한국컴퓨터정보학회 동계학술대회 논문집 제19권 제1호 (2011. 1)

GPGPU를 이용한 눈 영역 검출 기법

박영재°, 김계영*
°*숭실대학교 컴퓨터학과
e-mail: {p1y1j1, gykim11}@hcu.ac.kr

Method of extract eye zone using GPGPU

Young-Jae Park°, Gye-Young Kim*
°*Dept. of Computer, Soongsil University

● 요 약 ●

본 논문에서는 GPGPU를 이용한 눈 영역 검출 기법을 제안한다. 영상 전체의 평균과 분산을 기반으로 하여 각 마스크의 평균과 분산값을 비교는 비교적 간단한 알고리즘을 이용하여 눈 영역을 검출한다. 정확도의 경우 명암값의 대비를 이용한 기존의 방법과 비슷한 수준을 보였다. 하지만 연산속도의 경우 병렬처리 구간을 늘려 GPGPU를 사용한 제안된 방법이 우수한 성능을 보였다.

키워드: 눈 영역(Eye-zone), GPGPU(GPGPU), CUDA(CUDA)

I. 서 론

처리하고 출력해야할 데이터들이 점점 거대해지는 요즘, 처리된 데이터의 질을 유지하면서 실행시간을 줄이거나, 프로그램의 실행시간을 유지하면서 프로그램 결과의 질을 향상시키기 위하여 보다 빠른 컴퓨터가 필요하다. 그러나 CPU 물리적 발전의 한계(단일코어 프로세서의 클럭 속도를 높이는 일)와 프로세서의 실행 최적화에도 역시 한계가 존재한다. 따라서 최근에는 멀티코어 프로세서를 이용한 병렬처리(Pararllel Computing)로 해답을 찾고 있다. GPU는 어느 PC에나 존재하는 훌륭한 멀티코어 프로세서로 현재 복잡하고 연산 집약적인 문제를 풀어나가기 위해서 사용되는 경우가 늘어나고 있다.

GPU는 원래 그래픽 처리를 위한 산술 연산에 특화된 프로세스이다. 하지만 하드웨어가 발전함에 따라, GPU의 성능은 비약적으로 발전하게 되었다. 최근 이러한 성능을 가진 GPU가 대부분의 시간을 아무작업도 하지 않고 있다는 것에 착안하여 작업을 GPU에 분배시키는 방안을 찾게 되었고 이를 배경으로 등장한 것이 GPGPU(General-purpose computing on graphic processing unit)이다. 현재 컴퓨팅은 CPU중심의 중앙 프로세싱(Central Processing)에서 CPU와 GPU의 공동(Co-processing)으로 진화하고 있다. 이러한 새로운 컴퓨팅 패러다임을 실현하기 위해서는 GPGPU를 이용한 병렬 컴퓨팅 아키텍처의 개발이 필요하다. GPGPU의 프로그래밍에 대한 접근성을 높인 C형태의 언어로 CUDA와 OpenCL이 각광을 받고 있다.

OpenCL은 새로운 이종(Heterogeneous) 컴퓨팅 환경으로 개발자들이 GPU의 대량 병렬 컴퓨팅 능력을 이용하여 개선된 컴퓨팅 어플리케이션을 개발할 수 있게 하는 아키텍처로 애플사에서 GPU 컴퓨팅을 위한 크로스 플랫폼(Cross Platform) 환경을 구축하고자, 2008년 여름 nVidia와의 파트너쉽 하에 OpenCL을 크로너스(Khronos)에 제안하였다.[1]

CUDA(Computing Unified Device Architecture)는 nVidia 주도의 범용 병렬 컴퓨팅 아키텍처로써 GPU가 복잡한 컴퓨터 문제를 해결 하도록 한다. 현재 C언어를 사용하여 CUDA 아키텍처를 프로그램하고 CUDA 지원 프로세서에서 적용할 수 있으며 Fortran 및 C++등의 다른 언어도 지원할 예정이다.[2]

2장에서는 기존의 눈 영역 검출기법과 새로 제안하는 간단한 눈 영역 검출 기법을 설명하고 이것을 CUDA를 이용하여 가속화하는 방법에 대해 설명한다. 3장에서는 기존의 방법과 제안된 방법을 이용한 성능을 비교하고 4장에서는 결론을 내린다.

