H.264를 위한 효율적인 움직임 벡터 추정 알고리즘

준회원 정 인 철*, 정회원 한 종 기*

An efficient algorithm for motion estimation in H.264

In-Cheol Jeong*, Jong-Ki Han*

요 약

H.264/AVC [1]-[4]는 ITU-T H 264와 ISO/IEC 14496-10 (MPEG-4 Part 10)으로써 승인된 새로운 국제 비디오 압축 표준이다 H 264/AVC는 7가지 가변 블록에 대한 움직임 예측 및 RD[5], 디블로킹 필터[6], 다중 프레임 참조등 여러 가지 방식으로 인해 압축 효율은 높아졌으나 복잡도 또한 훨씬 증가하였다. 그러므로 압축시간 단축을 위한 효율적인 고속 알고리즘이 요구된다 본 논문에서는 H 264/AVC에서 매크로 블록의 모드와 움직임벡터를 결정하기 위한 효율적인 방법을 제안한다 H 264/AVC에서는 7가지 가변블록들에 대한 블록모드와 움직임 벡터를 찾고 그 중에 가장 효율적인 하나의 모드와 움직임 벡터를 선택한다. 이와 달리 본 논문에서는 가장 작은 블록(4×4)의 움직임 벡터를 이용하여 더 큰 블록들의 모드와 움직임벡터를 추정하는 방법을 제안한다. 컴퓨터 실험을 통해 제안하는 방법과 기존의 방법을 비교한 결과 영상의 화질은 거의 유사하면서 전체적인 부호화 시간은 단축되어짐을 알 수 있다.

ABSTRACT

In H 264, 7 modes {16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4} are used to enhance the coding efficiency. The motion vector estimation with 7 modes may require huge computing time. In this paper, to speed up the motion vector estimation procedure while the high image quality remains, we propose a motion vector refinement scheme using the temporary motion vector generated with little computation. The proposed estimation process consists of three phases Mode decision for a 16x16 macroblock, Composing a temporary motion vector, Refinement of the temporary motion vector We demonstrate the effectiveness of the proposed method by computer simulation In the results, the encoding time consumed by the proposed scheme has been reduced significantly while the encoded video quality remains unchanged

Ⅰ. 서 론

H 264/AVC는 ITU-T H.264와 ISO/IEC 14496-10 (MPEG-4 Part 10)으로써 승인된 새로운 국제 비디오 압축 표준이다. H.264/AVC 표준은 기존의 압축 표준들(H.264v2(H 263+) 또는 MPEG-4 Simple Profile)과 비교해서 약 50%이상의 비트율을 감소할 수 있다 또한 H 264/AVC는 7가지 가변 블록에 대한 움직임 예측 및 RD, 인트라 예측, 다중 프레임 참조등을 사용한다. H 264/AVC는 여러 가지 방식들로 인해 기존의 다른 표준들과 비교해서 압축 효율이 높아지고 화질은 좋아졌다. 하지만, 복잡도와 계산량이 매우 증가하였으며 부호화 시간도 많이 증가하였다 그러므로 부호화 시간을 단축시키기 위한 효율적인 고속 알고리즘이 요구된다

본 논문은 H 264/AVC에서 매크로 블록의 모드와 움직임벡터를 결정하기 위한 효율적인 방법을 제안한다. 본 논문에서는 가장 작은 블록모드인 4×4블록모드에서 구해진 16개의 움직임 벡터를 이용하여 8×8, 8×4, 4×8, 4×4모드 중 가장 적합한 하나의 블록모드를 선택한다. 선택된 블록모드가 8×8모드, 8×4모드 그리고 4×8모드 중 하나라면 선택된 블록모드의 움직

* 세종대학교 정보통신공학과 멀티미디어 신호처리 연구실(hjk@sejong ac kr),
논문번호 #030536-1202, 접수일자 2003년 12월 2일

임 벡터를 구하기 위해서 기준벡터(MV_{base})를 계산하여 이를 기준점으로 탐색영역을 ±1로 하여 움직임 예측을 수행한다 그리고 16×16, 16×8, 8×16모드에서 움직임 벡터는 탐색영역을 ±2로 하여 움직임 예측을 수행한다 작은 탐색영역에서 움직임 예측 후 얻어진 움직임 벡터를 최종적인 움직임 벡터로 한다 제안한 방법은 이러한 방식을 이용해서 블록모드를 예측하고 기존의 탐색영역보다 작은 탐색영역에서 움직임 벡터를 구함으로써 속도를 단축시킬 수 있다

본 논문에서는 제안하는 방법이 기존의 방법과 비교하여 영상의 화질은 거의 유사하면서 전체적인 압축시간은 단축시킬 수 있음을 알 수 있다.

