MULTILAYER BACKGROUND MODELING UNDER OCCLUSIONS FOR SPATIO-TEMPORAL SCENE ANALYSIS

A Dissertation
Presented to
The Academic Faculty

By

Shoaib Azmat

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2014

MULTILAYER BACKGROUND MODELING UNDER OCCLUSIONS FOR SPATIO-TEMPORAL SCENE ANALYSIS

Approved by:

Dr. Linda Wills, Advisor
School of ECE
Georgia Institute of Technology

Dr. Scott Wills, Co-Advisor
(Posthumous)
School of ECE
Georgia Institute of Technology

Dr. James Hamblen
School of ECE
Georgia Institute of Technology

Dr. Bo Hong
School of ECE
Georgia Institute of Technology

Dr. Aaron Lanterman
School of ECE
Georgia Institute of Technology

Dr. Jeffrey Vetter
College of Computing
Georgia Institute of Technology

Date Approved: June 10, 2014

In memory of Dr. Scott Wills

ACKNOWLEDGMENTS

First, I want to express my gratitude to my advisor Dr. Linda Wills for all her support, guidance, and patience during my doctoral research. Thank you Dr. Linda, without your support and encouragement in difficult times, I wouldn't have made it. I want to thank my late advisor Dr. Scott Wills for the time I spent with him. Dr. Scott, you left us so soon, and there were many things that I wanted to talk about with you. However, I am deeply grateful to you as they were your initial ideas that shaped my research. I thank you for giving me a great learning platform, which put me on track towards achieving my goal. You will be in my memories for ever as an exceptional mentor and friend.

I am grateful to Dr. James Hamblen, Dr. Bo Hong, Dr Aaron Lanterman, and Dr. Jeffrey Vetter for serving on my committee, and for providing valuable feedback and suggestions. I want to thank the Higher Education Commission of Pakistan and the Fulbright Program of USA for giving me scholarship, to pursue my graduate studies. I also want to thank my colleagues of the MOVES Lab at Georgia Tech, Dr. Dana Forsthoefel and Qianao Ju, for their support and company.

In the end, I want to thank my parents, R D Khan and M J Khan, and my siblings R Azmat, B Azmat, S Azmat, N Azmat, A Khan, H Khan, and R Khan. Their support and encouragement always acted as a catalyst for achieving my goals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BG	Background
CUDA	Compute Unified Device Architecture
EGMM	Extended Gaussian Mixture Model
FG	Foreground
FPS	Frames Per Second
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
MG	Midground
MMM	Multi Modal Mean
RGB	Red Green Blue
SIMT	Single Instruction Multiple Threads
SM	Streaming Multiprocessor
SP	Streaming Processor
*STM*3	Spatio Temporal Multi Modal Mean
TDP	Thermal Design Power
*TM*3	Temporal Multi Modal Mean

SUMMARY

This dissertation presents an efficient multi-layer background modeling approach to distinguish among *midground* objects, the objects whose existence occurs over varying time scales between the extremes of short-term ephemeral appearances (foreground) and long-term stationary persistences (background). The dissertation consists of three contributions.

In the first contribution, a multilayer object-based background modeling technique, called temporal multimodal mean TM3, is presented for video surveillance. The technique temporally models a scene in which there are multiple interacting midground objects occurring at different time scales. The approach correctly models scenes with long-term occlusions and ghost objects as compared to the multilayer pixel-based background modeling approaches. TM3 technique represents a scene, with multiple midground objects entering, leaving, and occluding each other at different points in time. This leads to richer information about temporal properties of a scene than traditional foreground/background segmentation. The information includes when a particular object arrived or left the scene, and the occlusion relationships among different objects while they are in the scene.

The multi-layer (and two-layer) background modeling techniques that model objects that have become stationary will incorrectly detect a new object if an existing midground or background object is displaced. The second contribution presents a novel spatio-temporal reasoning mechanism, called spatio-temporal multimodal mean STM3, based on multi-layer background modeling and objects appearances to conserve the state of moved objects in a scene. The algorithm is an extension of our temporal multimodal mean TM3 algorithm to spatial analysis. The STM3 algorithm, consistently models midground/background objects upon partial/full change of position, and maintains conservation of existing objects, only removing them once they leave the scene. An important feature of this algorithm is that it avoids false detections of new objects when existing objects are displaced in the scene.

Background modeling techniques for embedded computer vision applications must balance accuracy, speed, and power. Due to its inherent parallelism, robust adaptive background modeling, such as the Gaussian mixture model (GMM), has been implemented on graphical processing units (GPUs) with significant performance improvements over CPUs. However, these implementations are infeasible in embedded applications due to the high power ratings, in the range of 100 watts, of the targeted general-purpose NVIDIA GeForce GPU platforms. The third contribution focuses on how data and thread-level parallelism is exploited and memory access patterns are optimized to target a low-cost robust adaptive background modeling algorithm multimodal mean (MMM) to a low-power GPU NVIDIA ION with thermal design power (TDP) of only 12 watts. The algorithm has comparable accuracy with the GMM algorithm, but less computational cost. Accelerating this technique is also important because it is at the core of our spatio-temporal multi-layer background modeling algorithms TM3/STM3. We have achieved a frame rate of 392fps with a full VGA resolution (640x480) frame on the NVIDIA ION GPU. This is a 20X speed-up of the MMM algorithm on the GPU compared to the embedded CPU platform Intel Atom of comparable TDP. Moreover, our GPU implementation of MMM outperforms the GPU implementation of GMM by achieving a speed up of 6x. Subsequently, we extended the MMM GPU implementation to our multi-layer background modeling algorithm TM3, and achieved 5x speed up over the Atom CPU implementation.

CHAPTER 1

INTRODUCTION

The demand for video surveillance systems in public places and industry has increased dramatically. A recent survey shows that an estimated 1.85 million surveillance cameras have been deployed in the United Kingdom alone [3]. Many modern cities now have a network of surveillance cameras, deployed across metropolitan regions by multiple coordinated public/private agencies. These cameras are used in places such as streets, airports, subway stations, malls, and offices to detect abnormal activity. This enables many public safety applications including intruder detection, abandoned object detection, people counting, and traffic violation detection. Cameras are also extensively deployed in industry for process monitoring and product inspection, and in health facilities for improved patient care such as fall detection.

Requiring human operators to monitor video feeds is tedious, error prone, and simply infeasible. Advances in video technology has made automated video surveillance systems attractive in reducing the burden and tedium of manual monitoring. The desirability of portable and low-cost automated video surveillance systems, for example in outdoor settings, has led to the emergence of embedded smart surveillance cameras. These cameras have limited available power and computational resources, demanding efficient low-cost algorithms.

A core problem in automated visual surveillance is background modeling. This is the problem of separating salient, moving foreground from uninteresting, stationary background. Traditional background modeling divides a given scene into foreground and background regions. However, the real world can be much more complex than this simple classification, and object appearance events often occur over varying time scales. There are situations in which objects appear on the scene at different points in time and become stationary; these objects can get occluded by one another, and can change positions or be

removed from the scene. Inability to deal with such scenarios involving midground objects results in errors, such as ghost objects (when newly revealed background, due to removal of an object, is mistaken as a new midground object), miss-detection of overlapping objects, and aliasing caused by the objects that have left the scene but are not removed from the model. Modeling temporal layers of multiple objects can overcome these errors, and enables the surveillance of scenes containing multiple midground objects.

This dissertation is focused on modeling temporal layers of multiple objects and it specifically targets embedded surveillance systems, requiring a real-time, energy efficient and low-cost solution. One approach is to model these multiple midground objects using a tracking algorithm, but the computational cost is prohibitively high for applications in a resource-constrained embedded environment. This dissertation pursues the goal of efficiently modeling multiple midground objects using layers of low-cost background modeling, and discusses the challenges that arise in achieving this goal.

A few existing pixel-based approaches attempt to address this challenge by maintaining multiple layers [1], [4]. However, the problem with pixel-based modeling is that it is unable to deal with 1) long-term occlusions, and 2) ghost objects created by movement of objects in the original background. On a pixel level, one can delete object pixels not seen for a long time, but doing so will result in a new object in the scene if that object reappears. If an occluded pixel is not deleted, even if it has been occluded for a long period, then if the occluded object moves out of the scene, the pixel will remain in the model which will take extra space, and cause aliasing with overlapping objects. In addition, at the pixel level, it is difficult to reason about the order of occlusion among objects, and to suppress ghost objects created by movement of objects in the original background. Moreover, if an original background object is moved to a different location in a scene, then the existing multi-layer background modeling techniques will detect a new object at the new location in addition to a ghost object at the original location.

In this dissertation we present an object-based multilayer background model in which each new object is modeled as a midground layer. This object layer modeling handles long-term occlusions and ghost object removal, which enables the correct addition or removal of an object on its arrival or departure. In addition, objects are correctly detected and identified as existing objects when they change position. This conserves the number of existing objects in a scene, avoiding false detections of existing objects as new objects in a scene. In short, this dissertation provides a novel spatio-temporal reasoning mechanism using multilayer background modeling, based on the ages and occlusion relationships of the objects entering or leaving the scene.

In embedded applications, particularly in outdoor settings, wireless, portable, low-power but efficient systems are needed for automated video surveillance system, such as embedded smart cameras. These are in contrast to general-purpose computer-based systems in which the cameras send video over a dedicated link to the computer for image processing. Smart cameras, on the other hand, contain embedded architectures for performing the vision processing. However, they have limited available power and computational resources, demanding efficient low-cost algorithms and architectures. Therefore, for embedded smart cameras, we need an efficient background modeling technique that balances accuracy, speed, and power.

Basic background modeling techniques run fast, but their accuracy is not sufficient for computer vision problems involving dynamic background. In contrast, adaptive background modeling techniques are more robust, but run more slowly. Due to its high parallel computational characteristics, robust adaptive background modeling has been implemented on general-purpose graphics processing units (GPUs) with significant performance improvements over general-purpose CPUs [5], [6], [7]. These works have implemented many variants of the adaptive multimodal background techniques [8], [9], [10] on GPUs in order to accelerate their runtime performance. However, these implementations are infeasible in embedded applications due to the high power ratings of the targeted general-purpose

NVIDIA GeForce GPU platforms that consume around 100 watts. In practice, embedded applications must achieve real-time performance with the limited power of an embedded smart camera. The high-end embedded CPU-based smart camera's power consumption is in the lower tens of watts e.g. Intel Atom based NI177x from National Instruments [11], and Iris GT from Matrox [12], VIA-Eden-ULV based XCISX100C/XP from Sony [13], Analog Devices Blackfin based ILC-BL from Intellio [14]. The question then is how best to balance accuracy, speed, and power; so that we can achieve speed-up compared to a CPU by parallelizing an adaptive multimodal algorithm, and while doing so maintain the power of the system within limits of an embedded smart camera. This dissertation addresses these challenges for background modeling in two ways: 1) by using a robust algorithm that has low computational and memory costs, and 2) by exploiting the data and thread level parallelism in this algorithm, and optimizing its memory access patterns to target a low-power GPU platform NVIDIA ION.

1.1 Research statement and contributions

The goal of this dissertation is to develop an efficient object-based multi-layer background modeling approach to distinguish among objects, whose existence occurs over varying time scales between the extremes of short-term ephemeral appearances (foreground) and long-term stationary persistences (background). To achieve our goal we make the following three contributions.

The first contribution is a novel low-cost approach to multi-layer background modeling to *temporally* model a scene in which there are multiple interacting midground objects occurring at different time scales. To correctly model the objects, this contribution deals with long-term occlusions and ghost objects (Azmat et al.) [15], (Azmat et al.) [2].

The second contribution extends our temporal modeling algorithm to *spatial* analysis, to maintain correct modeling of existing objects invariant to their change of position. Our proposed algorithm maintains conservation of existing objects only removing them

once they leave the scene (Azmat et al.) [16].

The third contribution enhances the run-time performance of the adaptive background modeling and our multi-layer background modeling for resource constrained embedded platforms. First, we target the low-cost adaptive background modeling technique *multimodal mean* MMM to low-power parallel platforms. Accelerating this technique is important because not only is it at the core of our spatio-temporal multi-layer modeling algorithm, but it is a general adaptive background modeling technique for surveillance, particularly for outdoor scenes where dynamic background is present. Secondly, we target our multi-layer background modeling technique to low-power parallel platforms for further performance improvement (Azmat et al.) [17].

1.1.1 Contribution 1 - Multi-layer background modeling: Temporal scene analysis

The first contribution is to develop a highly efficient object-based multilayer background modeling technique for temporal scene analysis called *temporal multimodal mean* TM3. Rather than simply classifying everything in a scene as either dynamically moving foreground or long-lasting, stationary background, a temporal model is derived to place each scene object in time relative to each other. Foreground objects that become stationary (midground objects) are registered as layers on top of the background layer. In this process of layer formation, the algorithm deals with ghost objects, and noise created by dynamic background (such as fluttering leaves and rippling water) and moving foreground objects. Objects that leave the scene are removed based on the occlusion reasoning among layers. This technique allows us to represent a scene with multiple midground objects entering, leaving, and occluding each other at different points in time. This leads to a richer representation of temporal properties of scene objects than traditional foreground/background segmentation. In particular, the information includes when a particular object arrived or left, and the occlusion relationship among different objects while they are in the scene. The technique builds on the adaptive multimodal low-cost background modeling technique

multimodal mean MMM that makes it suitable for embedded, real-time platforms. Application scenarios of multiple midground objects occur frequently in the surveillance of homes, offices, and public places where many objects enter, exit, stop, or start moving at different points in time. Figure 1 shows the graphical summary of the first contribution. It shows our object-based multi-layer background modeling result in comparison with pixel-based approaches in a multiple midground objects scenario.

1.1.2 Contribution 2 - Multi-layer background modeling: Spatial scene analysis

Consider the following scenarios: a) the position of an object in the original background changes, b) a midground object is fully or partially displaced from its location, c) a midground object partially occluded initially, becomes unoccluded and is fully revealed. Multi-layer temporal background modeling cannot deal with these spatial displacement scenarios. It will incorrectly classify the objects in these cases as new objects.

Multiple object based tracking algorithms can deal with these spatial displacement scenarios, but the computational cost associated with using a tracking algorithm is too high especially for embedded platforms. For example, the cost for a popular appearance-based multiple-object tracking algorithm [18] is 15-20 frames per second (320x240 frame size) on a 3GHz P4 system. Our second contribution *spatio-temporal multimodal mean* STM3 extends our TM3 algorithm to spatial analysis. This contribution performs spatial reasoning based on multi-layer background modeling and color features of the moved object to conserve the number of objects in a scene, when only object positions have changed. The end result of this contribution is that we are able to correctly model the objects in the case of spatial displacements. This is important in detecting that an object in the original scene has only been moved to a different location and has not disappeared. Figure 2 shows the graphical summary of the second contribution. It shows how our algorithm deals with partial and full object displacements of midground/background objects.

1.1.3 Contribution 3 - Accelerating adaptive and multilayer background modeling on low-power GPUs

Background modeling techniques for embedded computer vision applications must balance accuracy, speed, and power. Basic background modeling techniques run fast, but their accuracy is not sufficient for computer vision problems involving dynamic background. In contrast, adaptive background modeling techniques are more robust, but run more slowly. Due to its high parallel computational characteristics, robust adaptive background modeling has been implemented on GPUs with significant performance improvements over CPUs. However, these implementations are infeasible in embedded applications due to the high power ratings of the targeted general-purpose GPU platforms. This contribution focuses on exploiting data and thread-level parallelism and optimizing memory access patterns to target a low-cost adaptive background modeling algorithm MMM to a low-power GPU with thermal design power TDP of only 12 watts. The algorithm has comparable accuracy with the GMM algorithm, but less computational and memory cost. We achieve a frame rate of 392fps with a full VGA resolution (640x480) frame on the low-power integrated GPU NVIDIA ION. This is a 20X speed-up of the MMM algorithm compared to the embedded CPU platform Intel Atom of comparable TDP. In addition our speed up compared to a GMM GPU implementation is up to 6x. Moreover, our GPU implementation of TM3 outperforms the CPU implementation by achieving a speed up of 5x. Figure 3 shows the graphical summary of the third contribution of accelerating adaptive and multilayer background modeling on low-power integrated GPUs. The figure shows the speed up achieved for both MMM and TM3 in different scenarios.

1.2 Summary of results

The results of this dissertation are summarized below:

- A novel multi-layer background modeling approach, temporal multimodal mean (TM3) is presented, to temporally model a scene in which there are multiple interacting

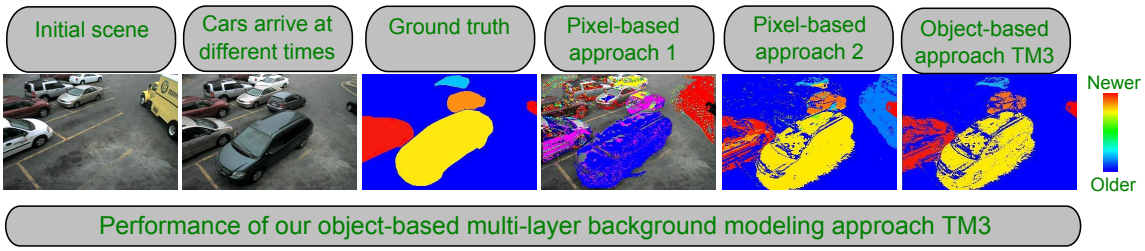


Figure 1: Graphical abstract of the first contribution: Temporal multimodal mean

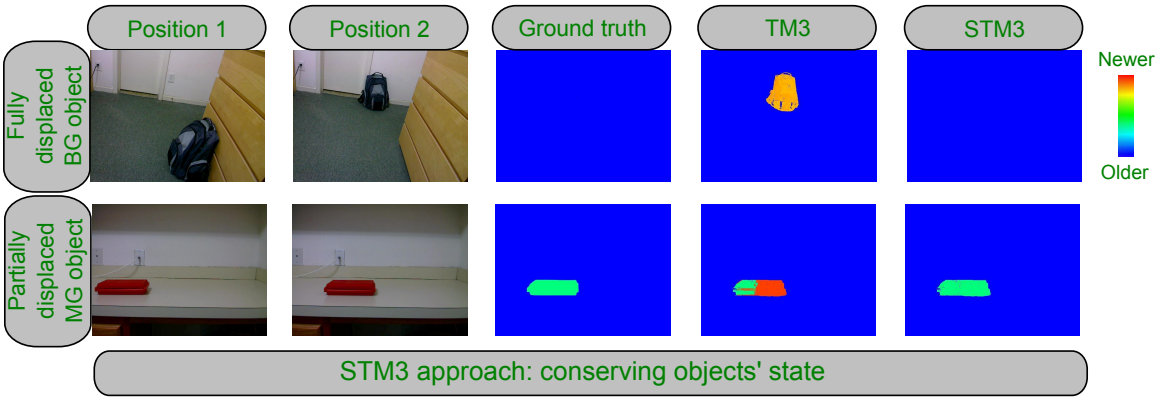


Figure 2: Graphical abstract of the second contribution: Spatio-temporal multimodal mean

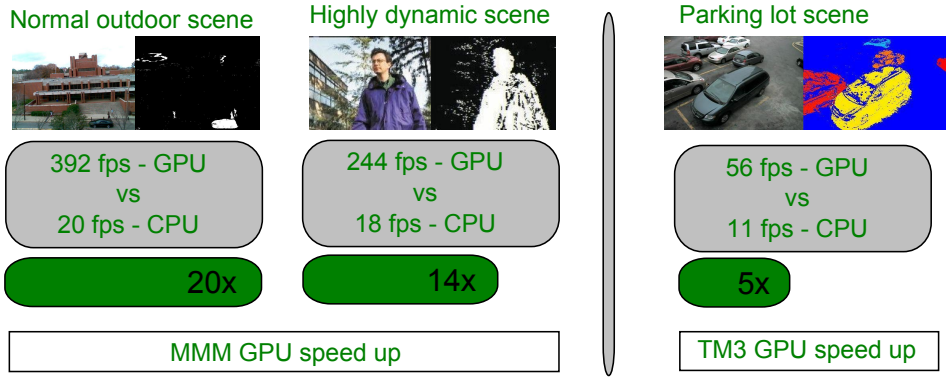


Figure 3: Graphical abstract of the third contribution: Accelerating adaptive and multilayer background modeling on low-power GPUs (640x480 frame size)

midground objects occurring at different time scales. The approach correctly models the scenes with long-term occlusions and ghost objects as compared to the pixel-based approaches.

