# การสรุปวีดิทัศน์สอดส่องดูแลและค้นหาแนววิถีแบบทันทีโดยการตรวจหาการชนที่เลื่อนในทางตรง

นายสิริวัฒน์ เกษมวัฒนาโรจน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

# REAL-TIME VIDEO SURVEILLANCE SUMMARIZATION AND TRAJECTORY SEARCH USING DIRECT SHIFT COLLISION DETECTION

Mr.Siriwat Kasamwattanarote

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science and Information Technology

Department of Mathematic and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2011

| | |
|---|---|
| Thesis Title | REAL-TIME VIDEO SURVEILLANCE SUMMARIZATION AND TRAJECTORY SEARCH USING DIRECT SHIFT COLLISION DETECTION |
| By | Mr.Siriwat Kasamwattanarote |
| Field of Study | Computer Science and Information Technology |
| Thesis Advisor | Assistant Professor Nagul Cooharojananone, Ph.D. |

---

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

..... Dean of the Faculty of Science
(Professor Supot Hannongbua, Ph.D.)

THESIS COMMITTEE

..... Chairman
(Assistant Professor Rajalida Lipikorn, Ph.D.)

..... Thesis Advisor
(Assistant Professor Nagul Cooharojananone, Ph.D.)

..... External Examiner
(Rawesak Tanawongsuwan, Ph.D.)

สิริวัฒน์ เกษมวัฒนาโรจน์: การสรุปวีดิทัศน์สอดส่องดูแลและค้นหาแนววิถีแบบทันทีโดยการตรวจหาการชนที่เลื่อนในทางตรง. (REAL-TIME VIDEO SURVEILLANCE SUMMARIZATION AND TRAJECTORY SEARCH USING DIRECT SHIFT COLLISION DETECTION) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ. ดร. นกุล คูหะโรจนานนท์, 59 หน้า.

การทบทวนเฝ้าระวังเนื้อหาสำหรับวีดีทัศน์เพื่อการตรวจตารักษาความปลอดภัย เป็นงานที่ใช้เวลานาน กระบวนการแบบทันทีแนวใหม่ถูกนำเสนอ โดยยึดหลักการเลื่อนอุโมงค์วัตถุในแกนเวลาสำหรับการสรุปวีดิทัศน์ เพื่อจุดมุ่งหมายในการลดระยะเวลาที่ใช้งานในงานที่เวลามีความสำคัญยิ่งยวด อุโมงค์ถูกนำเสนอเพื่อใช้เป็นโครงสร้างวัตถุอิสระในแกนเวลา งานวิจัยชิ้นนี้ได้นำเสนอสามขั้นตอน เพื่อที่จะสรุปวีดิทัศน์ที่ไม่สิ้นสุดให้มีความยาวสั้นลงและปรับแต่งได้ พร้อมทั้งมีส่วนต่อเติมเพื่อให้เข้าใจวัตถุอิสระได้ การตรวจหาการชนที่เลื่อนในทางตรงถูกพัฒนาขึ้น เพื่อการเลื่อนอุโมงค์วัตถุในแกนเวลาให้ติดกันเร็วอย่างมาก วีดิทัศน์ที่สรุปแล้วสามารถนำมาปรับแต่งได้ด้วยหลายวิธี การค้นหาแนววิถีเป็นวิธีแรกที่ถูกนำมาใช้ ซึ่งประยุกต์มาจากกระบวนการเดียวกับการตรวจหาการชนที่เลื่อนในทางตรงของเรา การแปลงระยะทางในทางตรง จะให้ค่าความสอดคล้องของแนววิถีระหว่างอุโมงค์ กับแนววิถีจากผู้ใช้กำหนดได้ในทันที ซึ่งขั้นตอนที่สำคัญในการบ่งชี้วัตถุอิสระ คือการลบพื้นหลังออก การปรับเปลี่ยนพื้นที่อย่างยืดหยุ่นถูกนำเสนอให้เป็นวิธีการลบพื้นหลังออก ซึ่งมีจุดมุ่งหมายเพื่อใช้เลือกวัตถุเบื้องหน้าที่ดีที่สุดก่อนจะสร้างอุโมงค์ การปรับเปลี่ยนพื้นที่อย่างยืดหยุ่นยังมีส่วนช่วยในการสรุปวีดีโอให้แม่นยำมากขึ้น กระบวนการที่ได้นำเสนอไป สามารถตอบสนองผลลัพธ์ได้ทันทีด้วยวิธีการที่มีประสิทธิภาพ โดยไม่สูญเสียเหตุการณ์ที่สำคัญของกระแสวีดิทัศน์ต้นฉบับ

ภาควิชา .....คณิตศาสตร์และวิทยาการคอมพิวเตอร์..... ลายมือชื่อนิสิต .....

สาขาวิชา ....วิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ..... ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา ..... 2554 .....

## 5273616223: MAJOR COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
KEYWORDS: VIDEO SUMMARIZATION / OBJECT TRACKING / TUNNEL PROCESSING / HOG / DIRECT SHIFT COLLISION DETECTION / DISTANCE TRANSFORM / FILM MAP GENERATION / JUST-IN-TIME RENDERER / DYNAMIC REGION ADAPTATION / BACKGROUND SUBTRACTION / FOREGROUND EXTRACTION

SIRIWAT KASAMWATTANAROTE : REAL-TIME VIDEO SURVEILLANCE SUMMARIZATION AND TRAJECTORY SEARCH USING DIRECT SHIFT COLLISION DETECTION. ADVISOR : ASST. PROF. NAGUL COOHAROJANANONE, PH.D., 59 pp.

Reviewing on video surveillance contents for security monitoring is a time-consuming task. A novel real-time video surveillance summarization framework intended for minimizing time usage for time critical tasks is presented, based on shifting of object tunnels in time-space. Tunnel is proposed as an individual structure of time-dimensioned object. In order to summarize an endless video into a shorter duration with customization and an extension for understanding of individual object, this research proposes three real-time algorithms. Direct Shift Collision Detection (DSCD) has been implemented for extremely fast shifting tunnels together in time-space. A summarized video can be customized by many approaches. An early trajectory searching is applied with the same technique of our DSCD. Direct Distance Transform (DDT) instantly gives trajectory similarity between tunnels and users query. The most important step for identifying each individual object is background subtraction. Dynamic Region Adaptation (DRA) is proposed as another background subtraction algorithm aimed to select the best objects foreground before making a tunnel. DRA also helps DSCD to summarize a video more accurately. The proposed framework is able to respond the results by real-time performance approaches without losing the major events of original video stream.

Department : .....Mathematic and Computer Science..... Student's Signature .....
Field of Study : ..... Computer Science and Information Technology ..... Advisor's Signature .....
Academic Year ..... 2011 .....

# Acknowledgements

Thesis is a compilation of the creative process that has been accumulated from one person life span. All knowledge from this thesis will be worthless if no one take them learn for making just another branch of knowledge. However, this thesis cannot be done solely by one author. The most important thing from really the first letter till the end is not about the story inside, but it is about the synergy of many involved people. One of an important person is my advisor, Asst. Prof. Nagul Cooharojananone for suggesting this topic to me and generously offering guidance and support during my study here. He is also gives me concepts of many perspectives for shaping my ideas. Asst. Prof. Rajalida Lipikorn who gives me comments and helps me refining my words with her natured mind. I also would like to express my gratitude to my bachelor degree advisor at Mahidol University, Dr. Rawesak Tanawongsuwan, who inspired me to be in this research area. Perhaps I will be in another research area without him.

I greatly appreciate for the International Internship program from National Institute of Informatics (NII), Japan, who gives me chances to live and study there. Since then, I have discovered many useful and invaluable experiences for my life. With great gratitude to Dr. Kitamoto Asanobu and Dr. Shinichi Satoh, the advisors who provide support, ideas and feedbacks for both research and living while staying in Japan. Prof. Philippe Bolon from University of Savoy who points me out the light from the darkness then explodes me to another world of color space. And one indispensable person is Dr. Sirod Sirisup, who provides me advices, teaching, and encouragement as well as his good quotations from a dining table can change my life forever. Living in the country, where they keep balancing between technologies and the mental growth of their population always made me remind many positive things for wishing to apply with our country in some day.

For the most grace direct to whom incomparable who sacrifice their life and always given me the best opportunity and also support all invaluable resource to my pilgrimage of knowledge. I would like to dedicate the whole knowledge to my family. Even though this is just only a small fraction, I will always give them gratefulness from the deepest of my heart throughout of my life.

# Contents

Chapter Page

# List of Tables

# List of Figures

Figure Page

Figure Page

# CHAPTER I

# INTRODUCTION

Video surveillance systems have been widely used as a common security monitoring system around the World. However, it is very time consuming for browsing and analyzing any subsequences along the large amount of footages contained endless information. Typically, reviewing an endless surveillance video involves manual seeking into recorded raw video. For example, browsing for a specific people in a video of one surveillance camera requires a playback device with capabilities for controlling frame positions such as jog/dial, backward, forward, go to, and etc.. These common controls are time consuming methods which require human experiences and abilities for identifying an individual activity time by time. Therefore, all those human abilities will be decreased by time, which affect the overall accuracy of browsing and analyzing the surveillance videos.

## 1.1 Research objectives

This research aims at building a video surveillance summarization framework as a replacement of a conventional system as seen in Fig. 1.1. for help reducing the time usage for rush exploring on the endless video surveillance contents. The objectives are:

1. To build a framework of a video summarization system.
2. To develop a real time system for summarizing a surveillance video.
3. To search human trajectory in a surveillance video.

## 1.2 Problem Formulation

1. Analyzing of surveillance video is a time consuming process, which requires an ability of a human to identify objects along the time. Thus the accuracy will be decreased in time.
2. Most summarization results of existing approaches contain all irrelevant information which leads to an ignorance of an appropriate human ability to be used in an analyzing process.

Figure 1.1: Conventional video surveillance system that involves with an operation on human abilities.

## 1.3 Expected outcomes

1. A framework of a video summarization that is extendable for the further research approaches.
2. A real time video summarization system that will be widely used by many organizations.

## 1.4 Scopes of work

1. Summarization is based on a non-PTZ video surveillance camera.
2. Human is a target object to be summarized.
3. Human size in a video should not be less than the size of a training set.
4. Experiments will not cover a crowded pedestrian of a surveillance video.
5. Variance of lighting condition such as day-night lighting condition is not considered.

## 1.5 Related works

Various approaches of video summarization have been proposed for the same aspect, which try to reduce irrelevant information such as space and time. The easiest time reducing technique is frame skipping or video skimming (Smith, 1997), where several frames are skipped according to user intentions such as object, color, motion, etc. Adaptive video fast forwarding was developed for the purpose of adjusting a playback speed of a video (Peker and Divakaran, 2004; Petrovic et al., 2005). However, the speed increases resulting in missing a lot of information as well.

In both approaches mentioned above, an entire frame is a key of summarizing. Another technique known as video montage (Kang et al., 2006), which is a spatial-temporal approach, considers space as an additional key. However, the effect of shifting of spatial dimension creates unavoidable artifact on images between different objects. The later approaches use an advantage of shifting objects through time dimension while keeping spatial locations intact, resulting in contexts unchanged. Video condensation by ribbon carving (Li et al., 2009) is an approach for removing ribbons (ripple frame in a spatial-temporal dimension) inspired by seam carving (Avidan and Shamir, 2007). However, ribbon only works well with a few differences of speed and direction of the moving objects, since a flex parameter, which control ribbon flexibility; have a limitation related to smoothness of stitched image after removal. Although removing ribbon result to reduce irrelevant information in space and time, it requires taking time for recalculating the energy of the rest moving objects

Video synopsis has been proposed with many complex computation steps (Rav-Acha et al., 2006; Pritch et al., 2007, 2008). Most of them are cost and energy computations that are organized into online phase and response phase. In online phase, object detection is used in time space to generate tube, then insert and remove a detected tube to and from an object queue. In response phase, the synopsis video is built by first constructing time lapse background, and then computing the corresponding time of tubes in the synopsis video. And finally, rendering the tubes with background to produce the synopsis result, which takes time for stitching moving objects to the correspondence time and also it depends on the amount of activity in the time period of interest.

Our recent methods for fast and robust object shifting in time space has been proposed in (Kasamwattanarote et al., 2010), where distance map based collision detection were first applied. Direct Shift Collision Detection (DSCD) is an object collision detection technique that can calculate a shortest distance between two groups of objects within $O(n)$. Just in Time renderer (JIT

renderer) has been developed in (Kasamwattanarote et al., 2010) for help reducing the CPU usage by processing only necessary frames. Even though the system can produce a summarization result, it also provides information such as trajectory of moving object for help identifying each specific tunnel.

Location based trajectories are more important with high consideration on a moving pattern of an object. Various approaches use different information to describe an individual trajectory such as shape, starting and ending time, location, velocity and acceleration of a moving object. Dynamic time wrapping (DTW) has been implemented in (Berndt and Clifford, 1996) which allow time stretching for moving objects to get a better match between trajectories. Several approaches in (Vlachos et al., 2002b,a) use longest common sub sequence (LCS) as a similarity measure for trajectories. The advantage of LCS is that it allows matching of moving object subsequences. DTW and LCS distance matching algorithms are commonly uses in widespread, however both of them have time complexity of $O(n^2)$. Calculating a trajectories similarity by using the definition of time and sub-sequences has been presented in (van Kreveld and Luo, 2007), where the speed of an object leads to temporal shifting inside the vertices which can be considered as dissimilar. Problems have been categorized according to the criteria that timing is fixed and/or whether the starting times of the sub-trajectories are equal. However, the distance function is the main measurement and the best time complexity of an exact match similarity with various duration is $O(n^2)$.

Another approach is based on edit distance (ED) in order to overcome the inefficiency of the previous approach in the case of noise and obstacles on the trajectory. Chen et al. (Chen et al., 2005) proposed a new distance function called edit distance on real sequences (EDR). EDR removes noise then changes one trajectory to another before measure. Thus, this technique increases the accuracy especially with the trajectories having Gaussian noise. Similarity search by using area in-between trajectories has been proposed in (Pelekis et al., 2007). GenLIP algorithm has been used as the spatial similarity search between trajectories. Also, only spatial similarity computation takes time complexity of $O(n \log n)$.

Clustered synopsis of video surveillance has been proposed in (Pritch et al., 2009), which takes the benefits of trajectory to reduce the information being summarized. Motion distance is defined as the similarity measure between two activities. This approach takes both appearance distance from SIFT features and motion distance into the creation of the training set of unsupervised clustering, which yields too much time consuming to search for trajectories of moving

object in a surveillance video.

We proposed a real-time trajectory search by using direct distance transform (DDT) (Cooharojananone et al., 2010), where distance map based for similarity measuring between trajectories were first introduced, which aim for fast searching on only location definition. By generating a distance map from trajectory information of a moving object aims to provide a fast and direct accessing to the similarity values, which results to retrieve trajectory similarity by accumulating values from a distance map through a specific trajectory within real-time. Also, the system will rank up all relevant objects ordered by similarity value, which helps increasing the efficiency of the activity tunnels for being analyzed.

