http://dx.doi.org/10.14400/JDC.2014.12.11.379

GPGPU를 이용한 Grabcut의 수행 속도 개선 방법에 관한 연구

김지훈*, 박영수**, 이상훈**
광운대학교 대학원*, 광운대학교 교양학부**

A Study of How to Improve Execution Speed of Grabcut Using GPGPU

Ji-Hoon Kim*, Young-Soo Park, Sang-Hun Lee****
Graduate School, Kwangwoon University*
Dept. of General Education, Kwangwoon University**

요 약 본 논문에서는 Grabcut 알고리즘의 수행 속도를 효율적으로 개선시키기 위하여 GPU(Graphics Processing Unit)에서 데이터를 처리하는 방법을 제안한다. Grabcut 알고리즘은 뛰어난 성능의 객체 추출 알고리즘으로 기존의 Grabcut 알고리즘은 전경 영역과 배경 영역을 분할한 후 배경 K-클러스터와 전경 K-클러스터로 할당한다. 그리고 할당 된 결과를 점진적으로 개선될 때까지의 과정을 반복한다. 하지만 Grabcut 알고리즘은 반복된 클러스터링 작업으로 인하여 수행 시간이 오래 걸리는 단점이 존재한다. 따라서 GPGPU(General-Purpose computing on Graphics Processing Unit)를 이용해 반복되는 작업을 병렬적으로 처리하여 Grabcut 알고리즘의 수행 속도를 효율적으로 개선시키는 방법을 제안한다. 제안하는 방법으로 Grabcut 알고리즘의 수행시간을 평균 약 90.668% 감소시켰다.

주제어 : GPU, GPGPU, 병렬 처리, Graphcut, GMM, Grabcut

Abstract In this paper, the processing speed of Grabcut algorithm in order to efficiently improve the GPU (Graphics Processing Unit) for processing the data from the method. Grabcut algorithm has excellent performance object detection algorithm. Grabcut existing algorithms to split the foreground area and the background area, and then background and foreground K-cluster is assigned a cluster. And assigned to gradually improve the results, until the process is repeated. But Drawback of Grabcut algorithm is the time consumption caused by the repetition of clustering. Thus GPGPU (General-Purpose computing on Graphics Processing Unit) using the repeated operations in parallel by processing Grabcut algorithm to effectively improve the processing speed of the method. We proposed method of execution time of the algorithm reduced the average of about 95.58%.

Key Words : GPU, GPGPU, Parallel transaction, Graphcut, GMM, Grabcut

* 본 논문은 2014년 광운대학교 교내 학술 연구비 지원에 의하여 연구되었음
Received 18 September 2014, Revised 25 October 2014
Accepted 20 November 2014
Corresponding Author: Sang Hun Lee
(Kwangwoon University)
Email: leesh58@kw.ac.kr

ISSN: 1738-1916

1. 서론

오늘날 3D 산업은 2009년 개봉된 영화 아바타를 기점으로 3D 콘텐츠의 관심이 증가하게 되었고 이에 따라 3D 콘텐츠의 수요가 증가하였다. 또한 2013년에 개봉한 영화 그래비티는 개봉한지 3일 만에 북미에서 5천 5백만 불의 이익을 거두고 타임지가 선정한 2013년 최고의 영화에 선정되면서 3D 산업의 전망을 더욱 더 밝게 만들었다[1].

3D 영상 제작 방법은 크게 네 가지 방법으로 구분 할 수 있다. CG 영상 제작 방법, 실사 촬영 제작 방법, 실사 촬영과 CG 영상을 결합한 제작 방법, 2D-3D 컨버팅 방법이 있다. 그러나 CG 영상 제작에는 많은 비용과 시간을 필요로 한다는 제약이 따른다. 이에 따라 비용과 시간적인 측면에서 효율적인 2D-3D 컨버팅 방법에 활발한 연구가 진행되고 있다. 2D-3D 컨버팅 방법은 영상 내에서 전경과 후경, 객체와 배경 등으로 영역을 분할하는데 기반으로 한 제작 방법이다[2,3].

