GPGPU 기반의 깊이 정보를 이용한 고속 얼굴 추적에 대한 연구

김우열* · 서영호** · 김동욱***

A Study on High Speed Face Tracking using the GPGPU-based Depth Information

Woo-youl Kim* · Young-ho Seo** · Dong-wook Kim***

본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2010-0012898)

요 약

본 논문에서는 얼굴을 검출하고 GPU 기반으로 얼굴을 고속으로 추적하는 알고리즘을 제안하였다. 얼굴 검출에서는 깊이영상과 RGB영상을 사용하고, 기존의 방법인 Adaboost을 이용하지만 움직임 영역과 피부색 영역을 이용하여 Adaboost의 입력영상을 제한하여 얼굴을 검출하였다. 얼굴 검출과는 다르게 얼굴 추적은 깊이 정보만을 사용하였다. 기본적으로 얼굴 추적에서는 템플릿과 매칭 된 블록을 찾는 템플릿 매칭 방법을 사용하였다. 또한 고속으로 얼굴을 추적하기 위해서 GPU를 이용하여 템플릿 매칭을 병렬하여 연산하였다. 실험결과 CPU와 GPU을 비교 하였을 때 GPU 수행속도가 최대 49배까지 향상되는 것을 확인하였다.

ABSTRACT

In this paper, we propose an algorithm to detect and track the human face with a GPU-based high speed. Basically the detection algorithm uses the existing Adaboost algorithm but the search area is dramatically reduced by detecting movement and skin color region. Differently from detection process, tracking algorithm uses only depth information. Basically it uses a template matching method such that it searches a matched block to the template. Also, In order to fast track the face, it was computed in parallel using GPU about the template matching. Experimental results show that the GPU speed when compared with the CPU has been increased to up to 49 times.

키워드

얼굴검출, 얼굴추적, 깊이정보, 범용그래픽처리장치, 템플릿 매칭

Key word

face detection, face tracking, depth information, GPGPU, template matching

* 종신회원 : 광운대학교 전자재료공학과
** 종신회원 : 광운대학교 교양학부
*** 종신회원 : 광운대학교 전자재료공학과(교신저자, dwkim@kw.ac.kr)

접수일자 : 2013. 01. 17
심사완료일자 : 2013. 03. 14

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.5.1119

Ⅰ. 서 론

인간생체의 일부를 검출하고 추적하는 방법은 컴퓨터 비전분야를 비롯한 다양한 분야에서 오래전부터 연구되어 왔으며, 보안시스템, 화상회의, 로봇 비전, HCI(Human-Computer Interface)에 의한 대화형 시스템, 스마트 홈 등에 널리 사용되고 있다[1][2]. 이 중 얼굴에 대한 연구가 가장 활발히 연구되어 왔으며, 그 목적은 빠르고 정확한 검출과 추적이었다.

기 제안된 얼굴검출 방법은 크게 지식-기반 방법, 특징-기반 방법, 템플릿 매칭(Template Matching) 방법, 외형-기반 방법(Appearance-based Methods)으로 분류할 수 있다[3]. 지식-기반 방법은 사람의 얼굴을 구성하는 눈, 코, 입 등의 기하학적인 특성을 파악하여 얼굴을 검출하는 방법이다[4]. 특징-기반 방법은 얼굴의 특징 성분인 얼굴요소, 질감 정보, 피부색, 또는 이들을 복합적으로 사용하여 얼굴을 검출한다. 템플릿 매칭 방법은 수동적으로 미리 대상이 되는 모든 얼굴에 대한 표준 얼굴패턴을 만들고 이를 입력영상과 비교하여 얼굴을 검출하는 방법이다[5]. 외형-기반 방법은 학습영상 집합을 입력받아 훈련과정을 통해 학습된 모델을 이용하여 얼굴을 검출하는 방법이다.

얼굴을 추적하는 방법에는 기존에 2차원 영상을 사용하던 방법과는 달리 3차원적 정보의 깊이지도(Depth-map)를 이용하는 방법들도 연구가 진행되고 있다. 또한 최근에는 SR4000[6] 같은 깊이카메라 또는 Microsoft사의 Kinect[7]을 이용하여 깊이정보를 실시간으로 획득하여 얼굴추적에 직접 사용하는 연구도 진행되고 있다. [16]에서는 Kinect와 SR4000의 깊이정보를 이용하여 얼굴을 추적하고, 손을 인식하는 방법도 제안하였다. 또한 [17][18]에서는 얼굴이 아니라 사람의 몸 전체를 추적하는 방법도 제안되었는데, [17]은 깊이영상을 이용하여 사람을 추출하고, 그 주변을 탐색영역으로 설정하여 YUV 영상으로 움직임을 추적하는 방법을 제안하였다. [18]는 줌(Zoom) 움직임에 초점을 맞춘 방법으로 깊이영상으로부터 대상 물체의 카메라에서의 거리 차이를 구하고, 그 주변을 탐색하여 물체를 추적하는 방법도 제안하였다.

