H.263에서 움직임 정합 블록을 위한 개선된 3단계 탐색 알고리즘

(An Improved Three Step Search Algorithm for the Motion Match Blocks in H.263)

심 종 채 † 박 영 목 †† 성 윤 주 ††† 유 경 종 ††††
(Jongchae Sim) (Yeongmok Park) (Yunju Seong) (Kyeongjong Yoo)
박 재 흥 ††††† 서 영 건 †††††
(Jaeheung Park) (Yeonggeon Seo)

요 약 H.263을 화상 회의에 이용할 때, 압축율과 마찬가지로 중요한 것이 인코딩과 디코딩 시간이다. 특히 인코딩 시간을 줄이기 위하여 많은 방법들이 제안되었는데, 그 중 한 가지가 인코딩 시간의 상당 부분을 차지하는 움직임 추정에서 복잡도를 감소시키는 방법이다. 이러한 움직임 추정의 복잡도를 규정짓는 요소로 비용 함수, 탐색 영역 인수, 움직임 탐색 알고리즘이 있다. 인코딩 시간을 줄이기 위해서는 이 세 가지의 복잡도를 줄이면 된다. 특히 비용 함수가 H.263 인코딩 시간의 상당 부분을 차지하고 있는데, 이는 비용 함수가 호출되는 횟수가 매우 많기 때문이다.

본 연구에서는 움직임 탐색 알고리즘의 복잡도를 감소시킴으로써 비용함수의 호출 횟수를 줄여 전체 인코딩 시간을 줄이고자 한다. 움직임이 적은 경우 TSS(Three Step Search) 및 NTSS(New TSS)와 비교하여 더욱 빠른 움직임 탐색이 가능하도록 하고, TSS보다 많은 체크포인트를 요구하는 NTSS의 단점을 개선한 ITSS(Improved TSS)를 제안하였다. 그리고 본 알고리즘과 다른 알고리즘의 PSNR, 파일 크기, SAD 호출 횟수 비교로 실험하였다.

키워드 : H.263, TSS, NTSS, ITSS, 움직임 추정, 복잡도, 비용함수

Abstract In video conferencing system using H.263, encoding and decoding time is as important as compression rate is. To reduce encoding time, a number of methods were proposed. We use a method of them that reduces the computational complexity in motion estimation. The complexity is determined by three factors, such as a cost function, a search range parameter, and a motion search algorithm. In fact, it takes a lot of time to encode the video data on account of the cost function factor. That's the reason that we use the factor to reduce encoding time.

In this paper, we tried to reduce total encoding time by reducing the number of calling the cost function. In case of a little moving, our algorithm enabled faster motion searching than TSS(Three Step Search) and NTSS(New TSS). Here, we called the algorithm by an ITSS(Improved TSS) that improves a shortcoming of NTSS requiring more checkpoints than TSS. For an experimentation, our algorithm was compared to other algorithms using PSNR, file size and SAD call times.

Key words : H.263, TSS, NTSS, ITSS, motion estimation, complexity, cost function

† 정 회 원 : 남해전문대학 컴퓨터응용정보과 교수
simjc@nc namhac.ac.kr
†† 비 회 원 : 경상대학교 컴퓨터과학과
ympark@cjcc.chinju.ac kr
††† 정 회 원 : 도서출판 비비컴
prince@bbcom.co kr
†††† 학생회원 : 경상대학교 컴퓨터과학과
mac30@rtp.gsnu.ac kr
††††† 비 회 원 : 경상대학교 컴퓨터과학과 교수
jhpark@gsnu.ac.kr
†††††† 종신회원 : 경상대학교 컴퓨터교육과 교수
young@gsnu.ac.kr
논문접수 : 2001년 5월 29일
심사완료 : 2001년 11월 20일

1. 서 론

H.261과 H.263[1]은 초기에 ISDN 상의 화상 회의용 비디오 코덱의 표준으로 ITU-T에서 제안되었는데 화질 개선 및 효율적인 압축 알고리즘의 개발로 인하여 도난 방지용 감시 시스템이나 인터넷 방송 등과 같은 다양한 응용 분야에서 널리 사용되고 있는 저비트율의 동영상 압축 표준이다[2]. 특히 H.263은 20Kbps 정도의 비트율(bitrate)을 목적으로 만들어졌기 때문에 코딩 효율이 우수하며 20Kbps보다 훨씬 낮은 비트율에서도 H.261에 비하여 50% 이상의 비트율을 절감할 수 있다.

화상 회의 어플리케이션의 특성상, H.263에서 압축율 못지 않게 중요한 것이 인코딩과 디코딩의 실시간성이다. 특히 인코딩 시간을 줄이기 위하여 많은 방법들이 제안되었는데, 그 중 한 가지가 인코딩 시간의 상당 부분을 차지하는 움직임 추정(motion estimation)에서 복잡도를 감소시키는 방법이다.

측정된 움직임 벡터가 가리키는 블록과 현재 프레임의 매크로블록간의 불일치 정도가 임계치를 넘으면 그 매크로블록은 공간적 중복성만을 제거한 INTRA 코딩이 되고, 두 블록간의 불일치 정도가 임계치 이하이면 공간적 시간적 중복성을 제거한 INTER 코딩을 하게 된다. 이러한 움직임 추정의 복잡도를 규정짓는 요소로 비용 함수(cost function), 탐색 영역 인수(search range parameter), 움직임 탐색 알고리즘(motion search algorithm)이 있는데, 인코딩 시간을 줄이기 위해서는 이 세 가지의 복잡도를 줄이면 된다. 이 중에서 특히 비용 함수가 H.263 인코딩 시간의 상당 부분을 차지하고 있는데, 이는 비용 함수 자체의 실행 시간이 오래 걸리는 것이 아니라 비용 함수가 호출되는 횟수가 매우 많기 때문이다. 결국 이러한 비용 함수의 호출 횟수를 결정짓는 것은 움직임 탐색 알고리즘이다.

본 연구에서는 움직임 탐색 알고리즘의 복잡도를 감소시킴으로써 비용함수의 호출 횟수를 줄이고, 궁극적으로는 전체 인코딩 시간을 단축시키고자 한다. 이를 위해서, 움직임이 적은 경우 TSS(Three Step Search) 및 NTSS(New TSS)와 비교하여 더욱 빠른 움직임 탐색이 가능하도록 하고, 움직임이 적지 않은 경우에는 TSS보다 많은 체크포인트를 요구하는 NTSS의 단점을 개선한 ITSS(Improved TSS)를 제안하였고, 실험은 PSNR, 파일 크기, SAD 호출 횟수로 성능 비교를 하였다.