영상 분할의 대표적인 알고리즘인 Grabcut 알고리즘은 기존의 Graphcut 알고리즘을 반복적으로 사용하여 단일 영상 내에서 물체 영역과 배경 영역을 지정하여 지정된 영역을 바탕으로 객체와 배경 영역을 분할하는 알고리즘으로 단일 영상 내에서 객체와 배경 영역을 분할하는데 효율적인 알고리즘이다[4].

우선 지정되지 않은 모든 전경 영역에 화소를 할당하고 화소를 유사한 색상의 클러스터로 묶은 후 전경과 배경 화소의 경계를 이용하여 전경 영역과 배경 영역 분할을 결정한다. 이때, 유사한 색상으로 화소를 묶을 때는 최적화 과정을 통해 수행하고, 상대적으로 비슷한 명암을 갖는 영역의 경계는 봉쇄하도록 한다. 이와 같은 클러스터링 과정을 반복하여 새롭게 최적화를 거친 분할 결과를 얻을 수 있다. 즉, 분할 결과를 점진적으로 개선할 때까지의 과정을 반복하여 수행한다. Grabcut 알고리즘은 단일 영상 내에서 주요 객체를 추출함에 있어서는 뛰어난 성능을 보이는 알고리즘이지만 개선 된 결과를 얻을 때까지의 과정을 반복하기 때문에 수행 속도가 느리다는 단점이 있다.

본 논문에서는 Grabcut 알고리즘의 수행 속도를 개선하기 위하여 반복되는 작업을 GPGPU(General Purpose Graphics Processing Unit)의 CUDA(Compute Unified Divice Architecture)를 이용하여 병렬로 처리하고자 한다[5].

2. 관련 연구

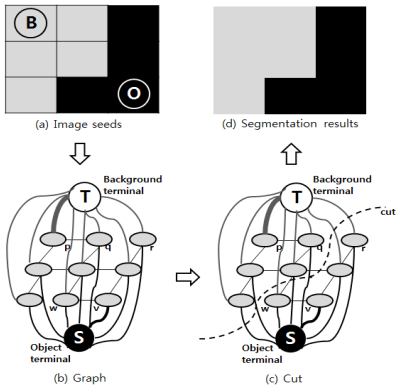
2.1 Graphcut

Graphcut 알고리즘은 객체와 배경 영역을 화소의 부분 집합으로 구분하여 모든 분할의 최소값을 찾아 분할하게 된다. 이때, 부분 집합의 $O \subset P$ 와 $B \subset P$ 에 의해 $O \cap B = \emptyset$ 가 성립할 때, 다음과 같은 조건을 만족해야 한다.

$$\forall_p \in O, \quad A_p = \text{"obj"} \qquad \text{(eq. 1)}$$

$$\forall_p \in B, \quad A_p = \text{"bkg"} \qquad \text{(eq. 2)}$$

[Fig. 1]은 간단한 이미지 분할의 예를 나타내고 있다.



[Fig. 1] A simple 2D segmentation example for image

우선 [Fig. 1](b)와 같이 최적의 최소컷을 계산하여 Background Terminal과 Object Terminal을 분리하여 객체 영역과 배경 영역으로 나뉜 그래프를 생성한다. Object Terminal을 source S 그리고 Background Terminal을 sink T라 정의하면 다음과 같은 관계가 성립된다.

$$v = P \cup \{S, T\} \qquad \text{(eq. 3)}$$

Edges ϵ는 n-links(neighborhood links) 그리고 t-links(terminal links)로 구성된다. 화소 p는 2개의 t-link {p, S}, {p, T}에 연결하고 N에 인접한 화소 {p, q}는 각 쌍의 n-link에 의해 연결된다. 이때, Edges ϵ는 다음과 같이 정의 된다.