Ⅱ. H.264/AVC 표준의 모드 결정 방법

본 절에서는 H.264/AVC 표준의 기본적인 모드 결정 방법을 설명한다. H.264/AVC 표준에서는 그림 1과 같이 7가지 가변 블록모드(16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4) 단위로 움직임 예측이 이루어진다. H.264/AVC에서는 이러한 7개의 블록모드들에 대해서 모두 동일한 방법으로 반복적인 움직임 예측이 수행된다. 움직임 벡터는 SAD(Sum of Absolute Difference)와 MVcost를 계산한 후 이 값이 가장 작을 때의 위치를 계산된다 여기서 MVcost는 움직임 예측하기 전의 이웃한 블록들로부터 계산된 PMV(Predicted Motion Vector)와 움직임 예측 후 얻어진 실제의 움직임 벡터와의 차이값이다. 그리고 Intra모드와 Skip모드를 포함한 모든 블록모드들로부터 모드선택을 위해 RD(Rate-Distortion)cost 계산이 수행되며 이 때, 가장 작은 RDcost를 가진 블록모드가 매크로 블록의 모드로 결정된다 RDcost는 다음의 식에 의해 계산된다 식(1)에서 Distortion은 식(2)에서와 같이 원영상의 매크로 블록과 복원된 매크로 블록

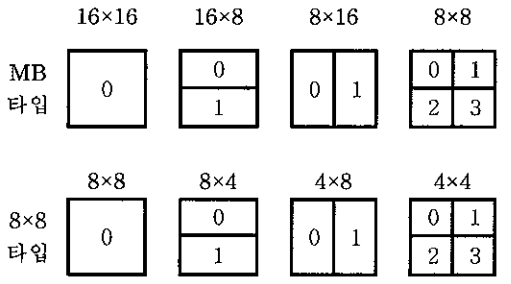


그림 1 움직임 예측을 위한 7가지 가변블록모드

간의 차이 제곱의 합이다. 여기서, $B(i,j)$과 $B'(i,j)$은 각각 현재의 매크로 블록과 복원된 매크로 블록의 (i,j)번째 화소값을 나타낸다 λ는 양자화 파라미터에 의해서 결정되는 상수값이며 rate는 CBP, 움직임 벡터, 블록모드등 매크로 블록의 정보를 표현하는데 필요한 비트수이다 위에서 말한 것과 같이 H.264/AVC는 7개의 가변블록들을 이용하기 때문에 하나의 매크로 블록은 여러 가지 서브블록들로 나누어질 수 있다 세밀하고 움직임이 큰 부분에서는 작은 블록을 이용할 수 있고, 단순하고 움직임이 거의 없는 부분에서는 큰 블록을 사용할 수 있다 그림 2는 하나의 매크로 블록이 여러 가지 서브블록들로 구성되는 하나의 예를 나타낸다

$$RDcost = Distortion + \lambda \times rate \quad (1)$$

$$Distortion = \sum_{i=1}^{16}\sum_{j=1}^{16} | B(i,j) - B'(i,j) |^2 \quad (2)$$

Ⅲ. 제안하는 모드 결정 방법

H.264/AVC 표준은 블록모드를 결정할 때에 7가지의 가변블록들에 대해서 모두 움직임 예측과 RDcost 계산을 수행하며 이 중 하나의 최적화된 모드를 선택하기 때문에 높은 압축율과 좋은 화질을 얻을 수 있다 하지만, 여러 가지 방식들로 인한 복잡도 증가와 반복된 계산 때문에 전체적인 부호화 시간이 크게 증가한다 본 절에서는 증가한 부호화 시간을 단축시키기 위한 방법을 제안한다 제안하는 방법은 먼저 가장 작은 블록모드인 4×4블록모드로 16개의 움직임

