JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 20, No. 12 : 2355~2362 Dec. 2016

GPU를 이용한 영상기반 고속 해무제거 기술

최운식[1] · 이윤혁[2] · 서영호[2] · 최현준[1*]

Digital Image based Real-time Sea Fog Removal Technique using GPU

Woon-sik Choi[1] · Yoon-hyuk Lee[2] · Young-ho Seo[2] · Hyun-jun Choi[1*]

[1*]Dept. of Electronic Engineering, Mokpo Maritime University, Mokpo 58628, Korea
[2]DDnT Lab., Kwangwoon University, Seoul 01897, Korea

요 약

해무 제거는 컴퓨터 비전과 영상처리 분야에서 상당히 중요하게 다루고 있는 분야이다. 해무 혹은 안개제거 기술은 자동 제어 시스템, CCTV, 영상인식 등과 같은 여러 분야에서 사용되고 있다. 이와 같이 컬러 영상의 해무 제거 기술이 다양하게 연구되고 있고 특히 Dark Channel Prior (DCP) 기술을 이용한 방법이 가장 활발하게 이용되고 있다. 본 논문에서는 DCP 알고리즘을 적용하여 해무를 빠르고 효율적으로 제거하는 기술을 소개한다. 이 기술은 GPU를 기반으로 구현한다. 병렬 프로그래밍과 최적화 과정을 거쳐 약 250배 정도의 연산속도를 빠르게 개선하였다. 이를 위해 기존의 프로그램 일부분을 몇 가지 과정을 거쳐 병렬화와 최적화 과정을 수행하였다. 제안한 GPU 프로그래밍 알고리즘과 구현결과는 선박의 안전항해, 지형조사, 지능형 자동차 등과 같은 분야에 적용될 수 있을 것으로 기대된다.

ABSTRACT

Seg fog removal is an important issue concerned by both computer vision and image processing. Sea fog or haze removal is widely used in lots of fields, such as automatic control system, CCTV, and image recognition. Color image dehazing techniques have been extensively studied, and expecially the dark channel prior(DCP) technique has been widely used. This paper propose a fast and efficient image prior - dark channel prior to remove seg-fog from a single digital image based on the GPU. We implement the basic parallel program and then optimize it to obtain performance acceleration with more than 250 times. While paralleling and the optimizing the algorithm, we improve some parts of the original serial program or basic parallel program according to the characteristics of several steps. The proposed GPU programming algorithm and implementation results may be used with advantages as pre-processing in many systems, such as safe navigation for ship, topographical survey, intelligent vehicles, etc.

키워드 : 해무, 해무 제거, GPU, 안전항해

Key word : Sea-fog, Sea-fog Removal, GPU, Safe Navigation

Received 20 July 2016, **Revised** 21 July 2016, **Accepted** 01 August 2016

* **Corresponding Author** Hyun-Jun Choi(E-mail:hjchoi@mmu.ac.kr, Tel:+82-61-240-7273)
Department of Electronic Engineering, Mokpo Maritime University, Mokpo 58628, Korea

Open Access http://doi.org/10.6109/jkiice.2016.20.12.2355 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

최근 여러 해양 사고를 계기로 해양 사고 및 선박 안전에 대한 관심이 높아지고 있다. 국가지표체계의 통계에 따르면 연안 선박의 해양사고 원인 중 해무(Sea-fog)가 있을 시 선박의 시계확보가 이루어지지 않아 선박간의 충돌 및 여러 가지 사고의 발생이 높은 비율을 차지하는 것으로 나타났다[1].

