http://dx.doi.org/10.3745/KIPSTB.2011.18B.4.173

연속 영상 기반 실시간 객체 분할

강 의 선† · 유 승 훈††

요 약

본 논문은 GPU(Graphics Processing Unit) 에서 CUDA(Compute Unified Device Architecture)를 사용하여 실시간으로 객체를 분할하는 방법을 소개한다. 최근에 감시 시스템, 오브젝트 추적, 모션 분석 등의 많은 응용 프로그램들은 실시간 처리가 요구된다. 이러한 단계의 선행부분인 객체 분할 기법은 기존 CPU 기반의 시스템으로는 실시간 처리에 제약이 발생한다. NVIDIA에서는 Parallel Processing for General Computation 을 위해 그래픽 하드웨어 제약을 개선한 CUDA platform을 제공하고 있다. 본 논문에서는 객체 추출 단계에 대표적인 적응적 가우시안 혼합 배경 모델링(Adaptive Gaussian Mixture Background Modeling) 알고리즘과 Classification 기법으로 사용되는 CCL (Connected Component Labeling) 알고리즘을 적용하였다. 본 논문은 2.4GHz를 갖는 Core2 Quad 프로세서와 비교하여 평가하였고 그 결과 3~4배 이상의 성능향상을 확인할 수 있었다.

키워드 : GPU(Graphics Processing Unit), CUDA(Compute Unified Device Architecture), 객체 분할

Real-Time Object Segmentation in Image Sequences

Eui-Seon Kang† · Seung-Hun Yoo††

ABSTRACT

This paper shows an approach for real-time object segmentation on GPU (Graphics Processing Unit) using CUDA (Compute Unified Device Architecture). Recently, many applications that is monitoring system, motion analysis, object tracking or etc require real-time processing. It is not suitable for object segmentation to procedure real-time in CPU. NVIDIA provide CUDA platform for Parallel Processing for General Computation to upgrade limit of Hardware Graphic. In this paper, we use adaptive Gaussian Mixture Background Modeling in the step of object extraction and CCL(Connected Component Labeling) for classification. The speed of GPU and CPU is compared and evaluated with implementation in Core2 Quad processor with 2.4GHz.The GPU version achieved a speedup of 3x-4x over the CPU version.

Keywords : GPU(Graphics Processing Unit), CUDA(Compute Unified Device Architecture), Object Segmentation

1. Introduction

Real-time segmentation of moving regions in image sequences is a fundamental step in many vision systems including automated visual surveillance, human-machine interface, and very low-bandwidth telecommunications [1]. A typical method is background subtraction. Background subtraction involves calculating a reference image, subtracting each new frame from this image and thresholding the result. This method suffers from many problems and requires a training period absent of foreground objects. The motion of background objects after the training period and foreground objects motion less during the training period would be considered as permanent foreground objects. In addition, the approach cannot cope with gradual illumination changes in the scene. These problems lead to the requirement that any solution must constantly re-estimate the background model [1].

Many adaptive background-model methods have been proposed to deal with these slowly-changing stationary signals. One of the successful solutions to these problems is to use a multi-color background model per pixel proposed by Grimson et al. [2-3]. An adaptive Gaussian mixture model can deal with lighting changes, slow-moving objects, and introducing and removing object from the scene. Dynamically, this system updates

† 정 회 원 : 숭실대학교 베어드학부 조교수
†† 정 회 원 : 삼성전자 책임연구원(교신저자)
논문접수 : 2010년 12월 2일
수 정 일 : 1차 2011년 2월 15일, 2차 2011년 4월 9일
심사완료 : 2011년 4월 20일

background models and performs the multiple Gaussian distribution more efficiently. Also, the Gaussian mixture modeling was recently used the many other places [1, 4, 5]. For the better foreground segmentation, color of YUV channel and depth that is obtained by the pair images are used to compose the Gaussian mixture model at each pixel in the [4]. Although this algorithm shows the efficient result, its computational cost is very expensive.

After the object detection, the detected objects must be classified. It usually used a connected components labeling (CCL) algorithm. In papers by Suzuki, Wu et al. [6-8], approaches and algorithms for CCL are categorized into a number of groups. Two pass algorithms show very high performance, but require large memory to store label equivalence. Multi-pass algorithms scan an image in the forward and backward raster directions alternately to propagate label equivalences until no label changes. The performance of these algorithms is very dependent on complexity of connected component and speed of resolving the label equivalences at analysis phase. In addition, many of them are focusing on sequential approaches and their optimizations for ordinary computer architecture.