얼굴 추적은 움직이는 사람의 얼굴을 검출하여 이동경로를 추적하는 것으로 실시간 환경에서의 빠른 수행속도를 가져야 만 한다. 하지만 얼굴을 추적하기 위해서는 방대한 양의 데이터를 연산해야하기 때문에 연산의 횟수를 줄이는 방법이 연구되어지고 있다[8][9][10]. [8]은 GPU기반의 실시간 템플릿 추적을 위한 효율적인 파티클 필터링을 구현하였으며, [9]에서는 높은 해상도에서 동시에 여러 명을 추적할 때, 고속으로 연산을 하기 위해서 스트림 처리를 이용하여 실시간 처리가 가능하도록 하였다. [10]은 GPU를 이용하여 픽셀 레벨의 분할과 정합 기반의 실시간 객체 추적을 위한 파티클 필터를 제안하였다.

본 논문에서는 얼굴을 추적하는 연산과정이 중복된다는 점을 착안하여, 이를 GPGPU(General-Purpose Graphic Processing Unit)[11]을 이용하여 병렬처리를 통해 고속으로 얼굴을 추적하는 방법을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 제안하는 알고리즘에 대하여 설명하고, 3장에서는 병렬처리를 이용한 얼굴 추적 방법을 설명하고, 4장에서는 실험결과를 기술하고, 5장에서 결론을 맺는다.

Ⅱ. 얼굴검출 및 추적 알고리즘

2.1. 얼굴 검출 방법

얼굴검출에서는 RGB영상과 깊이영상을 사용한다. 그림 1은 제안하는 얼굴검출 알고리즘을 블록도로 나타내었다.

본 논문에서 제안하는 얼굴 검출 방법은 움직임 검출과 피부색 영역에 대한 두 가지 정보를 가지고, 얼굴 검출이 가능한 네 가지 경우로 분류하였다. 얼굴검출은 초기에 한 번만 수행하며, 얼굴이 검출된 다음 프레임부터는 추적과정만 수행된다. 그러나 장면이 바뀌거나 추적과정에서 템플릿 매칭이 이루어지지 않은 경우 다시 검출과정을 수행한다. 본 논문의 얼굴검출 방법은 기본적으로 Adaboost 알고리즘[19]을 사용한다. 그러나 본 논문에서는 Adaboost 알고리즘에 적용할 영상의 크기를 움직임 검출과 피부색 영역을 이용하여 상당히 많은 부분을 줄였다.

2.1.1. 움직임 영역 검출

먼저, 입력된 깊이영상으로 움직임 영역을 검출한다. 이것은 이전 프레임과 현재 프레임간의 차이를 구하여 검출한다. 일단 차영상이 구해지면 그 차영상을 수평방

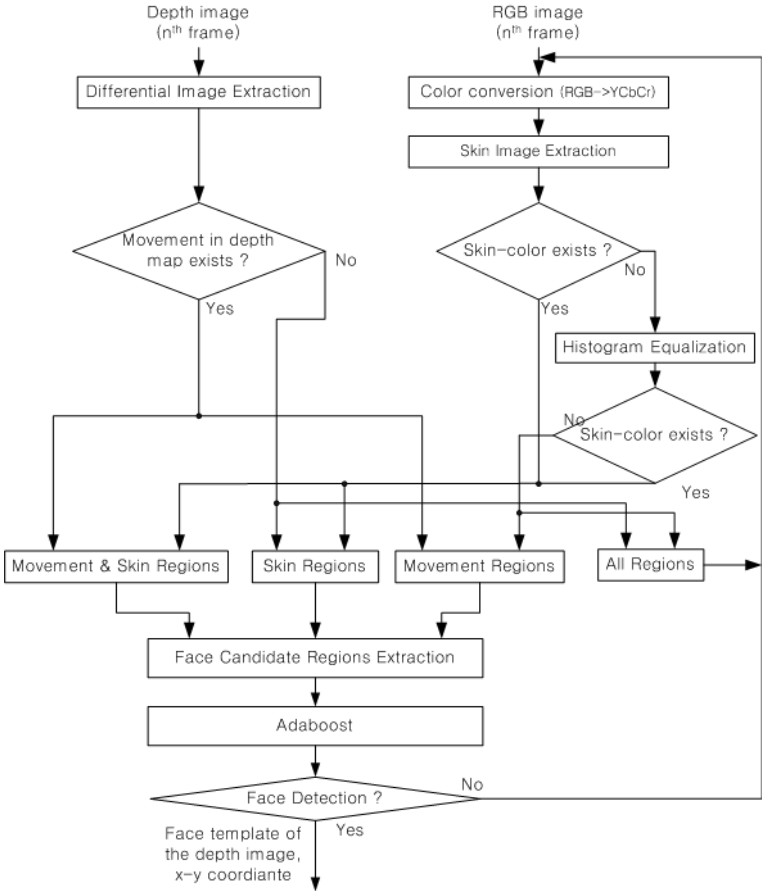

그림 1. 제안하는 얼굴 검출 알고리즘
Fig. 1 Proposed face detection algorithm

향과 수직방향으로 각각 누적덧셈을 수행한다.

그 결과는 하나의 행과 하나의 열로 나타나며, 각 화소의 값이 0이 아닌 화소들을 행과 열로 각각 확장하여 두 확장 영역의 교집합을 구한다. 그림 2는 움직임 영역을 보여준다.

2.1.2. 피부색 영역 검출

일반적으로 얼굴색은 조명에 크게 의존하므로, 본 논문에서는 그 의존도를 낮추기 위해 영상포맷을 RGB에서 YCbCr로 바꾸어 사용하였다. 이 중 Y성분은 조명에 가장 민감하므로 Cb와 Cr성분만 사용한다. 본 논문에서 피부색으로 사용한 색의 범위는 [12]의 피부색 참조맵이며, 식 (1)와 같다.

