This appears in: Proceedings of IEEE International Conference on Image Processing, 2006

HIERARCHICAL DATA STRUCTURE FOR REAL-TIME BACKGROUND SUBTRACTION

Johnny Park, Amy Tabb and Avinash C. Kak*

School of Electrical and Computer Engineering, Purdue University
{jpark,atabb,kak}@purdue.edu

ABSTRACT

This paper seeks to increase the efficiency of background subtraction algorithms for motion detection. Our method uses a quadtree-base hierarchical framework that samples a small portion of the pixels in each image and yet produces motion detection results that are very similar compared to the conventional methods that raster scan entire images. The hierarchical data structure presented in this paper can be used with any background subtraction algorithm that employs background modeling and motion detection on a per-pixel basis. We have tested our method using two common background subtraction algorithms: Running Average and Mixture of Gaussian. Our experimental results show that the application of the hierarchical data structure significantly increases the processing speed for accurate motion detection. For example, the Mixture of Gaussian method with our hierarchical data structure is able to process 1600 by 1200 images at 11~12 frames per second compared to 2~3 frames per second without using the hierarchical data structure.

Index Terms— Image processing, Image segmentation, Image motion analysis, Object detection

1. INTRODUCTION

Much effort has been devoted in recent years to developing efficient methods of motion detection using background subtraction (see the review articles [3], [5]). However, images of size 640 by 480 or larger present problems in real-time applications. To get around this difficulty, we have developed a hierarchical framework using a quadtree that allows background subtraction to be carried out in real-time even on large images.

The hierarchical framework described in this paper may be applied to any background subtraction algorithm that employs background modeling and foreground detection on a per-pixel basis. We have used the hierarchical data structure in two common background subtraction algorithms: Running Average (RA) [10] and Mixture of Gaussian (MOG) [6]. RA assumes a unimodal Gaussian distribution for each pixel's background model. If a new pixel value belongs to the corresponding distribution of the background model, the pixel is classified as background and the mean of the distribution is updated. Otherwise, the pixel is classified as foreground. In MOG, each pixel's background is modeled as a mixture of Gaussian distributions (typically 3 to 5) and the models are updated using on-line approximation.

There are many adaptations and extensions to RA and MOG. Wang et al. [9] modified MOG by including a shadow removal process and by altering the process of background update and subtraction. Others refined MOG's pixel-level classification by employing region and frame information [2][8]. Lee et al. [4] use a Bayesian framework of background segmentation based on MOG. All of these approaches could, with little or minor modification, utilize the hierarchical data structure technique presented in this paper. We are not aware of any previous work that uses a hierarchical data structure to increase the efficiency of a background subtraction algorithm.

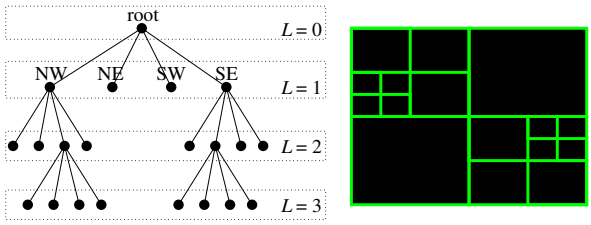

Fig. 1. Image represented by a quadtree

2. BACKGROUND SUBTRACTION USING HIERARCHICAL DATA STRUCTURE

In general, a background subtraction algorithm requires a set of images without any moving objects to create an initial background model. Then, for each image, all pixels are tested against the corresponding background model to detect foreground and to update the background model. The proposed method increases the efficiency of the foreground detection component of the algorithm using a hierarchical data structure.

The hierarchical data structure used in this work is based on a regularly decomposed region quadtree [7]. The root node of a quadtree corresponds to the entire image space. Four children of a node (labeled in order NW, NE, SW, SE) correspond to equal-sized quadrants of the region represented by that node. Let L denote the level in a quadtree (i.e., $L = 0$ for the root node, $L = 1$ for the children of the root node, and so on), and $N(L)$ be a set of nodes at level L. Figure 1 shows a quadtree with nodes upto level $L = 3$ and the corresponding block decomposition of the image.

