LEAP SEGMENTATION IN MOBILE IMAGE AND VIDEO ANALYSIS

A Dissertation
Presented to
The Academic Faculty

by

Dana Forsthoefel

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

Georgia Institute of Technology

December 2013

LEAP SEGMENTATION IN MOBILE IMAGE AND VIDEO ANALYSIS

Approved by:

Dr. Linda M. Wills, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. D. Scott Wills, Co-Advisor (Posthumous)
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Doug Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Jongman Kim
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Andrea Thomaz
College of Computing
Georgia Institute of Technology

Date Approved: August 23, 2013

In memory of Dr. Scott Wills, an exceptional teacher, mentor, and friend.

ACKNOWLEDGEMENTS

First, I wish to thank my advisor, Dr. Linda Wills, for taking me in as an undergraduate researcher and working tirelessly to turn me into the PhD candidate I am today. I would never have made it through this experience without her patience, guidance, encouragement, and friendship. Working for the past eight years in her lab, she has acted as mentor, advisor, and sometimes, surrogate mom. Thank you Linda, for sticking with me this long and for making this a wonderful experience I will never forget. I also wish to thank my late advisor Dr. Scott Wills for all of our research conversations. I will especially remember those times when we disagreed, because I loved the challenge of convincing you of my correctness… though you invariably proved to be right in the end. I like to think, had we had more time, I would have won one someday. Scott, I didn't know I'd lose you so soon, but I'm grateful for the time I had to learn from you.

I also appreciate Dr. Jongman Kim, Dr. Aaron Lanterman, Dr. Doug Blough, Dr. Andrea Thomaz, and Dr. Anthony Yezzi for serving on my committee and providing their helpful insights and feedback. Their input has helped to strengthen my research for which I am grateful. I would like to thank Dr. Doug Blough and Dr. John Peatman for their continued support and guidance during my career at Georgia Tech, ever since my first classes with them as an undergraduate. I would also like to thank my MoVES lab colleagues over the years, including Ryan Bales, Shoaib Azmat, Qianao Ju, Brian Valentine, and Senyo Apewokin. Against all odds, they made those long hours in the lab enjoyable and I thank them for all their helpful discussions and contributions to my research.

I wish to thank my family, especially my Mom and Dad for all their unwavering support during my academic career. Though in the beginning they had no idea what an electrical and computer engineering degree entailed, they never hesitated to support me through almost a decade in pursuit of one. Mom and Dad, I wouldn't have made it here without your encouragement and support. Thank you for being the best parents anyone could ask for.

Most of all, I wish to thank my fiancé Adam. Through all the sleepless nights, the frantic revisions, the hours in the lab, and the frustration at the hands of this doctoral program, he maintained absolute certainty that I would complete my dissertation before I turned 80. Adam, without you, I would be lost. I could never have done this without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

λ	Leap Segmentation adjacency parameter
ε	Leap segmentation equivalence parameter
α	Leap segmentation minimum size parameter
AOF	Aggregate Objective Function
CL	Chroma-Luminance
EDISON	Edge Detection and Image Segmentation
EGBIS	Efficient Graph-Based Image Segmentation
FPS	Frames Per Second
HSI	Hue, Saturation, Intensity
MCD	Maximum Component Difference
PRI	Probabilistic Rand Index
RGB	Red, Green, Blue
SAD	Sum of Absolute Differences
TDP	Thermal Design Power

SUMMARY

As demand for real-time image processing increases, the need to improve the efficiency of image processing systems is growing. The process of image segmentation is often used in preprocessing stages of computer vision systems to reduce image data and increase processing efficiency. This dissertation introduces a novel image segmentation approach known as *leap segmentation*, which applies a flexible definition of adjacency to allow groupings of pixels into segments which need not be spatially contiguous and thus can more accurately correspond to large surfaces in the scene. Experiments show that leap segmentation correctly preserves an average of 20% more original scene pixels than traditional approaches, while using the same number of segments, and significantly improves execution performance (executing 10x - 15x faster than leading approaches). Further, leap segmentation is shown to improve the efficiency of a high-level vision application for scene layout analysis within 3D scene reconstruction.

The benefits of applying image segmentation in preprocessing are not limited to single-frame image processing. Segmentation is also often applied in the preprocessing stages of video analysis applications. In the second contribution of this dissertation, the fast, single-frame leap segmentation approach is extended into the temporal domain to develop a highly-efficient method for multiple-frame segmentation, called *video leap segmentation*. This approach is evaluated for use on mobile platforms where processing speed is critical using moving-camera traffic sequences captured on busy, multi-lane highways. Video leap segmentation accurately tracks segments across temporal bounds, maintaining temporal coherence between the input sequence frames. It is shown that

video leap segmentation can be applied with high accuracy to the task of salient segment transformation detection for alerting drivers to important scene changes that may affect future steering decisions.

Finally, while research efforts in the field of image segmentation have often recognized the need for efficient implementations for real-time processing, many of today's leading image segmentation approaches exhibit processing times which exceed their camera frame periods, making them infeasible for use in real-time applications. The third research contribution of this dissertation focuses on developing fast implementations of the single-frame leap segmentation approach for use on both single-core and multi-core platforms as well as on both high-performance and resource-constrained systems. While the design of leap segmentation lends itself to efficient implementations, the efficiency achieved by this algorithm, as in any algorithm, is can be improved with careful implementation optimizations. The leap segmentation approach is analyzed in detail and highly optimized implementations of the approach are presented with in-depth studies, ranging from storage considerations to realizing parallel processing potential. The final implementations of leap segmentation for both serial and parallel platforms are shown to achieve real-time frame rates even when processing very high resolution input images.

Leap segmentation's accuracy and speed make it a highly competitive alternative to today's leading segmentation approaches for modern, real-time computer vision systems.

CHAPTER 1

INTRODUCTION

Over the past decade, the pervasiveness of cameras in almost all areas of modern life has created a growing need for efficient image analysis and understanding techniques. Camera use is ubiquitous:

- in "smart" cell phones [73] for image capture and minor image editing,
- in factories [87] for real-time monitoring and inspection of products,
- on streets [16] for catching traffic violations and illegal parking,
- in cars [38], [81] for improving highway safety,
- in hospital rooms [82] for remote monitoring of patient vital signs.

One of the more prevalent uses of cameras today is in video surveillance to monitor areas in combating crime. Surveillance cameras have become common in airports, businesses, and homes to identify and track suspicious behavior. Self-guided cameras have also been developed for use in combat environments for automated reporting of combat situations [29]. Often, surveillance cameras operate on mobile, resource-constrained systems, requiring image analysis methods to rapidly process images for conclusive identification of significant activity in real-time (e.g. [6], [8], [9]).

Employing vision processing in intelligent vehicle systems has also grown extensively over the past several years. Cameras have been placed in mobile vehicles for use in driver aid and alert systems [38], [48], and, in some developing car designs, in automatic steering systems [81]. Computer vision systems can be used to analyze traffic

scenes and alert drivers of potentially dangerous events as they occur in real time, thus increasing the safety of road ways. Prototype intelligent vehicle systems have already been demonstrated on highways across the country [54]. Due to their operating environment, intelligent vehicle systems are inherently mobile, requiring vision applications to be both accurate and efficient in their implementation for successful operation in this resource-constrained, real-time environment.

All these applications are driving ground-breaking research in embedded image processing to develop novel methods for image analysis and understanding. An important early step in most of these vision applications is *image segmentation*, which is critical in reducing image data and enabling efficient execution. Image segmentation separates an image into homogeneous, perceptually significant regions of pixels that can each be processed as a group. Segmentation is often used in vision applications to preprocess pixel data prior to image analysis methods, such as edge detection, stereo matching, and object tracking. For many segmentation approaches the primary objective is to accurately detect whole object positions and boundaries in an image, a process referred to as *under-segmentation*. However, under-segmentation often causes enormous loss of image detail as pixels are grouped to form overly-large segments. While these identified primitives are often useful for high-level visual processing tasks, there exists a separate class of high-level vision applications which instead require an *over-segmentation* of the input image. In over-segmentation, segments correspond not with whole image objects, but with homogeneous regions of pixels *within* image objects that can be similarly processed because of their affinity. High-level vision applications incorporate image over-segmentation techniques into their preprocessing stages primarily to reduce the image

data which must be processed by the vision system, thus improving overall execution efficiency. However, in many such applications, the resulting image blurring and loss of detail can be detrimental to the accuracy of the functioning system. For example, if one were to segment a photograph before performing facial recognition, the human facial features must remain discernible or the recognition accuracy would suffer. For a highway surveillance system to identify specific vehicles, the vehicle license plates must remain legible after segmentation. An intelligent vehicle vision system may require street sign and road marking information to autonomously make steering decisions. While small, these details cannot be blurred during segmentation without reducing the accuracy of the high-level steering system, which would have life-threatening ramifications.

Developing a New Segmentation Approach

The goal of this research is to develop a novel over-segmentation approach for the preprocessing stages of real-time vision applications that require salient-feature preservation to achieve accuracy. This specific class of vision applications is one for which traditional approaches, both under-segmentation and even over-segmentation, have proven inadequate due to loss of detail and blurring. In particular, this application class requires the following:

1. ***Efficiency***. Demand is steadily growing for image processing on portable, low power devices. Efficient implementations of computer-vision techniques are needed for computing on mobile systems such as cell phones and digital cameras. To meet this demand, an embedded-computing trend is emerging in the field of computer vision [50]. Recently, many traditional image-processing techniques have been revisited to develop faster, more efficient implementations that can

function in an embedded environment. Real-time, high-level vision applications apply image over-segmentation during preprocessing to improve their overall execution efficiency. The over-segmentation technique must therefore be efficient in its own execution to avoid counteracting any system performance improvement. Meeting performance requirements in this area has proven challenging. Existing segmentation approaches often fail to meet real-time processing standards, particularly when applied to high-resolution images. Research in novel over-segmentation approaches is needed that focuses on highly-efficient preprocessing of input image data for real-time applications.

2. ***Salient-Feature Preservation***. The question of how much detail should be preserved during segmentation depends on the target application. Many current high-level applications that apply image-segmentation techniques in preprocessing expect a certain degree of image blurring and detail loss when applying existing segmentation algorithms in their preprocessing stages. This detail loss is expected and useful in those applications which favor treating whole image objects as single entities, rather than several pieces forming a whole (segments). However, scene-interpretation applications (such as those for natural scene reconstruction, human facial recognition, and highway scene understanding) require a higher degree of salient-feature preservation during segmentation preprocessing for reliable performance. Excessive blurring of input images or loss of salient detail (e.g. street sign lettering, lane markings, or facial features) during segment formation is detrimental. A lack of detail preservation in preprocessing would greatly reduce the utility of these vision applications. The

specific class of applications of interest in this research requires a segmentation approach that balances salient-detail preservation with the reduction of image data.

In this dissertation, a novel approach, called *leap segmentation*, is developed that focuses on this task of improving segmentation preprocessing both in efficiency and feature preservation (Forsthoefel et al.) [34]. This approach efficiently transforms raw pixel data into feature preserving, palletized, color-similar and illumination-similar regions for use in preprocessing to facilitate performance improvements in high-level vision systems.

Leap segmentation is so named because the approach allows grouping of adjacent but non-neighboring pixel values. Pixels can "leap" across segmentation boundaries to join nearby chromatic and luminance-similar segments. This technique preserves salient scene details during segmentation as shown in Figure 1. The leap-segmented image on the right strongly resembles the original image on the left with little loss in detail. However, 154,401 pixels in the original image are now replaced by 132 regions, which

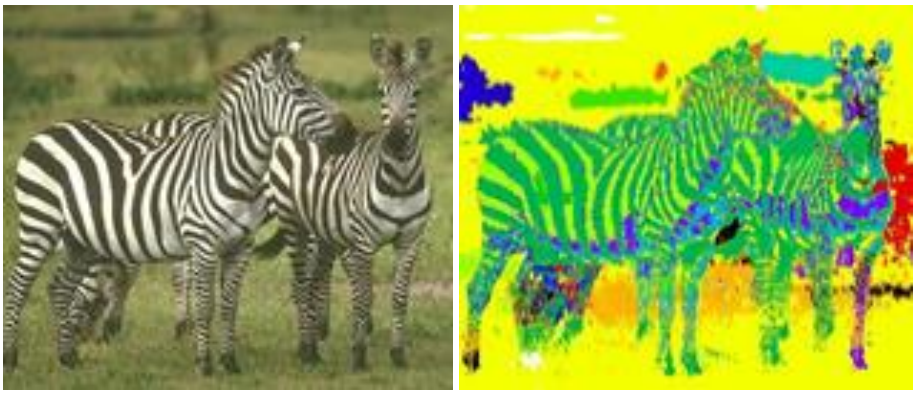
(a) **(b)**

Figure 1. Leap segmentation output example (Planes). (a) Original image 481x321 pixels. (b) Image segmented using leap segmentation with 132 segments.

can be more efficiently processed.

Segmentation algorithms typically partition an image into regions, often referred to as “superpixels” [66], which can be processed together because of their affinity (based on color, texture, intensity, etc.). Leap segmentation removes unimportant image features and minute pixel variations (such as texture and minor chromatic variations), while better preserving fine detail (such as vehicle license plate lettering or highway scene markings), than existing segmentation approaches. By relaxing strict adjacency constraints, leap segmentation produces larger groupings of similar pixels. In practice, this novel approach is able to produce perceptually correct groupings of non-contiguous regions such as stripes, as shown in Figure 2. Traditional segmentation approaches often needlessly segment each stripe into a separate segment, an inefficient use of resources.

Similarly, traditional segmentation approaches often have difficulty processing high-variation or porous regions, such as trees and grass. A traditional segmentation approach which builds only contiguous segments will attempt to segment each tree leaf



(a) **(b)**

Figure 2. Leap segmentation groups together non-contiguous segments such as stripes. (a) Original image. (b) Colorized representation of the image segmented using leap segmentation.

as a separate segment, a tremendous waste of resources. This method of processing also burdens high-level applications with the need to perform additional steps to group these leaf segments into a "tree" object. The leap segmentation approach groups pixels in high-variation regions such as sparse vegetation together within a specified adjacency neighborhood into a small number of segments representing the color information in these regions, thus eliminating the need for additional steps in high-level vision applications and reducing the resources required to represent the segmented image scene.

Admittedly, existing segmentation approaches could be redesigned to allow the grouping of non-contiguous pixels into their segmentations. However, such adjustments to these algorithms would dramatically increase their complexity, making them computationally infeasible for real-time applications. Leap segmentation is designed specifically to produce such output and thus is capable of doing so with reduced time and storage resources.

Multiple-Frame (Video) Segmentation

In addition to the challenges of single-frame image segmentation (efficiency and salient-feature preservation), this dissertation explores ways of meeting segmentation challenges in multiple-frame (video) applications. Video segmentation has been applied in many vision applications including video compression and video indexing and retrieval [39]. Many video segmentation techniques are designed to operate off-line, requiring all frames in the input video sequence as input [41]. Since future frames must be known, these approaches are not feasible for real-time applications where only current and past frames are available. A few on-line approaches exist in the literature, but they are limited in accuracy. Meeting both high accuracy and high efficiency requirements in

video segmentation is a challenging task, and further research in this field is needed to meet real-time processing standards.

Parallelizing Leap Segmentation

Finally, this dissertation explores the potential for parallelizing leap segmentation on multi-core hardware platforms. Modern demand for real-time image processing algorithms has inspired several research efforts in fast, multi-core image segmentation. However, contemporary approaches often require specialized hardware and achieve only moderate frame rates on low-resolution images and exhibit extremely slow frame rates when applied to high resolution images [1], [43], [58]. Real-time, multi-core implementations have not been fully realized. There remains much room for improvement to achieve real-time (>25 fps) image segmentation executions on commercially-available CPUs with multiple processing cores that do not require special hardware.

1.1. Problem Statement and Research Contributions

The goal of this research is to provide vision applications with a faster, more accurate image segmentation approach that is robust enough to be used in both single and multiple-frame scene analysis and efficient enough for embedded and mobile platforms. This goal will be achieved through the following contributions:

1. A novel, single-frame segmentation approach, called *leap segmentation*, is presented that efficiently reduces and restructures image data into regions while preserving the salient features in the image that are needed in scene analysis applications (Forsthoefel et al.) [31], (Forsthoefel et al.) [34].

2. The single-frame leap segmentation algorithm is extended to efficiently process video—multiple, consecutive frames in time—while maintaining region boundary continuity between image frames. Temporal analysis of the multiple-frame leap segmentation algorithm is performed to evaluate segmentation stability over time in video sequences from moving camera traffic scenes (Forsthoefel et al.) [32], (Forsthoefel et al.) [33].

3. Single-frame leap segmentation is parallelized in a multi-core implementation of the approach that achieves real-time frame rates when segmenting high-resolution input images on embedded, mobile platforms (Forsthoefel et al.) [35].

These three contributions to the image segmentation field are evaluated further in the following subsections.

1.1.1. Contribution 1: Single-Frame Leap Segmentation

The first contribution of this dissertation introduces *leap segmentation*, a highly-efficient, non-contiguous segmentation approach designed to reduce and restructure image information while accurately preserving salient details in the scene. Leap segmentation builds a new image representation, replacing individual pixel data with a map-indexed palette of chroma-luminance-similar regions that are adjacent but not necessarily contiguous. High-level algorithms can process this compact image representation for efficient execution. Leap segmentation is evaluated using both the Berkeley Segmentation Dataset and new, publicly available datasets that target real-time vision applications, such as those used in intelligent vehicle systems. In experiments, leap segmentation demonstrates high region-assignment accuracy and, compared to other

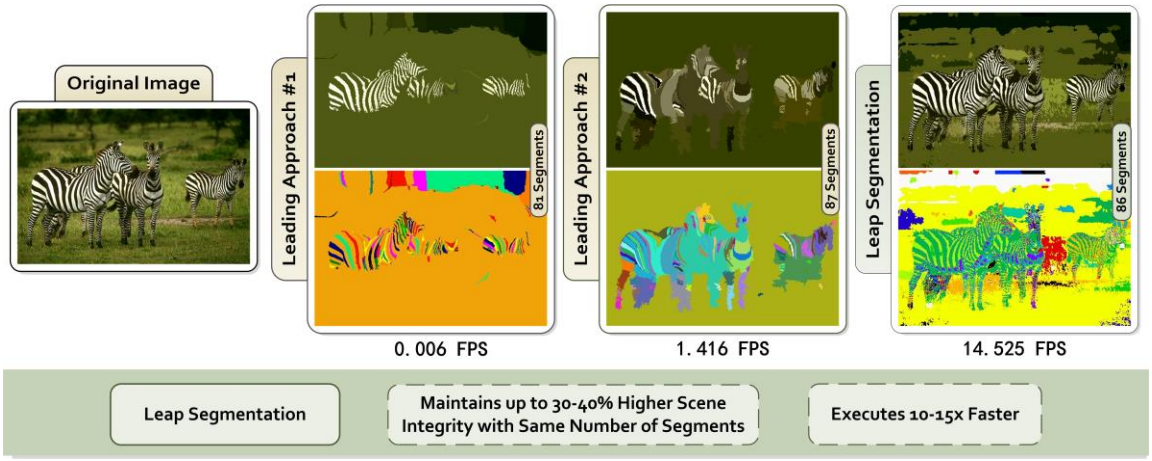


Figure 3. Graphical summary of the first dissertation contribution: single-frame leap segmentation.

approaches, preserves a higher level of scene integrity (up to 30-40% higher) using a given storage resource (Forsthoefel et al.) [34].

In addition, it is demonstrated that this novel segmentation technique can significantly improve scene layout analysis within 3D scene reconstruction (Forsthoefel et al.) [31]. Leap segmentation can be used in preprocessing to form homogeneous regions of pixels that need not be spatially contiguous and can thus more accurately correspond to larger surfaces in the scene. In this way, leap segmentation provides more meaningful spatial support to scene layout analysis methods. A detailed evaluation of the leap segmentation approach and comparisons with related, existing segmentation methods are provided. The presented implementation is computationally efficient, exhibiting execution time improvements of 10x - 15x over traditional approaches. The diagram in Figure 3 provides a full, graphical summary of this contribution.

1.1.2. Contribution 2: Leap Segmentation in Video Analysis

Multiple-frame (video) segmentation is an important step in many video analysis applications for identifying and tracking specific features as they move through a scene. In a mobile, resource-constrained environment, such as an intelligent vehicle system, video segmentation can be used to reduce image information and increase processing efficiency for high-level scene understanding applications. The second contribution of this dissertation introduces *video leap segmentation*, a highly efficient multiple-frame segmentation approach for use on embedded and mobile platforms where processing speed is critical. This novel video segmentation method is demonstrated to successfully track segments across spatial and temporal bounds, generating fast, stable segmentations of images from moving-camera video sequences (Forsthoefel et al.) [33]. Video leap segmentation is applied to the task of salient segment transformation detection for alerting potential drivers of critical scene changes that may affect steering decisions. Trial results demonstrate that video leap segmentation enables coarse detection of salient region transformations in traffic scenes, correctly detecting 80% of salient segment transformations in trial scenes with less than 5% false positives. Reducing high-level processing to salient areas using this approach can significantly improve the processing efficiency of scene interpretation applications in intelligent vehicle systems. The diagram in Figure 4 provides a graphical summary of this contribution.

A supplementary contribution of this research is the development of a publicly available image dataset called the *GTTraffic Dataset* (Forsthoefel et al.) [32]. GTTraffic is a collection of moving-camera traffic sequences captured at Georgia Tech for use in vision evaluation experiments. The sequences contain fast-moving traffic events, such as

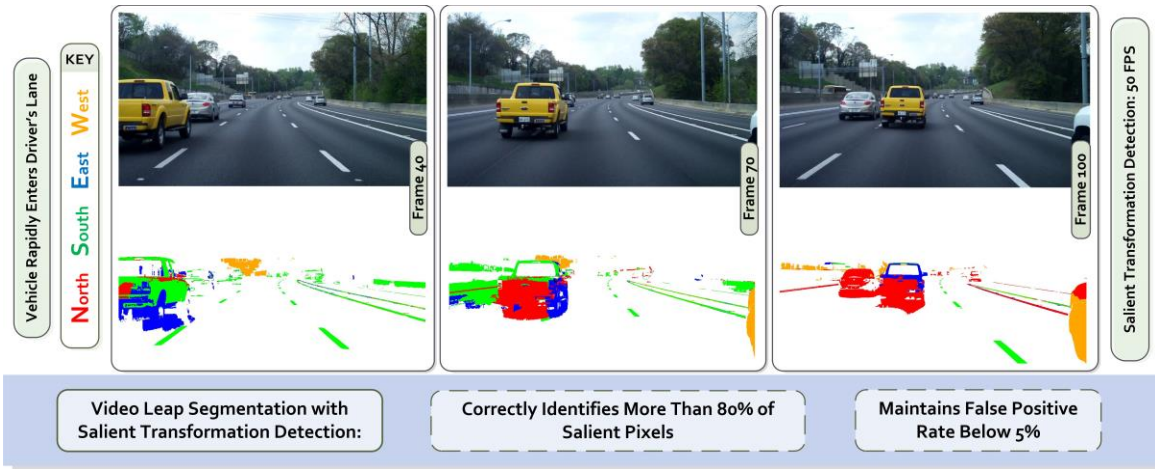


Figure 4. Graphical summary of the second dissertation contribution: video leap segmentation with salient transformation detection identifies salient foreground objects when everything is moving, including the camera. Color indicates segment direction relative to the camera.

vehicles quickly swerving into the driver's lane. These sequences are made publicly available as part of this research to motivate and evaluate vision-based approaches to improving highway safety.

1.1.3. Contribution 3: Embedded, Multi-Core Leap Segmentation

Existing segmentation approaches often fail to meet real-time processing standards and exhibit extremely slow frame rates when applied to high resolution images. The third contribution of this dissertation first presents a highly optimized serial implementation of the leap segmentation approach. This serial implementation is demonstrated to achieve frame rates exceeding that of the state-of-the art (it segments more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images). Leap segmentation is then analyzed further for its inherent parallelism and restructured for use on a multi-core system to achieve additional speed-up (Forsthoefel et al.) [35]. On a multi-core, mobile processing system with four threads, multi-core leap

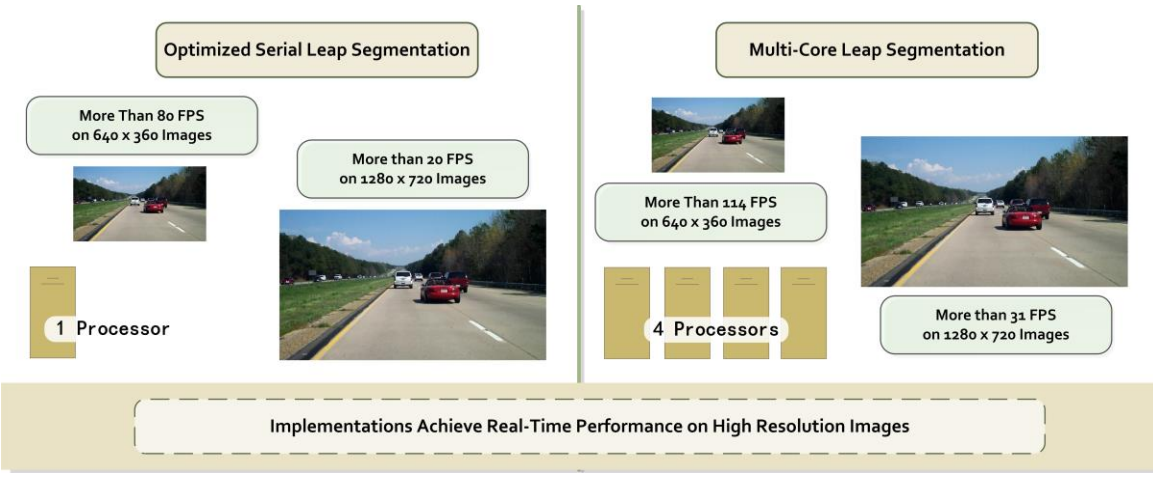


Figure 5. Graphical summary of the third dissertation contribution: embedded, multi-core leap segmentation.

segmentation achieves frame rates of over 114 fps on 640x360 images and more than 31 fps on 1280x720 images, thus easily exceeding real-time processing standards. The diagram in Figure 5 graphically summarizes this contribution.

1.2. Summary of Results

The key results of this dissertation are as follows:

- An efficient, non-contiguous segmentation approach designed to reduce and restructure image information while accurately preserving salient details in the scene is presented (Forsthoefel et al.) [34]. This *leap segmentation* approach demonstrates high region assignment accuracy and, compared to other approaches, preserves a higher level of scene integrity (up to 30-40% higher) using a given storage resource. The approach is also computationally efficient, exhibiting execution time improvements of 10x - 15x over traditional approaches.

- The leap segmentation approach is comprehensively evaluated in a 3D scene reconstruction application (Forsthoefel et al.) [31]. Leap segmentation can be used in preprocessing to form perceptually significant regions of pixels that need not be spatially contiguous and can thus more accurately correspond to larger surfaces in the scene. In this way, leap segmentation provides more meaningful spatial support to scene layout analysis methods.

- A highly efficient multiple-frame segmentation approach for use on embedded and mobile platforms where processing speed is critical is presented (Forsthoefel et al.) [33]. This novel *video leap segmentation* method is demonstrated to successfully track segments across spatial and temporal bounds, generating fast, stable segmentations of images from captured moving-camera video sequences.

- Video leap segmentation is applied to the task of salient segment transformation detection for alerting potential drivers of critical scene changes that may affect steering decisions (Forsthoefel et al.) [33]. Trial results demonstrate that with little added computation, video leap segmentation enables course detection of salient region transformations in traffic scenes, correctly detecting 80% of pixels in salient segment transformations with less than 5% false positives.

- A publicly available dataset of moving-camera traffic sequences (GTTraffic) collected at Georgia Tech is developed and presented for use in vision evaluation experiments (Forsthoefel et al.) [32].

- A highly optimized serial implementation of single-frame leap segmentation is given in (Forsthoefel et al.) [35]. This serial implementation is demonstrated to

achieve frame rates of more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images, far exceeding the state-of-the art in execution.

- A parallel implementation of the single-frame leap segmentation algorithm is developed for use on embedded, multi-core platforms (Forsthoefel et al.) [35]. On a multi-core, mobile processing system with 4 threads, this multi-core leap segmentation implementation achieves frame rates of over 114 fps on 640x360 images and more than 31 fps on 1280x720 images, easily meeting real-time processing standards.

1.3. Overview of Content

This dissertation is organized as follows. Chapter 2 outlines the novel, leap segmentation approach and presents the results of experiments that test leap segmentation using both classical and newly developed accuracy metrics. This chapter also presents comparisons with other well-known segmentation approaches and evaluates the use of leap segmentation in the preprocessing of a high-level 3D reconstruction application. In Chapter 3, leap segmentation is extended into a real-time, video segmentation approach. Video leap segmentation is then applied in the application of salient segment transformation detection in a mobile, intelligent vehicle vision application. A detailed analysis of video leap segmentation performance in this context is given. Chapter 4 outlines two highly efficient implementations of the leap segmentation approach for use on single-core and multi-core platforms and gives detailed performance analyses on both high-performance and resource-constrained hardware. Chapter 5 concludes this dissertation and discusses future work.

