on two image frames, A and B. The algorithm estimates the motion, or displacement, from A to B in terms of parameters such as translation, rotation, and dilation. To achieve precise results, the basic computation shown is repeated several times for the given pair of images. With each iteration, analysis moves to a higher resolution pyramid level, and to a smaller analysis window. In each iteration an array of local correlation values is computed within the pipeline machine. Then parameters of motion are obtained by fitting a motion model to the correlation data. This computation is performed by an external microprocessor.

The computation begins with image A warped towards image B in accordance with a priori motion estimate V_0. This reduces the frame-to-frame displacement that must be estimated by the computation, and hence increases its accuracy. Next a Gaussian pyramid is constructed to level 2 for both the warped A and the original B. The image A_2 is then shifted relative to B_2 by -1, 0, and 1 samples in x, and the two images are multiplied, sample-by-sample, to form three product images M^-, M^0, M^+. (In practice A_2 is shifted by -1, 0 and 1 in both x and y, to form nine product images.) Next, an 'integration' pyramid is constructed for each product image to level 4. The resulting samples represent local cross-correlation values between the reduced resolution image arrays, A_2 and B_2, each defined within a gaussian weighted neighborhood. These correlation values are then accessed by the external microprocessor for motion estimation.

The motion estimate V_1 of Iteration 1 is used in the second iteration to specify the warp for image A. The size of the focal analysis window is reduced by 2 in both x and y with each new iteration, and analysis is moved to the next higher resolution pyramid level.

The size of each rectangle in this image flow diagram indicates the total data in the corresponding array. A rectangle with a solid outline signifies that the image array resides in a memory buffer, while a dashed outline signifies that the array is an intermediate result of a pipeline process that is never stored in a buffer, but that is shown for clarity.

The overall pipeline computation is divided into distinct segments, $S1$, $S2$, ..., $S8$, as indicated in the figure. Note that segments $S1$ and $S2$ are on different data paths than $S3$ and $S4$, so can be run in parallel. Similarly, $S6$, $S7$ and $S8$ can be run in parallel. Segment $S5$ is particularly interesting as it includes multiple data paths that diverge then merge. All processing in $S5$ must be run synchronously, in parallel.

PERFORMANCE ANALYSIS

A time/utilization diagram for the motion computation shown in Figure 7. Three iterations of the computation are shown. With each iteration it is assumed that the analysis window is reduced in size by a factor of four, and analysis is moved to a higher resolution pyramid level. This means the initial pyramid construction steps, P2 and P4, are skipped in Iteration 2, and P2, P3, P4 and P5 are skipped in Iteration 3. (P6, P7 and P8 are not included in the analysis because they only perform a shift by 1 sample, a function that, in practice, would be incorporated with another processing step, such as multiplication, P9, P10 and P11).

In this case the total time for the standard pipeline would be $3T$ while that for the segmented pipeline is $1.73T$. The efficiency of the standard pipeline for the three iterations would be .28, .098, and .048, respectively, or .143 overall. The efficiency of the segmented system is 1.

SUMMARY AND DISCUSSION

Pyramid techniques will be an essential part of practical vision systems that perform challenging tasks in real time. But to take advantage of these techniques a system must be able to accommodate large changes in data load from processing step to processing step, and must accommodate moment-by-moment changes in the computations performed.

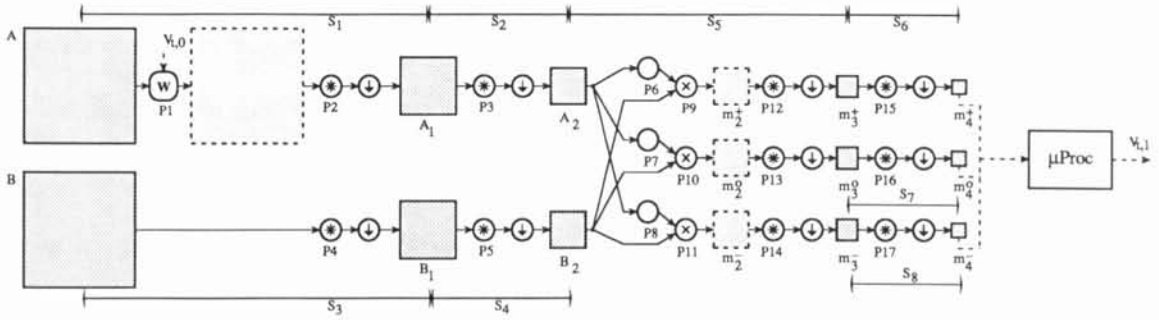

Figure 6. Motion computation diagram.