II. 본 론

1. 눈 영역 검출 기법

1.1 Eye-Map

Eye-Map은 Rein-Lien Hsu[3]가 제안한 방법으로 눈 영역을 검출하는 방법이다. Eye Map은 두 개의 연산을 통하여 만들어 지는데 하나는 색상값을 이용하는 방법(EyeMapC)이다. 이것은 YCbCr 색상 공간에서 산출하게 된다. EyeMapC는 (식 1) 과 같이 구할 수 있다.

$$EyeMapC = \frac{1}{3}\{(C_b^2) + (\widetilde{C_r})^2 + (C_b / C_r)\} \quad (1)$$

$C_b^2, (\widetilde{C_r}), C_b / C_r$은 0에서 255사이의 값으로 정규화되어 있으며, C_b, C_r은 각각 YCbCr의 Cb와 Cr값이며 $(\widetilde{C_r})$은 역변환된 Cr값이다. 피부영역의 색상은 대체적으로 Cr의 값이 Cb의 값보다 높다는 점에서 착안한 방법이다.

또 하나의 방법으로 명암의 대비를 이용하여 구하는 방법(EyeMapL)이 있는데 이것은 (식 2) 와 같은 방법으로 구할 수 있다.

$$EyeMapL = \frac{Y(x,y) \oplus g_\sigma(x,y)}{Y(x,y) \ominus g_\sigma(x,y) + 1} \quad (2)$$

흑백영상을 이용하여 팽창과 침식을 하여 명암대비를 이용한 EyeMapL을 구한다. $\oplus$는 팽창, $\ominus$는 침식을 하는것이고, g(x, y)는 흑백영상의 명암값이다. 구해진 두 개의 값을 이용하여 EyeMap을 (식 3)과 같이 만든다.

$$EyeMap = (EyeMapC)\mathrm{and}(EyeMapL) \quad (3)$$

이렇게 구해진 EyeMap을 임계값을 사용하여 이진화 하여 눈의 후보 영역을 추출한다. 그림 2는 EyeMap을 사용하여 구한 실험결과인데, 눈 영역만 추출되는 것은 아니다. 눈 이외에 다른 결과가 섞여 나올 수도 있는데 이 문제는 추후 해결해야할 과제이다.

1.2 통계학적 방법을 이용한 눈 영역 검출 기법

일반적으로 눈 영역의 경우 검은자위와 흰자위가 교차되어 있어 명암값의 분포가 넓다.[4] 본 논문에서는 이러한 특징을 이용하여 눈 영역을 검출하는 기법을 제안한다.

먼저 영상 전체에서 Hsu가 제안한 방법을 이용하여 피부영역을 검출한다. 피부영역에 해당하는 픽셀을 $S[1,2,3...k]$라고 할때 피부영역 내에 명암값과 분산값을 다음 식을 이용하여 구한다.

$$Luma_{avg} = \sum_{i=0}^{k} S_i \quad (4)$$

$$Dev_{avg} = \sum_{i=0}^{k} |Luma_{avg} - S_i| \quad (5)$$

픽셀별로 3x3 ~ 9x9 마스크를 이용하여 각각의 마스크 별로 분산을 구하여 각 픽셀의 분산값으로 설정한다. 각 픽셀의 대표 분산값을 모두 구한 다음 0~255사이로 정규화 하고 적당한 임계값을 이용하여 눈 영역을 검출한다. 본 논문에서는 임계값으로 80을 사용하였다.

2. 병렬 프로그래밍

2.1 CUDA

CUDA를 이용하여 병렬처리를 하기 위한 조건으로는 3가지정도가 있다. 첫째, 실수연산이 많아야 한다. 아키텍처마다 다르지만 Fermi기준으로(이후 아키텍처는 Fermi 기준임) CUDA코어하나에 부동소수점을 처리하는 유닛이 1개씩 배정되어 있다. 32개의 CUDA 코어당 4개씩 특별한 부동소수점 처리 유닛(SFU)이 배정되어있다. 따라서 실수연산이 많으면 많을 수록 성능향상을 기대할 수 있다. 둘째, Host 메모리 접근이 적어야 한다. GPGPU 프로그래밍에서 CPU에 해당하는 메인메모리를 Host, GPU에 해당하는 GPU 메모리를 Device라고 한다. 다시 말해서 메인메모리의 접근이 많으면 많을수록 데이터 전송량이 많아지며 이것은 프로그램 속도 저하의 원인이 된다. 셋째, 분기문이 적어야 한다. 하나의 SM(Streaming Multiprocessor)에는 32개의 CUDA 코어가 들어가 있다. SM을 하나의 SIMD(Single Instruction Multiple Data)로 보기 때문에 32개의 쓰레드는 동시에 한 개의 명령어만 수행할 수 있다. 따라서 명령어수행중에 분기문을 만나게 되면 분기문 이후의 여러 가지 명령어를 하나씩 순차적으로 처리하게 된다. 따라서 병렬화 구간이 줄어들고 연산속도 또한 저하된다.[5]