GPU on consumer-level graphic cards has evolved into a very powerful and flexible streaming processor, which includes fully programmable floating-point pipelines giving good computational power and memory bandwidth [9]. The power and flexibility of GPUs provide an attractive platform for computationally demanding tasks with respect to specific graphics and general-purpose computations (which is also the target of general-purpose GPU, called GPGPU [10].). In most cases, GPU-based implementations are much faster than comparable implementations on CPU in the fields of image processing, linear algebra, data sorting, computational physics, and database queries [9]. Recently, NVIDIA released its latest GPU model, CUDA (Compute Unified Device Architecture) that provides an extended version of ANSI-C for general-purpose applications based on GPU [11].

In this paper we present background modeling, object detection and classification algorithms for real-time object segmentation on GPU. The object segmentation needs to be implemented by considering GPU characteristics in order to utilize the maximum GPU performance. The importance of our approach is that almost all the computations are performed on the GPU, so that object segmentation can obtain faster results than CPU based approach. To fully utilize the computational power of GPU, the algorithm for object segmentation is modified according to the characteristics of GPU. Also, to maximize the GPU's computation power, the data transmission between CPU and GPU should be minimized.

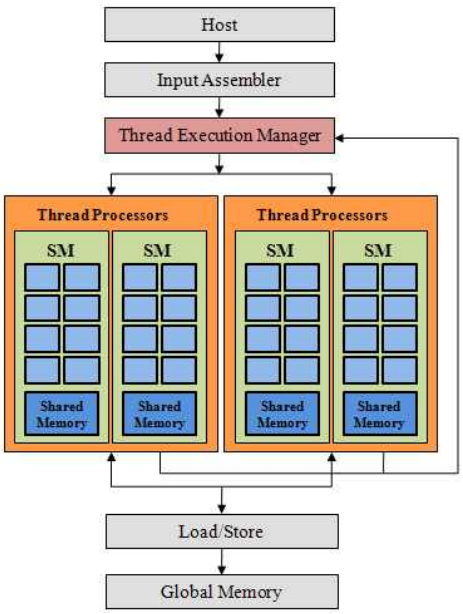
This paper is organized as follows. The architecture of CUDA and programming model are presented in Section 2, and the algorithms for object detection and classification are described in Section 3. Section 4 describes how to implement object segmentation algorithm using CUDA. Section 5 illustrates the experimental results of our proposed methods and shows performance comparisons in computation speed between CPU and GPU, Section 6 conclude our work.

2. CUDA

CUDA has become a standard platform for GPGPU computing in NVIDIA graphics cards with a chipset G80 or superior. In this section, we briefly review the CUDA architecture and programming model to help understand the development environment.

2.1 CUDA Architecture

CUDA contains many SIMD (Single Instruction Multiple Data) stream multi-processors (SM), and each SM consists of 8 stream processors (SPs). In case of NVIDIA G80 series, the chip has a total of 128 SPs distributed across sixteen multiprocessors, each with shared memory, cache, and registers[11]. Shared memory enables parallel data cache from global memory for accelerating memory access. (Fig.1) shows simplified version of CUDA architecture.



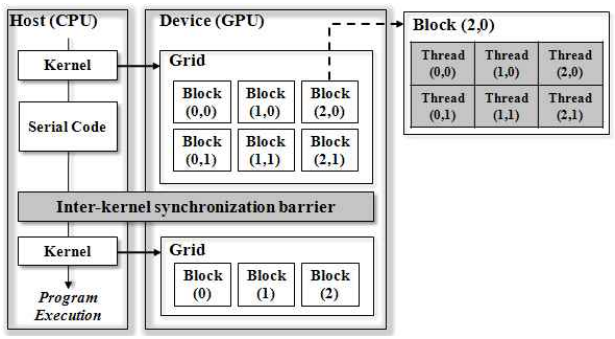
(Fig. 1) Simplified CUDA architecture

The advantages of CUDA architecture are the following:

- Hardware abstraction: NVIDIA has hidden the architectures of its GPUs beneath an application programming interface (API). Programmers need not to know the complex details of the GPU hardware.
- Comfortable development environment: CUDA programming interface provides a relatively simple path for users familiar with the C programming language to easily write programs for execution by the device.
- General DRAM memory addressing: CUDA provides general DRAM memory addressing for more programming flexibility. From a programming perspective, GPU can gather data from any location in DRAM, and also scatter data to any location in DRAM, just like on a CPU.
- Parallel data cache: CUDA has on-chip shared memory with very fast general read and write access, that threads use to share data with each other.
- Thread synchronization: Synchronization within a thread block is entirely managed in hardware. Synchronization among thread blocks is achieved by allowing a kernel to complete and starting a new kernel.