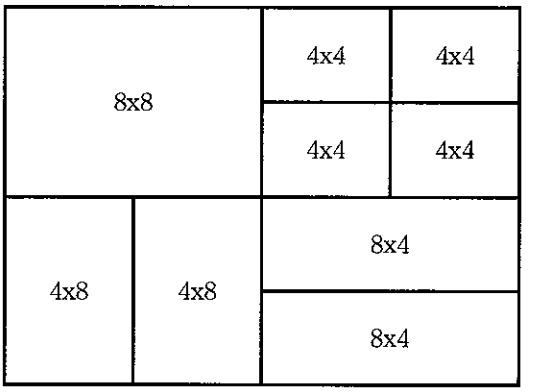


그림 2 매크로 블록내의 서브블록들 예

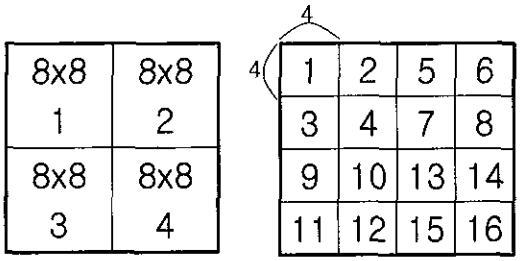


그림 3 매크로 블록의 8×8블록과 4×4블록 분할

벡터들을 구한다. 이렇게 얻어진 16개의 움직임 벡터들을 가지고 8×8, 8×4, 4×8, 4×4단위의 블록모드들 중에 하나의 모드로 결정하는 것이다. 그림 3과 같이 하나의 매크로 블록은 4개의 8×8블록들로 표현될 수 있고 16개의 4×4블록들로 나누어질 수도 있다. 그림 4는 한 개의 매크로 블록에서 4×4블록단위로 얻어진 16개의 움직임 벡터들의 배치도를 나타낸다. 특정 매크로 블록의 최종 모드 결정은 이 16개의 4×4블록의 움직임 벡터들을 이용하여 이루어진다. 그림 5는 제안하는 모드 결정 방법의 전반적인 순서를 나타낸다. 그림에서 Step 2의 과정을 자세히 설명하면 다음과 같다. 특정한 8×8블록에서 cost function 계산을 위해 해당 8×8블록에 있는 4×4블록 4개의 움직임 벡터간의 variance를 이용한다. 이 때, 각각의 variance는 식(3)과 식(4)에서와 같이 계산한다.

$$
\begin{aligned}
var_{12} &= variance(MV_{01}, MV_{02}) \\
var_{34} &= variance(MV_{03}, MV_{04}) \\
var_{13} &= variance(MV_{01}, MV_{03}) \\
var_{24} &= variance(MV_{02}, MV_{04}) \\
var(8\times 8) &= variance(MV_{01}, MV_{02}, MV_{03}, MV_{04})
\end{aligned}
\tag{3}
$$

$$
\begin{aligned}
var(8\times 4) &= var_{12} + var_{34} \\
var(4\times 8) &= var_{13} + var_{24}
\end{aligned}
\tag{4}
$$

식(3)에서 var_{12}는 그림 4의 첫 번째 8×8블록모드에서 수평방향으로 위치한 두 개의 4×4블록의 움직임 벡터 MV_{01}과 MV_{02}간의 variance이다. var_{34}는 첫 번째 8×8블록모드에서 두 개의 4×4블록의 움직임 벡터 MV_{03}와 MV_{04}간의 variance이다. 그리고 var_{13}, var_{24}는 각각 수직방향 MV_{01}와 MV_{03}간의 variance와 MV_{02}와 MV_{04}간의 variance이며 $var(8\times 8)$는 MV_{01}, MV_{02}, MV_{03}, MV_{04}간의 variance이다. 식(4)

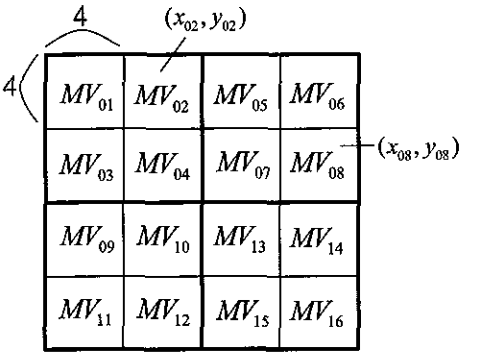


그림 4 4×4블록모드의 움직임 벡터

에서 $var(8\times 4)$은 var_{12}와 var_{34}을 더한 값으로 8×4블록모드로 결정될 때의 움직임 벡터 cost function값을 계산하기 위해서 사용한다. 그리고 $var(4\times 8)$는 4×8블록모드에 대한 cost function값을 계산하기 위해서 사용한다. 예를 들어, 만약 $var(8\times 4)$의 값이 $var(4\times 8)$의 값보다 작다면, 8×4블록모드로 결정될 확률이 높다고 생각할 수 있고, 이 때 8×4블록모드로 선택하게 된다. 이러한 방식으로 3개의 계산값 ($var(8\times 4)$, $var(4\times 8)$, $var(8\times 8)$)을 서로 비교하여 4종류 서브블록모드(8×8, 8×4, 4×8, 4×4) 가운데에 최적의 블록모드를 선택한다. 그림 6은 Step 2를 수행하기 위해 3개의 variance들을 서로 비교하여 각 8×8블록단위로 세부 블록모드를 결정하는 순서를 나타낸 것이다.