### 1.6 Organization of thesis

In this paper, we present a real-time video surveillance summarization framework built from our previous works in (Kasamwattanarote et al., 2010; Cooharojananone et al., 2010) as a time consuming reductive tool, which motivate by real-time approach aimed for help minimizing time usage for browsing and analyzing activities in a security monitoring and time-critical related tasks. We organize this paper as following. In section 1.7, we describe our framework and give the details of frameworks architecture design as a combination of multiple components. We show how simple, fast, and efficient of our novel algorithms work, which categorize in recorder module and player module, in section 2 and 3 such as DSCD, Film Map Generation, JIT Renderer, and DDT. Also, we introduce Dynamic Region Adaptation (DRA) as a contour based background subtraction. Then, we report results in section 4, which are in term of images captured from summarized video and time performance. Finally, the conclusion is discussed in section 5.

### 1.7 Propose Framework

We realize that our goal is to build a real-time video surveillance summarization framework, so we organize the processes into two core modules for balancing workloads. Therefore, both modules can be executed independently or paralleling to handle the tasks (see Fig. 1.2). The first module is called a recorder module. This module first captures video and detects a region of interest of objects in each video frame. Then, all objects are analyzed and processed to generate completed tunnels (section 2.1). Next, rectangular tunnels were performed by our DRA background subtraction (section 2.2) to make a moving silhouette for each moving object corresponded to an individual tunnel. After that, DSCD is then used for calculating a collision distance

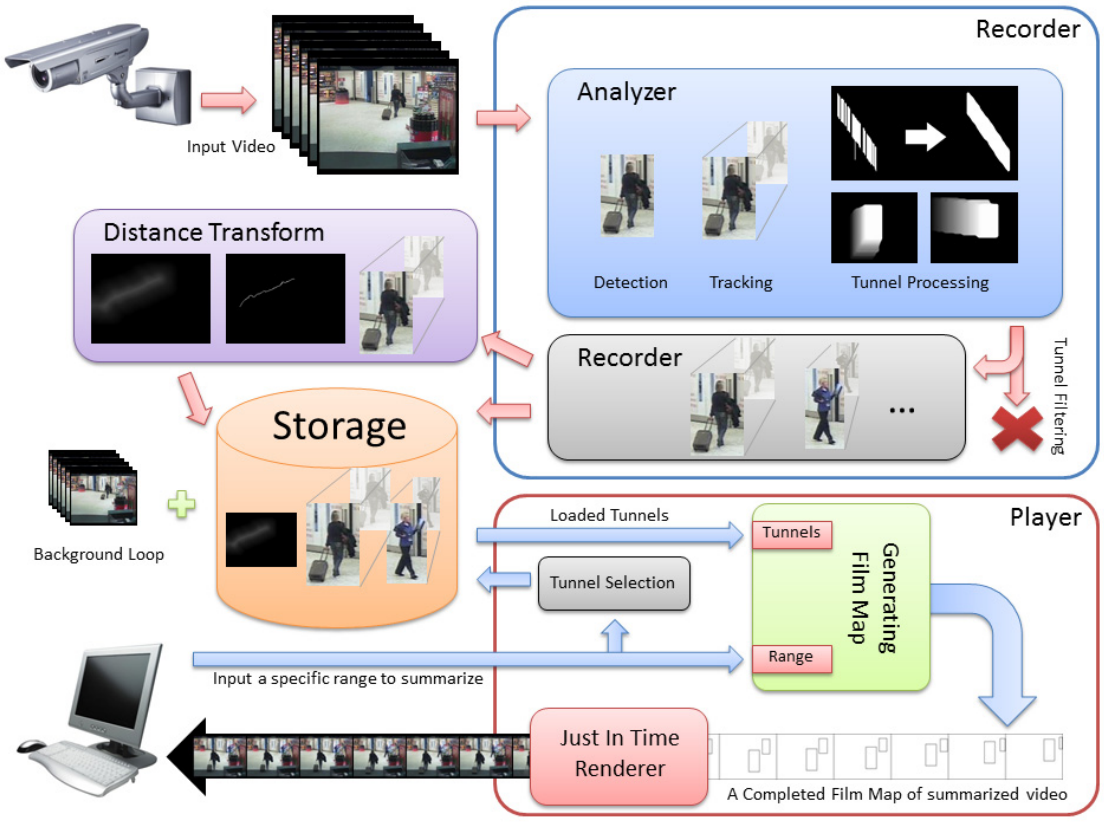



Figure 1.2: The proposed framework.

(section 2.3) and finally each tunnel is recorded separately.

The second module is called a player module which is for playing the summarized surveillance video. This module first gets a user input of playing duration such as time period. Then, the system collects tunnels which correspond to the specified range and also background source file is initialized. Film map generation is then used for making a draft version of a summarized video without rendering while JIT renderer is used for rendering only necessary frames to be shown (section 3.1). Finally, buffer is read out in real-time according to user seeking position.

Inside the first module, we embed all necessary information for describing an individual tunnel such as time, moving speed, direction, and moving trajectory. For fast searching moving trajectories, we also apply the distance transform generation within the recorder module for pre-generating the distance transform as the trajectory information for each moving object. The player module allows user to search for trajectories by just drawing a moving path through the screen. Then, DDT is then applied for searching the most similar trajectory (section 3.2) by direct accessing through all provided trajectory information in the player module. Also, these trajectory similarity results can be used for applying as a tunnel filter of summarized video.

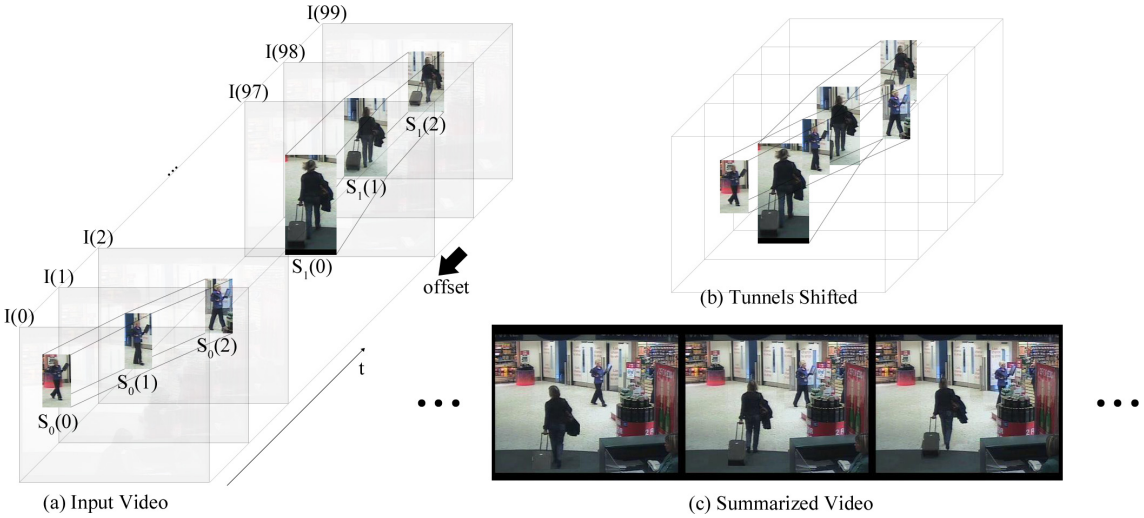



Figure 1.3: Our schematic illustration: (a) two different tunnels in 3D spatial-temporal volume (b) tunnels were shifted in time space (c) frame slides of a summarized video.

# CHAPTER II

# RECORDER MODULE

To provide information useful for a player module, we integrate the major processes for recording each tunnel and pre-calculating necessary parameters in real-time. This section describes the steps through recorder module, which contains a process of making a raw tunnel with necessary information, extracting a moving silhouette, and also pre-summarizing tunnels ahead of time with our simple, fast, and robust algorithm.

## 2.1 Tunnel Computing

In order to create a summarized video based on activity tunnel, each appropriate tunnel should be detected using any object detection algorithm. In this work, we apply HOG for human detection (Dalal and Triggs, 2005) by using a default human descriptor as our main detector.

### 2.1.1 Slice Tracking

To define a tunnel, $I(t)$ be an image or a frame in a time space where $t$ is a frame number, $N$ is the total number of frames, and $1 \leq t \leq N$. A detected object $O_j(t)$ in Fig. 2.1 corresponds to a region of such object at frame $t$ and $j$ is the index of an object detected from $I(t)$. A tunnel is composed of multiple slices $S(i)$ as seen in Fig. 1.3 where $i$ is a slice number, $M$ is the total number of slices, $0 \leq i \leq M$, and $T(n)$ in Fig. 2.2 is a tunnel, where $n$ is a tunnel number, $K$ is the total number of tunnels, and $0 \leq n \leq K$.

In order to track a tunnel, we need to find the closest detected object to a tunnel by Euclidean distance and then check the confidence value $C(m, o)$ by using multi attribute utility theory (MAUT), to estimate the possibility of being the next slice of such tunnel. The confidence equation can be described as follows

$$C(m, o) = w_m||m|| + w_o||o|| \tag{2.1}$$

where $m$ is a missing frame to the last slice of a tunnel, $o$ is an overlapping area of the last slice of a tunnel with the closest slice of all detected objects, $||m||$ and $||o||$ are the utility values,


Figure 2.1: Multiple objects can be detected in one frame.

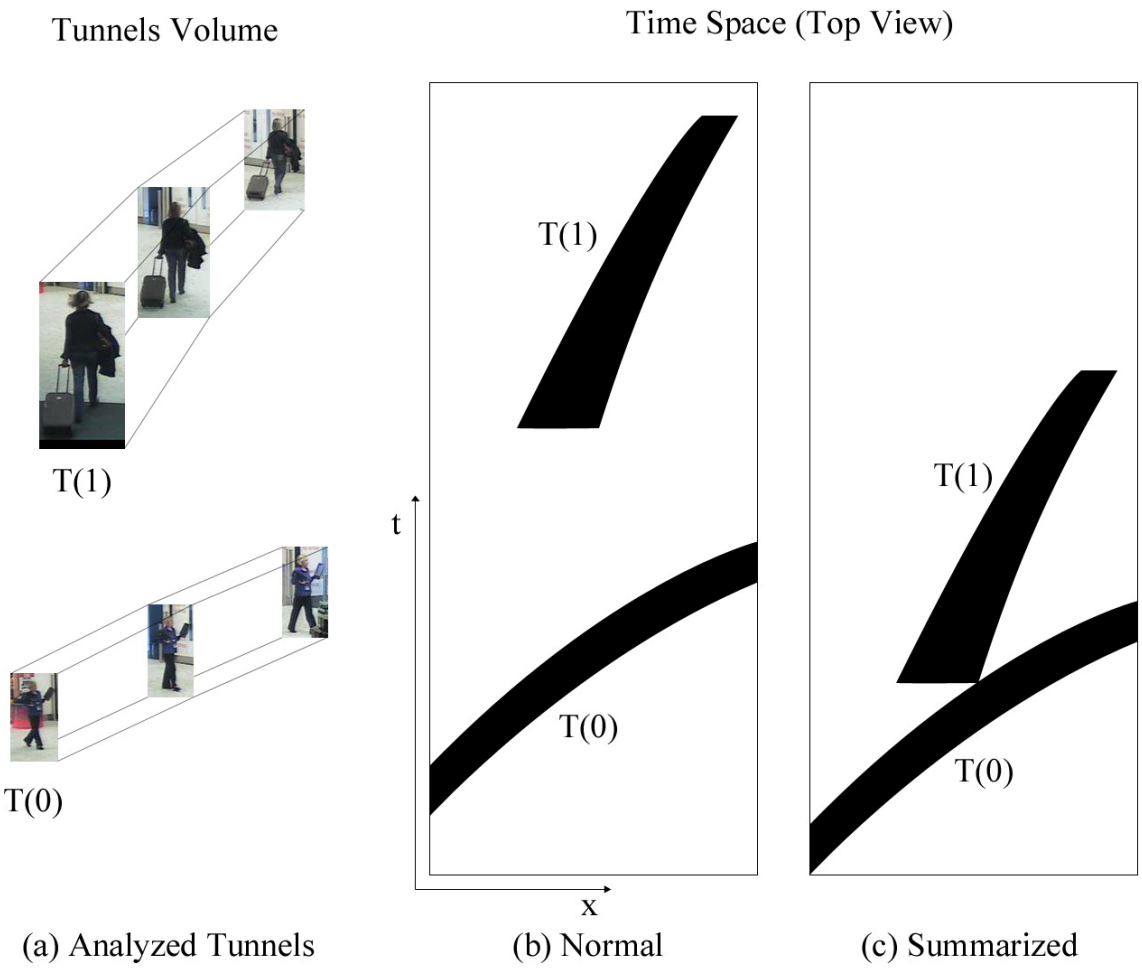
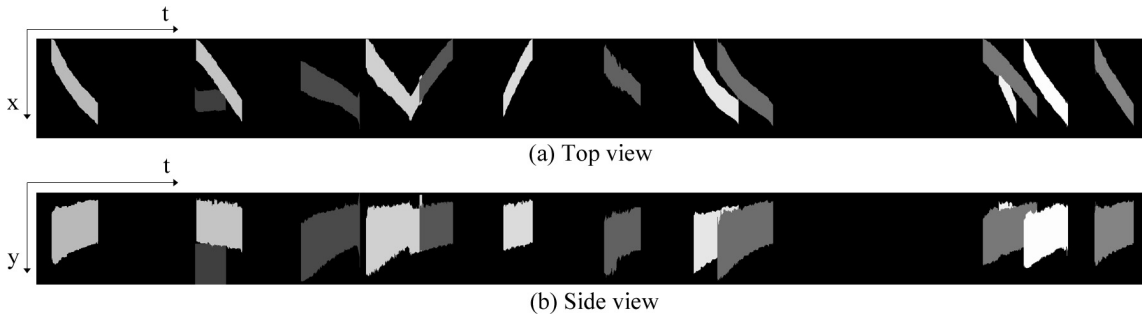



Figure 2.2: Detected tunnels: (a) tunnels of two different objects (b) top view time space (c) time space after summarized by shifting tunnels through time while keeping spatial locations intact.




(a) Top view

(b) Side view

Figure 2.3: Time space shows multiple tunnels after tracking and interpolating step.

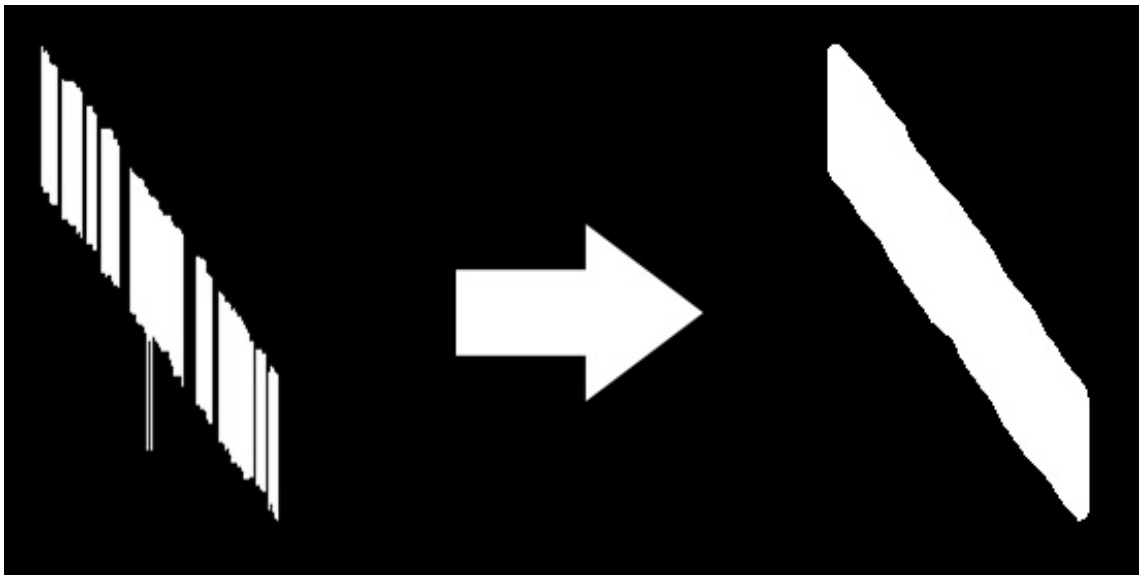


Figure 2.4: Top view of the time slice. (left) An unprocessed tunnel. (right) A processed tunnel.