- The TM3 technique represents a scene with multiple midground objects entering, leaving, and occluding each other at different points in time. The information includes when a particular object arrived or left the scene, and the occlusion relationships among different objects while they are in the scene.

- The spatial extension of TM3, called STM3 algorithm, consistently models objects upon change of position, and maintains conservation of existing objects, only removing them once they leave the scene. The change of position is not correctly modeled with previous multilayer background modeling approaches. An important new result of STM3 is that it avoids modeling a moved existing object as a brand new object in the scene.

- Parallelizing the low-cost multimodal mean MMM algorithm on the NVIDIA ION integrated GPU achieves a speed up as high as 20x in comparison with the Intel Atom CPU platform. The speed up of MMM compared to a GMM GPU implementation is 6x.

- Parallelizing our TM3 approach on NVIDIA ION integrated GPU achieving 5x speed up in comparison with the Intel Atom CPU platform.

1.3 Overview of content

This dissertation is organized as follows. Chapter two gives the detailed methodology of our first contribution of a novel multilayer background modeling approach temporal multimodal mean (TM3). The chapter shows the comparison of our object-based method with pixel-based multi-layer background modeling. It also analyzes the computational and memory requirement of our algorithm. Chapter three describes our second contribution,

which is an extension of our TM3 approach to spatial analysis: spatio-temporal multimodal mean (STM3). In this chapter we show the comparison of our new algorithm with TM3 in various object displacement scenarios. We also analyze the computational and memory overhead of the STM3 algorithm. Chapter four details our third contribution of accelerating background modeling on low-power integrated GPUs. In this chapter, we give details of our CUDA implementation and optimizations for multimodal mean MMM algorithm. We compare the performance of the multimodal mean MMM implementation with the Gaussian mixture model GMM implementation. We further show the parallelization of our TM3 approach on the integrated GPU. In the last chapter we conclude the dissertation and give guidelines for future work.

CHAPTER 2

MULTI-LAYER BACKGROUND MODELING: TEMPORAL SCENE ANALYSIS

2.1 Introduction

This chapter presents an efficient multi-layer background modeling approach, temporal multimodal mean (TM3), to distinguish among objects, whose existence occurs over varying time scales between the extremes of short-term ephemeral appearances (foreground) and long-term stationary persistences (background). We use the term *midground* [19] for this class of objects. Traditional background modeling techniques divide a scene into background or foreground. This mimics the human visual system which is adept at detecting rapidly moving objects. However, the human visual system is not well-suited for detecting objects which gradually become stationary over a period of time. These midground objects are the objects of interest, or salient objects, in applications such as abandoned luggage detection.

In traditional adaptive background modeling techniques midground objects are quickly assimilated into the background. Simple two-layer extensions of these techniques [19] coarsely model objects in the foreground to background transition, but do not differentiate among them based on their spatio-temporal properties. Object appearance events often occur over varying time scales, and there are situations in which many objects appear on the scene at different points in time and become stationary.

To analyze and visualize such scenarios, we need a mechanism to model multiple temporal layers of midground objects. This model would explicitly represent both the temporal order of object appearance events, and the spatial order of occluded objects. This would be helpful, for example, in analyzing a parking lot or a traffic intersection scene to determine the order in which vehicles have arrived. The mechanism should be able to deal with ghosts, created by objects leaving the scene from initial background, in this

multiple interacting objects scenario. In addition, it should remove the midground objects, when they leave the scene, which becomes more challenging in occlusion scenarios. These objects, if not removed, will remain in the background model and later cause aliasing with new objects. In short, a mechanism is needed to differentiate multiple objects, and correctly remove or add objects, even in occlusion situations. Application scenarios of multiple midground objects occur frequently in the surveillance of homes, offices, and public places where many objects enter, exit, stop, or start moving at different points in time.

We present an approach, called temporal multimodal mean (TM3), which treats background modeling in a continuous way by explicitly modeling temporal information in the scene. Rather than categorizing all scene elements as simply background or foreground, it differentiates them by the amount of time each is present in the scene. The technique, extends adaptive background modeling to model layers of midground objects based on object ages in the scene while temporally filtering out dynamic background noise. When an object leaves, occlusion reasoning is used to determine that the object's layer should be deleted. In addition, in this multi-layer representation, if an object leaves the original background and creates a ghost, it is recognized quickly and removed. In this way, our approach uses temporal reasoning based on object ages, and spatial reasoning based on occlusion relationships among the objects to correctly model, add and remove multiple midground objects entering or leaving the scene.

Our algorithm builds on the multimodal mean (MMM) background modeling technique [20]. The MMM technique is a low-cost adaptive background modeling algorithm for dynamic indoor and outdoor scenes. It executes four times faster than the widely used Gaussian mixture model (GMM) technique [8] on a general-purpose CPU platform, while exhibiting comparable performance in accuracy [20]. This makes the MMM approach amenable for use on embedded platforms, which have limited memory and execution power.

This chapter is organized as follows. Related work is discussed in Section 2.2. Section 2.3 presents our novel approach to multi-layer background modeling. Section 2.4 describes our detailed results with experiments based on multiple datasets and performance metrics. We conclude the chapter in Section 2.5.

2.2 Related work

This section starts by summarizing the main techniques of traditional background modeling, followed by two-layer background modeling, and finally outlines multiple object tracking and multi-layer background modeling in the context of modeling multiple midground objects.

2.2.1 Traditional background modeling

Background modeling to separate salient, moving foreground from uninteresting, stationary background is a key initial step in many video surveillance applications. Many techniques exist ranging from simple frame differencing to complex statistical modeling. The more complex techniques can handle difficult changing background scenarios, but it comes at the cost of more computational and memory resources. There are many ways in which we can categorize these techniques, one of which is their ability to handle dynamic, multimodal, backgrounds. Multimodal background means a background that can be different in different points in time at a particular location, like swaying trees and rippling water scenarios. We describe the main background modeling techniques based on this multimodal categorization. Throughout our discussion of the related work $I_{x,t}$ refers to an image pixel value x at time t.

2.2.1.1 Frame differencing

Frame differencing is the simplest background modeling technique, which detects the foreground by subtracting the pixel values in each new frame from the previous frame. If the

absolute difference is above a certain threshold E, then the pixel is declared as foreground:

$$|I_{x,t} - I_{x,t-1}| \leq E \ . \tag{1}$$

This technique is very fast, but its capability to handle complex scenarios is limited. The most notable disadvantage of this technique is a severe foreground aperture problem [21]. The technique also cannot deal with multimodal background.

2.2.1.2 Single parametric techniques

The single parametric technique presented in [22] is more robust than the simple frame differencing because it maintains a Gaussian for every background pixel. A pixel at a particular location is declared background if it falls within a certain threshold T times the standard deviation of the Gaussian. The mean and the variance are recursively updated. The technique is adaptive to noise due to gradual lighting changes, but as this technique maintains a single parameter for each background pixel, it cannot deal with multimodal backgrounds. Work related to this technique, with relatively low computational cost includes approximated median [23] and weighted mean [24], which maintain and update a median and a mean for each background pixel, respectively. There are non-recursive versions of single parameter background modeling techniques described in [25] with comparable accuracy.

2.2.1.3 Non-parametric

Elgammal et al. [26] present a non-parametric statistical technique for background modeling. This technique models the background based on a window of frames from the recent past. The technique estimates the density based on the values of each pixel in the recent past window, and the fate of a new pixel is decided based on the following kernel density estimation formula.

$$Pr(I_{x,t}) = \frac{1}{n} \sum_{k=1}^{N} K(I_{x,t} - I_{x,k}) \ , \tag{2}$$

where K refers to the kernel density, $I_{x,t}$ refers to the current pixel, and $I_{x,k}$ refers to the pixels in the recent past window. The technique is able to deal with multimodal dynamic

backgrounds, and its fast adaptation gives it sensitive detection. The disadvantage is that it is very time-consuming to estimate density based on a window of past frames [27].

2.2.1.4 Gaussian mixture model (GMM)

Stauffer and Grimson address the problem of modeling multimodal background by maintaining multiple Gaussians, instead of a single Gaussian, to model each background pixel [8]. The first B Gaussians in the mixture whose weight is above a certain threshold T are classified as background:

$$B = argmin_b \left(\sum_{k=1}^{b} w_{k,t} > T \right) . \tag{3}$$

The weight $w_{k,t}$ of the Gaussian k at time t is recursively updated as follow:

$$w_{k,t} \;=\; (1-\alpha) w_{k,t-1} \;+\; \alpha(M_{k,t}) \; . \tag{4}$$

$M_{k,t}$ is one when there is a match and zero otherwise, and α is the learning rate, which determines how quickly a recently appeared background pixel will be assimilated into the background. A match is declared when a pixel value is within 2.5 standard deviations of the Gaussian. A pixel that does not match, classified as foreground, is also added to the background model by replacing the Gaussian of the lowest weight. This is done for newly formed background portions that have recently started to appear.

The mean (μ_t) and the variance (σ_t^2) of the Gaussian k at time t is recursively updated as follows:

$$\mu_t \;=\; (1-\rho)\mu_{t-1} \;+\; \rho I_t \; , \tag{5}$$

$$\sigma_t^2 \;=\; (1-\rho)\sigma_{t-1}^2 \;+\; \rho(I_t - \mu_t)^T (I_t - \mu_t) \; , \tag{6}$$

where

$$\rho \;=\; \alpha \eta(I_t | \mu_k, \sigma_k) \; . \tag{7}$$

There are other high-end computationally-expensive machine learning techniques that can model more complex background scenarios. However, GMM is the most widely used

technique due to the balance it provides between accuracy and computational/memory cost [27].

2.2.1.5 Multimodal mean (MMM)

Even though GMM provides a good balance between accuracy and computational cost, the computational cost of maintaining multiple Gaussians is still too high for embedded platforms. Apewokin et al. [20] provided an alternative approach in the *multimodal mean (MMM)* technique. This technique segments the background and foreground of a scene by maintaining multiple means, instead of expensive Gaussians for each background pixel value. Each mean represents a background mode for the pixel.

More specifically, the MMM technique maintains a mean ($\mu_{x,t}$) representing a background mode for a pixel location x at time t, which is updated in the following way:

$$\mu_{x,t} = S_{x,t}/C_{x,t} \ , \tag{8}$$

where $S_{x,t}$ represents the running sum for the mode, and $C_{x,t}$ identifies the number of times that particular background mode has been seen. If the new image pixel value (I_t) does not fall within a certain threshold E, it is identified as foreground. Otherwise, the pixel intensity value is added to the sum $S_{x,t}$, and $C_{x,t}$ is incremented by one (i.e. the mean is updated). Count $C_{x,t}$ is checked to see if it is above a certain threshold T in which case the pixel is marked as background. The equation to declare a new image pixel value I_t as background is:

$$\bigwedge_i |\mu_{x,t-1.i} - I_{t.i}| \leq E \ \wedge C_{x,t-1} > T \ , \tag{9}$$

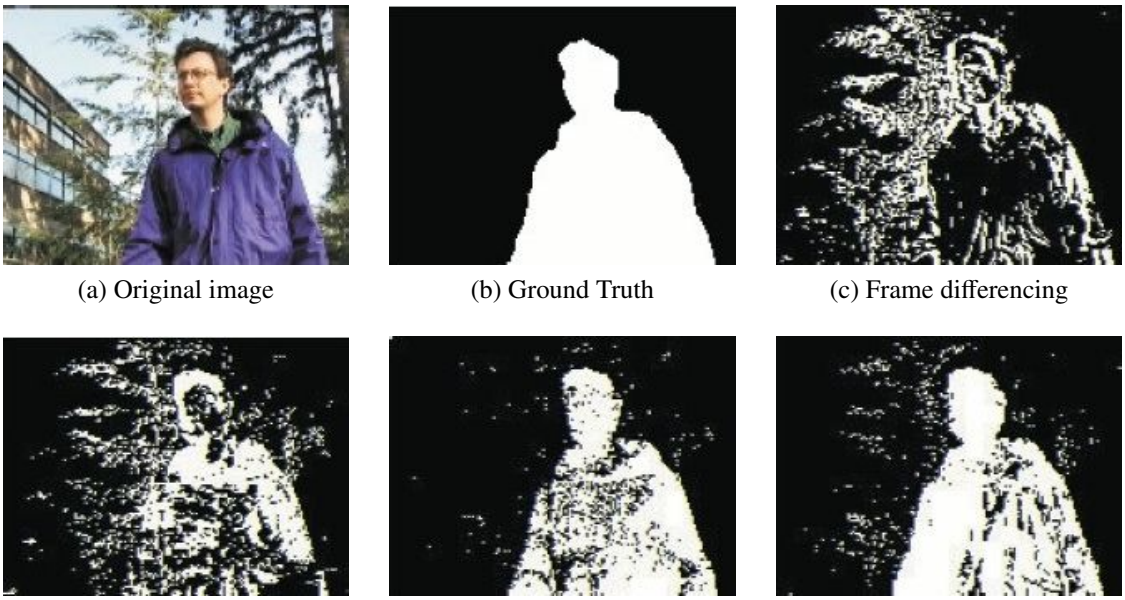
Like GMM, if a pixel is marked as foreground, its information is also added to the background model (using a least-recently-seen mode replacement policy), which allows for the case in which this pixel is part of the background that has recently appeared. Background modes for each pixel location are decimated after a certain time interval T_d, so that the newer pixels have more effect. This is done by halving the sum and the count values. Figure 4 shows the qualitative comparison of different background modeling techniques we

discussed. It can be clearly seen that the unimodal techniques of frame differencing and single parametric cannot effectively deal with multimodal dynamic background of swaying trees.

In [20], MMM is compared with the Gaussian mixture model (GMM) technique based on three datasets. Table 1 compares the accuracy of the MMM and GMM techniques. The percentage background error (BGE) is the ratio of false positives (FP) to true negatives (TN). A FP means a background pixel marked incorrectly as foreground, and a TN means a pixel correctly marked as background. The percentage foreground error (FGE) is the ratio of false negatives (FN) to true positives (TP). A FN means a foreground pixel marked incorrectly as background, and a TP means a pixel correctly marked as foreground. The overall percentage BGE (or FGE) error is the ratio of the sum of FPs (or FNs) to the sum of TNs (or TPs) of all the three datasets. The overall percentage BGE/FGE comparison, in the last column of the table, show that the accuracy of the MMM is very close to the GMM method. At the same time, the MMM technique requires fewer computational resources than GMM, and it executes four times faster on a general-purpose CPU platform. This performance improvement is primarily a result of maintaining inexpensive means, rather than Gaussians, requiring less computation and fewer memory accesses. Effectively, MMM is a low-cost approximation of the GMM technique.

Table 1: MMM Accuracy

Algo	%Error	Datasets			Overall
		Bootstrap	Outdoors	Trees	
MMM	BGE	<0.3	1	4	2
GMM	BGE	<0.3	1	1	1
MMM	FGE	77	51	13	36
GMM	FGE	77	62	22	42



(a) Original image (b) Ground Truth (c) Frame differencing

(d) Single parametric (e) GMM (f) MMM

Figure 4: Unimodal vs multimodal background modeling

2.2.2 Two-layer background modeling

Traditional background modeling techniques divide a scene into background and foreground. This mimics the human visual system which is efficient in detecting rapidly moving objects. However, the human visual system is not well-suited for detecting objects that gradually become stationary over a period of time. These objects, called midground objects [19], are the objects of interest in applications such as abandoned luggage detection. Traditional background modeling techniques maintain a strict division of foreground and background, which causes information to be lost in the event of midground object appearance because this event does not fall into the strict foreground/background division. The traditional background modeling techniques will assimilate these midground objects into the background as shown in Figure 5. How quickly these objects get assimilated depends upon the learning rate of the algorithm. The multiple modes, to handle dynamic background, in the GMM and the MMM technique are beneficial in that they can also be used to detect midground objects as described below.

Mathew et al. [28] use Stauffer and Grimson's GMM algorithm to detect midground objects that may appear in a scene. The algorithm has three states (three Gaussians of the background model), and they are called the foreground Gaussian, the background Gaussian, and the dominant background Gaussian. The detection of a midground object is based on the following events:

1. A change occurs in the dominant background Gaussian in a mixture;
2. A change occurs, from the foreground to the background to the dominant background Gaussian, in a short span of time;
3. The state remains as the dominant background Gaussian for a certain time threshold;
4. The new dominant Gaussian has weight above 0.5.

These conditions ensure that the algorithm only detects new midground objects, while filtering out dynamic background. Pixels of the dynamic background can also become the dominant Gaussian, but this may not occur in a short span of time, may not stay there for a while, and the weight of that Gaussian is most likely less than 0.5 with the total of three Gaussians to model the background.

Valentine et al. [19] detect midground objects using the MMM algorithm with low computational cost. They have two conditions for detecting the presence of a new midground object:

1. The "age" of the background pixel mode is above a certain threshold.
2. The observation frequency of the background pixel mode is above a certain threshold, in order to filter dynamic background.

Here age of a background pixel mode refers to the difference between the current time and the time when the pixel mode was first seen. Observation frequency is the number of times a background pixel mode is seen divided by its age.

The above mentioned techniques [28],[19] are pixel-level. This means they consider newly revealed background due to a removed object as a new midground object. For example, in Figure 6, a box, which was part of the original background is removed, and a new "ghost" object is incorrectly detected in its place by the pixel-level techniques.

As an alternative approach, region-level analysis can be used to differentiate between an abandoned object (real midground object), and a ghost object (when newly revealed background, due to removal of an object, is mistaken as a new midground object). One common approach focuses on the edge intensities of the blob created by the midground object [29] and [30]: a real midground object will have higher edge intensities around it, while a ghost object will have relatively lower edge intensities. The accuracy of these edge-intensity techniques is decreased in the presence of shadows around objects, which do not yield strong edges. The edge-based techniques are also not robust in cases of cluttered backgrounds because the basic edge intensity assumption is violated [31].

As an alternative, Ferrando et al. [32] compares the histograms of the midground object, the underlying background, and the surrounding bounding box. If the bounding box is more similar to the midground object than the background, it means an object has been removed, creating a ghost. Otherwise, it is indeed a new midground object. Another alternative is the technique given by Tian et al. [31], which is based on the region growing method. After identifying a new midground blob, it applies the region growing technique from inside of the blob towards the outer side. If the grown region is larger for the new midground blob than the underlying background, it means that an object has been removed. Otherwise, it is a new abandoned (midground) object. The region growing method is, however, computationally very expensive.

There are also techniques that detect midground objects using standard tracking algorithms instead of updating the background model. We will discuss them in the context of multi-layer background modeling, which is the topic of the next section.