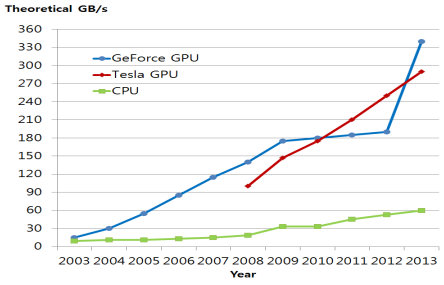
$$\epsilon = N \bigcup_{p \in P} \{p, S\}, \{p, T\} \quad \text{(eq. 4)}$$

이와 같이 최소값을 찾아 객체와 배경의 경계를 그려서 Graphcut 알고리즘을 수행하게 된다[6].

2.2 Parallel Processing

지난 몇 년 동안 기술의 발전으로 하드웨어와 소프트웨어 분야에 있어서 괄목할만한 성장을 이루었다. 그 결과로 CPU의 속도가 상당히 개선되었지만 현재에 이르러 메모리의 제한이 있었고 이상적인 수행 속도를 내기에 한계에 도달하게 되었다. 또한 보다 더 복잡한 응용 프로그램을 수행하기에는 많은 문제점이 존재했고 이런 문제점을 해결하기 위한 방법으로 병렬 분산 처리가 연구되었다. 현재에는 병렬 분산 처리가 영상 처리 분야에도 적용되고 있다[7].

병렬 분산 처리 연구의 결과로 GPGPU가 개발되었다. GPGPU(General-Purpose computing on Graphics Units)는 컴퓨터 그래픽을 위한 연산만을 다루는 GPU를 사용하여 응용 프로그램을 수행하는 기술이다. GPU(Graphic Processor Unit)는 엄청난 계산력과 매우 높은 메모리 대역폭, 고도의 병렬화, 멀티 쓰레드, 다중 코어 프로세서로써 실시간으로 고화질의 3D 그래픽과 프로그래밍이 가능하다[8]. [Fig. 2]는 CPU와 GPU의 메모리 대역폭을 그래프로 나타낸 것이다.

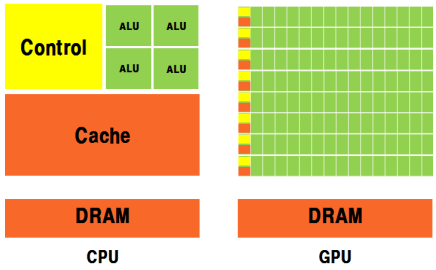


[Fig. 2] Memory Bandwidth for the CPU and GPU

CPU와 GPU의 성능 차이가 나타나는 이유는 각 각의 프로세서 간에 설계 구조가 다르기 때문이다.

CPU는 외형적으로는 작업을 순차적으로 수행하지만 실제로는 단일 쓰레드를 구성하는 명령어들을 병렬과 비순차적으로 실행하는 복잡한 제어 로직을 사용한다. 그리고 복잡한 응용 프로그램에서 명령어와 데이터의 접근 시간을 줄이기 위해 큰 캐시 메모리를 사용하기 때문에 최대 연산 속도에는 영향을 미치지 못한다.

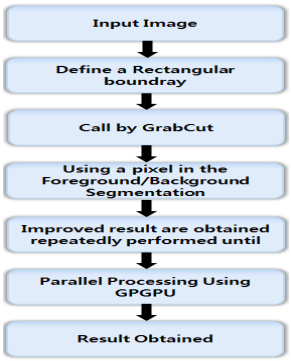
GPU는 반면에 간단한 메모리 모델을 사용할 수 있고, 기존에 작성된 소프트웨어에 대한 제약이 없어서 상대적으로 쉽게 메모리 대역폭을 높일 수 있다. 또한 많은 수의 쓰레드를 통해 처리량을 최적화 하도록 설계되어 있고, 메모리의 접근을 기다리는 동안 많은 수의 쓰레드들이 동시에 다른 작업을 할 수 있어서 동일한 메모리에 대해 여러 쓰레드들이 접근할 필요가 없기 때문에 작은 크기의 캐시 메모리를 제공하여 부동 소수점 연산에 매우 효율적이다[9,10]. [Fig. 3]은 CPU와 GPU의 설계 구조를 비교하고 있다.