만약 식 (1)에서 $B(x,y)=1$이면, 해당하는 픽셀은 피부로 간주하였다. 그림 3은 식(1)을 적용하여 검출 된 피부색 영역을 보여준다.

$$B(x,y)=\begin{cases}1, \text{if}(77\leq C_b\leq 127)\cap(133\leq C_r\leq 173)\\ 0, Otherwise\end{cases} \quad (1)$$

조명변화로 인해 피부색을 검출하지 못하는 경우가 있다면, 이때는 영상에 대하여 히스토그램 평활화(Histogram Equalize)를 적용하여 피부색을 검출하였다. 히스토그램 평활화는 식 (2)을 이용하여 수행하였다.

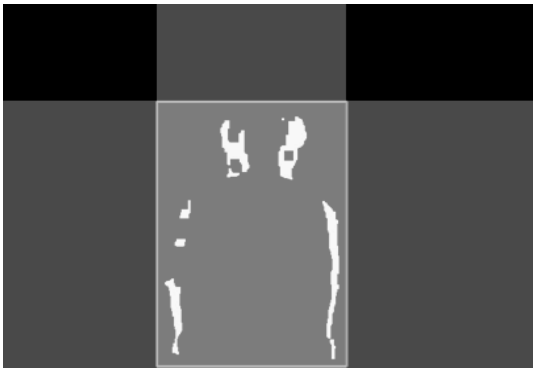

그림 2. 깊이영상에서의 움직임 영역 검출
Fig. 2 Movement region in the depth image

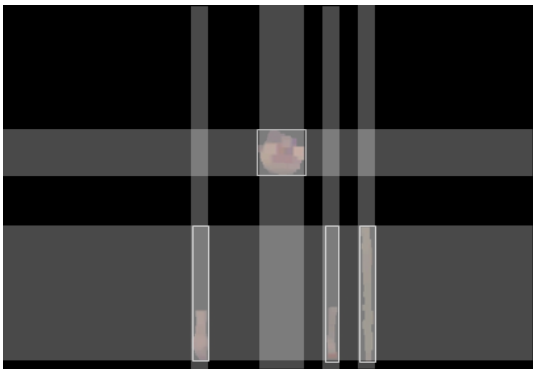

그림 3. 텍스처 영상에서의 피부색 검출
Fig. 3 Skin region in the textured image

$h(i)$은 픽셀 i에 대하여 정규화 된 값, G은 영상의 최대 밝기 값, N은 영상크기(전체 픽셀 수) 그리고 $H(i)$은 픽셀의 누적 빈도수에 해당한다.

$$h(i) = \frac{G}{N} H(i) \tag{2}$$

2.1.3. 얼굴 검출과 템플릿 정의

움직임 영역과 피부색 영역을 이용하여 얼굴 검출을 위한 탐색범위를 정의한 후에, Adaboost 알고리즘에 적용하였다. Adaboost 알고리즘에 대한 결과 (x,y) 좌표와 얼굴에 대한 텍스처 영상이 검출되었다. 그러나 얼굴추적을 위하여, 텍스처 영상에 해당하는 깊이 영상을 템플릿으로 정하였으며, 또한 (x,y) 좌표는 템플릿에 대한 위치 정보를 위해 추적과정으로 함께 보내졌다.

2.2. 얼굴추적 방법

검출 과정 후에 첫 번째 프레임에서, 템플릿은 얼굴추적을 위해 사용되었다. 제안하는 얼굴추적 알고리즘에 대한 블록도는 그림 4에 나타내었다. 검출과정과 다르게 추적과정은 조명변화에 RGB영상보다 강한 깊이 영상만을 이용하였다.

얼굴추적은 얼굴검출 과정에서 생성되거나 그 전 프레임에서 갱신(update)된 템플릿이 현재 영상에서 매칭이 되는 지점을 찾는 과정이다. 그러나 영상의 전체 영역을 대상으로 템플릿 매칭을 수행하면 과다한 시간이 소요된다. 따라서 본 논문에서는 탐색영역을 최소화하는 방법을 제안하며, 얼굴이 상하좌우 뿐만 아니라 앞뒤로 움직이는 경우를 포함하도록 하였다.

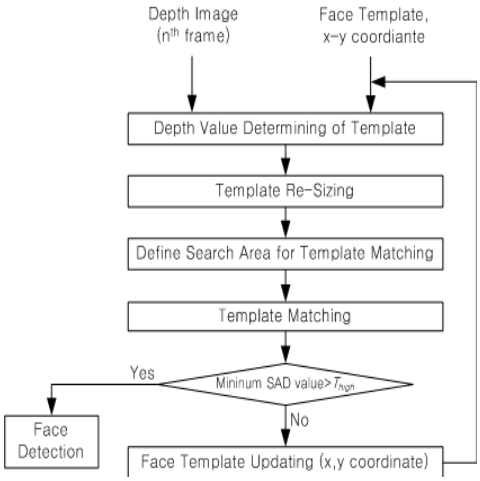

그림 4. 제안한 얼굴 추적 알고리즘
Fig. 4 Proposed face tracking algorithm

2.2.1. 템플릿의 깊이 값

템플릿의 깊이 값은 템플릿의 크기와 탐색범위를 결정하기 위해서 필요하다. 그래서 그 전 프레임에서 만들어진 템플릿을 사용하여 템플릿에 대한 깊이 값을 측정하였다. 측정방법은 템플릿을 각각 $m \times n$ 서브블록으로 나누고, 각 서브블록 (i,j)의 깊이 값 평균$(a_{i,j})$을 계산하여 그 중 최고치를 템플릿(z_{temp})의 깊이 값으로 선택한다. 이것을 식 (3)에 나타내었다.