해상에서의 기상악화로 인해 발생하는 시계저하 문제를 해결하기 위해 디지털 영상기기들을 이용하여 영상신호를 획득한 후 디지털 영상처리 기법을 이용하는 방법들이 제안되고 있다. 카메라를 통해 획득한 영상이 외부 환경의 해무나 연기, 황사 와 같은 대기의 입자에 의해 빛의 산란, 흡수 현상이 발생하여 깨끗하지 못할 경우 다양한 알고리즘을 이용한 영상처리가 쉽지 않다. 해무가 짙게 낀 상태에서 획득한 영상은 해무의 농도에 따라서 같은 색을 가진 객체라도 각기 다른 색을 가진다. 농도가 짙을수록 원래의 색을 많이 잃어버리므로 안개가 있는 영상을 안개가 없을 때 획득한 영상과 유사하게 만드는 다양한 방법들이 제안되었다.

최근에 발표된 안개제거 기술은 두 개의 서로 다른 편광렌즈 및 카메라를 이용하거나, 다양한 날씨 상황 아래 고정된 장소에서 여러 장의 사진을 이용한 안개를 제거하는 방법 등이 있다[2,3]. 하지만 이러한 방법들은 두 개 이상의 영상이 필요하거나 공간 정보에 대한 데이터가 필요하다는 문제점을 가지고 있다. 따라서 최근 연구되고 있는 안개제거 기술들은 단일 영상만을 이용하는 추세이다. 그 중 DCP(Dark Channel Prior) 알고리즘이 대표적인 단일 영상을 이용하는 방법이다[4,5]. 이 방법은 안개가 없는 영상에서는 각 화소의 RGB값 중에 한 값이 0에 가까운 화소 값이 존재한다는 관측에 기반 한다. 안개가 짙은 영역일수록 전체적인 화소가 밝아진다는 성질을 이용하여 전달량을 구하고 이를 통해 영상을 복원한다. 매우 깨끗한 결과 영상을 얻을 수 있으나 전달량을 정련하는 과정에서 Bilateral 필터를 이용하여 연산 량이 과다하여 실시간 처리에 걸림돌이 되고 있다. 매우 큰 행렬을 사용하기 때문에 메모리 사용량이 많고 속도 면에서도 제한이 있다. 따라서 본 논문에서는 DCP 알고리즘을 GPGPU(General Purpose computing on Graphic Processing Units, 이하 GPU)를 기반으로 구현하여 연산 속도를 개선시키는 기술을 제안하였고, 구현결과를 해무영상에 적용하여 연안을 항해하는 선박의 안전항해 기술로의 가능성을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 해무제거 기술의 기반이 되는 DCP 알고리즘에 대해 설명하고, 3장에서는 DCP 알고리즘의 GPU구현방법에 대해 기술한다. 4장에서는 구현한 GPU 기반의 해무제거 시스템을 실제 운항 중인 선박의 선교(Bridge)에 설치하여 성능을 검증한다. 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 해무제거 기술

2.1. DCP 알고리즘

해무제거 알고리즘을 이용한 해무제거 과정은 그림 1의 순서도와 같은 과정을 거친다. 입력된 영상에 대한 dark channel을 찾고 이를 이용하여 대기광을 추측한다. 구해진 대기광을 이용하여 영상을 정규화한 값으로 다시 dark channel을 찾는다. 이 dark channel을 이용하여 전달량을 구하고 전달량을 정제하면 해무가 제거된 영상에서 후광효과가 나타나게 된다. 이 후광효과를 줄이기 위해 bilateral 필터를 사용하여 전달량을 정제한다. 이렇게 정제가 끝나면 입력영상과 대기광, 전달량을 이용하여 연무가 제거된 영상을 얻을 수 있다.