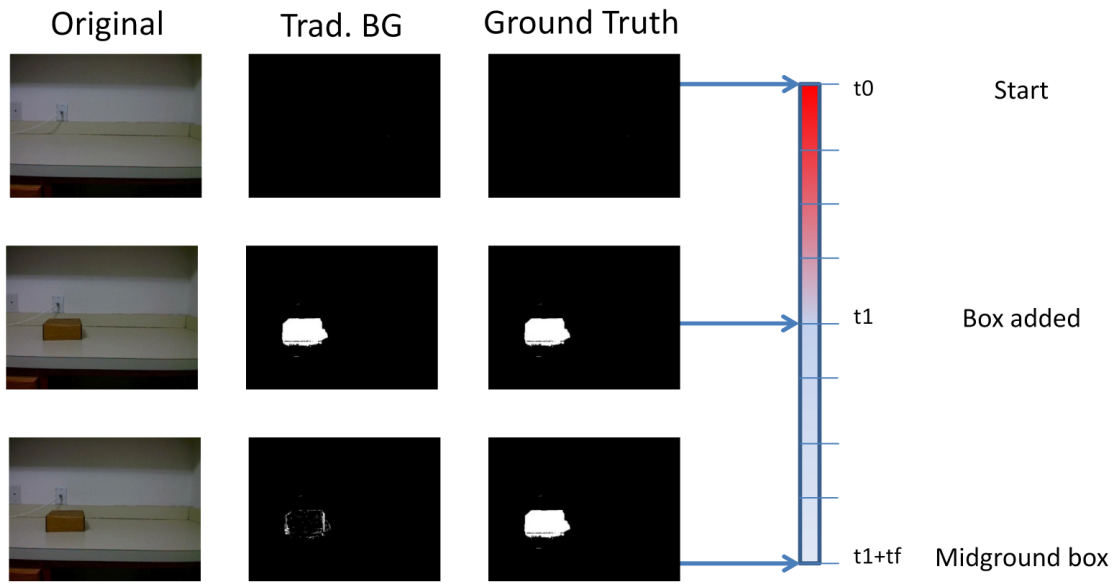


Figure 5: Traditional vs two-layer background modeling

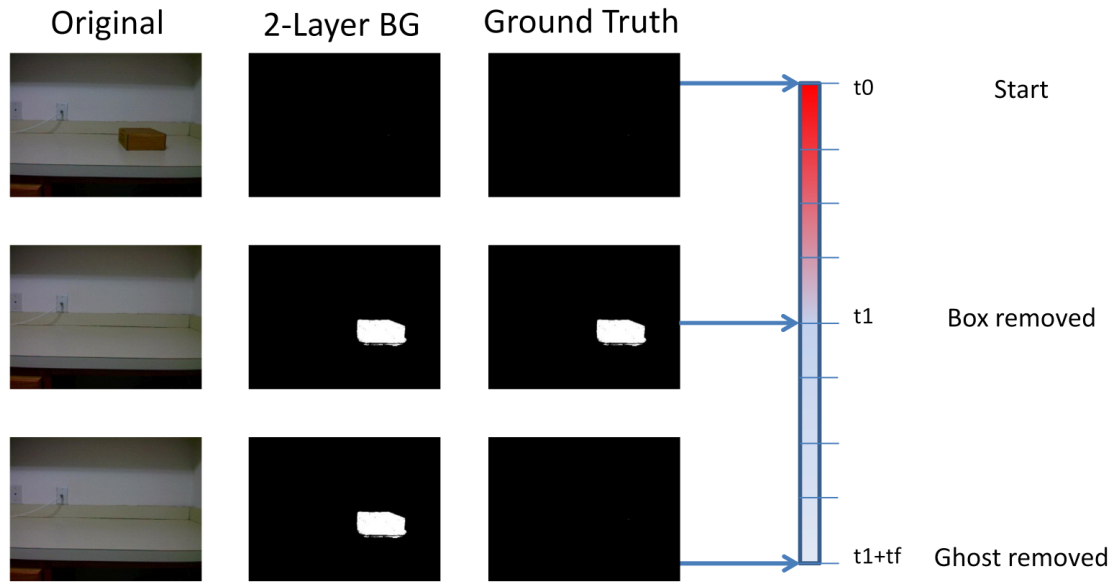


Figure 6: Two-layer background modeling pixel-level

2.2.3 Multi-layer background modeling

The real world can be much more complex than the simple scenario modeled by the two-layer background modeling. There are situations in which many objects appear on the scene at different points in time and become stationary; such objects can get partially or fully occluded by one another, and then after some time leave the scene. In other words, existence of the midground objects occur over varying time scales between the extremes of short-term ephemeral appearances (foreground) and long-term stationary persistences (background). To visualize such scenarios, we need a mechanism to model multiple temporal layers of midground objects. This object layer modeling can represent the scene based on the age of the midground objects in the scene, e.g. in a parking lot or traffic intersection scene it can help us understand the order in which vehicles have arrived. Figure 7 shows a parking lot scene, to demonstrate how two-layer adaptive background modeling with capability to detect midground objects in addition to moving foreground will deal with it, along with the expected output (ground truth) of the multilayer background modeling shown on a timeline of sequence of events. Figure 7(a) shows the initial scene and its background model. Figure 7(b) shows the same scene after the yellow truck, which was part of the background, left. Figure 7(c) shows the scene after two vehicles arrive, and people walk in front of the newly arrived van. The two-layer background model treats all objects same and one cannot tell their order of arrivals. The situation is made worse when some objects occlude others, such as people walking in front of the van. In this case, the newer objects (people) are not differentiable from the objects they occlude (van), and thus the newer objects may not be detected at all. Moreover, a ghost is created in Figure 7(c) where the yellow truck, which was part of background, left the scene. The techniques discussed above can deal with the ghost, but the ghost removal becomes more challenging in scenarios where multiple objects interact because the background becomes cluttered with objects. Figure 7(d) shows another vehicle arriving after some time, but everything in the two-layer background modeling is classified simply as either moving or stationary foreground or background. In addition,

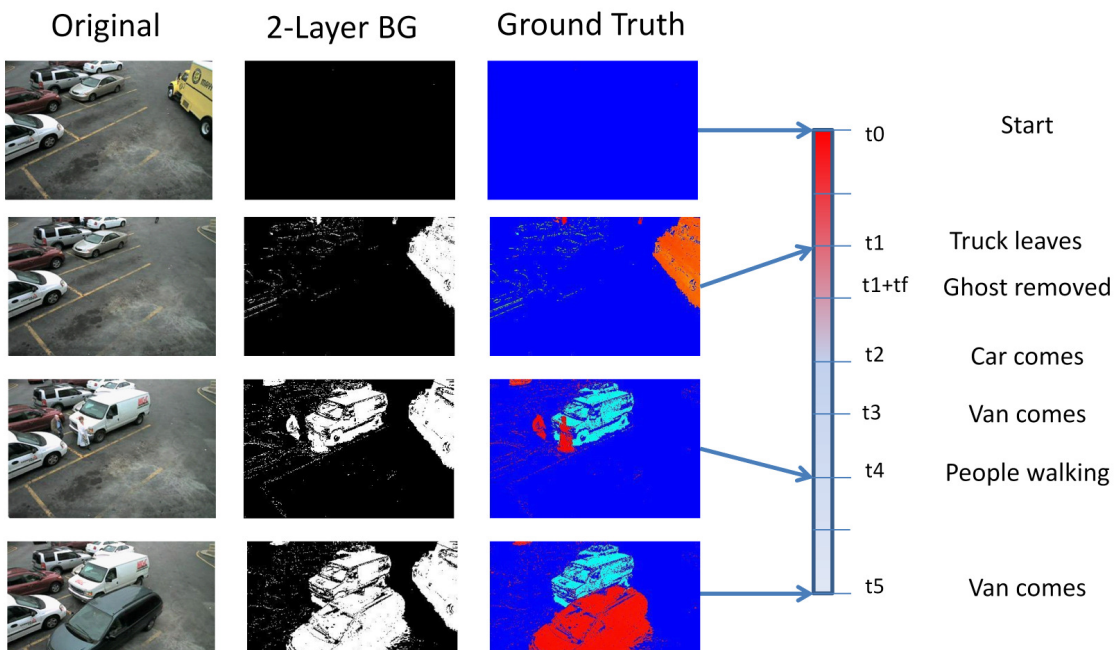


Figure 7: Need for multi-layer background modeling

in scenes with multiple objects, we need to remove the objects when they leave the scene which becomes more challenging in occlusion scenarios. These objects, if not removed, will remain in the background model and later cause aliasing with new objects. A mechanism is needed to differentiate multiple objects, and correctly remove or add objects even in occlusion scenarios.

2.2.3.1 Multiple layers using tracking

Tracking algorithms can offer an approach to modeling multiple object scenarios. Multiple midground objects can be modeled by techniques that track multiple objects under occlusions with non-uniform motion. A tracking algorithm based solely on motion estimation, such as Kalman filtering, can fail in the case of non-uniform motion. This means a tracking algorithm that uses a kernel of appearance and/or shape, and not only motion, is required. The techniques in this focused area of tracking are discussed.

Khan and Shah [33] model layers of foreground objects based on the Gaussian mixture model in addition to the background. A single multi-variate Gaussian in the mixture contains the color and the position information for each similar section, called class, of a

foreground object. The technique based on a MAP estimation framework tracks the objects from frame to frame. The algorithm keeps the parameters for the objects that are not visible due to occlusions, and uses these parameters to reassign when the objects reappear. The algorithm can lose track if the occluded object significantly changes its position or appearance during occlusion.

Yang et al. [18] model layers of foreground objects based on the color histogram on top of a simple background modeling technique. Their technique uses the merge and split approach, i.e., it stops updating the color histograms for the objects that merge, and instead starts tracking the merged blob. When the split occurs, it re-assigns the objects to their respective track. Unlike [33], this technique does not depend upon the position of the occluded objects, but it does require that their appearances not significantly change during occlusions.

Papadourakis and Argyros [34] also model both background and foreground color features using a mixture of Gaussians, similar to [33]. Their technique uses a straight through approach, which is to track and update individual object parameters even when the object undergoes occlusion, unlike the merge and split approach. Therefore, this technique does not have the re-assignment problem. To handle full occlusion, the technique uses object permanence natural phenomenon, i.e., the occluded object is expected to reappear in the vicinity of its occluder.

The above techniques are mainly based on the appearance of objects, and use deterministic motion scenarios. These techniques do not involve difficult motion scenarios such as missing frames. Tao et al. [35] model foreground objects as layers in a comprehensive manner in a MAP estimation framework using their appearance, shape and motion to handle difficult tracking scenarios.

2.2.3.2 Multiple layers using background modeling

The multi-layer object modeling using tracking, discussed above, can solve the scenario of multiple midground objects, but there are two problems. First these tracking algorithms

are computationally very expensive compared to the background modeling, and second these techniques have their own limitations in case of crowded environments [36]. We now discuss approaches that model multiple objects using background modeling.

Codebook by Kim et al. Kim et al. [1] model multiple layers of midground objects using a non-parametric multimodal background modeling technique which they call codebook [37]. The codebook background modeling technique, like GMM [8] and MMM [20], maintains multiple modes to handle dynamic backgrounds. They maintain a background model **M** that holds the original background and midground object layers, and they maintain a cache **H** that holds foreground objects whose fate is undecided yet. The steps involved in modeling multiple layers of background are as follows:

1. If a pixel not found in **M**, then search it in the cache **H**.

2. If a pixel also not found in **H**, then create a new codeword for this pixel of a foreground object and add it to **H**.

3. If a pixel does not re-appear in a time threshold T_H, then delete it from **H** as a pixel of a foreground object.

4. If a pixel remains in **H** for a time threshold T_{add}, mark it as a pixel of a midground object by moving it from **H** to **M**.

5. If a pixel is not seen in **M** for a long time threshold T_{delete}, then delete it from **M**, but only if it doesn't belong to the original background.

6. Show pixels using different colors based on their time of arrival which is recorded in the background codewords, to differentiate multiple objects.

The problem with this technique is that it also depicts the dynamic background as midground object layers.

Time scales by Jacobs and Pless. Jacobs and Pless [4] use causal filtering on pixel intensities to model time spent by objects in a scene. They apply multiple averaging filters with different time constants, where each filter shows in output the pixels that fall in a certain time scale based on the filter time constant. For example, a filter with small time constant will only show moving objects, while as the time constant is increased the midground objects will start showing up. The recursive equation for the low-pass filters $L_1, ..., L_N$ is:

$$L_i(x,t) \;=\; \alpha_i L_i(x,t-1) \;+\; (1-\alpha_i) I_{x,t}\;, \tag{10}$$

where $\alpha_i \in [0,1]$ specifies the filtering constants that determines how much of the original image ($I_{x,t}$) to be included for a particular filter $L_i(x,t)$, which in turn depends on the time scale of interest and the frame rate. The technique has good noise immunity. However, the limitation this technique has is its computationally expensive signal reconstruction, which is based on all the filtered output images $L_1, ..., L_N$, to show a final video summary with objects of different time scales at a given point in time.

Layered detection of multiple overlapping objects. Fujiyoshi and Kanade [38] multi-layer modeling detect multiple overlapping midground objects. They use a simple differencing background modeling technique to detect whether a pixel is transient or stationary. Subsequently, using region level analysis, a group of connected stationary pixels are registered as a new layer while subtracting the pixels of previously formed midground layers that spatially overlap with this new layer. In this way, they detect multiple overlapping midground objects.

The problem with pixel-based multi-layer background modeling is that it is unable to deal with long-term occlusions, and ghost objects created by movement in the original background. Both of these problems result in incorrect modeling of objects. Solving this is the objective of this chapter. The technique in [38] works at the region level, but its purpose is to detect overlapping midground objects in a unimodal background scenario, and it does not deal with removed objects from the original background. The other techniques [4] and

[1] that model and visualize multiple midground objects are pixel-based. On a pixel level, one can delete object pixels not seen for a long time as in [1], but doing so will result in a new object in the scene if that object reappears. If occluded pixels of an object are not deleted even if the object has been occluded for a long time, then if the occluded object moves out of the scene the pixels will remain in the model which will take extra space, and cause aliasing with overlapping objects. In addition, at the pixel level, it is difficult to reason about the order of occlusion among objects and to avoid ghost objects created by movement in the original background.

2.3 Temporal multimodal mean (TM3)

Our object-based multilayer background modeling technique, called temporal multimodal mean TM3, builds on the existing multimodal mean (MMM) background modeling technique [20]. In our implementation, we do not fix the number of modes to four which is the case in [20], but allow the number of modes to grow dynamically starting with one mode per pixel. Therefore, instead of mode replacement policy, we have mode removal policy. If the count of a mode reaches one as a result of several decimations, and that pixel mode is not seen for some time constant T_r, then this pixel mode is deleted from the background model. This step eliminates foreground pixels and background noise which is necessary to conserve resources and avoid aliasing.

The temporal extension of the MMM algorithm, TM3, must model midground objects that appear in a scene at different points in time, while dealing with ghost objects and long-term occlusions. The MMM multimodal nature because of dynamic background (like swaying trees and rippling water), allows seamless incorporation of multiple layers of midground objects since they too require multiple modes at all pixel locations. However, for temporal multimodal mean TM3 we define several extra fields in the data structure maintained for the modes of a pixel. Figure 8 shows the pixel data structure of a single mode in TM3. The observation count $OC_{x,t}$ is the number of times a background pixel is

seen. The birthday BD_x field records the time at which a pixel is first seen. The recency $RCN_{x,t}$ field records the time when a pixel was last seen. $LAYER_y$ field is the pointer to the midground object layer to which the pixel belongs.

$S_{x,t.r}$	$S_{x,t.g}$	$S_{x,t.b}$	$C_{x,t}$	$OC_{x,t}$	BD_x	$RCN_{x,t}$	$LAYER_y$

Figure 8: Pixel data structure for a single mode in TM3

To eliminate foreground pixels quickly, rather than wait for the relatively slow decimation process to eliminate them, we deal with foreground pixel deletion separately as done in [1]. These pixels use memory resources and cause aliasing if not removed. Two things are true about foreground pixels: 1) they have a low observation time compared to background or new midground objects, and 2) after initial observation they are not seen at all. If a pixel is seen for an observation interval below the threshold T_f, and after that it has not been seen for a period longer than the threshold T_{nf} then it will be deleted.

To begin, our TM3 algorithm calculates the temporal attributes of age and observability (observation frequency) for pixel x at time t as follows:

$$Age_{x,t} = t - BD_x \ , \tag{11}$$

$$OB_{x,t} = OC_{x,t}/Age_{x,t} \ . \tag{12}$$

Pixels whose observability is less than a certain percentage δ are filtered out as dynamic background or noise. The rest of the pixels, depending upon their age, are depicted using colors ranging from red (newest-foreground) to blue (background) based on the standard visible color spectrum. This filtering step is important because, without it the algorithm will render any dynamic background pixels as new midground pixels, which can make the scene noisy and can lead to the formation of false object layers. The observability threshold δ is kept at a certain optimum level so that the algorithm does not filter real objects, but filters dynamic background noise maximally.

The pixels which have observability greater than δ, and are observed for the time-period T_f, indicate a new midground object. The pixels which satisfy this condition are formed into a blob. When a blob is observed to stop growing, the blob is registered as a new object layer. Moreover, each pixel in the blob is assigned that particular layer number. Previously formed layers are subtracted from the formation of new layer based on their pixels timestamps. The layer data structure is given below in Figure 9:

ID	TS	C	X_{min}	X_{max}	Y_{min}	Y_{max}	V_t	O_t	U_t	D_t

Figure 9: Data structure for an object layer

ID is the layer identification, TS is the timestamp at the layer formation, C is the total number of pixels in the layer, and X_{min}, X_{max}, Y_{min}, and Y_{max} specify the area covered by the layer. The rest of the fields is used for occlusion reasoning among different object layers.

Pixels with layer numbers belong to real midground objects, and so they are not filtered based on observability threshold δ because their observability will ultimately fall below δ if they get occluded long-term. In addition, these pixels are not decimated in an interval in which they are occluded. This preserves layers of overlapping midground objects. Moreover, after the formation of new layers, midground pixels that are not classified as part of any layers are also filtered out as background noise. This filtering along with filtering based on observability deals with the noise caused by dynamic background, which allows clearer visualization of objects as shown in the results.

To better visualize the scene, objects that become older than a certain age are pushed to the background (marked blue). This background migration of older objects clears the scene space and focuses on newer objects. It also increases the dynamic color range available to newer objects making them more differentiable. Figure 10 shows the complete flowchart of our algorithm.

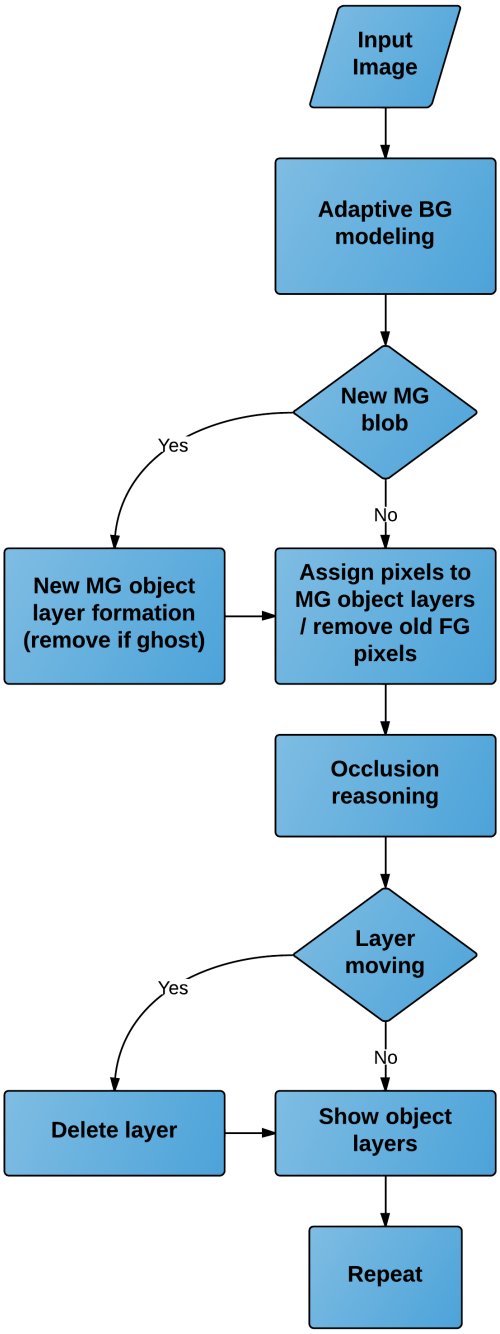


Figure 10: Flowchart of the TM3 algorithm

2.3.0.3 Ghost removal in multiple overlapping layers

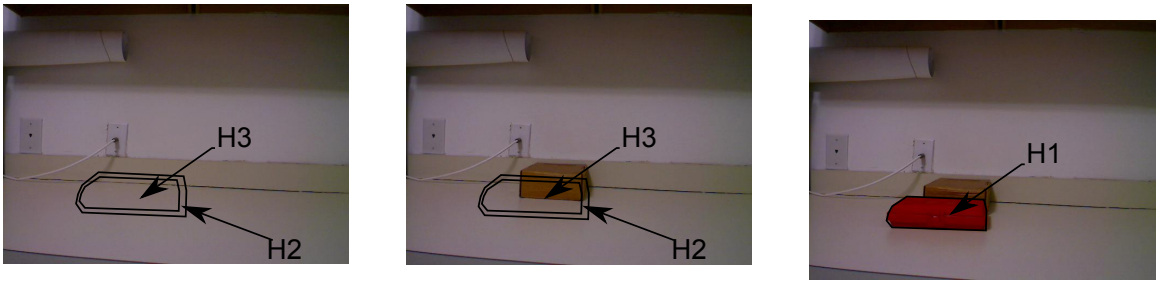
At the time of formation of a layer, it is possible that an object already in the background moves and creates a ghost, and it will be recognized as a new object layer. We differentiate between the ghost and a real object by histogram comparison as done in [32]. We prefer histogram comparison over edge detection [29], [30] for ghost removal because edge detection is not robust in the presence of cluttered background [31] and shadows. In our case, the background becomes cluttered with edges and shadows as new midground objects keep on appearing. The histogram comparison, used to remove ghosts, is based on three histograms. The first histogram is of the newly formed layer H1, the second is of the area surrounding the new layer H2 in the main background layer. The third H3 is a histogram of the main background layer under the new layer.