2. H.263의 움직임 추정 기법

이 장에서는 H.263에서 사용되는 움직임 추정 기법의 복잡도와 움직임 추정 알고리즘에 대하여 설명한다. 가장 기본적인 움직임 탐색 알고리즘으로 탐색 영역을 모두 검사하는 FS(Full Search) 알고리즘이 있고, 이것의 탐색 영역내의 특정 위치의 매크로블록만 검사하여 복잡도를 감소시킨 알고리즘인 FSA(Fast Search Algorithm)가 있다. 이 FSA에는 TSS(Three Step Search)[4-5], NTSS(New TSS)[6], 2DLOG(2D-Logarithmic Search)[7], CSA(Cross Search Algorithm)[8], 4SS(Four Step Search)[9], BBGDS(Block-Based Gradient Descent Search), OSA(Orthogonal Search Algorithm), HBMA (Hierachical Block Matching Algorithm) 등이 있다[10-12].

2.1 움직임 추정 기법의 복잡도

H.263에서 인코딩에서 가장 많은 시간을 차지하는 것이 움직임 추정 부분이다. 실시간 어플리케이션에서는 인코딩 시간이 중요한데, 이를 줄이기 위해서는 움직임 측정 부분의 복잡도를 줄이면 된다. 움직임 추정 기법의 복잡도는 탐색 범위 인수, 비용 함수, 탐색 알고리즘의 다음 세 가지로 나타낼 수 있다[3].

첫째, 탐색 영역 인수는 탐색 영역의 범위를 결정하기 위한 인수로서, H.263에서 일반적인 탐색 영역 인수값은 6이다[5].

둘째, 탐색 영역에서 16픽셀×16라인 크기의 후보 블록을 구하면 그 후보 블록이 현재 프레임의 매크로블록과 얼마나 유사한지를 추정해야 하는데, 이때 사용하는 함수가 비용 함수이다. 바꾸어 말하면 블록간의 유사도를 추정하는 것이 아니라 불일치 정도를 추정하는 것으로, 비용 함수의 결과가 최소인 후보 블록이 정합 블록이 된다. 비용 함수 중 가장 널리 사용되는 것으로 SAD(Sum-of-Absolute-Difference)가 있는데 공식은 다음과 같다.

$$SAD = \sum_{k=1}^{16}\sum_{l=1}^{16} | B_{i,j}(k,l) - B_{i-u,j-v}(k,l) |$$

위 식에서 $B_{i,j}(k,l)$는 현재 프레임의 (i, j)번째 16픽셀×16라인 매크로블록에서 (k, l)번째 픽셀을 의미하며, $B_{i-u,j-v}(k,l)$는 참조 프레임의 $(i-u, j-v)$번째 매크로블록에서 (k, l)번째 픽셀을 의미한다.

셋째, 탐색 알고리즘은 현재 프레임의 매크로블록이 이전 프레임의 어느 위치에 있었는가를 찾기 위해서 이전 프레임의 탐색 영역에서 수행하게 되는 알고리즘을 말한다.

위의 세 가지 요인에 따라 움직임 추정의 복잡도가 좌우되므로, 움직임 추정에서 속도를 개선하고자 할 때는 이 세 가지 요인의 복잡도를 줄이면 된다. 본 논문에

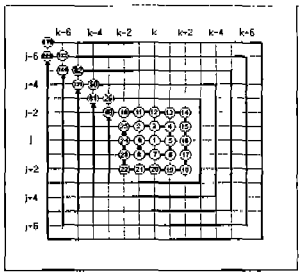

그림 1 나선형 탐색

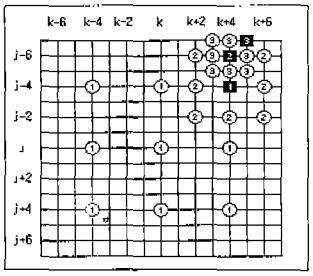

그림 2 TSS

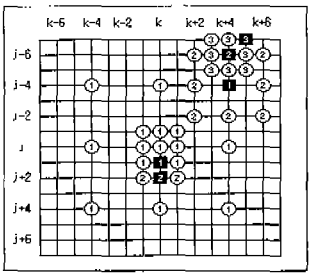

그림 3 NTSS

서는 다음에 소개되는 움직임 추정 기법들의 복잡도를 줄인 ITSS를 제안한다.

2.2 FS

FS(Full Search) 알고리즘은 가장 정확하며 단순하다. 그러나 이전 프레임의 탐색 영역을 모두 조사하므로 시간이 많이 걸리는 것이 단점인데, 실험에 의하면 300 프레임의 QCIF 포맷 화일을 압축할 때, 탐색 영역의 크기를 15픽셀×15라인으로 주면 5,364,019번에 걸쳐 비용함수를 호출함으로써 전체 인코딩 시간의 절반 가량을 차지하게 된다. 그림 1은 FS 알고리즘 중 나선형의 모양으로 탐색 영역을 검사하는 나선형 탐색(Spiral Search)을 나타낸다.

2.3 TSS

TSS(Three Step Search)[4] 알고리즘은 각 단계마다 스텝사이즈를 절반씩 줄임으로써 탐색 영역을 좁혀가는 방식의 탐색 알고리즘이다. 초기 스텝사이즈는 움직임 벡터의 수평, 수직 방향 최대 변위인 d의 절반으로서, d가 홀수인 경우 $d/2$보다 크고 $(d/2+1)$ 보다 작은 정수 $\lceil d/2 \rceil$가 초기 스텝사이즈가 된다. 각 단계마다 센터와 센터로부터 스텝사이즈만큼 떨어진 8개의 체크포인트가 검사되며, 이 중 비용 함수의 결과값이 최소인 체크포인트가 다음 단계의 시작 센터로 선택된다.

그림 2와 같이 d=7일 경우, (9+8+8)=25개의 체크포인트를 검사한다. d가 7보다 큰 경우 단계를 확장하여 동일한 방식으로 움직임 정합 블록을 찾아낼 수 있는데, 이때 필요한 체크포인트의 개수는 $\lceil 1+8\lceil \log_2(d+1) \rceil \rceil$와 같다.

2.4 NTSS

전형적인 화상회의 어플리케이션에서 영상은 사람의 어깨 위쪽 부분만 보이고 움직임이 적은 것이 일반적이다. NTSS(New TSS)[6] 알고리즘은 이러한 특성을 이용하여 TSS를 변형한 것으로, TSS의 1단계 체크포인트에 센터를 둘러싼 8개의 체크포인트가 추가되었다.

그림 3은 d=7인 경우에 가능한 두 가지의 탐색 경로를 보여준다. 아래쪽의 경로는 1단계에서 센터를 둘러싼 8개의 체크포인트 중 하나가 최적합점으로 선택된 경우로, 작은 움직임을 탐색하는데 효율적이다. 체크포인트의 개수는 (17+3)=20이다. 반면 그림 3에서 위쪽의 경로는 1단계에서 결정된 최적합점이 TSS의 1단계 체크포인트와 동일한 8개의 체크포인트 중 하나인 경우로서 이후의 진행과정은 TSS와 동일하다. 이때의 체크포인트 합계는 (17+8+8)=33이 된다.