CHAPTER 2

SINGLE-FRAME LEAP SEGMENTATION

2.1. Introduction

Image segmentation is the process of separating an image into perceptually significant regions of pixels that can each be processed as a group. Segmentation algorithms have been widely researched and are used in many vision applications to preprocess pixel data prior to image analysis methods, such as edge detection, stereo matching, and object tracking. Separating an image into segments of pixels for processing can significantly reduce the amount of computational resources needed to analyze an image in a high-level vision system. This reduction of resource usage has the potential to increase algorithmic processing speed.

This chapter presents a highly-efficient image segmentation approach, called *leap segmentation* (Forsthoefel et al.) [34], that focuses on the task of improving segmentation preprocessing both in efficiency and feature preservation to facilitate performance improvements in high-level vision systems. A primary objective for most existing segmentation approaches is to accurately detect object positions and boundaries in an image. Leap segmentation has a different emphasis: to efficiently transform raw pixel data into feature preserving, palletized, color-similar and illumination-similar regions for improved scene analysis. Rather than process each image pixel individually, vision applications can use leap segmentation to preprocess image pixels into groups that can be processed more rapidly. An example of leap segmentation output is shown in Figure 6.

(a) **(b)**

Figure 6. Leap segmentation output example (Polo). (a) Original image 481x321 pixels. (b) Image segmented using leap segmentation (Forsthoefel et al.) [34] with 180 segments.

Vision applications rely on preprocessing segmentations to accurately maintain important image features while reducing the data in the image. In addition, many applications require their segmentation preprocessing steps to perform quickly and efficiently. Leap segmentation is applicable to a broad range of segmentation tasks and is especially appropriate for embedded and mobile platforms where processing speed is critical. Traditional image segmentation approaches often blend or remove small image details when building contiguous regions, and processing time often exceeds the camera frame period. Leap segmentation better preserves salient features while achieving a significant improvement (> 10x the state of the art) in execution performance.

In this chapter, leap segmentation is evaluated using images from the well-known Berkeley Segmentation Dataset. Its use in real-time applications, such as intelligent-vehicle vision systems where detailed feature preservation is vital, is also evaluated. In experiments, leap segmentation demonstrates high region-assignment accuracy and, compared to other approaches, preserves a higher level of scene integrity using a given storage resource.

To further demonstrate the benefits of leap segmentation, it is used to improve the performance of a high-level vision task for 3D scene reconstruction (Forsthoefel et al.) [31]. Surface-layout analysis applications for 3D scene reconstruction often evaluate complex geometric cues over large regions to determine the orientations of large surfaces within the scene. These regions can contain contiguous pixels, such as those in solid walls, or non-contiguous pixels such as those in tree leaves or shrubs. Traditional segmentation approaches partition homogeneous, non-contiguous pixels into many smaller segments that must then be further analyzed and grouped by the high-level layout application. Leap segmentation can form homogeneous regions of pixels that need not be spatially contiguous and can thus more accurately correspond to larger surfaces in the scene. In this way, leap segmentation provides more meaningful spatial support to scene layout analysis methods, significantly improving processing efficiency.

This chapter is organized as follows. Related work in image segmentation is summarized in Section 2.2. Section 2.3 presents the novel, leap segmentation approach. Section 2.4 discusses the fast leap segmentation implementation. Section 2.5 shows a detailed parameter evaluation and sensitivity analysis. Section 2.6 compares the accuracy and efficiency of leap segmentation with other well-known segmentation approaches when applied to intelligent vehicle highway scenes and on diverse Berkeley Segmentation Dataset images. Section 2.7 evaluates leap segmentation using several well-known, classical accuracy metrics. Section 2.8 describes a popular high-level vision application for image labeling and reconstruction and demonstrates the benefits of applying leap segmentation to this task. Experiments show that leap segmentation correctly maintains an average of 20% more original scene pixels than traditional

approaches while using the same number of segments and significantly improving execution speed (>10x faster than existing approaches). Section 2.9 concludes this chapter and discusses future work.

2.2. Related Work

Image segmentation has been explored in many previous research efforts, resulting in several broad classes of algorithms, including region-based, feature-space clustering, and graph-based segmentation. Early image segmentation approaches typically use *region-based segmentation*. These region-growing [2], [19] and split-and-merge [46] methods are conceptually simple. They typically rely heavily on input threshold parameters and they often have trouble processing regions of high variation [61]. The watershed approach [77] is a popular example of region-based segmentation. In general, watershed transformation-based algorithms [10], [61] are fast and efficient with time complexities linear in the number of pixels [67]. However, they are sensitive to noise and highly-textured regions and often require extra, costly preprocessing steps to produce useful gradient input [78].

Finally, the jump connection approach [68] is a region-grouping approach recently applied in color segmentation with mathematical morphology operators [5]. While it closely resembles leap segmentation in name, the two approaches are very different in operation. The jump connection approach assesses jumps in color space between neighboring image pixels and, unlike leap segmentation, the jump connection approach requires segments to be spatially contiguous.

Segmentation methods that use *feature-space clustering* attempt to find modes (clusters) in a distribution by using each image pixel's features as sampled data from the

distribution's probability density function. The k-means clustering method [52], while simple and well-known, relies heavily on correct user input of cluster count and initial cluster center placements to produce a good segmentation [47]. Mixture of Gaussians (MoG) clustering with Expectation Maximization (EM) [26] has been used in preprocessing for recent applications [11], [18]. However, EM calculations are vulnerable to becoming stuck in local minima and can be slow to converge [85]. The MoG with EM approach also relies heavily on its input parameters, such as an accurate estimate of cluster count, to provide a useful solution.

The mean-shift technique [21], [22] also uses feature-space clustering. According to Pantofaru and Hebert [63], output segmentations from mean-shift correspond well to human perception. A disadvantage is its sensitivity to parameter change and the necessity for input parameter tuning to obtain good segmentations [86]. In addition, mean-shift suffers from being computationally expensive making it too slow for real-time applications. This is due in part to the expensive sliding-window approach it applies to image pixels during processing. Several techniques for improving mean-shift have been proposed [17], [20], [37], [80]. For example, Christodias et al. [20] proposed combining mean-shift with edge detection to increase segmentation accuracy in EDISON. However, there is still room for improvement as these algorithms require on the order of minutes to process one second of video [65].

In *graph-based segmentation* an image is represented as a weighted, undirected graph. Graph-based segmentation based on minimum cuts was first introduced by Wu and Leahy [84]. Shi and Malik [71] then introduced the normalized cut (NC) criterion to avoid the bias for partitioning undersized segments that plagued Wu and Leahy's earlier

approach. The NC algorithm requires few input parameters from the user when compared to mean-shift [86]. However, NC is expensive to run and is too slow to be used in real-time applications; finding the minimum NC based on Shi and Malik's proposed criterion is an NP-hard problem [30]. They present methods to approximate the calculation but these methods still prove computationally intensive. Several improvements to the NC approach have been proposed [53], [60] such as adding a boundary detector to reduce clutter and enhance segmentation performance. Cour et al. [24] focus on the parallelization of the existing normalized cuts approach for speed gain and propose an efficient multiscale variant of the normalized cuts approach that runs in linear time. However, these algorithms are still many times too slow for use in real-time applications, requiring at least several seconds to process a single frame [24].

Segmentation by weighted aggregation (SWA) [69] is a recent multiscale approach that reduces the normalized cut minimization problem using algebraic multigrids [15]. SWA preserves image boundaries more accurately in output segmentations and is more efficient than the original NC approach, possessing linear time complexity in the number of input image pixels. Despite these improvements, the SWA approach and a recently proposed improvement known as the probabilistic aggregation approach (PA) [4] which eliminates user-defined parameter reliance, are still slow, requiring tens of seconds to process a single image frame [28].

A popular graph-based segmentation technique, EGBIS [30], is considered to be state of the art in computational efficiency [28], [65]. It uses pair-wise component comparisons to segment an image in $O(mlogm)$ time, where m is the number of graph edges. A drawback to this method is its sensitivity to its input parameter k and its

tendency to create small, unneeded regions at the borders of valid image segments. In addition, the graph cuts segmentation approach [13], [14] is a popular graph-based method that uses Markov random fields [40]. However, this technique is primarily applied to binary segmentation, which is outside the scope of this research.

In the next section presents the novel leap segmentation technique. The leap segmentation algorithm is first defined and then evaluated for efficiency and accuracy performance using images from publicly available segmentation datasets. In this evaluation, leap segmentation performance and segmentation results are compared to two widely known segmentation approaches: a mean-shift segmentation approach (EDISON) and a graph-based segmentation approach (EGBIS).

2.3. Leap Segmentation Algorithm

The leap segmentation approach (Forsthoefel et al.) [34] identifies pixels that are related by adjacency within a specified neighborhood constraint and by a given chroma-luminance affinity metric. The reflexive, symmetric, transitive closure of these pixel relations provides equivalence groupings of adjacent, but not necessarily contiguous, pixels that are similar in chromaticity and luminance. The final segmentation includes each such grouping that satisfies a minimum size constraint requiring its area to be greater than a minimum-size threshold α.

In particular, the equivalence relation *region-equivalent* is defined to capture the relationship between all pixels in the same segment. It is the reflexive, symmetric, transitive closure of the binary relation *adjacent-matches* between pairs of pixels. Pixel P_1 adjacent-matches P_2 iff

a.) P_1 and P_2 are *CL-similar* (chroma-luminance affinity defined below) and

b.) P_1 and P_2 are *adjacent* within a specified neighborhood (not necessarily nearest neighbors).

2.3.1. Chroma-Luminance Affinity

Two pixels are *CL-similar* if their chroma-luminance difference is within a given threshold, ε. The measure of difference depends on the image color model (e.g., YCrCb, HSI, etc.). While luminance and chromaticity participate in the relation, they need not be orthogonally represented in the color model. In the leap segmentation implementation, described in Section 2.4, a red-green-blue component (RGB) color model is used to eliminate translation time. The CL-similar relation is defined using the maximum component difference (MCD): P_1 and P_2 are CL-similar iff

$$\max\begin{pmatrix}|R_1 - R_2|\\ |G_1 - G_2|\\ |B_1 - B_2|\end{pmatrix} \leq \varepsilon \ . \qquad \textbf{(1)}$$

2.3.2. Adjacency

While existing segmentation algorithms require member pixels to be spatially contiguous, leap segmentation allows member pixels to be separated by a pixel adjacency parameter, λ. For a given pixel P, the neighborhood of P, $n(P)$, is defined as all pixels within a $\lambda x \lambda$ square window centered around P. Figure 7 shows examples with λ=1 and λ=2. Two pixels P_1 and P_2 are adjacent iff $P_1 \in n(P_2)$ equivalently $P_2 \in n(P_1)$. P_1 and P_2 need not be nearest neighbors.

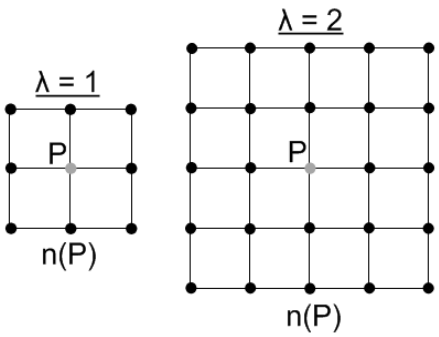


Figure 7. Definition of the neighborhood of *P*, *n(P)*, for λ = {1, 2}.

2.3.3. Region Equivalence

Region equivalence, which relates all pixels grouped into the same segment, is the reflexive, symmetric, transitive closure of the adjacent-matches relation. Pixels that are region-equivalent (i.e., in the same segment) are not required to be directly connected with immediate neighbors or even to be reachable through a chain of contiguous pixels. For example, in Figure 8, multiple contiguous regions (on left) are within a $\lambda x \lambda$ neighborhood and are grouped as a single segment *B*. A diagonal occlusion (on right) does not fragment segment *A* into two segments. This allows segments to span large regions of an image by connecting pixels through multiple "leaps" over other segments in the image with the restriction that no leap can be greater than λ.

Traditional image segmentation approaches could, potentially, be redesigned to allow the grouping of non-contiguous pixels into their segmentations. However, such adjustments to these algorithms cause dramatic increases in complexity. For example, the popular graph-based EGBIS approach [30] can be adjusted to include edges between non-adjacent pixels. However, this would require an exponential increase in the number of edges of the manipulated graph, in turn causing a marked decrease in the approach's

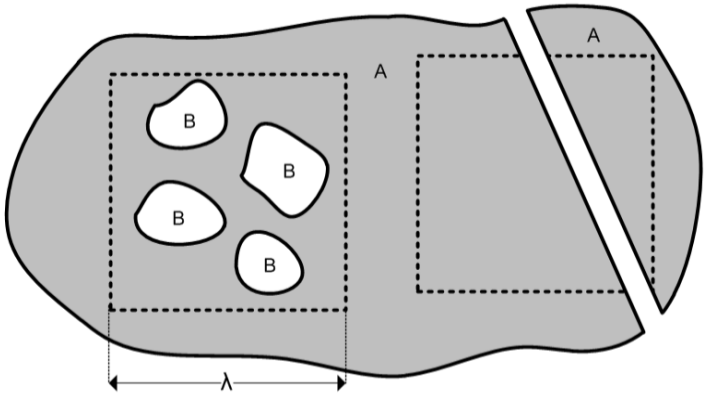


Figure 8. The leap segmentation adjacency definition allows more flexibility, eliminating noise (left) and occlusion (right) problems.

execution performance. Conversely, the innovative leap segmentation approach is designed specifically to produce such non-contiguous segment output and thus is capable of doing so with reduced time and resources.

2.4. Leap Segmentation Implementation

This section presents a fast and resource-efficient implementation of the leap segmentation algorithm. The workflow is shown in Figure 9.

2.4.1. Segmentation Constraints

To begin, the input image is discretized using the adjacency parameter, λ, by dividing it into non-overlapping λxλ square regions called tiles. Each tile is then scanned using the CL-similar constraint to locate candidate regions in each tile. If a pixel is CL-similar to pixels within an existing region, it is added to that region. Otherwise, it forms a new candidate region.

Pixels within a region contribute their component values to a ratiometric mean via component sums and a pixel count, shown in Figure 10. Each scanned pixel in a region is

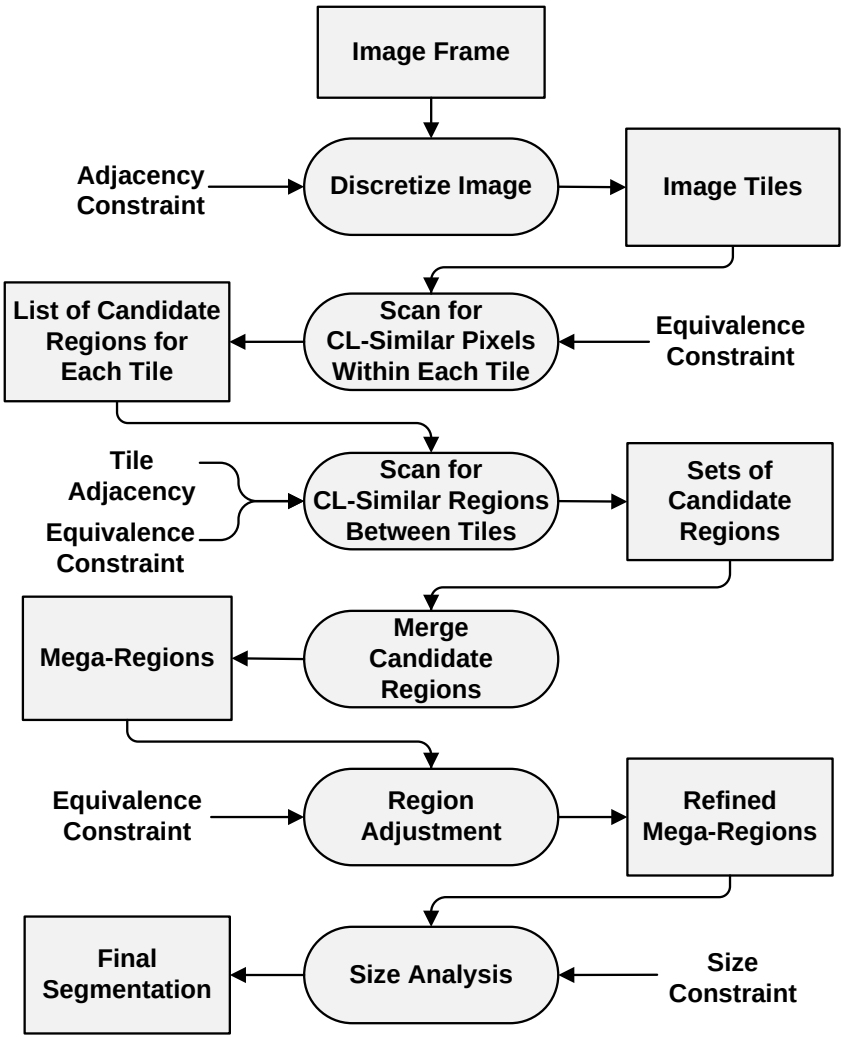


Figure 9. Workflow of resource-efficient leap segmentation algorithm.

compared to the mean component values (e.g. R, G, and B) of each candidate region. After identifying candidate regions within each tile, these regions are compared between neighboring, contiguous tiles. Regions whose mean component values satisfy the CL-similar relation are merged into a mega-region. This process continues until a final set of candidate mega-regions are identified. At this point, all ratiometric component means are locked to fixed component averages that no longer depend on member pixels.

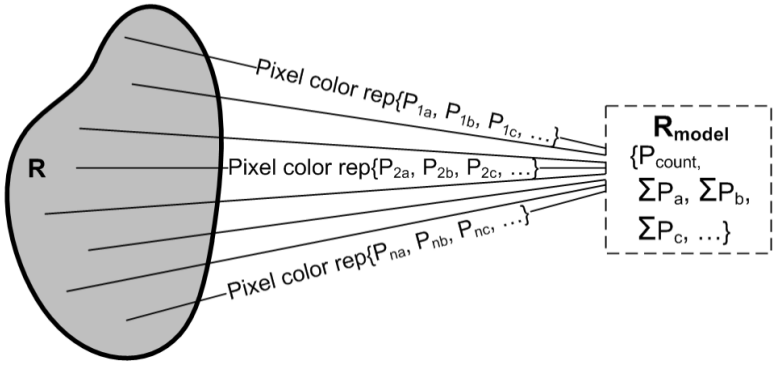


Figure 10. Efficient storage of region member-pixel information.

2.4.2. Region Adjustment and Size Analysis

When a pixel joins a candidate region, it adds its component values to the region's pixel component sums. Certain scene features such as large, slowly changing gradients can cause region component means to drift, occasionally leaving some member pixels outside of the CL-similar bounds.

This is corrected in a post-process region-adjustment step. Pixels are scanned for incorrect assignments in region membership. If a large number of incorrectly assigned pixels are identified, a new mega-region is created. The effect of region adjustment is examined in Section 2.6.4. This step also applies the minimum-size constraint to mega-regions, appropriately assimilating small regions to nearby mega-regions based on spatial and color similarities. The resulting mega-region list becomes the final segmentation.

2.5. Parameter Variation and Analysis

Leap segmentation input parameters include an adjacency parameter λ, an equivalence threshold ε, and a minimum size threshold α. The optimal parameter choice is determined by evaluating accuracy and compression objective functions across a

diverse collection of datasets. In this parameter assessment, both quantitative assessment and qualitative assessment are considered. While the optimization is performed primarily through the minimization of quantitative objective functions (e.g., number of segments), qualitative assessments (e.g., appropriate scene feature preservation), are also used to select the best parameters. In this section, evaluation metrics are defined, an optimal parameter set is presented, and parameter variation sensitivity analysis is explored.

2.5.1. Objective Functions

In this evaluation, two quantitative objective functions are used to assess compression and accuracy performance. The first metric, *number of segments* assesses image compression. One goal of leap segmentation is to transform pixel data into a much smaller number of similar regions that are more easily processed. The number of segments produced by an algorithm is a measure of how well it meets this objective. However, used alone, pursuit of compression would result in an undesirable loss of salient image features.

The second metric, *nonmatching pixel percentage* assesses segmentation accuracy. It measures the percentage of image pixels in the segmentation output that are not CL-similar to their original image color. Calculation of the nonmatching pixel percentage is shown in Equations 2-4. The equivalence function E applies the CL-similar relation (Equation 1) to assess pixel affinity. P_{NM} is the number of pixels in the final segmentation which are not CL-similar within the matching threshold τ to their original image ($\tau = 30$ was used in all experiments) and P_{TOTAL} is the total number of pixels in the image. P_{ORIG} holds the original input image, and P_{SEG} holds the pixels in the output segmentation.

$$E(P_A, P_B, \tau) = \begin{cases} 1, & \text{if } P_A, P_B \text{ are } \textit{CL-Similar} \\ 0, & \text{otherwise} \end{cases}, \quad \textbf{(2)}$$

$$P_{NM} = P_{TOTAL} - \left[\sum_i E(P_{ORIG}[i], P_{SEG}[i], \tau) \right], \quad \textbf{(3)}$$

$$Nonmatching\ \% = 100 \times \frac{P_{NM}}{P_{TOTAL}}. \quad \textbf{(4)}$$

A high accuracy image segmentation result achieves a low nonmatching pixel percentage, indicating that a small number of pixels have been assigned to a region color that is significantly different from their original color. This metric is a good measure of the preservation of scene integrity during the segmentation process.

Alternative quantitative metrics of image quality include mean squared error loss (MSE) and other cumulative pixel error measures. However, leap segmentation strives to preserve the maximum number of pixels in the original image, rather than assess the magnitude of distortion of disrupted pixels. *Qualitative* assessment is also used to adjust parameters near the quantitative optimum. Inspection of segmentation output reveals small adjustments of the parameters that improve the perseveration of important scene features. However these adjustments must benefit the process across a wide range of scene collections.

This section assesses the sensitivity of the algorithm parameters to scene composition, chromaticity, and illumination, to evaluate its applicability to a wide range of different scenes. For each parameter variation experiment, both the cumulative nonmatching pixel percentage and the cumulative number of segments are evaluated and compared using eight different scene collections, each containing 300 images. These collections include the Berkeley Segmentation Dataset [55], [56] (see Figure 11 for sample images) and seven mobile camera sequence collections captured at Georgia Tech

as part of the GTTraffic dataset [32] (discussed in Section 3.4). When computing the cumulative nonmatching pixel percentage and cumulative number of segments, all eight datasets are evaluated separately, each generating an average objective function value over each frame in the collection. The cumulative nonmatching pixel percentage and cumulative number of segments are the sum of the average values in each of the eight collections. The dataset scene diversity tests the generality of parameter values.

Figure 11. Sample images from the Berkeley segmentation dataset [55], [56] for use in segmentation evaluation experiments.

Table 1

Leap Segmentation Parameter Variation

	Adjacency	Equivalence	Size
Symbol	λ	ε	α
Range of Values	2 to 32	2 to 32	10 to 90
Optimal Value	8	20	50

Each leap segmentation parameter is varied across a wide range of values, shown in Table 1, to generate approximately 720 parameter combinations for evaluation. This bracketing assures that the best parameters are captured. Both accuracy and compression objective functions contribute to overall segmentation quality. While the relative benefits of each function are dependent on the application, an aggregate objective function (AOF) is useful in optimizing segmentation parameters. The AOF is defined as the normalized sum of the accuracy and compression objective functions. This equality weighting preserves the convexity of the objective functions and simplifies optimization.

To explore parameter sensitivity near the optimum, the best assessed parameters are defined (λ = 8, ε = 20, and α = 50) and each parameter is independently varied about this point. The following sections present the results.

2.5.2. Adjacency

The adjacency parameter, λ gives the maximum spatial extent that a pixel can be separated from an existing segment and still be eligible for membership. The value of λ is varied between 2 and 32 pixels. The effect of adjacency on nonmatching pixel percentage is shown in Figure 12a. As λ is reduced, more pixels match their original color following segmentation. For λ between 2 and 8, the cumulative nonmatching percentage remains

below 10% over all eight scenes. However, as λ increases above 8 pixels the cumulative nonmatching pixel percentage increases linearly.

An opposite trend occurs in the analysis of the cumulative number of segments produced, shown in Figure 12b. As λ increases, the number of segments produced by leap segmentation dramatically decreases as pixels are more readily grouped into segments that span large areas in the image. For λ values of 4 or less, the large cumulative segment

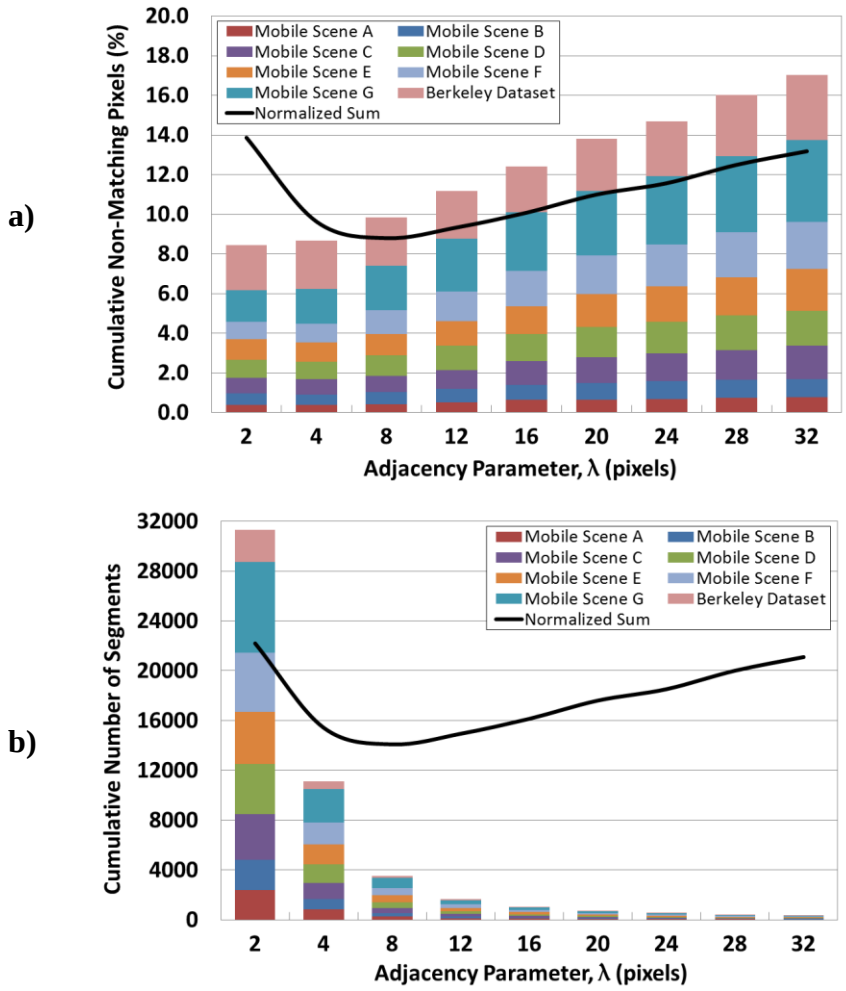


Figure 12. Analysis over several mobile camera scene runs for the adjacency parameter (λ) varying between 2 and 32 pixels. (a) The cumulative nonmatching pixel percentage increases as λ increases. (b) The cumulative number of segments decreases as λ increases. The aggregate objective function is overlaid in black.

(a) **(b)**

Figure 13. Qualitative image comparison, adjacency parameter (λ). Segmentation visual quality decreases as adjacency constraints are relaxed from a) λ = 2 (1219 regions) to (b) λ = 32 (25 regions).

counts diminish the compression effect, as shown in Figure 13a. The individual scene collection performances in Figure 12b show that the effect of λ is similar across diverse scenes.

Increased λ has two effects. Locally, segments are less affected by noise and small occlusions that disrupt growth; regions are able to *leap* over non-similar obstacles. At a larger scale, increased λ allows segments to extend across greater areas in the image, further reducing similar but spatially disjoint segments.

Excessively large values of λ adversely affect segmentation quality. As segments encompass a larger number of pixels, the mean color components of the region can drift, and no longer match member pixels. While this is corrected in a post-process region adjustment, it can distort segment boundaries, as shown in Figure 13b.

The aggregate objective function (AOF), overlaid on the results in Figure 12, shows an optimum near λ = 8, which corresponds to the value of the best assessed parameter set.