After calculating normalized histograms, Bhattacharyya distance is calculated between H1 and H2 dist(1,2), and H2 and H3 dist(3,2). If the difference of dist(3,2) and dist(1,2) is greater than a threshold, then the layer object is classified as a ghost, i.e. if H1 is more similar to H2 as shown in Figure 11. Moreover, if an object layer is declared as a ghost then the main background layer is updated for a correct reference for future ghost detections. The update is done by assigning new layer mode values to the main background layer, and then this superficial new layer is deleted.

It is important to note here, the main background layer is the first layer which is formed at the time of background model initialization and contains the original background. Because the background continuously changes as new layers are added or deleted, H3 and H2 is computed based on the main background layer. The change of the background is shown in Figure 12, when the red box is added, the background underneath has been changed. To perform correct ghost removal analysis for the red box we calculate H2 and H3 based on the main background shown in part (a), and not on the changed background shown in part (b).

(a) (b)

Figure 11: Ghost removal based on three histograms



(a) (b) (c)

Figure 12: Changing background scenario: (a) Original BG (b) Brown box added (c) Red box added (use main background layer (a) to calculate H2 and H3)

2.3.0.4 *Occlusion reasoning*

In the layer data structure, three counts are used for occlusion reasoning. V_t tracks the number of pixels visible of an object layer at a particular instant of time, O_t counts the number of pixels that are being occluded by pixels of other layers, and U_t represents the number of pixels of the layer which have moved from their place, resulting in the appearance of pixels belonging to underlying layers.

These three counts give important information about the current occlusion relationship of an object layer with other object layers. The objects that have moved from the scene are deleted based on the fact that their U_t approaches 100% (or their V_t reaches 0%) of the the area that is not occluded, while O_t remains the same. D_t counts the number of times this situation exists, after which the layer is deleted. On the other hand, if a new object occludes the old ones the V_t decreases and O_t increases, while U_t remains the same. This deletion of layers conserves the resources and avoids aliasing by an object that has left the scene, but whose pixels remain in the background model. In this occlusion reasoning, the main background layer is the first layer which is formed at the time of background model initialization, with all other registered midground layers on top of it, and the moving foreground is the unregistered last layer. Moreover, to make the decision to remove an object layer, a portion of the object needs to be visible. This is due to the fact that if an object is fully occluded you cannot detect the U_t even if the object moves. In this fully occluded scenario, the moved object cannot be deleted, and will be dealt with only when the area where it was present starts getting unoccluded. The main steps in our TM3 algorithm are given in the form of pseudo-code in Figure 13.

2.4 Results

The algorithm is implemented on a single core of an AMD Phenom™ II system (processor clock frequency of 2.7 GHz), with Windows 7 operating system. Visual studio C++ environment was used for development, performance analysis, and optimization of the code.

We compare our approach with Kim et al. [1], which is a representative pixel-based approach. A difference in the output is that they use a different color spectrum to show midground objects, and they show moving foreground and background as they originally appear.

---

for every **input** frame from the video sequence **do**

1. **perform** multimodal mean background modeling
2. **if** $OC_{x,t} < T_f \wedge t - RCN_{x,t} > T_{nf} \wedge Layer_j = Null$ **then** delete foreground object's pixel
3. **if** $Age_{x,t} > T_f \wedge OB_{x,t} > \delta \wedge Age_{x,t} < PrevLayerAge$ **then** mark pixel as midground
4. **when** a blob of midground pixels stops growing **then** register a new layer C = pixel-count, $V_t = C, O_t = U_t = D_t = 0$
5. **if** $dist(3,2) - dist(1,2) > threshold$ **then** assign this layer's values to main background layer and delete this ghost layer
6. **if** a pixel of $Layer_j$ is occluded by another layer **then** increment O_t of $Layer_j$
7. **if** a pixel of a layer, occluded by $Layer_j$, becomes visible **then** increment U_t of $Layer_j$
8. **if** $U_t/(C - O_t) \approx 100\% \wedge (C - O_t)/C > \%threshold$ **then** increment D_t
9. **if** $D_t > threshold$ **then** delete the layer
10. **mark** layers according to their age in the **output** frame

end for

Figure 13: TM3 algorithm

Four datasets were used to test different aspects of our algorithm. The details of the datasets are shown in Table 2. The Outdoors dataset highlights our algorithm's ability to detect an abandoned midground object in a densely crowded outdoor setting. Blocks dataset is used to demonstrate the basic working of our algorithm. In this dataset, multiple blocks are added to a scene at different points in time in an indoor environment. The third

dataset, Boxes, shows the addition of multiple objects that occlude each other. This experiment shows how our algorithm correctly removes an object based on occlusion reasoning. The last dataset, Cars, is a more elaborate real world example occurring in a parking lot, where different vehicles arrive and/or leave the scene. This dataset is used to show different working aspects of our algorithm such as ghost removal, object removal based on occlusion reasoning, and noise filtering in an outdoor scene.

Table 2: Datasets Specifications

Dataset	Outdoors	Blocks	Boxes	Cars
Resolution	640x480	640x480	640x480	352x232
No. of frames	2900	672	1770	12000
fps	10	5	15	15

2.4.1 Experiments

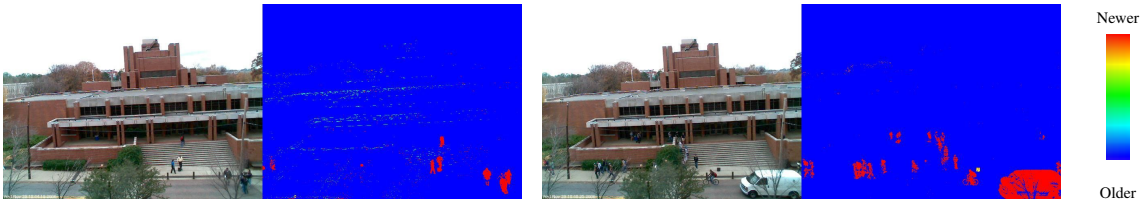
Abandoned object detection. The scenario shown in Figure 14 demonstrate an abandoned object detection application of our algorithm. It shows an object that has been abandoned in a very crowded outdoor environment. Our algorithm correctly detects and differentiates the abandoned midground object from the moving crowd of people forming the foreground. This is a difficult task for the human visual system.

Basic operation based on Blocks dataset. This scenario shows the basic working of our algorithm. To begin, there are multiple objects present in the scene Figure 15(a). They are all part of the background. Then some objects are added to the image as depicted in Figure 15(b). Next a few more objects are added and removed from the scene (Figure 15(c)). One object that was removed, marked by a white circle, is part of the background. Our algorithm correctly identifies it as a ghost and not a new object. Moreover, the longer an object stays in the scene the bluer it becomes, Figure 15(d), which means it is nearer to

the original background in terms of its age. The summary of the sequence of events in the scene provided in a single frame Figure 15(d) is also a difficult task for the human visual system.

Occlusion reasoning based on Boxes dataset. In this scenario, we show our algorithm performing occlusion reasoning and the removal of the layers in an occlusion scenario. Figure 16 shows the addition of two objects in different points in time, as indicated by their assigned colors in Figure 16(b); the one closer to background in color is older than the other. Next we remove the older brown box, Figure 16(c). We can see that the brown box was partially occluded by the red box, but based on occlusion reasoning; our algorithm correctly removes the brown box layer. This is verified by the fact that when we add a box with similar color, the older layer does not show up, and the whole object is marked as a new object, Figure 16(d). This is not the case in the pixel-based approaches [1],[2] as shown in Figure 17.

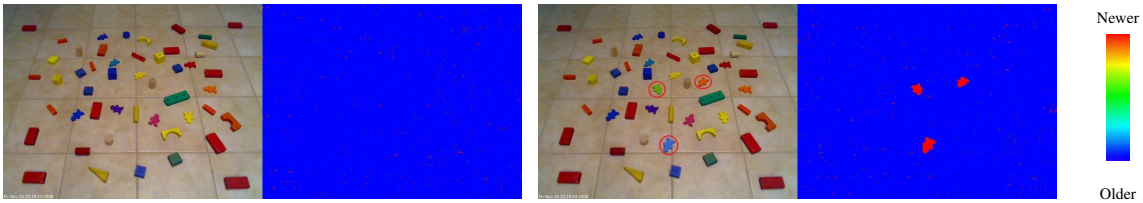
Real-world example based on Cars dataset. The last example is a more elaborate real world scenario in a parking lot, where different vehicles arrive and leave. Figure 18(a) shows the original background scene where there are few cars. Later, one car which is part of the background moves, and it is correctly identified as a ghost object and removed, as shown in Figure 18(b). In the subsequent frames, Figure 18(c, d, e, f), more cars come and leave with our algorithm correctly identifying their time of arrival and departure. The summary of the sequence of events in the scene is provided in a single frame Figure 18(f), which would also be a difficult task for the human visual system to do. Figure 19 shows the resulting last frame 18(f), based on the pixel-based approaches [1],[2] (Figure 19(c,d)) and our current object-based approach (Figure 19(e)). It can be seen in Figure 19(c,d) that the remnants of previous layers that match in color have corrupted the current layers. This is because the pixel-based approach cannot remove object layers based on occlusion reasoning and cannot identify the ghost objects. These problems are resolved by our object-based approach as shown in Figure 19(e).



(a) Frame-1185 (b) Frame-1416

(c) Frame-1867 (d) Frame-2432

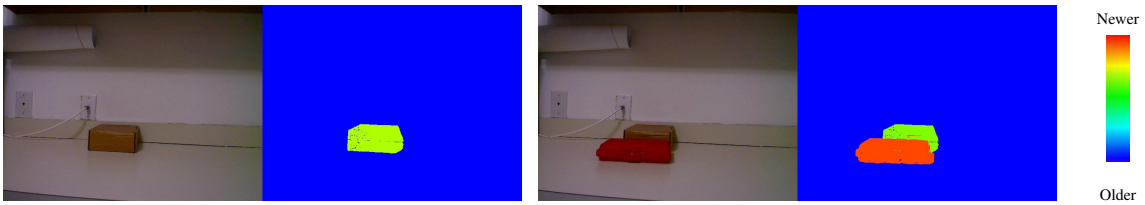
Figure 14: Abandoned object detection in a crowded scene at different points in time



(a) Frame-197 (b) Frame-199

(c) Frame-286 (d) Frame-346

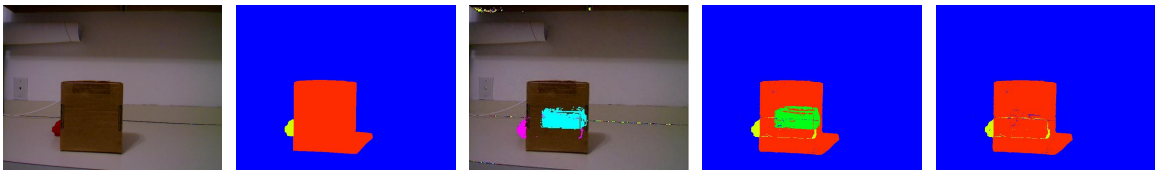
Figure 15: Blocks added and removed at different points in time, a red circle indicates new entries while blue indicates the ones already there, a black circle indicates the objects removed while white indicates the ones removed from the initial background: (a) Initial background (b) Three blocks added (c) Three more blocks added, three removed including one from initial background (d) Three more blocks added



(a) Frame-1244 (b) Frame-1340

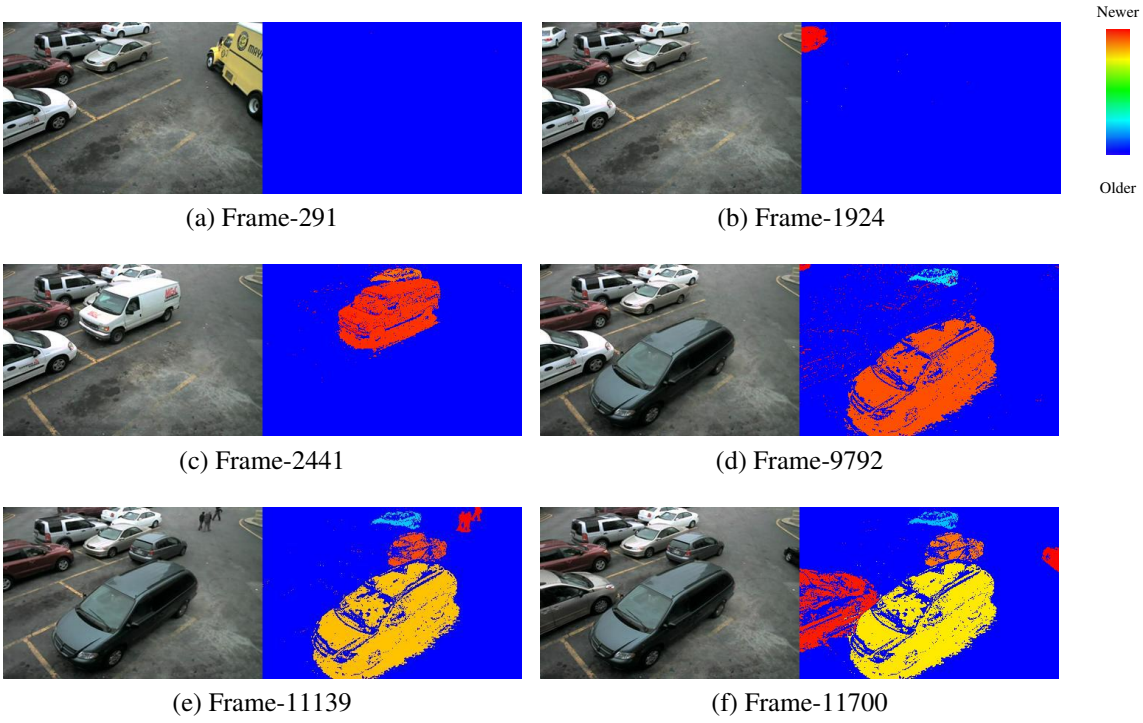
(c) Frame-1670 (d) Frame-1754

Figure 16: Object layer removal based on occlusion reasoning: (a) Brown box added (b) Red box added occluding brown box (c) Brown box removed (d) Brown box added occluding red box



(a) (b) (c) (d) (e)

Figure 17: Occlusion reasoning effect: (a) Original image (b) Ground truth (c) Pixel-based [1] (d) Pixel-based [2] (e) Object-based [TM3]

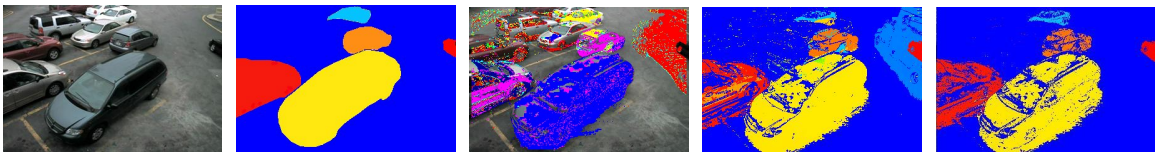


(a) Frame-291 (b) Frame-1924

(c) Frame-2441 (d) Frame-9792

(e) Frame-11139 (f) Frame-11700

Figure 18: Cars entering and leaving a parking lot: (a) Initial background (b) Yellow truck leaves (c) White car and van arrive (d) White van and white car from initial background leaves and black van arrives (e) Gray car arrives (f) Silver and black cars arrive



(a) (b) (c) (d) (e)

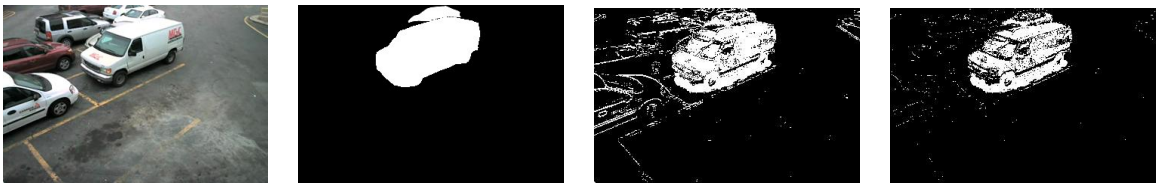
Figure 19: Pixel vs. object-based modeling: (a) Original image (b) Ground truth (c) Pixel-based [1] (d) Pixel-based [2] (e) Object-based [TM3]

2.4.2 Performance metrics

We now present our algorithm performance metrics starting with accuracy, and then computational speed and memory utilization.

Figures 20, 21 show our accuracy results at observability threshold of 50%, with all layers shown as white foreground, in outdoor and indoor environment respectively. Part (c) shows layers without any filtering, and part (d) shows the result after applying filtering at the given observability threshold. Figure 22(a), 23(a) show that after filtering, the false positives decrease, i.e., background noise is reduced. On the other hand, false negatives increase due to the fact that some foreground pixels are wrongly classified as background during filtering.

The main advantage of this filtering is that it eliminates background noise, Figure 20(c), 21(c), which can be considered as new object layers by the higher level module while forming object layers. The natural question is what if we decrease or increase the observability threshold. If we reduce it, we will have less foreground error but more and more background noise will start accumulating, which can result in false layers. On the other hand, if we increase it too much, we will have less background noise but more foreground error which can ultimately result in erroneous layer formation, i.e. multiple layers for a single object. Figure 22(c) and 23(c) show the relationship between observability threshold and number of layer errors for the outdoor and indoor scenes, with 11 and 3 respective actual number of layers. The difference between the outdoor and the indoor scene is that the outdoor scene has more persistent dynamic background noise (mostly due to reflections from vehicles), Figure 22(b) vs. 23(b), which results in formation of false layers as shown in Figure 22(c) but not in Figure 23(c). These false layers go away when we increase the observability threshold. On the other hand, increasing the filtering constant too much results in erroneous layers (multiple layers for a single object) both in indoor and outdoor environments with maximum error at 100% observability threshold. Based on both the outdoor and indoor environment analysis, a observability threshold range between 50-75%



(a) (b) (c) (d)

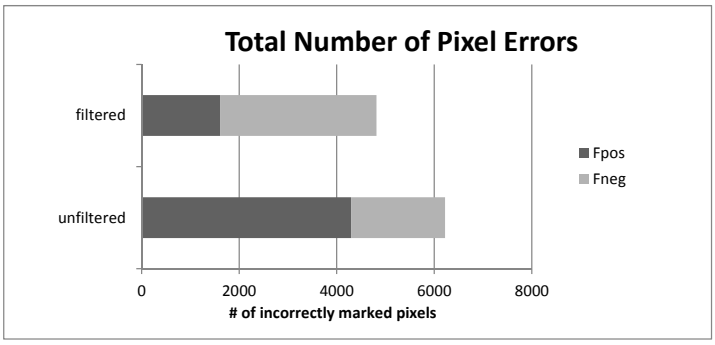
Figure 20: Outdoor cars, filtering at 50% observability threshold: (a) Original image (b) Ground truth (c) Unfiltered (d) Filtered

(a) (b) (c) (d)

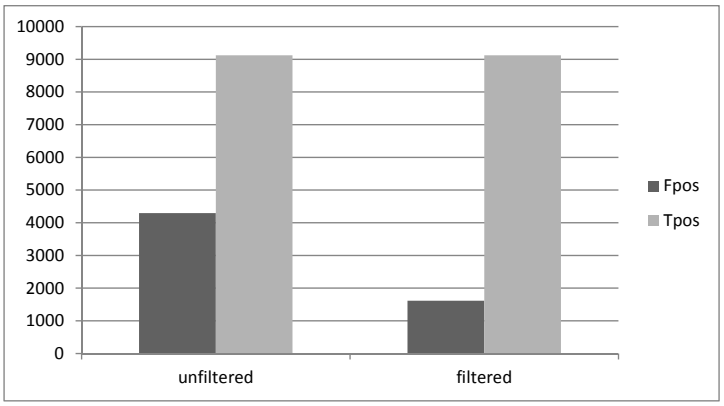
Figure 21: Indoor boxes, filtering at 50% observability threshold: (a) Original image (b) Ground truth (c) Unfiltered (d) Filtered

is optimum in giving the least pixel/layer errors.