$w_m$ and $w_o$ are the weight of each attribute with the property that

$$w_{total} = w_m + w_o = 1 \tag{2.2}$$

In this work, we use $w_m = 0.3$ and $w_o = 0.7$ since our tunnel tracking emphasizes on the overlapping area rather than the missing frame. The values of $m$ and $o$ are normalized to 0-1 scale. Thus, the highest and the lowest values of $C(m, o)$ will be 1 and 0 respectively.

The confidence value will be used to indicate the possibility of a new detected object to be a new slice of the closest tunnel, therefore, we use the threshold of 0.8 to measure which slice has the confidence value greater than or equal to the threshold; it will be accepted as a new slice of that tunnel. The results are shown in Fig. 2.3.

### 2.1.2 Tunnel Processing

An original tunnel in Fig. 2.4(left) shows a raw tunnel after being analyzed from multiple slices where many missing slices and noisy slices (peek in width or height) can be notified. These problems are caused by object occlusion, color merging, and some noises from the camera, which affect the result of our object detection. In the summarizing step, a process of calculating collision distance will be affected a lot by most of the noisy slices because the collision should not occur on a noisy slice.

To eliminate all the problems, tunnel processing should be performed at the video capturing step which may result in good captured frames, low noise, clear vision, and a good detected tunnel. However, in such cases, the overall performance will be dropped because of spending more CPU time on it. Thus, our approach needs real time performance, and processes with a raw tunnel that uses less data than processes the whole frame along the time space. The reason is because a tunnel $T(n)$ is a collection of slices $S(i)$, where each slice is a rectangle composed of 4 corner points.

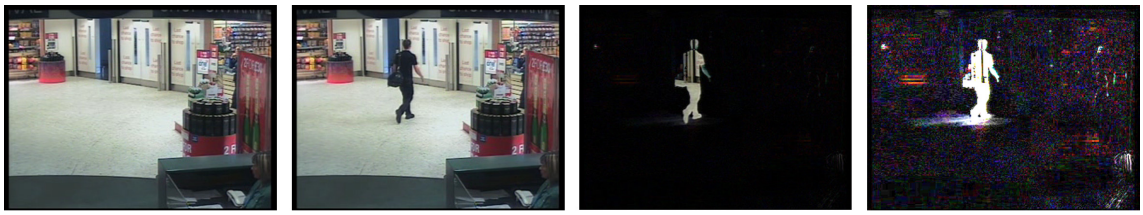
For example, if a tunnel contains 200 slices (200 frames / 25 fps = 8 seconds, most tunnels are within this range), then we need to process only 200 slices $\times$ 4 points per slice or the total of 800 points which are much fewer than the whole time space of 320-pixel width $\times$ 240-pixel height $\times$ 200 frames or 15,360,000 pixels. Then, the result after processing a tunnel will be the tunnel as shown in Fig. 2.4(right).

#### 2.1.2.1 Tunnel Interpolating

As it can be seen from Fig. 2.4 that there are many missing slices which are needed to be filled, a simple interpolation approach works well for this situation (see Fig. 2.3) by using slice properties around the missing frame such as spatial location, dimension, and slice number to generate new slices for those gaps.

#### 2.1.2.2 Tunnel Cleansing

Many approaches for noise reduction can be applied for this step. Most of them try to eliminate noisy data and leave only the original details. In this case, we need only the smoothest tunnel with as less noise as possible (see Fig. 2.4(right)) for recovering from all problems that may occur during our object detection. The most preferably method that can smooth a tunnel as we need is the simple moving average method or SMA. This method is very simple, uses only a



(a) A background Model (b) A people as a new foreground object (c) Absolute differencing result (d) A result alter normalized intensity

Figure 2.5: Images show before and after object moved into an area of background. (a) A background scene shows static object. (b) A people moving into a background area which we need to find the best silhouette mask for the corresponding foreground. (c) Result after done an absolute differencing shows the clear cut foreground area by our vision. (c) In computer, we still have background noise from many possible variations such as a staff's head, moving doors, and the affect of the flickering light source and camera quality.

few operations, and does not require any sorting algorithms; however, it gives us the most suitable result over all other algorithms.

## 2.2 Extracting a moving foreground by Dynamic Region Adaptation

Tunnel is a composed of time-dimensional slices bounded from region of interest of real-time object detection, which generally are rectangle. Shifting tunnel together in time dimension (section 2.3) is a purpose of time gap removal between moving objects, which results to play the same activities in the shorter amount of time. Our DSCD provides two types of summarized result that are non-overlapping tunnel ($DSCD_{multiplier} = 1$) and overlapping tunnel ($DSCD_{multiplier} > 1$). In case of non-overlapping tunnel, the collision distance in time dimension between tunnels is the minimum distance between two consecutive tunnels as seen in Fig. 2.2, which is an illustration of shifting of rectangular tunnel. Although, the time gap was removed, there still exists the space gap between the particular objects inside the rectangles.

According to our approach aimed for shifting only temporal and lefts spatial location intact to avoid making artifact as in (Kang et al., 2006). We applied a dynamic region adaptation (DRA) for help more shifting in time dimension by subtracting the background area from the whole rectangular image. This yields two consecutive tunnels collide closer into the particular object as seen in Fig. 2.6, which will effect to the overall summarization period to be shorter without overlapping tunnel.

Subtracting background from an image, we first need to know background area by finding the differences between foreground and background. The image in Fig. 2.5c shows a result after processed an absolute differencing of two images, we then clearly see foreground area (the

bright pixels) on the dark background. However, it contains a lot of background noise, since a background itself is variance due to the still objects were moved such as an opening door, the flickering of light sources, or even an effect of low quality surveillance camera. Thus, it leads to the masking problem since we cannot create a perfect mask for that foreground by using just absolute differencing result (see Fig. 2.5d).

To find the clear cut area of foreground, we introduce a novel contour based foreground extraction by using DRA. DRA is an algorithm for searching the best cutting threshold for brighter color selection covered almost foreground area without any background noise. We organize the algorithm into three major steps. First, area extraction is a step to process all possible thresholds for generating series of affected amount of pixels coverage of a foreground object. Second step is a process of data transformation, which will transform the series of area coverage into series of slope, difference, and pulse. The last step is a process for finding the best threshold of a corresponding foreground, which will be used to extract almost perfect foreground comparing with and ideal foreground result.

Through all the processes for extracting a moving foreground, our approach can create a silhouette foreground mask for any moving object as seen in Fig. 3.6. Then we applied the best threshold for each tunnel separately for retaining the real-time performance of the overall framework while still be able to extract the clear foreground out of slices of each tunnel (see Fig. 2.10).

### 2.2.1 Area Extraction

Selecting color by using different DRA threshold on a result of absolute differencing between images as seen in Fig. 2.5c will effect to the coverage area of contour as seen in Fig. 2.7. To generate an area graph (see Fig. 2.8a), we take advantages of both color information and luminance information from R, G, and B channels of RGB color space and L channel of CIE L*a*b* color space, where color information can help us identify the high-contrast area and luminance information will be used to identify the low-contrast area.

We process all possible thresholds to build an area graph $Area(S(i))$ as seen in Fig. 2.8a that will be used for searching the best threshold, which is varies according to different foregrounds.

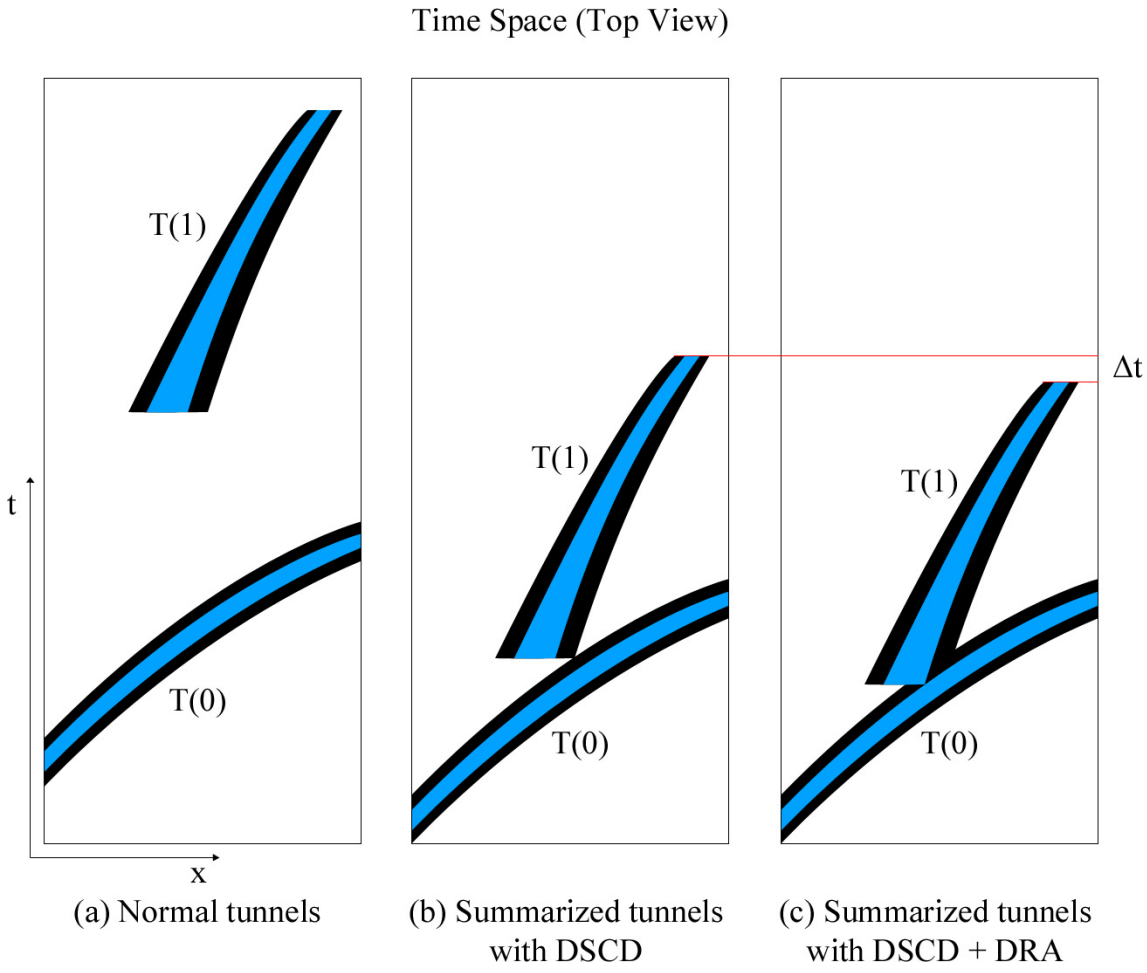



Figure 2.6: An illustration of top view time space shows a comparison between normal, summarized, and summarized with DRA. Where the particular objects is located inside the larger rectangular tunnels. And $\Delta t$ is a shorter time difference with DRA.

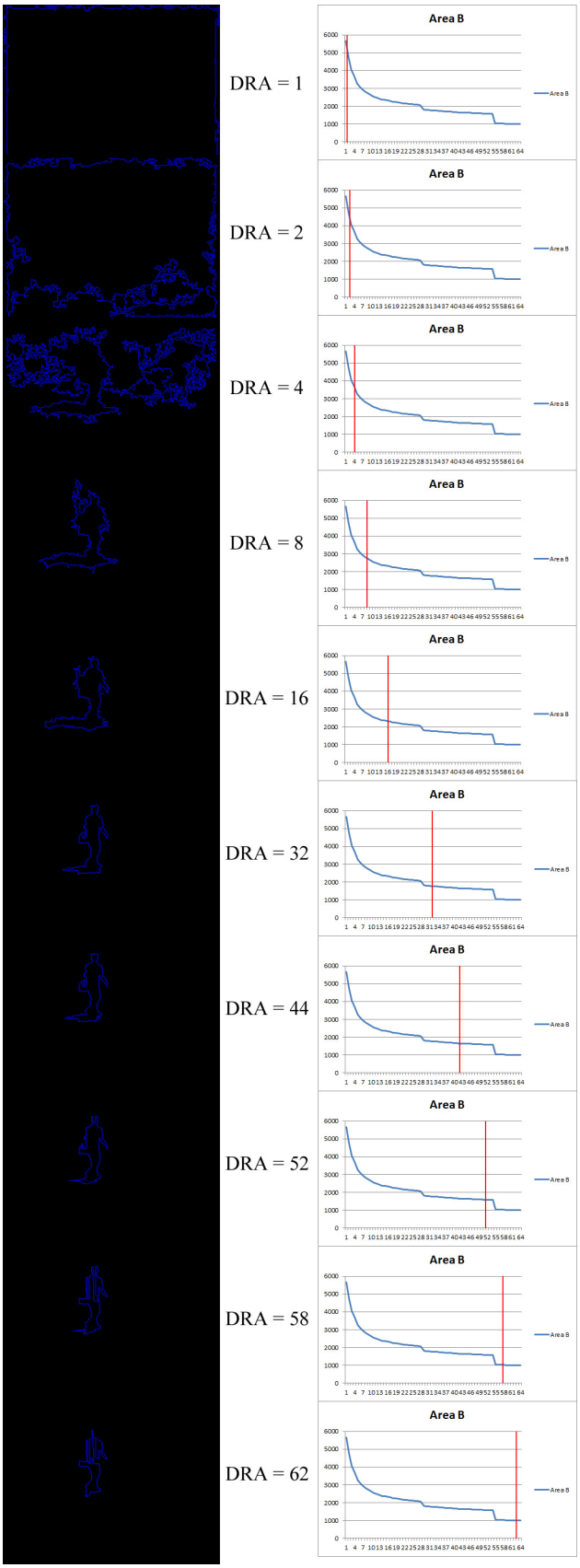



Figure 2.7: Images show different thresholds, which process on B channel of RGB color space, yield different result of the selected foreground area. And the vertical lines for each graph show the selected threshold.