After an initial background model is generated, the quadtree based decomposition is applied to every input image of the video sequence. Initially, a quadtree at the maximum level at $L = l_{init}$ is used as the base data structure of the input image where l_{init} is set by the user. Then, for each node in $N(l_{init})$, a random pixel is sampled where the sampling process includes the classification of a pixel as foreground or background and the update of the background model of that pixel location (how exactly a pixel is classified as foreground or background and how the background model is updated depend on the background subtraction algorithm chosen). If the pixel is classified as background, the next node in $N(l_{init})$ is considered. If the pixel is classified as foreground, the node is subdivided into the next level, then a randomly selected pixel in each of the four children nodes is

*Sponsored by the Appalachian Fruit Research Station, ARS/USDA

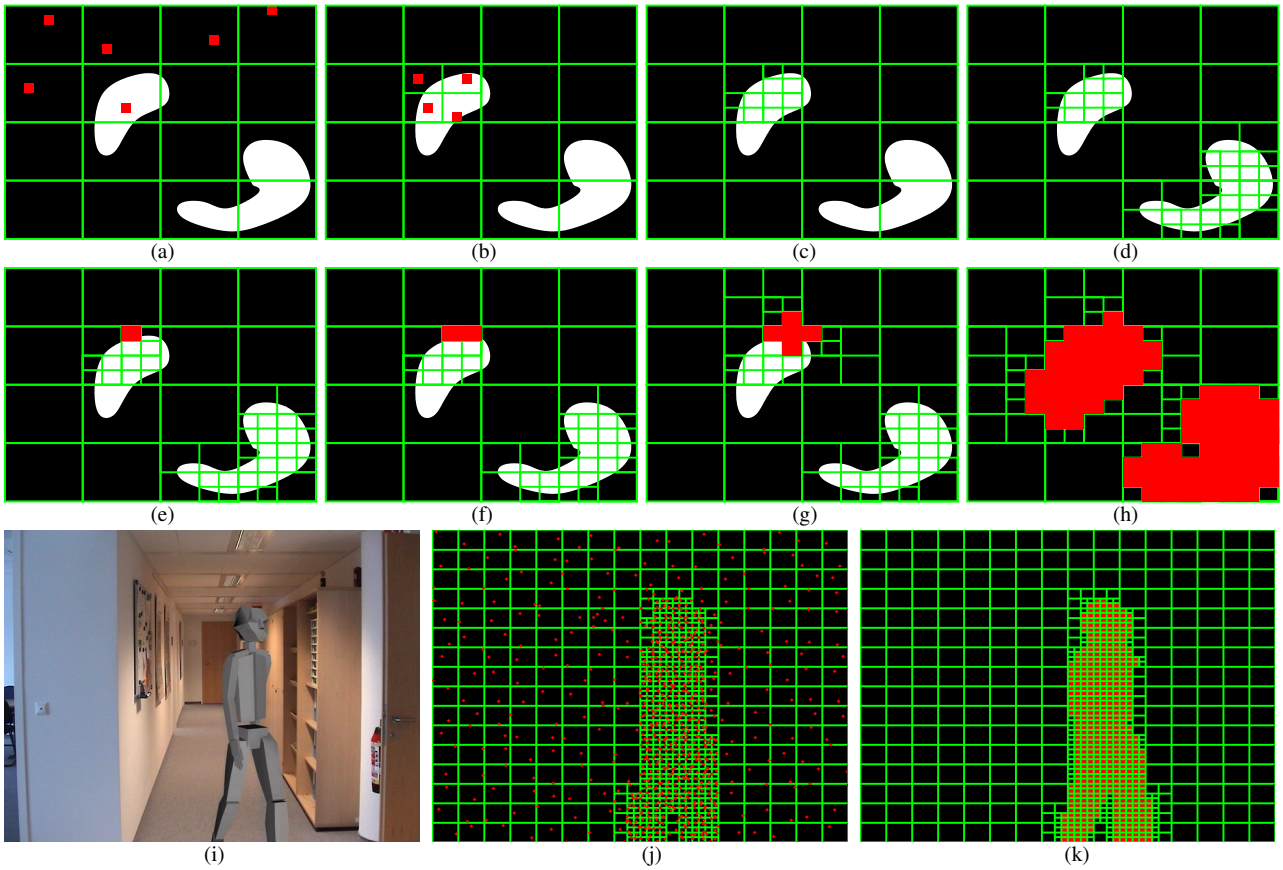

Fig. 2. Illustration of the background subtraction using hierarchical data structure