2.5.3. Equivalence Threshold

The equivalence threshold parameter, ε, defines how color- and luminance-similar

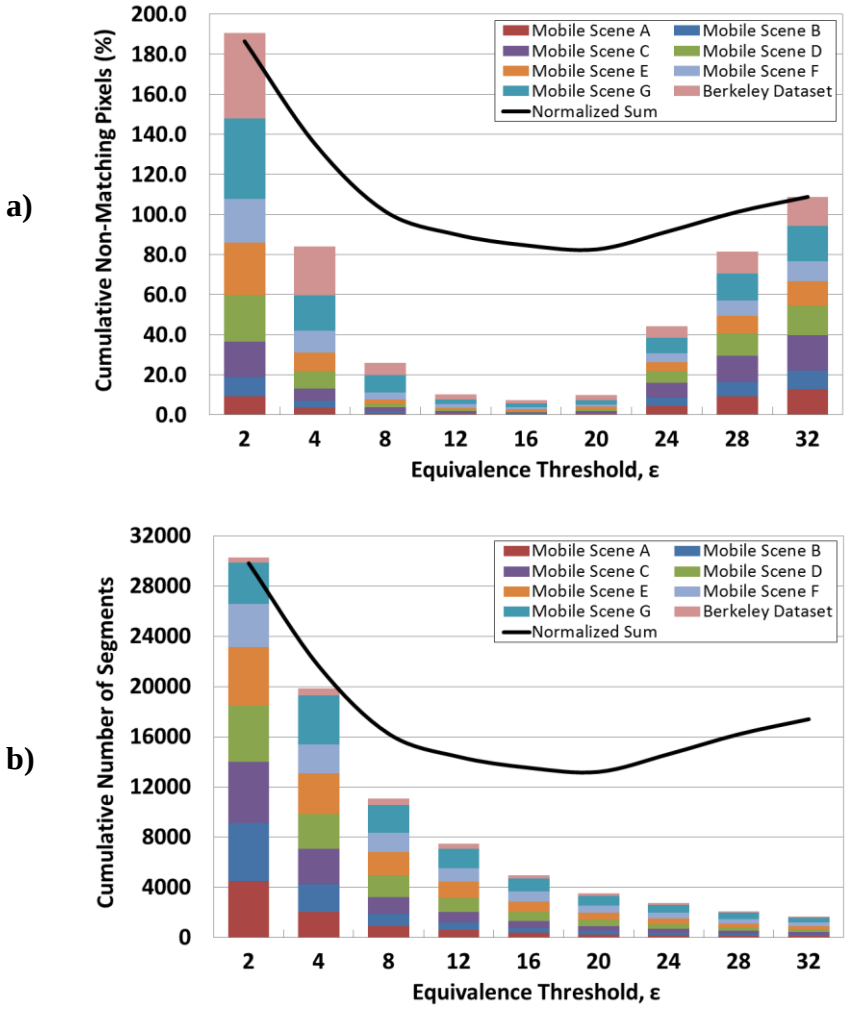


Figure 14. Analysis over several mobile scene runs for equivalence thresholds (ε) varying between 2 and 32. (a) The cumulative nonmatching pixel percentage first decreases as ε increases, then increases as ε increases. (b) The cumulative number of segments decreases as ε increases. The aggregate objective function is overlaid in black.

a pixel must be to gain membership in a segment. Equivalence is defined as the CL-similar relation, shown in Equation 1. The value of ε is varied between 2 and 32.

The effect of equivalence threshold on cumulative nonmatching pixel percentages is shown in Figure 14a. For ε values between 12 and 20, cumulative nonmatching pixel percentages remain below 10% across the eight scene collections. Outside of this range, cumulative nonmatching pixel percentages increase rapidly. This is expected; more color diverse pixels are admitted into segments as ε increases. The reduction in accuracy when ε falls below 12 occurs for a different reason. This stricter requirement for equivalence reduces segment size, which increases the area of the image represented by segments that are below the minimum segment size. Since these small segments are assimilated into

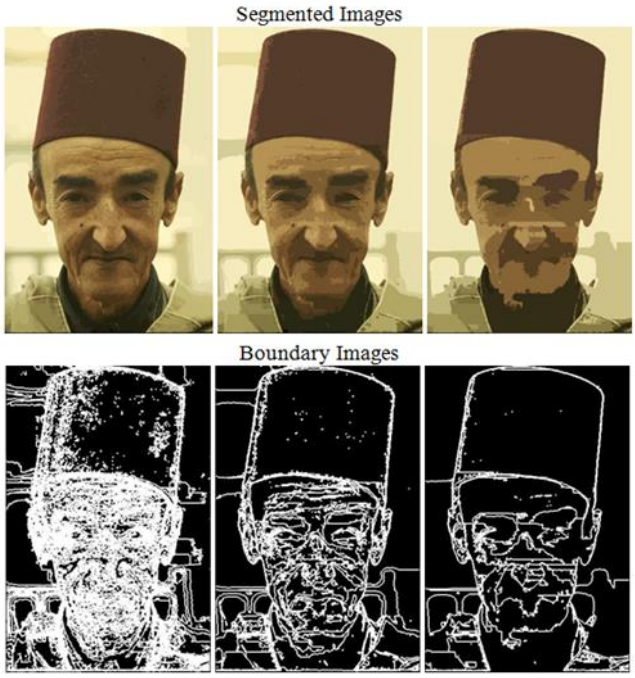


Figure 15. Qualitative image comparison, equivalence threshold (ε). (a) Over-segmentation, ε = 2 (2035 regions). (b) Optimal segmentation, ε = 20 (330 regions). (c) Under-segmentation, ε = 32 (125 regions).

larger ones (with resulting color distortion), the overall segmentation accuracy is reduced.

The impact of the equivalence threshold on compression is shown in Figure 14b. The cumulative number of segments increases as ε is reduced. A small ε results in *over-segmentation*: scene features are transformed into many small, similar segments, as shown in Figure 15a. A large ε produces *under-segmentation*: multiple scene features are merged into a segment, distorting object boundaries, as shown in Figure 15c.

The normalized sum of accuracy and compression objective functions is overlaid on the results in Figure 14. It suggests an optimum near $\varepsilon = 20$. A qualitative study of leap segmentation supports this equivalence threshold value. At this value of ε, the over- and under-segmentation of key scene features is minimized.

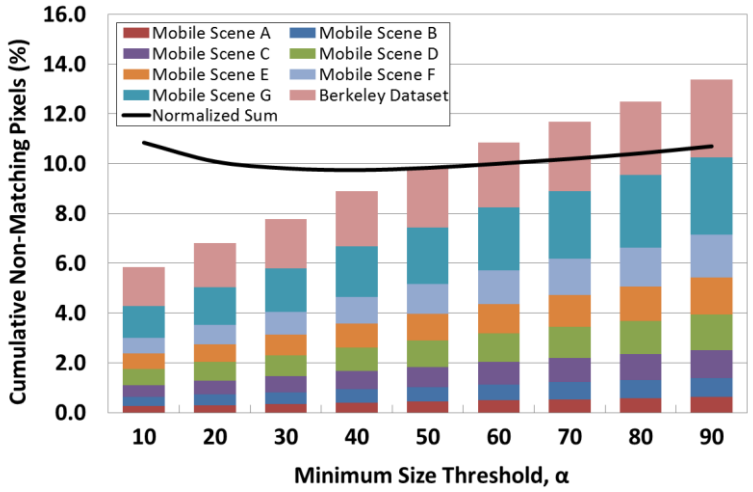
2.5.4. Minimum Size Threshold

The minimum size threshold parameter, α, determines the minimum area of an independent segment. It is defined as the ratio of segment area, c to the corresponding adjacency neighborhood area (λ x λ):

$$\alpha = 100 \times \frac{c}{(\lambda \times \lambda)}. \quad \textbf{(5)}$$

Segment area is significantly affected by adjacency. The adjacency area specifies a maximum ignorable occlusion size. Defining the minimum segment area in terms of the adjacency area maintains segment size discrimination as adjacency changes. Increasing or decreasing adjacency provides a corresponding increase or decrease in minimum segment area. The minimum segment size threshold is evaluated for values between 10 and 90. The cumulative nonmatching pixel percentage increases with increasing α, as shown in Figure 16a. This contrasts with the compression trend, in Figure 16b, where the

a)



b)

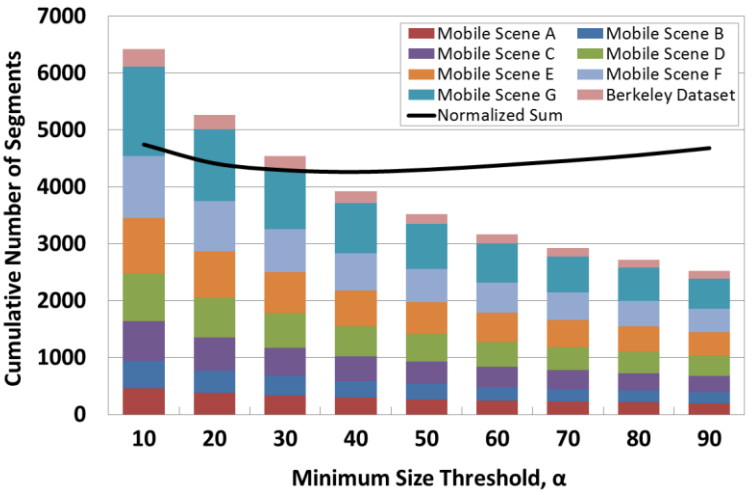


Figure 16. Analysis over several mobile scene runs for minimum size thresholds (α) varying between 10 and 90 percent. (a) The cumulative nonmatching pixel percentage increases as α increases. (b) The cumulative number of segments decreases as α increases. The aggregate objective function is overlaid in black.

cumulative number of segments decreases as α is increased. For large values of α, fine scene details are lost as small segments are assimilated into larger nearby segments. Small values of α preserve insignificant scene details, and reduce scene compression.

Evaluation of the minimum size threshold is dependent on qualitative assessment of appropriate scene details, shown in Figure 17. The AOF has the least pronounced

a)

b)

Figure 17. Qualitative image comparison, minimum size threshold (α). Significant blurring occurs over traffic scene details such as headlights and street signs from range (a) α = 10 (1659 regions) to (b) α = 90 (527 regions).

optimum in the minimum size threshold, near α = 40. The overall change in the accuracy and compression objective functions is small across this range α. In the qualitative analysis of diverse scene types, especially from the Berkeley Segmentation Dataset, the human assessed quality improvement near the AOF optimum is almost unperceivable. When assessing the appropriate segmentation of fine scene details, the best algorithm performance occurred at α = 50. This minimum size threshold improves scene compression by approximately 10% over the AOF optimum.

The parameter values (λ = 8, ε = 20, and α = 50) combine accuracy and compression objective functions and qualitative assessment to achieve optimal segmentation across diverse scene types.

2.6. Experimental Results: Intelligent Vehicle Traffic Scenes and the Berkeley Segmentation Dataset

This section evaluates the quality and performance of leap segmentation compared with two well-known image segmentation algorithms: Mean-Shift Clustering with Edge Detection (EDISON) [20], and Efficient Graph-Based Image Segmentation (EGBIS) [30]. All three segmentation algorithms are implemented in C and executed in a Linux environment.

The primary dataset used for this comparison is 300 images from the Berkeley Segmentation Dataset [55], [56]. This provides a diverse collection of scene types with varying feature sizes and scales. Additional scene collections at Georgia Tech (the GTTraffic dataset [32] (discussed in detail in Section 3.4)) were captured using a forward-mounted Kodak Zi6 on an automobile dashboard. This camera provides a fixed focus, fixed aperture, and fixed field of view with electrically controlled gain and sensitively. The images extracted from the captured mpeg4 videos are at a resolution of 1280 x 720 pixels. To ensure a consistent comparison, all algorithms were adjusted to produce similar levels of segmentation.

Figure 18 shows a collection of segmentation results.[1] The figure displays segmentation output for each technique and is labeled with segment count information. These segmentation results consistently show that leap segmentation is able to preserve significantly more detail from the input scenes than the EGBIS and EDISON approaches

[1] Complete tables of segmentation comparison results are available at: <www.ece.gatech.edu/research/pica/pdf/LeapFullResults.pdf>.

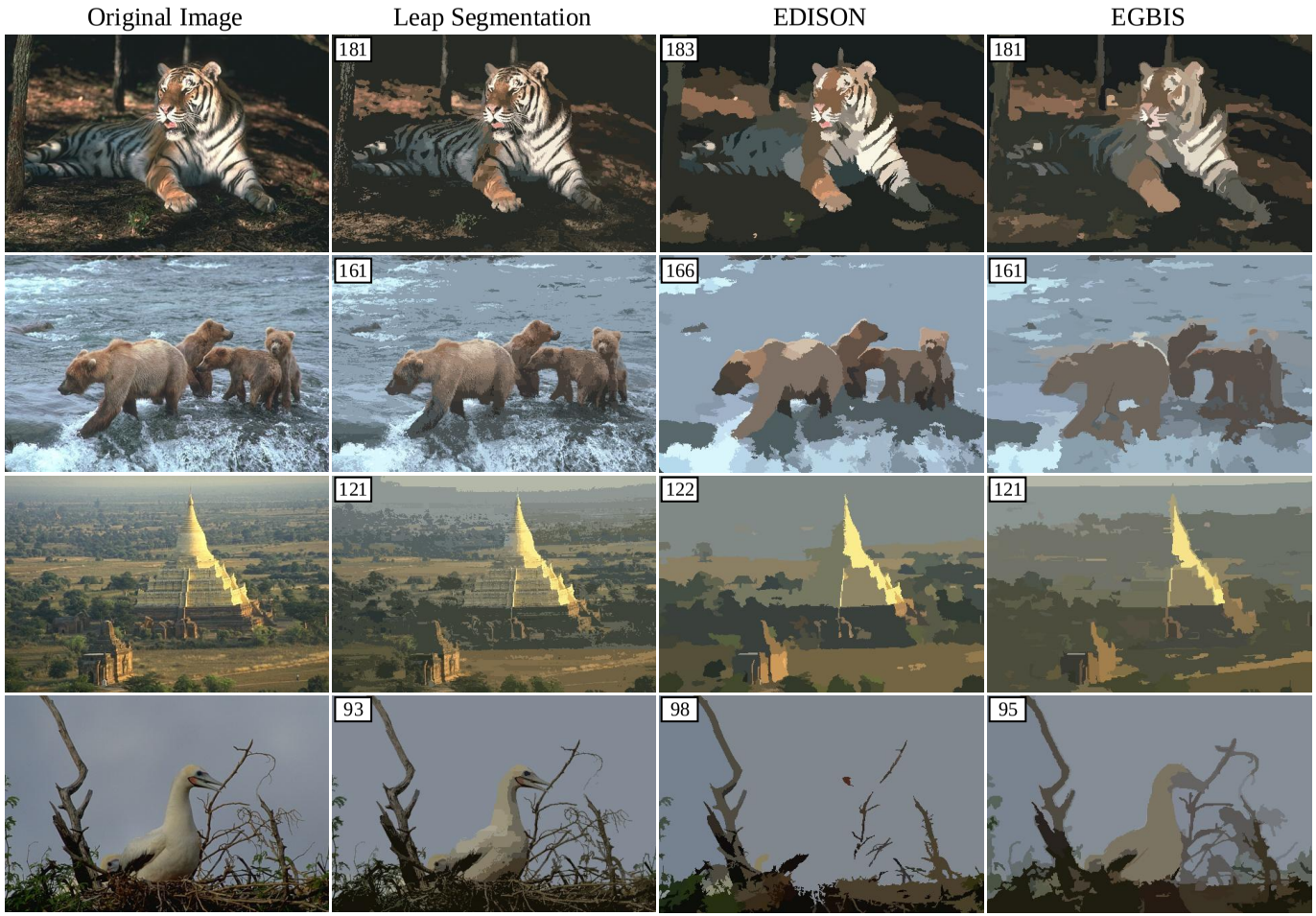


Figure 18. The segmentation output images for each approach for comparison. The number of segments for each output is labeled. For a given scene, the outputs from the segmentation approaches are very similar in segment count so that they may be compared fairly.

while using the same number of segments. See the Appendix for additional comparison results.

To the human eye, the leap segmentation results are difficult to differentiate from their corresponding original images. The approach is able to significantly reduce and restructure the information needed to represent an image while still preserving salient features that are necessary for scene analysis.

2.6.1. Segmentation Comparison – Traffic Scene

The quantitative objective functions introduced in Section 2.5.1, the *number of segments* (to evaluate compression) and the *nonmatching pixel percentage* (to assess segmentation accuracy), are applied in the segmentation approach comparison. In Figure 19, the resulting segmentations of a traffic scene (1280x720 pixels) are shown for each technique. The three techniques are adjusted to produce similar segment counts (~800 segments) for comparison. The nonmatching pixel percentages are listed above each segmentation output. Figure 19a shows the merged segmentation output for each technique. Figure 19b shows a binary matching map for the final segmentation, where nonmatching pixels are plotted in white.

Of the three approaches, leap segmentation provides the highest accuracy with a low nonmatching pixel percentage (3.1%). EDISON and EGBIS exhibit larger nonmatching percentages of 14.0% and 29.8%, respectively. The non-matching maps in Figure 19b suggest that leap segmentation achieves greater accuracy in high variability regions like trees and grass, even when they cover a large portion of the image. Leap segmentation benefits from non-contiguous adjacency, allowing pixels within an adjacency threshold to be joined in a segment.

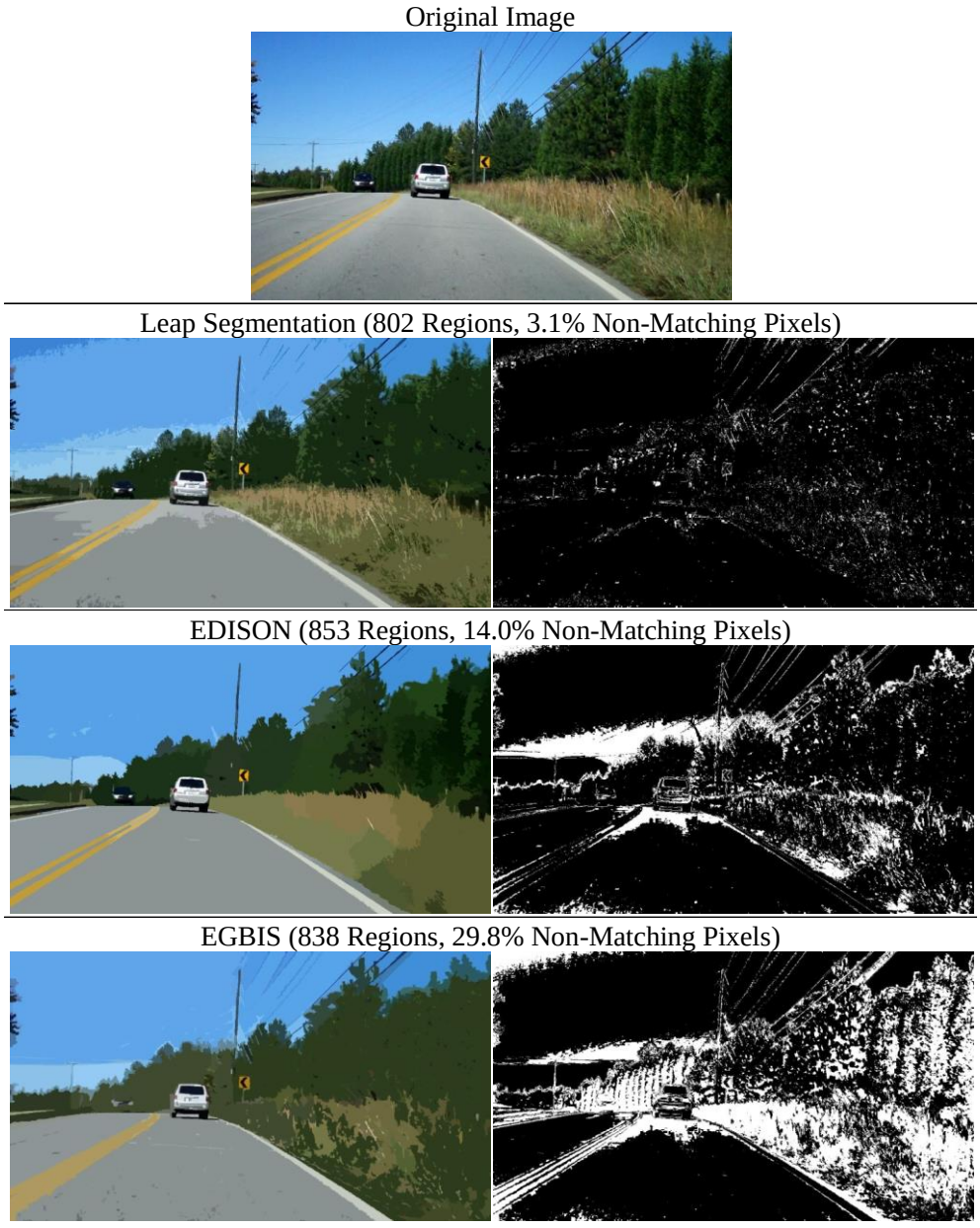


Figure 19. Segmentation comparison images, traffic scene (1280x720 pixels). (a) The merged segmentation output for each technique. (b) A binary map of nonmatching pixels in the output segmentation.

2.6.2. Segmentation Comparison – Animal Scene

In Figure 20, the resulting segmentations of a zebra scene (481x321 pixels) are shown for each approach. As before, the three approaches are adjusted to produce similar segment counts (~80 segments). Figure 20a shows the merged segmentation output for each technique. To help discern region membership in the merged image, an artificially colorized segmentation is given in Figure 20b. Contrasting color assignments show region pixel membership. Figure 20c shows a binary map of nonmatching pixels in the final segmentation. Leap segmentation is the most accurate at maintaining original image information, providing the lowest nonmatching pixel percentage (5.9%) of the three approaches. EDISON and EGBIS produce more than five times more nonmatching pixels.

Images with stripes often present a problem for classical segmentation algorithms. While similar colors may exist within the stripes, these colors are not directly connected in the image and so cannot be grouped by traditional techniques. Ideally, a segmentation technique would require only two segments to cover a zebra. In the colorized images in Figure 20b, EDISON and EGBIS assign each zebra stripe to a different segment. Leap segmentation allows same-color stripes to be grouped even when they are not touching. Leap segmentation achieves an efficient two color segmentation of each zebra.

The three approaches produce a substantial variation in preserved detail, shown in Figure 20a. By achieving greater pixel coverage with each segment, leap segmentation captures more detail from the original image. If the purpose of segmentation output is to identify object boundaries, this detail may be irrelevant. However if segmentation is being used for feature recognition, salient details are valuable in the process.

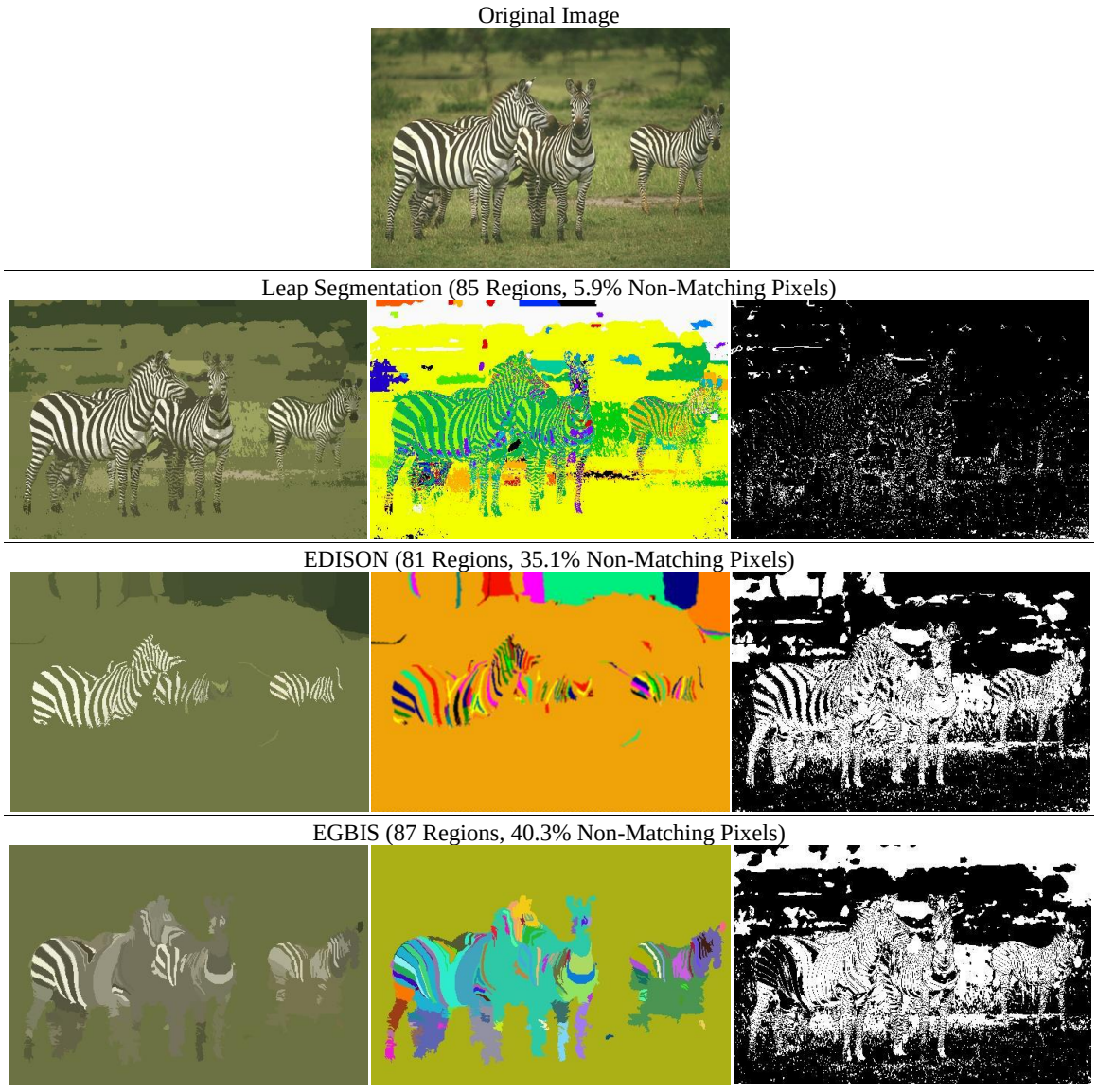


Figure 20. Segmentation comparison images, animal scene (481x321 pixels). (a) The merged segmentation output for each technique. (b) A colorized representation of the segmentation to show region membership clearly. (c) A binary map of nonmatching pixels in the output segmentation.

2.6.3. Detail Preservation Experiment

Leap segmentation strives to identify regions of homogeneous pixels in an image and to remove unimportant image features, such as texture and minor chromatic variations, thus reducing the amount of information necessary to represent a scene. It also avoids removing scene information that is important for analysis, such as road sign lettering, lane markers and facial features. This section presents an evaluation of the three segmentation approaches based on their ability to preserve these salient details.

Each technique's segmentation of road sign images are shown in Figure 21. For scene understanding, the letters on each sign should be preserved in the segmentation output with minimal blurring and distortion. The EGBIS approach does not preserve details, and often blocks lettering into more convex segments. EDISON preserves some letters, but blurs others limiting text recognition. Leap segmentation provides the most accurate lettering representation. In each example, sign letters are visible with minimal

Figure 21. Lettering on street signs is processed using different segmentation approaches for detail preservation comparison.

blurring and distortion.

2.6.4. Image Gradient Evaluation

All segmentation approaches strive to group color-similar pixels into a large region. Slow-changing gradients allow a region's color to drift, expanding the tolerated variance during the segmentation process. The lack of contrasting edges to limit a region's extent can result in decreased color accuracy. When processing gradients, a balance must be struck between segment area and color accuracy.

Initially, leap segmentation, EDISON, and EGBIS produce similar results for gradient images, shown in Figure 22. Road and sky gradients produce large-area regions that capture these scene elements well. But the increased range in member colors produces lower matching accuracy with the resulting segment color. This is seen in both the nonmatching pixel percentages (24.4%, 25.9%, and 18.9% for EDISON, EGBIS, and Leap) and the nonmatching binary images in Figure 22b.

Leap completes the segmentation process with a post-process region adjustment step (described in Section 2.4.2) that divides gradient regions into multiple segments. While this reduces the resulting segment size and typically fractures a single object (e.g., road, sky) into multiple segments, it dramatically improves the color accuracy, from an 18.9% nonmatching percentage to a 1.7% nonmatching percentage.

While some applications favor minimizing the number of segments and the fractionation of objects (e.g., identifying object boundaries), these algorithms must still handle this segmentation condition for objects that are composed of high-contrast color elements.