2.2 CUDA Programming Model

The CUDA programming model is similar to the familiar SPMD (single-program multiple data) model in their styles[12]. (Fig. 2) shows the CUDA programming model with an example of CUDA code sequence.



(Fig. 2) CUDA programming model

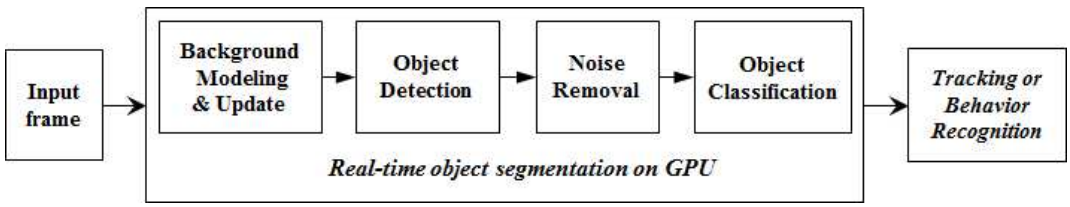
The programmer supplies a single source program encompassing both host (CPU) and kernel (GPU) code. The host code transfers data to and from the GPU's global memory and initiates the kernel code by calling a function. A kernel runs several blocks of threads and each thread performs a single computation. These threads are organized into a hierarchy of grids of thread blocks. A grid of thread blocks consists of a number of blocks that execute the same kernel. A thread block consists of a batch of threads that access data from the shared memory and executes instructions in parallel. The maximum number of threads per block is 512. In (Fig. 2), first kernel 2-D grid is 3x2 thread blocks and each block is 3x2 threads. Kernels are separated by an inter-kernel synchronization barrier.

Threads may access data from multiple memory spaces during their execution. Each thread has a private local memory. Each thread block has a shared memory visible to all threads of the block that has the same lifetime as the block. Finally, all threads have access to the same global memory[11].

3. Real-time Object Segmentation

The object segmentation aims at segmenting regions corresponding to moving objects from the rest of an image. Subsequent processes such as tracking and behavior recognition are greatly dependent on it. Generally, the process of object segmentation involves background modeling, object detection, and object classification.

The background model and update techniques are based on the Gaussian mixture background model and their algorithm deals robustly with lighting changes, repetitive motion of scene elements. After objects are extracted from the background model, the noises are eliminated to get the good object detection, and each object is classified by using our proposed connected component labeling algorithm technique. All of the processes are implemented on GPU.



(Fig. 3) The processing steps for object segmentation

3.1 Background Model and Update

Firstly, each pixel has the background models for moving object extraction. The history of each pixel,{X_1, ...X_t}, is modeled by using a mixture of K Gaussian distributions as background models. The probability is expressed as:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} \times \eta(x_t, \mu_{i,t}, \Sigma_{i,t}) \quad (1)$$

K is normally set to be between 3 and 7. We use 3 because our experimental environment is performed indoor that does not have many changing factor compared to outdoor environment. K Gaussian mixture background model is composed $\mu_{i,t}$ mean, $\Sigma_{i,t}$ covariance of RGB color, and $\omega_{i,t}$ weight for i_{th} distribution at the time t. η is a Gaussian probability density function:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \det\Sigma^{1/2}} \exp^{-\frac{1}{2}(X_t-\mu_t)^T \Sigma^{-1}(X_t-\mu_t)} \quad (2)$$

If a pixel distribution of current input image matches one of the K Gaussian distribution (M = 1), i_{th} Gaussian distribution of matched pixel becomes background, so the factors of background models are updated as follows:

$$w_{i,t} = (1-\alpha) w_{k,t} + \alpha \times M \begin{cases} \text{match} : 1 \\ \text{unmatch} : 0 \end{cases} \quad (3)$$

$$\mu_t = (1-p)\mu_{t-1} + pX_t$$

$$\delta_t^2 = (1-p)\delta_t^2 - 1 + p(X_t-\mu_t)^T(X_t-\mu_t)$$

$$p = \alpha\eta(X_t \mid \mu_k, \Sigma_k),$$

where α is the learning rate. In the our system, if the value of pixel does not match any Gaussian distribution, then we find the minimum weight and replace the attribute values of background distribution by those of the current object distribution(M = 0). The reason is that the object should be merged to the background if it does not move for a longtime. If the pixel value matches several Gaussian distributions, then we select the largest weight, and update its background model's components.