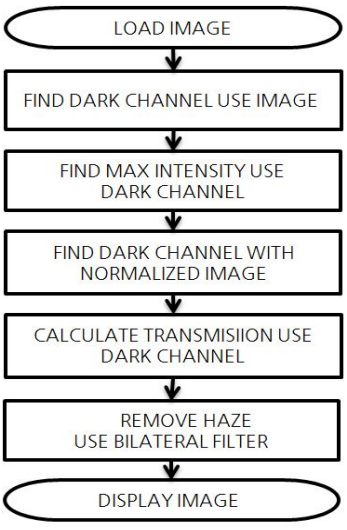

Fig. 1 Flowchart of dark channel prior algorithm

2.2. 해무 영상

그림 2(a)와 같은 해무가 포함된 영상은 아래의 식 (1)과 같은 함수로 표현할 수 있다.

$$I(x) = J(x)t(x) + A(1-t(x)) \quad (1)$$

여기서 *I(x)*는 입력된 영상, *A*는 해무의 밝기 정도를 나타내며, *t(x)*는 빛의 전달량(transmission) 으로 빛이 산란되지 않고 카메라까지 도달된 정도를 나타낸다. *J(x)*는 해무가 제거된 영상(그림 2(b))이다. 따라서 해무 제거 영상처리는 입력 영상 *I(x)*로부터 *A*, *t(x)*, 그리고 *J(x)*를 구하는 것이다.

(a)

(b)

Fig. 2 (a) input image, (b) dehazing result of (a)

2.3. Dark Channel Prior

Dark channel이란 영상에서 일정한 패치사이즈 안에서 한 화소의 RGB값 중에 가장 작은 값의 집합으로 정의한다. RGB 화소 값 중 가장 작은 값을 구한 후 이 값을 그 영역의 대표 값으로 지정하여 최소값 채널(minimum channel)을 구한다. 그리고 이 최소값 채널을 이용하여 일정한 크기의 마스크(mask)를 통해 영역내의 모든 화소 값을 대표 값으로 대체하여 얻은 채널이 dark channel이다. 구체적인 과정은 그림 3과 같다.

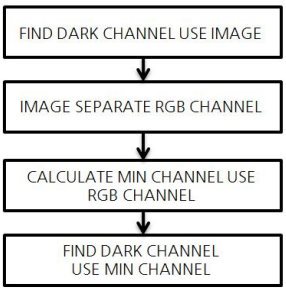

Fig. 3 Dark channel calculation process

이 때 마스크의 크기에 의해 dark channel을 계산하는 실행 시간이 결정된다. 마스크의 크기가 크면 실행 시간이 길어지고 큰 패치 사이즈로 인해서 연무가 제거되는 과정에서 연무가 제거 되지 않고 뿌옇게 후광이 남아있는 것을 그림 4를 통해 볼 수 있다.

Fig. 4 Halo effect image

이 후광 효과를 없애기 위해서 전달량 정제 시 필터를 통한 정제를 하게 된다. 이때, 최적의 마스크 크기는 실험을 통해 설정하게 되는데, 본 논문에서는 9×9 마스크를 사용하였다.

또한 해무가 없는 깨끗한 영상에서는 각 화소의 RGB값 중에 한 값이 0에 가까운 값이 되는 것을 볼 수 있다. 이를 식으로 표현하면 다음의 식 (2)와 같다.

$$J^{dark}(x) = \min_{c\in r,g,b}(\min_{y\in\Omega(x)}(J^c(y))) \quad (2)$$

식 (2)에서 JC는 J의 각 컬러 채널을 의미하고, $\Omega(x)$는 x점을 중심으로 한 일정 구간을 나타낸다. 해무가 없는 일반적인 영상에서 dark channel을 보면 하늘을 제외한 대부분의 영역에서 '0'에 가까운 값을 가지므로 다음과 같이 표현된다.

$$J^{dark} \approx 0 \quad (3)$$

해무가 없는 영상들의 dark channel을 회색조 영상(gray-scale image)으로 보면 그림 5와 같은 낮은 명도의 영상임을 알 수 있다. 따라서 식 (2)를 이용해 계산한 dark channel을 이용하여 안개의 밀도(양)을 예측할 수 있고 전달량을 구할 수 있다.

dark channel을 이용하여 해무를 제거하는 자세한 내용은 [5]의 내용을 참고하기 바란다.