III. 실험결과

본 실험을 위해 사용한 컴퓨터는 Intel i7 930 2.80Ghz CPU와 8.00Gb RAM을 사용하였고 VGA 카드는 nVidia Geforce GTX580 SLI, 운영체제는 Windows 7 64bit를 사용하였다. 사용언어는 Microsoft Visual Studio 2005, CUDA 3.1을 사용하였다. 사용 영상은 인터넷에 배포되어 있는 저작권에 문제가 없는(Google 검색 라이센스 제한 없음 옵션 사용) 영상을 모아 자체적으로 구축한 데이터베이스를 사용하였다. 이 데이터베이스에는 유해영상 1200장, 인물영상 600장, 일반영상 700으로 구성되어 있다.

표 1. 눈 영역 검출 정확도
Table 1. Accuracy ratio of Extracting Eye-Zone

알고리즘	정확도	
	인물영상	일반영상
Eye-Map	94%	92%
제안된 방법	93%	91%

표 1은 Eye-Map과 제안된 방법의 정확도 측정 결과이다. 인물영상은 사람이 있는 영상으로 영상내에 눈이 존재하는 영상이며, 일반영상은 사람이 없는 영상으로 영상내에 눈이 존재하지 않는 영상이다. 정확도 측면에서는 Eye-Map과 제안된 방법간의 성능차이는 크지 않았으나 Eye-Map의 정확도가 약간 높은 결과를 얻을 수 있었다.

표 2. 눈 영역 검출 연산시간
Table 2. Computing Time of Extracting Eye-Zone
(단위 : ms)