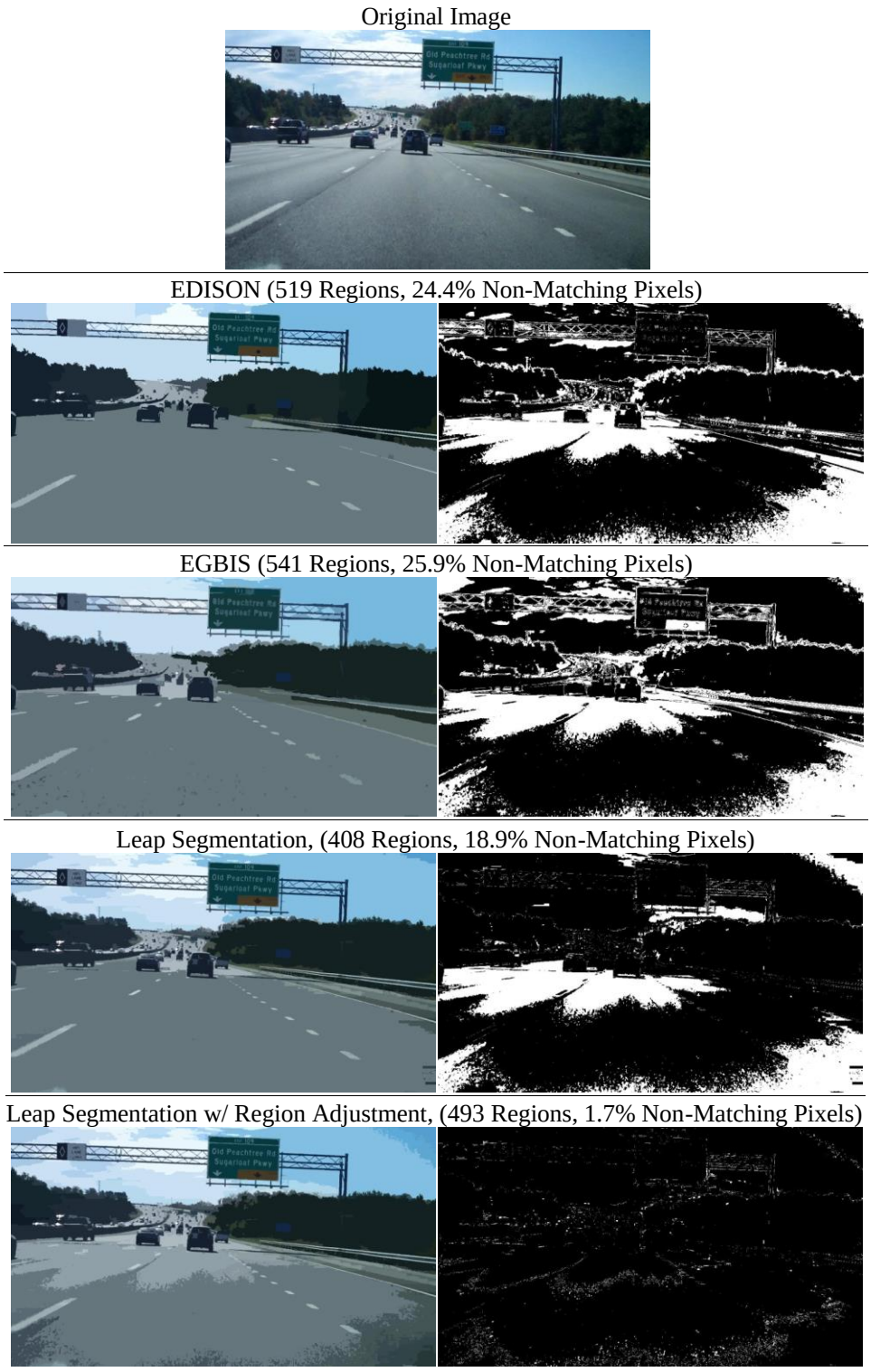


Figure 22. Image gradients evaluation, EDISON, EGBIS and Leap Segmentation. (a) The merged segmentation output for each technique. (b) A binary map of nonmatching pixels in the final segmentation.

2.6.5. Matching Accuracy and Run-Time Analysis

The approaches are evaluated using a 2.13 GHz Intel Core I3-330M processor running 64-bit Ubuntu 10.04. The algorithms were not parallelized or otherwise altered for the platform. Figure 23 presents a reference table containing sub-images to identify the actual images used during testing. The full-size versions of these reference images are used in the evaluation. The left four images are from the Berkeley Dataset [55], [56] and are of size 481x321 pixels. The remaining three images are from mobile camera traffic sequences captured at Georgia Tech and are 1280x720 pixels in size.

Leap segmentation produces superior accuracy results over the EGBIS and EDISON approaches. Figure 23a lists the tabulated nonmatching pixel percentage results for each approach. In all trial runs, the leap segmentation approach exhibits lower

	Red Hat	Striped Shirt	Zebras	Tiger	Highway	Two Cars	Rural Road
	481x321	481x321	481x321	481x321	1280x720	1280x720	1280x720
							

a) Non-Matching Pixel Percentage (%)

	Red Hat	Striped Shirt	Zebras	Tiger	Highway	Two Cars	Rural Road
LEAP	1.1	0.8	3.9	4.0	1.7	1.4	3.1
EDISON	13.1	15.1	35.1	18.4	24.4	10.7	14.0
EGBIS	32.8	20.3	40.3	23.5	25.9	12.2	29.8

b) Number of Segments

	Red Hat	Striped Shirt	Zebras	Tiger	Highway	Two Cars	Rural Road
LEAP	85	114	86	181	493	621	802
EDISON	91	112	81	183	519	617	853
EGBIS	87	114	87	181	541	605	838

c) Run Time (ms)

	Red Hat	Striped Shirt	Zebras	Tiger	Highway	Two Cars	Rural Road
LEAP	11.6	13.3	14.4	15.4	77.9	75.7	92.4
EDISON	12,360.0	13,850.0	15,950.0	13,990.0	534,550.0	535,100.0	827,730.0
EGBIS	144.2	127.2	150.6	145.8	1,262.5	1,230.1	1,271.4

Figure 23. Segmentation accuracy, compression, and run-time analysis. (top) A table of reference images and their IDs. (a) The nonmatching pixel percentages for each test image segmentation. (b) The total number of segments in each output segmentation. (c) The execution times required to produce each segmentation output.

nonmatching pixel percentages than the classical approaches with varied input image scenes.

In addition, leap segmentation is computationally more efficient than the EGBIS and EDISON approaches. Figure 23c lists the tabulated algorithm run-times for each approach. An analysis of the trial results reveals that leap segmentation, in all runs, is over 900x faster than EDISON and 10x-15x faster than EGBIS, the current state of the art.

In this section, an extensive comparison of leap segmentation with two classical segmentation approaches reveals that leap segmentation is both highly accurate, detail preserving, and computationally efficient. These qualities make the successful utilization of the leap segmentation approach highly promising.

2.7. Classical Performance Metrics

In addition to the detailed performance evaluations presented in this section, investigations of two classical, human-based metrics were also performed to further evaluate leap segmentation accuracy: the F-measure [76] and the Probabilistic Rand Index (PRI) [75]. Accuracy measures based on human-labeled ground truths such as these tend to discount segmentation approaches that maintain image detail despite any corresponding reductions in image data. This is because humans tend to segment whole image objects into large segments, resulting in a very low segment count. Because of this, the author notes that these human-based metrics are more appropriate for evaluating boundary detection or under-segmentation approaches than for evaluating preprocessing over-segmentations such as the approach outlined in this chapter. Despite this, the author found that leap segmentation provided performance numbers comparable to classical

segmentation approaches using these metrics, consistently showing F-measure and PRI scores equal to or only slightly lower than classical methods. The following sections provide a detailed analysis of all experiments performed using these classical performance metrics and a discussion of relevant findings.

2.7.1. Experimental Setup

The accuracy of classical segmentation approaches is typically evaluated with comparisons to human-labeled ground truth images. Humans tend to segment whole image objects into large segments, resulting in a very low segment count. An example human segmentation is shown in Figure 24b. The entire tiger in the image is grouped into one segment in the human segmentation. The average human-labeled ground truth segmentation from the Berkeley dataset is composed of only ~18 segments.

Comparing with human segmentations is a good strategy for evaluating classical segmentation approaches because the two share similar goals. However, the objectives of classical segmentation and leap segmentation differ. Classic segmentation approaches attempt to accurately detect object boundaries in a scene. Leap segmentation focuses on

(a) **(b)**

Figure 24. Example of a human segmentation. (a) Original Image. (b) Human segmentation output.

eliminating less significant detail in the image, such as texture and minor chromatic variations, while accurately representing essential scene content. Leap segmentation results are therefore not directly comparable with classical segmentations results.

Nevertheless, leap segmentation is evaluated using two classical, human-based accuracy metrics to determine if leap segmentation, while not designed for classical segmentation uses, can still provide accuracy results that are comparable to classical approaches. The following two widely-known, classical, human-based metrics are used to evaluate leap segmentation accuracy: the F-measure [76] and the Probabilistic Rand Index (PRI) [75].

The evaluation is performed in a similar manner as Hanbury and Stöttinger [42] in their paper on segmentation evaluation metrics. The EGBIS and EDISON approaches are executed over specific parameter ranges. For the EDISON approach, a spatial bandwidth of $h_s = 12$ is used, chosen according to the size of the input image. The range bandwidth, h_r, is evaluated for values between 4 and 20. For the EDISON approach, a Gaussian smoothing input value of $\sigma = 0.8$ is used and the threshold k is evaluated for values between 50 and 1050. The leap segmentation approach is evaluated using the optimal parameters for the adjacency threshold, $\lambda = 8$, and the minimum region size, $\alpha = 50$. The equivalence threshold, ε, is evaluated for values between 8 and 64. The F-measure is evaluated using the resources from the Berkeley Segmentation Dataset and Benchmark [55], [56]. The Probabilistic Rand Index is evaluated using the resources from the Image Segmentation Benchmark Indices Package [83]. It is important to recall that leap segmentation is penalized by these benchmarks for its alternative segmentation objectives.

2.7.2. Boundary Precision-Recall

The *F-measure* [76] is used to compare the output of boundary detection (or segmentation) algorithms with human-segmentation ground truths. The F-measure, shown in Equation 6, is computed using both the precision (P) and the recall (R) of a boundary image and outputs a measure of algorithm performance. Precision is the fraction of boundary pixels in the output segmentation that correctly match boundary pixels in the human segmentation. If a large amount of noise is present in the output segmentation, its precision score will be low. Recall is the fraction of boundary pixels in the human segmentation that are correctly identified by the output segmentation. Recall represents the portion of the human segmentation ground truth that is correctly detected.

$$F = \frac{2 \times P \times R}{P + R}. \qquad \textbf{(6)}$$

Ideally, both the precision and recall of a segmentation are high, near a value of one. The range of the F-measure is from zero to one, with higher values indicating better, more accurate segmentations.

The precision-recall curves for each approach are presented in Figure 25. The figure legend lists the value of the optimal F-measure achieved for each precision-recall curve. The EGBIS approach yields the highest F-measure over all approaches (F=0.61). The mean-shift-based EDISON approach's F-measure is lower (F=0.55). This is likely due to a lack of attention to texture cues by the EDISON approach. Other evaluations of mean-shift from the literature report higher F-measures (F=0.64) [42] due to implementation differences.

The leap segmentation approach yields F-measure scores that are lower, but

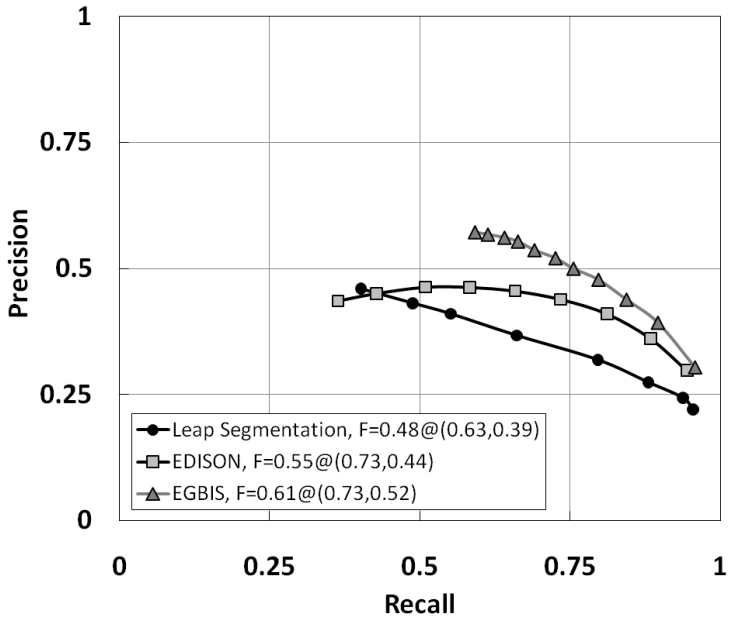


Figure 25. Boundary precision-recall curves with corresponding F-measure results for each segmentation approach.

comparable to the other approaches (F=0.48). This is due to leap segmentation's consistently low precision scores. These low precision scores are due to the large amount of boundary detail maintained by leap segmentation when it applies its flexible adjacency constraint. While this allows the approach to create segmentations that very accurately represent the original image using a small amount of information, it lowers the average precision of the approach as this extra detail is viewed as noise when compared to human segmentation ground truths.

2.7.3. Probabilistic Rand Index

The probabilistic rand index (PRI) [75] operates under the assumption that if two pixels in the human segmentation are identified as part of the same segment, then this pixel pair should also be a part of the same segment in the output segmentation. The PRI

measures the fraction of pixel pairs whose segmentation membership corresponds correctly between the human segmentation and the segmentation being tested. The PRI range is from 0 to 1. A higher PRI indicates a more accurate segmentation. To provide a fair method of comparing performance with this metric, the PRI must be evaluated with respect to the number of segmentation regions used. In Figure 26, the average PRI for each segmentation approach is plotted against the average number of segmentation regions produced to provide a fair method of comparing performance with this metric.

The EDISON approach yields the highest PR index for greater numbers of segments. However, the EGBIS approach performs slightly better for region counts near 100 regions. The leap segmentation approach provides comparable performance numbers, consistently showing a PRI only slightly lower than or equal to the other methods.

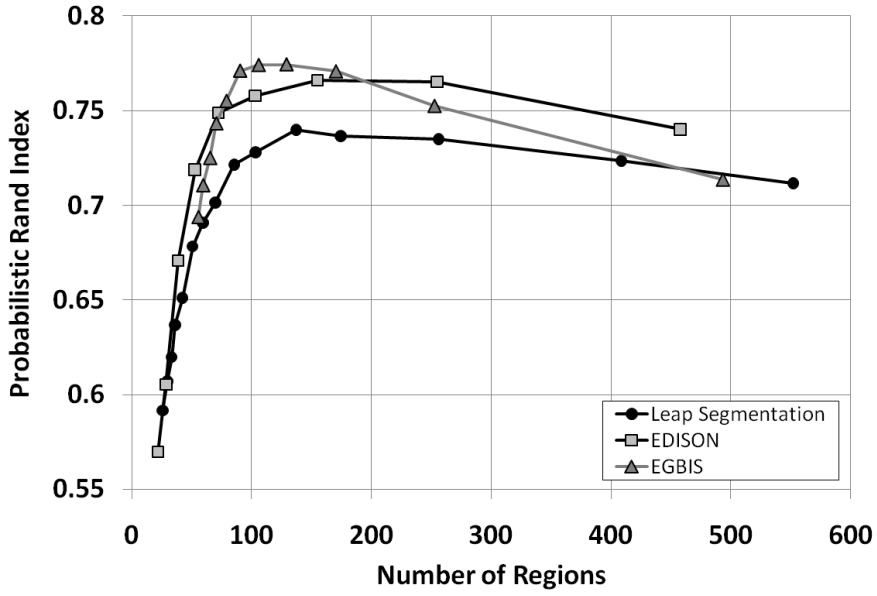


Figure 26. Average Probabilistic Rand Index (PRI) versus the average number of regions in the output for each segmentation approach.

2.8. Experimental Results: Image Labeling and 3D Reconstruction

To further demonstrate the performance benefits of leap segmentation, the following section shows that applying the novel leap segmentation technique significantly improves the efficiency of a 3D scene reconstruction task (Forsthoefel et al.) [31]. The scene layout reconstruction approach developed by Hoiem et al. [45] is used as a representative approach in the task of automatic scene labeling and 3D reconstruction from a single image. Figure 27 and Figure 28 give examples of automatic 3D scene reconstruction based on surface layout using this approach. This application is used to demonstrate the performance benefits achieved when leap segmentation is used in the preprocessing stages of a high-level vision application.

2.8.1. Application Background

Recovering the surface layout of a scene is an important step in scene understanding research. The addition of object orientation and depth information to automated vision systems can drastically improve their scene perception and analysis

(a) **(b)**

Figure 27. Example output of automatic 3D reconstruction using Hoiem et al.'s approach. (a) Original input image by Liebowitz et al. [51]. (b) Novel 3D view from [44].

(a) **(b)**

Figure 28. More example outputs of automatic 3D reconstruction using Hoiem et al.'s approach. (a) Original input image. (b) Novel 3D view from [44].

performance, allowing these systems to better understand and operate in their 3D environment. Like many high-level vision tasks, 3D scene layout applications often use image segmentation techniques to preprocess pixel data prior to image analysis. Rather than processing each pixel individually, these vision applications use segmentation to group image pixels into segments that can be processed more rapidly.

Several approaches to the challenge of 3D layout reconstruction exist. A general survey of 3D modeling research is given by Besl and Jain [12]. Many early 3D modeling approaches use photometric stereo to estimate scene depth, in which multiple views of a scene are collected and analyzed for scene depth information. For the purposes of this research, the use of monocular vision to achieve accurate 3D scene reconstruction is of the most interest. A recent survey of 3D reconstruction techniques from single images is

provided by Mohan and Mani [59]. The described approaches vary in their appropriate applications. Some approaches perform best on structured objects such as faces, while others are more amenable to unstructured scenes.

2.8.2. Representative Approach

The 3D scene reconstruction approach developed by Hoiem et al. [45] is used as a representative approach in the task of automatic scene labeling and reconstruction from a single image. Hoiem et al.'s approach classifies outdoor scene surfaces into three main geometric classes: ground, vertical, and sky. Surfaces that are parallel to the ground (i.e., roads) fall into the ground class. Surfaces that stick up from the ground (i.e., walls) become part of the vertical class. Sky pixels are grouped together to form the final geometric class. The vertical class pixels are further classified into several subclasses. A planar surface, such as a wall, is classified as "left," "center, or "right" depending on its orientation. Non-planar surfaces, such as tree leaves or wires, are classified as either "porous" or "solid."

Hoiem et al.'s technique involves gradually building knowledge of scene structure. First, a segmentation pre-processing step is used to divide input image pixels into groups called "superpixels." Next, superpixels are grouped into larger sets called constellations. Constellations of superpixel regions are homogeneous (all member superpixels have same-label assignments), but need not be spatially contiguous. Multiple segmentations of superpixels into constellations are evaluated and the best configuration is selected. Statistical learning is then applied to compute the geometric label for each constellation from training data. This procedure incorporates location, color, texture, and perspective statistics for classification. It demonstrates that a 3D model of the scene can

be easily reconstructed from these geometric surface labels.

Hoiem et al. incorporate image segmentation into the preprocessing stages of their layout technique to improve the computational efficiency of their overall approach. In their implementation, the authors selected the EGBIS approach [30] for segmentation preprocessing. The EGBIS approach is state of the art in computational efficiency [65], making it a common choice for vision application developers. In the following sections the use of alternative segmentation approaches in this labeling procedure, including the developed leap segmentation approach, is investigated to identify all possible performance benefits.

2.8.3. Dataset and Evaluation Method

The framework for comparison of image segmentation techniques in scene labeling preprocessing procedures consists of three stages: segmentation, labeling, and analysis [31]. In the segmentation stage an image is preprocessed using one of the three candidate image segmentation techniques described in Section 2.6. The segmentation results are then passed into the labeling stage, in which Hoiem et al.'s automatic scene labeling approach [45] is performed on the input segmentation. The final labeling results from each candidate segmentation technique are then compared using both accuracy and efficiency performance metrics.

The dataset used for comparison is Hoiem et al.'s publicly available library of ground truth images for evaluating the accuracy of labeling tasks. The library consists of 300 outdoor images of various sizes and scales. Ground truth labels have been manually assigned to each library image. The accuracy of classification for both main class and subclass labeling is measured as the fraction of image pixels whose labels match ground

truth assignments.

Many vision applications apply image segmentation during preprocessing to improve their execution efficiency. The applied segmentation technique must therefore be efficient in its own execution; a slow execution could counteract any facilitated system performance improvement. Therefore, the speed of the segmentation procedure when used in preprocessing is used to assess segmentation performance. Efficiency is measured as the average execution time of segmentation preprocessing procedures over 300 dataset images. The diverse collection of image sizes present in the evaluation dataset allows rigorous testing of algorithmic execution efficiency and scaling.

As in previous experiments, an aggregate objective function (AOF) is useful in determining overall segmentation performance. The AOF is defined as sum of the normalized accuracy and efficiency objective functions. Divide-by-maximum normalization is used to scale the efficiency objective function for comparison. The efficiency performance ceiling is evaluated at 32 fps to approximate real-time processing efficiency standards.

2.8.4. Results

In Figure 29, the accuracy and efficiency performance results for both main class (left) and vertical subclass (right) labeling are shown for each segmentation technique. The three segmentation approaches produce very similar accuracy results for main class labeling (~87%), differing by less than 2%, as shown in Figure 29(top). The subclass labeling accuracy results are also comparable (~69%), differing by less than 3%. However, leap segmentation processing proves significantly more computationally efficient (at least 10x faster) than both the EGBIS and EDISON approaches, as shown in

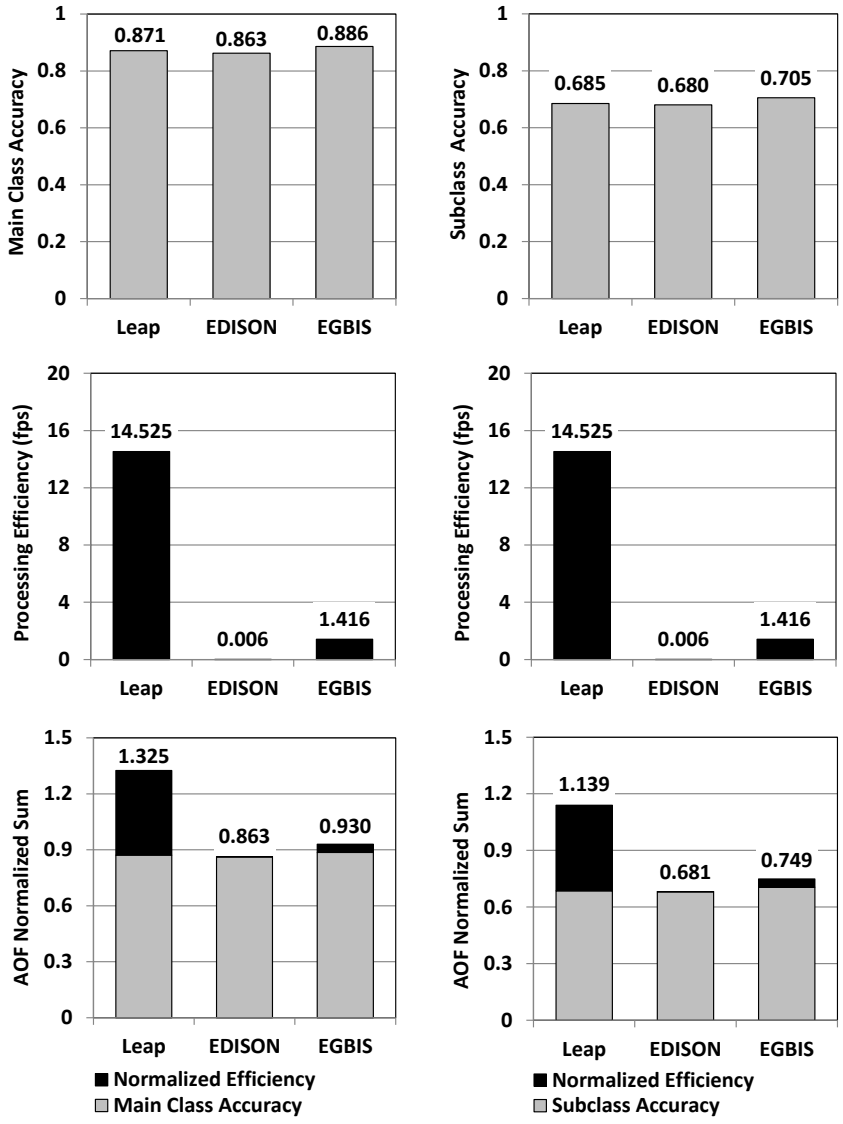


Figure 29. Performance results for both main class (left) and vertical subclass (right) labeling. (top) Labeling accuracy performance. (center) Segmentation processing efficiency performance. (bottom) Overall performance results using AOF normalized sum.

the execution efficiency plots in Figure 29(center).

Overall performance is illustrated using the AOF plots shown in Figure 29(bottom). Of the three approaches, leap segmentation yields the highest performance results by far. This is due to the huge improvements in computational efficiency achieved by leap segmentation over both the EGBIS and EDISON approaches while maintaining comparable labeling accuracy performance. Examples of labeling results for each segmentation technique along with their ground truth labeling assignments are shown in Figure 30 for qualitative comparison.

The equality weighting used to compute overall performance is appropriate for those applications that require both accuracy and efficiency performance considerations and may not be appropriate for applications that are purely accuracy driven and that lack efficiency standards. *Those vision applications that require preprocessing to perform*

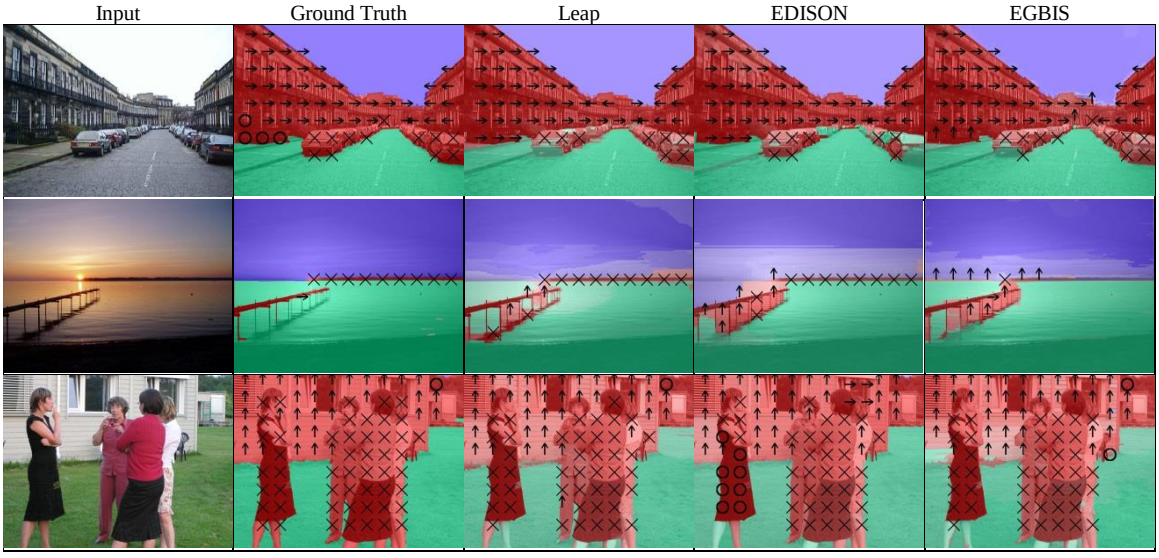


Figure 30. Scene labeling results for qualitative comparison of segmentation performance. Main class labels are indicated by color (green=support, red=vertical, blue=sky). Subclass labels are indicated by symbols (planar surfaces use arrows left, up (center), and right to indicate surface orientation, non-planar surfaces use 'O' for porous and 'X' for solid).

both accurately and efficiently, such as the labeling procedure used in these experiments, can achieve significant improvement in performance by applying leap segmentation in preprocessing.

2.9. Conclusion

This chapter introduces leap segmentation, an efficient, non-contiguous image segmentation approach that employs novel techniques to use resources efficiently and to produce output segmentations that accurately represent salient features from input image scenes. In experiments, leap segmentation demonstrates high region-assignment accuracy and, compared to other approaches, preserves more scene details using a given storage resource. Leap segmentation's ability to maintain salient image details during segmentation sets it apart from traditional approaches which tend to blur or discard these important details. Experiments show that leap segmentation is able to correctly maintain an average of 20% more original scene pixels than traditional approaches despite using the same number of segments and while exhibiting a significant improvement in execution speed (> 10x faster than the state of the art). The salient features maintained by leap segmentation could be used in mobile traffic scene applications for improved scene analysis.

The usefulness of applying this novel view of image segmentation in the preprocessing stages of a high-level vision application was evaluated and compared with existing segmentation approaches. Through the evaluation of both accuracy and efficiency objective functions, it was demonstrated that the performance of a high-level image layout reconstruction task can be dramatically improved by applying the leap segmentation technique during preprocessing. Leap segmentation provides layout

applications with reliable segmentations into fewer regions that are unconstrained by noise and provide meaningful spatial support for scene layout analysis, allowing more efficient estimation of overall scene structure. In addition, leap segmentation exhibits execution times 10x-15x times faster than the state of the art.

The next contribution of this research extends the leap segmentation algorithm to process multiple consecutive frames in time (video) with the goal of maintaining region boundary continuity between image frames. A temporal analysis study of this multiple-frame leap segmentation is essential in evaluating region continuity and segmentation stability over time.

CHAPTER 3

LEAP SEGMENTATION IN VIDEO ANALYSIS

3.1. Introduction and Related Work

Over the past decade research into employing vision processing in intelligent vehicle systems has grown extensively. Computer vision systems can be used to analyze traffic scenes and alert drivers of potentially dangerous events as they occur in real time, thus increasing the safety of road ways. Intelligent vehicle systems are mobile, requiring vision applications to be both accurate and efficient in their implementation for successful operation in this resource-constrained, real-time environment.

Many vision applications apply image segmentation techniques during preprocessing to reduce image information for increased processing efficiency. Multiple-frame segmentation, also referred to as spatio-temporal or video segmentation, has been studied a great deal and is an important step in many video analysis applications for identifying and tracking specific features as they move through a scene. In its most simple form, multiple-frame segmentation can be achieved by applying a traditional single-frame segmentation approach to each individual frame in a sequence. Each frame is segmented separately and the segments mapped between frames. However, segmentation results could vary drastically between frames, making it difficult to maintain temporal continuity from one frame to the next with this approach [64].

Video segmentation has been applied in many vision applications including video compression and video indexing and retrieval [39]. Many video segmentation techniques are designed to operate off-line, requiring all frames in the input video sequence as input

[41]. Since future frames must be known, these approaches cannot be applied in real-time applications where only current and past frames are available. On-line approaches exist in the literature, but are fewer in number. These methods are limited to processing past frames and often use Kalman filtering to track segments over time [49]. Paris et al. [64] use isotropic diffusion and Gaussian convolution to achieve real-time performance using only past frame data. However, this approach has limited accuracy when segmenting fast-moving objects.