sampled. This subdivision process is repeated until $L = l_{final}$. Typically, for a 640 by 480 image, the values $l_{init} = 4$ and $l_{final} = 6$ work well. Figures 2(a) ~ (d) illustrate how a quadtree is constructed by the steps just mentioned. In (a), a random pixel in each of the nodes in $N(l_{init})$ is sampled. If a sampled pixel is classified as foreground, then the corresponding node is subdivided into the next level as shown in (b). A random pixel is sampled in each of the subsequent children. Each new node thus created is subdivided again if the sampled pixel is foreground until $L = l_{final}$, as shown in (c). The final tree structure for the example is shown in (d).

The quadtree so constructed provides the locations of the foreground objects in the image at a coarse level. In order to retrieve the fine boundaries of the foreground objects, the following procedure is carried out. For each node in $N(l_{final})$, all pixels contained by the node are sampled. The number of foreground pixels, n_f, and the number of background pixels, n_b, are counted. If $\frac{n_f}{n_f+n_b} < \tau$ where τ is some threshold typically ranging from 0.1 to 0.2, no other action is taken and the next node in $N(l_{final})$ is considered. On the other hand, if $\frac{n_f}{n_f+n_b} \geq \tau$, which implies that the node contains a significant amount of foreground pixels and that its neighborhood is likely to be part of the foreground object, its four-connected neighbor nodes are sampled. By recursively repeating this process, we obtain a connected region that encloses a foreground object. Since all the pixels within the connected region have been scanned, the precise boundary of the foreground object can be obtained. The process ends when all nodes in $N(l_{final})$ are scanned. Figures 2(e) ~ (h) illustrate the process of forming connected regions. In (e), the first node in $N(l_{final})$ is scanned. Since this node contains only a small number of foreground pixels (determined by the threshold τ), the next node is considered as shown in (f). Since this node contains a large number of foreground pixels, its four-connected neighbors are scanned. Note that the top and the right neighbors do not belong to $N(l_{final})$, which implies those two neighbors do not exist in the current tree structure. Therefore, the tree needs to be modified appropriately as shown in (g). The top and the right neighbors do not contain foreground pixels, thus the recursion terminates for those two nodes. The bottom neighbor, on the hand, belong to a foreground object (again determined by the threshold τ), so the recursion continues to its four-connected neighbors. The final tree structure and the nodes that were scanned are shown in (h). As a result of the quadtree, the algorithm samples only the pixels contained by the connected regions and the pixels randomly selected to construct the tree previously, which significantly reduces the inclusion of redundant data and improves real-time performance.

Figures 2(i) ~ (k) show the hierarchical data structure used in a real image. (i) shows an image with a human model as the foreground object, (j) shows the sampled pixels and the initial tree structure at the step equivalent to (d) in the previous example, and finally (k) shows all the scanned nodes and the final tree structure.

At medium to high frame rates, there will be some overlap of regions of foreground objects in consecutive images in a sequence. Therefore, after an image is processed, the resulting tree is used as the initial tree structure for the next image in a sequence. Conse-

	RA	RA w/ tree	MOG	MOG w/ tree
Seq 1	68.58	166.10	13.86	114.05
Seq 2	67.94	162.18	13.19	104.67
Seq 3	54.77	68.14	14.37	67.18
Seq 4	10.91	38.76	2.36	11.55

Table 1. Processing speed (frames per second)

quently, more random pixels get sampled around the regions of the foreground objects in the previous image. This increases the probability of generating a more accurate tree with less time spent on generating the tree structure.