3. 개선된 알고리즘 ITSS

여기에서는 H.263 인코딩의 처리 속도를 향상시키기 위해 기존의 TSS 알고리즘을 화상 회의라는 어플리케이션의 특성에 맞게 개선한 ITSS(Improved TSS) 알고리즘을 제안한다.

3.1 기존 알고리즘과의 비교

이 절에서는 실제 움직임 벡터의 위치에 따른 TSS, NTSS 및 ITSS 1회 탐색시 요구되는 체크포인트의 개수를 비교한다. 비교시 각 단계에서의 최적합점은 정합 블록과 가까울수록 유사도가 높다는 가정을 전제로 하여 선택된다. 그림 4는 15픽셀×15라인 크기의 탐색 영역에서 센터로부터 각각 0, 1, 2, 3, 7만큼의 거리에 있는 최적합점이 있는 경우를 각각 나타낸다. 즉 A는 탐색 영역의 센터이며, B는 센터와의 거리가 1인 8개의 체크포인트 중 하나를 나타낸 것이다. 마찬가지로 C, D,

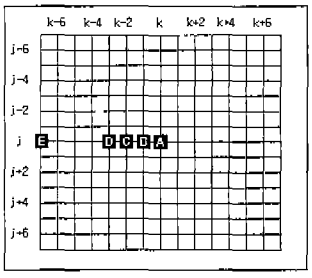

그림 4 테스트 블록 위치

E 또한 센터와의 거리가 2, 3, 7인 각각 8개의 체크포인트를 대표하여 나타낸 하나의 예이다.

정합 블록의 원점의 위치가 A, B, C, D, E에 있는 경우에 대하여 각각 TSS, NTSS, ITSS 탐색 알고리즘을 각각 비교하되, d=7일 때 FS는 15×15=225개로 고정된 개수의 체크포인트를 검사하므로 비교 대상에서 제외된다.

먼저 A를 원점으로 하는 블록이 정합 블록인 경우이다. 정합 블록의 원점이 탐색 영역의 센터인 경우로서 인코딩하려는 블록의 움직임이 없었음을 의미한다. 이 경우 TSS는 1단계에서 센터를 검사하지만 25개의 체크포인트를 모두 검사함으로써 3단계를 끝내야만 1을 최적합점으로 확정짓는다(그림 5). NTSS는 1단계에서 17개의 체크포인트를 검사한 후 최적합점으로 결정한다(그림 6). ITSS는 1단계에서 1을 검사하지만 2단계까지의 17개의 체크포인트를 검사한 후 1을 최적합점으로 결정한다(그림 7).

두 번째로 그림 4의 B가 최적합점인 경우에도 TSS는 2단계에서 3을 검사하지만, 25개의 체크포인트를 모두 검사한 후에야 3을 최적합점으로 결정짓는다(그림 8). NTSS는 1단계에서 1을 검사하지만 2단계까지의 20번의 검사를 마친 후에 1을 최적합점으로 결정지으며(그림 9), ITSS는 1단계에서 17개의 체크포인트를 검사하고 2를 최적합점으로 결정짓는다(그림 10).

세 번째로 C가 최적합점인 경우를 비교해 보자. 첫 번째 비교와 두 번째 비교에서 나타났듯이 TSS는 동일

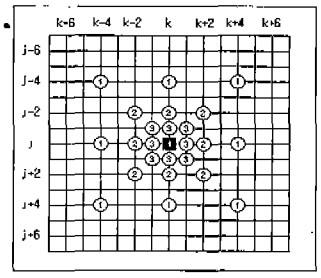

그림 5 비교1 - TSS

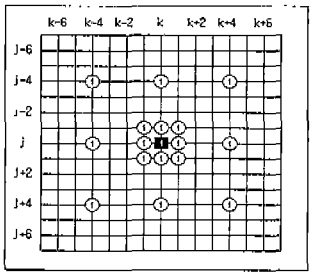

그림 6 비교1 - NTSS

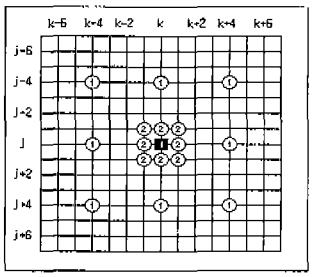

그림 7 비교1 - ITSS

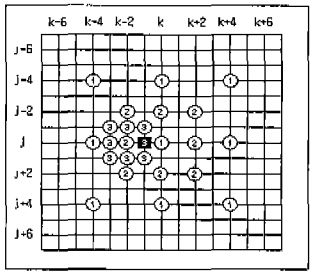

그림 8 비교2 - TSS

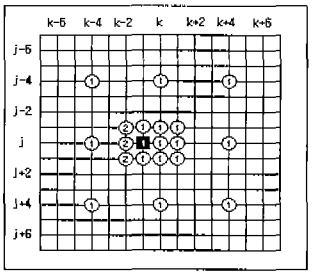

그림 9 비교2 - NTSS

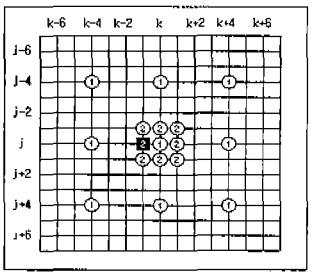

그림 10 비교2 - ITSS

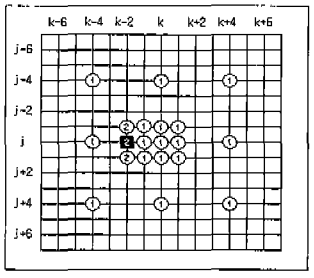

그림 11 비교3 - NTSS

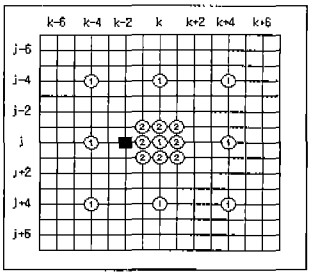

그림 12 비교3 - ITSS(1)

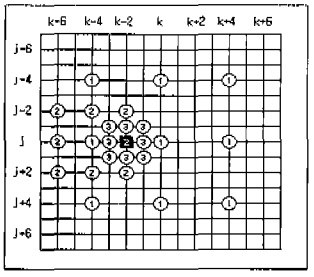

그림 13 비교3 - ITSS(2)

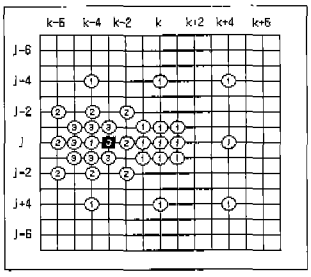

그림 14 비교4 - NTSS

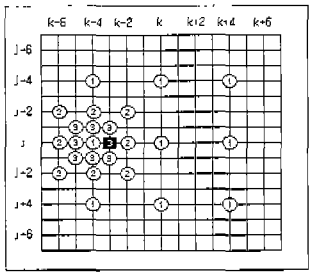

그림 15 비교 4 - ITSS

한 방식으로 진행된다. NTSS는 1단계에서 2에 이웃한 체크포인트 ①을 찾은 후, 2단계에서 3번의 추가 검색으로 2를 최적합점으로 결정할 수 있다(그림 11).