$$z_{temp} = \max(a_{1,1}^{temp},...,a_{n,m}^{temp}) \quad (3)$$

여기서 아래첨자 $temp$은 템플릿임을 표시하며, 위첨자는 서브블록의 위치를 표시한다. 그리고 $a_{n,m}^{temp}$은 서브블록의 깊이 값을 나타낸다.

2.2.2. 템플릿 크기 조절

추적과정에서 얼굴이 전후로 움직일 때 깊이가 변화하기 때문에 현재의 템플릿을 사용할 경우 정확한 추적을 할 수 없다.

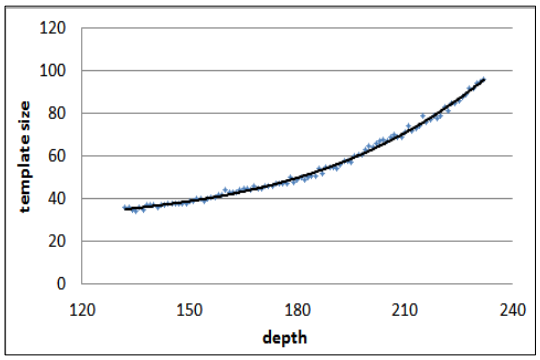

그림 5. 템플릿의 크기와 깊이의 관계
Fig. 5 The relationship template size and depth

따라서 깊이가 변화함에 따라 템플릿의 크기도 변화시켜야 한다. 그림 5는 깊이에 따른 템플릿의 크기를 측정한 결과이다. 물체가 깊이 z에 있을 때의 템플릿의 크기를 s라 하면 그림 5의 결과를 fitting하여 식 (4)과 같은 관계를 얻을 수 있다.

$$s = 0.00004z^3 - 0.0165z^2 + 2.3613z - 84.032 \quad (4)$$

2.2.3. 탐색범위의 결정

얼굴추적을 위해 탐색하여야 하는 영역은 얼굴의 움직임 속도와 관련이 있다. 본 논문에서는 추적할 얼굴이 전방의 영상을 시청하고 있다고 가정한다. 따라서 얼굴은 거의 정면을 바라보고 있으며, 영상을 시청하면서 움직이는 상황을 가정하였다. 얼굴의 움직임에 대해서는 시청하면서 움직일 수 있는 최대의 움직임 속도를 고려하였다.

물체의 깊이에 따라 움직임 속도, 즉 움직임 양이 달라지므로 깊이에 따른 움직임 양을 일반적으로 표현하기 위해서 본 논문에서는 현재의 템플릿에 대비한 상대적인 크기로 탐색영역을 정의한다.

본 논문에서는 먼저 얼굴의 최대 움직임 속도를 실험을 통하여 전후, 좌우, 상하방향에 대해 측정하였으며, 이를 두 프레임 간의 거리를 환산하여 탐색영역을 정의하였다. 표 1은 전후, 좌우, 상하방향에 대하여 실험을 통해 측정 한 깊이에 따른 움직임 양을 보여준다. 표 1을 이용하여 깊이에 따라 탐색영역을 정의하였다. 표 1에서 움직임에 대한 양은 픽셀에 대한 변화량으로 나타내었고, t-1과 t은 이전 프레임과 현재 프레임을 나타낸다.

표 1. 깊이에 따른 움직임에 대한 변화량
Table. 1 Variation for movement according to depth

Left&Right Movement			Up&Down Movement		
Depth		Maximum amount of Movement	Depth		Maximum amount of Movement
132		8	132		8
148		8	148		10
167		10	172		10
188		12	196		14
212		16	220		18
228		20	234		24
Forth Movement			Back Movement		
Depth		Maximum amount of Movement	Depth		Maximum amount of Movement
t-1	t		t-1	t	
149	155	4	151	148	8
160	164	6	169	164	8
180	188	6	188	180	8
194	196	8	201	196	10
214	219	10	220	215	10
236	243	14	236	230	14

위와 같이 결정한 탐색영역을 그림 6에 나타내었는데, 좌우상하 움직임은 각 방향으로 28% 범위 내에 각각 있었다.

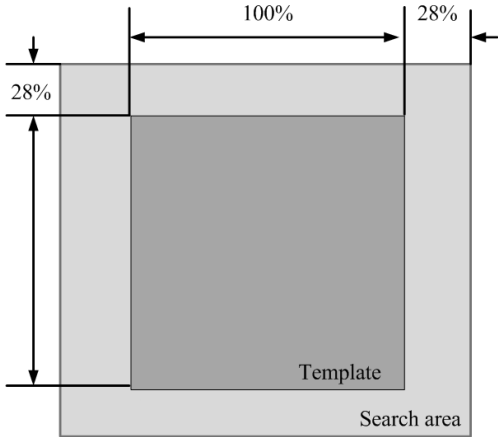

그림 6. 탐색영역
Fig. 6 Search range

2.2.4. 템플릿 매칭

얼굴추적의 다음 단계는 앞 절에서 재조정된 템플릿을 사용, 재조정된 탐색영역을 조사하여 매칭이 되는 점을 찾는 것이다. 하지만 RGB영상을 가지고 템플릿 매칭을 했을 경우, 조명변화로 인해 다음 프레임에서 얼굴을 찾지 못하고 다른 부분을 찾는 경우가 발생한다. 그래서 본 논문에서는 RGB영상보다 조명변화에 강한 깊이정보를 이용하여 템플릿 매칭을 하였다.