Fig. 5 Dark Channel of natural images

Ⅲ. GPU 구현

GPU의 동작은 SIMT(Single Instruction Multiple Thread)로 수행된다. 즉, 하나의 명령어로 여러 쓰레드를 동작하는 형태이다. GPU내의 하나의 SP(Streaming Processor, 실제 연산하는 H/W)는 4개의 쓰레드(thread)를 하나의 명령어로 수행한다. 하나의 SM내의 모든 SP는 하나의 명령어를 동시에 처리하는데 이를 워프(warp)라 하고 이 단위로 조건에 대하여 처리할 경우 GPU의 최대 성능을 낼 수 있다. 워프 단위가 2일 때, 표 1과 2는 그림 6(a)와 (b)의 코드의 수행동작을 나타내었다. 따라서 워프 단위 조건문을 수행하지 못할 경우 호스트(CPU)상에서 처리하는 것이 유리하다.

```
if(tid % 2 ==0) a++;
else            a--;
```

(a)

```
if(tid < 2 ==0) a++;
else            a--;
```

(b)

Fig. 6 Warp conditional units; (a) Code A, (B) Code B

Table. 1 Perform a non-conditional basis Warp

Cycle	0	1	2	3
Thread 0	++	wait	wait	wait
Thread 1	wait	--	wait	wait
Thread 2	wait	wait	++	wait
Thread 3	wait	wait	wait	--

Table. 2 Perform conditional units Warp

Cycle	0	1	2	3
Thread 0	++	wait		
Thread 1	++	wait		
Thread 2	wait	--		
Thread 3	wait	--		

식 (4)는 dark channel 추출을 위한 식으로 $J^{dark}(x)$는 x좌표의 dark channel이고 $J^c(y)$는 입력 영상이다. 아래 식에서 *min*은 최소값을 추출하는 함수로 매 화소 값마다 조건문이 적용된다. 따라서 dark channel의 추출 함수는 호스트 상에서 하는 것이 유리하다.

$$J^{dark}(x) = \min_{c\in r,g,b}\{\min_{y\in\Omega(x)}(J^c(y))\} \quad (4)$$

따라서 해무 제거 알고리즘에서 조건문이 없는 전달량 정제의 bilateral 필터링만 GPU를 이용하여 수행하였다. 식 (5)는 bilateral 필터링의 식으로 $I^f(x)$는 x 좌표의 필터링 된 밝기이고 $f_r(\|I(x_i)-I(x)\|)$는 Gaussian 함수로 x_i와 x좌표의 밝기에 의한 커널이고 $g_s(\|x_i-x\|)$는 좌표에 의한 커널이다. W_p는 가중치 함수로 식 (6)으로 나타낼 수 있다.

$$I^f(x) = \frac{1}{W_p}\sum_{x_i\in\Omega} I(x_i) f_r(\|I(x_i)-I(x)\|) g_s(\|x_i-x\|) \quad (5)$$

$$W_p = \sum_{x_i\in\Omega} f_r(\|I(x_i)-I(x)\|) g_s(\|x_i-x\|) \quad (6)$$

식 (6)의 bilateral 필터링은 조건문 없는 컨벌루션의 형태이므로 GPU를 이용한 병렬 프로그램을 수행 시 큰

성능 향상을 얻을 수 있다.

3.1. 호스트 프로그램 최적화

시스템을 구성할 때 입력 영상(카메라로부터 입력)의 크기가 고정되어 있다면 매 프레임마다 메모리 할당(초기화 과정)을 수행하는 것은 필요 없는 과정이다. 따라서 이 과정은 시스템이 처음 시작될 때 한 번의 초기화 과정을 통하여 모든 메모리를 할당하고 시스템이 종료되는 시점에서 한 번의 해제를 하는 것이 시스템의 성능 저하를 줄일 수 있다[6].

또한 함수에 따라 1D 배열 혹은 2D 배열로 처리 하는 것이 유리할 경우가 있는데 이를 위해서 1D 배열(혹은 2D 배열)로 처리한 결과를 다음 함수를 위해 2D 배열(혹은 1D 배열)로 변형을 위해 변환을 위하여 메모리 복사를 할 경우 성능의 저하를 발생 시킨다.