C가 거리상으로 센터와 ITSS 1단계 체크포인트의 중간에 있기 때문에, C가 최적합점인 경우에는 1단계에서 최적합점으로 센터가 선택되는 한 가지 경우와 센터가 아닌 8개의 체크포인트 중 하나가 선택되는 또 다른 경우로 나뉠 수 있다. 전자의 경우는 그림 12와 같다. ITSS는 2단계에서 검색을 끝내므로 17개의 체크포인트를 검사한 후 C에 근접한 ②를 최적합점이라고 결정하게 된다. 결과적으로 움직임 벡터는 수평 방향으로 1만큼 오차가 생기게 된다. 그림 13은 C가 최적합점이고, ITSS의 1단계에서 센터가 아닌 8개의 체크포인트 중 하나가 선택된 경우이다. 이때는 2단계에서 2를 검사하지만, 3단계가 끝난 후 최적합점으로 확정짓는다.

마지막으로 센터로부터 3만큼 떨어진 D와 7만큼 떨어진 E가 정합 블록인 경우를 살펴보자. 최적합점과 센터 간의 거리가 3이상이 되면 NTSS와 ITSS는 TSS와 동일하게 동작하므로 각 알고리즘이 D와 E 지점에서 실행되는 과정 또한 동일하다. 예를 들어 D가 최적합점인 경우를 살펴보면, NTSS는 초기 체크포인트 17개를 가지는 것을 제외하면 나머지는 TSS와 동일하게 3단계에서 3을 검색하고 총 체크포인트의 개수는 33개가 된다(그림 14).. 이에 반해 ITSS는 TSS와 완전히 동일하여 3단계에서 3을 검색하고 총 체크포인트의 개수는 25개가 된다(그림 15).

표 1 15×15 탐색 윈도우에서 체크 포인트의 수 비교

알고리즘	A	B	C	D
FS	225	225	225	225
TSS	25	25	25	25
NTSS	17	20	20	33
ITSS	17	17	17 or 25	25

표 1은 지금까지의 비교를 정리한 것이다.

TSS는 단지 최적합점에 가까운 위치일수록 유사도가 높다는 가정을 기반으로 하였을 뿐 영상의 특성을 이용하지는 않았다. 반면 NTSS는 화상 회의 영상의 특성에 기반하여 움직임이 작은 영상에서 TSS보다 더 적은 수의 체크포인트만으로도 정확한 움직임 측정이 되도록 하였다. 그러나 NTSS는 1단계에 추가된 8개의 체크포인트로 인하여 변위가 2이상이 되면 TSS보다도 더 많은 33개의 체크포인트를 요구한다는 단점이 있다.

이에 본 논문에서, NTSS와 마찬가지로 화상 회의 영상의 특성을 고려하여 움직임이 작은 경우 TSS 및 NTSS와 비교하여 더욱 빠른 움직임 탐색이 가능하도록 하고, 움직임이 작지 않은 경우에는 TSS보다 많은 체크포인트를 요구하는 NTSS의 단점 또한 개선한 ITSS를 제안한다.

3.2 ITSS의 구조 및 블록 정합 과정

그림 16의 (a)에서 ITSS는 현재 매크로블록의 움직임을 추정하기 위하여 현재 매크로블록이 현재 프레임에서 수평, 수직 방향으로 몇 번째 픽셀에 위치하는지를 나타내는 좌표 (*current_x*, *current_y*)를 입력받는다. 그리고 이 좌표와 탐색 영역 인수 *d*를 이용하여 이전

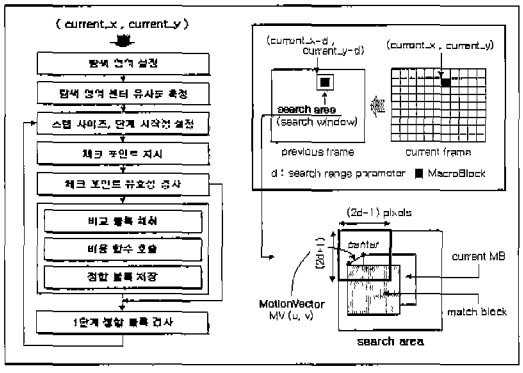

그림 16 (a) ITSS의 구조 (b) QCIF 포맷에서의 탐색 영역

프레임에서 (*current_x-d*, *current_y-d*)를 원점으로 하는 탐색 영역을 설정하게 되는데, 그림 16의 (b)는 QCIF 포맷의 한 프레임에서 탐색 영역이 어떻게 설정되는가를 보이고 있다. 탐색 영역의 크기는 $(2d+16)\times(2d-16)$이지만, 움직임 탐색 알고리즘에서 정합 블록을 찾기 위하여 탐색하게 되는 픽셀들은 센터를 중심으로 한 $(2d+1)\times(2d+1)$의 영역에 있는 픽셀들이 된다. 탐색 영역이 설정되면, 다음으로 센터에서 비용 함수를 호출하고 그 결과값을 최대 유사도로 저장한다. 여기까지의 과정은 모든 탐색 알고리즘에서 공통적인 과정이다.

ITSS에서의 블록 정합 과정을 단계별로 구체적으로 살펴보면 아래와 같다.

【 1 단계 】

a. 체크포인트와 센터간의 거리를 의미하는 스텝사이즈를 정한다. 스텝사이즈 초기값은 탐색 영역 파라미터의 절반이 된다.
b. 원점으로부터 스텝사이즈만큼 떨어진 8개의 점 중에서 시작점을 정한다. ITSS에서는 왼쪽 상단의 점에서부터 시작한다.
c. 지시된 체크포인트가 탐색 영역을 벗어난다면 검사할 수 없으므로, 체크포인트가 탐색 영역 내에 있는지 그 유효성을 검사한다.
d. 탐색 영역 내에 있는 체크포인트에 한해서 비용 함수를 호출한다.
e. 비용 함수의 결과가 최대 유사도보다 크다면 최대 유사도로 저장한다. 그리고 현재 매크로블록과 정합 블록간의 수평, 수직 방향 픽셀의 차이값, 즉 움직임 벡터 또한 저장한다.
f. 센터로부터 스텝사이즈만큼 떨어져있는 나머지 7개의 체크포인트에서도 c~e의 과정을 반복 한다.
g. 8개의 체크포인트에서 검사하여 결정된 1단계의 최적합점이 센터인지 아닌지를 검사한다.