템플릿 매칭을 할 경우에 기본적으로는 탐색영역 전체를 탐색한다. 이 경우 탐색하여야 하는 위치 수는 $(s_h^{search}-s_h^{temp})\times(s_v^{search}-s_v^{temp})$이며, 여기서 s_h^{temp}와 s_v^{temp}은 각각 템플릿의 수평 및 수직방향 크기이며, 이에 해당하는 현재 프레임의 탐색범위의 값은 각각 s_h^{search}과 s_v^{search}이다. 본 논문에서는 탐색할 때 비용함수로 SAD(sum-of-absolute differences)를 사용한다. 즉, 식 (5)을 이용하여 탐색범위 내의 모든 위치를 탐색하여 그 중 가장 작은 SAD값을 갖는 위치($\min(SAD_{i,j})$)를 선택한다. 식 (5)에서 t_{width}와 t_{height}은 각각 템플릿의 가로와 세로의 크기를 의미하며, $I_{ref}(i_r,j_r)$은 참조 영상의 템플릿의 위치를 의미한다.

$$SAD_{i,j}=\sum_{h=0}^{t_{width}}\sum_{v=0}^{t_{height}}|I_{ref}(i_r+h,j_r+v)-I_{curr}(h,v)| \quad (5)$$

2.2.5. 얼굴검출 과정으로 귀환

앞에서 언급한 바와 같이 얼굴추적 과정을 수행하다가 템플릿 매칭에 실패하면 얼굴검출 과정으로 귀환하여 얼굴검출을 다시 수행한다. 이것은 장면이 바뀌거나 사람이 화면에서 사라지는 경우 등에 나타난다. 이와 같은 경우는 추적과정에서 SAD값이 매우 크게 나타나는데, 본 논문에서는 식 (6)과 같이 탐색범위 전체를 탐색한 후 각 위치에서의 SAD값의 최솟값이 문턱치 값 T_h보다 큰 경우로 결정한다.

$$\min(SAD_{i,j})>T_h \quad (6)$$

Ⅲ. GPU을 이용한 얼굴 추적 고속화

3.1. GPU 구조

CPU와는 달리 GPU는 구조적으로 연산능력이 매우 뛰어나다. 그림 7[13]을 보면 CPU보다 GPU가 더 많은 ALU(Arithmetic and Logic Unit)을 갖고 있음을 볼 수 있다. ALU의 개수가 많으면 더 많은 양의 데이터들을 효율적으로 연산할 수 있다. 그래서 방대한 연산 처리가 필요한 다양한 분야에서 이미 GPU가 많이 활용되고 있다. 위와 같이 다소 연산양이 많은 알고리즘의 연산처리를 위해서 NVidia사에서 개발한 GPU기반 병렬처리 아키텍처 CUDA을 사용하고 있다[14]. CUDA은 GPU을 구성하고 있는 다수의 스트림 코어들을 통해 동시에 다수의 작업들을 연산할 수 있는 시스템이다.

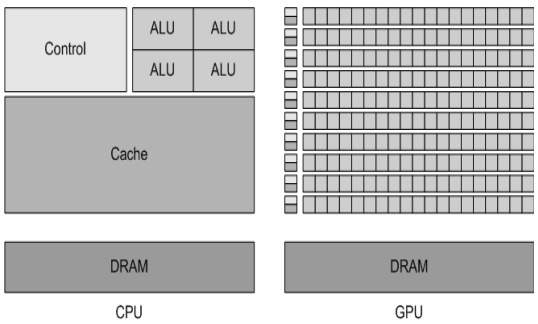

그림 7. CPU와 GPU 구조
Fig. 7 Structure of CPU and GPU

GPU는 Grid와 Block 이라는 단위로 작업들을 나우어서 GPU의 각각의 스트림 코어에 할당을 하고 동시에 연산을 수행한다. 일반적으로 스트림 코어의 개수가 많으면 많을수록 동시에 연산할 수 있는 작업의 개수도 늘어난다.

3.2. 템플릿 매칭 연산의 고속화

얼굴 추적은 탐색영역에서 근사 값에 해당하는 위치로 하여 얼굴을 찾기 때문에 탐색영역 내의 모든 위치를 탐색해야 한다. 그렇기 때문에 연산양이 많아서 고속으로 연산하는 기법이 매우 중요하다. 본 논문에서는 탐색영역에서 얼굴을 찾기 위한 방법으로 템플릿 매칭 기법을 사용한다. 이는 템플릿의 크기와 그에 따른 탐색영역에 따라 연산양이 달라진다.