[Fig. 3] Structure of the CPU and GPU

3. 제안하는 방법

제안하는 방법은 먼저 입력 된 영상에서 직사각형의 경계를 정의한다. 그리고 Grabcut 알고리즘을 호출하여 화소의 경계를 이용해 객체와 배경 영역을 분할하고 개선된 결과를 얻을 때 까지 이 작업을 반복하여 수행하게 된다. 이때 반복되는 작업으로 인한 Grabcut 알고리즘의 처리 속도의 효율성을 위해 GPGPU를 기반으로 하여 병렬 분산 처리로 작업을 수행한다. [Fig. 4]는 제안하는 방법의 흐름도를 나타낸다.



[Fig. 4] A flowchart of proposed method

3.1 Grabcut

Grabcut 알고리즘은 단일 영상 내의 객체 또는 배경에 속하는 화소의 일부인 레이블을 사용한다. 부분 레이블링에 기반을 두어 객체와 배경 영역을 분할하게 된다.

3.1.1 초기화

우선 라벨링을 위하여 영역을 지정하게 된다. 지정 된 영역의 안은 객체영역, 지정된 영역의 밖은 배경으로 분류되는 trimap T를 생성한다. trimap T는 객체 영역인 T_F, 배경 영역인 T_B, 그리고 나머지 영역인 T_U로 구성된다. 초기화 단계에서 불완전한 라벨링이 되지 않도록 하기위해 전체 trimap T 대신에 $T_F = 0$ 을 제외하고 배경 영역을 T_B로 지정하도록 한다.

3.1.2 색상 데이터 모델링

RGB 색상 공간에서 단일 영상의 이미지는 화소 z_n으로 구성된다. 정확한 색 공간 히스토그램을 구성하기 위하여 GMM(Gaussian Mixture Model)을 이용하고 객체와 배경에 대한 GMM은 전체 공분산으로 간주한다[11]. 이때, 발생하는 에너지 함수에 대한 정의는 다음과 같다.

$$E(\underline{\alpha}, k, \underline{\theta}, z) = U(\underline{\alpha}, k, \underline{\theta}, z) + V(\underline{\alpha}, z) \quad \text{(eq. 5)}$$

식 (5)에서 $\underline{\alpha}$는 단일 영상에서 객체와 배경 분할의 각각의 화소에서 불투명한 값을 나타내고 z는 단일 영상의 배열을 나타낸다. 그리고 k=5를 할당받고 데이터 U는 다음과 같이 정의된다.

$$U(\underline{\alpha}, k, \underline{\theta}, z) = \sum_n D(\alpha_n, k_n, \underline{\theta}, z_n) \quad \text{(eq. 6)}$$

$$D(\alpha_n, k_n, \underline{\theta}, z_n) = -\log\pi(\alpha_n, k_n) + \frac{1}{2} log\det\sum(\alpha_n, k_n) + \frac{1}{2}[z_n - \mu(a_n, k_n]^\top \sum(\alpha_n, k_n)^{-1}[z_n - \mu(\alpha_n, k_n)] \quad \text{(eq. 7)}$$

여기서 히스토그램 모델 변수 $\underline{\theta}$는 다음과 같이 정의 할 수 있다.

$$\underline{\theta} = \pi(\alpha, k), \mu(\alpha, k), \sum(\alpha, k) \quad \text{(eq. 8)}$$

식 (8)에서 α는 객체나 배경에서 각 화소의 고유한 GMM 구성 요소를 나타내고 π, μ, $\sum$는 객체와 배경 분할에 대한 $2K$ 가우시안 구성요소이다. V에 대한 정의는 다음과 같다.

$$V(\underline{\alpha}, z) = \gamma \sum_{(m,n)\in C} [\alpha_n \neq \alpha_m] \exp -\beta\|z_m - z_n\|^2 \quad \text{(eq. 9)}$$

3.1.3 반복적 최소화 에너지 분할

첫 번째, 각 화소 n의 값 k_n에 대해 파악한다.