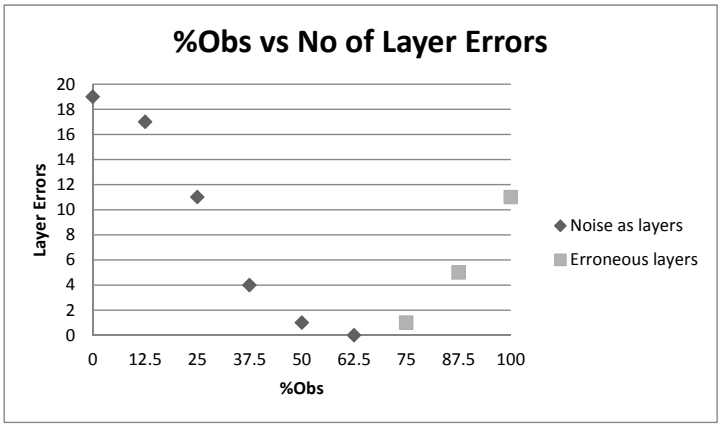
Table 3 shows the speed comparison between the basic MMM algorithm and the TM3 algorithm in frames per second. Variability in speed may be caused by different number of midground objects entering the scene in the case of TM3. The main factor in the code optimization is to make the main memory accesses only when required. The algorithm TM3 with all of its new capabilities requires approximately twice the latency of the baseline MMM algorithm. Previous experiments [20] show that the MMM algorithm runs 4.23x faster than the widely used Gaussians mixture model (GMM) background modeling technique [8] on a general-purpose CPU platform. This high speed of TM3 is mainly due to the fact that we model objects only on the basis of multi-layer background modeling. In other words, we don't track each object continuously, instead we form a blob, and subsequently the object layer, only when an object becomes stationary.



(a)

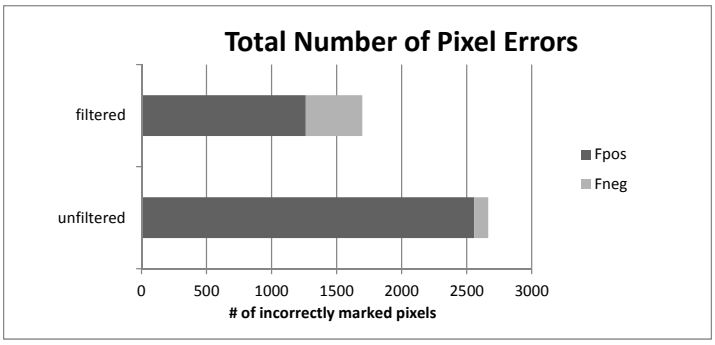


(b)

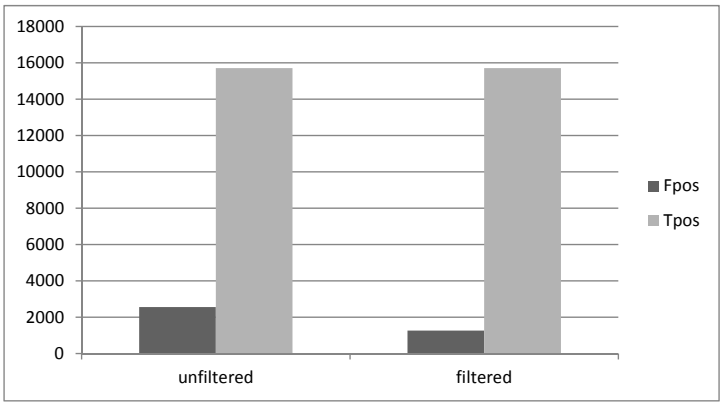


(c)

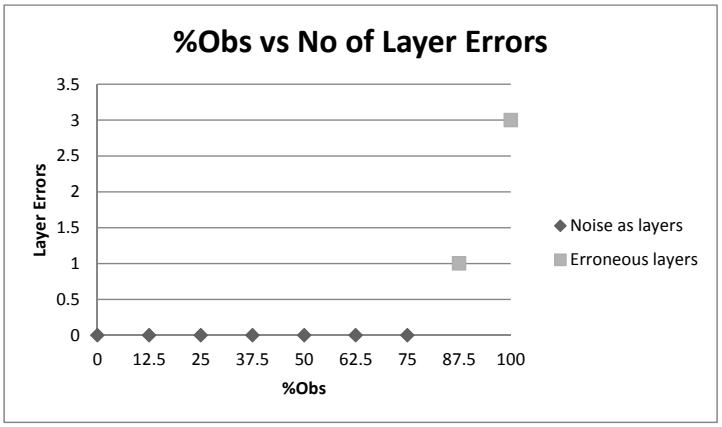
Figure 22: Outdoor cars: (a) Total number of pixel errors at 50% observability (b) FP vs TP at 50% observability (c) % Observability vs. no. of layer errors



(a)



(b)



(c)

Figure 23: Indoor boxes: (a) Total number of pixel errors at 50% observability (b) FP vs TP at 50% observability (c) % Observability vs. no. of layer errors

Table 3: MMM and TM3 Speed Comparison in FPS

Algorithm/Dataset	Outdoors	Blocks	Boxes	Cars
MMM	54	59	60	173
TM3	25	31	40	92

In terms of memory, TM3 requires three extra integers for every pixel mode over MMM which requires five integers as shown in Figure 24. Therefore, the memory overhead is 8/5 per pixel mode. The average number of modes per pixel in the given datasets with limited frames is approximately the same for both TM3 and MMM. The dynamic nature of the background and overlapping midground objects both require multiple modes per pixel, and so they balance the requirement of the modes per pixel in both MMM and TM3. However, the number of modes is bound to be more for TM3 in longer datasets with less dynamic background, as more and more overlapping midground objects are added to the scene. Another place where we require extra memory in TM3 is for writing the color output to show different object layers. This color output as an RGB value requires three bytes per pixel. In addition, we require an integer per pixel in blob formation for the density map, and a pointer per pixel to point to the current status of background model in the ghost removal step. The memory required for layer and blob data structure is negligible. Therefore, the total memory overhead for the TM3 algorithm is a minimum of 11 words per pixel compared to 5 i.e. approximately 2x that of MMM.

$S_{x,t.r}$	$S_{x,t.g}$	$S_{x,t.b}$	$C_{x,t}$	$RCN_{x,t}$	BD_x	$OC_{x,t}$	$LAYER_j$
$S_{x,t.r}$	$S_{x,t.g}$	$S_{x,t.b}$	$C_{x,t}$	$RCN_{x,t}$			

Figure 24: Mode comparison of MMM and TM3

The high speed and low memory requirement with good accuracy makes our algorithm amenable to embedded platforms which have limited memory and execution power.

2.5 Conclusion

In this chapter, a multilayer background modeling technique, called TM3, is presented for video surveillance. Rather than simply classifying everything in a scene as either dynamically moving foreground or long-lasting, stationary background, a temporal model is used to place each scene object in time relative to each other. Foreground objects that become stationary are registered as layers on top of the background layer. In this process of layer formation, the algorithm deals with ghost objects, and noise created by dynamic background and moving foreground objects. Objects that leave the scene are removed based on the occlusion reasoning among layers. This technique allows us to represent a scene with multiple midground objects entering, leaving, and occluding each other at different points in time. This leads to richer representation of temporal properties of scene objects than traditional foreground/background segmentation. In particular, the information includes when a particular object arrived or left, and the occlusion relationship among different objects while they are in the scene.

The TM3 technique builds on a low-cost MMM background modeling technique [20] which makes it suitable for embedded, real-time platforms. It adds approximately twice the latency and storage requirements of MMM. However, these costs remain relatively low given that the MMM algorithm runs 4.23x faster than the widely used Gaussian mixture model (GMM) technique [8] on a general-purpose CPU platform, while exhibiting comparable performance in accuracy.

CHAPTER 3

MULTI-LAYER BACKGROUND MODELING: SPATIO-TEMPORAL SCENE ANALYSIS

3.1 Introduction and related work

In traditional background modeling techniques midground objects are quickly assimilated into the background. Simple two-layer extensions of these techniques [19] coarsely model objects in the foreground to background transition, but do not differentiate among them based on their spatio-temporal properties. Object appearance events often occur over varying time scales, and there are situations in which many objects appear on the scene at different points in time and become stationary. Modeling these scenarios is helpful, for example, in analyzing a parking lot or a traffic intersection scene to determine the order in which vehicles have arrived. To analyze and visualize such scenarios, our TM3 approach models multiple temporal layers of midground objects. This mechanism differentiates multiple objects, and correctly removes or adds objects, even in occlusion situations.

The problem with multilayer and two-layer background modeling techniques is that they are unable to deal with partial or full midground/background objects displacements. Issues arise when the position of an object in the original background changes, or a midground object is fully or partially displaced from its location. These objects will be recognized as new objects upon change of position. Multiple-object tracking algorithms, presented in section 2.2, can deal with these spatial displacement scenarios, but the computational cost associated with using a tracking algorithm is too high, especially for embedded platforms. For example, the cost for a popular appearance-based multiple-object tracking algorithm [18] is 15-20 frames per second (320x240 frame size) on a 3GHz P4 system.

This chapter proposes a novel mechanism, called *spatio-temporal multimodal mean* (STM3), which performs spatio-temporal reasoning based on multi-layer background modeling and color features of the displaced object to conserve the state of the moved objects in a scene. We extend our temporal multimodal mean (TM3) algorithm to spatial analysis, in order to correctly model objects upon change of position. A set of removed midground/background objects is maintained based on object removal information provided by the TM3 algorithm. The newly observed objects are compared with the set of removed objects based on the object's appearance model to decide whether the object is a brand new entity, or if it is old and has only been displaced. The end result is that we are able to correctly model the objects in the case of spatial displacements. Application scenarios of multiple midground objects occur frequently in the surveillance of homes, offices, and public places where many objects enter, exit, stop, or start moving at different points in time. An important application scenario of our algorithm is its ability to tell that an object in the original scene has only been moved to a different location and has not disappeared.

This chapter is organized as follows. Section 3.2 presents our proposed method of spatio-temporal multi-layer background modeling. Section 3.3 describes our experiments and results. We conclude the chapter in Section 3.4.

3.2 Spatio-temporal multimodal mean (STM3)

The spatio-temporal multimodal mean (STM3) algorithm incorporates salient spatial information into the TM3 approach to produce a richer spatio-temporal scene understanding mechanism. The spatial extension of the TM3 algorithm must model midground/background objects upon their displacements. We will consider both full and partial displacement, starting with the full object displacement.

The TM3 algorithm gives us information about background object removal based on ghost object detection, and midground object removal based on occlusion reasoning. Using this information, a set of removed objects is maintained along with their histogram

of objects color features. The removed object data structure is shown in Figure 25:

ID	TS	RTS	$P_{n,1}$

Figure 25: Data structure for a removed object

ID is the object layer identification, TS and RTS are the object layer formation and the removal timestamps respectively, and $P_{n,1}$ is an n-bin histogram column vector of object color features. We quantize each of the three RGB color components (a color component consists of 256 color intensity values) into m bins (each incorporating 256/m color intensity values), which makes the number of our RGB color histogram bins n equal to m^3.

When a new midground object is detected, it is compared with each object in the removed object set based on the Bhattacharyya distance [39] as given below:

$$dist(PQ) = \sqrt{1 - \rho(PQ)}\,, \tag{13}$$

$\rho(PQ)$ is the Bhattacharyya coefficient given by:

$$\rho(PQ) = \sum_{k=1}^{n} \sqrt{P(a_k)Q(b_k)}\,. \tag{14}$$

where $P(a_k)$ and $Q(b_k)$ are the two normalized histograms with n number of bins specified by a_k and b_k respectively. If the Bhattacharyya distance between the histograms, of the new object and a removed object, is below a threshold then the new object is marked as an old midground object or a background object, depending upon whether it was a real midground object or a background object that created a ghost. Moreover, the layer identification ID and the timestamp TS of the new object are assigned the values of the removed object. An object is kept in the removed object set for a certain time, after which it is deleted from the set, based on its removal timestamp RTS, assuming that the object has permanently left the scene.

In the case of partial object displacement or a newly appeared part of a midground object, the newly detected midground blob's bounding box will intersect with the old midground object. If intersection happens, the histogram comparison of the blob and the

midground object layer is performed as described above to determine whether the blob is part of the old midground object or not. If the blob is marked as part of the old midground object then the blob pixels are assigned to the old object layer. Furthermore, the bounding box (X_{min}, X_{max}, Y_{min}, and Y_{max}) and the count C of the layer are re-calculated. In addition, in the case of a partially displaced object, the layer pixels from where the object has been displaced are deleted from the model.

3.3 Results

The algorithm is implemented on a single core of an AMD Phenom™ II system (processor clock frequency of 2.7 GHz), with Windows 7 operating system. Visual Studio C++ environment was used for development, performance analysis, and optimization of the code.

We use a dataset of boxes, with 640x480 resolution and 15 frames per second, to demonstrate various scenarios of partial and full object displacement and how our STM3 algorithm deals with them. The objects, depending upon their age, are depicted using colors ranging from red (newest) to blue (oldest, original background) based on the standard visible color spectrum.

In the first scenario, an object in the original background has been moved to a different location. This scenario is shown in row 1 of Figure 26. The TM3 algorithm detects a ghost at the original location of the object and marks it as background, but it will also detect a new object at the location to which the object is moved part (d), whereas the reality is that it is the same old object which is part of the background as shown by our STM3 algorithm in part (e). The second scenario, in which the moved object is not part of the original background, but a midground object, is shown in row 2 of Figure 26. TM3 will detect a new object at the location where the object is moved part (d), whereas the reality is that it is an old midground object which has been displaced from its location in the scene as shown by our STM3 algorithm in part (e). In the third scenario, a midground object is partially displaced from its original location as shown in row 3 of Figure 26. In this case,

a portion of the object will be detected as a new object by the TM3 algorithm as shown in part (d), whereas the actual situation is as shown by our STM3 algorithm in part (e). In the final scenario, a midground object is partially occluded when it first appeared on the scene as shown in row 4 of Figure 26. When this object is unoccluded, TM3 treats the recently revealed portion as a new object part (d), whereas the actual situation is that it is a portion of an old object and not a new object as shown by our STM3 algorithm in part (e).

We use a second dataset, also with 640x480 resolution and 15 frames per second, to show our algorithm working for scenario-1 with another real-world example Figure 27. The figure shows that the position of a bag present in a room has been changed. The two-layer and multilayer background modeling algorithms like TM3 will signal an alarm of the presence of a new object Figure 27 (d), whereas in reality it is an object which was already there in the scene as shown by our STM3 algorithm Figure 27(e). This is an important application scenario of our algorithm with its ability to tell that an object in the original scene has only been moved to a different location and has not disappeared.

We have tested our algorithm on m equal to 4, 8, 16 ($256/m$ equal to 64, 32, 16 intensity values per bin respectively), which makes the RGB color histogram 64, 512, and 4096 (m^3) bins respectively. 512-bin histogram gives the best result since the gap between an object distance with itself ($dist(PP)$) and a different object ($dist(PQ)$) is the greatest, as shown in Figure 28(b). In 64-bin histogram the gap is reduced because $dist(PQ)$ decreases, Figure 28(a), as bin size (64) is too wide. In 4096-bin histogram the gap is reduced because $dist(PP)$ increases, Figure 28(c), as bin size (16) is too narrow.

The TM3 technique adds approximately twice the latency and storage requirements to a low-cost adaptive background modeling technique MMM [20]. However, these costs remain relatively low given that the MMM algorithm runs four times faster than the widely used Gaussian mixture model (GMM) adaptive background modeling technique [8] on a general-purpose CPU platform, while exhibiting comparable performance in accuracy, as shown in [20]. The latency overhead of STM3 in comparison with TM3 is small. The

STM3 algorithm runs at 33 fps whereas the TM3 algorithm runs at 38 fps for the given boxes dataset. In terms of memory storage, the STM3 algorithm requires additional 515 words (see Figure 25, considering n equal to 512) for each removed object, which is negligible compared to a minimum of 11 words per pixel in a 640x480 (307200 pixels) frame scene, required for TM3.

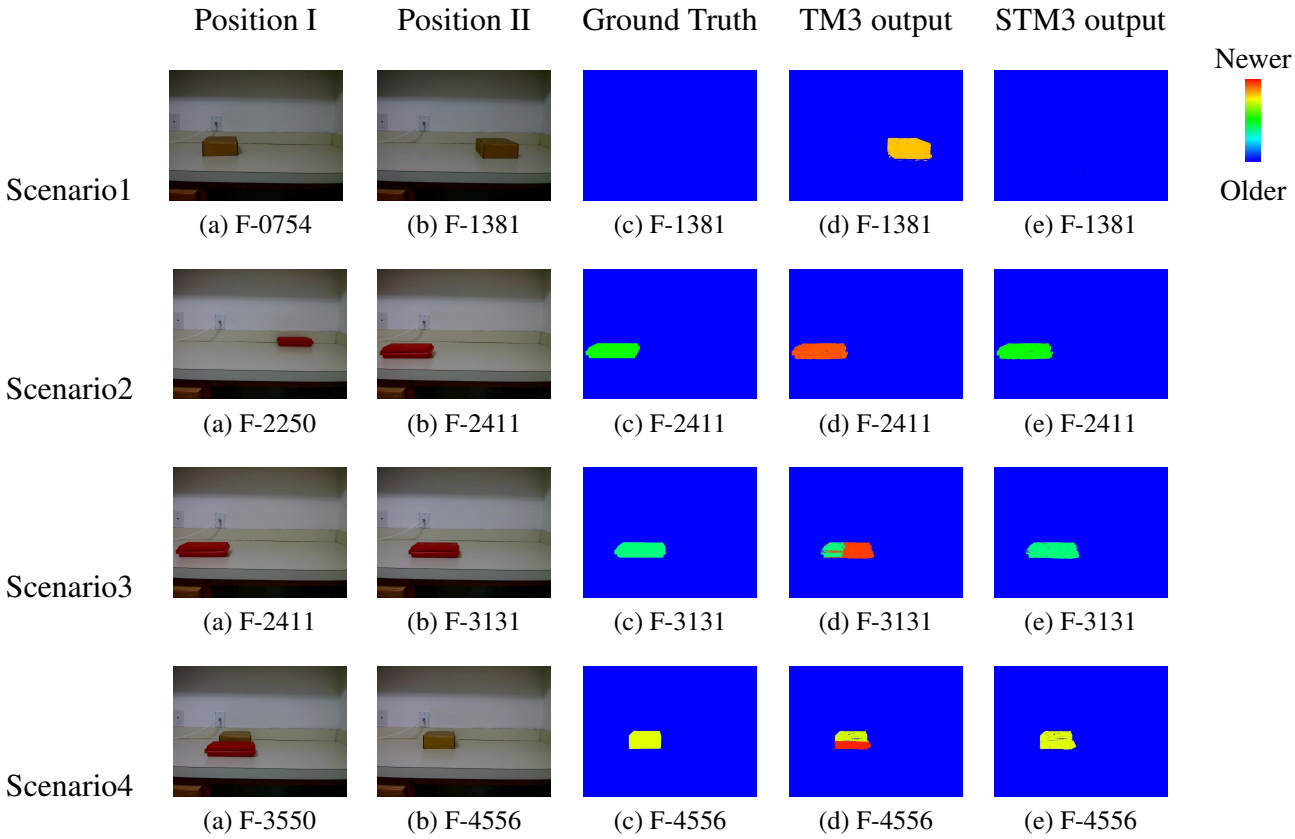


Figure 26: Spatial displacement scenarios: Scenario1, moved object from original background; Scenario2, moved object; Scenario3, partially displaced object; Scenario4, partially occluded object