3. EXPERIMENTAL RESULTS AND DISCUSSION

We tested our algorithm on the following four image sequences. The first sequence is an indoor hallway scene with a synthetic human model walking in the hallway. The second is an outdoor scene with multiple synthetic human models walking. These two image sequences were obtained from [11]. The third sequence is a busy two-way road scene with pedestrians and waving trees in the background. Finally, the fourth is an indoor laboratory scene with a few people moving in the room. The first three sequences have an image size of 640 by 480 captured at 30 frames per second. The fourth sequence has an image size of 1600 by 1200 captured at 7.5 frames per second. All the experimental results using the hierarchical data structure were obtained by setting $l_{init} = 4$ and $l_{final} = 6$ for the first three sequences, and $l_{init} = 5$ and $l_{final} = 7$ for the fourth sequence.

Table 1 shows the average processing speeds of four different background subtraction algorithms: the Running Average (RA), the Running Average using our hierarchical data structure (RA w/ tree), the Mixture of Gaussian (MOG), and the Mixture of Gaussian using our hierarchical data structure (MOG w/ tree). The use of our hierarchical data structure clearly increases the processing speeds of the algorithms especially for the Mixture of Gaussian, which requires computationally expensive foreground detection and background modeling.

Figure 3 shows the foreground detection results. The images in the first column show the input images, the second column shows the foreground detection results by RA, the third column the results by RA w/ tree, the fourth column MOG, and the fifth column MOG w/ tree. The foreground objects detected by the methods with and without the hierarchical data structure are very similar. In order to show a more precise comparison, Figure 4 shows the graphs that plot the number of foreground pixels detected using the Running Average method with and without the hierarchical data structure for each sequence. Observe that the number of foreground pixels detected is, in general, very similar between the methods with and without the hierarchical data structure. However, the methods with the hierarchical data structure detects slightly less foreground pixels consistently throughout the sequence. This discrepancy is due to the fact that some small isolated regions of foreground objects can be missed when no pixel in that region gets sampled during the initial tree construction process. In most cases, those missing regions are erroneous foreground objects. For example, see the foreground images of Seq 3 in Figure 3. Some erroneous foreground pixels on the left that were detected by RA are missing on the foreground image by RA w/ tree. In some cases, the missing foreground pixels correspond to the objects that are just beginning to enter the scene. For example, consider Seq 2 in Figure 3. The human on the far right who is just entering the scene is not detected by RA w/ tree. The object will continue to be missing until a random pixel gets sampled anywhere on that region, which then activates the subdivision of the tree and the region growing process that eventually detects the entire object. One can certainly increase the value of l_{init} (i.e., smaller block size for $N(l_{init})$) that would detect newly entered objects more quickly at the cost of decreased processing speed.

The entire sequence of images used for the experiment and all the foreground detection results are available at the following URL:

`http://rvl1.ecn.purdue.edu/RVL/Projects/HierarchicalBackgroundSubtraction/`

4. CONCLUSION AND FUTURE WORK

We presented a hierarchical data structure concept for background subtraction algorithms. Our experimental results show that our method significantly increases the processing speed of background subtraction. We also showed that the background subtraction results between the methods with and without the hierarchical data structure are very similar. Possible applications involve any situation where the computing resource is limited (such as a camera attached to an embedded system), the number of pixels to be processed is large, or multiple cameras need to be processed by one computer.

The method proposed in this paper can be improved by adjusting the values of l_{init} and l_{final} dynamically in a sequence. For example, suppose a system needs to process the background subtraction algorithm at a certain processing speed. When the current processing speed gets below the desired speed, the values of l_{init} and l_{final} can be decreased to speed up the process at the cost of less accurate foreground detection, and vice versa.

5. REFERENCES

[1] R.T. Collins, A. Lipton, H. Fujiyoshi, and T. Kanade, "Algorithms for Cooperative Multisensor Surveillance," Proc. IEEE, Vol. 89, No. 10, pp. 1456-1477, 2001.

[2] M. Harville, "A Framework for High-Level Feedback to Adaptive, Per-Pixel, Mixture-of-Gaussian Background Models," Lecture Notes in Computer Science, Proc. ECCV, Vol. 2352, 2002.

[3] W. Hu,, T. Tan, L. Wang, and S. Maybank, "A Survey on Visual Surveillance of Object Motion and Behaviors," IEEE Trans. SMC, vol. 34, no. 3, pp. 334-352, 2004.