그림 7(a)는 1D 배열의 메모리 할당한 경우 x, y 좌표에 해당하는 데이터를 접근할 때를 나타내었고, (b)는 1D배열을 2D 배열로 맵핑(mapping) 후에 *x, y* 좌표에 해당하는 데이터를 접근할 때를 나타내었다. 2D 배열로 접근 할 때 2번의 메모리 접근이 필요하지만 같은 메모리에 접근 한다. 1D 배열에서 2D 배열로 그리고 다시 1D 배열로 처리해야 할 부분이 생길 경우 메모리 복사를 이용하면 총 4번의 메모리 접근이 필요하지만 주소 맵핑을 이용하면 2번의 메모리 접근으로 처리가 가능하다.

일반적인 메모리 할당을 했을 때 영상과 같은 큰 메모리를 할당할 경우 가상 메모리 기술에 의하여 보조 기억장치(HDD)에 도움을 받아 해결을 한다[7]. 이는 컴퓨터 혹은 운영체제에 의해서 제어 되는데 현재 사용하는 메모리의 데이터를 DRAM에 올리고 사용하지 않는 메모리의 데이터를 보조 기억장치로 가상으로 큰 메모리 공간을 페이지 단위로 분할하여 제공한다. 현재 사용하지 않는 데이터의 물리적 주소는 보조 기억장치로 이동하는데 이는 GPU를 이용한 병렬 프로그래밍을 수행할 경우 두 번의 메모리 복사를 수행한다[8].

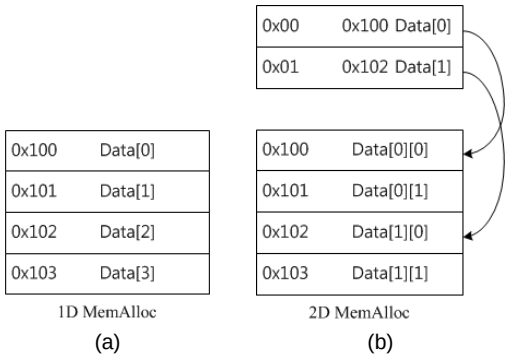

Fig. 7 1D, 2D arrangement address mapping; (a) 1D MemAlloc, (b) 2D MemAlloc

이러한 제약을 해결하기 위해 CUDA API의 고정된 메모리(pined memory)를 이용하여 성능을 향상 시킬 수 있다.

3.2. 병렬 프로그램 장치 최적화

영상 데이터 같은 큰 데이터를 위한 메모리 공간으로 메모리 복사할 때는 128-Byte 형태의 블록 형태로 전송을 수행하고, GPU내의 모든 쓰레드에서 접근이 가능하다. 반면 공유 메모리는 GPU 온칩(on-chip) 형태의 캐시 메모리에 할당이 되며 하나의 블록내의 쓰레드에서만 서로 공유하며 접근이 가능하다. 공유 메모리는 캐시 메모리 형태로 되어 있기 때문에 DRAM의 글로벌 메모리에 비해 빠른 접근이 가능하나 적은 량의 데이터만 저장할 수 있다(약 16KB). 따라서 공유 메모리는 자주 사용되는 공통의 파라미터를 저장하여 사용하는 것이 유리하다.

다음은 GPU를 이용한 병렬처리를 할 때의 순서이다. 아래 ② 과정은 디바이스가 수행되는 시간으로 호스트에서는 디바이스의 처리가 끝날 때까지 대기 상태를 유지하게 된다. 그림 8은 아래 과정을 수행 중일 때의 타이밍을 나타내었다.