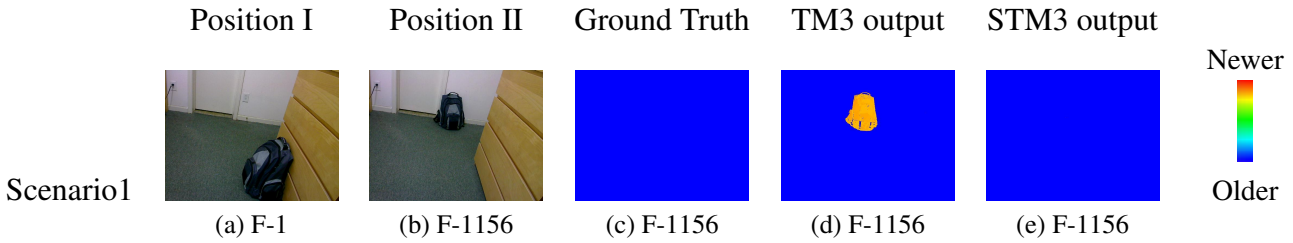
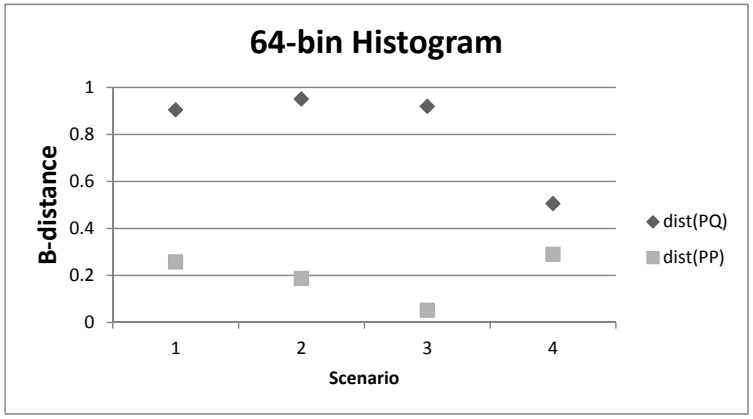
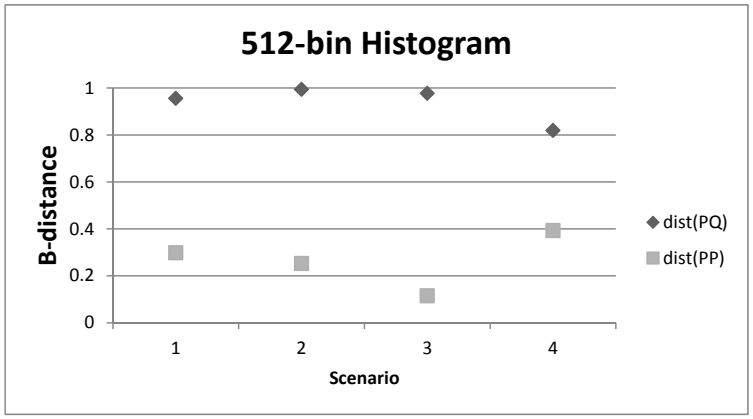


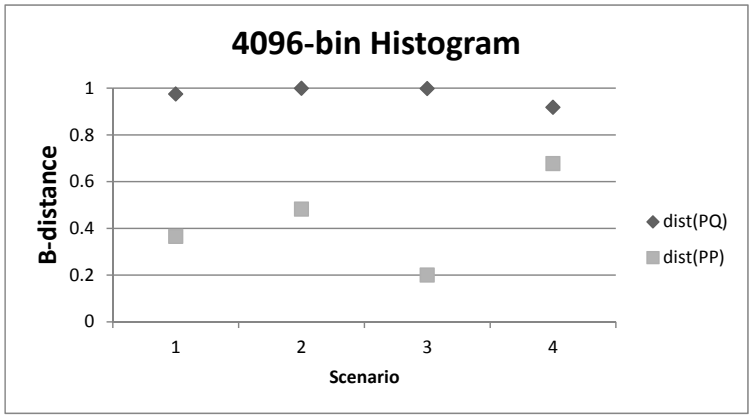
Figure 27: A change in a bag position has been recognized



(a)



(b)



(c)

Figure 28: An object distance with itself *dist*(*PP*) and a different object *dist*(*PQ*) in the four scenarios: (a) 64-bin histogram (b) 512-bin histogram (c) 4096-bin histogram

3.4 Conclusion

The background modeling techniques that model objects that have become stationary will incorrectly detect a new object if an existing stationary object is displaced. A novel spatio-temporal reasoning mechanism is presented based on multi-layer background modeling to conserve the state of moved objects in a scene. The mechanism models layers of the foreground objects that have become stationary, along with moving foreground and background. Objects that change their place, partially or fully, are recognized based on their color histogram appearance model. The end result is the correct modeling of objects in the case of spatial displacements. This provides us with a richer mechanism of analyzing and visualizing spatio-temporal scene events than the traditional binary foreground/background segmentation.

CHAPTER 4

ACCELERATING ADAPTIVE AND MULTILAYER BACKGROUND MODELING ON LOW-POWER GPUS

4.1 Introduction and related work

Background modeling is a key initial step in many video surveillance applications. As more and more smart cameras are deployed for surveillance tasks across the globe, an efficient background modeling technique is required that balances accuracy, speed, and power. Basic background modeling techniques, such as frame differencing and single parametric [25] (see section 2.2), run fast, but their accuracy is not sufficient for computer vision problems involving dynamic background, such as waving tree branches and rippling waves. In contrast, adaptive multimodal background modeling techniques, such as the state of the art Gaussian mixture model (GMM) [8] are more robust to dynamic background, but run more slowly.

The fine-grain parallelism inherent in many background modeling techniques motivates their implementation on GPUs, which are known for their ability to exploit massive parallelism. Since the advent of parallel computing architectures, such as CUDA, that allow for easy development of general-purpose applications on GPUs, more and more surveillance applications are being targeted to GPU platforms, and adaptive multimodal background modeling is no exception. Many variants of the multimodal background modeling techniques have been implemented on general purpose GPUs since then, as shown in Table 4. Carr [40] and Fabian and Gaura [5] implement the popular GMM technique [8] discussed in section 2.2, Pham et al. [6] implement an extended version of the GMM technique, while Poremba et al. [7] implement a different adaptive background modeling technique also based on the Gaussian mixture model. The implementations in Table 4 are listed in the order of increasing cores and therefore increasing speed ups over single core CPU, but the power required by each platform also increases accordingly. Apart

from Carr's implementation, other implementations achieve significant speed-ups, but with general-purpose NVIDIA GeForce GPUs that consume approximately 100 watts. This high power consumption of the general purpose GPUs is not suited for embedded platforms. In practice, embedded applications must achieve real-time performance with the limited power of an embedded smart camera. A typical high-end embedded CPU-based smart camera consumes power in the lower tens of watts (e.g. Intel Atom based NI177x from National Instruments [11] (12 watts), and Iris GT from Matrox [12] (10 watts), VIA-Eden-ULV based XCISX100C/XP from Sony [13] (18 watts), Analog Devices Blackfin based ILC-BL from Intellio [14] (18 watts)). The question then is how best to balance accuracy, speed, and power; so that we can achieve speed-up over an embedded CPU by parallelizing adaptive multimodal algorithm, and while doing so maintain the power of the system within limits of an embedded smart camera.

We address this for background modeling in two ways: 1) by using a robust algorithm that has low computational and memory costs, and 2) by exploiting the data and thread level parallelism in this algorithm and optimizing its memory access patterns to target a low-power GPU platform. In particular, we focus on the multimodal mean (MMM) algorithm [20], which performs adaptive background modeling for indoor or outdoor scenes that may include dynamic, multimodal background, such as fluttering leaves or rippling waves. As discussed in section 2.2, MMM requires fewer computational and memory resources, and it executes four times faster than the GMM technique on a general-purpose CPU platform, while exhibiting comparable performance in accuracy [20]. For efficient execution of this algorithm, we target a low-power integrated GPU: the NVIDIA ION. The ION GPU has 16 CUDA cores and a maximum thermal design power (TDP) of only 12 watts. This contribution focuses on how data and thread-level parallelism is exploited and memory access patterns are optimized to target adaptive background modeling algorithm to this low-power GPU. The first task of this contribution is to accelerate the traditional multimodal mean algorithm MMM and then extend it to our multi-layered TM3 version.

This chapter is organized as follows. Section 4.2 presents our implementation of the MMM and the multi-layer TM3 algorithms. Section 4.3 describes our experiments and results. We conclude the chapter in Section 4.4.

Table 4: Multimodal Background Modeling on GPUs

Algorithm	GPU model	Cores	TDP in watts	Speed-up	Authors
GMM [8]	GF 8600M GT	32	22	5x	Carr [40]
Extended GMM [9]	GF 9600 GT	64	96	10x	Pham et al. [6]
GMM [10]	GF 9800 GT	112	105	18x	Poremba et al. [7]
GMM [8]	GF 260 GTX OC	216	182	26x	Fabian and Gaura [5]

4.2 Methodology

We describe here the design methodology for a CUDA implementation of the MMM algorithm. We begin this section by giving an overview of the CUDA (Compute Unified Device Architecture) architecture. We then present our implementation of the MMM algorithm on the CUDA platform. Subsequently, we extend the MMM GPU implementation to our multilayer TM3 implementation.

4.2.1 CUDA platform

CUDA is a hardware and software architecture by NVIDIA for general purpose computing on its GPUs. Figure 29 shows an overview of an NVIDIA GPU based on the CUDA architecture. The CUDA hardware architecture is based on a hierarchy of compute and memory resources. An NVIDIA GPU based on CUDA has N streaming multiprocessors (SMs), with each SM having M streaming processor (SP) cores based on a SIMT (Single Instruction Multiple Threads) architecture. Each SP has its private register space, each SM has fast on-chip cache shared by its SPs called shared memory, and the whole GPU has an off-chip main memory called device memory or global memory shared by all of its SMs.

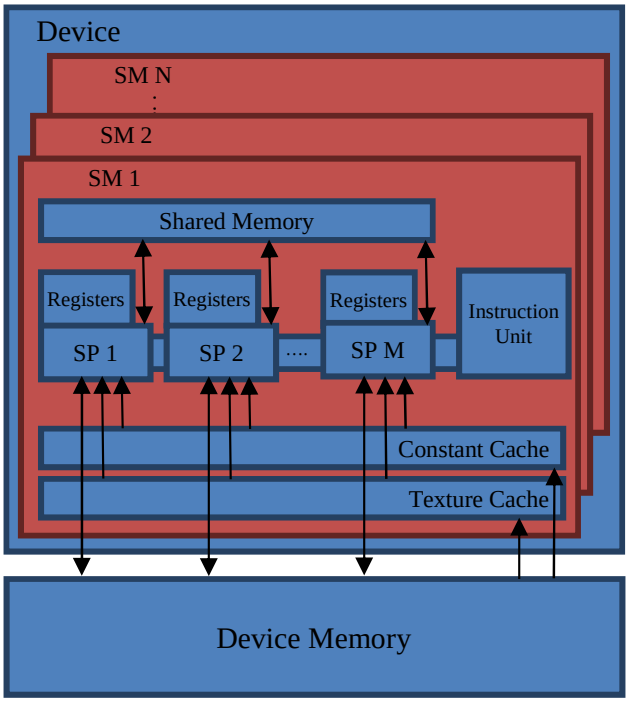


Figure 29: CUDA based NVIDIA GPU architecture

(In our discussion, we use the terms device memory and global memory interchangeably.)

With respect to its software architecture, the CUDA programming model has a hierarchy of threads [41]. At any given instance of time, a single thread runs on top of an individual SP core, and the thread has access to the private register space of that core. Blocks of threads run on top of an SM, and the threads from a block are executed in a group of 32 (called a warp) onto M SPs (on our platform M = 8) of an SM. Threads in the same block can use the shared memory of the SM for information sharing and synchronization. Ultimately, these thread blocks form a grid which runs on the whole GPU. All the threads in the grid have access to the global memory, and threads within different blocks can only share information and synchronize through the global memory. CUDA's programming language is an extension of C with additional APIs to handle code execution on the GPU. The main function runs on the CPU (host), which then invokes a parallel GPU (device) code function referred to as a kernel. (In our discussion, we use the terms device and GPU, and

host and CPU interchangeably.)

The constraints on compute and memory resources of the CUDA platform are dictated by its compute capability. These constraints on the resources include registers per SM, warps per SM, shared memory per SM etc. They also include types of instructions the device can execute. For details see [41]. Keeping these constraints in mind, one has to efficiently and maximally utilize the available resources.

4.2.2 MMM implementation

Given this background on the CUDA platform architecture, we are now ready to describe implementation details of the MMM algorithm on this platform. The basic implementation approach is to find enough parallel work, while hiding the memory latency so that the GPU compute resources are kept maximally busy.

4.2.2.1 Basic implementation

In our implementation of the background modeling algorithm, we use two GPU kernels. At the start, the first image is copied from the CPU (host) memory to the GPU (device) memory, and then a kernel is launched to create and initialize the background model in the device memory. After this, for every new image, the program copies the image from the host memory to the device memory, launches a kernel to perform background subtraction for that image, and then writes the result back to the host memory. We will focus on the implementation of only the background subtraction kernel since the background model initialization kernel is used only once at the start.

In implementing the background subtraction GPU kernel, the key idea is to have enough parallel work, while hiding the memory latency so that the GPU compute resources are kept maximally busy. First, we need to find enough independent tasks that can be run as parallel threads on the whole GPU. In the MMM algorithm, operations performed on an individual pixel are independent of the others, and an image has enough pixels to occupy the whole GPU. Second, there are two commonly known ways by which the memory latency

```
int threadsPerBlock = 64;
int blocksPerGrid = (Width*Height + threadsPerBlock - 1) / threadsPerBlock;

// invoking the kernel function for frame N from CPU
MMM<<<blocksPerGrid, threadsPerBlock>>>(frame,frame_bin,N);

//CUDA background subtraction kernel function to be run on GPU
__global__ void MMM(unsigned char* frame, unsigned char* frame_bin,  int frame_no)
{
        unsigned int tId = blockIdx.x * blockDim.x + threadIdx.x;
        // operations on individual pixel, with tId as index, goes here
        // as explained in the MMM algorithm
}
```

Figure 30: The background subtraction kernel

can be hidden. The first is data reuse, so that the data can be reused from fast memories like the register space or shared memory. The second is by activating a massive number of threads. We do not have the first option since threads have little data reuse (mostly in the form of reading pixel RGB values and current mode to the private registers spaces), and global memory is accessed for almost every instruction by individual threads (memory-bound problem). However, we do have a massive number of threads equal to the number of pixels in an image. This means, we can have more threads in a block and more blocks on an SM, depending upon the available resources. In this way, the scheduler can put off a thread warp waiting for a memory access, dispatch a new thread warp, and later on return to the first warp. The scheduler can make a zero-overhead switch among the warps available on an SM, which effectively hides the memory latency. This means that the more warps there are per SM, the more we can hide the memory latency. Figure 30 shows the skeleton code for our kernel for background subtraction, and its launch from the CPU.

In this basic implementation, we use the GPU texture cache to read images from the device memory because image pixels have spatial locality, and are read-only. A one-dimensional texture reference is used for texture fetch since images are stored in the device memory as linear arrays of unsigned characters. This achieves a high cache hit rate (87%).

In addition, we use constant memory for fast access of the parameters used by the MMM algorithm.

4.2.2.2 Performance optimizations

To improve upon the basic implementation and to maximize speed-up, we apply a variety of architectural performance optimizations.

Coalesced global memory for accessing the background model. Perhaps the single most important consideration in a CUDA implementation is to coalesce the global memory accesses [42]. The bandwidth specified for a GPU can only be achieved if the accesses are coalesced. When certain access requirements are met, global memory accesses by threads of half warp (for devices of compute capabilities 1.x) are coalesced by device into the fewest transactions possible [42]. The requirements relate to contiguity and alignment of memory accesses, and depend upon the CUDA compute capability of the specific device in use. To coalesce our accesses, we store our background model as a structure of arrays rather than an array of structures as shown in Figure 31. In this Figure, the first subscript refers to the thread number and the second refers to the mode number, where only the first mode is completely shown since the rest follows the same pattern. In addition, here R, G, and B identify the RGB running sums, and C identifies the count of a mode in the background model. The figure shows that mode value (R, G, B, or C) accessed by the threads of a half warp are adjacent in a structure of arrays storage pattern, rather than an array of structures storage pattern. This allows memory accesses by the threads of a half warp to be coalesced.

Pinned memory for zero copy. Pinned memory in CUDA is used in certain situations for performance improvements because this memory cannot be swapped to disk as virtual memory on the CPU. Pinned memory is in mapped mode, when the GPU is required to access the CPU memory instead of copying data over to its own memory. Using mapped pinned memory in integrated GPUs is always a performance gain because the CPU and the GPU are sharing the same memory [42]. Mapped pinned memory returns a device pointer that is used by the GPU to access the CPU data from the memory, and so superfluous

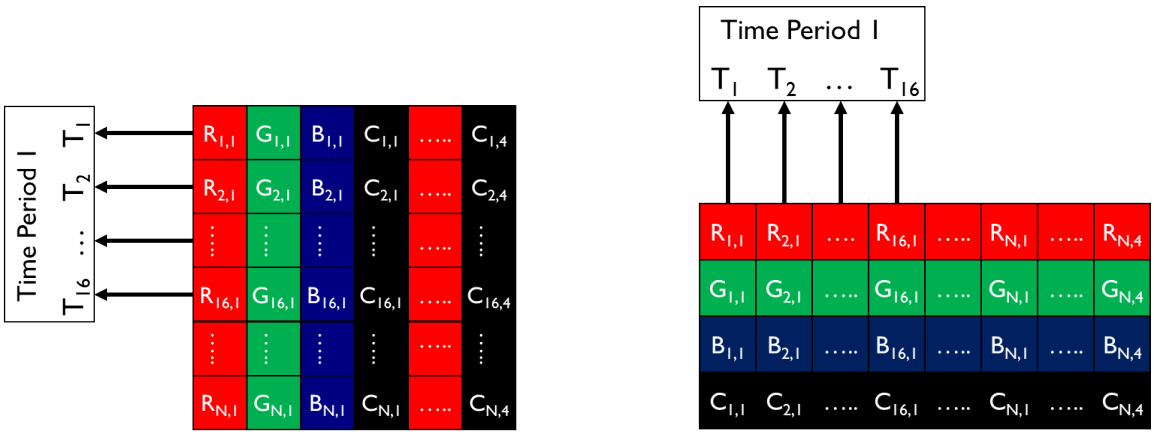


Figure 31: Un-coalesced array of structures (left), coalesced structure of arrays (right)

copying from the CPU to the GPU is avoided. We use mapped pinned memory both ways, i.e. reading new images and writing back the binary background subtraction result.

Floating point operations and vector data-type. The RGB and C values of a mode are stored as 32-bit integers by the MMM algorithm. The problem here is that the NVIDIA CUDA GPU, of compute capability 1.x, takes multiple instructions to complete 32-bit integer multiplications/divisions because of the lack of built in support which degrades the performance. To overcome this performance bottleneck, we type-cast the RGB sums to float while calculating the means. Moreover, since the RGB sums and C count are always used together, we store them in a packed int4 vector data-type of CUDA for improved performance. A performance improvement results because the 16 threads of a half warp, in a coalesced memory framework, now access the data in two 128-byte transactions rather than four 64-byte transactions.

Multiple pixels per thread. The final important optimization is specific to our algorithm: we assign multiple pixels to a single thread. In addition to employing data reuse, a common way to hide memory latency is to increase the occupancy of the device by launching a massive number of threads. Moreover, a less common way to achieve the same performance is to have multiple independent memory accesses in a single thread [43]. This

is possible in our algorithm since accesses for each pixel are independent of each other. Therefore, when we have multiple pixels per thread we do have those independent memory accesses.