### 2.2.2 Data Transformation

$Pulse(r)$ is a pulse index of the specified rank $r$, which order by ascending of the variation ratio 2.4, that occurs in the pulse graph of slice $i$ as follow

$$Pulse(r) = IndexOf(Rank_r(Pulse_{S(i)})) \tag{2.3}$$

The pulse graph of slice $i$ can be computed by the following

$$Pulse_{S(i)} = Diff(S(i))/Slope(S(i)) \tag{2.4}$$

where $Pulse_{S(i)}$ is a pulse graph of slice $i$ (see Fig. 2.8d), which shows the relation between $Diff(S(i))$ and $Slope(S(i))$, describing the variation ratio among a different value and a slope value on the same index, where a low pulse value indicates a high variation of signal changing on the area graph. In contrast, the pulse value near zero indicates a low variation of area graph. Calculating the different graph $Diff(S(i))$ and the slope graph $Slope(S(i))$ can be done by the following

$$Diff(S(i)) = Diff(Area(S(i))) \tag{2.5}$$

$$Slope(S(i)) = LinearRegress(Area(S(i))) \tag{2.6}$$

where $Diff(S(i))$ is a different graph (see Fig. 2.8b) calculated from a simple moving subtract method along an area graph of slice $i$, and $Slope(S(i))$ is the slope graph (see Fig. 2.8c) calculated from a linear regression method by using a filter size of 7.

### 2.2.3 Finding the Best Threshold

Form our experiments; a threshold for extracting the best foreground is located at the most stable value between the first step and the last step of an area graph, which we describe by the following

$$DRA_{S(i)} = IndexOf(\min_{Pulse(1) \rightarrow Pulse(2)}(|Slope(S(i)))|) \tag{2.7}$$

where $DRA_{S(i)}$ is the best threshold value for the corresponding slice $i$, which is at an index of the minimum value between top two minimum pulses along the slope of slice $i$.

## 2.3 Summarizing by Direct Shift Collision Detection

An endless video from a surveillance camera is hard to browse and analyze without video summarization. Summarizing the video by using object-based video summarization, which uses the detected objects for the selection of related frames, was introduced in (Smith, 1997; Ferman and Tekalp, 1997; Kim and Hwang, 2000). Our approach goes beyond those techniques by shifting the objects through a time space to create the summarized video in order to make a better use of space and time with direct shift collision detection (DSCD).

We introduce a new approach for finding the distance between a point of view and an interested pixel without the use of any scanning methods such as tracing, or brute force method. Our approach uses the advantage of depth map image, which is able to give the distance of an object, for this implementation.

The algorithm is organized into two phases: drawing phase and checking phase. In the drawing phase, a previous tunnel is drawn on an image to create a depth map with distance embedded in a pixel. And in the checking phase, all the corners of all slices of a candidate tunnel will be checked to find the collision distance to the previous tunnels. As a result, our approach can run in $O(n)$.

### 2.3.1 Drawing Phase

In this phase, all slices of a previous tunnel will be drawn on an image from the first slice to the last slice respectively as shown in Fig. 2.9. Normally, a depth map image can be represented using a gray scale image, which is only 256 levels per pixel or an intensity of 0-255. Thus, we try to embed other information such as slice index into those pixels by up scaling the gray shade from an integer to a decimal precision. The image in Fig. 2.9 represents the normalized intensity of our depth map image as a back view of time space. An integer part will be used to represent the overlapping flag, and the fraction part will be used to represent a slice number. For example,

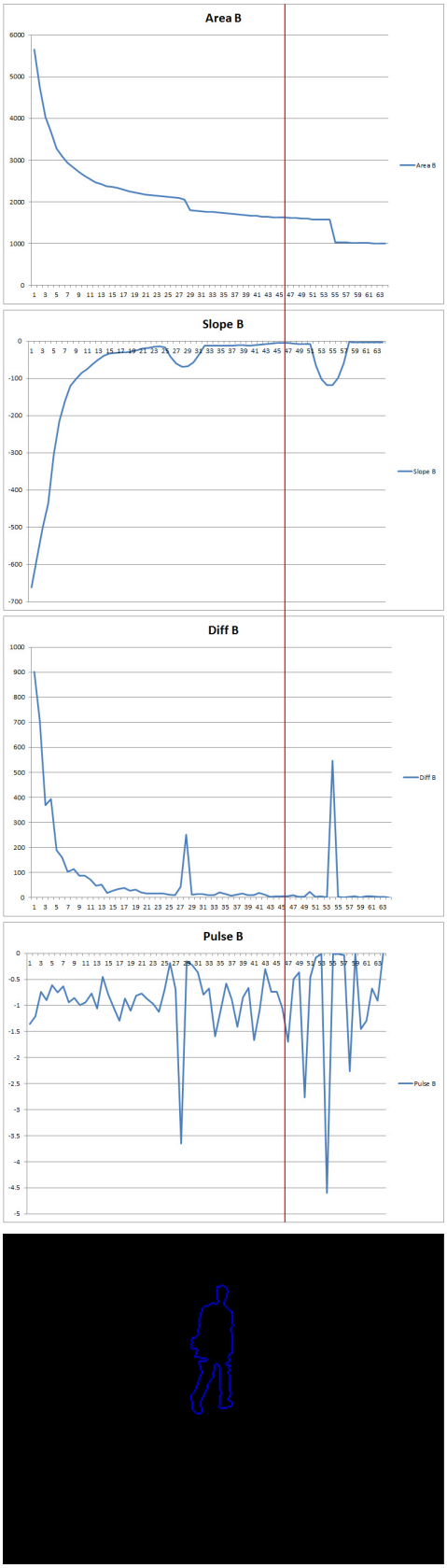



Figure 2.8: The graphs show results for each step along the DRA process on B channel of RGB color space. (a) An area graph calculated by tracing a coverage area of the selected contour from different thresholds. (b) A different graph calculated by simple moving subtracting on an area graph. (c) A slope graph calculated by passing through a linear regression method. (d) A pulse graph calculated by $Diff(S(i))/Slope(S(i))$. (e) Result after select the color by using DRA threshold. And the vertical line shows a selected DRA threshold.

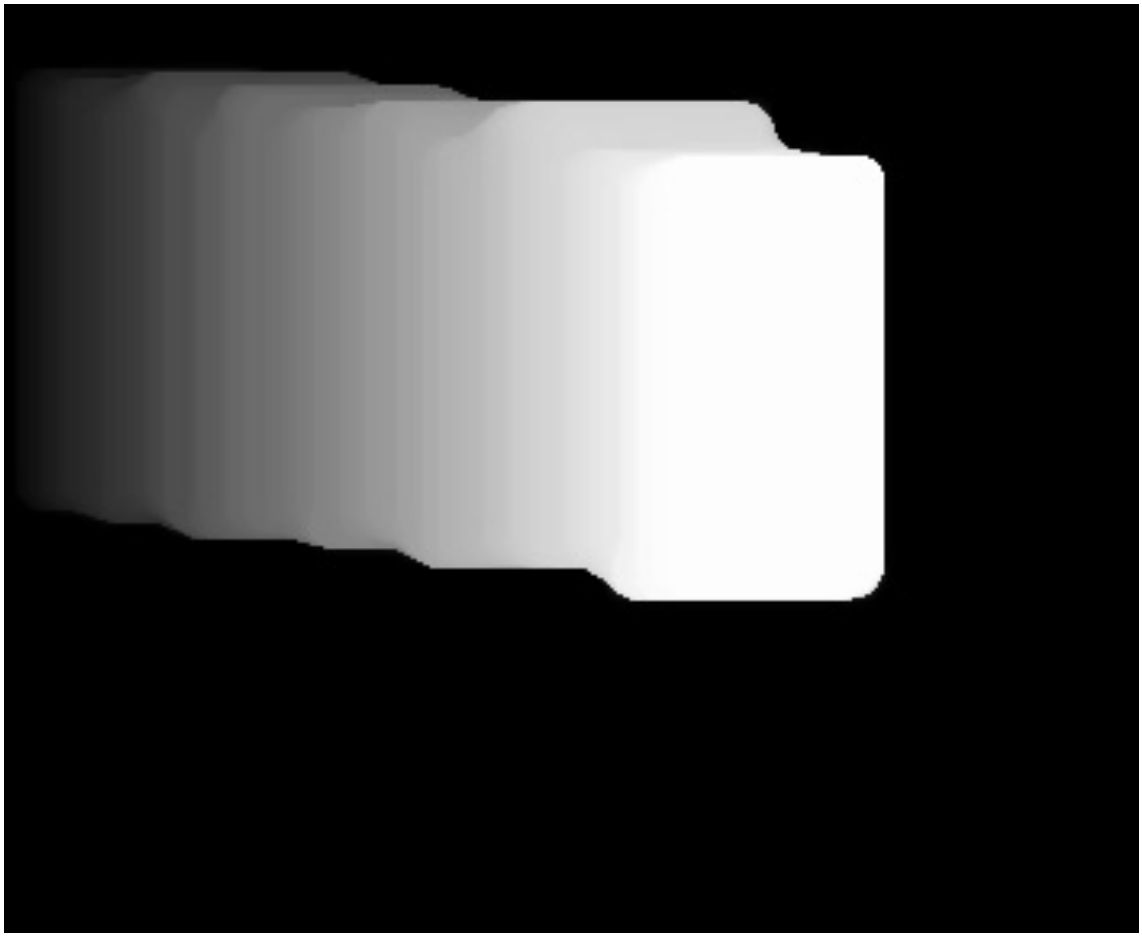


Figure 2.9: A normalized depth map shows back view tunnel in time space.

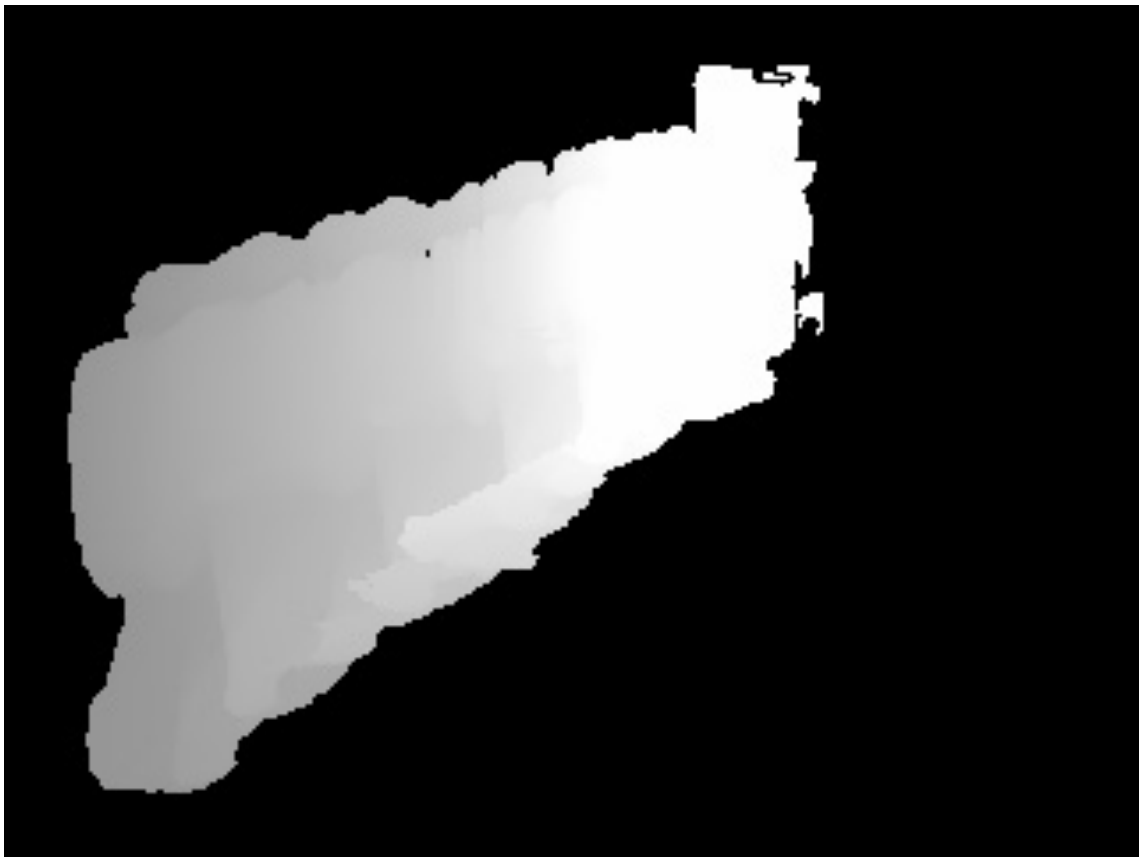


Figure 2.10: A normalized depth map shows back view tunnel after processed with our DRA background subtraction.

to draw slices with gray intensity range from 254.000 to 254.999, it can support the slice up to 1,000 slices per tunnel, or intensity range from 254.0000 to 254.9999 can support up to 10,000 slices etc. For more information on up scaling, we explain more on appendix A.1.

### 2.3.2 Checking Phase

In this phase, instead of drawing into a depth map image, we only need to check with the gray intensity for each point along the contour of all slices of the selected tunnel. Any point that overlaps the previous slice or its location is on the top of the previous slice; its gray intensity will be greater than the threshold. Then the fraction portion will be used to extract a slice index of the collision distance.

The collision distance between two tunnels can be computed from the closest slices among both tunnels giving such distance which is a minimum offset that makes both tunnels stitch together without overlapping. For implementing of checking phase, we explain more on appendix A.2. We define a collision distance for summarizing activity tunnels as the following function

$$DSCD(n) = \min(T(n), I(n-1)) \tag{2.8}$$

where $DSCD(n)$ is a minimum distance between selected tunnel to the previous tunnel, $I(n)$ is a depth map image of the specified tunnel, and the minimum distance DSCD can be implemented as algorithm 1.

**Algorithm 1** Direct Shift Collision Detection

```
1: for all S1 in T(n-1) do
2:     I.draw(S1, intensity)
3: end for
4: for all S2 in T(n) do
5:     if I.overlap(S2) then
6:         minDist = S2.frame - I.frame(S2.contour)
7:         DSCD.update(minDist)
8:     end if
9: end for
```

The time complexity of our algorithm (1) is $O(n)$ which is faster than other closest point

algorithm (2) that takes $O(n^2)$.

**Algorithm 2** Other Closest Point

```
1: for all S1 in T(n-1) do
2:     for all S2 in T(n) do
3:         if S1.overlap(S2) then
4:             minDist = S2.frame - S1.frame
5:             DSCD.update(minDist)
6:         end if
7:     end for
8: end for
```

The collision calculated from DSCD gives us a real-time result of a collision distance per tunnel. When we need to summarize all tunnels of the entire range, the distances will be increased accumulatively from the first tunnel of the range thus we can reduce the time during the compression step in the player module by calculating a collision distance in advance.

To accumulate all DSCDs of the previous tunnel, we use dynamic programming and organize the collision distance into local DSCD and global DSCD. Local DSCD is the distance to the previous tunnel, while global DSCD is an accumulated distance from the first tunnel of the range to the previous tunnel. Dynamic programming can be used to accumulate the previous local distance as the following recurrence relation

$$\begin{aligned} DSCD_G(0) &= DSCD_L(0) \\ DSCD_G(1) &= DSCD_L(1) + DSCD_G(0) \\ DSCD_G(n) &= DSCD_L(n) + DSCD_G(n-1) \end{aligned} \tag{2.9}$$

For example, tunnel A has $DSCD_L = 10$ and tunnel B has $DSCD_L = 5$, then after accumulating all DSCDs from the first tunnel, tunnel A has $DSCD_G = 10$ and tunnel B has $DSCD_G = 15$. Thus, the global DSCD will be used as the offset distance for shifting tunnels through time space as shown in Fig. 2.11. The DSCD distance can be used as a distance for compacting tunnels without overlapping. However, we can apply a multiplier to DSCD for adjusting an offset distance to make overlapping tunnels ($DSCD_{multiplier} > 1$).