3.2 Foreground Segmentation

When the distribution of each pixel value becomes smaller than 2.5 standard deviation of Gaussian distribution on the background models and the weight of distribution is higher than a threshold, our system recognizes the pixel as a moving object. If the weight of background distribution is lower than the threshold, the pixel is not recognized as the object. However, the attribute values of background distribution are updated similarly as in the previous section.

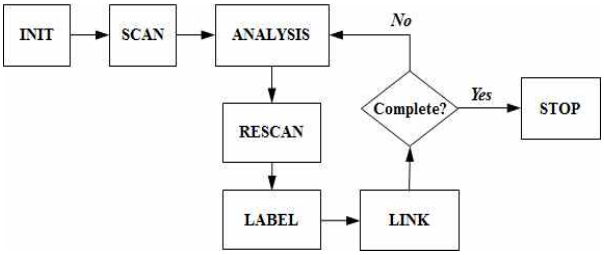
The detected moving objects contain noises, and we need remove the noises. For the removal of noise, we use two methods. One is the morphological operation that is composed by the dilation and erosion. The other is the median filter. The erosion can reduce the size of the spot of a small noise. After that, we use the median filter to remove the smaller noise spot. The use of dilation twice combines the separated nearby segments by enlarging them. After the noise elimination, moving objects are detected by the background modeling and the distribution of the current image pixels. (Fig.4) shows the results after noises are removed.

3.3 Object Segmentation

After detecting the moving object, we must classify

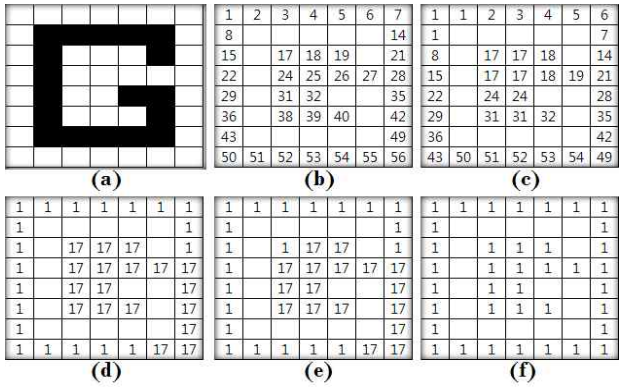
(Fig. 4) The result of object detection

each object for tracking. As a kind of multi-pass algorithm, our proposed algorithm has 6 phases and requires a number of iterations to complete labeling operations



(Fig. 5) Sequence diagram of proposed CCL algorithm

Init phase assigns initial labels of each pixel belonging to objects. If a pixel is belonging to object, unique index of each thread (allocated to each pixel) is assigned to index of the pixel in the Label array. Otherwise, background value is assigned. Thread operations with background pixels will be ignored in following phases. In Scanning phase, examines of directly neighboring pixels belonging to a mask introduced as 'forward scan mask' in [7-8]. After that, threads find and write lowest label including itself to the Label Array. In Analysis phase, each thread find representative label as a root of each pixel to propagate it to each sub-region by using labels written in Label Array. Representative label means a label written in Label Array equal to each pixel index. If threads find each representative label, threads write them into Label Array. Link phase links each connected sub-region to build a fully labeled connected component. Each thread examines labels of neighborhood pixels and find lowest label. If a pixel in a sub-region is directly



(Fig. 6) Results of each phase in Label Array
(a)Input image; (b)Init; (c)Scan; (d)Analysis; (e)Link; (f)Label.

neighboring with other sub-regions, lower or higher representative labels can be found. In Label phase, each thread finds its representative label to its pixel. After this phase, Label Array will have a number of large sub-regions or fully labeled connected components if all directly connected sub-regions are linked. After Label phase, each thread scans labels of all neighborhood pixels to check all directly connected sub-regions are linked. If component labeling is not completed, Rescan phase set a flag to true, then host will repeat execution of Analysis, Link, Label, and Rescan phase in order. (Fig.6) shows the results of each phase in Label Array.