예를 들어 템플릿의 크기가 각 방향으로 s_h^{temp}, s_v^{temp} 이고, 탐색영역이 각 방향으로 s_h^{search}, s_v^{search}이면 연산하는 횟수는 $((s_h^{search}-s_h^{temp})\times(s_v^{search}-s_v^{temp}))\times(s_h^{temp}\times s_v^{temp})$이다. 위와 같이 많은 연산 때문에 템플릿 매칭을 고속으로 연산해야 한다. 본 논문에서는 CUDA의 병렬처리 연산을 통해 템플릿 매칭을 고속으로 연산하였다. 그림 8은 탐색영역에 대하여 각 위치에서의 템플릿 매칭 연산이 GPU을 통해서 병렬적으로 동시에 처리됨을 나타내고 있다. 각 쓰레드는 식 (5)을 이용하여 SAD값을 계산하여 인덱스(템플릿의 위치) 값과 함께 전역 메모리에 저장하였다.

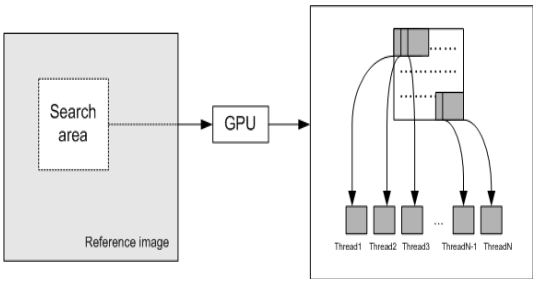

그림 8. 템플릿 매칭의 병렬화 구조
Fig. 8 Parallelization structure of template matching

3.3. 병렬축소를 통한 성능의 최적화

각 쓰레드에서 계산된 SAD값을 병합하여 최종적인 결과인 $\min(SAD_{i,j})$을 구하였다. 전역 메모리에 저장된 각각의 SAD값에 대한 $\min(SAD_{i,j})$을 구하기 위해서 전역 메모리의 데이터를 공유 메모리로 읽어왔다.

그리고 블록의 스레드는 인접한 요소의 배열의 SAD 값을 비교하면서 결과를 얻었다. 그것을 루프를 통해 반복하면서 블록의 마지막 결과를 얻으면 다시 전역 메모리로 값을 읽어왔다.

하지만 위와 같은 방식으로 병렬축소를 구현을 하게 되면 CUDA 환경에서 효과적으로 동작하지 않는 부분이 있을 수 있다. 그 한 가지 문제로 공유 메모리 접근을 통한 뱅크 충돌이다. 뱅크 충돌을 해결하기 위해서 메모리 접근 방식을 바꿔서 그림 9와 같은 순차적 어드레싱[15]이 적합하다. 루프를 통해 반복하면서 가장 작은 SAD값을 찾았고, 그것에 해당하는 인덱스(템플릿의 위치)를 전역 메모리에 저장하였다. 공유 메모리를 사용함으로써 전역메모리에 접근하는 횟수를 최소화 하였다.

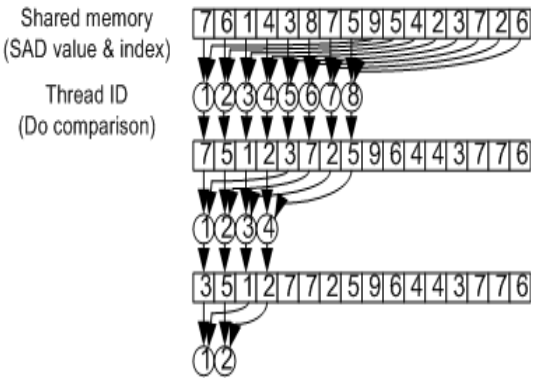

그림 9. 병렬축소를 위한 순차적 어드레싱
Fig. 9 Sequential addressing for parallel reduction

Ⅳ. 실험결과

본 논문에서 제안하는 방법을 구현하고 여러 테스트 시퀀스를 대상으로 실험을 수행하였다.

구현은 Microsoft window7 운영 체제에서 Microsoft visual studio 2010, OpenCV Library 2.4.3과 CUDA 5.0을 이용하였으며, 실험에 사용된 PC의 사양은 2.67GHz의 Intel Core i5 CPU, NVIDIA GeForce GTX 285와 6GB RAM이었다.

표 2에 실험에 사용한 테스트 시퀀스들을 나열하였다. 알고리즘의 테스트를 위해 직접 제작한 전후, 좌우, 상하 움직임에 대한 테스트 시퀀스를 이용하였고, 각 테스트 시퀀스는 200프레임으로 제작하였다.

표 2. 실험에 사용된 테스트 시퀀스
Table. 2 Test sequences used experiments

Source	Sequence	Resolution		Frame
		RGB	Depth	
Self-made	LR	640×480	640×480	200
	BF	640×480	640×480	200
	UD	640×480	640×480	200

4.1. 검출 및 추적 오차율

표 3은 평균적인 얼굴 검출률에 대하여 Viola&Jones 방법[19]와 본 논문에서 제안한 방법을 비교한 결과이다. 표 2에서 보듯이 검출 성공률은 유사하나, 오검출률이 14.84%가 작게 나타나 우수한 결과를 보이고 있음을 확인할 수 있다.