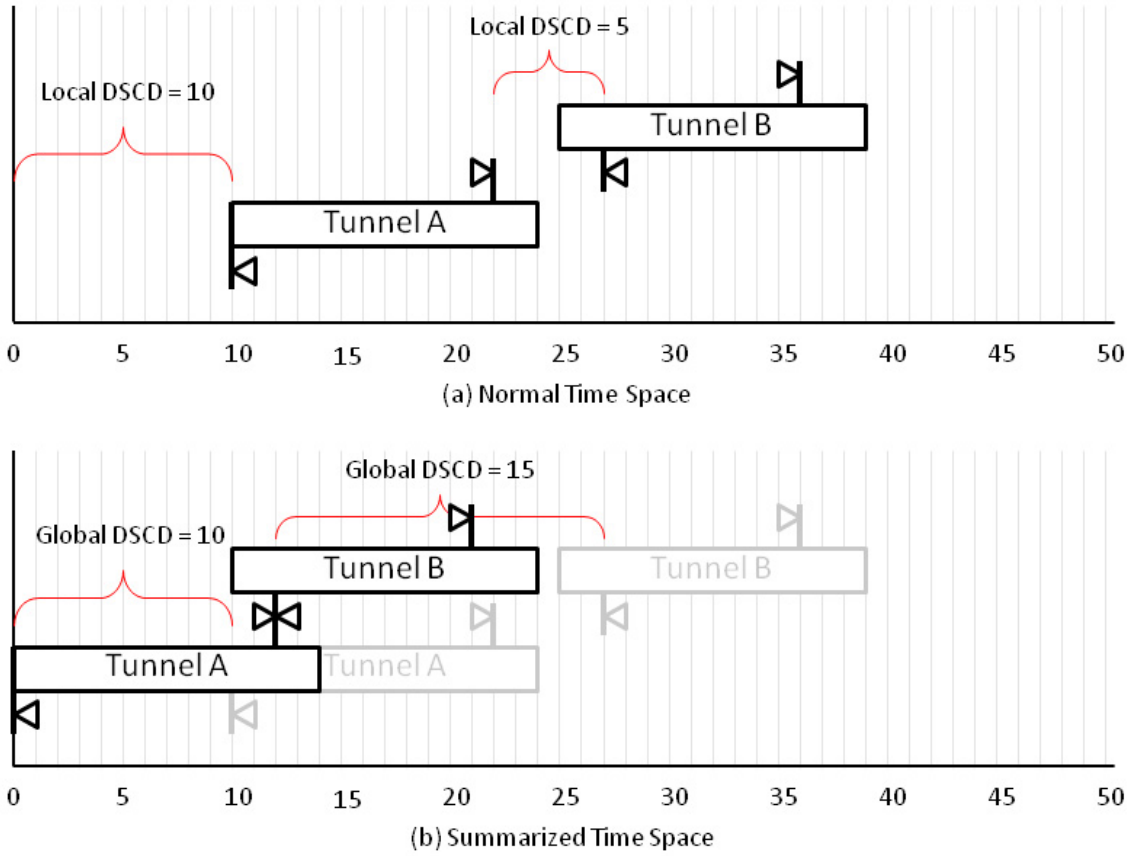



Figure 2.11: Time space shows tunnels with the collision pointers, where horizontal axis is frame number. (a) Local DSCDs for colliding with the previous tunnel (b) Global DSCDs for shifting to the summarized position.

# CHAPTER III

# PLAYER MODULE

This section proposes the playback process with result customization. We describe how a user can interact with the system and get back his request in real-time. With pre-processed parameters from recorder module, we can generate a summarization result instantly. Also, we explain about how buffering system helps our playback process gains higher performance for the whole framework and introduce a sample extension where our framework can be applied.

## 3.1 Playback with Film Map Generation and JIT Renderer

As mentioned earlier, packing tunnels together is the way to produce a summarized video, thus pre-computing of an offset distance yields an advantage to a film map generation.

Film map (see Fig. 1.2) is the overall layout of a summarized video, which describes a temporal position converted from global DSCD for each slice of the tunnels. This step causes the activities to appear simultaneously while they originally appear at different time. However, this step does not involve with a rendering process, so it can accomplish a summarized film map in a few milliseconds. Film map can be described as a complete movie film and just-in-time renderer can be described as a player. When a user wants to see a movie, he just inserts this film into the player, and then the player will project only the frames needed to be shown on the screen.

We implement a just-in-time renderer (JIT Renderer) for rendering only the necessary frames to be shown to a user. By using film map, JIT renderer (see Fig. 1.2) will render the video into the frame buffer according to the seek position, which gives us a real time output of the summarized video. We also implement a FIFO frame buffering system for keeping rendered frames. Whenever the buffer is not full, a system will use another thread to render the next frame into it, and a frame will be removed from the frame buffer after being played.

## 3.2 Searching by Direct Distance Transform

Distance transform for each tunnel is the distance between any point on an image to the points on a trajectory which we need to search for. We select trajectory information for being a source of making a distance. In order to find the trajectory similarity, each appropriate distance transform for each tunnel should be generated using any distance function. In this work we use

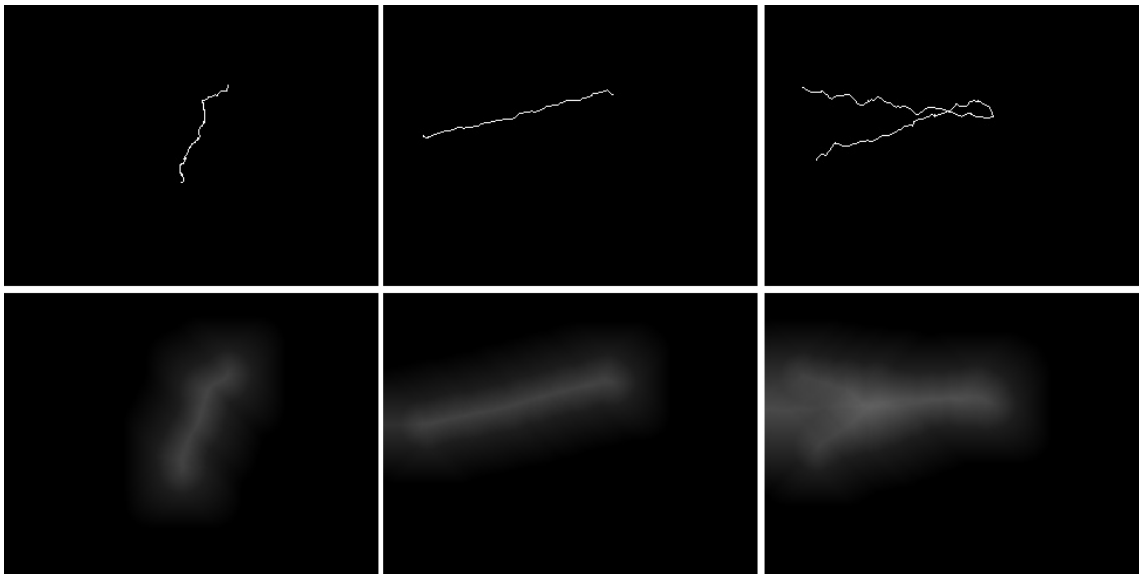


Figure 3.1: (top) Trajectories of tunnels. (bottom) Distance transforms.

Figure 3.2: (left)Moving object with trajectory. (right)The distance transform.

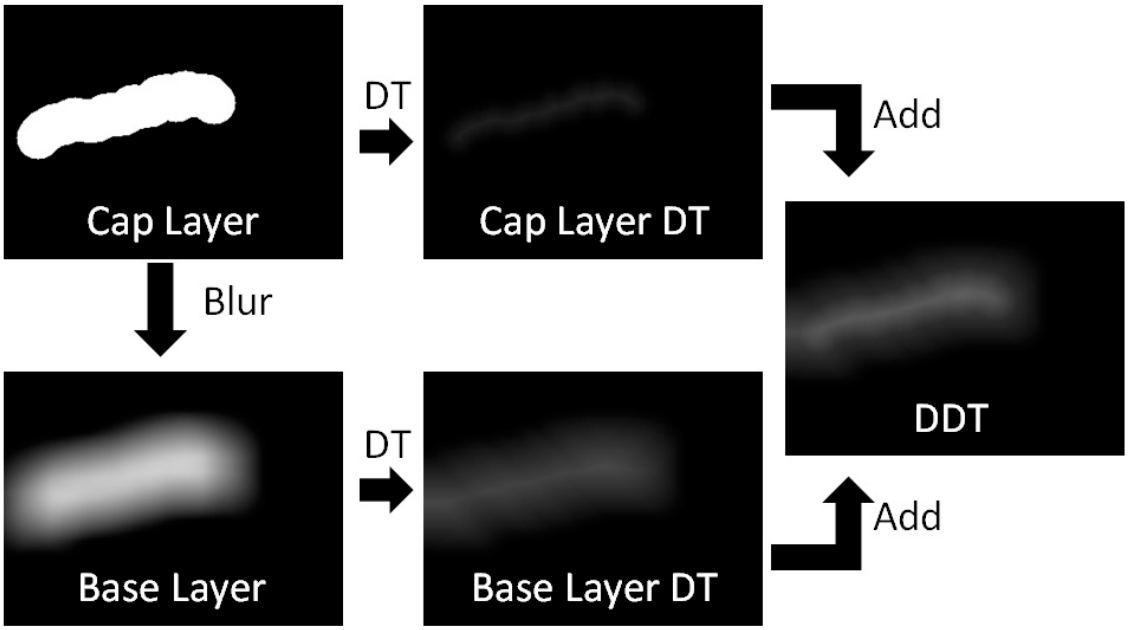



Figure 3.3: A process for making a double layer of a direct distance transform.

the Euclidian distance as our main distance function for calculating a distance as shown in Fig. 3.1. Two Phases Distance Transform is proposed for creating a distance transform.

### 3.2.1 Trajectory Expansion

At the time after a new tunnel is first analyzed, we plot the embedded information as a line represents the trajectory of such object tunnel. However, distance transform is the process for calculating a distance from outside to inside of an object, thus we need to expand a tiny line to make it larger enough for finding the distance as mentioned. A trajectory of a moving object shown in Fig. 3.2(left) needs to be processed to make the direct distance transform as shown in Fig. 3.2(right). To define a distance transform, let $DT(n)$ be a distance transform function using Euclidian distance and $T(n)$ be an analyzed tunnel where $n$ is a tunnel number, $K$ is the total number of tunnels, and $0 \leq n \leq K$. Trajectory is then mapped onto an image using $Traj(T(n))$. We also organize the distance transform into double layers which are

#### 3.2.1.1 Cap Layer

Let $Cap(n)$ be a distance transform at the core of an object trajectory. Particularly, this function just draws a bold trajectory over a gray image to represent a distance mapping as a function of intensity. Thus, the cap layer is for identifying the most similarity group of trajectories

to the user defined trajectory as shown in Fig. 3.3(top).

#### 3.2.1.2 Base Layer

Let $Base(n)$ be a distance transform at the base of an object trajectory. Its function is similar to the cap layer, its main purpose is for catching the rest of trajectories that are far away from the cap layer. Resulting in variant of the distance be spread out to most areas on the frame as wide as possible. We also apply a simple blur filter on the trajectory image to spread it out, and it is denoted as $Blur(Traj(T(n)))$.

Because a user defined trajectory may be out in the nearby area, it should be able to calculate the distance as shown in Fig. 3.3(bottom). Therefore, the equation can be described as follow

$$DT(n) = Cap(n) + Base(n) \tag{3.1}$$

$$Cap(n) = Draw(Traj(T(n))) \tag{3.2}$$

$$Base(n) = Draw(Blur(Traj(T(n)))) \tag{3.3}$$

### 3.2.2 Trajectory Similarity Measurement

In this phase, we apply the advantage of distance mapping as our DSCD algorithm to get the direct distance information. To define a distance function, let $DDT(RawTraj)$ be a direct distance transform for similarity measurement between trajectories to be searched within the range of summarizing tunnels, and $RawTraj$ be a trajectory model of a user specified trajectory which composes of a series of points, where $p$ is the number of point falling within the range, $L$ is the total number of points, and $0 \leq p \leq L$. We measure the distance by summing up directly from the intensity $Int$ of a distance transform at the position of any given points. The similarity measurement for each tunnel can be represented as follow

$$DDT(RawTraj) = \sum_{p=0}^{L} Int(RawTraj(p), Traj(T(n)) \tag{3.4}$$

The direct distance transform function can be used to sum up the similarity value for each tunnel. This process only takes time complexity of $O(n)$ to search one tunnel. We also normalize

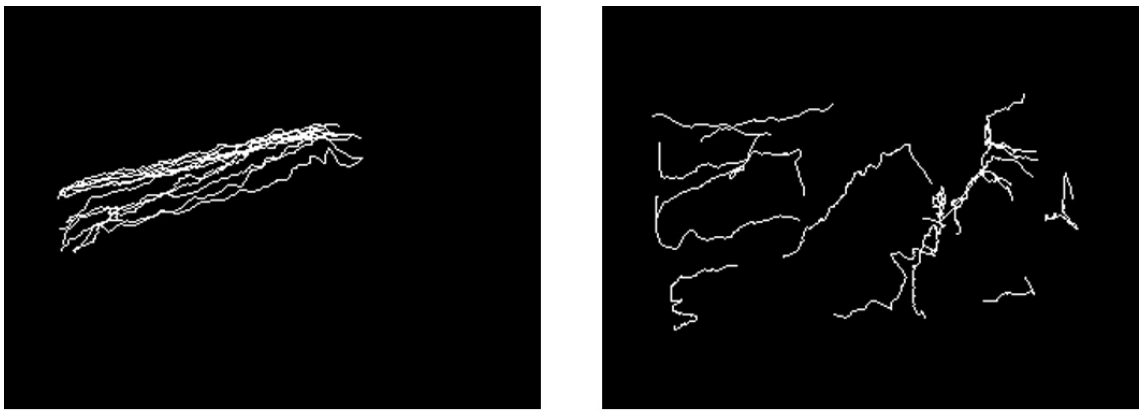



Figure 3.4: (a) A group of similar trajectories. (b) The rest are unmatched.