알고리즘	연산시간	
	CPU	CPU+GPU
Eye-Map	667	581
제안된 방법	1123	221

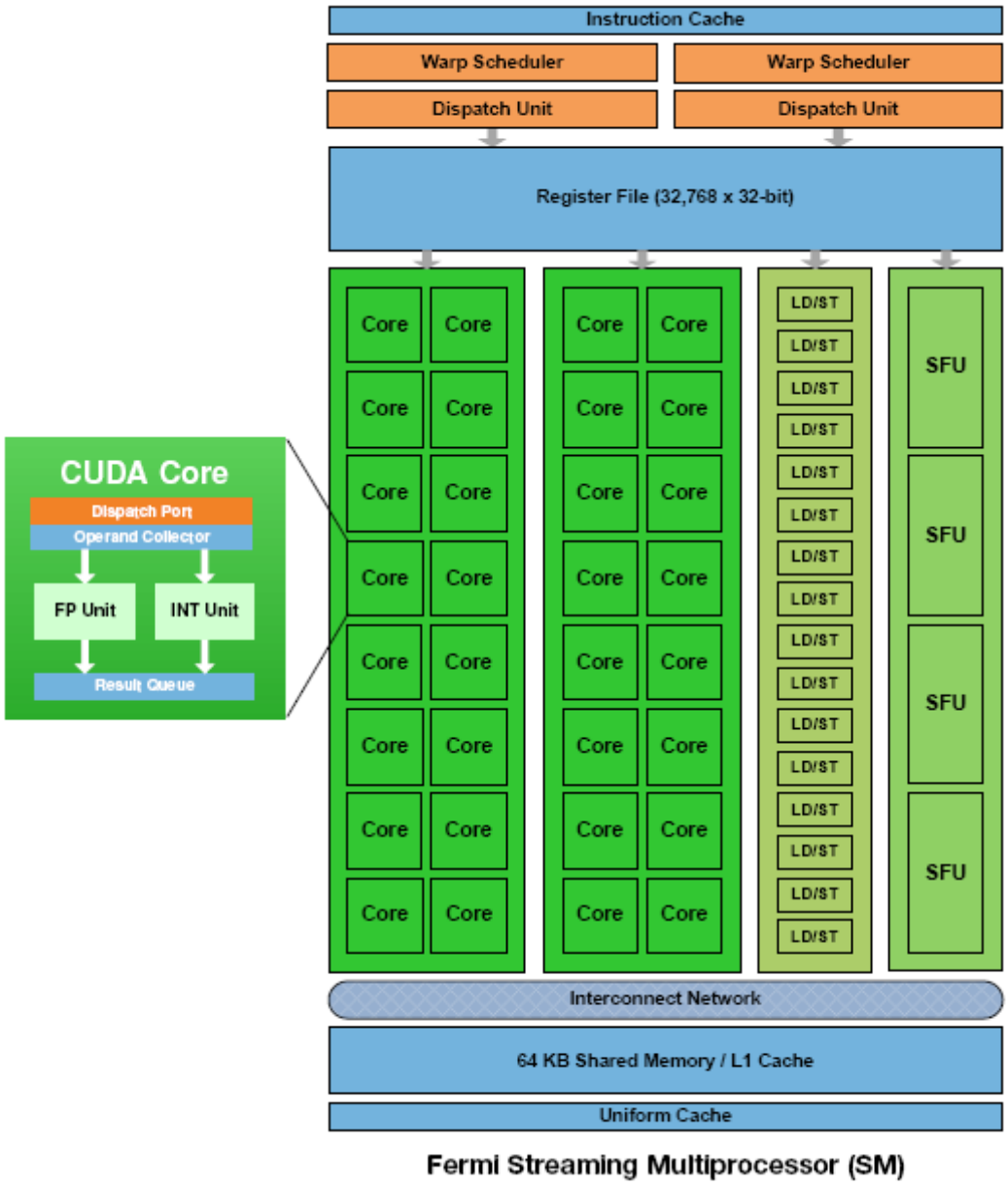

그림 1. Fermi의 Streaming Multiprocessor(SM) 구조
Fig. 1. Streaming Multiprocessor(SM) Architecture of Fermi

표 2는 Eye-Map과 제안된 방법의 눈 영역 검출 연산시간을 측정한 결과이다. Eye-Map의 경우 병렬화 를 고려하지 않고 작성한 알고리즘이라 병렬화할 수 있는 구간이 거의 존재 하지 않는다. 그 결과 GPU를 사용하였을 때 속도향상이 거의 이루어 지지 않았다. 하지만 제안된 방법의 경우 CPU에서는 기존의 방법에 비해 2배에 가까운 시간이 걸렸지만 GPU를 사용하였을 경우 제안된 방법의 1/2수준의 시간에 연산이 종료되었다.

Ⅳ. 결 론

본 논문에서는 GPGPU를 이용한 눈 영역 검출기법을 제안하였다. 제안된 방법은 GPGPU사용을 위하여 알고리즘상 다소 비효율적이나 병렬화 구간을 최대로 할 수 있게 설계하였다. 그 결과 만족할만한 결과를 얻을 수 있었다. 이처럼 연산량이 너무 많아 관심을 받지 못했던 알고리즘을 병렬처리가 가능한 GPGPU를 사용하여 처리함으로써 알고리즘 개발에 있어서 큰 문제중 하나인 연산량에 대한 제약을 상당부분 완화시킬 수 있을 것으로 기대된다.

참고문헌

[1] Khronos Group, “Khronos Launches Heterogeneous Computing Initiative”, http://www.khronos.org, 16 Jun 2008

[2] Jason Sanders and NVIDIA Corporation, “Introduction to CUDA”, NVIDIA Theater Presentations, SIGGRAPH 2010

[3] Rein-Lien Hsu, Mohamed Abdel-Mottaleb, Anil K. Jain, “Face Detection in Color Images”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 5, pp. 696~706, May 2002

[4] Young-Jae Park, Seok-Woo Jang and Gye-Young Kim, “A Study on Extraction of Skin Region and Lip Using Skin Color of Eye Zone”, Journal of the Korea Society of Computer and Information, vol.14, no.4, pp. 19~30, 2009

[5] “NVIDIA's Next Generation CUDA Compute Architecture : Fermi”, Technical Paper, nVidia Corporation, 2009