Methods for multiple-frame segmentation, surveyed in [57], can largely be grouped into three categories regardless of their on-line or off-line behavior: spatial-then-temporal methods, temporal-then-spatial methods, and joint spatial-temporal methods. Spatial-then-temporal methods [27], [36], [72], [79] first segment a frame spatially. They then track regions in the segmentation over time. These methods conceptually extend single-frame segmentation to operate in the temporal domain. Methods in this category can operate either on-line or off-line. Temporal-then-spatial methods [3], [7], [23], perform temporal segmentation first by monitoring several points to obtain their movement trajectories in the image sequence. These trajectories are then grouped together using spatial motion segmentation. Methods in this category require information from future frames for processing and thus must be implemented off-line. Lastly, joint spatial-temporal methods [25], [39], [70] study the spatial and temporal dimensions jointly as a single volume for segmentation. Methods in this category are also inherently off-line because they require knowledge of future frames.

A highly efficient, on-line method for multiple-frame segmentation, called *video leap segmentation*, is presented for use on embedded and mobile platforms where

processing speed is critical (Forsthoefel et al.) [33]. This novel approach extends the fast, single-frame leap segmentation approach presented in Chapter 2 to develop an efficient, multiple-frame segmentation approach that accurately tracks segments between consecutive input frames and successfully maintains temporal segmentation continuity. With this approach, segmentations for each frame are generated quickly without segmenting each frame individually, which is computationally expensive. As each consecutive image frame is processed, the scene's segmentation is evolved to continuously track objects as they move through the mobile scene. This approach is evaluated using moving-camera traffic sequences captured on congested, multi-lane highways. The captured GTTraffic dataset sequences (Forsthoefel et al.) [32] contain fast-moving traffic events, such as vehicles quickly swerving into the driver's lane. These sequences are made publicly available as part of this research to motivate and evaluate vision-based approaches to improving highway safety.

In this chapter, the video leap segmentation approach is introduced for generating fast, stable segmentations of images in mobile video sequences. This chapter is organized as follows. First the fast video leap segmentation approach is described in Section 3.2. Then the application of the video leap segmentation approach to the task of salient segment transformation detection is demonstrated in Section 3.3. Quick detection of salient segment transformations in mobile scenes could be highly useful in an intelligent vehicle system, aiding driver alert systems in quickly detecting dangerous traffic situations that may require immediate driver attention. Trial results, discussed in Section 3.4, demonstrate that with little added computation, video leap segmentation can be used

for salient region detection in traffic scenes with high accuracy. Section 3.5 concludes this chapter and discusses future work.

3.2. Fast Video Leap Segmentation

The fast, multiple-frame leap segmentation approach (Forsthoefel et al.) [33] is an extension of the single-frame leap segmentation approach presented in Chapter 2. In the video leap segmentation method, the initial leap segmentation data structures are exploited for efficient detection of segment changes in subsequent frames. Specifically, structured lists of tile sets indicating tile cell membership are used to quickly compare pixels with surrounding segmentation cell assignments to detect slight segment shifts between frames. This reduces comparisons between pixels in consecutive frames to be on the order of the number of cells in a tile set which concisely represents the color neighborhood of a pixel instead of on the order of the total size of the pixel neighborhood. This significantly reduces the number of comparisons required to determine overall segment movement between frames.

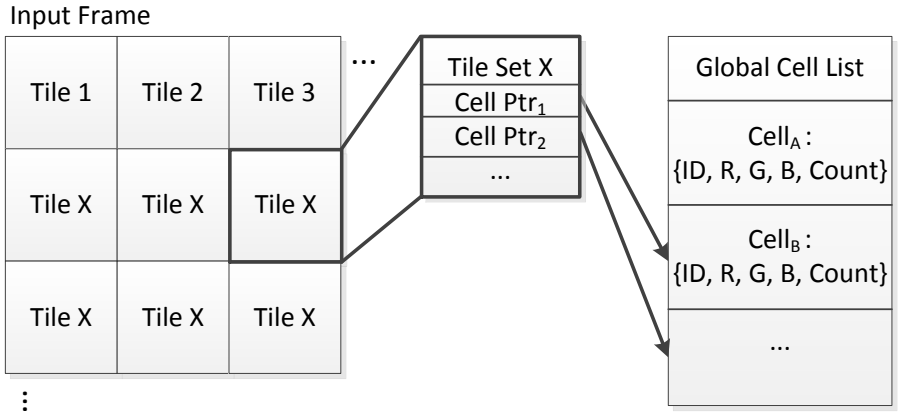


Figure 31. The initial leap segmentation passes a global cell list (right) and a list of tile cell sets (left) for each tile in the discretized image.

After the initial leap segmentation step is completed, the segmentation results are passed to subsequent frames in the form of three data structures: a global cell list, a region map, and a comprehensive list of tile cell sets. The global cell list, shown in Figure 31 (right), contains the color information (RGB) and presence count for each color segment (cell) identified in the image. The region map contains a mapping from each pixel in the image segmentation to its corresponding color cell on the global cell list. Lastly, the list of tile cell sets generated during leap segmentation and shown in Figure 31 (left), contains, for each discretized tile in the image, a set of cell pointers to the global cell list to indicate cell membership of CL-similar pixels within tiles. Cell pointers are not duplicated in individual tile sets, so that a single tile set contains a condensed list of cell pointers to the global cell list.

The data structures provided by leap segmentation are leveraged to yield a fast, resource-efficient approach to the temporal tracking of regions in subsequent frames as follows. Let *f(x, y, t)* denote a frame in the input video sequence at time *t*. Let *s(x, y, t-1)* denote the video sequence segmentation cell assignments obtained from processing the previous sequence frame (held in the region map of the segmentation). Note that *f* contains all pixels in the current input frame ($f(x,y,t) = pixel_{x,y,t}$) while *s* holds the global segmentation cell assignment for each pixel location ($s(x,y,t) = cell_{ID,t}$). Let N_x and N_y denote the number of horizontal and vertical tiles in the discretized image, respectively. Then for pixel location *(x,y)* one can define:

$$I(x,y) := (T_y \times N_x) + T_x \text{ , where } T_x = \frac{x}{\lambda} \text{ , and } T_y = \frac{y}{\lambda} \text{ , } \quad \textbf{(7)}$$

where T_x and T_y are the horizontal and vertical tile indices for the chosen pixel location, respectively, and *I(x,y)* holds the computed tile index, pointing to the tile covering pixel location *(x,y)* in the image. Let $E(P_A,P_B,\tau)$ (Equation 2) define the CL-similar relation (Equation 1) between two RGB values (P_A, P_B) for some chosen threshold τ (τ = 30 was chosen in the current implementation). A review of Equations 1 and 2 from Chapter 2 is given below for clarity:

$$\max\begin{pmatrix}|R_1 - R_2|\\ |G_1 - G_2|\\ |B_1 - B_2|\end{pmatrix} \leq \varepsilon \ . \qquad \textbf{(1)}$$

$$E(P_A, P_B, \tau) = \begin{cases}1, \text{ if } P_A, P_B \text{ are } \textit{CL-Similar}\\ 0, \text{ otherwise}\end{cases}. \qquad \textbf{(2)}$$

Let *T(I(x,y), t-1)* be the complete list of tile cell sets obtained during segmentation of the previous frame. To begin, set *T(I(x,y), t) = T(I(x,y), t-1)*. The segmentation of the current video sequence frame *s(x,y,t)* is obtained using the following three-step method. First, the current frame is directly compared with previous segmentation cell assignments for matching within some threshold τ:

$$s(x, y, t) = s(x, y, t-1), \text{ if } E(s(x, y, t-1), f(x, y, t), \tau) \ . \qquad \textbf{(8)}$$

If a match is not obtained from this initial comparison, then the search window is widened to include those cells, *Z*, in the tile set of the segmentation, $T_C(Z)$, which contains the current pixel location *(x,y)*:

$$T_C(Z) := T(I(x, y), t) \ , \qquad \textbf{(9)}$$

then,

$$s(x,y,t) = T_C(z) \text{ , if } \exists\, z \in Z : E(T_C(z), f(x,y,t), \tau) \quad . \tag{10}$$

If a match with the previous segmentation is still not forthcoming, the search is again widened to include those pixels in the tile sets of the tiles in the neighborhood of the current tile. Define $T_N(Z)$ as the list of cells, *Z*, in a neighboring tile that contains the neighboring pixel locations (x_N, y_N):

$$T_N(Z) := T(I(x_N, y_N), t) \quad , \tag{11}$$

then,

$$s(x,y,t) = T_N(z) \text{ , if } \exists\, z \in Z : E(T_N(z), f(x,y,t), \tau) \quad . \tag{12}$$

If a cell match is detected in a neighboring tile set, the current tile set is updated to include a pointer to the matched global list cell for fast future comparisons.

The implementation workflow for video leap segmentation is shown in Figure 32. The three step method makes use of the data structures provided from the initial leap segmentation and updates these structures to represent changes to the segmentation that occur between consecutive image frames. In a post-processing step, groups of pixels that do not match between the frames are labeled as new object candidates. If a sufficiently large and chromatically similar group of nonmatching pixels is present, a new segment is created for these pixels to represent the new scene object. The implementation data structures passed between frames during segmentation of the video sequence are designed for optimal resource usage. The segmentation region map and list of tile cell sets contain only pointers to the global cell list. Therefore, segment information is stored

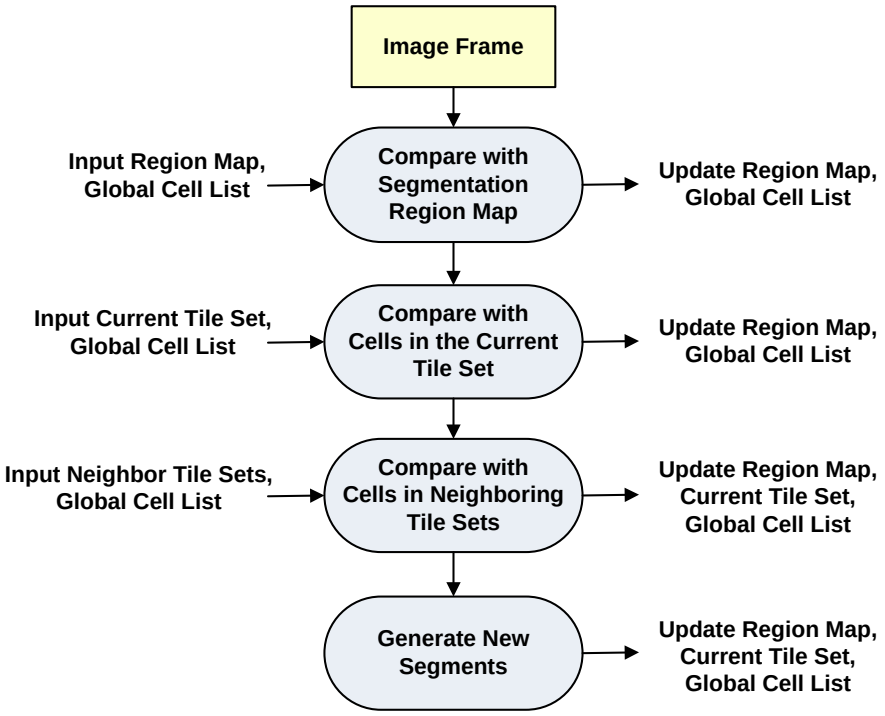


Figure 32. Workflow of the fast, resource-efficient video leap segmentation algorithm.

only once in the global cell list, while the region map and list of tile cell sets convey segmentation structure using lightweight pointers.

The three step method allows for fast comparisons between the current frame and the previously obtained segmentation with an increasingly large search space. The search space can be easily constrained with this design to conform to specific application goals. In the current implementation, the search space is limited to tiles directly neighboring the current tile set to reduce computation time. The ability of this segmentation procedure to produce stable segmentations of video sequences is evaluated in Section 3.4.1 over various traffic scenes.

3.3. Recognition of Salient Segment Transformations

In addition to generating fast, stable segmentations of images in video sequences, the presented method for video leap segmentation can be applied to the task of rough salient segment transformation detection for alerting potential drivers of important scene changes that may affect future steering decisions.

Salient transformation detection is performed using a fast, two-frame segmentation comparison. Comparing segmentation assignments between frames can be a slow task if performed on the pixel level. Instead, a cell-level comparison is performed using the region map cell assignments obtained during video leap segmentation. As described in Section 3.2, the video leap segmentation approach quickly identifies shifted segments between frames using the tile set data structures introduced by leap segmentation. Using this video leap segmentation approach, segment movement is quickly identified over a large search window without the onerous step of directly comparing each pixel in the search window. Video leap segmentation outputs a region map of cell assignments at each pixel location. These outputted cell assignments correctly model the positions of shifted segments and accurately represent overall scene structure despite scene changes between frames. The successful utilization of the video leap segmentation approach in a simple recognition task is demonstrated with the following approach for recognition of salient segment transformations.

Let $r_t(x,y)$ denote a region map at time t containing the segmentation cell assignments as indices into the global cell list. Let $r_{t-1}(x,y)$ denote the region map cell assignments for the previous sequence frame segmentation. These two region maps are compared to determine the locations of salient segment transformations in the scene.

First, the direct spatial neighborhood of each pixel is evaluated to form a movement vector *mv(x,y,t)* for each pixel location *(x,y)* at time *t*. Let *n(P)* represent the neighborhood of pixel *P*. The cell assignments of all pixels in *n(P)* are compared to *P* to form the movement vector for *P*. Movement vector assignments are binary indicators of cell assignment comparisons. The movement vector for *P* holds a binary value for each pixel in *n(P)*. If neighboring cell assignments match *P*, the movement vector is assigned a binary 0 for those locations. Otherwise, the movement vector is assigned a binary 1 for those non-matching cell assignments in the neighborhood of *P*.

The computed movement vectors are compared between consecutive image segmentation region maps (r_t , r_{t-1}) in the sequence for fast segment transformation detection. Figure 33 shows an example of movement vector assignments for two consecutive image frames at various pixel locations. The binary movement vector assignments are evaluated in clockwise order (i.e. N,E,S,W). A nonzero movement vector denotes an edge pixel in the segmentation. Cell movement is detected by comparing the movement vectors for edge pixels in the consecutive frames. These calculations are performed using quick binary comparisons of vector values. A change in movement vector assignments from 1 to 0 in the East direction indicates movement has occurred East. However, a change in the movement vector assignments from 0 to 1 in the East direction would indicate movement has occurred in the West direction. If the consecutive movement vectors for a pixel are nonzero and unchanging between frames, this indicates a stable edge pixel in the scene.

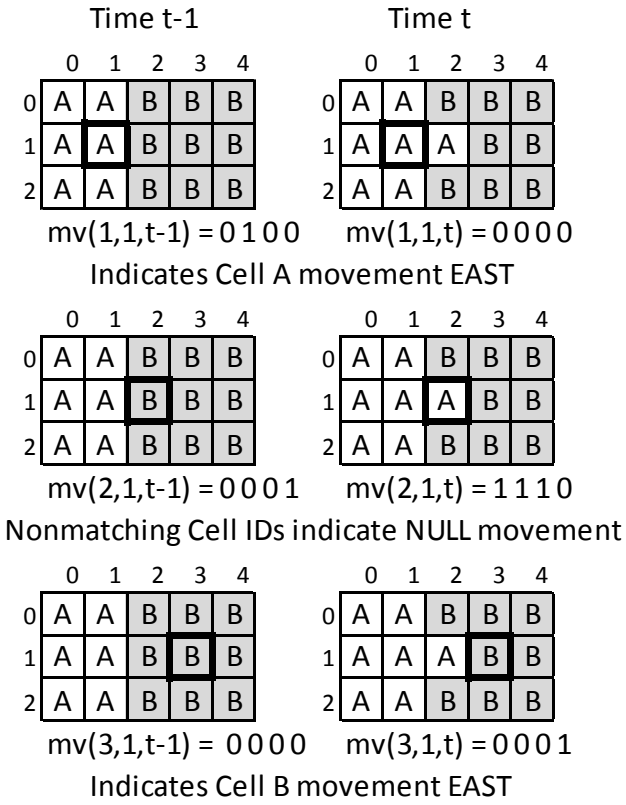


Figure 33. Example of binary movement vector assignments at various pixel locations. Assignments are evaluated in clockwise order (i.e. N,E,S,W).

In the example in Figure 33 (top), the movement vectors indicate that a Cell A pixel moves East between frames *t-1* and *t*. Another pixel in Figure 33 (middle) changes cell membership between frames (from Cell B to Cell A) so the computed movement vectors are ignored and no movement is recorded for either cell. Lastly, in Figure 33 (bottom) a Cell B pixel is detected moving in the East direction. These directional classifications are recorded for each cell in the segmentation. The overall direction of transformation of a segmentation cell is calculated as the maximum present transformation direction detected in the cell over all cell pixels. Furthermore, a cell is identified as "stable" if the detected number of stable edge pixels in the cell exceeds the

total number of non-stable edge pixels present in the cell.

This simple calculation and binary comparison of movement vector assignments allows for a quick and comprehensive assessment of cell transformations between consecutive frames. The presented method evaluates cell transformations only at the edges of segmentation cells where cell movement is most identifiable and avoids the complicated and cost-inefficient method of determining total cell movement over all cell member pixels. This simple, binary method can be implemented using low-cost integer operations. The utilization of video leap segmentation at the base of this approach for the detection of segment shifts across temporal and spatial bounds facilitates the fast and efficient detection of salient segment transformations in video scenes.

3.4. Experimental Results

The discussed multiple-frame leap segmentation approach is implemented in the C programming language and is developed in a Linux environment. A publicly available set of moving-camera traffic scene sequences collected at Georgia Tech is used in these evaluation experiments (see Figure 34). The captured GTTraffic dataset sequences (Forsthoefel et al.) [32] contain fast-moving traffic events such as vehicles quickly swerving into the driver's lane. These sequences are being made publicly available as part of this research to motivate and evaluate vision-based approaches to improving highway safety.

These scene collections were captured at Georgia Tech using a forward-mounted Kodak Zi6 on an automobile dashboard. This camera provides a fixed focus, fixed aperture, and fixed field-of-view with electrically controlled gain and sensitivity. The

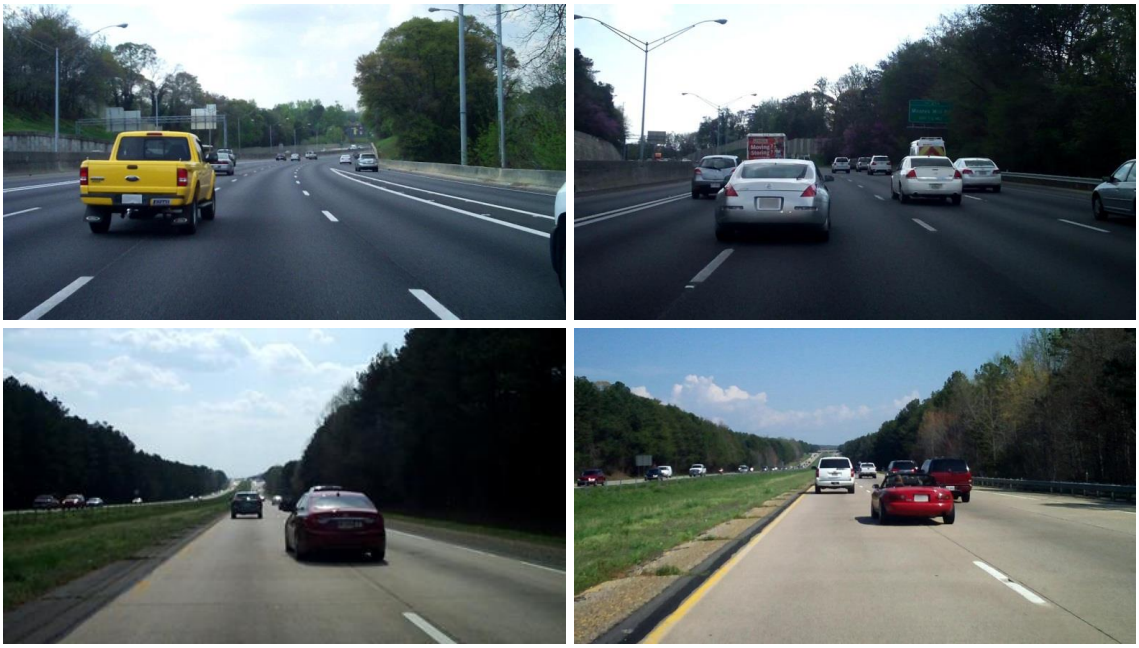


Figure 34. Sample images from the GTTraffic dataset [32].

images extracted from the captured mpeg4 videos are at a resolution of 1280 x 720 pixels. The sequences were collected at a frame rate of 32 fps.

3.4.1. Video Leap Segmentation Stability

In this evaluation, the quantitative objective function *nonmatching pixel percentage* is used to assess segmentation stability. It is measured as the percentage of image pixels in the video segmentation output that are not CL-similar to their original image color. Let us recall Equations 3 and 4 from Chapter 2:

$$P_{NM} = P_{TOTAL} - \left[\sum_{i} E(P_{ORIG}[i], P_{SEG}[i], \tau)\right], \quad \textbf{(3)}$$

$$Nonmatching\ \% = 100 \times \frac{P_{NM}}{P_{TOTAL}}. \quad \textbf{(4)}$$

Calculation of the nonmatching pixel percentage is shown above in the review of Equations 3 and 4. The equivalence function E is given in Equation 2 and applies the CL-similar relation in Equation 1 to assess pixel affinity. P_{NM} is the number of pixels in the final segmentation which are not CL-similar within the matching threshold τ to their original image color (τ = 30 was used in all experiments) and P_{TOTAL} is the total number of pixels in the image. P_{ORIG} holds the original input image, and P_{SEG} holds the pixels in the output segmentation.

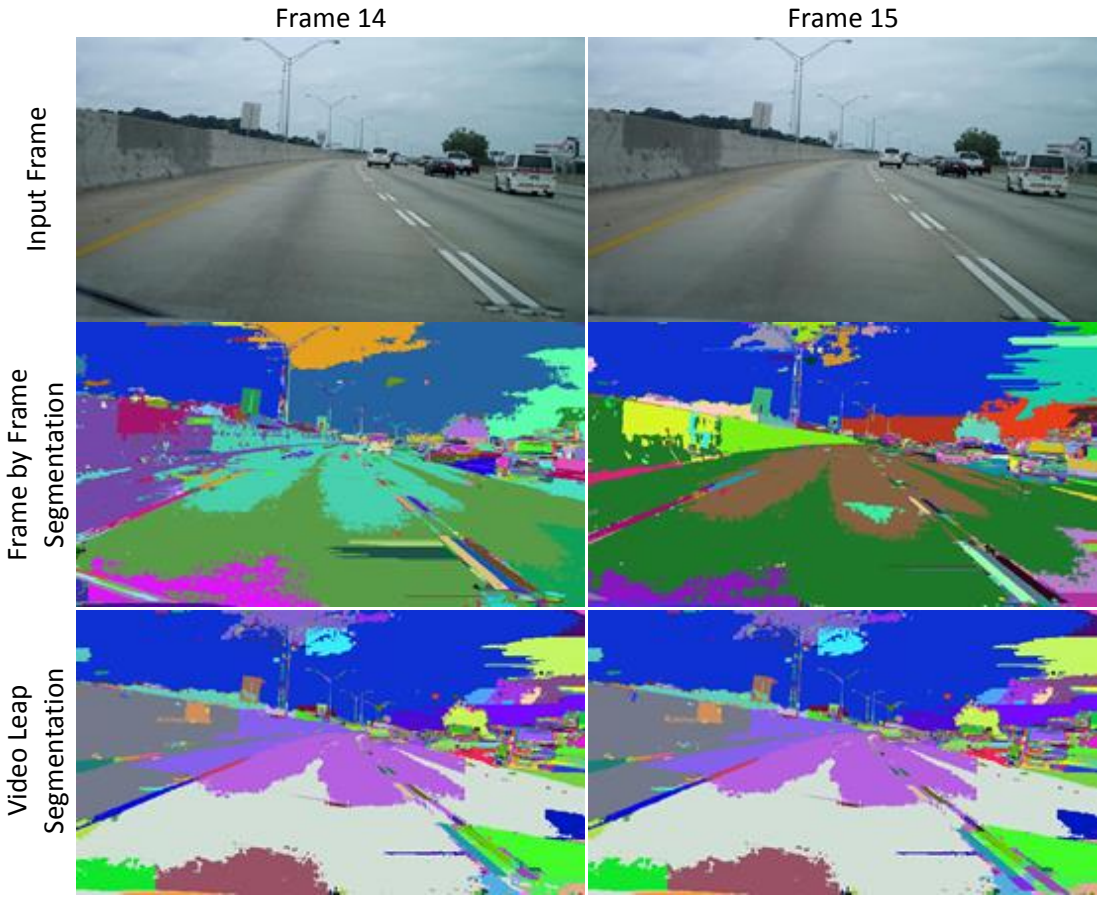


Figure 35. Video leap segmentation results for two consecutive image frames. A colorized representation of segmentations is given to show region membership more clearly. A frame by frame approach (middle) produces segmentations that change rapidly between frames. Applying the video leap segmentation approach (bottom) allows easy maintenance of temporal coherence between frames.

A high accuracy image segmentation result achieves a low nonmatching pixel percentage, indicating that a small number of pixels have been assigned to a region color that is significantly different from their original color. This metric is a good measure of the preservation of scene integrity during the segmentation process.

Figure 35 qualitatively displays the stability of video leap segmentation results when compared to a frame-by-frame segmentation approach (in which each frame is segmented separately). To help discern region membership, artificially colorized segmentation images are given. Contrasting color assignments show region pixel membership. With a frame-by-frame segmentation approach, segmentation results can vary drastically between frames, making it infeasible to maintain segment continuity from one frame to the next. The video leap segmentation approach successfully matches segments across temporal bounds, maintaining temporal coherence between the input sequence frames.

The input parameters used to generate the initial leap segmentation at the base of the video leap segmentation approach are λ=2, ε=20, and α=50. The equivalence (ε) and minimum size (α) parameters are chosen based on optimal performance determined previously in Section 2.5. A minimal adjacency parameter is chosen (λ=2) in order to better facilitate salient segment transformation detection, discussed in Section 3.3, on the collected traffic scenes. A larger adjacency parameter input would allow larger changes in segment movement between frames to be detected, but may reduce the accuracy of detection of small segment shifts between frames. If input frames are spaced far apart in time, the use of a larger tile size may be appropriate. However, as proof of concept, a minimal adjacency parameter was chosen for these experiments because GTTraffic

Table 2

Video Leap Segmentation Stability

Average Nonmatching Pixel Percentages Over 200 Frames					
Traffic1	**Traffic2**	**Traffic3**	**Traffic4**	**Traffic5**	**Traffic6**
1.817%	1.424%	0.896%	0.716%	3.829%	2.075%

dataset images were collected at a high frame rate, causing small segment shifts to dominate these scenes. A full video leap segmentation parameter variation assessment is planned.

To quantitatively assess the stability of video leap segmentation over time, the nonmatching pixel percentage is calculated over all image frames in several GTTraffic sequences. Table 2 shows the average nonmatching pixel percentages over six different traffic sequences, each containing 200 frames. Each of the chosen input sequences contains substantial scene changes, such as the introduction of new vehicles into the scene. Video leap segmentation produces nonmatching pixel percentages of less than 4% and as low as 0.7% when processing these input image sequences despite the long sequence length and the frequent introduction of new objects into the scenes. The developed approach is able to maintain this high level of stability by adapting the sequence segmentation at each new input frame and by carefully introducing new scene segments when new objects appear in the scene.

3.4.2. Salient Segment Transformation Detection

The salient segment transformation detection approach outlined in this chapter is designed to be very fast in its execution, using the output from the provided video leap segmentation approach to quickly determine rough areas of saliency in an input image

scene. No statistical processing or high-level model development is performed to produce these results. Those more computationally expensive approaches for tracking regions could later be added on top of this approach to clean up the detection process and remove noise.