[4] D-S Lee, J.J. Hull, and B. Erol, "A Bayesian Framework for Gaussian Mixture Background Modeling," IEEE Proc. ICIP. Vol. 3, pp.973-6, 2003.

[5] M. Piccardi, "Background Subtraction Techniques: A Review," IEEE Proc. SMC, Vol. 4, pp. 3099- 3104, 2004.

[6] C. Stauffer, and W. Grimson, "Learning Patterns of Activity Using Real-time Tracking," IEEE Trans. PAMI, vol. 22, no.8, , pp. 747-757, Aug 2000.

[7] H. Samet, "The Quadtree and Related Hierarchical Data Structures," Computing Surveys, Vol. 16, No. 2, pp. 187-260, 1984.

[8] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and Practice of Background Maintenance," IEEE Proc. ICCV, Vol. 1, pp. 255 - 261, 1999.

[9] H. Wang, and D. Suter, "A Re-evaluation of Mixture of Gaussian Background Modeling," IEEE Proc. ICASSP, Vol. 2, No. 2, pp. 1017 - 1020, 2005.

[10] C.R. Wren, A. Azarbayejani. T. Darrcll, and A. Pentland, "Pfinder: Real-time Tracking of the Human Body," IEEE Trans. PAMI, Vol. 19. no. 7. pp. 780-785, 1997.

[11] European Union MUSCLE Network of Excellence (FP6-507752), `http://muscle.prip.tuwien.ac.at/`