【 2 단계 】

a. 1단계의 g에서 최적합점이 탐색 영역의 센터이면 2단계의 스텝사이즈를 1로 하고, 최적합점이 바깥쪽 8개의 체크포인트 중 하나이면 2단계의 스텝사이즈는 1단계 스텝사이즈의 절반이 된다. 그리고 1단계의 최적합점은 2단계의 센터가 된다.
b. 센터로부터 수직 또는 수평 방향으로 스텝사이즈만큼 떨어진 8개의 체크포인트 중 시작점을 설정한다.
c. 1단계의 c, d, e, f를 수행한다.

【 3 단계 】

a. 2단계의 스텝사이즈가 1이면 3단계의 d로 간다. 2단계의 스텝사이즈가 1이 아니면, 3단계의 스텝사이즈는 2단계 스텝사이즈의 절반이 된다. 2단계의 최적합점이 3단계의 센터로 설정된다.
b. 센터로부터 수직 또는 수평 방향으로 스텝사이즈만큼 떨어진 8개의 체크포인트 중 시작점을 설정한다.
c. 1단계의 c, d, e, f를 수행한다.
d. 최대 유사점에 저장되어 있는 좌표가 최종적인 최적합점, 즉 정합 블록의 원점이 된다.

다음 알고리즘 1은 위와 같은 ITSS 정합 과정을 알고리즘으로 나타낸 것이다.

알고리즘 1 ITSS

```
1:  algorithm ITSS ;
2:  begin
3:   step := 0 ;
4:   step_size := ⌈ d / 2 ⌉
5:   x := x_curr - step_size ;
6:   y := y_curr - step_size ;
7:   x_next := x_curr ;
8:   y_next := y_curr ;
9:   while step_size ≥ 1 do
10:  begin
11:   step := step + 1 ;
12:   for check_count := 0 upto 7 do
13:   begin
14:     if x ≥ xlow and x ≤ xhigh and y ≥ ylow and y
            ≤ yhigh then
15:    begin
16:       check_block := search_area + (x-xlow) + (y
                    -ylow) × h_length ;
17:       mismatch_curr := cost_function(check_block,
                          act_block, Min_Mismatch) ;
18:      if mismatch_curr < Min_Mismatch then
19:      begin
20:        MV.x := x - x_curr ;
21:        MV.y := y - y_curr ;
22:        x_next := x ;
23:        y_next := y ;
24:        Min_Mismatch := mismatch_curr
25:      end
26:    end
27:    if     check_count < 2 then x := x + step_size
28:       else if check_count < 4 then y := y +
                  step_size
29:       else if check_count < 6 then x := x -
                  step_size
30:    else    y := y - step_size
31:    end ;
32:     if step = 1 and MV.x = 0 and MV.y = 0 then
                  step_size := 1
33:    else
34:    begin
35:      if   step_size = 1 then step_size := 0
36:      else step_size := ⌈ step_size / 2 ⌉
37:    end
38:   x := x_next - step_size ;
39:   y := y_next - step_size
40:  end ;
41: end.
```

ITSS 알고리즘은 TSS의 1단계 결과에 따라 스텝사이즈를 다르게 정해주는데, 알고리즘 1에서 32~37번째 라인까지가 이에 해당한다. 그 외의 부분은 TSS의 알고리즘과 동일하다.

3.3 ITSS를 이용한 QCIF 영상의 움직임 추정

이 절에서는 실제 비디오 시퀀스를 이용하여 ITSS의 움직임 탐색 과정을 보인다. 여기에서 사용되는 영상은 carphone 비디오 시퀀스의 첫 번째 프레임과 두 번째 프레임이며 탐색 영역 인수는 7로 한다(그림 17). 비디오 시퀀스의 첫 번째 프레임은 항상 INTRA 코딩되는데, DCT 및 양자화를 거친 후 다시 역양자화와 역DCT를 거쳐 재구성된 프레임이 저장되었다가 다음 프레임의 INTER 코딩을 위하여 사용된다.

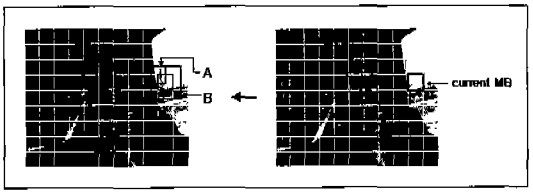

그림 17 carphone 비디오 시퀀스의 첫 번째와 두 번째 프레임

그림 17에서 현재 코딩하려는 매크로블록은 두 번째 프레임의 "current MB"이며, 이 MB에 대한 움직임 탐색 영역은 B가 된다. 그러나 사실상 움직임 탐색에서 검사하게 되는 체크포인트는 블록들의 원점이므로 A영역이 검사 영역이 된다. "current MB"의 좌표는 (144, 48)이며 탐색 영역 인수는 7이므로, 탐색 영역의 수평 성분의 범위는 첫 번째 프레임의 (137~151), 수직 성분의 범위는 (41~55)가 된다.

ITSS의 1단계는 그림 18과 같이 [1]~[9]의 순서로 탐색하며, 최적합점은 탐색 영역의 센터인 [1]로서 초기화된다 그리고 체크포인트를 하나씩 검사하면서 저장되어 있는 정합 블록보다 더욱 불일치도가 낮은, 즉 유사한 블록을 찾게 되면 그 블록의 불일치 정도와 좌표로 대체하게 된다.

1단계에서 [4]의 검색 순서가 되었을 때 유사도를 추정하면 최적합점으로 저장되어 있는 [1]번의 유사도보다 더 유사한 결과가 나오게 되어 최적합점은 [4]가 된다. 그림 18에서 오른쪽 그림은 1단계 각각의 체크포인트에서 검사된 16픽셀×16라인 크기 블록의 모양을 실제로 보인 것이다. 그리고 다시 [5]의 위치를 검색하면 이 지점의 유사도가 현재의 최적합점보다 더 높게 나오게 되

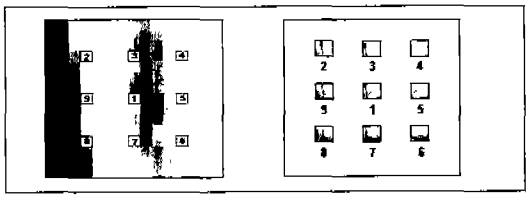

그림 18 1단계의 체크 포인트와 유사 블록

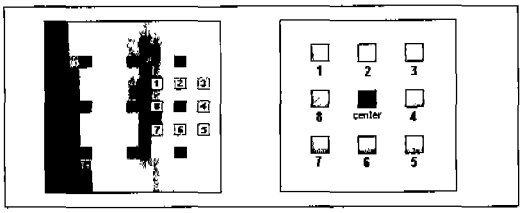

그림 19 2단계의 체크 포인트와 유사 블록

고 결국 1단계에서 최적합점은 [5]가 된다. 따라서 [5]는 2단계의 센터가 된다.