$$k_n = \arg\min_{k_n} D_n(\alpha_n, k_n, \theta, z_n) \quad \text{(eq. 10)}$$

두 번째는 가우시안 파라미터 추정방법으로 데이터 z에서 GMM 매개 변수를 알 수 있다. GMM 구성요소의 부분집합 즉, 객체의 화소는 $F(k) = \{z_n : k_n = k \text{ and } \alpha_n = 1\}$로 정의할 수 있다.

$$\underline{\theta} = \arg\min_{\underline{\theta}} U(\underline{\alpha}, k, \underline{\theta}, z) \quad \text{(eq. 11)}$$

마지막으로 최소컷을 사용하여 전역 최적화를 한다.

$$\min_{\alpha_n : n\in T_U} \min_k E(\underline{\alpha}, k, \underline{\theta}, z) \quad \text{(eq. 12)}$$

첫 번째 단계부터 세 번째 단계를 반복하여 각각의 변수 k, $\underline{\theta}$, $\underline{\alpha}$에 대한 에너지를 최소화 할 수 있다. 따라

서 에너지가 감소하여 수렴이 가능하게 된다[12].

3.2 GPGPU를 이용한 병렬화

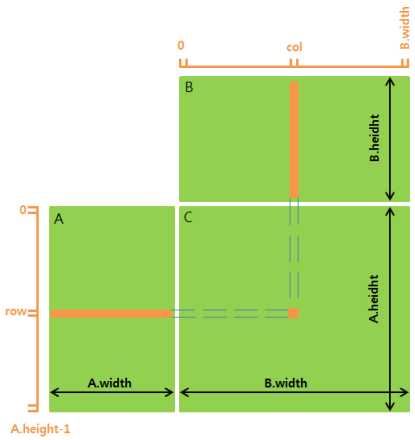
기존 CPU는 단방향으로 직렬 수행하는데 최적화 되어있지만 이는 복잡한 응용 프로그램을 수행하기에 한계에 다다르게 되었고 이에 따라 GPU를 이용한 병렬 분산 처리 방식이 개발되었다. GPGPU를 이용한 GPU에서의 처리 방법은 CUDA를 사용하여 다음과 같이 진행된다.

3.2.1 초기화

초기화하는 동안 시스템의 각 장치는 콘텍스트를 만들게 된다. 이 콘텍스트는 장치의 기본적인 콘텍스트 이며 응용 프로그램의 모든 호스트 쓰레드 간에 공유를 하게 된다. 호스트 쓰레드를 호출하게 되면 이 호스트 쓰레드가 현재 작동중인 장치의 기본 콘텍스트를 제거하고 이 장치에 대한 새로운 기본 콘텍스트를 만들어 현재 작동중인 장치가 호스트 쓰레드를 만들어 다음 함수를 호출하게 된다.

3.2.2 메모리 공유

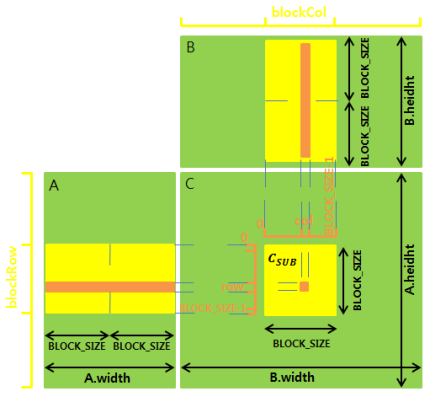
CUDA 프로그래밍의 호스트 및 장치들은 별도의 메모리가 각각 구성된 시스템으로 가정한다. 커널은 장치 메모리에서 작동하므로 메모리 장치 사이에서 메모리를 할당, 할당해제, 복사뿐만 아니라 데이터 전송의 기능을 한다. 메모리 장치는 메모리의 선형 또는 배열 중 하나로 할당될 수 있다.