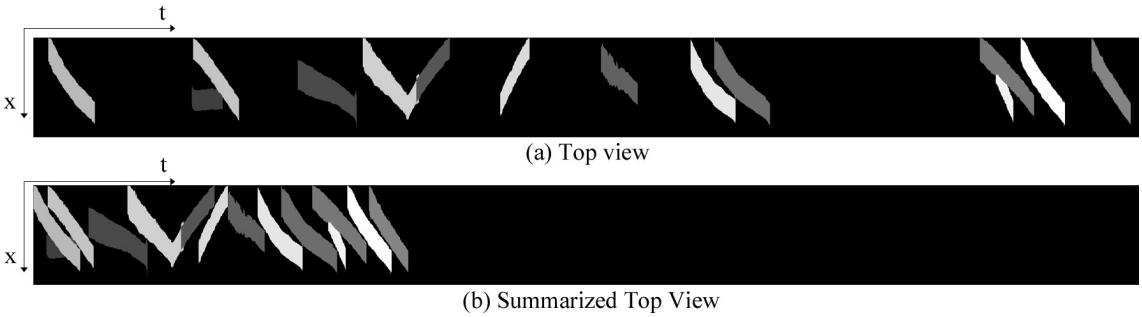
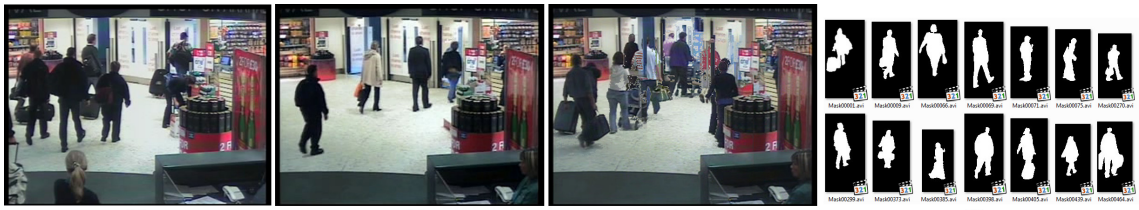



Figure 3.5: Time space shows multiple tunnels in time-space.

values of the trajectories similarity into 0-1 scale for checking the ratio of similarity, where 0 is low and 1 is high. Since we need to rank all tunnels for summarizing the relevant tunnels based on trajectory similarity as shown in Fig. 3.4, a sorting algorithm is applied at this step.

Normally, the DSCD value of each tunnel has been calculated which is computed with the nearby tunnels and the distances will be increased accumulatively from the first tunnel. The tunnels are now ready to be summarized. In this case, a tunnel will not be next to each others because some tunnels might not fall within the acceptable range of high similarity. Therefore, the DSCD of all filtered tunnels need to be recalculated such that it can be shifted to close to each other in time space



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

Figure 3.6: Sample frames from a summarized video. (a) DSCD: 2.0, Trajectory: not specified. (b) DSCD: 1.0, Trajectory: specified. (c) DSCD: 2.5, Trajectory: specified. (d) A collection of moving silhouette for each tunnel.

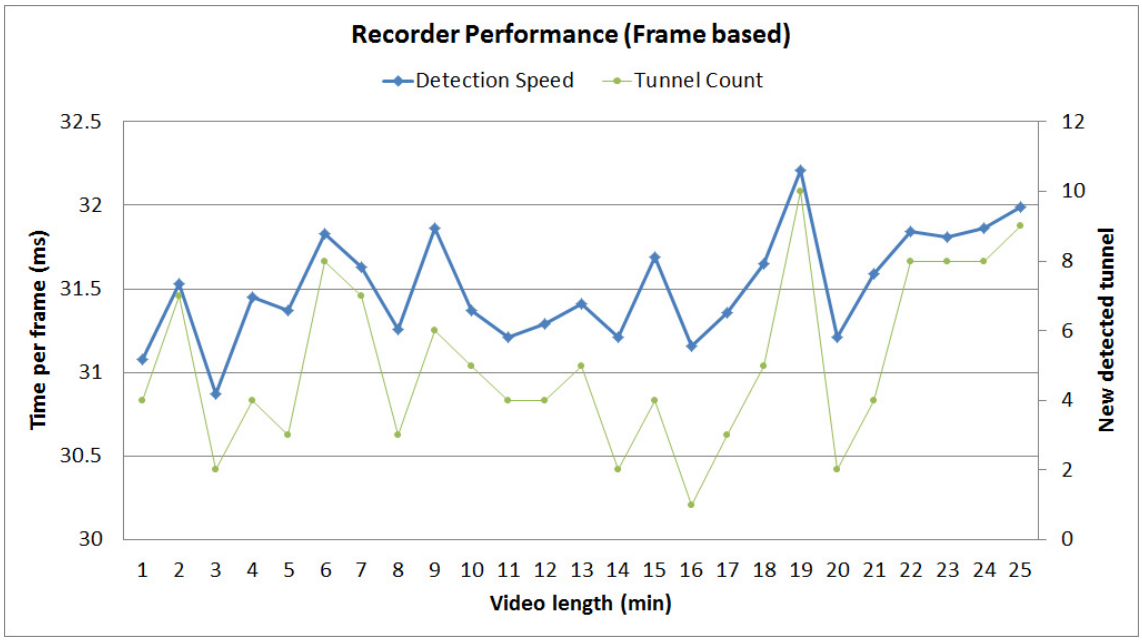



Figure 3.7: A performance evaluation of a recorder module. The top line shows a time usage per one frame for object detection and object tracking processes, which vary depend on the amount of new detected object as seen in the bottom line.

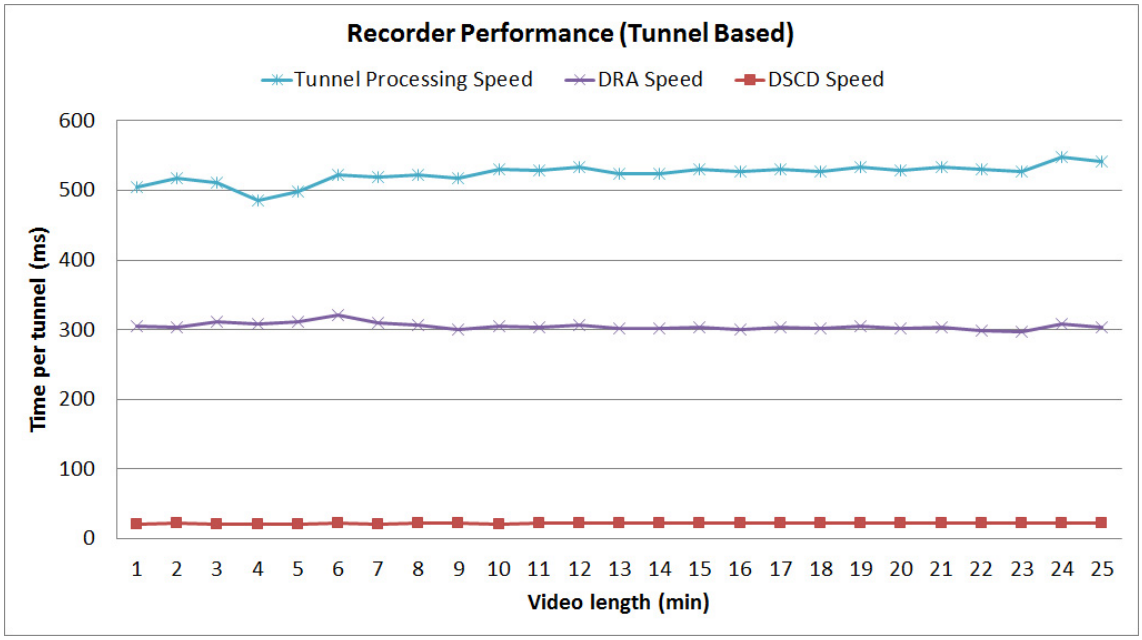



Figure 3.8: A performance evaluation of a recorder module shows the time usage to process with the different tasks for each tunnel. The asterisk line shows the time usage of tunnel processing step, which contains tunnel interpolating and tunnel cleansing. The cross line shows the time usage for extracting foreground out of each individual tunnel by using DRA. The square line shows the time usage for calculating the shortest distance for shifting together between tunnels.

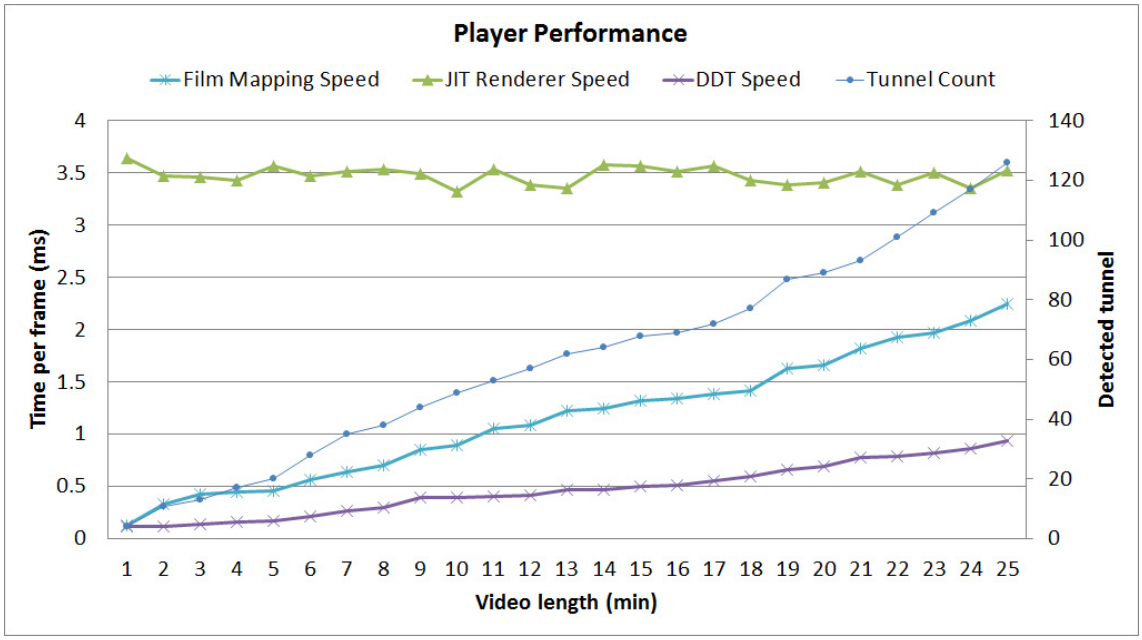



Figure 3.9: The graphs show the performance evaluation of player module comparing with the amount of detected tunnels for each video length. The asterisk line shows the speed of film mapping process to generate a draft of summarized video. The cross line shows a time usage for finding the trajectory similarity by using DDT. The speed of both film mapping and trajectory search is depended on the amount of collected tunnels. In contrast, the speed of JIT renderer is constant since it will process the frames when needed as shown in triangular line.

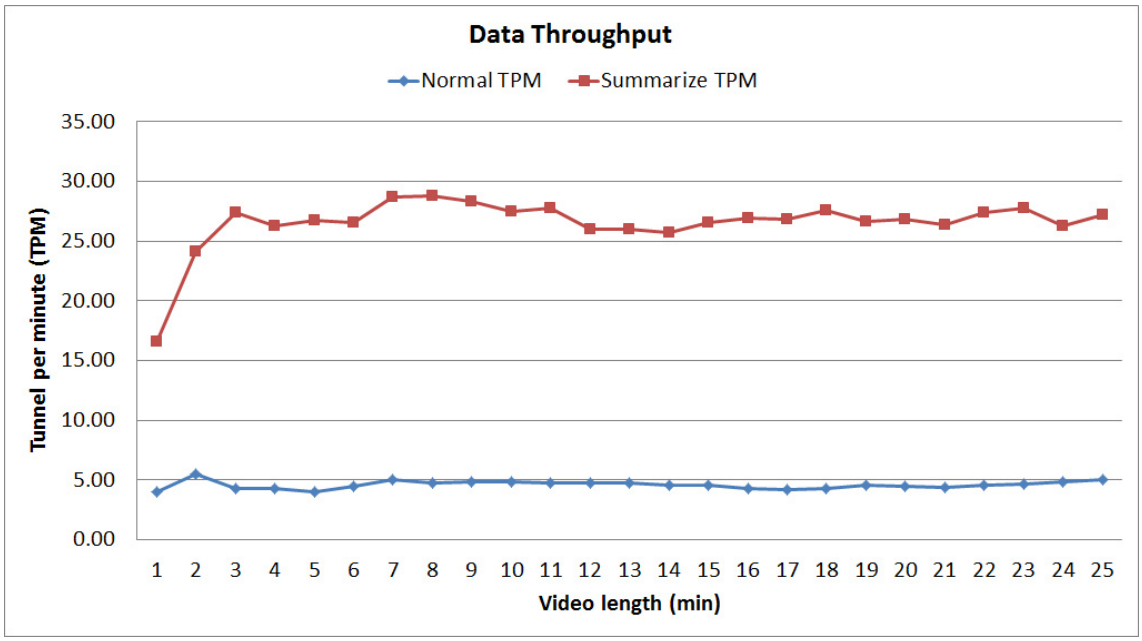



Figure 3.10: The data throughput graph shows the tunnel per minute rate (TPM) was improved after summarized with our framework.

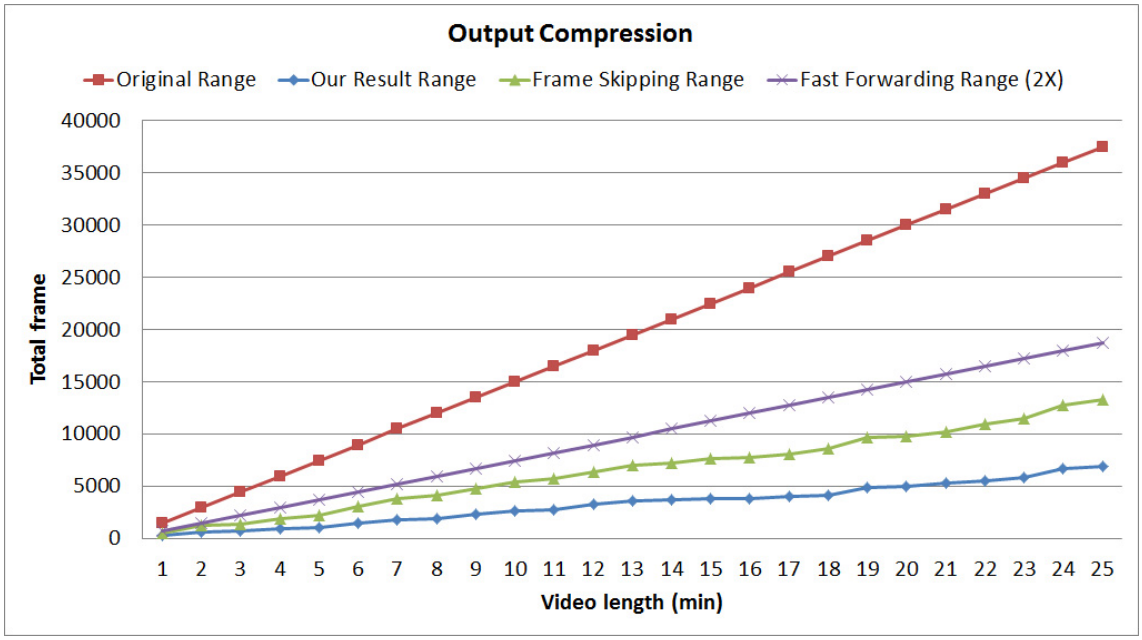



Figure 3.11: The length comparison of each summarized video by using different algorithms. The square line shows the total frame of original videos. The cross line shows the output length from a fast forwarding algorithm $2\times$ was cut by half. The triangular line shows the output length form frame skipping algorithm. And the dot line shows the shortest summarized results from our algorithms.