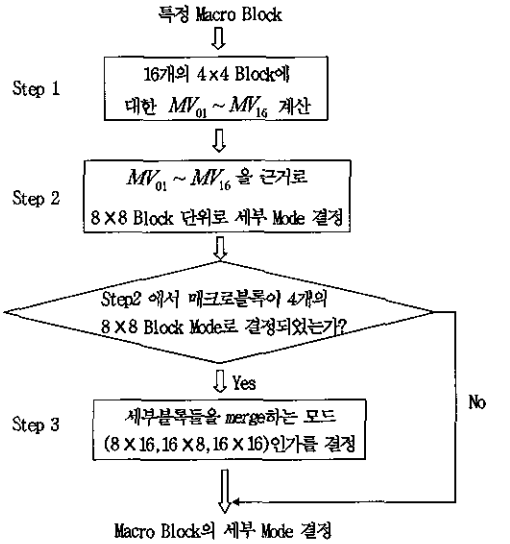


그림 5 제안한 모드 결정 방법의 순서도

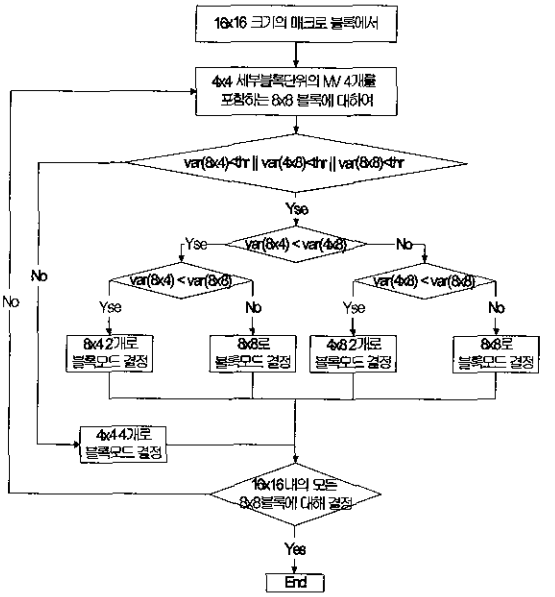


그림 6 8×8블록에서 3개의 variance를 비교하여 세부블록모드를 결정하는 순서도 (Step 2)

$$var(16\times16) = variance(MV_{01}, MV_{02}, \quad , MV_{16})$$
$$var(16\times8) = variance(MV_{01}, MV_{02}, MV_{03}, MV_{04}, MV_{09}, MV_{10}, MV$$
$$+ variance(MV_{05}, MV_{06}, MV_{07}, MV_{08}, MV_{13}, MV_{14}, MV_{15}, MV_{16})$$
$$var(8\times16) = variance(MV_{01}, MV_{02}, MV_{03}, MV_{04}, MV_{05}, MV_{06}, MV$$
$$+ variance(MV_{09}, MV_{10}, MV_{11}, MV_{12}, MV_{13}, MV_{14}, MV_{15}, MV_{16})$$

(5)

식(5)에서 $var(16\times16)$은 그림 4의 특정한 매크로 블록에 있는 모든 16개의 4×4블록의 움직임 벡터들간의 variance이다 $var(16\times8)$과 $var(8\times16)$은 식(5)에서와 같이 8개의 4×4블록의 움직임 벡터간의 variance들을 계산하여 얻어진 각각의 2개의 variance를 더한 값이다 위에서 보인 예와 마찬가지로 만약 $var(16\times8)$의 값이 $var(8\times16)$의 값보다 작다면, 16×8블록모드로 결정될 확률이 높다고 생각할 수 있고, 이 때 16×8블록모드로 선택하게 된다 이러한 방식으로 3개의 계산값 ($var(16\times16)$, $var(16\times8)$, $var(8\times16)$)을 서로 비교하여 4종류의 매크로 블록모드(16×16, 16×8, 8×16, 8×8) 가운데에 최적의 블록모드를 선택한다. 그림 7은 위와 같은 방법을 가지고 Step 3을 수행하기 위해 3개의 variance들을 서로 비교하여 각 16×16블록단위로 세부블록모드를 결정하는 순서를 나타낸다. Step 3에서 사용되는 variance값은 식(5)와 같이 서술된다.

Ⅳ. 제안하는 움직임 벡터 결정 방법

앞 절에서 제안한 방법과 같이 매크로 블록내의 세부모드를 결정한 후에는 해당 세부모드별로 움직임 벡터를 찾아야한다 본 절에서는 이를 위한 효율적인 움직임 벡터 추정 알고리즘을 제안한다. 본 논문에서는 부호화 시간을 효율적으로 단축시키기 위해서 4×4 블록모드에서 얻어진 16개의 움직임 벡터들을 이용하여 기준벡터(MV_{base})를 계산한 후, 이로부터 최종 움직임 벡터를 계산한다. 이 때, 기준벡터란 앞서 얻어진 16개의 움직임 벡터를 이용해 계산한 움직임 벡터들의 중간값이다. 이러한 기준벡터를 기준점으로 해서 기존의 탐색영역보다 작은 탐색영역에서 탐색을 하여 최종 움직임 벡터(MV)을 찾는다 이 때, 작은 서브블록타입(8×8, 8×4, 4×8, 4×4)의 블록모드에서는 탐색영역을 ±1로 하고, 큰 서브블록타입(16×16, 16×8, 8×16)의 블록모드에서는 탐색영역을 ±2로 하여 수행한다. 이 탐색영역의 크기는 컴퓨터 실험을 통해 경험적으로 결정하였다 그림 8은 이러한 방법의 예를 나타낸다. 그림에서 16×16 블록타입의 블록모드가 결정되었다고 할 때, 기준벡터(MV_{base})가 (5, 5)의 위치에 있다고 가정하면 이 위치를 기준점으로 하여 탐색영역을 ±2로 탐색하여 움직임 벡터를 찾는다. 이와 같은 과정을 통하여 적은 계산량으로도 정확한 움직임 벡터를 추정할 수 있다.