① 호스트(CPU)에 메모리(DRAM) 병렬 처리할 데이터를 디바이스(GPU)의 메모리(GDDR)로 데이터 전송
② 디바이스에서 GDDR의 메모리를 이용한 병렬 처리
③ 디바이스의 메모리에서 처리된 데이터를 호스트 메모리로 전송

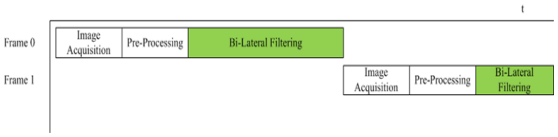

Fig. 8 Using universal GPU parallel programing timing chart

CUDA API에서는 비동기 수행을 지원하는데 스트림(stream)을 통하여 ①~③ 과정을 비동기로 수행하고, 호스트는 디바이스의 동작과는 독립적인 동작을 수행

할 수 있다. 독립적인 동작이 종료된 후에 스트림 동기화(synchronous stream)을 통하여 디바이스에서 수행된 결과를 획득 할 수 있다. 그림 9는 비동기 동작 수행을 프레임별로 수행하였을 때의 타이밍도를 나타내었다.

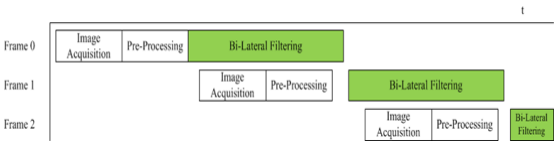

Fig. 9 Asynchronous operation timing chart when performed

3.3. GUI 구현

그림 10은 GPU 구현 시 OpenCV내의 카메라영상 획득 라이브러리를 이용하여 카메라로부터 실시간 영상을 입력 받고 구현한 C/C++ 프로그램을 DLL(Dynamic Linked Library)를 생성하여 National Instrument사의 LabVIEW를 통하여 GUI로 구현하였다.

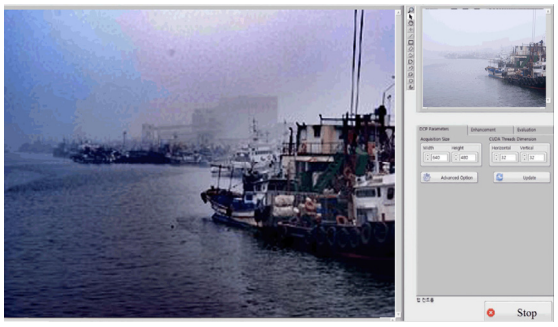

Fig. 10 GUI of dehazing technique base on GPU

Ⅳ. 선박 내 설치 및 검증

선박 내 실험은 본 기관에서 보유하고 있는 5,000톤급 중형 선박에서 진행하였다. 개발된 해무제거 시스템을 실제 운항 중인 선박에 탑재하여 정상동작 및 성능에 대한 테스트를 수행하였다.

성능실험과 함께 선교 내에 본 시스템을 설치할 적절한 위치를 선정하기 위해 현직 항해사들과 함께 실험을 진행하였다. 시스템을 설치하는 위치선정 시 항해사 혹은 조타수의 동선을 고려하여야 하는데, 이는 항해사의 시점에서 정확한 전방시계 정보가 획득되어야만 선박 전체의 운항 요원에게 신속하고 정확한 명령을 내릴 수 있기 때문이다. 적절한 위치선정을 위해 항해사와 조타수의 동선은 그림 11과 같다.

그림 12는 실제 항해사의 시점에서 바라본 해무제거 시스템이다. 현재는 선교 전면부의 선반 위에 시스템이 놓여 있는 상황이지만, 최대한 항해사의 시야를 방해하지 않도록 고려해야 한다.