The important question is where does this result in a performance improvement. The answer lies in the dynamic nature of our MMM algorithm where a background pixel can have many modes. A new image pixel can match any of those modes from 1 to the maximum number of modes K, at which point the pixel thread is done with its share of work in a single pixel per thread case. If multiple pixels are assigned per thread, the threads which finish early can move on to their next pixel and so on. There will be typically multiple such threads at each step depending on the nature of background at different points (dynamic or non-dynamic) in the scene. The end result is that a thread with multiple pixels will take less time than multiple threads each with a single pixel. This results in great performance improvement.

One important question is how much can we increase the number of pixels per thread. This is determined by the minimum device occupancy requirement in terms of blocks per SM since an increase in the number of pixels per thread results in a decrease in the number of threads and blocks. If we keep on increasing the number of pixels per thread, a point will be reached where the device will not be fully occupied and at that point performance will start to degrade. The number of pixels per thread should also be a power of two to ensure memory alignment for coalesced accesses. Furthermore, there should be block-level load-balancing on the device. For a highly dynamic background, as we increase the number of pixels per thread, we should continue to achieve improvement in speed until we reach the minimum occupancy limit, while speed saturation should reach earlier for less dynamic backgrounds.

4.2.3 Multi-layer background modeling TM3 implementation

We now consider the parallelization of the multilayer background model TM3. As in the case of MMM, the first step is to find the parallel work to be run as threads. Our object-based multi-layer background modeling algorithm involves two major processing tasks: at the pixel-level, which has a great deal of parallelism, and at the region level (which involves area density estimation for object layer formation and is predominantly sequential). Of the two, pixel level processing takes most of the time: 98%, and has the most inherent parallelism in that pixels can be processed independently of each other in this task. Keeping Amdahl's law in mind we have taken a hybrid approach that runs the pixel-level analysis on the GPU and the region-level analysis on the CPU, to achieve a high speed up overall.

We build our TM3 GPU kernel on top of the highly optimized MMM kernel discussed in the previous section. We have to make the number of modes fixed in the background model for TM3, rather than allowing to change dynamically for efficient memory access. The major change in TM3 from the MMM implementation is managing and optimizing memory usage in this hybrid CPU/GPU setting. This is challenging because in TM3 algorithm, both the GPU and the CPU access the background model. In the original MMM implementation only the GPU accessed the background model, and it resided in the GPU memory. However, in TM3, CPU also needs to access and update the background model while performing region-level analysis for the formation of object layers. In a discrete GPU case, CPU (host) and GPU (device) memories are separate, and data has to be copied back and forth over a PCI bus between CPU and GPU for information sharing. Therefore, in the case of a discrete GPU, one has to perform a time-consuming copy of the background model data from/to GPU to/from CPU while performing region-level analysis. However, an integrated GPU platform, like ours, gives an important advantage for the TM3 implementation as the CPU and the GPU share the memory. In an integrated GPU, a portion of memory is allocated for the GPU using system BIOS setting, which is called by the same name of device memory as in the discrete GPU case, but in reality it is

a portion of the same memory hardware used by the CPU. In addition, the GPU can also use the CPU memory directly, without copying, using *host-mapped* pinned memory. In the MMM implementation, we only use host-mapped pinned memory to our advantage by avoiding time-consuming copying of an image frame from/to the CPU memory to/from the GPU memory. In TM3, however, we allocate the background model using host-mapped pinned memory as well, instead of the dedicated device memory of GPU, so that both the CPU and GPU can access the model.

We allocate the memory as *write-combined* in addition to host-mapped, so that we can avoid the delay caused by the CPU cache hierarchy. The host-mapped write-combined pinned memory, on an integrated GPU, gives us as higher a performance on the GPU side as if we are accessing the device memory allocated for the GPU [44]. On the other hand, the CPU side reads becomes very slow (about 6x slower) because no caching is performed, and because there are no SSE4 instructions on our CPU to efficiently read such I/O mapped data[44]. However, the overall performance improves since the CPU accesses the background model only during region-level analysis, which is a small percentage (2%) of the total run time.

The region-level analysis forms object layers in the TM3 algorithm. This layer information also needs to be accessed by the GPU pixel-level analysis kernel for updating the occlusion reasoning counts. Therefore, the layer data structure is also allocated using write-combined host-mapped pinned memory, so that the GPU pixel-level kernel can directly access this information from the CPU memory for updating the occlusion reasoning counts. Since an object layer has many pixels, these counts are updated by multiple threads of the GPU requiring that atomic operations are used here. This will inevitably affect the performance, because atomic operation serializes the threads. In addition, to update these occlusion counts, the TM3 algorithm needs to access main memory for every mode of a pixel, unlike the MMM algorithm. This makes the multiple pixel per threads optimization ineffective on the GPU, since threads cannot go any further as main (global) memory is

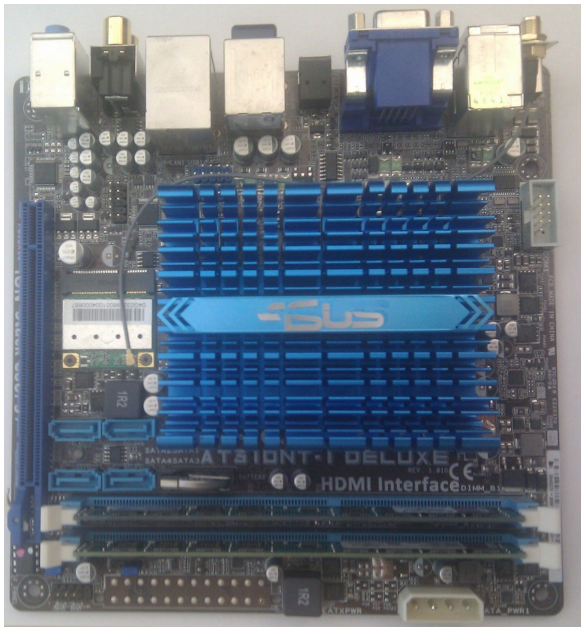


Figure 32: Asus AT3IONT-I NVIDIA ION GPU platform

accessed for every mode of a pixel. Despite these limitations which partly serializes our algorithm, we achieve a high (five times) speed up for TM3 on ION GPU platform over Atom CPU platform as described in the next section.

4.3 Experimental setup and results

Our integrated GPU platform is the Asus AT3IONT-I Deluxe, Figure 32. It has an NVIDIA ION GPU with 16 cores (2 SMs) running at 1.1GHz (shader clock speed) with TDP of 12 watts. The CUDA device compute capability is 1.1. In addition, the platform has an Intel Atom 330 Dual-Core CPU running at 1.6GHz with TDP of 8 watts. The memory shared by both the GPU and the CPU is 4 GB DDR3, out of which 512MB is allocated for the GPU device. The tested device memory bandwidth is 6.9GB/sec. We use three datasets, summarized in Figure 33, to test the performance on both GPU and CPU. The maximum gcc compiler optimizations option (O3) is used on the Atom CPU platform. The CUDA compute visual profiler and the CUDA occupancy calculator are used for the GPU performance analysis.

(a) Trees-HD[45]
287 frames-640x480 resolution

(b) Outdoors
400 frames-640x480 resolution

(c) Pets-S1L1V2[46]
221 frames-768x576 resolution

Figure 33: Datasets used to run MMM on ION GPU

4.3.1 MMM performance results

Figure 34 shows speed-ups compared to the implementation on a single core of the Atom CPU, after applying each of the GPU optimizations discussed for the three datasets. These optimizations are applied cumulatively, i.e. a later optimization is applied on top of all the previous optimizations. We achieve great performance improvement reaching nearly 20x. The overall performance improvement is less for Trees dataset because of the highly dynamic nature of the background, which causes more divergent branches reducing the effectiveness of coalesced memory accesses. On the other hand, as shown in Figure 35, the multiple pixels per thread PPT optimization performs best for the Trees dataset because of prevalence of dynamic background. For highly dynamic backgrounds, as we increase the number of pixels per thread, we continue to achieve improvement in speed until we reach the minimum occupancy limit, while speed saturation is reached earlier for less dynamic backgrounds. At 512, the performance goes down because the limit of minimum occupancy is crossed. The performance decreases the most at this point for the Pets dataset because there is also a block-level load imbalance on SMs. Table 5 gives a comparative summary of our results in frames per second on a single core of Intel Atom CPU verses the 16-core NVIDIA ION GPU. This result shows that we have achieved up to 392 fps for a full VGA frame on a low-power integrated GPU platform compared to 20 fps on a CPU platform of similar power specifications.

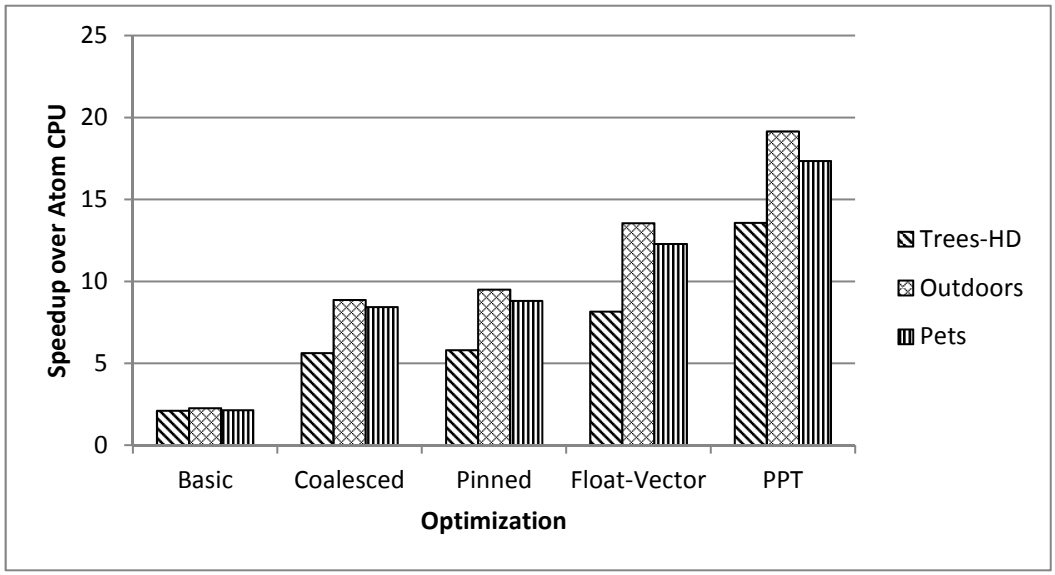


Figure 34: Speed ups over a single core of Atom CPU as a result of various performance optimizations, cumulatively applied left to right

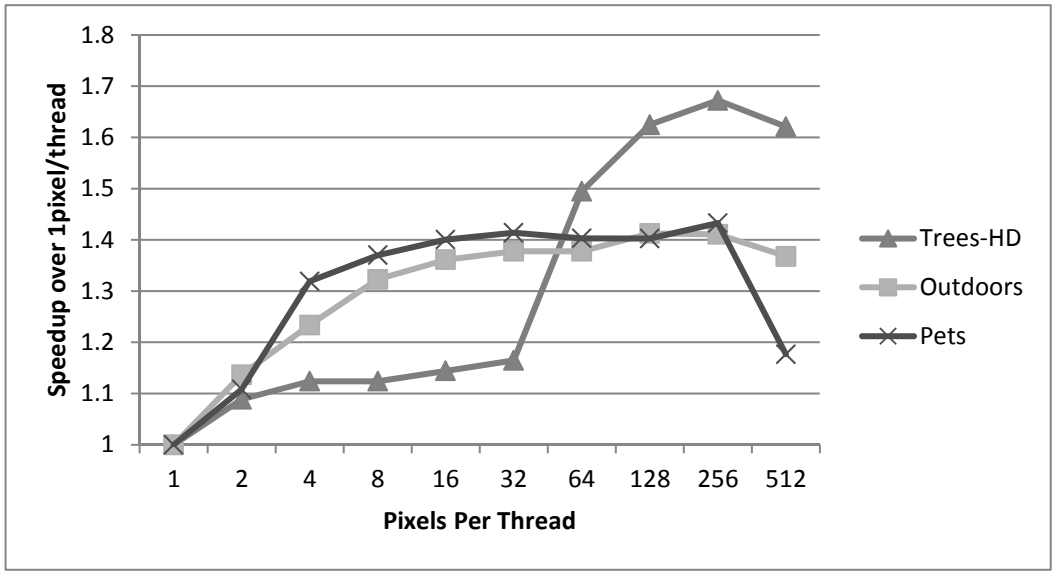


Figure 35: Speed ups for different number of pixels per thread implementations over a single pixel per thread implementation

Table 5: MMM on ION-GPU and Atom-CPU

Platform	Cores Used	TDP-watts	Speed-fps		
			Trees-HD	Outdoors	Pets
Atom-330	Single	8	18	20	14
ION-1	16	12	245	392	242

Figure 36 shows the performance comparison of our MMM GPU implementation with the GPU implementation [6] of an extended version of Gaussian mixture model EGMM [9] and the original Gaussian mixture model GMM [8]. We modified the EGMM implementation to use *host-mapped* pinned memory for fair comparison on our integrated GPU platform. Changing threads per block from 128 to 64 also significantly increased the speed of EGMM/GMM algorithms on the ION GPU. Our implementation runs up to 6x faster than these implementations. On the other hand, on a general-purpose CPU platform MMM is shown to run 4.23x faster than the GMM algorithm [20]. Our major gain over the EGMM/GMM implementation [6] is a result of the multiple pixel per thread optimization. Table 6 compares the speed in terms of frame rate of our implementation with the GMM GPU implementations. The table shows our gain in terms of speed while having comparable accuracy to GMM as described in section 2.2.

Table 6: MMM and GMM on NVIDIA ION

Algorithm	Speed-fps		
	Trees-HD	Outdoors	Pets
GMM	48	66	49
EGMM	51	73	52
MMM	245	392	242

Finally, we perform sensitivity and scaling analyses for our MMM implementation with respect to video frame size. Figure 37 shows the effect of video frame size on our algorithm performance by showing the actual and expected performance as we keep on reducing the video frame size by half, with original frame size of 640x480. We see that the gap between actual and expected performance becomes evident at/after 100x96, which is the point where we do not have enough pixels to hide the 400-800 cycles memory latency. In 100x96 case, we have 150 warps per SM considering 64 threads/block, and a single pixel per thread, which means only 600 cycles of memory latency can be hidden as we have no data reuse.

4.3.2 TM3 performance results

We now discuss the results for the multilayer background modeling TM3 algorithm. For testing our TM3 algorithm implementation, we use the Cars dataset of over 10K frames with 640x480 frame size and 15 frames per second. This is an outdoor parking lot scene in which multiple vehicles arrive/leave over a long period of time. This allows multiple object layers to be added and deleted throughout the course of time for more accurate performance measurement. We have fixed the number of modes to four in both the CPU and the GPU implementation. Increasing the number of modes results in more accuracy but less speed, and vice versa.

We achieve a high speed up of 5x over the Atom CPU platform for the TM3 algorithm. This is not as high as the MMM algorithm speed up, and in this section, we analyze why this is the case. Figures 38 and 39 show the performance results of the TM3 in comparison with the traditional background modeling MMM, and TM3-pixel algorithm [2] with no region-level analysis for object layer formation. Figure 38 shows that the TM3-pixel algorithm speed is approximately 0.7x of MMM both on Atom CPU and ION GPU. This is because it requires more memory accesses and processing mainly for the age and observability calculation of the pixels. The case is not the same for the full TM3 algorithm. The TM3 algorithm speed is 0.6x on CPU in comparison with the MMM, which is due to

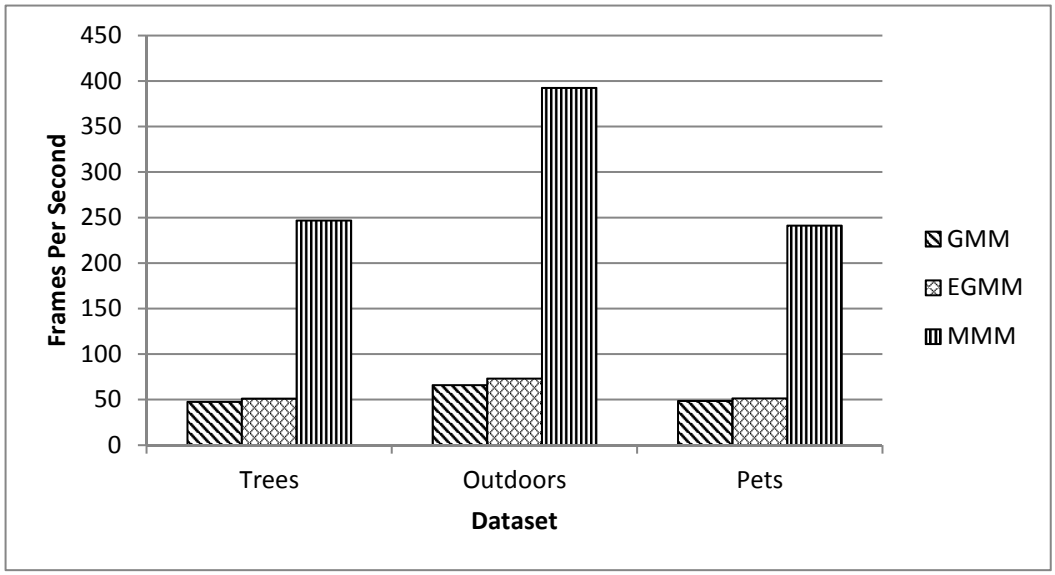


Figure 36: Frame rate comparison between MMM and GMM/EGMM on ION GPU: Speed up of 5-6x

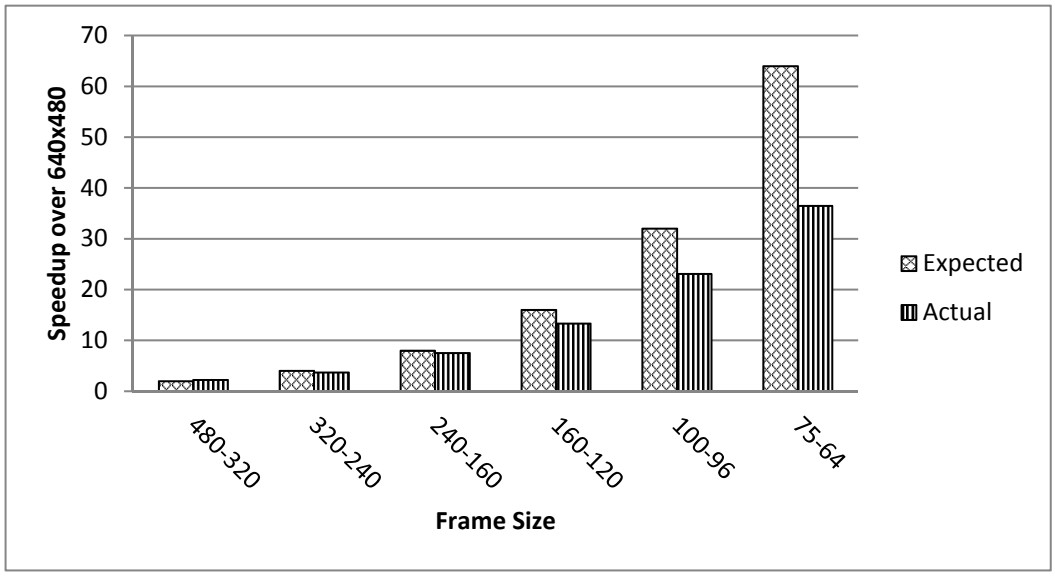


Figure 37: Speed ups over 640x480 frame size implementation as we decrease the frame size by half at each step

extra layer-formation processing. On GPU, however, it is 0.23x, in comparison with the MMM algorithm. The main reason for this degradation in performance is the access of main memory by the TM3 algorithm for every pixel mode for occlusion reasoning, which makes the prolific multiple pixels per thread optimization ineffective on the GPU. The other important reason is the atomic operations of occlusion reasoning counts on the GPU kernel which sequentializes the threads. The sequential region-level analysis code (2% of the total runtime), which runs on the CPU degrades the performance a little in accordance with Amdahl's law. The effect of these performance bottlenecks is shown in Figrue 39. The figure shows TM3 speed bottlenecks temporarily removed for testing (column 2-4), which results in higher fraction of the MMM speed for TM3 on ION GPU. The first and last column again shows TM3 speed in terms of a fraction of the MMM speed on ION and Atom respectively. Figure 40 shows the final frame rate achieved for TM3 algorithm on ION GPU in comparison with Atom CPU for the Cars dataset. Despite the limitations caused by object layer formation analysis, which partly serializes our TM3 algorithm, *we still achieve a high frame rate of 56 fps for full VGA frame size (640x480) on ION GPU platform compared to 11 fps on Atom CPU platform, which is a 5x speed up*.