2단계에서는 1단계에서 결정된 센터를 중심으로 2단계의 스텝사이즈만큼 떨어진 거리의 8개 체크포인트를 검사한다. 현재 최적합점은 1단계의 [5] 지점, 즉 2단계의 센터이다. 다시 그림 19와 같은 순서로 체크포인트를 검사해나가면, [2]에서 추정한 블록 유사도가 현재의 최적합점보다 높게 나온다. 2단계 이후 검사하게 되는 [3]~[8]은 최적합점보다 유사도가 높지 않다. 따라서 2단계에서 최적합점으로는 [2]가 결정되고, 이것이 3단계의 센터가 된다. 스텝사이즈는 다시 절반으로 재설정된다. 그림 18과 같이 그림 19에서 오른쪽 그림 또한 2단계의 각각의 체크포인트에서 검사된 16픽셀×16라인 크기 블록의 모양을 실제로 나타낸 것이다.

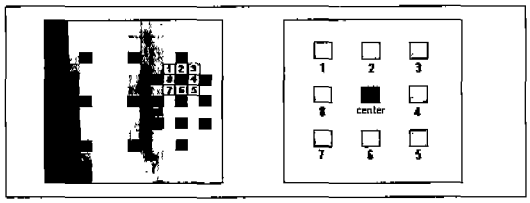

그림 20 3단계의 체크 포인트와 유사 블럭

3단계의 체크포인트는 그림 20과 같다. 그림 20에서 오른쪽 그림은 3단계의 각각의 체크포인트에서 검사된 16픽셀×16라인 크기 블록의 모양을 실제로 나타낸 것이다. 3단계에서는 [6]이 최적합점으로 선택되고 ITSS가

종료된다. 그리고 6과 탐색 영역 센터 간의 수평 수직 방향의 차이값이 움직임 벡터로 저장된다. 즉, 움직임 벡터는 ((이전 프레임의 정합 블록의 수평 성분-현재 매크로블록의 수평 성분), (이전 프레임에서 정합 블록의 수직 성분-현재 매크로블록의 수직 성분))으로 저장된다.

이 과정을 거쳐 발생되는 carphone 영상의 두 번째 프레임의 각 매크로블록 움직임 벡터는 그림 21과 같다. 그림 21에서 움직임 벡터가 표시되어 있지 않은 매크로블록은 영벡터로서 움직임이 없었음을 뜻한다.

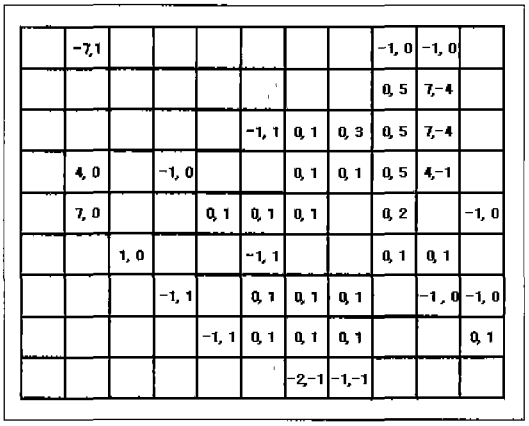

그림 21 carphone의 두 번째 프레임의 움직임 벡터

4. 실험 및 결과

이 장에서는 ITSS의 성능을 평가하기 위하여 수행한 H.263 인코딩 실험에 대하여 설명한다. 4.1절에서는 실험 환경 및 실험 방법 그리고 실험의 대상이 된 비디오 시퀀스에 대하여 설명하며, 4.2절에서는 실험의 결과를 보이고 분석한다.

4.1 실험 환경 및 실험 방법

실험을 수행한 시스템의 사양은 펜티엄 II 266MHz, 64MB이고, 운영체제는 윈도우즈 98이다. 실험 툴은 Visual C++ 6.0 MFC를 사용하였으며, H.263 인코더는 C 언어로 만들어진 TMN 2.0[13]을 이용하였다. 인코딩 결과를 확인하기 위해 사용한 H.263 디코더는 H.263 Player Ver. 1[14]을 이용하였다.

실험 결과에서 화질은 평균 PSNR(Peak Signal-to-Noise Ratio)[15]로써 비교하고, 인코딩 시간은 SAD 함수의 호출 횟수로써 비교하였는데, SAD 함수 호출 횟수는 VC++의 디버깅 모드에서 profile 기능을 이용하여 다음과 같이 측정되었다.

Func Time	%	Func+Child Time	%	Hit Count	Function
4720.184	18.8	4720.184	18.8	14751	_FindHalfPel
2546.090	10.1	2546.090	10.1	89100	_Dct
2363.007	9.4	2363.007	9.4	21420	_idctref
2235.456	8.9	2235.456	8.9	217049	_SAD_Macroblock
1845.230	7.3	1845.230	7.3	150	_ReadImage
1765.262	7.0	1765.262	7.0	89100	_Quant

알고리즘들을 비교하기 위하여 TMN 2.0의 움직임 추정 함수 motionestimation() 중 움직임 탐색 알고리즘만을 바꾸어가며 인코딩하였으며, 각 알고리즘은 모두 9가지의 비디오 시퀀스를 이용하여 실험하였다. 실험시 커맨드 라인에는 QCIF 화일명과 인코딩의 마지막 프레임 번호를 명시하였고, 프레임을 건너뛰지 않도록 -S 옵션을 0으로 주었다. 그 외의 나머지 설정들은 모두 TMN 2.0에서 제공하는 기본 설정을 사용하였다. 즉 무제한 움직임 벡터 모드를 비롯한 H.263의 협상가능한 옵션은 사용하지 않는 베이스라인 모드에서 양자화 인수 10, 참조 프레임을 30fps로 인코딩하였다. 또한 움직임 탐색 영역 파라미터는 7로 주어 탐색 영역의 크기가 15픽셀×15라인이 되도록 하였다.

실험에 사용된 비디오 시퀀스[16]는 모두 QCIF 포맷으로서, 그림 22에서 각 비디오 시퀀스의 첫 번째 프레임을 보이고 있다.

그림 22 비디오 시퀀스의 첫 번째 프레임

각각의 비디오 시퀀스마다 프레임 수가 많은 차이가 있는데, 150 프레임이 전부인 trevor, suzie, miss America는 150 프레임을 모두 실험에 이용하였고 그

외의 나머지 비디오 시퀀스는 300 프레임만을 대상으로 하였다. 비디오 시퀀스들을 비교하는 것이 목적이 아니라, 동일한 비디오 시퀀스에 대하여 각기 다른 알고리즘의 성능 비교이므로 비디오 시퀀스간의 프레임 수의 차이는 문제가 되지 않는다.