다음은 각 세부모드별 기준벡터를 표현한다. 식(6)~(11)에서 알 수 있듯이, 기준벡터는 각 세부모드에 포함된 4×4단위의 움직임 벡터들의 평균을 취하거나 Median 필터를 통과해서 얻는다

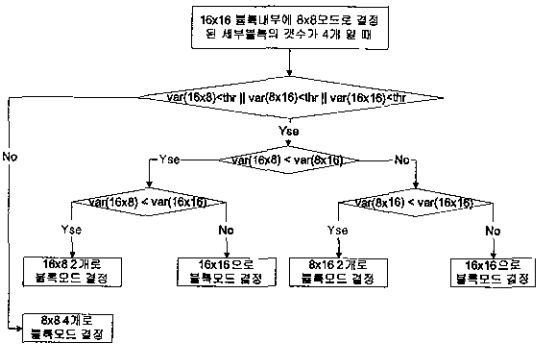


그림 7 16×16 매크로블록에서 3개의 variance를 비교하여 모드를 결정하는 순서도 (Step 3)

$$8\times4\ Mode\ block\ \ MV_{base1}=Averge(MV_{01},MV_{02})$$
$$8\times4\ Mode\ block\ \ MV_{base2}=Averge(MV_{03},MV_{04}) \quad (6)$$

$$4\times8\ Mode\ block\ \ MV_{base1}=Averge(MV_{01},MV_{03})$$
$$4\times8\ Mode\ block\ \ MV_{base2}=Averge(MV_{02},MV_{04}) \quad (7)$$

$$8\times8\ Mode\ block\ \ MV_{base}=Median(MV_{01},MV_{02},MV_{03},MV_{04}) \quad (8)$$

$$16\times8\ Mode\ \ MV_{base1}=Median$$
$$(MV_{01},MV_{02},MV_{03},MV_{04},MV_{05},MV_{06},MV_{07},MV_{08})$$
$$16\times8\ Mode\ \ MV_{base2}=Median \quad (9)$$
$$(MV_{09},MV_{10},MV_{11},MV_{12},MV_{13},MV_{14},MV_{15},MV_{16})$$

$$8\times16\ Mode\ \ MV_{base1}=Median$$
$$(MV_{01},MV_{02},MV_{03},MV_{04},MV_{09},MV_{10},MV_{11},MV_{12}) \quad (10)$$
$$8\times16\ Mode\ \ MV_{base2}=Median$$
$$(MV_{05},MV_{06},MV_{07},MV_{08},MV_{13},MV_{14},MV_{15},MV_{16})$$

$$16\times16\ Mode\ \ MV_{base1}=Median$$
$$(MV_{01},MV_{02},MV_{03},MV_{04},MV_{05},MV_{06},MV_{07},MV_{08} \quad (11)$$
$$,MV_{09},MV_{10},MV_{11},MV_{12},MV_{13},MV_{14},MV_{15},MV_{16})$$

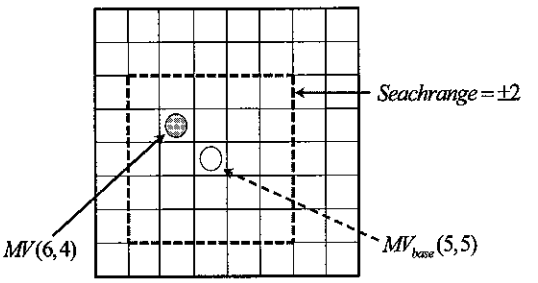


그림 8 움직임 예측에 기준점이 되는 기준벡터와 탐색영역 ±2에서 예측한 움직임 벡터

식(6), 식(7)은 8×4블록모드와 4×8블록모드의 기준벡터를 구하는 것을 나타낸다. 그림 1에서와 같이 하나의 8×8블록에서 8×4블록모드는 2개의 8×4블록으로 나누어지며 식(6)에서와 같이 2개의 8×4블록의 기준벡터를 구한다. 그림 4를 참조해서 설명하면, 첫 번째 8×4블록의 기준벡터는 4×4블록의 첫 번째 움직임 벡터 MV_{01}와 두 번째 움직임 벡터 MV_{02}간의 평균값으로 하고, 두 번째 블록의 기준벡터는 세 번째 움직임 벡터 MV_{03}와 네 번째 움직임 벡터 MV_{04}간의 평균값으로 한다. 그리고 만약 8×8블록모드가 선택된다면 기준벡터는 식(8)과 같이 하나의 8×8블록에 포함되는 4×4블록단위의 4개의 움직임 벡터들의 중간값으로 한다. 더 큰 블록 단위(16×16, 16×8, 8×16)의 블록모드를 선택할 경우에는 4×4블록의 움직임 벡터 16개를 가지고 알고리즘을 수행한다. 만약 16×8블록모드 또는 8×16블록모드가 선택된다면 식(9)과 식(10)에서와 같이 4×4블록의 움직임 벡터 8개의 중간값을 기준벡터로 한다. 16×16블록모드가 선택된다면 이것의 기준벡터는 식(11)과 같이 16개의 움직임 벡터간의 중간값으로 한다. 하지만, 4×4블록모드가 선택된다면 움직임 벡터는 이미 계산된 16개의 움직임 벡터를 그대로 사용한다. 그림 9는 제안하는 움직임 벡터 결정 방법의 순서를 나타낸 것이다.