추적 위치 에러는 식 (7)과 같이 ground-truth 위치와 추적한 위치 사이를 유클리드 거리로 계산하였다. 식 (7)의 x, y은 ground-truth의 위치이다. 그리고 x', y'은 추적한 위치에 해당한다.

$$E_d = \sqrt[2]{(x'-x)^2+(y'-y)^2} \quad (7)$$

표 3. 평균 검출률 비교
Table. 3 Average detection rate comparison

Method	Detection Success Rate(%)	Detection Failure Rate(%)	Wrong Detection Rate(%)
Viola& Jones	98.66	1.34	15.05
Proposed Method	98.67	1.33	0.66

그림 10은 각각의 시퀀스를 가지고 추적한 위치 에러를 측정한 결과를 보여준다. 그림 10에서 보여주듯이 전후(Back&Forth, BF), 좌우(Left&Right LR), 상하(Up& Down, UD)방향에 추적위치에러 평균은 각각 5.74(BF), 6.55(LR), 6.60(UD)으로 ground-truth의 위치와 거의 차이가 없는 것을 알 수 있다.

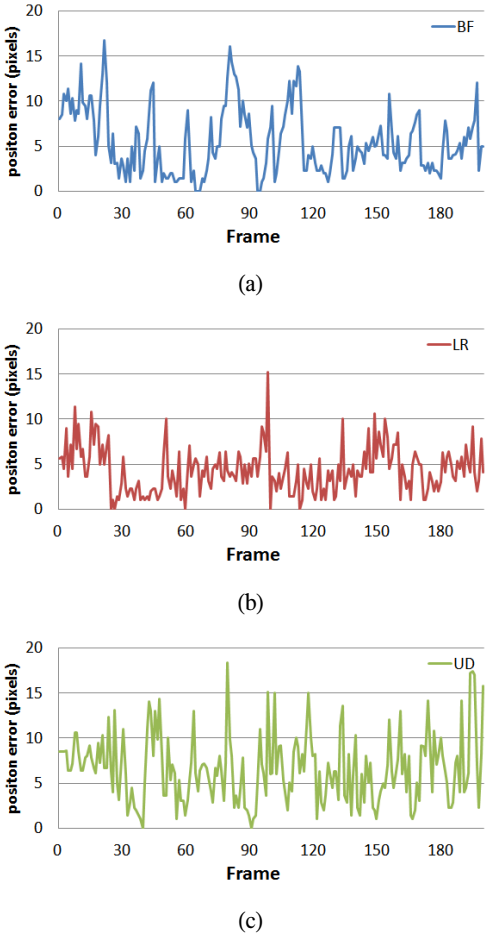

그림 10. 각 테스트 시퀀스에 해당하는 위치에러 (a) BF, (b) LR, (c) UD
Fig. 10 Position error corresponding to each test sequences (a) BF, (b) LR, (c) UD

4.2. 검출 및 추적 수행시간

그림 11은 제안한 얼굴 검출에 대한 평균 수행시간을 계산한 것이다. 그림 11에서 볼 수 있듯이, 제안하는 검출 방법의 속도는 Viola&Jones(V&J) 방법과 비교하여 평균 26.65ms로 비교적 빠른 속도를 보였다.

표 4는 제안한 얼굴 추적을 각각 CPU와 GPU에 대하여 수행시간을 계산한 것이다. 카메라와의 거리에 따른 템플릿의 크기와 그에 따른 연산하는 횟수가 다르기 때

문에 거리가 가까울수록 즉 템플릿의 크기가 클수록 수행시간이 오래 걸리는 걸 알 수 있다. CPU와 GPU의 수행속도를 비교했을 때는 최대 49배 까지 속도가 향상되는 것을 확인 할 수 있었다.

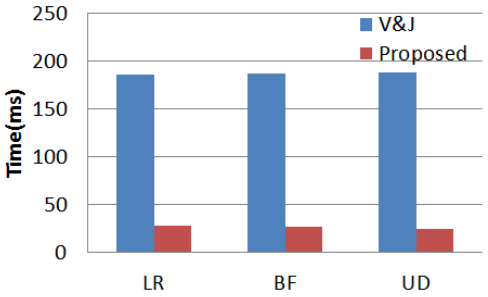

그림 11. 검출 수행시간 비교
Fig. 11 Detection processing time comparison

표 4. CUP와 GPU 추적 수행시간 비교
Table. 4 Tracking processing time comparison of CUP and GPU

Distance (cm)	Template Size	CPU	GPU	Speed-up
100	98*98	245ms	5ms	x49
130	70*70	66ms	3ms	x22
160	58*58	25ms	2ms	x12.5

V. 결 론

본 논문에서는 얼굴을 검출하고 GPU 기반으로 얼굴을 고속으로 추적하는 알고리즘을 제안하였다. 얼굴 검출 방법에서는 기존의 방법인 Adaboost을 이용하지만 깊이영상과 RGB영상을 이용하여 Adaboost의 입력영상을 제한하여 얼굴을 정확하고 빠르게 검출하였다. 그리고 얼굴 추적 방법에서는 영상의 깊이에 따른 적절한 템플릿의 크기조절과 탐색영역을 설정하여 상하, 좌우, 전후의 얼굴 움직임에 대하여 좀 더 정확하게 얼굴을 추적할 수 있도록 하였다. 얼굴추적은 깊이 영상만을 이용하여 템플릿 매칭을 하였기 때문에 얼굴 추적 시 조명변화에도 강하였다. 또한 얼굴 추적 알고리즘을 고속으로 구현하기 위해서 GPU를 이용하여 템플릿 매칭 부분을 병렬하여 연산하였다.