표 2 테스트 비디오 시퀀스

비디오 시퀀스	화일 크기	프레임 수	실험 프레임
trevor	5,569	150	150
suzie	5,569	150	150
missAmerica	5,569	150	150
carphone	14,182	382	300
foreman	14,850	400	300
salesman	16,670	449	300
claire	18,340	494	300
grandma	32,299	870	300
motherdaughter	35,678	961	300

trevor, carphone, foreman, salesman 영상을 제외한 나머지 비디오 시퀀스들은 일반적인 화상 회의의 양식대로 입술과 머리 등을 주기적으로 움직이는 것으로 움직임이 거의 없다. 각 비디오 시퀀스의 움직임 특성을 명확히 하기 위하여 간단한 실험을 한 결과 그림 23과 같은 움직임 특성을 간추릴 수 있었다. 특히 carphone과 foreman에서는 움직임 벡터의 변위가 2 이상인 경우가 다른 비디오 시퀀스보다 비율이 높았으며 나머지 비디오 시퀀스에서는 움직임 벡터의 변위가 0인 경우가 상당 부분 차지함을 알 수 있다.

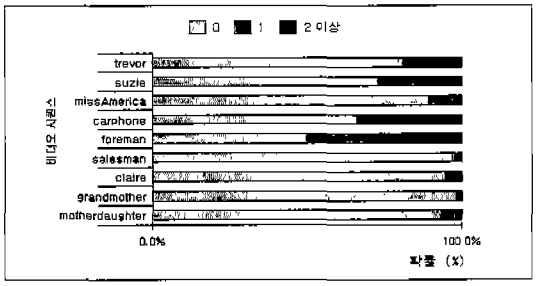

그림 23 움직임 벡터의 분포

4.2 실험 결과

H.263 인코딩시 각 비디오 시퀀스에 대한 FS, TSS, NTSS, ITSS 알고리즘간의 PSNR 변화는 그림 24와 같다. ITSS는 foreman에서 NTSS보다 PSNR의 평균치가 작게 나왔으며, claire 및 missAmerica에서는 FS보다 작게 나왔다. 그러나 그 차이가 매우 근소하였으며, 다른 나머지 비디오 시퀀스에서는 모두 ITSS가 FS와 동일한 PSNR을 나타내었다.

각 비디오 시퀀스에 대한 4가지 알고리즘 적용시의 화일 크기의 변화는 그림 25와 같다. 화일의 크기는 carphone과 foreman에서 ITSS가 다소 크게 나타났으나 이 경우에도 TSS와는 동일하였으며, 나머지 비디오 시퀀스에서는 FS와 화일 크기가 동일하였다.

마지막으로 각 비디오 시퀀스에 대한 4가지 알고리즘 적용시의 SAD 함수 호출 횟수의 변화는 그림 26과 같다. FS 알고리즘은 trevor를 비롯한 150 프레임 실험 비디오 시퀀스에서는 2,722,379번, 나머지의 300 프레임을 실험 대상으로 한 비디오 시퀀스에서는 5,463,029번의 일정 횟수로 SAD 함수를 호출하였으며, TSS의 SAD 함수 호출 횟수는 FS의 약 10% 정도로 급격히 감소되었다. 각 비디오 시퀀스에서 NTSS의 SAD 함수 호출 횟수를 살펴보면 salesman, claire, grandmother, motherdaughter에서의 SAD 함수 호출 횟수가 다른 비디오 시퀀스보다 TSS와 차이가 많음을 알 수 있다. 이는 그림 23의 움직임 벡터의 분포에서 나타나듯이 salesman 등의 비디오 시퀀스가 움직임이 극히 적어서, 움직임이 적은 경우 더 적은 수의 체크포인트 검색만으로 정합 블록을 찾는다는 NTSS 본래의 목적에 가장 적절히 부합되기 때문이다. 이렇게 움직임이 적은 salesman 등의 비디오 시퀀스에서 ITSS는 NTSS보다 적은 SAD 함수 호출 횟수를 나타내는데, NTSS의 2~5%를 감소시켰기 때문에 사실상 거의 유사하다고 할 수 있다. 그러나 나머지 영상에서는 NTSS에 비하여 ITSS의 SAD 함수 호출 횟수가 상당히 줄었으며, 특히 foreman과 carphone에서는 각각 131,939와 91,439가 감소되어 비율로는 ITSS가 NTSS의 SAD 함수 호출

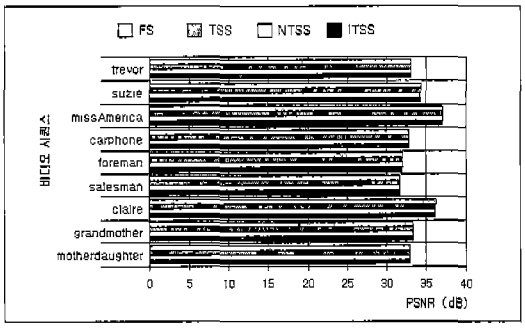

그림 24 PSNR

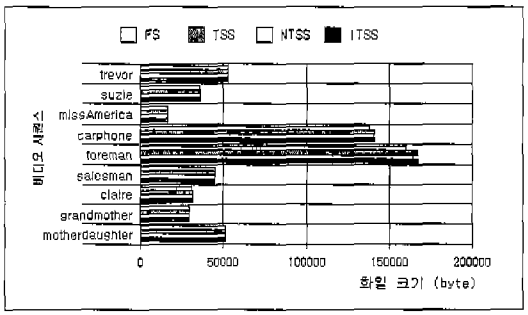

그림 25 파일 크기

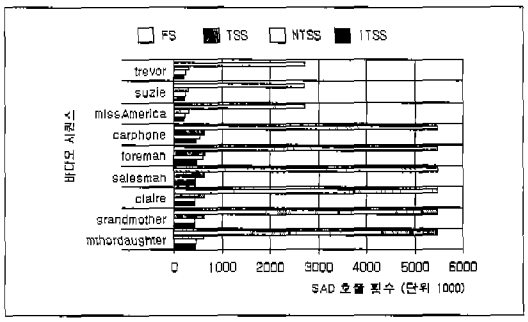

그림 26 SAD 호출 횟수

횟수를 20%까지 감소시켰다. 이것은 foreman과 carphone이 움직임이 다소 크고, 움직임이 큰 경우 33개의 체크포인트를 가지는 NTSS의 단점을 ITSS가 개선하였기 때문에 가능하다. 결론적으로, 각각의 비디오 시퀀스에서 ITSS는 TSS와 NTSS보다 적은 수의 SAD 함수 호출 횟수를 가진다는 것을 알 수 있다.