[Fig. 5] Matrix Multiplication without Shared Memory

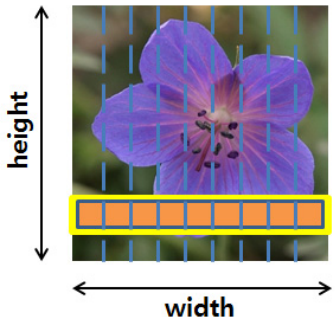
기존의 공유 메모리가 없는 행렬은 각 쓰레드가 하나의 행 및 열을 읽고 해당 요소를 계산하게 된다. 예를 들어 전역 메모리에서 B.width 를 읽고서 다시 A.Height를 읽는 것과 같다. [Fig. 5]는 기존의 메모리 할당 방식을 나타내고 있다.

기존의 메모리 할당 방식은 공유 메모리는 사용하지 않아서 비효율적인 문제가 존재한다. 이와 같은 문제를 해결하기 위하여 쓰레드 블록 C중 하나의 부분 행렬 C_{SUB}를 할당한다. 부분 행렬 C_{SUB}에서 같은 행과 열을 읽어서 작업을 수행하게 되고 공유 메모리에 각각 행렬 하나의 요소로부터 전역 메모리 행렬에 대응하게 된다. [Fig. 6]은 부분행렬 C_{SUB}을 할당하여 나타낸 그림이다.



[Fig. 6] Matrix Multiplication without Shared Memory

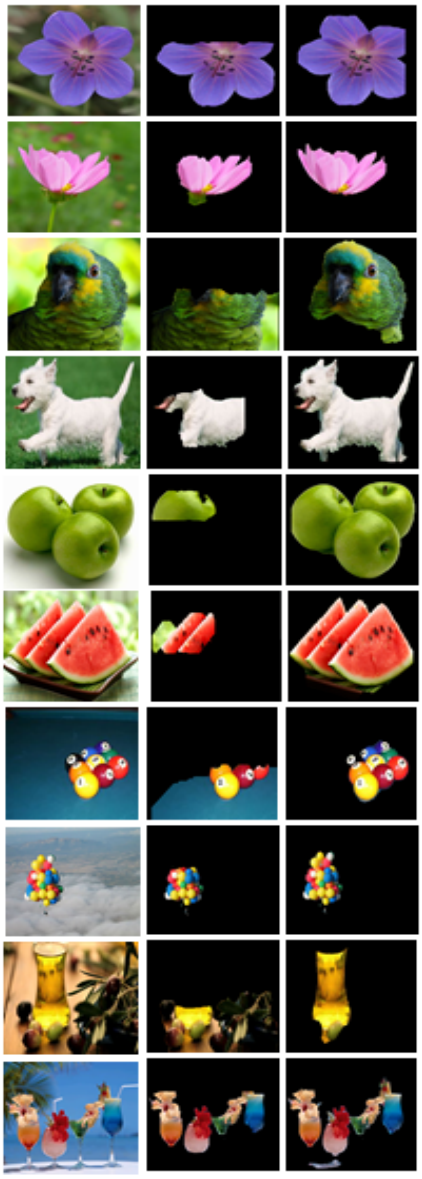
이와 같은 과정을 거쳐서 GPGPU를 이용하여 GPU에서 하나의 행과 하나의 열을 읽어 들여 한 클록으로 처리되는 과정을 [Fig. 7]에서 나타내고 있다.



[Fig. 7] Matrix Multiplication without Shared Memory

4. 실험 및 고찰

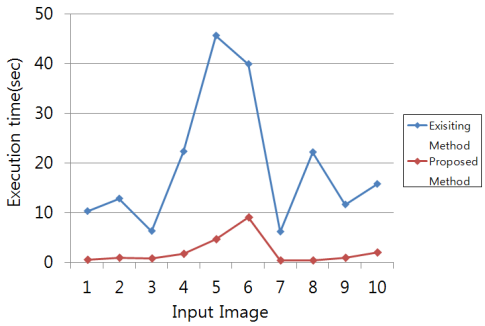
본 실험은 NVIDIA GeForce 9600GT 으로 수행 하였으며 객관적인 비교 실험을 위하여 각각의 이미지마다 10회씩 걸쳐 진행하였다. [Fig .8]은 실험 결과이다.