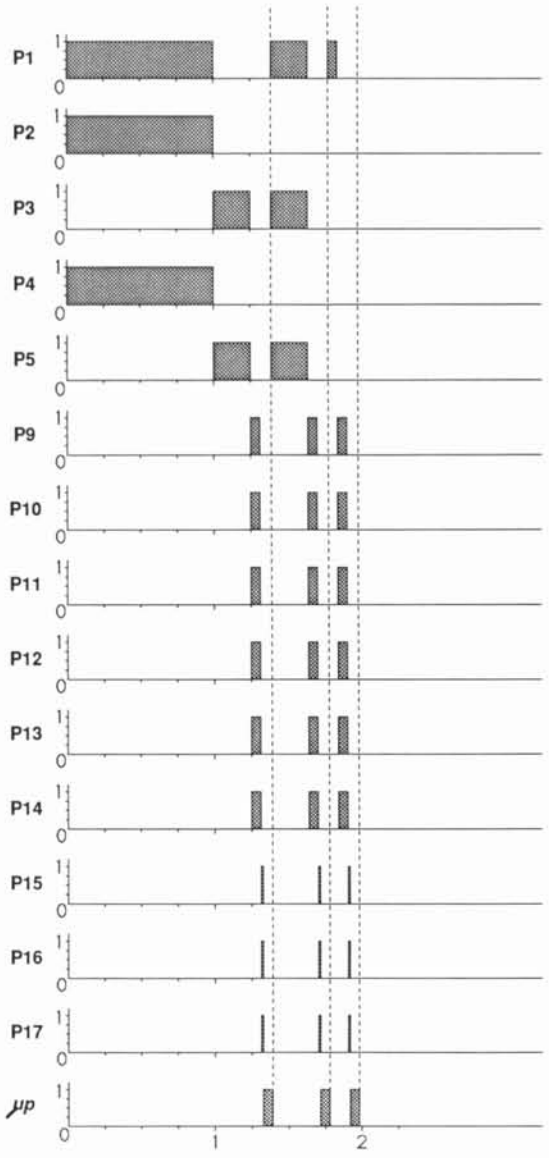

Figure 7. Time utilization for three iterations of the corase-fine, focal, motion analysis procedure.

We have described a segmented pipeline architecture that can maintain efficiency in the presence of such changes by decomposing a complex pipeline computation into a set of simple pipelines, each with roughly constant data load. Processing elements are shifted between pipeline segments dynamically, in the course of computations.

A machine to implement such segmented pipeline processing must be capable of rapidly reconfiguring its set of processing elements in order to assemble pipeline segments as data flow through the computation steps. In addition, it must have a supervisor process to assign resources to segments and coordinate the flow of data blocks through these segments as they are assembled. These aspects of the design are beyond the scope of the present paper.

REFERENCES

[1] P. J. Burt and G. van der Wal, "An architecture for multiresolution, focal, image analysis," *Proc. 10 Intr. Conf. on Pattern Recognition*, pp. 305-311, 1990.

[2] P. Anandan, " A unified perspective on computational techniques for the measurement of visual motion," *First Intr. Conf. on Computer Vision*, pp. 219-230, 1987.

[3] J. R. Bergen and E. H. Adelson, "Hierarchical computationally efficient motion estimation," *J. Opt. Soc. Am., Vol. 4*, pp. 35, 1987.

[4] P. J. Burt, J. R. Bergen, R. Hingorani, S. Peleg, and P. Anandan, "Dynamic analysis of image motion for vehicle guidance," *Proc. IEEE Intr. Conf. on Intell. Motion Control*, pp. IP-75-82, 1990.

[5] P. J. Burt, C. Yen, and X. Xu, "Multiresolution flow-through motion analysis," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, pp. 246-252, 1983.

[6] S. Peleg, J. Bergen, P. J. Burt, and R. Hingorani, "Computing two motions from three frames," Intr. Conf. on Computer Vision, 1990.

[7] P. J. Burt, J. R. Bergen, R. Hingorani, R. Kolczynski, W. A. Lee, A. Leung, J. Lubin, and H. Shvaytser, "Object tracking with a moving camera, an application of dynamic motion analysis," *IEEE Workshop on Visual Motion,* pp. 2-12, 1989.

[8] P. J. Burt, "Smart sensing within a pyramid vision machine," *Proceedings of the IEEE, Vol. 76*, pp. 1006-1015, 1988.