4.4 Conclusion

Background modeling is a key initial step in many video surveillance applications. As more and more smart cameras are deployed for surveillance tasks across the globe, an efficient background modeling technique is required that balances accuracy, speed, and power. Due to its high parallel computational characteristics, robust adaptive background modeling has been implemented on GPUs with significant performance improvements over CPUs. However, these implementations are infeasible in embedded applications due to the high power ratings of the targeted general-purpose GPU platforms. This chapter focuses on how data and thread-level parallelism is exploited and memory access patterns are optimized to target an adaptive background modeling algorithm MMM to a low-power GPU with TDP

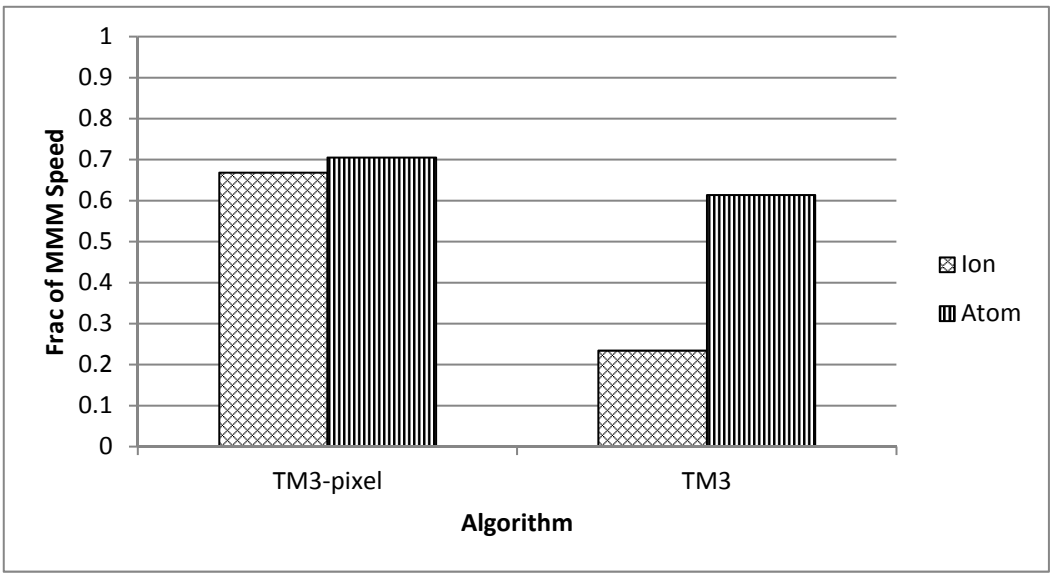


Figure 38: TM3-pixel and TM3 speed comparison vs. MMM on ION GPU and Atom CPU

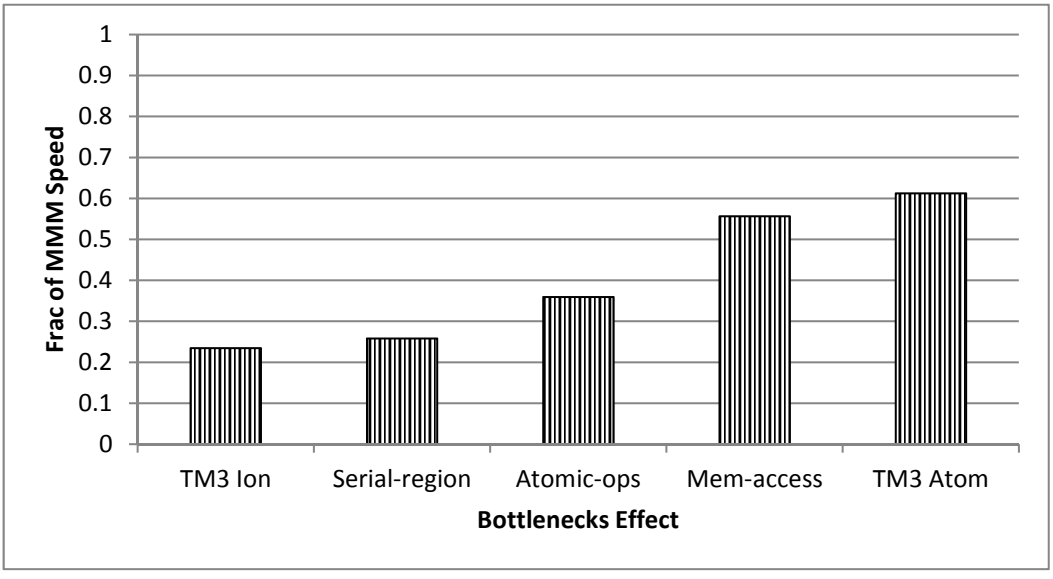


Figure 39: TM3 speed bottlenecks temporarily removed for testing (column 2-4) results in higher fraction of the MMM speed for TM3 on ION GPU, the first & last column again show TM3 speed as a fraction of MMM on ION and Atom respectively from the previous figure

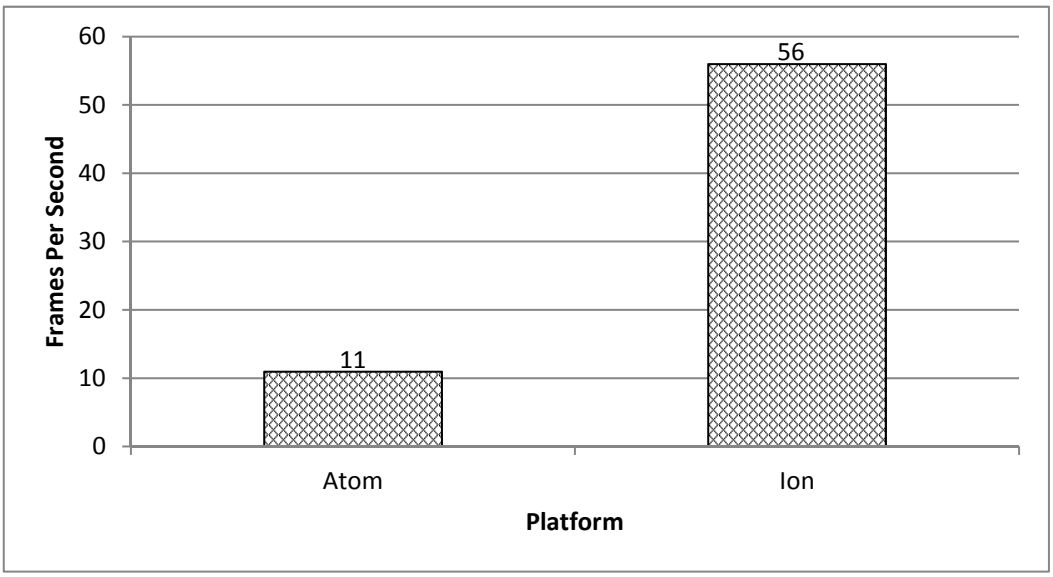


Figure 40: Frame rate of TM3 algorithm on ION GPU and single core of Atom CPU: Speed up of 5x

of only 12 watts. The algorithm has comparable accuracy with the GMM algorithm, but less computational and memory cost. We have achieved a frame rate of 392fps with a full VGA resolution (640x480) frame on a low-power integrated GPU NVIDIA ION. This is a 20X speed-up of the MMM algorithm on the GPU compared to an embedded CPU platform Intel Atom of comparable TDP. Moreover, our GPU implementation of MMM outperforms the GPU implementation of GMM by achieving a speed up of 6x. In addition, for the multi-layer background modeling algorithm TM3, the speed up achieved is 5x.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The dissertation has developed an efficient object-based multi-layer background modeling approach to distinguish among *midground* objects, the objects whose existence occurs over varying time scales between the extremes of short-term ephemeral appearances (foreground) and long-term stationary persistences (background). The dissertation consists of three contributions.

In the **first contribution**, a multilayer background modeling technique, temporal multimodal mean TM3, is presented for video surveillance. The technique temporally model a scene in which there are multiple interacting midground objects occurring at different time scales. The approach correctly models the scenes with long-term occlusions and ghost objects as compared to the multilayer pixel-based background modeling approaches. TM3 technique allows us to represent a scene, with multiple midground objects entering, leaving, and occluding each other at different points in time. This leads to richer information about temporal properties of a scene than traditional foreground/background segmentation. The information includes when a particular object arrived or left the scene, and the occlusion relationships among different objects while they are in the scene.

The TM3 technique builds on a low-cost MMM background modeling technique [20] which makes it suitable for embedded, real-time platforms. It adds approximately twice the latency and storage requirements of MMM. However, these costs remain relatively low given that the MMM algorithm runs 4x faster than the widely used Gaussian mixture model (GMM) technique [8] on a general-purpose CPU platform, while exhibiting comparable performance in accuracy.

The multi-layer (and two-layer) background modeling techniques that model objects that have become stationary will incorrectly detect a new object if an existing midground

or background object is displaced. The **second contribution** presents a novel spatio-temporal reasoning mechanism, spatio-temporal multimodal mean STM3, based on multi-layer background modeling and objects appearances to conserve the state of moved objects in a scene. The algorithm is an extension of our temporal multimodal mean TM3 algorithm to spatial analysis, adding only a little computational and memory overhead over TM3. STM3 algorithm, consistently models midground/background objects upon partial/full change of position, and maintains conservation of existing objects, only removing them once they leave the scene. An important result of this algorithm is that it avoids false alarms of new objects when existing objects are displaced in the scene.

Background modeling techniques for embedded computer vision applications must balance accuracy, speed, and power. Due to its inherent parallelism, robust adaptive background modeling, such as GMM, has been implemented on GPUs with significant performance improvements over CPUs. However, these implementations are infeasible in embedded applications due to the high power ratings, in the range of 100 watts, of the targeted general-purpose NVIDIA GeForce GPU platforms. The **third contribution** focuses on how data and thread-level parallelism is exploited and memory access patterns are optimized to target a low-cost robust adaptive background modeling algorithm MMM to a low-power GPU NVIDIA ION with TDP of only 12 watts. The algorithm has comparable accuracy with the GMM algorithm, but less computational cost. Accelerating this technique is also important because it is at the core of our spatio-temporal multi-layer background modeling algorithms TM3/STM3. We have achieved a frame rate of 392fps with a full VGA resolution (640x480) frame on the NVIDIA ION GPU. This is a 20X speed-up of the MMM algorithm on the GPU compared to the embedded CPU platform Intel Atom of comparable TDP. Moreover, our GPU implementation of MMM outperforms the GPU implementation of GMM by achieving a speed up of 6x. Subsequently, we extended the MMM GPU implementation to the multi-layer background modeling algorithm TM3, and achieved 5x speed up over the single core Atom CPU implementation.

Currently in our multi-layer background modeling approach TM3, new objects, whose layers are yet to be formed, are required to be non-overlapping at the time of their object layer formation; otherwise, they will be grouped into a single layer. In addition, overlapping objects are required to be dissimilar in color since our occlusion reasoning mechanism uses color features to differentiate among the object layers. Future work can focus on extending our algorithm to handle overlap among new objects at the time of object layer formation, and to handle similar color overlapping objects using texture and shape information in addition to color appearance models.

In our spatio-temporal multimodal mean (STM3) approach, objects with similar appearance models can cause false matches because of the simple nature of the color histogram object identification method. Future work can explore enhancing our technique with richer object identification methods than the color histogram. In addition, parallelizing the STM3 algorithm, on the NVIDIA ION GPU platform, for further performance improvement is another future work direction.

Being able to extract multiple temporal midground layers in a scene makes possible many future applications in video surveillance. This would aid humans, particularly in the task that is difficult for the human visual system of detecting the objects that gradually become stationary. In a semi-autonomous setting, for example, our algorithms can provide a human operator with a single frame summary of the ordered object occurrences that happened in the past in a long video sequence. This would be a tedious and error-prone task for a person who is continuously monitoring a scene. In addition, the spatio-temporal scene analysis mechanism presented in this dissertation can provide a part of a low-cost early vision engine, on top of which high-level computer vision applications, such as video summarization and scene understanding, can efficiently run on future embedded platforms.

REFERENCES

[1] K. Kim, D. Harwood, and L. S. Davis, "Background updating for visual surveillance," in *International Symposium on Visual Computing (ISVC)*, pp. 337–346, 2005.

[2] S. Azmat, L. Wills, and S. Wills, "Temporal multimodal mean," in *2012 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, pp. 73–76, 2012.

[3] "4.2 million cameras? that's what we've been told but new research paints a different picture." CCTV Image, official publication of the CCTV user group, Winter 2011.

[4] N. Jacobs and R. Pless, "Time scales in video surveillance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 8, pp. 1106–1113, 2008.

[5] T. Fabián and J. Gaura, "Parallel implementation of recursive background modeling technique in CUDA for tracking moving objects in video traffic surveillance." `http://www.fi.muni.cz/memics07/2008/pres/fabian_cuda.pdf`, 2008.

[6] V. Pham, P. Vo, V. T. Hung, *et al.*, "GPU implementation of extended gaussian mixture model for background subtraction," in *2010 IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future*, pp. 1–4, 2010.

[7] M. Poremba, Y. Xie, and M. Wolf, "Accelerating adaptive background subtraction with GPU and CBEA architecture," in *2010 IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 305–310, 2010.

[8] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 22, pp. 747–757, 2000.

[9] Z. Zivkovic and F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognition Letters*, vol. 27, no. 7, pp. 773–780, 2006.

[10] T. Horprasert, D. Harwood, and L. S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," in *IEEE International Conferecne on Computer Vision (ICCV)*, pp. 1–19, 1999.

[11] "NI 177X series smart cameras." http://sine.ni.com/ds/app/doc/p/id/ds-370/lang/en, last accessed, June 2014.

[12] "Matrox IRIS GT smart camera." http://www.matrox.com/imaging/en/products/smart_cameras/iris_gt, last accessed, June 2014.

[13] "Sony XCISX100C-XP smart camera." http://pro.sony.com/bbsc/ssr/cat-camerasindustrial/cat-cismartcameras/product-XCISX100C%2FXP/, last accessed, June 2014.

[14] "Intellio ILC-BL series smart cameras." http://www.videoline-tvcc.com/upload/pdf/ILC-BL_series_datasheet_ENG.pdf, last accessed, June 2014.

[15] S. Azmat, L. Wills, and S. Wills, "Multilayer background modeling under occlusions," *Machine Vision and Applications (MVA)*, Apr 2014.

[16] S. Azmat, L. Wills, and S. Wills, "Spatio-temporal multimodal mean," in *2014 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, pp. 81–84, 2014.

[17] S. Azmat, L. Wills, and S. Wills, "Accelerating adaptive background modeling on low-power integrated GPUs," in *International Workshop on Embedded Multicore Systems (ICPP-EMS 2012), held in conjunction with the 41st IEEE International Conference on Parallel Processing*, pp. 568–573, 2012.

[18] T. Yang, Q. Pan, J. Li, and S. Li, "Real-time multiple objects tracking with occlusion handling in dynamic scenes," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 970–975, 2005.

[19] B. Valentine, S. Apewokin, L. Wills, S. Wills, and A. Gentile, "Midground object detection in real world video scenes," in *IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 517–522, 2007.

[20] S. Apewokin, B. Valentine, D. Forsthoefel, L. Wills, S. Wills, and A. Gentile, "Embedded real-time surveillance using multimodal mean background modeling," in *Embedded Computer Vision, (B. Kisacanin, S. Bhattacharyya and S. Chai, eds.)*, pp. 163–175, Springer, 2010.

[21] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in *The Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV)*, pp. 255–261, 1999.

[22] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 19, no. 7, pp. 780–785, 1997.

[23] N. J. McFarlane and C. P. Schofield, "Segmentation and tracking of piglets in images," *Machine Vision and Applications (MVA)*, vol. 8, no. 3, pp. 187–193, 1995.

[24] S. Jabri, Z. Duric, H. Wechsler, and A. Rosenfeld, "Detection and location of people in video images using adaptive fusion of color and edge information," in *15th International Conference on Pattern Recognition*, pp. 627–630, 2000.

[25] S. C. Sen-Ching and C. Kamath, "Robust techniques for background subtraction in urban traffic video," in *Electronic Imaging 2004*, pp. 881–892, International Society for Optics and Photonics, 2004.

[26] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *European Conference on Computer Vision (ECCV)*, pp. 751–767, 2000.

[27] T. Bouwmans, F. El Baf, B. Vachon, *et al.*, "Statistical background modeling for foreground detection: A survey," *Handbook of Pattern Recognition and Computer Vision*, pp. 181–199, 2010.

[28] R. Mathew, Z. Yu, and J. Zhang, "Detecting new stable objects in surveillance video," in *IEEE 7th Workshop on Multimedia Signal Processing*, pp. 1–4, 2005.

[29] J. Connell, A. W. Senior, A. Hampapur, Y.-L. Tian, L. Brown, and S. Pankanti, "Detection and tracking in the IBM peoplevision system," in *2004 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1403–1406, 2004.

[30] P. Spagnolo, A. Caroppo, M. Leo, T. Martiriggiano, and T. D'Orazio, "An abandoned/removed object detection algorithm and its evaluation on PETS datasets," in *IEEE International Conference on Video and Signal Based Surveillance (AVSS)*, 2006.

[31] Y.-l. Tian, R. Feris, A. Hampapur, *et al.*, "Real-time detection of abandoned and removed objects in complex environments," in *The Eighth International Workshop on Visual Surveillance (VS2008)*, 2008.

[32] S. Ferrando, G. Gera, and C. Regazzoni, "Classification of unattended and stolen objects in video surveillance system," in *IEEE International Conference on Video and Signal Based Surveillance (AVSS)*, 2006.

[33] S. Khan and M. Shah, "Tracking people in presence of occlusion," in *Asian Conference on Computer Vision (ACCV)*, pp. 1132–1137, 2000.

[34] V. Papadourakis and A. Argyros, "Multiple objects tracking in the presence of long-term occlusions," *Computer Vision and Image Understanding*, vol. 114, no. 7, pp. 835–846, 2010.

[35] H. Tao, H. S. Sawhney, and R. Kumar, "Object tracking with bayesian estimation of dynamic layer representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, no. 1, pp. 75–89, 2002.

[36] F. Porikli, "Detection of temporarily static regions by processing video at different frame rates," in *IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 236–241, 2007.

[37] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, "Real-time foreground-background segmentation using codebook model," *Real-time Imaging*, vol. 11, no. 3, pp. 172–185, 2005.

[38] H. Fujiyoshi and T. Kanade, "Layered detection for multiple overlapping objects," *IEICE Transactions on Information and Systems*, vol. 87, no. 12, pp. 2821–2827, 2004.

[39] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of Calcutta Mathematical Society*, vol. 35, no. 1, pp. 99–109, 1943.

[40] P. Carr, "GPU accelerated multimodal background subtraction," in *Digital Image Computing: Techniques and Applications (DICTA)*, pp. 279–286, 2008.

[41] NVIDIA Corporation, "NVIDIA compute unified device architecture C programming guide v6.0." `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`, last accessed, June 2014.

[42] NVIDIA Corporation, "NVIDIA compute unified device architecture C best practices guide v6.0." http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf, last accessed, June 2014.

[43] V. Volkov, "Better performance at lower occupancy." http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf, *Presentation in GPU Technology Conference, 2010*.

[44] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Pearson Education, 2013.

[45] "Test images for wallflower paper." Available online: http://research.microsoft.com/en-us/um/people/jckrumm/wallflower/testimages.htm, 1999.

[46] "PETS 2009 benchmark data." Available online: http://www.cvg.rdg.ac.uk/PETS2009/a.html#s1l1, 2009.