(a) (b) (c)

[Fig. 8] (a) Input Image, (b)Existing Method, (c)Proposed Method

제안하는 방법으로 실험한 결과 기존의 방법보다 단일 영상 내 객체 추출에 있어서 동등하거나 더 뛰어난 성능을 보였음을 확인하였다. [Fig. 9]는 실험 결과를 그래프로 나타낸 것이다.



[Fig. 9] Experimental Result Graph

[Fig. 9]의 그래프는 각각의 이미지마다 기존의 방법을 CPU로 처리하였을 때와 제안하는 방법인 GPGPU를 이용하여 GPU에서 처리 하였을 때의 결과를 나타내고 있다. 첫 번째 입력 이미지를 CPU로 10회 수행 했을 때 평균 수행시간은 10.204sec, GPGPU로 10회 수행 했을 때의 평균 수행시간은 0.575sec를 나타내었다. 따라서 CPU로 작업을 처리하는 것보다 GPGPU를 이용하여 GPU에서 작업을 처리하는 것이 수행 시간이 감소되었음을 보였다.

각각의 이미지를 기존의 방법과 제안하는 방법으로 실험한 수행 시간의 평균을 <Table 1>에 나타내었다.

〈Table 1〉 Existing Method and Proposed Method of Average

	Existing Method	Proposed Method
Execution times(sec)	19.266 (sec)	2.125 (sec)

기존의 방법인 CPU에서 작업을 수행하는 것보다 제안하는 방법인 GPGPU를 이용하여 GPU에서 작업을 수행한 결과 수행 시간이 평균 약 90.668% 감소시킬 수 있음을 확인하였다.

5. 결론

Grabcut 알고리즘은 기존의 Graphcut 알고리즘을 응용한 알고리즘으로써 단일 영상 내에서 주요 객체를 추출함에 있어서는 뛰어난 성능을 보이는 알고리즘이다. 하지만 전경과 배경 영역을 나누고 화소의 경계를 이용하여 분할한 후 결과가 개선될 때까지의 반복적인 클러스터링 작업을 수행하기 때문에 수행 속도가 느리다는 단점이 있다. 따라서 본 논문에서는 Grabcut의 반복되는 작업을 GPU의 한 클록을 설정하여 하나의 마스크내의 연산을 하나의 GPU 쓰레드가 GPGPU를 이용해 병렬로 수행함으로써 단일 영상 내에서 객체를 효율적으로 추출할 수 있는 방법을 제안하였다. GPGPU를 이용하여 수행시간이 단축되었음을 확인하였다.

향후 연구로는 병렬 분산 처리 방법으로 작업을 수행시 병렬도가 높은 응용 프로그램에서만 뛰어난 성능을 보이는 부분의 보완이 필요하다. 본 논문에서 제안하는 병렬 분산 처리 방법을 3D 영상 제작 방법에 활용한다면 시간과 비용적인 측면에서 효율성을 발휘하여 3D 콘텐츠의 수요를 충족시켜 3D 산업을 한층 더 밝게 할 수 있다.

ACKNOWLEDGMENTS

This study is sponsored by the 2014 research fund of Kwangwoon University.

REFERENCES

[1] Hyun-Ho Han, Gye-Dong Chung, Young-Soo Park, Sang-Hun Lee, Foreground Extraction and Depth Map Creation Method base on Conversion, The Journal of Digital Policy & Management, Vol. 11, No. 1, pp. 243-248, 2013.

[2] Tae-Hoon Yoo, Gang-Seong Lee, Young-Soo Park, Jong-Yong Lee, Sang-Hun Lee, A Study of Depth Estimate using GPGPU in Monocular Image, The Journal of Digital Policy & Manaagement, Vol. 11, No. 12, pp. 345-352, 2013.