We construct the binary images from the detected moving object image, and then classify each extracted object by its boundary box which encloses it.

<Table 1> shows the result of a comparison of our system to reference papers. As shown table.1, the performance of our system using GPU is very high, compared to reference system.

(Fig. 7) The result of object classification

〈Table 1〉 The comparison of Speed in our system and reference papers

Method	Frame Size	Speed
Adaptive Background Mixture Modeling[15]	160 * 120	11~13 frame/sec
Modeling Using Color and Depth[16]	320 * 240	15 frame/sec
Effective Gaussian Mixture Learning[17]	160 * 120	15 frame/sec
Out System	320 * 240	83 frame/sec

4. Implementation

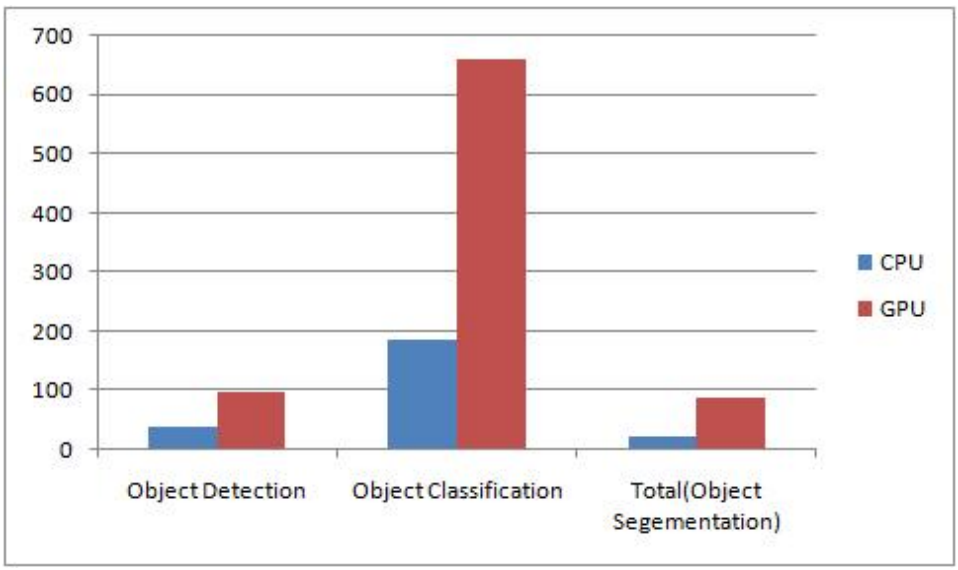
The CUDA programming style does not so look different traditional C language because CUDA is a minimal extension of the C programming language. While CUDA requires analysis of algorithms and data to find the optimal numbers of threads and blocks that will keep the GPU fully utilized [13]. The size of global data and the number of thread processors and block in the GPU can have a significant impact on the overall performance. To help the optimization of performance, NVIDIA provides an Excel spreadsheet called the Occupancy Calculator (OCC) [13]. It considers all factors to suggest the best method of decomposing the data.

We control the number of threads per block to maximize the occupancy of multiprocessor through OCC. In case of input image size 320x240, we put 256 threads in each block and a grid is consisted of 20x15 blocks. In addition, all of the processes only use global memory without shared memory because many operations in algorithms need not data sharing and largely per-pixel independent. The entire images are loaded on to the GPU once before execution of operations to avoid latency due to data transfer. Although our proposed CCL algorithm requires several iterations for object images with connectivity, required iterations are just a few and overall execution time is less than expectation. In a test with 2048x2048 images, they were labeled in just 3 iterations.

5. Experiments

We tested the speed of object segmentation with 320x240 input frames. The GPU is NVIDIA Geforce GTX 260 with 216 stream processors, and the CPU is Intel Core2Quad Q6600 processor (2.4GHz). All of the processes were implemented in C language including CUDA syntax, and compiled under the same condition.

As shown (Fig.8), our experimental result shows that



(Fig. 8)Performance evaluation results (unit:frame/sec.)

our algorithm on GPU is much faster than that on CPU. For object detection including noise removal, the CUDA version achieves 3x speedup compared to CPU version. In case of object classification using CCL algorithm, we compared our proposed algorithm with a conventional labeling algorithm proposed in [14]. For object classification, the speed enhancement of GPU is far bigger than CPU. Finally, we can achieve real-time object segmentation on GPU using CUDA and provide the basis for real-time tracking or behavior analysis.