Figure 36 shows the approach output for recognition of salient segment

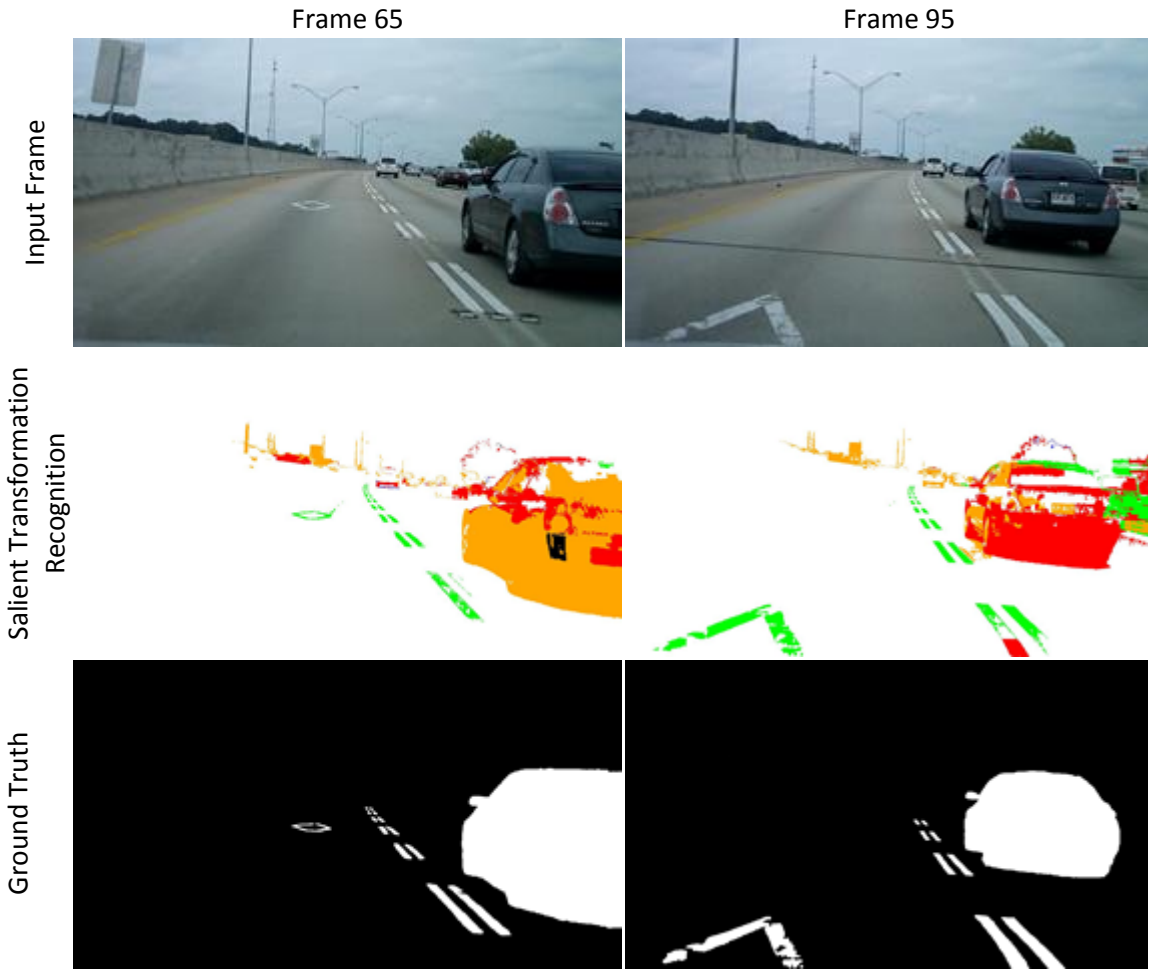


Figure 36. Salient segment transformation recognition results for two frames of an input video sequence. Top: Input frame. Middle: A colorized representation of detected salient segment transformations in the scene (White = Stable, Red = North, Green = South, Blue = East, Orange = West, Black = Movement detected in all directions (segment grew in size in the scene). Bottom: Ground truth images, salient pixels plotted in white.

transformations. The salient transformation result images in Figure 36 (middle) show those pixels that were identified as salient by the developed algorithm. Stable pixels are shown in white, while salient pixels are colored based on their detected direction of transformation. Corresponding ground truth images are also given in Figure 36 (bottom), with salient pixels shown in white. These results demonstrate that with little added computation, the video leap segmentation results can be used for rough salient region detection in traffic scenes with surprising accuracy. The salient detected areas can be passed to a higher-level vision system for determining the appropriate response to the detected salient regions. Reducing higher-level processing to the detected salient areas using this quick approach has the potential to significantly reduce the processing time of scene understanding approaches.

Due to the rough nature of the developed method for detection of salient transformations, the results can contain some noise. This is to be expected, and further research will explore post-processing schemes to alleviate this. Several ground truth images were created and compared with the salient transformation detection output to quantify the accuracy of this approach. On average, the developed approach correctly identifies ~80% of salient ground truth pixels as salient in the output. In addition, the approach correctly identifies over 95% of non-salient ground truth pixels as non-salient in the output. This indicates that developed approach achieves a false negative rate of about 20% while keeping the false positive rate below 5%. The accuracy achieved by the presented algorithm is acceptable in achieving the proposed goal of a rough salient transformation detection system. Post-processing steps can be taken to improve the transformation image and further reduce false positives and false negatives. In particular,

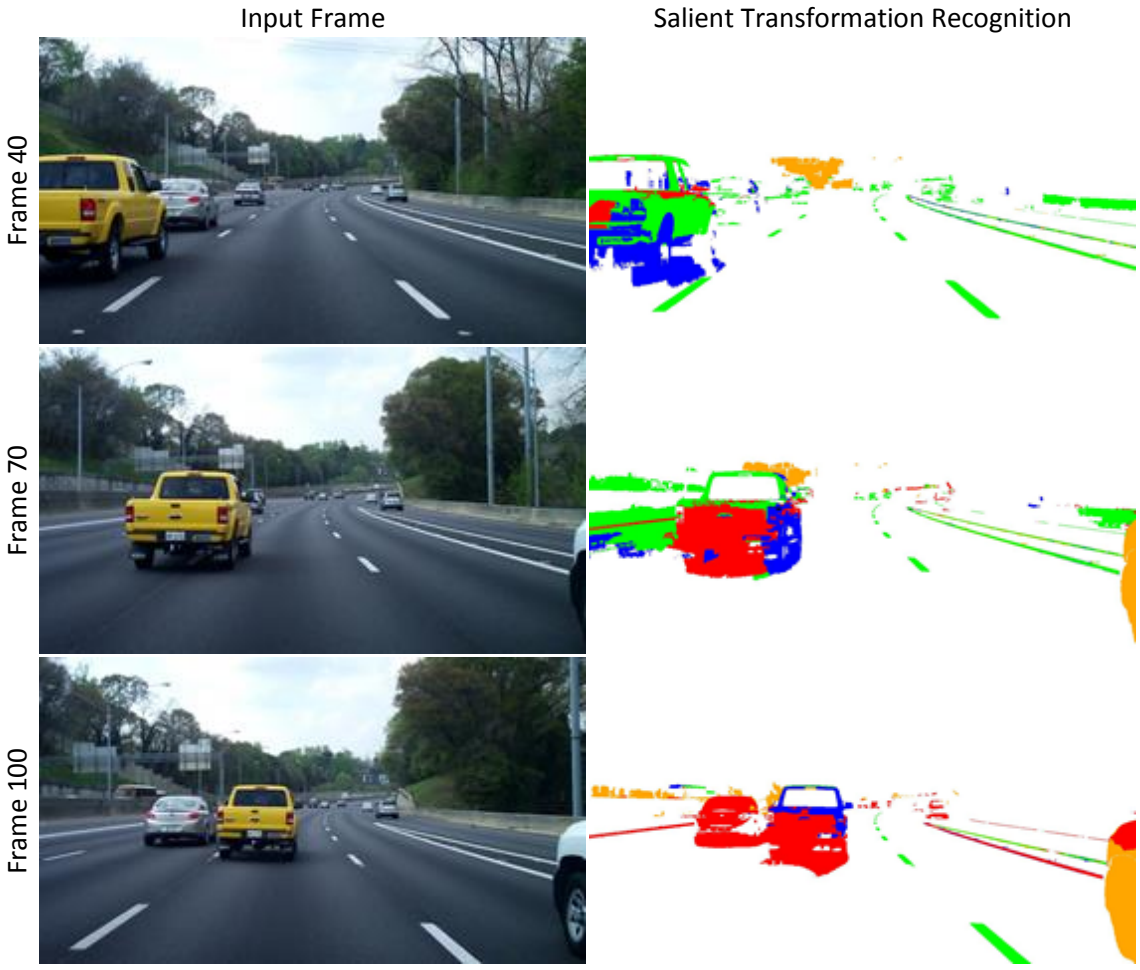


Figure 37. Salient segment transformation detection results for a video scene in which a vehicle rapidly enters the driver's lane.

false negatives arise most often in the reflective windows of vehicles in the scene. These could be removed with the implementation of a post-processing region-fill step to detect and correct these holes in the recognition output.

Figure 37 shows an example sequence of frames of a vehicle rapidly swerving into a driver's lane. The developed salient transformation recognition technique is able to quickly identify and track the rapidly moving car. This technique could be used in

preprocessing to aid driver alert systems in quickly detecting dangerous traffic situations such as these that may require immediate driver attention.

3.4.3. Run-Time Analysis

The outlined video leap segmentation approach and salient transformation detection system are evaluated using a 2.13 GHz Intel Core I3–330M processor running 64-bit Ubuntu 10.04. The algorithms were not parallelized or otherwise altered for the platform. Sequences from the GTTraffic dataset which contain images 1280x720 pixels in size are used in this evaluation.

The implementations of the discussed algorithms have not yet been fully optimized for efficient execution. A preliminary investigation of execution performance is presented here, pending a complete review of optimization capability. Even without an in-depth optimization of implementation, the video leap segmentation approach proves computationally efficient. An analysis of trial results reveals that the current implementation runs at an average ~90 ms per frame, or over 11 fps. The salient transformation detection system also shows excellent execution efficiency in preliminary tests, executing in an average time of ~20 ms per frame.

There are several possible avenues for optimization of the presented approach, including converting the single-core video leap segmentation approach to a multi-core platform using parallel processing. An investigation of this, along with additional optimization techniques is planned for future work.

3.5. Conclusion

This chapter presents a novel approach to multiple-frame segmentation, called video leap segmentation, for use on embedded and mobile platforms where processing

speed is critical. Through the evaluation of both accuracy and efficiency objective functions, it was demonstrated that the provided approach successfully tracks segments across spatial and temporal bounds, generating fast, stable segmentations of images from moving-camera video sequences. The approach was then applied to the task of salient segment transformation detection. The resulting salient transformation recognition technique is able to quickly identify and track the rapidly moving, salient objects in input video scenes. This technique could be applied in preprocessing to aid collision avoidance systems in quickly detecting dangerous traffic situations that may require immediate driver attention.

Several possible avenues of future work have been identified, including an in-depth parameter variation analysis of video leap segmentation parameters, a full optimization of the video leap segmentation approach implementation, and parallelization of the approach, targeting a multi-core platform.

CHAPTER 4

EMBEDDED, MULTI-CORE LEAP SEGMENTATION

4.1. Introduction

Many vision applications apply image segmentation techniques during preprocessing to reduce image information for increased processing efficiency. However, the processing times of most existing single-frame image segmentation approaches exceed input camera frame periods when processing high-resolution images, making them impractical for use in real-time environments.

The goal of this research is to achieve real-time (>25 fps) image segmentation execution performance on a commercially-available CPU with multiple processing cores that does not require specialized hardware. To that end, this chapter first introduces a highly optimized serial implementation of the *leap segmentation* approach developed in Chapter 2. Numerous parallelization techniques are then applied to different portions of this segmentation approach to achieve further speed-up on a multi-core system. The final, parallel leap segmentation implementation easily achieves real-time execution when processing high-resolution images.

This chapter builds directly upon the results of previous chapters. Leap segmentation, developed in Chapter 2, is a novel approach to single-frame segmentation which forms homogeneous regions of pixels that need not be spatially contiguous. Leap segmentation is designed for use in embedded, resource-constrained environments while maintaining accuracy comparable to traditional approaches. The design of leap segmentation lends itself to an efficient implementation as shown in Chapter 2, but can

be optimized further. This chapter outlines an optimized serial implementation of leap segmentation which achieves frame rates of more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images (Forsthoefel et al.) [35].

This serial implementation of leap segmentation proves useful in many embedded, resource-constrained environments where processing speed is critical. However, "real-time" processing standards in image processing vary widely. More stringent standards on real-time frame-rates typically enforce matching to collection frame-rates without frame-skipping (processing every other frame or every third frame). Under these standards, an approach must run at the least at the image collection rate of the source camera. Therefore real-time processing frame rates typically range from 25 to 30 fps at a minimum. The highly optimized serial implementation of leap segmentation presented in this chapter achieves real-time processing on 640x360 images (80 fps) but falls short of these real-time standards on high resolution (1280x720) images, processing at just 20 fps.

To achieve real-time execution of leap segmentation on high-resolution images, a multi-core leap segmentation implementation is developed in this chapter. Numerous parallelization techniques are applied to different portions of the leap segmentation algorithm to achieve further speed-up. The steps taken to parallelize each leap segmentation subtask are described in detail. The developed multi-core leap segmentation implementation achieves frame rates on commodity hardware of more than 29 fps on 1280x720 images using two threads and more than 31 fps when using four threads, thus meeting even the more stringent real-time processing standards (Forsthoefel et al.) [35].

This chapter is organized as follows. Related work in the field of image segmentation is outlined in Section 4.2, including information on real-time segmentation efforts. Section 4.3 discusses overall leap segmentation implementation workflow and the framework of leap segmentation resources. Both the highly optimized serial and parallel implementations of leap segmentation are presented and analyzed in Section 4.4. Performance evaluations of the developed implementations on both high performance and resource-constrained platforms are presented and discussed in detail in Section 4.5. Section 4.6 concludes this chapter.

4.2. Related Work

This section summarizes related work in the field of image segmentation and then describes recent advancements in fast, multi-core image segmentation. For a more in-depth description of general single-frame image segmentation related work, please refer back to Chapter 2.

Image segmentation has been widely researched, resulting in several broad classes of algorithms including region-based, feature-space clustering, and graph-based segmentation. The region-based segmentation category includes all “region-growing” and “split-and-merge” techniques. The watershed approach [77] is a popular example of region-based segmentation. Segmentation methods that use feature-space clustering attempt to find modes (clusters) in a distribution by using each image pixel’s features as sampled data from the distribution’s probability density function. Mixture of Gaussians clustering with expectation maximization [26] and mean-shift clustering [22] fall into this category. In graph-based segmentation, an image is represented as a weighted, undirected graph. Popular graph-based approaches include normalized cuts [71] and efficient graph-

based image segmentation (EGBIS) [30]. A detailed review of image segmentation research can be found in [74].

The mean-shift clustering technique [22] and the efficient graph-based image segmentation technique (EGBIS) [30] mentioned above are two well-known and popular segmentation algorithms. According to Pantofaru and Hebert [63], output segmentations from mean-shift correspond well to human perception. A disadvantage is its sensitivity to parameter change and the necessity for input parameter tuning to obtain good segmentations [86]. In addition, mean-shift is computationally expensive making it too slow for real-time applications. This is due in part to the expensive sliding-window approach it applies to image pixels during processing. Several techniques for improving mean-shift have been proposed [17], [20], [80]. For example, Christodias et al. [20] proposed combining mean-shift with edge detection to increase segmentation accuracy in EDISON. However, these algorithms often require on the order of minutes to process one second of video [65]. The popular graph-based segmentation technique, EGBIS [30], is considered to be state of the art in computational efficiency [28], [65]. It uses pair-wise component comparisons to segment an image in $O(mlogm)$ time, where m is the number of graph edges. A drawback to this method is its sensitivity to its input parameter k and its tendency to create small, unneeded regions at the borders of valid image segments.

Modern demand for real-time image processing algorithms has inspired several research efforts in fast, multi-core image segmentation. In recent research, Abramov et al. [1] use a GPU for parallel image segmentation but achieve just 30 fps frame rates on small (256x320) images. In [58], Meribout and Nakanishi present an approach which requires a dedicated parallel hardware architecture to achieve real-time segmentation

performance. In [43], Happ et al. propose a multi-core region-growing approach for use on high resolution images. However, there is still room for improvement as this approach requires on the order of tens of seconds to process a single image. In this chapter, the goal is to achieve real-time (>25 fps) image segmentation execution on a commercially-available CPU with multiple processing cores that does not require special hardware.

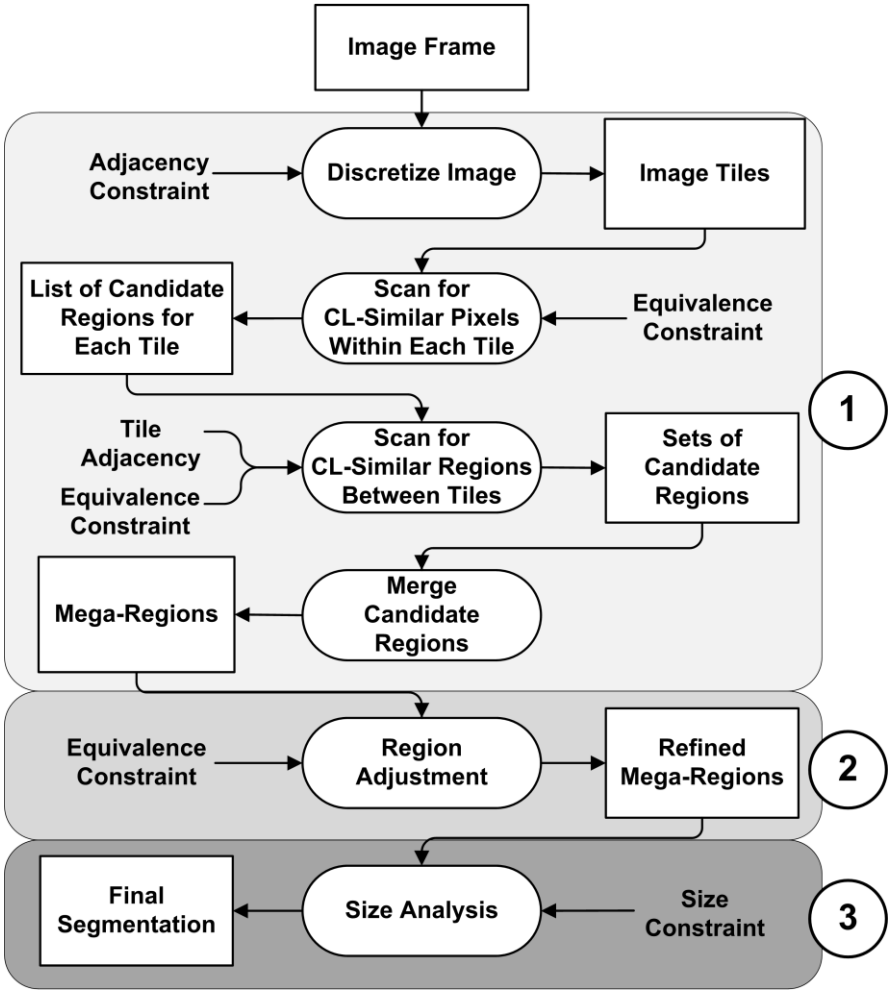


Figure 38. Workflow of the leap segmentation algorithm broken down into three subtasks for parallelization: 1. region building, 2. region adjustment, 3. size analysis.

4.3. Leap Segmentation Implementation

The leap segmentation implementation workflow, shown in Figure 38, is partitioned into three main subtasks: *region building*, *region adjustment*, and *size analysis*. In region building, input image pixels are grouped based on leap segmentation adjacency and equivalence constraints to form mega-regions of pixels. During region adjustment, the segmentation output from the region building subtask is evaluated for possible irregular pixel assignments and new regions are synthesized to represent any new scene objects that arise in this evaluation. Size analysis applies the minimum size constraint to mega-regions, appropriately assimilating small regions to nearby mega-regions based on spatial and color similarities. The resulting mega-region list becomes the final segmentation.

A high-level analysis of leap segmentation execution performance can be seen in Figure 39. The chart shows the percentage of processing time consumed by each of the three main subtasks. Region building is by far the most expensive subtask, encompassing

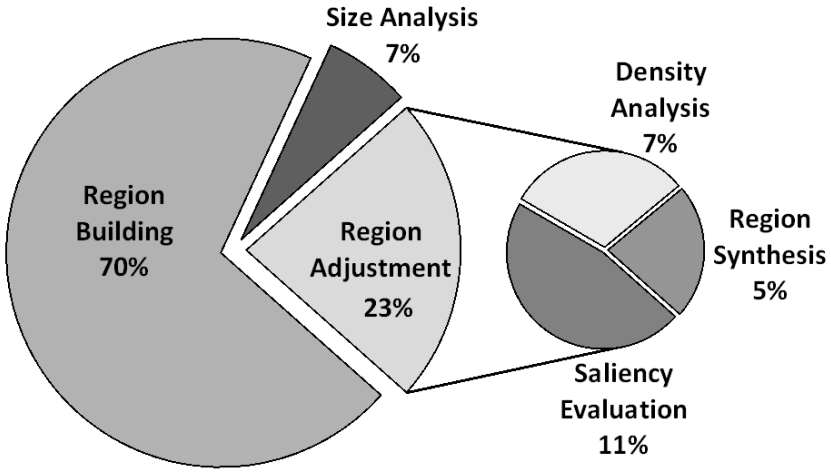


Figure 39. A processing usage chart indicating percentages of leap segmentation time dedicated to each of the three main subtasks: region building, region adjustment, and size analysis.

70% of the leap segmentation execution time. The next subtask, region adjustment, requires 23% of the total processing time. This subtask can be further broken down into its child methods and analyzed separately (as discussed in Section 4.4.2). The third subtask, size analysis, consumes just 7% of the total execution time.

The leap segmentation data structures, displayed in Figure 40 and mentioned previously in Section 3.2, are designed for optimal resource usage. Three structures are used. The *Global Cell List* (middle) contains the color information (RGB) and presence count for each color segment identified in the image. The *Region Map* (right) contains a mapping from each pixel in the image segmentation to its corresponding color cell on the global cell list. The *Comprehensive List of Tile Cell Sets* (left) contains, for each discretized tile in the image, a set of cell pointers to the global cell list to indicate cell membership of CL-similar pixels within tiles.

Segment information is stored only once in the global cell list, while the region map and comprehensive list of tile cell sets convey segmentation structure using lightweight pointers. In addition, as discussed in Chapter 2, pixels within a region contribute their component values to a ratiometric mean via component sums and a pixel

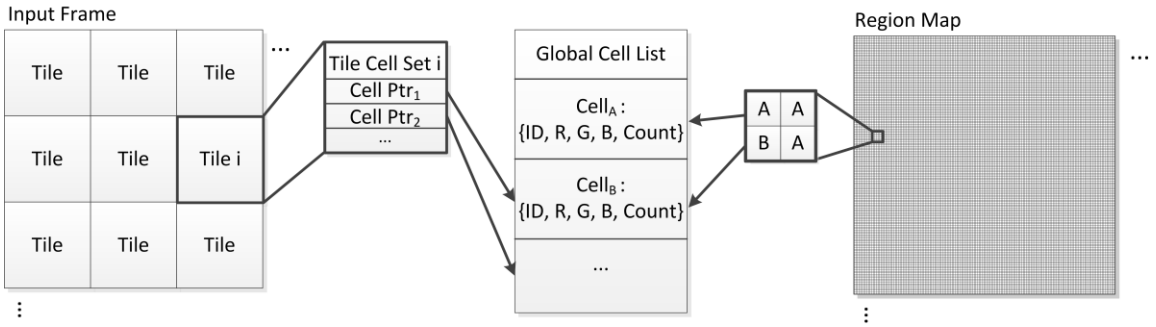


Figure 40. The leap segmentation data structures are designed for optimal resource usage.

count, shown in Figure 10. During segmentation, pixels are compared to the mean component values (e.g. R, G, and B) of candidate regions for rapid analysis of affinity. See Chapter 2 for further leap segmentation algorithm details along with a detailed parameter sensitivity analysis and full comparisons with leading approaches.

In the following sections, highly optimized leap segmentation subtask implementations are presented for single-core platforms. Further research then tests the hypothesis that these subtasks can achieve high speed-up when their base algorithms are parallelized and ported to a multi-core platform (Forsthoefel et al.) [35]. An analysis of each subtask's potential for parallelization is provided along with detailed before-and-after comparisons of the execution rates of these subtasks before and after parallelization.

4.4. Implementation Analysis

This section contains an analysis of the developed fast and resource efficient implementations (both single-core and multi-core) of the leap segmentation algorithm. Each of the three leap segmentation subtasks is analyzed and discussed. Any single-core optimizations within the subtasks are described in detail. Then, each subtask is analyzed for opportunities for parallelization and restructuring for use on a parallel processing system.

4.4.1. Subtask 1: Region Building

As shown in Figure 39, the region building subtask requires the highest percentage of processing time. Since this is also the first task performed in leap segmentation, it is an ideal place to begin the performance analysis.

Serial Implementation

In the developed serial implementation of the region building subtask, the input image is discretized using the adjacency parameter, λ, by dividing it into non-overlapping λxλ square regions called tiles. Each tile is scanned using the CL-similar constraint (Equation 1) to locate candidate regions within each tile. If a pixel is CL-similar to pixels within an existing region, it is added to that region. Otherwise, it forms a new candidate region. After identifying candidate regions within each tile, these regions are compared between neighboring, contiguous tiles. Regions whose mean component values satisfy the CL-similar relation are merged into a mega-region. This process continues until a final set of candidate mega-regions are identified. At this point, all ratiometric component means are locked to fixed component averages that no longer depend on member pixels.

To optimize this serial approach, first the implementation structure must be designed for optimal segment evolution during execution. A two-dimensional image cannot simply be processed for segmentation in row-column order, though this would have promising cache efficiency implications. The segmentation process itself is widely viewed as an inherently sequential mechanism in which segments grow and evolve as more pixels in the image are processed. Pixels within the same image neighborhood depend on each other during segment formation. Therefore, the two dimensions of the image should be traversed at a comparable rate to ensure the highest accuracy in segment growth across the image.

In order to address this issue, the image is traversed and processed along image diagonals starting from the top left corner (origin) of the image to the bottom right corner (see Figure 41). By evaluating tiles in diagonal order, the segmented area of the image

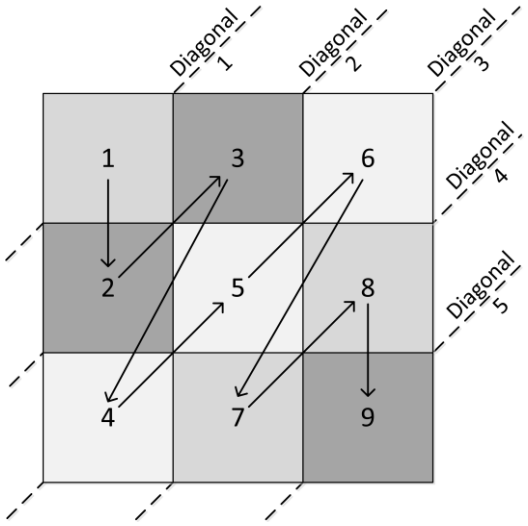


Figure 41. Serial leap segmentation image traversal; the image is processed in diagonal order. The image tile traversal is shown with arrows.

grows in two dimensions simultaneously, facilitating a quick and comprehensive evaluation of pixel adjacency in the image during segmentation.

Parallel Implementation

Within this framework of image discretization and diagonalization, one can identify the dependencies present within the algorithm and determine the best method for tile-level parallelization.

Let D_j be the current diagonal in which tiles are being evaluated for segmentation. Tiles in D_j will compare with neighboring tiles in diagonal D_{j-1} for matching CL-similar regions. In this way, regions are able to move seamlessly across the image as they are pulled from diagonal to diagonal during segmentation. Comparisons with neighboring tiles are performed on those tiles to the north and west of the current tile being processed, as these are the tiles located along a previously evaluated image diagonal. An illustration of tile dependencies is displayed in Figure 42. One can see in this graph that

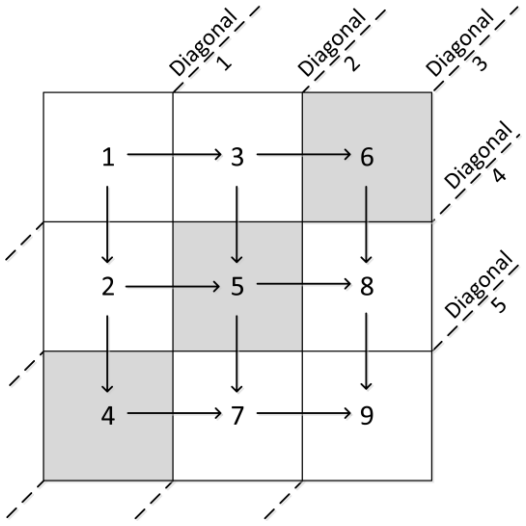


Figure 42. Leap segmentation diagonal dependencies are shown using arrows. In order to process the shaded diagonal tiles, all tiles in the previous diagonal must first be completely processed.

dependencies in execution arise between image diagonals. However, no data-flow dependency exists between tiles within the same diagonal. For example, the shaded diagonal in Figure 42 (D_3) includes the set of tiles {4,5,6}. According to this dependency graph, before processing tile 4 information is needed from tile 2. Similarly, tile 5 needs information from tiles {2,3} before it can be processed.

Tiles within image diagonals can be processed in parallel with low contention. Let N_X be the number of image tiles in the x direction and N_Y be the number of image tiles in the y direction. As described above, the discretized global image domain D is split into image diagonal sub-domains D_j , $j = 0:(N_X - 1)+(N_Y - 1)$. Let i be the number of tiles in subdomain D_j:

$$D_j = D_{j,0} \cup D_{j,1} \cup D_{j,2} \ \ldots \ D_{j,i-1}. \qquad \textbf{(13)}$$

Tiles within each subdomain are processed in parallel. In tests, this form of parallelization does not adversely affect the accuracy of the segmentation output and

avoids the costly contention of threads on resources. It is important to emphasize, as stated before, that diagonals must be processed sequentially; D_{j-1} must be processed before D_j. This level of sequential processing is expected and largely unavoidable, as leap segmentation contains an inherently sequential process in which comparisons between tiles are required to grow segments across adjacency bounds.

Performance evaluations for the region building subtask are given in Section 4.5.3 for both the presented serial and parallel implementations of leap segmentation.