[3] Yeong-Kang Lai, Yu-Fan Lai, Ying-Chang Chen, An Effective Hybrid Depth-Generation Algorithm for 2D-to3D Conversion in 3D Displays, Display Technology, Vol. 9, pp. 154-161, 2013.

[4] Wang Rui, Peng Jinye, Che Liping, Hou Yuting, Improved color image segmentation algorithm base on Grabcut, Applied Mechanics and Materials, Vol. 373-375, pp. 464-467, 2013.

[5] Tae-Hee Lee, Bo-Hyun Hwang, Jong-Ho Yun, Myung-Ryul Choi, A Road Extraction Using OpenCV CUDA To Advance The Processing Speed, Journal of digital convergence, Vol. 12, No. 6, pp. 231-236, 2014.

[6] Boykov, Y.Y., Jooly, M. -P., Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images, Computer Vision, ICCV2001, Vol. 1, pp. 105-112, 2001.

[7] Prajapai, H.B., S.K., Analytical Study of Parallel and Distributed Image Processing, Image Information Processing, ICIIO, pp. 1-6, 2011.

[8] Don-Geon Lee, Dong-Kun Shin, The compare performance of CUDA with OpenMP Application for research of GPGPU programming model, IEEK, Vol. 2010, No. 11, pp. 499-500, 2010.

[9] Feng Ji, Heshan Lin, Xiaosong Ma, RSVM: A Region-based Software Virtual Memory for GPU, Parallel Architectures and Compilation Techniques, PACT, pp. 269-278, 2013.

[10] Corporation NVIDIA, CUDA C PROGRAMING GUIDE (version 6.0), NVIDIA Corporation, 2014.

[11] Chen, D., Chen, B., Marnic, G., Fookes, C., Sridharan, S., Improved Grabcut Segmentation via GMM Optimisation, Digital Image Computing : Techniques and Applications, DICTA, pp. 39-45, 2003.

[12] Pother, C., Kolmogorov, V., Blake, A., Grabcut : Interactive Foreground Extraction using Iterated Graph Cuts, ACM Transaction on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2004, TOG, vol. 23, pp. 309-314, 2004.

김 지 훈(Kim, Ji Hoon)

- 2014년 2월 : 광운대학교 컴퓨터공학과(학사)
- 2014년 2월 ~ 현재 : 광운대학교 대학원 플라즈마바이오디스플레이학과
- 관심분야 : 네트워크, 영상인식, 3D 영상처리
- E-Mail : kimjihoon@kw.ac.kr

박 영 수(Park, Young Soo)

- 1996년 2월 : 광운대학교 전산대학원 전산학과(석사)
- 2006년 2월 : 광운대학교 대학원 컴퓨터과학과(박사)
- 2011년 2월 ~ 2012년 2월 : 광운대학교 정보과학교육원 컴퓨터공학과 주임교수
- 2013년 2월 ~ 현재 : 광운대학교 조교수
- 관심분야 : 소프트웨어엔지니어링, XML, 웹 서비스, 분산처리, 무인인터넷, 모바일 컴퓨팅, 3D 영상처리
- E-Mail : yspark@kw.ac.kr

이 상 훈(Lee, Sang Hun)

- 1983년 2월 : 광운대학교 응용전자공학과(학사)
- 1987년 2월 : 광운대학교 대학원 전자공학과(석사)
- 1992년 2월 : 광운대학교 대학원 전자공학과(박사)
- 1990년 2월 ~ 현재 : 광운대학교 정교수
- 2001년 2월 ~ 2007년 2월 : 세계기능경기대회 (심사위원)
- 2006년 2월 ~ 2007년 2월 : 서울특별시 기능경기위원회 (기술위원장)
- 2010년 2월 ~ 2012년 2월 : 광운대학교 교양학부장
- 2012년 2월 ~ 2013년 2월 : 광운대학교 정보통신처장
- 2013년 2월 ~ 현재 : 광운대학교 학생복지처장
- 관심분야 : 무선인터넷, 무선네트워크, USN, 영상인식, 3D 영상처리
- E-Mail : leesh58@kw.ac.kr