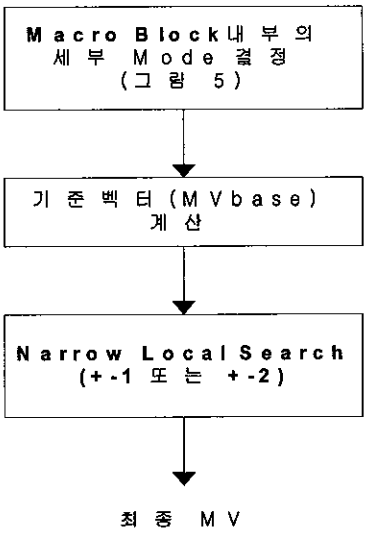


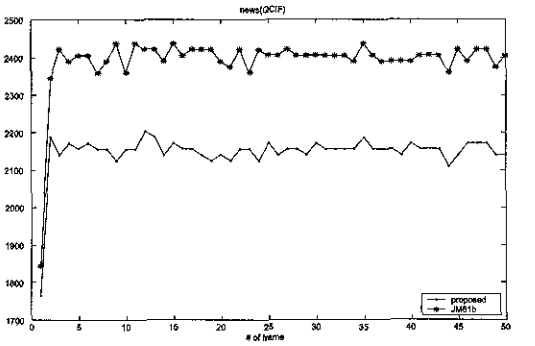
그림 9 제안하는 움직임 벡터 결정 방법의 순서도

V. 실험 결과

본 실험은 H.264/AVC 표준의 베이스라인 프로파일에서 jm61b 소스코드를 이용하였고, 펜티엄-4 2GHz PC에서 50장의 Coastguard 영상, Akiyo 영상 그리고 News QCIF(176×144)영상을 가지고 수행하였다. 각 영상은 첫 번째 영상만 Intra 프레임으로 하고 나머지 프레임들은 모두 P 프레임으로 하였다. 모의실험에서의 기본 조건은 가변블록크기(16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4)의 움직임 예측 및 움직임 보상, ±16의 탐색 영역, 4×4 정수형 변환, RD등을 사용하였다.

그림 10은 기존 방법인 jm61b와 제안하는 방법을 사용했을 때 각 프레임들을 코딩하는 시간(ms)을 비

교해서 나타낸 그래프이다. 그림에서 세 종류 영상(Coastguard, News와 Foreman)의 경우 모두 제안하는 방법이 jm61b와 비교해서 각 프레임마다 0.2 ~ 0.3초정도 빠르게 부호화됨을 볼 수 있다. 그림 11은 jm61b와 제안하는 방법을 PSNR측에서 비교한 것이다. 실험한 세 영상의 경우에서, 제안하는 방법의 PSNR이 기존 방법의 결과와 비교해서 PSNR 손실이 대략 0 1 ~ 0.15dB정도에 불과하다는 것을 알 수 있다. 이는 제안하는 기법을 사용함으로써 화질의 차이는 무시할 수 있을 정도로 작으면서 시스템의 복잡도를 줄일 수 있음을 설명한다 그리고 그림 12는 jm61b와 제안하는 방법을 픽처당 비트의 수를 비교해서 나타낸 것이다. 이 결과에서 세 영상의 경우 모두 제안하는 방법이 jm61b와 비교해서 약간의 비트율은 증가하였다. 하지만, 부호화 속도가 많이 단축되었음을 볼 때, 본 논문에서 제안한 방법이 기존의 방법과 비교해서 우수하다는 것을 알 수 있다