# CHAPTER IV

# EXPERIMENTAL RESULT

We propose a video surveillance summarization framework as shown in Fig. 1.2, which is organized into two major modules for real-time sophisticated browsing for rush exploration of video monitoring task. Our framework is built to support the video captured from practical surveillance camera. However, we also show the results that were processed from one of the most standard sources of video for retrieval evaluation, which is the airport scene from TRECVID London Gatwick surveillance video as show in Fig. 3.6.

The experiments were implemented on a normal laptop powered by Intel core 2 duo T8100 with the processor speed of 2.1 GHz. The whole framework was build with C# .NET programming language which is enough to give us a real-time performance of our framework. We prepared the TRECVID data named LGW_20071101_E1_CAM1.avi by splitting the original video into 25 footages of the first 1 minute of the video to the first 25 minutes of the video. Each video has the resolution of $320 \times 240$ pixels with 25 fps.

The recorder module needs to be performed as a video capturing, object detection, object tracking, tunnel processing, background subtraction, and recording an extracted object into an individual file. This mode can run with the real-time performance of about 32 milliseconds per frame (see Fig. 3.7) or about the speed of more than 30 frames per second, which is enough for a typical surveillance camera that mostly capture the video with the frame rate of about 15 frames per second for generally web camera, 25 frames per second for PAL system camera, and 30 frames per second for NTSC system camera under an indoor lighting condition.

The player module can be run parallel or on a separate process or even on another computer by accessing the tunnels information through a network. This module runs in real-time by rendering only necessary frames as described in section 3.1.

The experimental results are summarized in Fig. 3.7 - 3.11. The graph in Fig. 3.7 shows the overall frame based performance of the recorder module comparing with the amount of new detected tunnels. We detect the human for every frames by using HOG (Dalal and Triggs, 2005), which gives us the result of region of interest for each moving human. Then, the tracking process will handle a task of connecting pieces of each tunnel together. The combination of detection and

tracking steps, it take time approximately 32 milliseconds per frame, where the speed of detection depends on the number of occurring tunnels.

The graph in Fig. 3.8 shows the overall tunnel based performance of the recorder module. The process of generating tunnel will occur once only after the whole tunnel was collected. This step includes a tunnel processing, a DRA for background subtraction, and a DSCD for objects collision detection in time-space. A tunnel processing step, which processes a tunnel interpolating and a tunnel cleansing, takes time about 530 milliseconds per tunnel including I/O as shown in an asterisk line. DRA for background subtraction, which processes all the hard-working tasks in section 2.2, takes time about 300 milliseconds per tunnel as shown in a cross line. And the core process of our summarization is DSCD, which will detect the collision distance between tunnel, takes time about 20 milliseconds per tunnel as shown in a square line.

For the player module, we summarized the overall performance evaluation into the graph in Fig. 3.9, which shows the overall speed comparing with the total amount of detected tunnels. The asterisk line shows the speed of making a draft of video summarization result in a film mapping process, which increases from 0.1 milliseconds to 2.25 milliseconds due to the number of detected tunnels. Also, the DDT process was affected by the number of collected tunnels as seen in the cross line from 0.1 to 1 millisecond, since it needs to find the best trajectory similarity from all tunnels. However, JIT renderer can render the result constantly about 3.5 millisecond per frame as show in a triangular line.

The result in Fig. 3.10 is to show the amount of tunnel is increased in a summarization video. The dot line shows the data throughput of the original video, which has the rate of tunnel per minute about 5 tpm. Then, the square line shows the tunnel rate in the summarized video was increased to about 25 tpm, which means we can see the moving object passed the screen approximately 5 times frequent than the original video.

For the data compression in Fig. 3.11, we shows a comparison on the video length after summarized with different algorithms. The square line shows the length of the original video from 1 minute or 1,500 frames to 25 minutes or 40,500 frames. Fast forwarding algorithm Petrovic et al. (2005) is an approach which play the video by skipping consecutive frames equally. For example, $2\times$ fast forwarding means after playing one frame, the next frame will not be played but play next two frame instead. At this time, we compare with $2\times$ fast forwarding then the cross line shows the result of the length of s summarized video was cut by half. Frame skipping algorithm Smith

Table 4.1: The summarization results of each video lengths.

| Original Length (in minute) | Tunnel Count | Normal TPM | Summarize Length (in minute) | Summarize TPM | Compression Ratio | Time Saved (in percentage) |
|---|---|---|---|---|---|---|
| 1 | 4 | 4.00 | 0.24 | 16.57 | 4.14:1 | 75.87 |
| 2 | 11 | 5.50 | 0.46 | 24.09 | 4.38:1 | 77.17 |
| 3 | 13 | 4.33 | 0.48 | 27.35 | 6.31:1 | 84.16 |
| 4 | 17 | 4.25 | 0.65 | 26.23 | 6.17:1 | 83.80 |
| 5 | 20 | 4.00 | 0.75 | 26.71 | 6.68:1 | 85.03 |
| 6 | 27 | 4.50 | 1.02 | 26.57 | 5.91:1 | 83.07 |
| 7 | 35 | 5.00 | 1.22 | 28.72 | 5.74:1 | 82.59 |
| 8 | 38 | 4.75 | 1.32 | 28.76 | 6.05:1 | 83.48 |
| 9 | 44 | 4.89 | 1.56 | 28.28 | 5.78:1 | 82.71 |
| 10 | 49 | 4.90 | 1.79 | 27.44 | 5.60:1 | 82.14 |
| 11 | 52 | 4.73 | 1.87 | 27.79 | 5.88:1 | 82.99 |
| 12 | 57 | 4.75 | 2.19 | 26.00 | 5.47:1 | 81.73 |
| 13 | 62 | 4.77 | 2.39 | 25.98 | 5.45:1 | 81.64 |
| 14 | 64 | 4.57 | 2.49 | 25.75 | 5.63:1 | 82.25 |
| 15 | 68 | 4.53 | 2.56 | 26.51 | 5.85:1 | 82.90 |
| 16 | 69 | 4.31 | 2.56 | 26.90 | 6.24:1 | 83.97 |
| 17 | 72 | 4.24 | 2.68 | 26.83 | 6.33:1 | 84.21 |
| 18 | 77 | 4.28 | 2.80 | 27.53 | 6.43:1 | 84.46 |
| 19 | 87 | 4.58 | 3.27 | 26.59 | 5.81:1 | 82.78 |
| 20 | 89 | 4.45 | 3.32 | 26.80 | 6.02:1 | 83.39 |
| 21 | 93 | 4.43 | 3.53 | 26.38 | 5.96:1 | 83.21 |
| 22 | 101 | 4.59 | 3.69 | 27.37 | 5.96:1 | 83.22 |
| 23 | 108 | 4.70 | 3.89 | 27.73 | 5.91:1 | 83.07 |
| 24 | 117 | 4.88 | 4.46 | 26.23 | 5.38:1 | 81.41 |
| 25 | 125 | 5.00 | 4.60 | 27.18 | 5.44:1 | 81.61 |

(1997) is an approach to play the video by playing only the frames contain the contents such as moving object, then skipping other frames. The triangular line shows the summarized result of frame skipping which are shorter than video fast forwarding.

Our result in Fig. 3.11, the dot line shows the shortest summarized output for all videos[1]. Table 4.1 shows the compression ratio between original video with our summarized result, which are around 4:1 to 6:1 and it can save time for reviewing the surveillance video to about 85%. And also, in table 4.2 shows the summarization results based on the calculation for new detected tunnel. The overall results are the conclusion that the length of each summarization result is depended on the amount of new detected such as the lower number of object for each period of time yields the shorter of summarized result. And at the $16^{th}$ minutes, the compression ratio of Max means the algorithm can compress the video in this period of time into zero minute, since there occurs a time-space left inside the $15^{th}$ minutes that is enough for a tunnel of $16^{th}$ minutes to be shifted into. Our approach can summarize the surveillance video into the shorter duration without remove any activity objects as we expected. Not only a natural speed of moving objects were produced from our framework, but also it can increase the number of moving object within the same period of time.

[1]We encourage readers to view more video examples in ftp://tppwan1.dyndns.org/Video%20Summarization

Table 4.2: The summarization results for new detected objects of each video lengths.

| Original Length (in minute) | New Tunnel | Normal TPM | Summarize Length (in minute) | Summarize TPM | Compression Ratio | Time Saved (in percentage) |
|---|---|---|---|---|---|---|
| 1 | 4 | 4.00 | 0.24 | 16.57 | 4.14:1 | 75.87 |
| 2 | 7 | 5.50 | 0.22 | 24.09 | 4.64:1 | 78.47 |
| 3 | 2 | 4.33 | 0.02 | 27.35 | 53.57:1 | 98.13 |
| 4 | 4 | 4.25 | 0.17 | 26.23 | 5.79:1 | 82.73 |
| 5 | 3 | 4.00 | 0.10 | 26.71 | 9.93:1 | 89.93 |
| 6 | 8 | 4.50 | 0.27 | 26.57 | 3.74:1 | 73.27 |
| 7 | 7 | 5.00 | 0.20 | 28.72 | 4.93:1 | 79.73 |
| 8 | 3 | 4.75 | 0.10 | 28.76 | 9.74:1 | 89.73 |
| 9 | 6 | 4.89 | 0.23 | 28.28 | 4.26:1 | 76.53 |
| 10 | 5 | 4.90 | 0.23 | 27.44 | 4.35:1 | 77.00 |
| 11 | 4 | 4.73 | 0.09 | 27.79 | 11.72:1 | 91.47 |
| 12 | 4 | 4.75 | 0.32 | 26.00 | 3.12:1 | 67.93 |
| 13 | 5 | 4.77 | 0.19 | 25.98 | 5.14:1 | 80.53 |
| 14 | 2 | 4.57 | 0.10 | 25.75 | 10.14:1 | 90.13 |
| 15 | 4 | 4.53 | 0.08 | 26.51 | 12.61:1 | 92.07 |
| 16 | 1 | 4.31 | 0.00 | 26.90 | Max | 100.00 |
| 17 | 3 | 4.24 | 0.12 | 26.83 | 8.38:1 | 88.07 |
| 18 | 5 | 4.28 | 0.11 | 27.53 | 8.82:1 | 88.67 |
| 19 | 10 | 4.58 | 0.47 | 26.59 | 2.11:1 | 52.60 |
| 20 | 2 | 4.45 | 0.05 | 26.80 | 20.00:1 | 95.00 |
| 21 | 4 | 4.43 | 0.20 | 26.38 | 4.89:1 | 79.53 |
| 22 | 8 | 4.59 | 0.16 | 27.37 | 6.07:1 | 83.53 |
| 23 | 8 | 4.70 | 0.20 | 27.73 | 4.92:1 | 79.67 |
| 24 | 8 | 4.88 | 0.57 | 26.23 | 1.76:1 | 43.33 |
| 25 | 9 | 5.00 | 0.14 | 27.18 | 7.25:1 | 86.20 |

# CHAPTER V

# CONCLUSION

We have proposed a novel real-time video surveillance summarization framework intended for minimizing time usage for rushes exploration of video monitoring task based on shifting of moving object in time-space. Our framework can produce summarization results without remove any significant contents, while still keeping the real-time performance. The main advantage and novelty of our proposed framework is the combination of fast and robust algorithms combined with an efficient implementation for balancing the workflows. We also introduced DSCD for fastest collision detection, film map generation for making a draft summarization, JIT renderer for rendering only necessary frame, DDT for similarity measurement, and DRA for contour based background subtraction.

Our presented approach is based on a tunneling system, which can help increasing human ability and efficiency of browsing and analyzing of a surveillance video by reducing time usage for reviewing the video. The earlier application for our tunneling system is to search the best match for object trajectory, which is one of an identifier for interesting object. The combination of these algorithms make our framework to be not only have less computational cost, but also responding the results by real-time performance.

In this framework, we use HOG with a default human descriptor to detect human. For some cases of crowded people occurred in the scene, the detection will not be able to detect any human who is behind the others. So the framework will not be able to include all those human into the system for making a summarized result.

The presented framework also provides a lot information described each activity tunnel such as timing, trajectory, contour, etc. So the future release can apply all those information to help identifying and understanding on each moving object or the overall behaviour and also, this framework can be applied with other objects such as car, bicycle, etc. Also, trajectory searching can be processed with other properties such as direction, time, speed, and size of object to help identify more exact to the interesting object. Another example is the summarized result can be filtered and show only unusual tunnels, then the analyzing process will be more easy for the human to judge yielded more speed and gained up accuracy of the real-time security monitoring task.

# References


Shai Avidan and Ariel Shamir. 2007. Seam carving for content-aware image resizing. ACM Transactions on Graphics 26.

Donald J. Berndt and James Clifford. 1996. Advances in knowledge discovery and data mining. chapter Finding patterns in time series: a dynamic programming approach, pp. 229–248. American Association for Artificial Intelligence.

Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, SIGMOD '05, pp. 491–502. ACM.

Nagul Cooharojananone, Siriwat Kasamwattanarote, Shin'ichi Satoh, and Rajalida Lipikorn. 2010. Real time trajectory search in video summarization using direct distance transform. In IEEE 10th International Conference on Signal Processing, ICSP '10, pp. 932–935. IEEE Computer Society.

Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1 of CVPR '05, pp. 886–893. IEEE Computer Society.

Ahmet M. Ferman and A. Murat Tekalp. 1997. Multiscale content extraction and representation for video indexing. In Proceedings of Multimedia Storage and Archiving Systems, volume 2, pp. 23–31. SPIE.

Hong-Wen Kang, Xue-Quan Chen, Yasuyuki Matsushita, and Xiaoou Tang. 2006. Space-time video montage. In Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 2 of CVPR '06, pp. 1331–1338. IEEE Computer Society.

Siriwat Kasamwattanarote, Nagul Cooharojananone, Shin'ichi Satoh, and Rajalida Lipikorn. 2010. Real time tunnel based video summarization using direct shift collision detection. In Proceedings of the 11th Pacific Rim conference on Advances in multimedia information processing: Part I, PCM '10, pp. 136–147. Springer-Verlag.

Changick Kim and Jenq-Neng Hwang. 2000. An integrated scheme for object-based video abstraction. In Proceedings of the eighth ACM international conference on Multimedia, MULTIMEDIA '00, pp. 303–311. ACM.

Zhuang Li, Prakash Ishwar, and Janusz Konrad. 2009. Video condensation by ribbon carving. IEEE Transactions on Image Processing 18:2572–2583.

Kadir A. Peker and Ajay Divakaran. 2004. Adaptive fast playback-based video skimming using a compressed-domain visual complexity measure. In International Conference on Multimedia and Expo, volume 3 of ICME '04, pp. 2055–2058. IEEE Computer Society.

Nikos Pelekis, Ioannis Kopanakis, Gerasimos Marketos, Irene Ntoutsi, Gennady Andrienko, and Yannis Theodoridis. 2007. Similarity search in trajectory databases. In Proceedings of the 14th International Symposium on Temporal Representation and Reasoning, pp. 129–140. IEEE Computer Society.

Nemanja Petrovic, Nebojsa Jojic, and Thomas S. Huang. 2005. Adaptive video fast forward. Multimedia Tools and Applications 26:327–344.

Yael Pritch, Alex Rav-Acha, Avital Gutman, and Shmuel Peleg. 2007. Webcam synopsis: Peeking around the world. In IEEE 11th International Conference on Computer Vision, ICCV '07, pp. 1–8. IEEE Computer Society.