4.4.2. Subtask 2: Region Adjustment

During region adjustment, the segmentation output from the region building subtask is evaluated for possible outliers within regions and new regions are synthesized to represent any new scene objects that arise in this evaluation. An outlier can arise during region building in several situations. In most cases, slow-changing gradients cause region component means to drift, allowing some pixels to fall out of segment CL-similar bounds. This behavior is desired for some applications which favor minimizing the number of segments and the fractionalization of objects (e.g. for the identification of object boundaries). However, this is not the case for applications which require reliable original pixel color representation in the segmentation model. Region adjustment is implemented in order to maintain a standard level of accuracy in leap segmentation output over all scenes.

This section first presents a highly optimized serial implementation of the region adjustment subtask of leap segmentation. Then, the steps required to retarget this approach for use on a parallel processing system are described in detail.

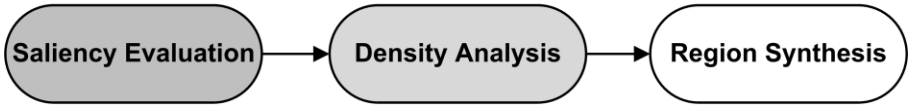


Figure 43. Region adjustment workflow. The region adjustment subtask is comprised of three stages: saliency evaluation, density analysis, and region synthesis.

Serial Implementation

The region adjustment subtask is divided into three distinct stages shown in Figure 43: *saliency evaluation*, *density analysis*, and *region synthesis*. During region adjustment, pixels are scanned for outliers in region membership. If a large number of outliers are identified, a new mega-region is created to represent the new scene object.

Let *f(x,y)* denote an input image frame submitted for segmentation. Let *s(x,y)* denote the output from the region building subtask before region adjustment procedures have been executed. During the saliency evaluation stage of region adjustment, the original image and the output segmentation are compared to determine the measure of CL-similarity present in the frame. This saliency evaluation encompasses about 11% of the total leap segmentation processing time (see Figure 39).

Recall Equations 1 and 2 from Chapter 2. Let $E(P_A,P_B,\tau)$ (Equation 2) define the CL-similar relation (Equation 1) between two RGB values (P_A, P_B) for some chosen threshold τ:

$$\max\begin{pmatrix}|R_1 - R_2|\\ |G_1 - G_2|\\ |B_1 - B_2|\end{pmatrix} \leq \varepsilon \ . \qquad \textbf{(1)}$$

$$E(P_A, P_B, \tau) = \begin{cases} 1, \text{ if } P_A, P_B \text{ are } \textit{CL-Similar} \\ 0, \text{ otherwise} \end{cases}. \qquad \textbf{(2)}$$

During saliency evaluation, the input frame is directly compared with the output of leap segmentation for matching within some threshold τ (τ = 30 was chosen in the current implementation). This comparison is performed to locate the salient portions of the image which potentially contain new scene objects:

$$salient(x, y, \tau) = \begin{cases} 0, & \text{if } E(f(x, y), s(x, y), \tau) \\ 1, & \text{otherwise} \end{cases} . \quad \textbf{(14)}$$

This new, saliency map identifies those frame locations that were assigned segment colors that were not CL-similar to their original image color during the leap segmentation procedure.

The second stage of region adjustment (density analysis) accounts for 7% of the total leap segmentation execution time. In this stage, once a saliency map has been generated, the density of the map is computed to determine the locations of contiguous salient pixels. The density analysis map is computed as follows for a search feature size of ρ:

$$DensityMap(x, y) = \sum_{i,j} salient(x + i, y + j, \tau),$$
$$i, j = -\lfloor \rho/2 \rfloor : \lfloor \rho/2 \rfloor, \quad x = 0 : width, \quad y = 0 : height . \quad \textbf{(15)}$$

To better exploit locality in the data cache during density analysis, the image is scanned in row-column order. Each pixel location is read from the cache only once, and its saliency is calculated. If the pixel is deemed salient, those pixels in the $\rho x \rho$ neighborhood surrounding that pixel are each incremented in the density map. This implementation makes the following assumption on neighborhood symmetry:

$$\text{if } P_x \in neighborhood(P_y), \text{then } P_y \in neighborhood(P_x) . \quad \textbf{(16)}$$

This is a reasonable assumption when implementing rectangular search neighborhoods. This 'neighborhood-incremental' approach in which densities are accumulated over time is far more efficient than a naïve density calculation approach, in which the density of each pixel location is calculated in-full and in-order, requiring multiple reads of the same pixel locations. In contrast, the incremental density calculation requires pixel locations to be read only once, and quick, atomic instructions can be used to increment the corresponding density map neighborhood locations. This reduces the number of comparisons required to perform this density analysis calculation from $N*\rho^2$ comparisons where N is the number of pixels in the image and ρ is the search feature size, to just N comparisons in the incremental approach. In performance trials, this incremental approach for density analysis enabled a more than 20x speed-up in density analysis execution over the naïve approach.

Region synthesis is the third and final stage of region adjustment and consumes just 5% of the total leap segmentation processing time. In this stage, the density map provided during density analysis is scanned for possible new object candidates. Once these object candidates have been identified, they are analyzed and added to the global cell list to represent new scene objects.

Parallel Implementation

The identification and classification of new object candidates in image scenes is often a computationally heavy procedure, requiring multiple searches across the input frame to determine the locations and sizes of new object candidates accurately. Several portions of the region adjustment subtask are amenable to parallelization. The saliency evaluation stage, which performs the image comparison for salient pixel detection, can be

processed in parallel fully (each pixel in parallel) as there are no data dependencies between computations in this stage. Pixel locations are simply divided among the available processors for processing.

In addition, the incremental approach for density analysis can also be parallelized for speed-up. In a parallel implementation, pixel locations can be processed in parallel with marked speed-up as long as their $\rho x \rho$ search neighborhoods do not overlap, causing threads to stall as they wait for density map locations to be freed for updating. This separation of the image into non-overlapping portions for density analysis parallelization is trivial with the assumption that the feature size $\rho << N$, where N is the number of pixels in the input image ($\rho = 5$ was chosen in the current implementation) and the thread count $t << N$ which is a reasonable assumption on commodity CPUs.

Performance evaluations for the region adjustment subtask are given in Section 4.5.3 for both the presented serial and parallel implementations of leap segmentation.

4.4.3. Subtask 3: Size Analysis

The final leap segmentation subtask, size analysis, applies the minimum size constraint to mega-regions, appropriately assimilating small regions into nearby mega-regions based on spatial and color similarities. The resulting mega-region list becomes the final segmentation.

Serial Implementation

Size analysis begins with a scan of the global cell list for cells that are too small to form their own segments. These cells are marked as garbage. As described in Section 4.3, the leap segmentation region map contains pointers onto the global cell list to convey segment membership at each pixel location (see Figure 40). Once garbage cells have been

identified in the global list, the region map is scanned for the presence of pointers to these garbage cells. If a pointer to a garbage cell is found, the adjacency neighborhood surrounding that pixel location is scanned for a possible replacement segment. The replacement segment is selected based on a minimum *sum of absolute differences* (SAD) comparison with the original cell assignment. The sum of absolution differences calculation is as follows:

$$SAD = \sum \begin{Bmatrix} |R_1 - R_2|, \\ |G_1 - G_2|, \\ |B_1 - B_2| \end{Bmatrix}. \quad \textbf{(17)}$$

In this way, collections of pixels that are too small to form their own segmentation regions are assimilated into larger, spatially-similar and chromatically-similar regions which meet required size constraints.

Parallel Implementation

The presented serial implementation of the size analysis subtask lends itself to a couple forms of parallelization. The global list scan to identify garbage cells can be performed completely in parallel without contention as there are no dependencies between cells during garbage classification. In addition, the region map scan, including the minimum sum of absolute differences calculations, is also highly parallelizable. Each pixel location can be evaluated in parallel, as this scan largely requires reads from the global list. However, updates to cell counts in the global list as garbage cells are assimilated into new regions must be performed atomically to ensure correctness, a limitation which has the potential to reduce parallel processing efficiency in this stage of execution.

Performance evaluations for the size analysis subtask are given in Section 4.5.3 for both the presented serial and parallel implementations of leap segmentation.

4.4.4. Storage Implementation Considerations

Implementation-specific storage control mechanisms applied in both serial and parallel implementations of the leap segmentation algorithm have the potential to highly affect leap segmentation performance in execution. This section describes, in detail, these storage implementation considerations for both the developed serial and parallel implementations of leap segmentation.

Serial Implementation

For the highly-optimized serial implementation of leap segmentation, a large support storage framework for fast allocation/de-allocation of leap segmentation resources has been developed. Examples of highly used leap segmentation resources include the cells contained in the global cell list and the tile sets contained in each discretized image tile. These resources are repeatedly allocated, accessed, and de-allocated during leap segmentation processing, prompting a need for an efficient storage control framework for these particular resources. In the developed serial implementation of leap segmentation, these resources are allocated as part of an initialization procedure in large blocks on the heap and passed to the segmentation procedure whenever a resource is needed during execution. This ensures that the program memory remains roughly contiguous on the heap and allows for faster memory accesses in the data cache.

All freed leap segmentation resources are added to a free list which is maintained during program execution. When a leap segmentation resource is required, the free list is consulted first for any possible previously-allocated resources. If none are available, a

large, new resource block is allocated on the heap and resources are distributed from that block as processing moves forward.

Parallel Implementation

The storage control framework used in the presented serial implementation, while highly efficient for sequential processing, is not ideal in a parallel environment. A central free list would become a highly contended resource during execution and would create a bottleneck in processing. In addition, whereas the spatial locality of cells in the data cache is desired in sequential processing, this is not always the case in a parallel environment. Two threads (t_1, t_2) operating on different resources in the same cache line (r_1, r_2) can be affected by false sharing. For example, if t_1 modifies its resource r_1, the entire cache line will be invalidated, causing interference with t_2 which cannot access its unmodified resource r_2 until the cache line has been updated.

Because of these limitations, the novel serial leap segmentation run-time storage control framework is removed in the developed parallel implementation of leap segmentation in order to reduce contention among threads.

4.5. Experimental Results

Both the described serial and parallel leap segmentation approaches are implemented in the C programming language and developed in a Linux environment. The input parameters chosen for use in leap segmentation are λ=8, ε=20, and α=50. These adjacency (λ), equivalence (ε), and minimum size (α) parameters are chosen based on optimal performance obtained from the full parameter evaluation based on compression and accuracy objective functions given in Section 2.5.

The parallel implementation of leap segmentation is built using the OpenMP API [62] on a shared memory architecture. The OpenMP API allows for quick parallelization of existing C code. In addition, the POSIX threads (Pthreads) API is used to enforce spin lock implementations where explicit locks are required for critical section locking.

A set of moving-camera traffic scene sequences collected at Georgia Tech is used in evaluation experiments (discussed in Section 3.4). The captured GTTraffic dataset sequences [32] contain fast-moving traffic events such as vehicles swerving into visible lanes and thus provide highly diverse scenes for analysis. These scene collections were captured at 32 fps using a forward-mounted Kodak Zi6 on an automobile dashboard. Over 3,000 images were selected from the GTTraffic dataset for use in these experiments. In addition, scaled versions of these 1280x720 images were created at several resolutions (960x540, 640x360, and 320x180) for use in evaluation experiments.

The following sections contain detailed performance evaluations of the presented serial and parallel leap segmentation implementations. First, it is demonstrated that leap segmentation accuracy is consistent in the serial and parallel implementations developed in this chapter. Next, an overall execution performance analysis is presented for experimental trials on both high-performance and resource-constrained hardware. Then, each leap segmentation subtask is evaluated separately on resource-constrained hardware in order to outline the performance benefits of retargeting leap segmentation to a multi-core platform.

4.5.1. Serial vs. Parallel Implementation Accuracy

In Section 2.6, the accuracy of the leap segmentation approach was discussed and compared with two well-known segmentation approaches from the literature (see Figure

20). The original, single-core leap segmentation approach was shown to maintain a high level of scene integrity when compared to traditional approaches. In this section, it is further demonstrated that leap segmentation accuracy is consistent in the developed serial and parallel leap segmentation implementations.

The quantitative objective functions introduced in Section 2.5.1, the *number of segments* (to evaluate compression) and the *nonmatching pixel percentage* (to assess segmentation accuracy), are applied in this comparison (see Equations 1-4). In Figure 44, the resulting serial and parallel leap segmentation implementation outputs of a zebra scene (481x321 pixels) are shown. The nonmatching pixel percentages are listed above each segmentation output. Figure 44a shows the merged segmentation outputs. To help discern region membership in the merged image, an artificially colorized segmentation is

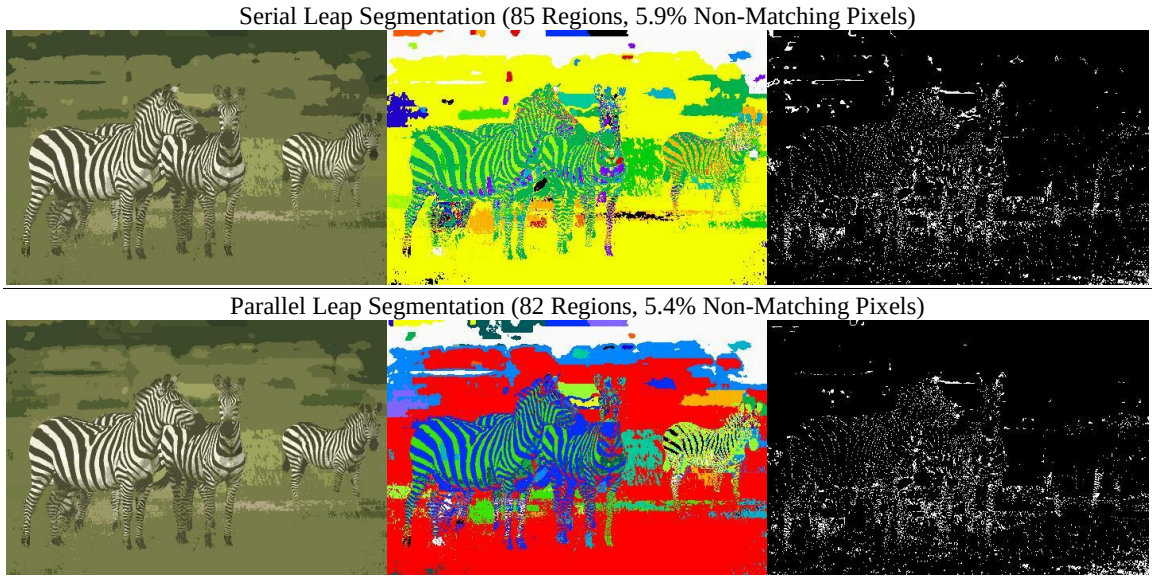


Figure 44. Serial vs. parallel leap segmentation accuracy comparison images (481x321 pixels). (a) The merged segmentation output. (b) A colorized representation of the segmentation to show region membership clearly. (c) A binary map of nonmatching pixels in the output segmentation.

shown in Figure 44b. Contrasting color assignments show region pixel membership. Figure 44c shows a binary matching map for the final segmentation, where nonmatching pixels are plotted in white.

Both implementations maintain similar levels of scene integrity during segmentation, as shown in Figure 44a. The serial leap segmentation implementation achieves a very low nonmatching pixel percentage (5.9%). The parallel leap segmentation implementation produces an even lower nonmatching pixel percentage (5.4%) indicating that, for this scene, segmentation accuracy slightly improves when moving to a parallel implementation.

Analysis of trial runs on over 3000 dataset images indicate that overall segmentation accuracy changes little between the described serial and parallel implementations. This was to be expected, as no sacrifices were made to the core leap segmentation algorithm to parallelize the approach. In addition, slight changes to segment structure are possible when moving to a parallelized approach as the order in which pixel locations are processed may change (see Figure 44b). However, in trials these segment structure changes were not shown to affect segmentation accuracy.

4.5.2. Overall Performance Analysis

In the following sections, two systems are used during evaluation experiments to assess implementation performance. The first system contains high-performance hardware and is used to analyze parallel leap segmentation performance at high thread counts. The second system contains embedded hardware and is used to evaluate overall performance under resource-constrained conditions.

High-Performance Hardware

In these experiments, a pair of Intel Xeon E5-2670 (20M cache) processors running at 2.60 GHz with a total of 16 Sandy Bridge-EP cores with TDP of 115 watts is used to evaluate parallel leap segmentation performance. During all trials this system is operating using Red Hat Fedora release 17.

The effect of image size on the frame rate of execution of parallel leap segmentation can be seen in Figure 45 for a wide range of thread counts. As expected, decreasing the image size significantly increases the frame rate at which parallel leap segmentation executes. A more interesting trend is also visible regarding thread count and its effect on execution rates. For most frame sizes, the rate of execution is highest when using four threads and drops off linearly as the number of threads increases or decreases from that value. This indicates an optimal execution state at a thread count of

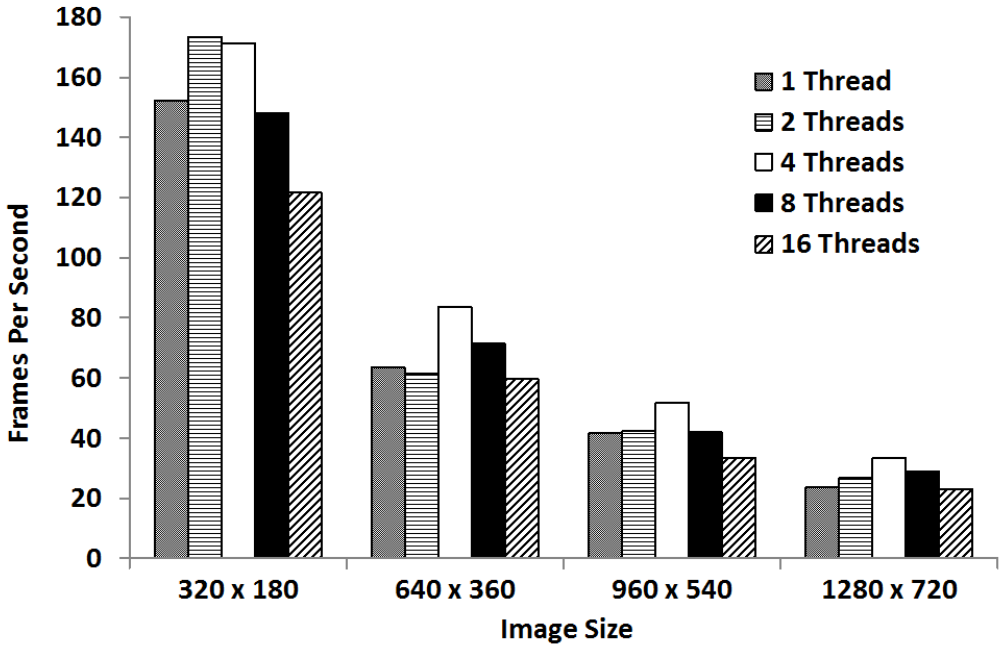


Figure 45. Plot of the effect of image size on frame rate for various thread counts on a pair of Intel Xeon E5-2670 processors

four. An increase in frame rate as the available thread-count increases is to be expected as more threads share the work load. However, the corresponding decrease from an optimal thread count is also to be expected. This is the point at which resource contention among threads causes performance to suffer and eventually bottom-out.

This phenomenon occurs more rapidly in smaller images. In Figure 45, the very small 320x180 image shows an optimum thread-count of only two threads. In tests, this contention on resources arises primarily in the region building stage of leap segmentation. The image is discretized into tiles, and tiles along the same image diagonal are distributed between the available threads for processing. If one assumes tile sizes are fixed, a smaller image results in fewer tiles along the image diagonal, and thus fewer tiles to distribute among threads for processing. This facilitates the hazardous operating condition in which threads operate on tiles either directly neighboring or nearby each other along the image diagonal. The closer tiles are within the image space, the more likely they are to share pixels within the same segment, causing contention between threads as they attempt to simultaneously update the same locations on the shared global cell list of image segments.

One can improve upon this discovered optimum by simply restricting thread count in the region building subtask of leap segmentation to four, while allowing higher thread counts in the less contention-prone regions of the approach. As just 30% of the approach is affected by the additional threads when region building is fixed, only a small level of speed-up is to be expected (see Figure 39). The results of this experiment are shown in Figure 46 for large images in which the effects of this operating condition can be seen more clearly. Restricting thread count in the region building subtask while

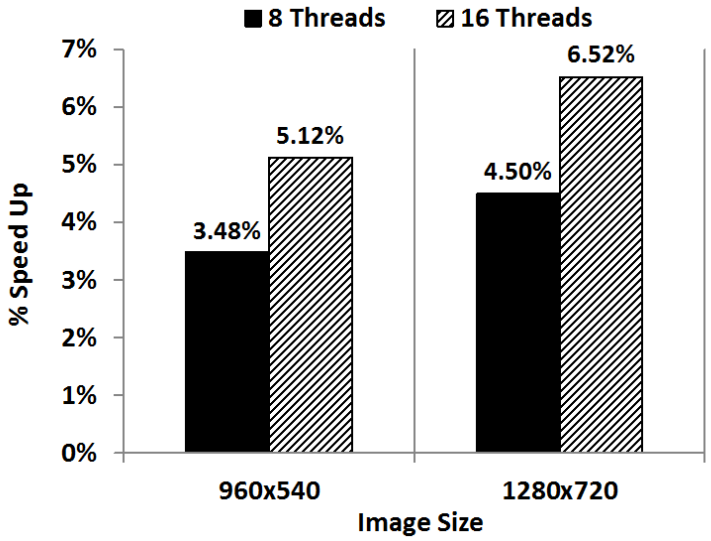


Figure 46. Plot of the percentage speed-up in frame rate as overall thread count increases above four while keeping region building subtask thread count fixed at four to limit resource contention on an a pair of Intel Xeon E5-2670 processors.

increasing thread counts in the other subtasks does facilitate additional speed-up. When processing 1280x720 images, the overall rate of execution increases 4.5% with 8 threads and over 6.5% with 16 threads.

Resource-Constrained Hardware

The analysis of parallel leap segmentation execution with up to 16 cores on a high-performance machine given in the previous section is useful in theory. However, the leap segmentation approach was designed specifically for use in an embedded, resource-constrained environment. Such an environment will likely provide only a fraction of this number of processing cores. Therefore, the leap segmentation optimum processing state at four cores is, in fact, ideal for the desired execution environment where four cores are likely the most one will have ready access to.

The overall performance of the developed leap segmentation approach implementations (both serial and parallel) is further tested using a mobile Intel Core I3–330M processor (3M cache, 2.13 GHz) running 64-bit Ubuntu 12.04. The I3-330M is a mobile processor with 2 Nehalem cores (4 hyperthreaded cores) with TDP of 35 watts. It is important to note that this is a dual core machine with hyperthreading to provide four threads for execution. As this is not a four core machine, full performance scaling to four hardware cores is not expected.

The averaged frame rates for trials of more than 3000 test frames for both the serial and parallel leap segmentation implementations are given in Table 3. The highly-optimized serial leap segmentation implementation achieves execution rates of more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images. Furthermore, the multi-core leap segmentation implementation achieves frame rates of more than 29 fps on 1280x720 images using two threads and more than 31 fps when using four threads, thus easily meeting real-time processing standards. Frame rates for both the serial and parallel implementations increase exponentially as image sizes decrease. Parallel leap segmentation tests with four threads exhibit frame rates up to 55 fps on 960x540 images, up to 115 fps on 640x360 images and even 370 fps on 320x180

Table 3

Resource Constrained Hardware Execution Rates (FPS)

	Size 320x180	***Size 640x360***	***Size 960x540***	***Size 1280x720***
Serial	314.71	81.44	36.53	20.41
1 Thread	286.16	74.83	33.59	18.71
2 Threads	390.38	113.34	52.66	29.83
4 Threads	370.59	114.51	54.98	31.71

images. These very high frame rates exhibited by both the serial and parallel leap segmentation implementations are state-of-the-art execution performance on both low and high resolution images.

Figure 47 includes plots of performance speed-up of the parallel leap segmentation implementation over the serial implementation when executing on resource-constrained hardware. The parallel implementation is evaluated for threads counts of 1, 2, and 4. As frame sizes increase, the percentage speed-up of execution

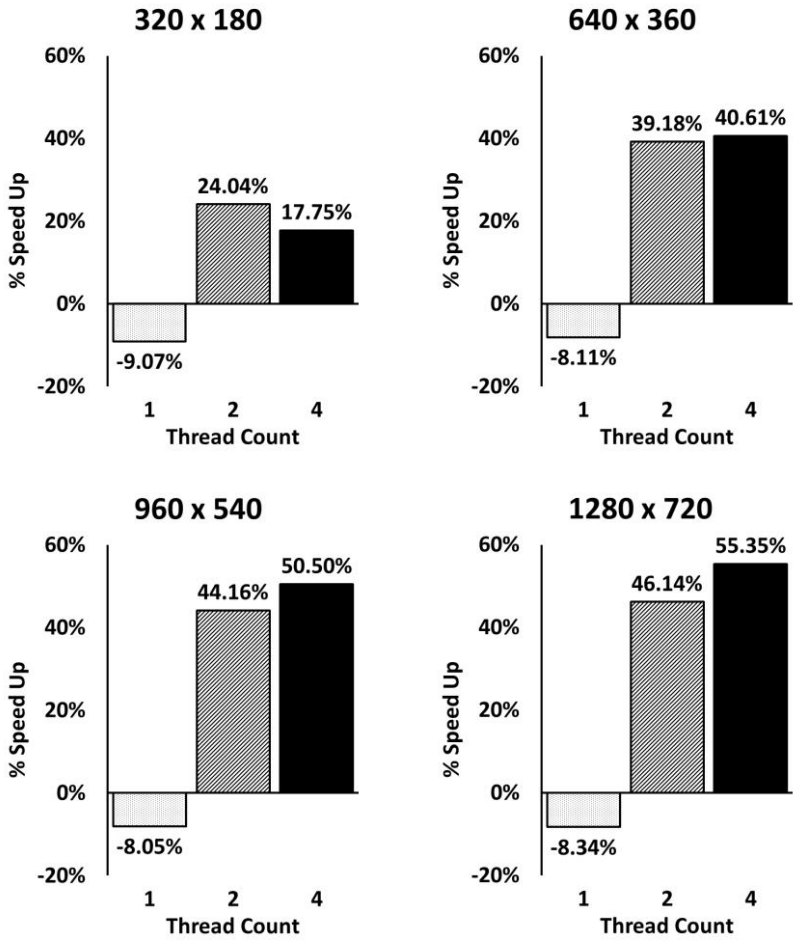


Figure 47. Percentage speed-up of parallel leap segmentation over serial leap segmentation on an Intel Core I3-330M processor for 1, 2, and 4 thread counts and at various frame sizes.

increases across the board. At a thread count of one, the negative speed-up percentages exhibited by the parallel leap segmentation implementation are due to the high overhead of parallelization associated with Open-MP (thread creation, initialization, scheduling, assignment, etc.). At thread counts of two and four, consistent speed-up is achieved over the serial leap segmentation approach. Once again, it must be noted that the platform used in these trials contains a dual-core processor with hyperthreading to provide four threads for execution. Because of this, performance is not expected to scale as if four hardware cores were available.

For 1280x720 frame sizes, parallel leap segmentation achieves a 46% speed-up over the serial implementation with two threads and 55% speed-up when four threads are used. These results with high resolution images on resource-constrained hardware are highly promising and show enormous potential for the use of the developed parallel leap segmentation implementation in the preprocessing stages of high level vision applications operating in real-time, embedded environments.

4.5.3. Subtask Performance Analysis

In the following sections, performance evaluations on the resource-constrained Intel Core I3-330M mobile processor described previously are presented for each leap segmentation subtask for both the serial and parallel implementations of leap segmentation presented in this chapter. A detailed comparison of execution rates is given both before and after parallelization for each subtask. A high resolution image size was chosen for all subtask performance experiments (1280x720) in order to push the limits of the algorithms and better assess algorithmic performance.

Table 4

Region Building Execution Performance

	Serial	*1 Thread*	*2 Threads*	*4 Threads*
FPS	30.80	27.61	39.71	45.98
% Speed Up	-	-10.33%	28.93%	49.30%

Subtask 1: Region Building

The execution performance of the leap segmentation region building subtask is displayed for both the developed serial and parallel implementations in Table 4. Recall that this subtask is parallelized along image diagonals to avoid data dependencies between threads. The results in Table 4 indicate that a thread count of two is required for speed-up of the parallel approach over the highly-optimized serial approach due to the high overhead of parallelization with OpenMP (thread creation, initialization, scheduling, assignment, etc.). When four threads are available for execution, parallel leap segmentation achieves ~15 fps speed-up over the serial approach (an almost 50% speed-up in execution performance).

Subtask 2: Region Adjustment

Table 5 displays execution rates for assessment of the saliency evaluation portion of the region adjustment subtask for both the serial and parallel implementations of leap segmentation developed in this chapter. Recall that this portion of the subtask can be fully parallelized, as no data dependencies exist between pixels. The results in Table 5 show that an over 100% speed-up is achieved after adding just one thread to the execution for a total of two available threads. However additional threads do not yield as much performance improvement. When operating with four available threads, the parallel

Table 5

Saliency Evaluation Execution Performance

	Serial	***1 Thread***	***2 Threads***	***4 Threads***
FPS	184.91	160.60	372.36	421.31
% Speed Up	-	-13.15%	101.37%	127.84%

leap segmentation saliency evaluation speed-up over serial leap segmentation is ~127%. This may be due to the fact that the experimental setup contains just two hardware cores and the execution of four threads is achieved with hyperthreading.