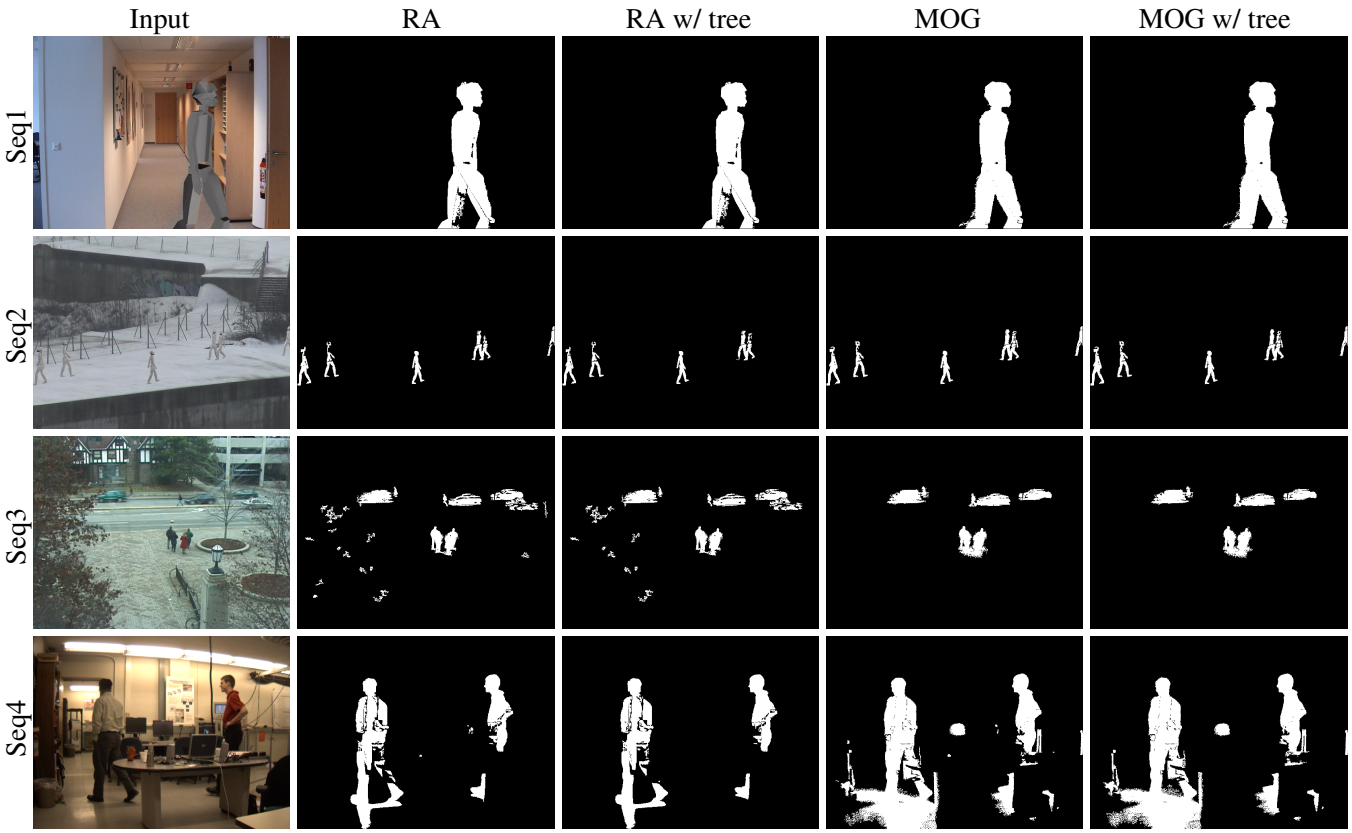

Fig. 3. Foreground detection results

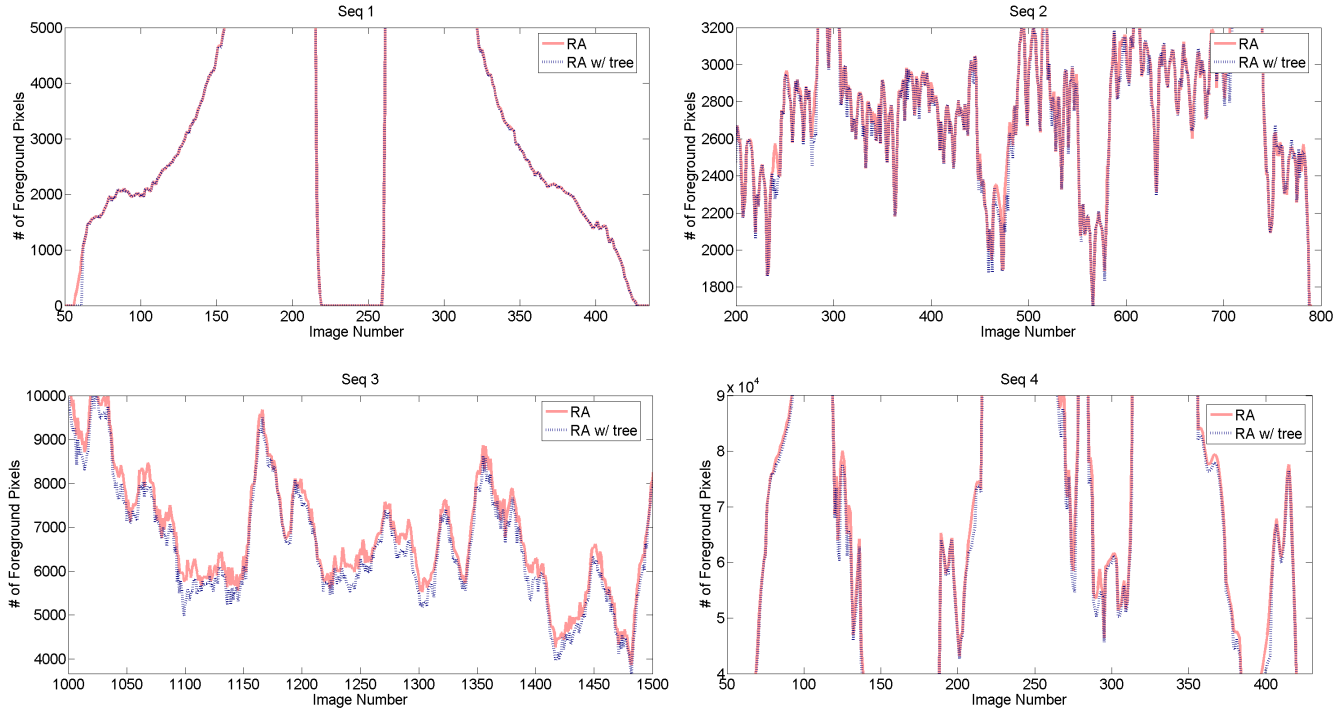

Fig. 4. Graphs showing the number of foreground pixels detected by RA and RA w/ tree