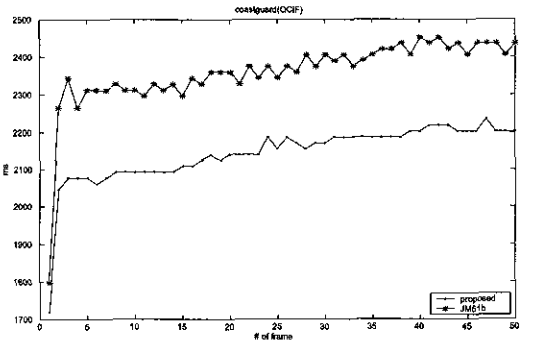
(c) News(QCIF) 영상에서 시간의 비교

그림 10

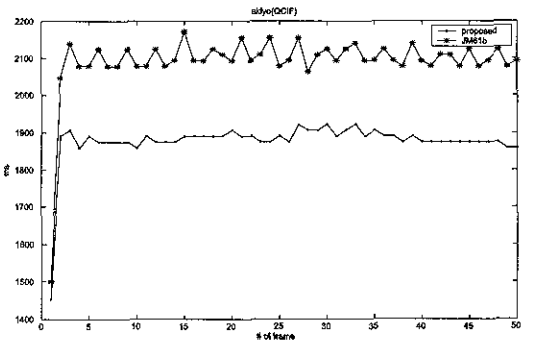
(a) Coastguard 영상에서의 JM61b와 제안한 방법과의 시간의 비교 그래프

(b) Akiyo 영상에서의 JM61b와 제안한 방법과의 시간의 비교 그래프

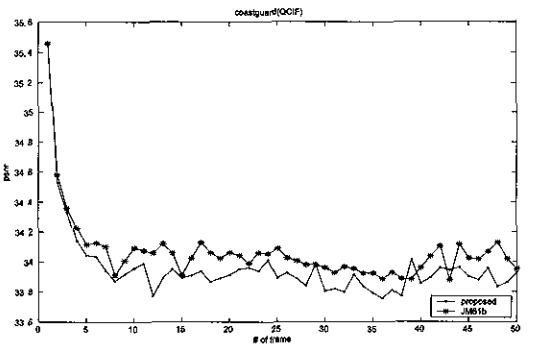
(c) News 영상에서의 JM61b와 제안한 방법과의 시간의 비교 그래프