Yael Pritch, Alex Rav-Acha, and Shmuel Peleg. 2008. Nonchronological video synopsis and indexing. IEEE Transactions on Pattern Analysis and Machine Intelligence 30:1971–1984.

Yael Pritch, Sarit Ratovitch, Avishai Hendel, and Shmuel Peleg. 2009. Clustered synopsis of surveillance video. In Proceedings of the 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS '09, pp. 195–200. IEEE Computer Society.

Alex Rav-Acha, Yael Pritch, and Shmuel Peleg. 2006. Making a long video short: Dynamic video synopsis. In Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pp. 435–441. IEEE Computer Society.

Michael A. Smith. 1997. Video skimming and characterization through the combination of image and language understanding techniques. In Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition, CVPR '97, pp. 775–781. IEEE Computer Society.

Marc van Kreveld and Jun Luo. 2007. Trajectory similarity of moving objects. In Young Researcher Forum, pp. 229–232.

Michail Vlachos, Dimitrios Gunopoulos, and George Kollios. 2002a. Discovering similar multi-dimensional trajectories. In Proceedings of the 18th International Conference on Data Engineering, ICDE ’02, pp. 673–. IEEE Computer Society.

Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002b. Robust similarity measures for mobile object trajectories. In Proceedings of the 13th International Workshop on Database and Expert Systems Applications, DEXA ’02, pp. 721–728. IEEE Computer Society.

# APPENDICES

# APPENDIX A

# IMPLEMENTATION

In this chapter, we provide an implementation of depth mapping process for embedding distance information of each tunnel. Normally, depth map image is an image illustrating the depth of objects aligned in the z-axis of three dimensional Cartesian coordinate. The video also be able to explain with 3D coordinate by using time-axis instead of z-axis. The time-axis of a video is usually counted as frame number, thus our detected object also creates a tunnel depth in time dimension. That means an endless video from surveillance camera will create endless amount of detected tunnels which assigned by each specific frame number in time dimension.

## A.1 Depth Mapping

Depth map can be represented by using grayscale image, which contains an information of intensity from minimum to maximum of 0 to 255 by using one byte or 8 bits per pixel. Storing a tunnel depth information into an image, we need a space more than 256 levels to describe a time dimension for the whole video. Therefore, separate keeping an absolute starting frame then embedding only a relative time-depth information of an individual tunnel is the best solution for taking a use of depth map image as seen in Fig. A.1. However, without modification of a grayscale image, we can store only 256 slices per tunnel, or only about 10 seconds per tunnel for a video frame rate at 25 frames per second. To increase an amount of tunnel depth, we store the data on all effective pixels by doing a trick on a data structure of image.

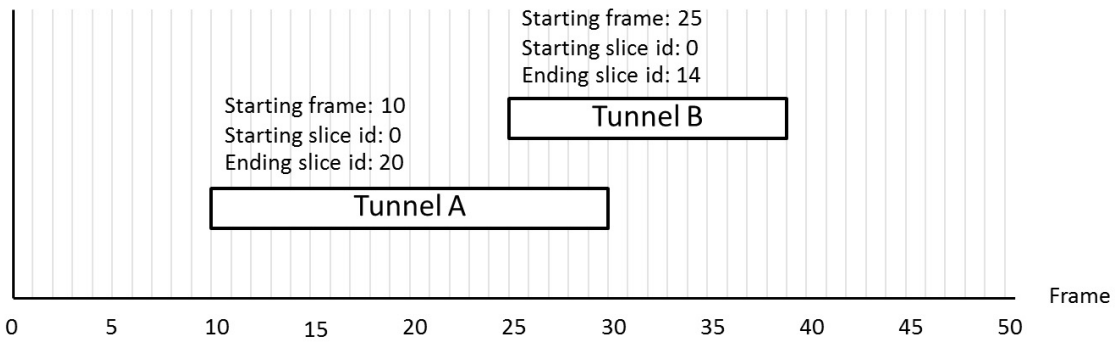



Figure A.1: This image illustrates the slice id is counted relatively from 0 to the end of tunnels for be able to store in the gray based depth map image. And storing absolute starting frame separately.

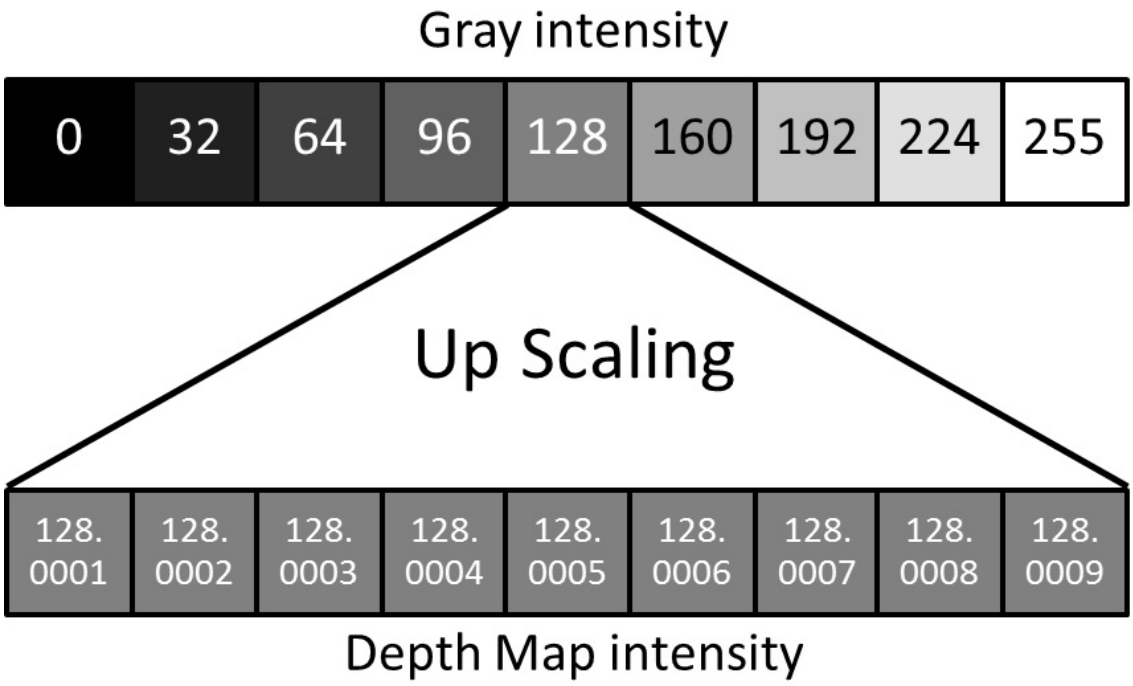



Figure A.2: This image shows the intensity value for each gray level comparing to the depth map value after up scaling from a standard gray intensity.

Grayscale image normally has a depth of byte or 8 bits depth. To store more data, we simply need more bit depth. Although we can use other data type such as int16(16-bit or $2^{16}$ values), int32(32-bit or $2^{32}$ values), and double(64-bit or $2^{64}$ values), the OpenCV will not allow implementing of gray value more than 256. Thus, we still be able to implement a depth mapping image on a data type of float contained gray values from 0 to 255, which yield a capability of storing a decimal intensity as seen in Fig. A.2.

In order to store depth information inside an image pixel, we use the fraction part of gray intensity to keep a slice number of tunnel, which is relative to the starting frame number of tunnel. Hence, the integer part we use it as a flag for checking the occurrence pixels of tunnel as seen in Fig A.3. And the value for each slice of tunnel was assigned by the following

$$Int(S(i)) = Flag + SliceID_i/Precision \tag{A.1}$$

where $Int$ is a depth function for returning the intensity of the corresponding slice $i$, $Flag$ is a number between 0 to 255 to be use as a flagged value in the integer part, $SliceID$ is the slice

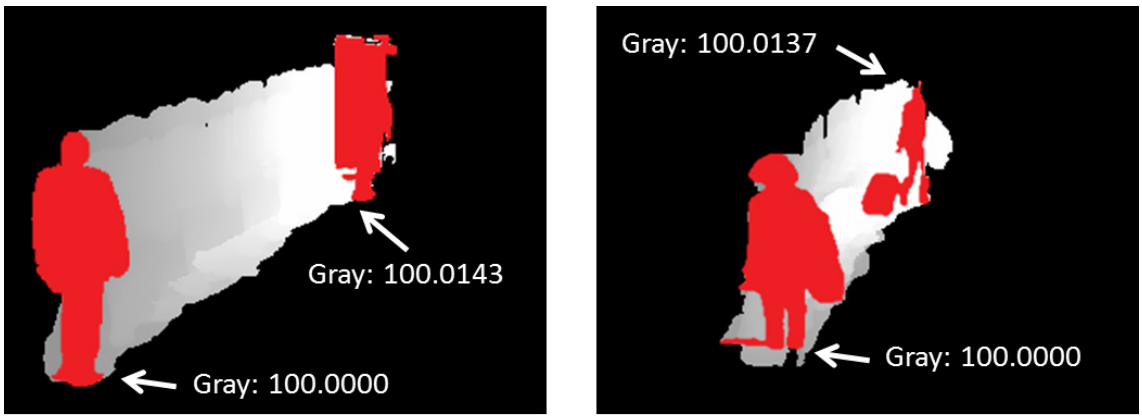



Figure A.3: Back view tunnels in time-space show the gray value was assigned differently in the decimal part. As you can see, we set a flag value of integer part to be 100, and the decimal part is depends on the slice number inside a tunnel.

number on a tunnel, and $Precision$ is the maximum number of slice to be handled. For example, in Fig. A.3a shows a depth map image of tunnel, which contains 144 slices. The first slice has a gray intensity of 100.0000 for the first index, and the last slice has a gray intensity of 100.0143 for the $144^{th}$ index. In the same way, in Fig. A.3b shows a depth map image of tunnel contained 138 slices. The first slice has a gray intensity of 100.0000 for the first index, and the last slice has a gray intensity of 100.0137 for the $138^{th}$ index.

## A.2 Checking Phase in Action

The major process of DSCD is to calculate the shortest collision distance. In this section, we will show you how to calculate the collision distance between tunnels.

In figure A.4, we show the depth map image between two consecutive tunnels that was drawn in the different order. For each checking phase, supposing we are in the middle between with two tunnels $T(n-1)$ and $T(n)$. When we look forward to $T(n-1)$, we should see a back-side of tunnel in time space as seen in Fig. A.4a1 and a2. In contrast, to look backward to $T(n)$ we will normally see a front-side of tunnel as seen in Fig. A.4b1 and b2. So, we or the collision detector located in between two tunnels will know the relative distance from the depth maps of a back-side of $T(n-1)$ and a front-side of $T(n)$. Therefore, calculating the shortest collision distance between two tunnels is to find the minimum distance between a back-side of $T(n-1)$ and a front-side of $T(n)$, or we simply say that this process is just subtracting the depth map of $T(n-1)$ and $T(n)$ then find the position of the minimum value which is occurred at the flagged

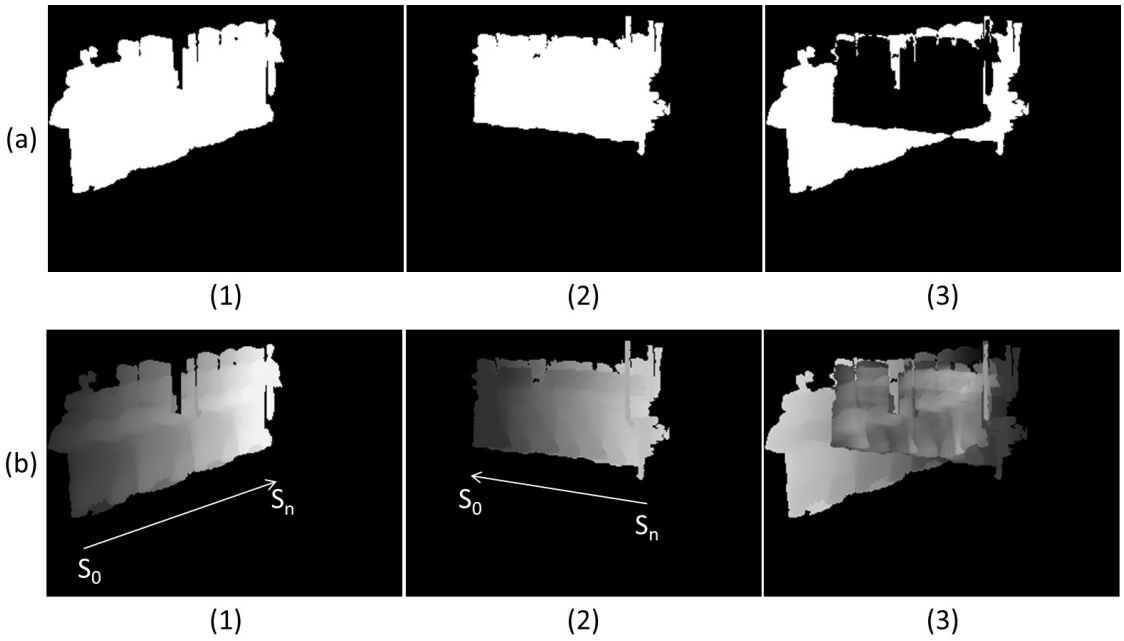



Figure A.4: This figure shows depth maps while using in a process of DSCD. Row (a), the normal depth map. Row (b), the normalized depth map for showing the depth to our vision. Column (1), the depth map looking from a back view of $T(n-1)$. Column (2), the depth map looking from front view of $T(n)$. Column (3), The absolute different result of two depth maps. The arrows in (b1) and (b2) show the different slice order for drawing a depth map of back view tunnel (b1) and front view tunnel (b2)

position as seen in Fig. A.4a3 and b3.

# APPENDIX B

# LIST OF PUBLICATIONS

Parts of this work are published in the following articles.

**International Journals**

1. Siriwat Kasamwattanarote, Nagul Cooharojananone, Shin'ichi Satoh, and Rajalida Lipikorn, "Automated real-time video surveillance summarization framework", Journal of Real-Time Image Processing, submitted. Springer (2012)

**International Conference Proceedings**

1. Siriwat Kasamwattanarote, Nagul Cooharojananone, Shinichi Satoh, and Rajalida Lipikorn, "Real Time Tunnel Based Video Summarization using Direct Shift Collision Detection", 11th Pacific Rim Conference on Multimedia, Shanghai, China, pp. 136-147, Sep. 21-24, 2010.
2. Nagul Cooharojananone, Siriwat Kasamwattanarote, Shinichi Satoh, and Rajalida Lipikorn, "Real Time Trajectory Search in Video Summarization using Direct Distance Transform", 10th International Conference on Signal Processing, Beijing, China, vol. 6297/2010, pp. 932- 935, Oct. 24-28, 2010.

# Biography

Siriwat Kasamwattanarote was born in Bangkok, Thailand, on $8^{th}$ November, 1986. He received his B.Sc. in Information and Communication Technology from Mahidol University in 2009. Currently, he is pursuing his master degree in image and video processing on surveillance camera at Computer Science and Information Technology Program, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. He had worked in IWANE laboratories for computer vision and image processing in 2008. He has also received two scholarships for short-term researches on time-series agricultural images and surveillance video from National Institute of Informatics, Tokyo, Japan, in year 2010 and 2011, respectively. His research interests include real-time multi-dimensional image and video processing, computer vision, code optimization, and computer networking.