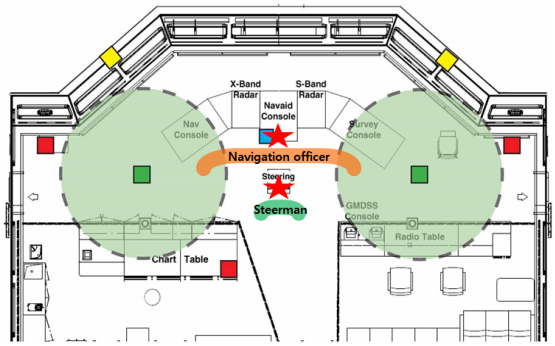

Fig. 11 Deck officer traffic for the positioning of the system on the navigation bridge

Fig. 12 Positioning of dehazing system

그림 13은 실험에 사용한 GPU의 특성을 나타내었다. PC는 Intel i7 CPU와 16GB의 호스트 메모리의 PC를 사용하였고 GPU는 NVidia의 GTX680을 이용하여 구현하였다. GTX 680의 계산 능력은 3.0으로 Fermi 구조이고 코어 클럭은 1.6GHz에 메모리의 밴드폭(band width)은 256bit이고 글로벌 메모리는 2GB이다. 또한 공유 메모리의 크기는 48KB이고 워프 크기는 32이다.

표 3은 실험 영상을 이용하여 각 단계별 평균 수행시간을 나타내었다. 여기서 CASE 1은 호스트 메모리 공유 및 고정 메모리를 사용한 경우, CASE 2는 GPU를 이용한 수행, CASE 3은 비동기 수행을 의미한다. dark channel을 찾기 위한 윈도우 크기는 9×9이고, bilateral

필터를 위한 커널의 크기는 15×15로 하였다. 반복정제를 위한 반복수는 3번으로 제한하였다. 호스트의 메모리 공유 및 고정 메모리 사용 시 약 5배의 성능 향상을 보였고, GPU를 이용한 병렬 처리를 하였을 때에는 약 250배의 성능 향상을 보였다. 또한 비동기적 수행을 통한 파이프라인화 하였을 때 약 730배의 성능 향상을 보였다.

```
*************************************************
  Device 0 - Name                : GeForce GTX 680

  ------ Core   Properties ------
  Computation Capablility        : 3.0
  Core Clock Freq.               : 1.06 GHz
  Number of Registers(SM/Block)  : 65536 / 65536 EA
  Support L1 Cache(Global/Local) : n / y
  L2 Cache Size                  : 512 KB
  Number of Multi-Processors     : 8 EA
  Number of Async. Engines       : 1 EA
  Support Concurrent Kernels     : y
  Support ECC                    : n
  Support Stream Priority        : n

  ------ Board  Properties ------
  PCIe Bus ID                    : 1
  PCIe Device ID                 : 0
  PCIe Domain ID                 : 0
  Multi-GPU on Board             : n
  Multi-GPU Group ID             : 0

  ------ Memory Properties ------
  Memory Clock Freq.             : 3.00 GHz
  Memory Bus Width               : 256 bits
  Global Memory Size             : 2048 MB
  Shared Memory Size(SM/Block)   : 48 / 48 KB
  Constant Memory Size           : 64 KB

  ------ Thread Properties ------
  Number of Max Threads(SM/Block): 2048 / 1024 EA
  Max Dimension of Blocks(x,y,z) : 1024, 1024, 64
  Max Dimension of Grids(x,y,z)  : 2147483647, 65535, 65535
  Warp Size in Threads           : 32 EA
*************************************************
```

Fig. 13 Performance GPU used in the experiment

Table. 3 Experimental Results (time[ms])

	Original	CASE 1	CASE 2	CASE 3
Resolution	320×240	320×240	320×240	320×240
Refine time	59,510.8	11,687.73	210.36	70.27
Average run time	59,533.7	11,710.67	245.97	81.95

Ⅴ. 결 론

본 논문에서는 선박에 설치된 카메라로부터 획득한 영상을 대상으로 DCP 기반의 해무제거 알고리즘을 이용하여 해무를 제거하였고, GPU를 이용하여 실시간 처리가 가능하도록 연산속도를 높였다.