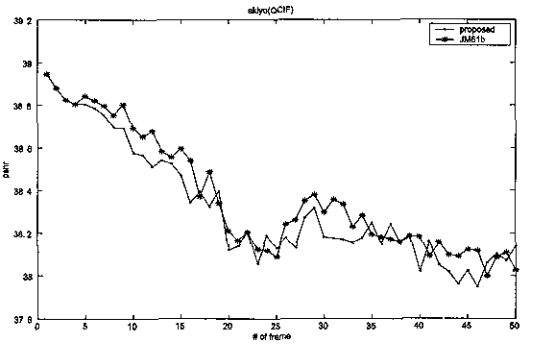
(a) Coastguard(QCIF) 영상에서 시간의 비교



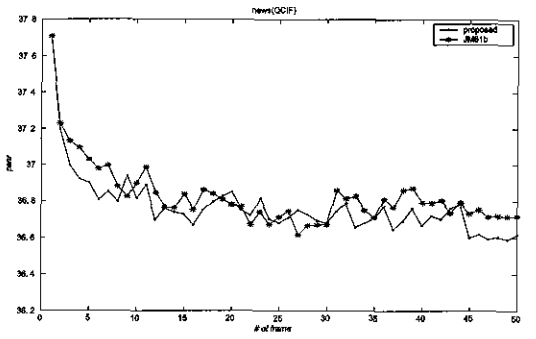
(b) Akiyo(QCIF) 영상에서 시간의 비교



(a) Coastguard(QCIF) 영상에서 PSNR 비교

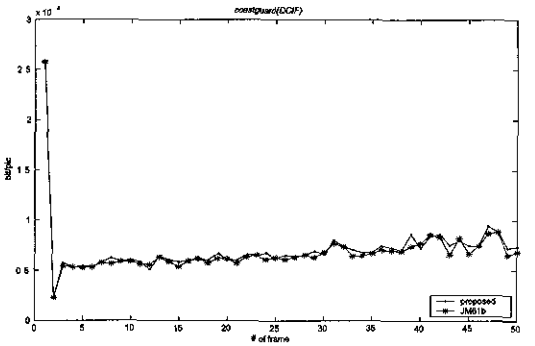


(b) Akiyo(QCIF) 영상에서 PSNR 비교

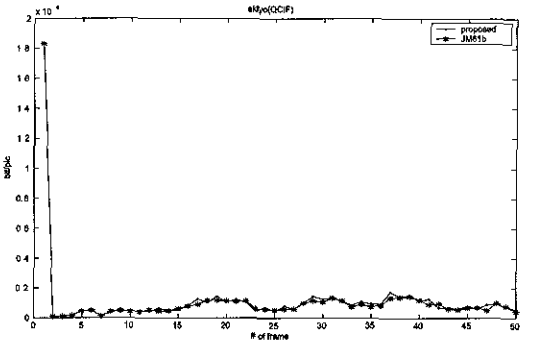


(c) News(QCIF) 영상에서 PSNR 비교

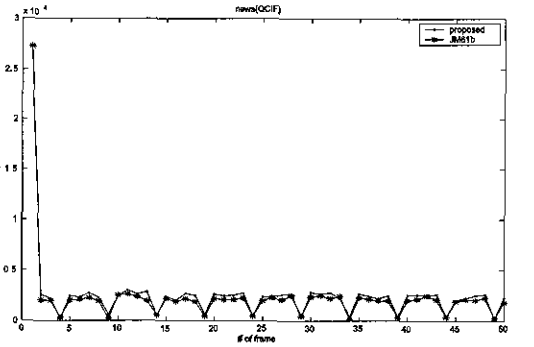
그림 11
(a) Coastguard 영상에서의 JM61b와 제안한 방법과의 PSNR 비교 그래프
(b) Akiyo 영상에서의 JM61b와 제안한 방법과의 PSNR 비교 그래프
(c) News 영상에서의 JM61b와 제안한 방법과의 PSNR 비교 그래프



(a) Coastguard(QCIF) 영상에서 bit/pic 비교



(b) Akiyo(QCIF) 영상에서 bit/pic 비교



(c) News(QCIF) 영상에서 bit/pic 비교

그림 12
(a) Coastguard 영상에서의 JM61b와 제안한 방법과의 bit/pic 비교 그래프
(b) Akiyo 영상에서의 JM61b와 제안한 방법과의 bit/pic 비교 그래프
(c) News 영상에서의 JM61b와 제안한 방법과의 bit/pic 비교 그래프

VI. 결 론

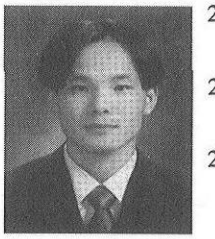
H.264/AVC는 새로운 압축 표준으로 기존의 영상 압축표준들과 비교하여 높은 압축률과 좋은 화질을 제공하고 있다. 하지만, 여러 가지 압축방법들을 사용하였기 때문에 계산이 매우 복잡해졌고 압축시간도 많이 증가하였다. 본 논문은 이러한 압축시간을 줄이기 위한 속도 단축 방법을 제안하였다. 모의실험결과 제안하는 방법이 jm61b 소스코드보다 화질과 비트율은 거의 비슷하게 유지하면서 전체적인 부호화 시간은 약 15%정도 단축되었음을 알 수 있었다. 앞으로 H.264/AVC의 실시간 부호기를 빠르고 효율적으로 만드는데 있어서 좋은 참고자료가 될 것이다.

참 고 문 헌

[1] Thomas Wiegand, Joint Final Committee Draft of Join Video specfication (ITU-T Rec. H.264 | ISO/IEC 1496-10 AVC), JVT-G050, March 2003.
[2] Thomas Wiegand, Gary J Sullivan, and Ajay Luthra, "Overview of the H.264 / AVC Video Coding Standard", IEEE Trans. Circuits Syst. Video Technol, July 2003.
[3] Ralf Schafer, Thomas Wiegand, and Heiko Schwarz, "The Emerging H.264 /AVC

Standard", EBU Technical Review, Jan. 2003.
[4] "Emerging H.264 Standard: Overview and TMS320DM642-Based Solutions for Real-Time Video Applications", UB Video Inc. www.ubvideo.com
[5] Gary J.Sullivan and Thomas Wiegand, "Rate-Distortion Optimization for Video Compression", IEEE Signal Processing Magazine, pp. 23-50, Nov. 1998.
[6] P. List, A. Joch, J. Lainema, G. Bjøntegaard, M. Karczewicz: "Adaptive Deblocking Filter," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp. 614-619, July 2003.
[7] "H.264/MPEG-4 AVC Video Compression Tutorial", LSI Logic Corporation, January 2003.
[8] http://bs.hhi.de/~suehring/tml
[9] http://www.vcodex.com

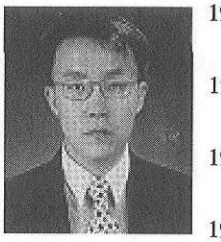
정 인 철 (In-Cheol Jeong) 준회원



2002년 2월 : 세종대학교 정보통신공학과 학사
2004년 2월 : 세종대학교 정보통신공학과 석사
2004년 3월~현재 : 미디어코러스(주) 연구원

<관심분야> MPEG-2, MPEG-4, H.264, 영상처리 및 압축

한 종 기 (Jong-Ki Han) 정회원



1992년 2월 : KAIST 전기 및 전자공학과 학사
1994년 2월 : KAIST 전기 및 전자공학과 석사
1999년 2월 : KAIST 전기 및 전자공학과 박사
1999년 3월~2001년 8월 : 삼성전자 DM(디지털 미디어) 연구소 책임 연구원
2001년 9월~현재 : 세종대학교 정보통신공학과 조교수

<관심분야> MPEG-2, MPEG-4, H.264, Transcoder, SVC, 음성신호 처리 및 압축, 영상신호 처리 및 압축.