Table 6 shows execution rates for trial runs of the density analysis stage of the region adjustment subtask. This stage implements an incremental approach to density analysis and is able to be fully parallelized with each pixel location in parallel. Therefore, pixel locations are divided equally among the available threads for execution. With four cores, an over 115% speed-up is achieved by parallelizing this computation. The comparably minor speed-up results exhibited in this stage are most likely due to the high overhead of OpenMP scheduling dominating the processing of an already proportionally low computation portion of leap segmentation.

The third and final stage of the region adjustment subtask, region synthesis, does

Table 6

Density Analysis Execution Performance

	Serial	***1 Thread***	***2 Threads***	***4 Threads***
FPS	228.36	188.22	387.37	491.36
% Speed Up	-	-17.58%	69.63%	115.17%

Table 7

Size Analysis Execution Performance

	Serial	*1 Thread*	*2 Threads*	*4 Threads*
FPS	420.69	358.59	580.53	728.68
% Speed Up	-	-14.76%	38.00%	73.21%

not prove amenable to parallelization with the high overhead cost of OpenMP as it encompasses only 5% of total leap segmentation processing. Therefore, this stage of region adjustment is left to function sequentially.

Subtask 3: Size Analysis

In Table 7, the execution performance for the size analysis subtask for both the serial and parallel leap segmentation implementations developed in this chapter is shown. Recall that each image location can be evaluated in parallel during this subtask and that the task largely requires only reads from shared memory. However, shared memory writes, while infrequent during this subtask, must be performed atomically and therefore could hinder performance on a parallel processing system. Despite these synchronization concerns, Table 7 shows that significant speed-up occurs after adding just one thread to the execution. With four threads, a ~73% speed-up of the size analysis subtask execution is achieved over the serial approach.

4.6. Conclusion

The goal of this research is to achieve real-time (>25 fps) image segmentation execution performance on a commercially-available CPU with multiple processing cores that does not require special hardware. To that end, first a highly optimized serial

implementation of the leap segmentation algorithm is introduced. This highly-optimized serial approach is shown to achieve frame rates of more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images. This serial implementation of leap segmentation will prove useful in many embedded, resource-constrained environments where processing speed is critical.

To achieve real-time execution of leap segmentation on high-resolution images, a multi-core leap segmentation implementation is presented. Numerous parallelization techniques are applied to different portions of the leap segmentation algorithm to achieve further speed-up over the serial implementation. The steps taken to parallelize each leap segmentation subtask are described in detail. In experiments, the developed parallel implementation of leap segmentation executes at over 114 fps on 640x360 images and over 31 fps on high-resolution, 1280x720 images, easily meeting real-time processing standards (and achieving a 55% speed-up over the serial approach).

The original leap segmentation approach (Chapter 2) was designed for use in embedded, resource-constrained environments. An embedded platform may be limited to a single processing core, in which case this chapter's highly optimized serial leap segmentation implementation is ideal and achieves real-time behavior on a wide range of image sizes. Embedded platforms with multiple available processing cores can make use of this chapter's parallel implementation of leap segmentation. This parallel implementation executes in real-time on high resolution images with peak performance at a thread count of four. With the frame rates exhibited in performance trials, both of the presented approaches show enormous potential for use in real-time, embedded environments with high-resolution input images.

CHAPTER 5

CONCLUSION AND SUMMARY OF RESULTS

This dissertation investigates image segmentation in embedded, real-time applications. It presents a novel approach, called *leap segmentation*, that efficiently reduces and restructures image data into regions while preserving necessary salient features in the image (Forsthoefel et al.) [34]. Leap segmentation is evaluated using both standard datasets from the literature [55], [56] and the new, GTTraffic dataset:

- GTTraffic, a publicly available dataset of moving-camera traffic sequences collected at Georgia Tech (Forsthoefel et al.) [32], is developed and presented for use in vision evaluation experiments.

The leap segmentation approach is extensively compared with prior efforts using both classical metrics of performance (e.g. the F-measure [76], the probabilistic rand index (PRI) [75]) and other, newly developed metrics designed for more extensive evaluations (e.g. the non-matching pixel percentage (Forsthoefel et al.) [34]).

- Leap segmentation demonstrates high region-assignment accuracy and, compared to other approaches, preserves a higher level of scene integrity (up to 30-40% higher) using a given storage resource.
- In experiments, this approach exhibits execution time improvements of 10x-15x over traditional approaches.

In addition, the usefulness of applying this novel method of image segmentation in the preprocessing stages of a high-level vision application for image labeling and 3D reconstruction is evaluated and compared with existing segmentation approaches (Forsthoefel et al.) [31].

- The efficiency of a high-level image layout and 3D reconstruction task can be dramatically improved by applying the leap segmentation technique during preprocessing.

In the second contribution of this dissertation, the single-frame leap segmentation algorithm is extended to efficiently process video while maintaining region boundary continuity between image frames. Temporal analysis of this video leap segmentation algorithm is performed to evaluate segmentation stability over time in video sequences from moving camera scenes (Forsthoefel et al.) [32], (Forsthoefel et al.) [33].

- Video leap segmentation successfully tracks segments across spatial and temporal bounds, generating fast, stable segmentations of images from moving-camera video sequences.

Video leap segmentation is then applied to the task of salient segment transformation detection for alerting drivers of critical scene changes that may affect steering decisions. The resulting salient transformation recognition technique quickly identifies and tracks rapidly moving, salient objects in traffic video sequences (Forsthoefel et al.) [33].

- Trial results demonstrate that with little added computation, video leap segmentation can detect salient regions in traffic scenes with high accuracy, correctly detecting 80% of salient segment transformations in trial scenes with less than 5% false positives.

In the third contribution of this dissertation, a parallel, multi-core implementation of leap segmentation is presented (Forsthoefel et al.) [35]. The goal is to achieve real-time (>25 fps) segmentation performance on a commercially-available CPU with multiple processing cores that does not require special hardware. To that end, a highly optimized serial implementation of the leap segmentation algorithm is introduced. All optimizations built into this serial implementation are described in detail including those optimizations made to leap segmentation data structures.

- This optimized serial implementation is demonstrated to achieve frame rates of more than 80 fps on 640x360 images and more than 20 fps on high resolution (1280x720) images, thus far exceeding the state-of-the art in execution speed.

To achieve real-time execution of leap segmentation on high-resolution images, a multi-core leap segmentation implementation is then presented. Numerous parallelization techniques were applied to different portions of the leap segmentation algorithm to achieve further speed-up over a serial implementation. The steps taken to parallelize each leap segmentation subtask are described in detail.

- On a multi-core, mobile processing system with four threads, this multi-core leap segmentation implementation achieves frame rates of over 114 fps on 640x360 images and more than 31 fps on 1280x720 images, thus easily meeting real-time processing standards.

The experiments to evaluate the performance of both the highly optimized serial implementation and the parallel implementation of leap segmentation are performed on various image sizes and under various operating conditions. With the execution frame rates exhibited in these performance trials, both of the developed approaches show enormous potential for use in real-time, embedded environments with high-resolution input images.

5.1. Future Work

Several avenues of future work are possible, particularly in video leap segmentation. These include an in-depth parameter variation analysis of video leap segmentation parameters and its optimization and parallelization on multi-core, mobile platforms. Parallelization of this approach targeting a multi-core platform, such as a GPU, appears highly promising because most comparisons between pixels occur across frames in time. Therefore, operations performed on each pixel location within a frame have the potential to be performed in parallel. The larger the input frame, the more opportunities for parallelism, particularly on systems which are able to provide high thread counts to their vision processing procedures.

Other avenues of future work will explore additional real-world applications of both single-frame leap segmentation and video leap segmentation. These include high-

level image correspondence and image registration applications, especially for use on high resolution input images, and content-based image retrieval (CBIR) applications.

As camera usage on mobile devices increases, the demand for fast, accurate image processing techniques will continue to drive computer vision researchers to develop new and innovative methods for improving the efficiency of the modern image processing pipeline. This dissertation explores the challenging field of image segmentation and proposes techniques for segmentation preprocessing that can provide much-needed speed-up when used as part of contemporary high-level image analysis and interpretation systems.

APPENDIX

ADDITIONAL LEAP SEGMENTATION RESULTS

This appendix includes comparison tables of segmentation output images from three segmentation approaches: the Leap Segmentation approach introduced in Chapter 2 of this dissertation (Forsthoefel et al.) [34], the Mean-Shift Clustering with Edge Detection (EDISON) approach [20], and the Efficient Graph-Based Image Segmentation (EGBIS) approach [30]. These evaluation experiments are extended from those introduced in Section 2.6. The dataset used for these extended comparisons is 300 images from the Berkeley Segmentation Dataset [55], [56]. This dataset provides a diverse collection of scene types with varying feature sizes and scales. To ensure a consistent comparison, all algorithms were adjusted to produce similar levels of segmentation. The following figures display segmentation output for each technique and are labeled with segment count information and non-matching pixel percentages (see Equations 2-4 from Chapter 2 for more details).

Original Image

Leap Segmentation (85 Regions, 1.1% Non-Matching Pixels)

EDISON (91 Regions, 13.1% Non-Matching Pixels)

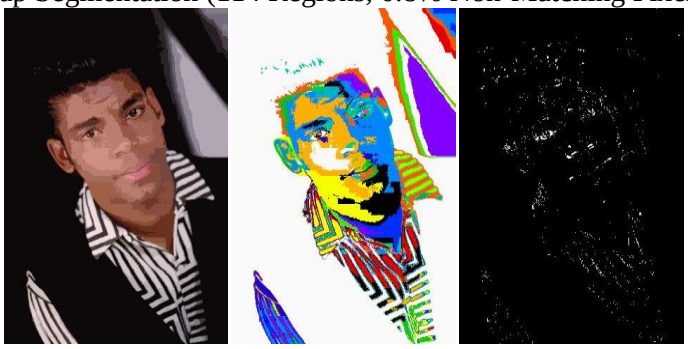
EGBIS (87 Regions, 32.8% Non-Matching Pixels)

(a) **(b)** **(c)**

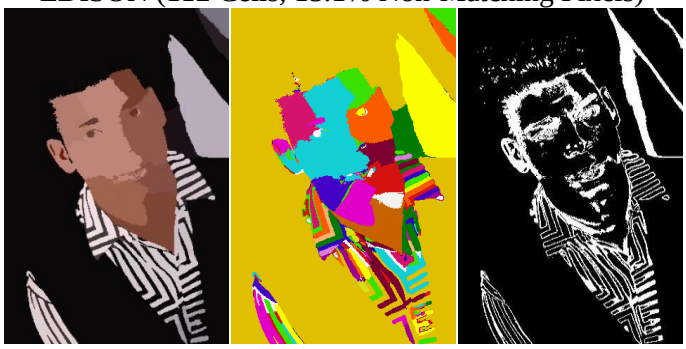
Figure 48. Segmentation comparison images, human face (321x481 pixels). (a) The merged segmentation output for each technique. (b) A colorized representation of the segmentation to show region membership clearly. (c) A binary map of nonmatching pixels in the output segmentation.

Original Image

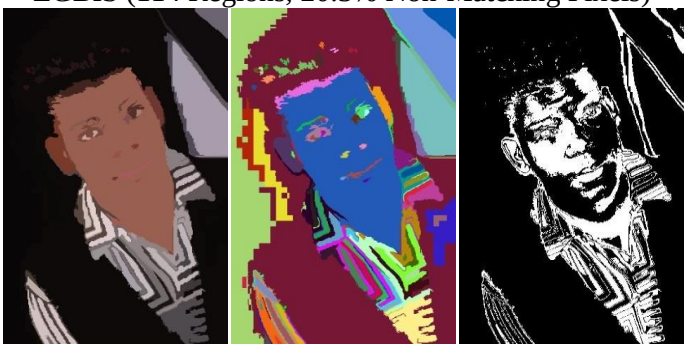
Leap Segmentation (114 Regions, 0.8% Non-Matching Pixels)



EDISON (112 Cells, 15.1% Non-Matching Pixels)



EGBIS (114 Regions, 20.3% Non-Matching Pixels)



(a) **(b)** **(c)**

Figure 49. Segmentation comparison images, human striped shirt (321x481 pixels). (a) The merged segmentation output for each technique. (b) A colorized representation of the segmentation to show region membership clearly. (c) A binary map of nonmatching pixels in the output segmentation.

REFERENCES

[1] A. Abramov, T. Kulvicius, F. Wörgötter, and B. Dellen, "Real-time image segmentation on a GPU," *Proc. Facing the Multicore-Challenge*, pp. 131-142, 2011.

[2] R. Adams and L. Bischof, "Seeded region growing," *IEEE Trans. Pattern Analysis and Machine Intel1ligence*, vol. 16, no. 6, pp. 641-647, June 1994.

[3] M. Allmen and C.R. Dyer, "Computing spatiotemporal relations for dynamic perceptual organization," *CVGIP: Image Understanding*, vol. 3, no. 58, pp. 338-351, 1993.

[4] S. Alpert, M. Galun, R. Basri, and A. Brandt, "Image segmentation by probabilistic bottom-up aggregation and cue integration," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition*, 2007.

[5] J. Angulo, J. Serra, "Color segmentation by ordered mergings," *Proc. IEEE Int. Conf. Image Process.*, vol. 2, pp. 125-128, 2004.

[6] S. Apewokin, B. Valentine, D. Forsthoefel, D.S. Wills, L.M. Wills, and A. Gentile, "Embedded Real-Time Surveillance Using Multimodal Mean Background Modeling", *Embedded Computer Vision*, (B. Kisacanin and S. Bhattacharyya and S. Chai, eds.), Springer, pp. 163-175, 2009.

[7] G. Baldi, C. Colombo, and A. Del Bimbo, "A compact and retrieval-oriented video representation using mosaics," *Proc. 3rd Int'l Conf. Visual Information Systems*, LNCS 1614, Springer: Amsterdam, The Netherlands, pp. 171-178, June 1999.

[8] M.R. Bales, D. Forsthoefel, B. Valentine, D.S. Wills, and L.M. Wills, "BigBackground-Based Illumination Compensation for Surveillance Video", *EURASIP Journal on Image and Video Processing,* vol. 2011, Article ID 171363, 22 pages, 2011.

[9] M.R. Bales, D. Forsthoefel, D.S. Wills, and L.M. Wills, "Illumination Change Compensation Techniques to Improve Kinematic Tracking", *Proc. IEEE Workshop on Applications of Computer Vision*, pp. 434–439, 2011.

[10] R. Beare, "A locally constrained watershed transform," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1063–1074, 2006.

[11] S. Belongie, C. Carson, H. Greenspan, and J. Malik, "Color and texture-based image segmentation using EM and its application to content-based image retrieval," *Proc. 6th IEEE Int'l Conf. Computer Vision*, pp. 675-682, 1998.

[12] P. Besl and R. Jain, “Three dimensional object recognition,” *ACM Comput. Surv.*, vol. 17, no. 1, pp. 75-145, 1985.

[13] Y. Boykov and G. Funka-Lea, “Graph cuts and efficient N-D image segmentation,” *Int’l Journal of Computer Vision*, vol. 70, no. 2, pp. 109-131, 2006.

[14] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222-1239, 2001.

[15] A. Brandt, “Algebraic multigrid theory: the symmetric case,” *Applied Mathematics and Computation*, vol. 19, no. 1-4, pp. 23-56, July 1986.

[16] M. Brockway, “Red Light Cameras to Spot Parking Violators? City Asks Bidders For Details,” *DNAinfo*, http://www.dnainfo.com/chicago/20130311/chicago/red-light-cameras-issuing-parking-tickets-city-asks-bidders-for-details, 11 March 2013.

[17] M.Á. Carreira-Perpiñán, “Acceleration strategies for Gaussian mean-shift image segmentation,” *Proc. IEEE Int’l Conf. Computer Vision and Pattern Recognition*, 2006.

[18] C. Carson, S. Belongie, H. Greenspan, and J. Malik, “Blobworld: image segmentation using expectation-maximization and its application to image querying,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 8, pp. 1026–1038, 2002.

[19] Y.-L. Chang and X. Li, “Adaptive image region-growing,” *IEEE Trans. Image Processing*, vol. 3, no. 6, pp. 868-872, 1994.

[20] C.M. Christoudias, B. Georgescu, and P. Meer, “Synergism in low level vision,” *Proc. 16th Int’l Conf. Pattern Recognition*, Quebec City, Canada, vol. 4, pp. 150–155, August 2002.

[21] D. Comaniciu and P. Meer, “Mean shift analysis and applications,” *Proc. IEEE Int’l Conf. Computer Vision*, vol. 2, pp. 1197-1203, 1999.

[22] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Trans. Pattern Analysis Machine Intelligence*, vol. 24, no. 5, pp. 603-619, May 2002.

[23] J. Costeira and T. Kanade, “A multi-body factorization method for motion analysis,” *Proc. IEEE Int’l Conf. Computer Vision*, pp. 1071-1076, 1995.

[24] T. Cour, F. Benezit, and J. Shi, “Spectral segmentation with multiscale graph decomposition,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, vol. 2, pp. 1124-1131, 2005.

[25] D. DeMenthon, “Spatio-temporal segmentation of video by hierarchical mean shift analysis,” *Proc. Statistical Methods in Video Processing Workshop*, June 2002.

[26] A. Dempster, N.M. Laird, and D.B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[27] Y. Deng and B. Manjunath, “Unsupervised segmentation of color-texture regions in images and video,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 800-810, 2001.

[28] F. Drucker and J. MacCormick, “Fast superpixels for video analysis,” *Proc. IEEE Workshop on Motion and Video Computing*, pp. 1-8, 2009.

[29] K. Drummond, “Darpa Wants Self-Guiding, Storytelling Cameras,” *Wired*, http://www.wired.com/dangerroom/2010/03/darpa-wants-self-guiding-storytelling-cameras/, 17 March 2010.

[30] P. Felzenszwalb and D. Huttenlocher, “Efficient graph-based image segmentation,” *Int’l Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[31] D. Forsthoefel, D.S. Wills, and L.M. Wills, “Leap segmentation for recovering image surface layout,” *Proc. IEEE Southwest Symp. Image Anal. Interp.*, Santa Fe, NM, April 2012.

[32] D. Forsthoefel, D.S. Wills, and L.M. Wills, "The GTTraffic dataset," *Mobile Vision Embedded Systems Lab (MoVES)*, Georgia Institute of Technology, http://www.ece.gatech.edu/research/labs/pica/, April 2012.

[33] D. Forsthoefel, D.S. Wills, and L.M. Wills, "Unsupervised video leap segmentation for fast detection of salient segment transformations in mobile sequences," *Proc. 15th IEEE Int’l Conf. Intelligent Transportation Systems (ITSC)*, pp.728-733, 16-19 September 2012.

[34] D. Forsthoefel, D.S. Wills and L.M. Wills, “Fast leap segmentation,” *Image and Vision Computing,* under review, 17 pages, submitted May 2013.

[35] D. Forsthoefel, D.S. Wills, and L.M. Wills, “Real-Time, parallel segmentation of high resolution images on multi-core platforms*,” Journal of Real-Time Image Processing*, under review, 15 pages, submitted May 2013.

[36] M. Gelgon and P. Bouthemy, “A region-level motion-based graph representation and labeling for tracking a spatial image partition,” *Pattern Recognition*, vol. 33, no. 4, pp. 725-740, 2000.

[37] B. Georgescu, I. Shimshoni, and P. Meer, "Mean shift based clustering in high dimensions: a texture classification example," *Proc. 9th IEEE Int'l Conf. Computer Vision*, pp. 456-463, 2003.

[38] L. Greenemeier, "Smart Headlights Can See through Rain, Sleet and Snow," *Scientific American*, http://www.scientificamerican.com/article.cfm?id=smart-headlights-see-through-rain-sleet-snow, 1 September 2012.

[39] H. Greenspan, J. Goldberger, and A. Mayer, "Probabilistic space-time video modeling via piecewise GMM," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 3, pp. 364-396, March 2004.

[40] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society, Series B*, vol. 51, no. 2, pp. 271-279, 1989.

[41] M. Grundmann, V. Kwatra, M. Han, and I. Essa. "Efficient hierarchical graph-based video segmentation," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition*, 2010.

[42] A. Hanbury and J. Stöttinger, "On segmentation evaluation metrics and region counts," *Proc. 19th Int'l Conf. Pattern Recognition*, pp. 1-4, December 2008.

[43] P.N. Happ, R.S. Ferreira, C. Bentes, G.A.O.P Costa, and R.Q. Feitosa, "Multiresolution segmentation: a parallel approach for high resolution image segmentation in multicore architectures," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. 4, 2010.

[44] D. Hoiem, A.A. Efros, and M. Hebert, "Automatic photo pop-up," *ACM SIGGRAPH*, 2005.

[45] D. Hoiem, A.A. Efros, and M. Hebert. "Recovering surface layout from an image," *Int'l Journal of Computer Vision*, vol. 75, no. 1, pp. 151-172, 2007.

[46] S.L. Horowitz and T. Pavlidis, "Picture segmentation by a tree traversal algorithm," *Journal of the Association for Computing Machinery*, vol. 23, no. 2, pp. 368-388, 1976.

[47] A. Jain, M. Murty, and P. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264-323, 1999.

[48] Q. Ju, D. Forsthoefel, S. Azmat, L.M. Wills, D.S. Wills, and R. Ying, "PathMark: A Novel Fast Lane Detection Algorithm for Embedded Systems", *Proc. 4th International Symposium on Information Science and Engineering (ISISE 2012)*, Shanghai, China, December 2012.

[49] J. Kim and J.W. Woods, “Spatiotemporal adaptive 3-D Kalman filter for video,” *IEEE Trans. Image Processing*, vol. 6, no. 3, pp. 414-424, 1997.

[50] B. Kisacanin, S. Bhattacharyya, and S. Chai, eds., *Embedded Computer Vision (Advances in Pattern Recognition)*, Springer, 2009.

[51] D. Liebowitz, A. Criminisi, and A. Zisserman, “Creating architectural models from images,” *Proc. Euro-Graphics*, vol. 18, 1999.

[52] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” *Proc. Fifth Berkeley Symp. Mathematical Statistics and Probability*, pp. 281-297, 1967.

[53] J. Malik, S. Belongie, T. Leung, and J. Shi, “Contour and texture analysis for image segmentation,” *Int’l Journal of Computer Vision,* vol. 43, no. 1, pp. 7–27, 2001.

[54] J. Markoff, “Google Cars Drive Themselves, in Traffic,” *The New York Times*, http://www.nytimes.com/2010/10/10/science/10google.html?_r=0, 9 October 2010.

[55] D. Martin and C. Fowlkes, “The Berkeley Segmentation Dataset and Benchmark,” [Online], University of California, Berkeley, http://www.cs.berkeley.edu/projects/vision/grouping/segbench, June 2007.

[56] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” *Proc. 8th Int’l Conf. Computer Vision*, vol. 2, pp. 416–423, July 2001.

[57] R. Megret and D. DeMenthon, “A survey of spatio-temporal grouping techniques,” Technical Report CS-TR-4403, Language and Media Processing, Univ. of Maryland, Aug. 2002.

[58] M. Meribout and M. Nakanishi. “A new real time object segmentation and tracking algorithm and its parallel hardware architecture,” *Journal of VLSI Signal Processing*, vol. 39, no. 3, pp. 249-266, 2005.

[59] S. Mohan and L. Mani, “Construction of 3d models from single view images: a survey based on various approaches,” *Proc. Int. Conf. Emerg. Trends Elec. Comp. Tech.*, pp. 557–562, 2011.

[60] G. Mori, “Guiding model search using segmentation,” *Proc. 10th Int’l Conf. Computer Vision*, vol. 2, pp. 1417–1423, 2005.

[61] E. Navon, O. Miller, and A. Averbuch, “Color image segmentation based on adaptive local thresholds,” *Image and Vision Computing*, vol. 23, no. 1, pp. 69-85, 2005.

[62] OpenMP Architecture Review Board, "OpenMP Application Program Interface Version 3.0," http://www.openmp.org/mp-documents/spec30.pdf, May 2008.

[63] C. Pantofaru and M. Hebert, "A comparison of image segmentation algorithms," Technical Report CMU-RI-TR-05-40, The Robotics Institute, Carnegie Mellon University, Sept. 2005.

[64] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," *Proc. European Conf. Computer Vision*, pp. 460-473, 2008.

[65] S. Paris and F. Durand, "A topological approach to hierarchical segmentation using mean shift," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 1-8, 2007.

[66] X. Ren and J. Malik, "Learning a classification model for segmentation," *Proc. 9th Int. Conf. Comp. Vision*, vol. 1, pp. 10-17, 2003.

[67] J.B.T.M. Roerdink and A. Meijster, "The watershed transform: definitions, algorithms, and parallelization strategies," *Fundamenta Informaticae*, vol. 41, pp. 187-228, March 2000.

[68] J. Serra, "Connections for sets and functions," *Fundamenta Informatica*, vol. 41, pp. 147-186, 2000.

[69] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt, "Hierarchy and adaptivity in segmenting visual scenes," *Nature*, vol. 442, no. 7104, pp. 810-813, June 2006.

[70] J. Shi and J. Malik, "Motion segmentation and tracking using normalized cuts," *Proc. 6th IEEE Int'l Conf. Computer Vision*, pp. 1151-1160, 1998.

[71] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905, 2000.

[72] S.M. Smith and J.M. Brady, "ASSET-2: real-time motion segmentation and shape tracking," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 814-820, 1995.

[73] C. Sorrel, "What's Inside: The iPhone 4S Camera," *Wired*, http://www.wired.com/gadgetlab/2011/10/whats-inside-the-iphone-4s-camera/, 5 October 2011.

[74] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st edition, Springer, 2010.

[75] R. Unnikrishnan, C. Pantofaru, and M. Hebert, "Toward objective evaluation of image segmentation algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 929–944, 2007.

[76] C. Van Rijsbergen, *Information Retrieval*, 2nd edition, Department of Computer Science, University of Glasgow, London: Butterworths, 1979.

[77] L. Vincent and P. Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 1991.

[78] D. Wang, “A multiscale gradient algorithm for image segmentation using watersheds,” *Pattern Recognition*, vol. 30, no. 12, pp. 2043-2052, 1997.

[79] J. Wang and E. Adelson, “Representing moving images with layers,” *IEEE Trans. Image Processing*, vol. 3, no. 5, pp. 625-638, 1994.

[80] J. Wang, P. Bhat, R.A. Colburn, M. Agrawala, and M.F. Cohen, “Interactive video cutout,” *ACM Trans. Graphics*, vol. 24, no. 3, pp. 585–594, 2005.

[81] B. Wojdyla, “GM Puts the First Robotic Vision System in a Production Car,” *Popular Mechanics*, http://www.popularmechanics.com/cars/news/industry/gm-puts-the-first-robotic-vision-system-in-a-production-car?click=main_sr, 30 September 2011.

[82] T. Wolverton, “Monitoring your vitals with a webcam,” *Lowell Sun*, http://www.lowellsun.com/tech/ci_22742589/wolverton-monitoring-your-vitals-webcam, 11 March 2013.

[83] J. Wright and A. Yang, “Image Segmentation Benchmark Indices Package,” University of California, Berkeley, http://www.eecs.berkeley.edu/~yang /software/ lossy_segmentation/SegmentationBenchmark.zip, March 2007.

[84] Z. Wu and R. Leahy, “An optimal graph theoretic approach to data clustering: theory and its application to image segmentation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1101-1113, Nov. 1993.

[85] L. Xu and M.I. Jordan, “On convergence properties of the EM algorithm for Gaussian mixtures,” *Neural Computation*, vol. 8, pp. 129–151, 1996.

[86] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: a survey," *ACM Computing Surveys*, vol. 38, no. 4, article 13, pp. 1-45, Dec. 2006.

[87] R. Zurer, “Bun-Making Goes High Tech,” *Wired*, http://www.wired.com/ magazine/2011/09/st_perfectbuns/, 27 September 2011.

VITA

Dana Forsthoefel was born on November 17th, 1986 in Tallahassee, Florida. In 2004, she graduated one year early from John Paul II Catholic High School and went on to attend the Georgia Institute of Technology. In May 2008, she graduated *with Highest Honor* with a Bachelor of Science in Computer Engineering. She was accepted into the 5-year joint BS/MS degree program for highly qualified students, allowing her to graduate in 2009 with a Master of Science in Electrical and Computer Engineering from Georgia Tech. During her Masters program, her primary technical focus was in Computer Systems and Software with a minor in Computer Science. Her secondary technical interest areas included VLSI Systems and Telecommunications. In August of 2009, she began her doctoral program in Electrical and Computer Engineering at Georgia Tech in the Mobile Vision and Embedded Systems (MoVES) Laboratory. While in pursuit of her doctoral degree, she held research assistantships both in the MoVES lab and in the Sensors and Electromagnetic Applications Lab (SEAL) at the Georgia Tech Research Institute (GTRI). In addition, she solely taught a core computer engineering lab class for a semester during her doctoral program.

Dana was awarded the Google Anita Borg Memorial Scholarship in 2009 as support for her continued graduate studies in computing and technology. In 2012, she received a distinguished performance award from GTRI for her outstanding performance and superior technical achievements. Her primary research interests include embedded computer vision and fast, resource-efficient, automated image processing. She is a member of IEEE, SWE, WECE, and Eta Kappa Nu.