연산속도를 높이기 위해 NVidia의 GTX 680 GPU를 사용하였고, 그 결과 약 250배의 성능을 향상시켰다. 비동기 수행을 사용 하였을 때는 약 730배의 성능 향상을 보였다. 따라서 GPU를 이용하여 실시간으로 처리가 가능함을 확인할 수 있었다.

본 논문에서 제안한 GPU 기반의 해무제거 기술은 향후 고속으로 이동하는 선박(고속정, 군함 등), 해양 항만 관제 시스템 등의 여러 분야까지 수용할 수 있을 것으로 보인다. 또한, 최근 스마트폰, 태블릿 PC 등과 같은 소형 모바일 장치들에 GPU가 내장되어 있어 보다 폭 넓은 응용분야에 적용될 수 있을 것으로 기대된다.

ACKNOWLEDGMENTS

This study was financially supported by Mokpo National Maritime University, 2015 year

REFERENCES

[1] National Metrics Framework. Maritime accident statistics [Internet]. Available: http://www.index.go.kr/potal/main/EachDtlPageDetail.do?idx_cd=1770#quick_02.

[2] S. Shwartz, E. Namer, and Y. Y. Schechner, “Blind haze separation,” in *Proceeding of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1984-1991, 2006.

[3] S. G. Narasimhan, S. K. Nayar, “Contrast restoration of weather degraded images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 6, pp. 713-724, June 2003.

[4] J. Tarel and N. Hautiere, “Fast Visibility Restoration from a Single Color or Grey Level Image,” *in Proceeding of the IEEE 12th International Conference on Computer Vision*, Kyoto, pp. 2201-2208, 2009.

[5] C. Tomasi, R. Manduchi, “Bilateral filtering for gray and color images,” *in Proceeding of the IEEE 6th International Conference on Computer Vision*, Bombay, pp. 839-846, 1998.

[6] D. Chen, W. Chen, and W. Zheng, “CUDA-Zero: a

framework for porting shared memory GPU applications to multi-GPUs," *Science China Information Sciences*, vol. 55, no. 3, pp. 663-676, Feb. 2012.

[7] K. Shirahata, H. Sato, and S. Matsuoka, "Out-of-core GPU memory management for Map Reduce- based large-scale graph processing," *in Proceeding of the 2014 IEEE International Conference on Cluster Computing*, Madrid, pp. 221-229, 2014.

[8] Y. Yang, P. Xiang, and H. Zhou, "A GPGPU compiler for memory optimization and parallelism management," *in ACM SIGPLAN Notices*, Toronto, vol. 45. no. 6, pp. 86-97, 2010.

최운식(Woon-Sik Choi)

2016년 : 목포해양대학교 일반대학원 (공학석사)
※관심분야 : 영상처리, 실감미디어

이윤혁(Yoon-Hyuk Lee)

2014년 ~ 현재 : 광운대학교 일반대학원 박사과정
ORCID: orcid.org/0000-0001-7184-6896
※관심분야 : 디지털 홀로그램, SoC 설계

서영호(Young-Ho Seo)

2004년 : 광운대학교 일반대학원 졸업(공학박사)
2003년 ~ 2004년 : 한국전기연구원 연구원
2005년 ~ 2008년 : 한성대학교 조교수
2008년 ~ 현재 : 광운대학교 인제니움학부대학 교수
ORCID : orcid.org/0000-0003-1046-395X
※관심분야 : 실감미디어, 2D/3D영상 신호처리, 디지털 홀로그램

최현준(Hyun-Jun Choi)

2009년 2월 : 광운대학교 전자재료공학과 공학박사
2010년 3월 ~ 2011년 8월 : 안양대학교 정보통신공학과 교수
2015년 1월 ~ 2016년 2월 : 네브라스카주립대(UNO) 방문교수
2011년 8월 ~ 현재 : 목포해양대학교 전자공학과 교수
※관심분야 : 하이브리드(디지털/광) 영상신호처리, 하드웨어(FPGA/ASIC) 설계, 암호학