6. Conclusion

Real-time object segmentation of moving regions in image sequences is a fundamental step in many vision systems including automated visual surveillance, human-machine interface, and low-bandwidth telecommunications. In this paper, we have presented a new novel GPU-based algorithm for real time object segmentation by employing graphics hardware. We have proposed a method to detect moving objects using existing Gaussian Mixture Model and fast connected-component labeling algorithm to classify detected objects. Through our experiments, GPU version shows faster performance than CPU version. In processing of object segmentation, GPU version was at least 3 times faster than CPU version. As a result, the evaluation proves the GPU even good for a real-time system that handles a log of data at the same time. Finally, we can achieve real-time object segmentation on GPU using CUDA and provide the basis for real-time tracking or behavior analysis.

References

[1] P. Kaew, T. Pong and R. Bowden, "An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection," Proc. European Workshop Advanced Video Based Surveillance Systems, Sep., 2001

[2] G. Wel, S. C. Romano R. Lee L. "Using adaptive tracking to classify and monitor activities in a site," in Proceedings.1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1998.

[3] Stauffer C, Grimson W. E. L, "Learning patterns of activity using real-time tracking," IEEE Transactions on Pattern Analysis &Machine Intelligence, vol.22(8), pp.747-57, 2000.

[4] G. M. Harville and J. Woodfill."Foreground Segmentation Usingadaptive Mixture Models in Color and Depth," Workshopon Detection and Recognition of Events in Video, 2001.

[5] P. W.Power and J. A.Schoonees. "Understanding backgroundmixture models for foreground segmentation," IVCNZ, Nov., 2002.

[6] K. Suzuki, I. Horiba and N. Sugie, "Linear-time connected component labeling based on sequential local operations," Comput. Vis. Image Underst. 89(1), pp.1-23, 2003.

[7] L. He, Y. Chao, K. Suzuki and K. Wu, "Fast connected-component labeling," Pattern Recognition.42, pp.1977-1987, 2009.

[8] K. Wu, E. Otoo, K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," Pattern Anal. Applic.12, pp.117-135, 2009.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," Comput. Graph.Forum 26, pp.80-113, 2007.

[10] M. Pharr and R. Fernando, "GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation", Addison Wesley, Messachusetts, 2005.

[11] NVIDIA, 2007, CUDA Technology. Available from: http://www.nvidia.com/CUDA.

[12] Nickolls, J. Buck, I. Garland, M. Skadron. K,"Scalable Parallel Programming with CUDA", ACM6(2), pp.40-53, 2008

[13]Halfhill, T.R.,2008, "Parallel Processing With CUDA", Microprocessor Report [Online] Available from: http://www.MPRonline.com

[14] P. Kumar, K. Palaniappan, A. Mittal, and G. Seetharaman, "Parallel Blob Extraction Using the Multi-core Cell Processor", ACIVS 2009, LNCS 5807, pp.320-332, 2009.

[15] C.Stauffer, W.E.L. Grimson. "Adaptive Backgroud Mixture Models for Real-Time Tracking," Proc. CVPR, Vol.2, pp. 246-252, 1999.

[16] M.Harville, G.Gordon, and J. Woodfill"Adaptive Video Background Modeling Using Color and Depth," Proc. IEEE. Published in the 2001 International Conference on Image Processing (ICIP-2001), October, 7-10, 2001

[17] D. S. Lee"Effective Gaussian Mixture Learning for Video Background Subtraction," IEEE Transactions on Pattern Analysis and Machine Inteeligence, Vol.27, No.5, May, 2005.

강 의 선

e-mail : kanges86@naver.com

2002년 숭실대학교 컴퓨터학과(공학석사)

2007년 숭실대학교 미디어학과(공학박사)

2007년~현 재 숭실대학교 베어드학부 조교수

관심분야 : 멀티미디어, 모바일 웹

유 승 훈

e-mail : capstoney@korea.ac.kr

2003년 한성대학교 정보통신공학과(학사)

2010년 고려대학교 전자컴퓨터공학과 (공학박사)

2010년~2010년 한국과학기술연구원 박사 후 과정

2010년~현 재 삼성전자 책임연구원

관심분야 : 영상 정합, 영상 인식 및 추적, 증강현실 등