직접 제작한 동영상에 제안한 알고리즘을 적용하여 실험한 결과 얼굴검출 방법에 있어서는 Viola&Jones의 방법과 비교 하였을 때 검출율은 비슷하였으나, 오검출률은 상당히 낮은 것을 확인하였다. 추적위치 에러 또한 전후, 좌우, 상하방향에 있어서 ground-truth의 위치와의 거리가 7픽셀 아래로 상당히 낮은 것을 확인하였다. 수행시간은 얼굴검출 방법에 있어서는 Viola&Jones의 방법과 비교 하였을 때 약 7배의 속도 향상을 보였다. 얼굴추적 방법에서는 CPU와 비교 하였을 때 GPU 수행속도가 최대 49배까지 향상되는 것을 확인하였다.

따라서 제안한 방법은 초당 30 프레임 이상의 실시간 얼굴추적 시스템에 사용하기 적합하며, 다양한 분야에서 적용하여 사용할 수 있을 것으로 사료된다.

감사의 글

본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2010-0012898)

참고문헌

[1] G, Q, Zhao, et al., "A Simple 3D face Tracking Method based on Depth Information," Int'l Conf. on Machine Learning and Cybernetics, pp. 5022-5027, Aug. 2005.

[2] C. X. Wang and Z. Y. Li, "A New Face Tracking Algorithm Based on Local Binary Pattern and Skin Color Information," ISCSCT, Vol. 2, pp. 20-22, Dec. 2008.

[3] M. H. Yang, et al., "Detecting Faces in Images; A Survey," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 24, No. 1, pp. 34-58, Jan. 2002.

[4] G. Z. Yang and T. S. Huang, "Human Face Detection in Complex Background," Pattern Recognition, Vol.

27, No. 1, pp. 53-63, Jan. 1994.
[5] L. Craw, D. Tock, and A. Bennett, "Finding Face Features," Proc. Second European Conf. Computer Vision, pp. 92-96, 1992.
[6] MESA SR4000 MESA Imaging, 2011 [Online]. Available:http://www.mesa-imaging.ch/prodview4k.php
[7] Kinect Website, http://www.microsoft.com/en-us/kinectforwindows/
[8] R. Cabido, "High-performance template tracking", J. Vis. Commun. Image, pp 271-286, 2012.
[9] Oscar Mateo Lozano, Kazuhiro Otsuka, "SIMULTANEOUS AND FAST 3D TRACKING OF MULTIPLE FACES IN VIDEO BY GPU-BASED STREAM PROCESSING", ICASSP, 2008.
[10] Claus Lenz, Giorgio Panin, Alois Knoll, "A GPU-accelerated particle filter with pixel-level likelihood", 2008.
[11] GPGPU Website, http://gpgpu.org
[12] Douglas Chai, et al., "Locating Facial Region of a Head-and -Shoulders Color Image," Int'l Conf. Automatic Face and Gesture Recognition, pp. 124-129, April 1998.
[13] David B. Kirk and Wen-mei W. Hwu, "Programming Massively Parallel Processors: A Hands-on Approach 1st Ed", ELSEVIER
[14] NVidia Website, http://www.nvidia.co.kr/
[15] Mark Harris, "Optimizing Parallel Reduction in CUDA," NVIDIA Developer Technology, 2007.
[16] X. Suau, J. Ruiz-Hidalgo and J. Casas, "Real-Time Head and Hand Tracking Based on 2.5D Data", IEEE Trans. Multimedia, Vol. 14, No. 3, pp. 575-585, June 2012.
[17] S.-K. Kwon and S.-W. Kim, "Motion Estimation Method by Using Depth Camera," J. Broadcast Engineering, Vol. 17, No. 4, pp. 676-683, Jul. 2012.
[18] S.-K. Kwon, Y.-H. Park and K.-R. Kwon, "Zoom Motion Estimation Method by Using Depth Information," J. Korea Multimedia Society, Vol. 16, No. 2, pp. 131-137, Feb. 2013.
[19] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," Computer Vision, Vol. 52, No. 2, pp. 137-154, 2004.

저자소개

김우열(Woo-Youl Kim)

2012년 2월 : 안양대학교 정보통신공학과 졸업(공학사)
2012년 3월 ~ 현재 : 광운대학교 전자재료공학과 석사과정

※관심분야 : 3D 영상처리, Tracking, Stereo Vision

서영호(Young-Ho Seo)

1999년 2월 : 광운대학교 전자재료공학과 졸업(공학사)
2001년 2월 : 광운대학교 일반대학원 졸업(공학석사)
2004년 8월 : 광운대학교 일반대학원 졸업(공학박사)
2005년 9월 ~ 2008년 2월 : 한성대학교 조교수
2008년 3월 ~ 현재 : 광운대학교 교양학부 부교수

※관심분야 : 실감미디어, 2D/3D 영상 신호처리, 디지털 홀로그램, SoC 설계

김동욱(Dong-Wook Kim)

1983년 2월 한양대학교 전자공학과 졸업(공학사)
1985년 2월 한양대학교 공학석사
1991년 9월 Georgia공과대학 전기공학과(공학박사)
1992년 3월 ~ 현재 광운대학교 전자재료공학과 정교수
2009년 3월 ~ 현재 광운대학교 실감미디어 연구소 연구소장

※관심분야 : 3D 영상처리, 디지털 홀로그램, 디지털 VLSI Testability, VLSI CAD, DSP설계, Wireless Communication