5. 결 론

이 연구에서는 H.263 동영상 압축에서 전체 인코딩 시간의 상당 부분을 차지하는 움직임 탐색 알고리즘 중 TSS와 NTSS의 단점을 개선함으로써 H.263 인코딩 시간을 단축시키고자 하였다. 즉, 탐색 알고리즘 TSS가 FS에 비해서는 움직임 탐색 시간을 상당히 감소시켰으나, 연속된 프레임간에 움직임이 적은 경우에도 고정된 수의 체크포인트를 검색해야 하는 단점이 있었다. NTSS는 이러한 TSS의 단점을 개선한 것으로 움직임이 적은 경우에 TSS보다 작은 수의 체크포인트만으로 움직임을 검출해낼 수 있었으나 움직임이 적지 않은 경우에는 오히려 TSS보다 더 많은 체크포인트를 요구한다는 문제가 있었다.

이에 본 연구에서는 TSS와 NTSS의 단점을 개선하여, 움직임이 적은 경우 NTSS와 비슷한 움직임 검색 시간이 걸리면서도, 움직임이 적지 않은 경우 TSS와 동일하게 동작하는 ITSS 알고리즘을 제안하였다. ITSS는 적은 수의 비용함수 호출로서 인코딩 시간을 감소시켰으며, 압축된 화일 크기 및 화질은 TSS와 가장 비슷하고, FS, NTSS와도 거의 차이가 나지 않았다. 향후 ITSS를 다른 동영상 압축 표준에서 테스트할 계획이고, 또한 ITSS를 이용하여 저대역폭에서도 움직임 끊김이 적은 화상회의 어플리케이션을 개발할 계획이다.

참 고 문 헌

[1] ITU-T, "Draft H.263: Video Coding For Low Bitrate Communication," ITU, May 1996.

[2] 박일환, 차호정, 김혁만, "RTP 기반의 H.263 스트리밍 시스템 구현", 한국정보과학회 가을학술발표논문집, 25 (2) : 674-676, 한국정보과학회, 1998. 10.

[3] Borko Furht, Joshua Greenberg, Raymond Westwater, Kluwer Academic Publishers, "Motion Estimation Algorithms for Video Compression," 1997.

[4] T. Koga, K. Ilnuma, A. Hirano, Y. Iijima, T. Ishiguro, "Motion compensated interframe coding for video conferencing," Pro. Nat. Telecommun. Conf., New Orleans, pp. G5.3.1-5.3.5, Nov. 1981.

[5] 윤성규, 유환종, 임명수, 임영환, "Three-step 알고리즘을 이용한 H.263 기반의 움직임 측정", 한국정보과학회 가을학술발표 논문집, 26(2):389-391, 한국정보과학회, 1999. 10.

[6] R. Li, B. Zeng, M. L. Liou, "A new three step search algorithm for block motion estimation," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 4, No. 4, pp. 438-442, Aug. 1994.

[7] J. R. Jain, A. K. Jain, "Displacement measurement and its application in interframe image coding." IEEE Trans. on Communications, Vol. COM-29, No. 12, pp.1799-1808, Dec. 1981.

[8] M. Ghanbari, "The cross search algorithm for motion estimation," IEEE Trans. Commun., Vol. COM-38, pp.950-953, July 1990.

[9] L. M. Po, W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 6, No. 3, pp. 313-317, Jun. 1996.

[10] Humaira Nisar, Tae-Sun Choi, "An Advanced Center Biased Three Step Search Algorithm For Motion Estimation," ICME 2000, IEEE, Vol. 1, pp. 95-98, July 2000.

[11] 이수익, 장주익, "화상 회의 영상의 특성을 이용한

H.263 화상 부호화기의 움직임 검색 속도 개선", 한국정보과학회 가을 학술발표논문집, 25(2):656-657, 한국정보과학회, 1998.10.
[12] 유환종, 강의선, 강석찬, 김영환, 김진구, 임영환, "H.263 인코딩 속도 향상 연구", 한국정보과학회 가을학술발표논문집, 26(2):392~394, 한국정보과학회, 1999. 10.
[13] Telenor R&D, H.263 encoder/decoder TMN Ver. 2.0, http://www.nta.no/brukere/kol, June 14, 1996.
[14] Thurdsak Tanakitprapa, http://www.angelfire.com/in/H261/, June 1999.
[15] Yun Q. Shi, Huifang Sun, CRC Press, "Image and Video Compression for Multimedia Engineering," pp. 20-21, pp. 221-249, 2000.
[16] http://isc.stanford.edu/video.html

심 종 채

1987년 경상대학교 전산통계학과(학사). 1987년 ~ 1989년 (주)경원시스템 SE. 1989년 ~ 1998년 호남석유화학(주) 전산실. 1995년 중앙대학교 컴퓨터소프트웨어학과(공학석사). 1998년 ~ 2002년 경상대학교 전자계산학과 박사. 1998년 ~ 현재 남해전문대학 컴퓨터응용정보과 조교수. 관심분야는 데이타베이스, 정보시스템, 멀티미디어, 가상대학

박 영 목

1983년 ~ 1989년 경상대학교 전산통계학과 학사. 1997년 ~ 1999년 경상대학교 산업정보공학과 석사. 2000년 ~ 현재 경상대학교 컴퓨터과학과 박사과정. 관심분야는 소프트웨어 공학(소프트웨어 시험, 신뢰성), Network(TCP/IP, ATM)

성 윤 주

1995년 ~ 1999년 경상대학교 컴퓨터과학과 학사. 1999년 ~ 2001년 경상대학교 컴퓨터과학과 석사. 2001년 ~ 현재 도서출판 비비컴 근무. 관심분야는 멀티미디어 통신, H.263

유 경 종

1992년 ~ 1998년 경상대학교 컴퓨터과학과 학사. 1998년 ~ 2000년 경상대학교 컴퓨터과학과 석사. 2000년 ~ 현재 경상대학교 컴퓨터과학과 박사과정. 관심분야는 MPEG4

박 재 홍

1978년 충북대 수학교육과 졸업(학사). 1980년 중앙대 대학원 전산과(석사). 1989년 중앙대 대학원 전산과(박사). 1983년 ~ 현재 경상대 컴퓨터과학과 교수. 관심분야는 소프트웨어 공학, 테스팅, 소프트웨어 신뢰성

서 영 건

1987년 경상대학교 전산통계학과 학사. 1989년 숭실대학교 대학원 전자계산학과 석사. 1997년 숭실대학교 대학원 전자계산학과 박사. 1989년 ~ 1992년 삼보컴퓨터. 1997년 ~ 현재 경상대학교 컴퓨터교육과 조교수. 관심분야는 멀티미디어 통신, 영상인식