

	(19) 대한민국특허청(KR) (12) 공개특허공보(A)	(11) 공개번호 10-2008-0081407 (43) 공개일자 2008년09월10일
(51) Int. Cl. <i>H04N 7/24</i> (2006.01) (21) 출원번호 10-2007-0021299 (22) 출원일자 2007년03월05일 심사청구일자 없음	(71) 출원인 경희대학교 산학협력단 경기도 용인시 기흥구 서천동 1 경희대학교 국제 캠퍼스내 (72) 발명자 서덕영 경기 성남시 분당구 수내동 푸른마을 벽산아파트 201-202 이용현 경기 수원시 권선구 구운동 삼환아파트 3동 209호 박광훈 경기 성남시 분당구 분당동 45번지 동아빌라 B동 302호 (74) 대리인 양문옥	

전체 청구항 수 : 총 1 항

(54) 하이브리드 다시점 및 계층화 비디오 코딩 방법과 장치

(57) 요약

1. 청구범위에 기재된 발명이 속한 기술분야

본 발명은 계층화 비디오 코딩과 다시점 비디오 코딩을 결합한 복합형 비디오 코딩 방법 및 장치에 관한 것임

2. 발명이 해결하려고 하는 기술적 과제

본 발명에서는 계층화 비디오 코딩(SVC 또는 Scalable Video Coding)과 다시점 비디오 코딩(MVC 또는 Multi-view Video Coding)을 결합한 복합형 비디오 코딩 (HMSVC or Hybrid Multiview and Scalable Video Coding) 방법을 제시한다. 다시점 비디오 코딩은 여러 가지 각도에서 획득한 비디오 영상을 모두 전송하여 부호화, 전송, 복호화에 매우 많은 자원이 이용된다. 본 발명에서는 디스플레이 되는 뷰는 충분한 품질을 제공하고 다른 뷰에 대해서는 제한된 품질만 유지하는 방법을 제안한다. 따라서, 제한된 품질로 부호화되는 여러 뷰들은 MVC를 이용하여 부호화 및 복호화되고 선택된 뷰는 SVC를 이용하여 향상된 계층을 디스플레이할 수 있게한다. 여기서 품질의 향상은 공간적 계층화 (spatial scalability), 시간적 계층화 (temporal scalability), 품질 계층화 (quality scalability), 또는 이들 방법을 조합한 방법을 이용하여 가능하게 한다. 이 같은 계층화 방법은 일 실시예로 JVT SVC에서 제공하는 계층화 방식을 이용할 수 있다.

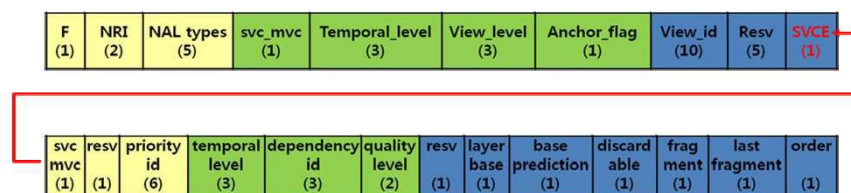
3. 발명의 해결방법의 요지

본 발명은 MVC를 이용하여 다수의 뷰를 동시에 전송하고 선택된 뷰는 고품질로 디스플레이할 수 있게하는 방법 및 장치를 이용하여 적은 자원을 이용하여 다시점 비디오 서비스가 가능하게 하는 방법을 포함함.

4. 발명의 중요한 용도

본 발명은 다시점 비디오 서비스에서 자원의 절약 및 응용의 다양성을 위해 이용된다.

대 표 도 - 도8



특허청구의 범위

청구항 1

계층화 비디오 코딩(SVC)과 다시점 비디오 코딩(MVC)을 결합한 하이브리드 다시점 및 계층화 비디오 코딩 장치를 이용하여 효율적으로 서비스하는 방법.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

- <12> 현재 표준화가 진행중인 다시점 비디오 코딩(Multi-view Video Coding; MVC)은 기존의 동영상 국제표준인 MPEG-4 part 10 Advanced Video Coding (AVC; H.264)을 기반으로하여 코딩 성능을 향상 시키고 있다. 시간 방향으로 JSVC (Joint Scalable Video Coding)에서 시간적 스케일러빌리티(temporal scalability)를 지원하기 위해 수행되는 방법인 계층적인 B-픽처(hierarchical B-pictures) 코딩을 수행하고 또한 뷰간(inter-view)의 예측을 수행하여 코딩의 향상을 이루고 있다.
- <13> 도 1은 현재 표준화가 진행중인 다시점 비디오 코딩에서 8개의 뷰가 존재할 경우의 예로써 시간 방향의 GOP (Group of Pictures)의 크기가 8일 경우의 예측 구조를 보여준다. S0, S1, S2, S3, ..., S7은 각각 하나의 뷰(view)를 나타내고 T0, T1, T2, T3, ..., T100은 시간방향의 영상을 나타낸다.
- <14> 보통 모든 뷰는 서로 같은 시간대에서 참조관계를 가지고, 다른 뷰를 참조하지 않고 독립적으로 복호화가능한 뷰를 기본 뷰 (base view)라고 한다. 각각의 뷰에 속한 프레임들은 시간적으로 서로 참조관계를 가지며, 랜덤 액세스와 에러 전파 끊기를 위해 주기적으로 앵커를 삽입한다. 여기서 앵커란 시간적으로 이전의 데이터에 독립적으로 복호화할 수 있는 프레임을 말한다. 그림에서는 S0가 기본 뷰이고, T0, T8, T96에 있는 프레임들이 앵커 프레임들이다.
- <15> 도 1에서 각 뷰의 시간방향의 코딩에서는 계층적인 B-픽처 구조를 사용하여 예측을 수행하고 있고, 각 뷰의 첫 번째 시간(T0)의 영상들과 그 이후로 시간 방향으로 8 프레임씩 계속해서 떨어져 있는 영상(즉, 시간 방향의 GOP 크기만큼 계속해서 떨어져 있는 영상)들 (즉, T8, T16, T24,...)은 이웃하는 뷰로부터의 예측만을 수행하고 있다. 즉 S2 뷰는 S0 뷰로부터, S1 뷰는 S0 뷰와 S2뷰로부터, S4 뷰는 S2 뷰로부터, S3 뷰는 S2 뷰와 S4 뷰로부터, S6 뷰는 S4 뷰로부터, S5 뷰는 S4 뷰와 S6 뷰로부터, S7 뷰는 마지막 뷰이기 때문에 S6 뷰로부터 예측을 수행한다. 그리고 그 이외의 매 두번째 뷰(즉, S1, S3, S5, S7)에서는 이웃하는 뷰로부터의 예측을 시간 방향의 예측과 함께 수행한다. 즉 S1 뷰는 S0 뷰와 S2 뷰로부터, S3 뷰는 S1 뷰와 S4 뷰로부터, S5 뷰는 S3 뷰와 S6 뷰로부터 예측을 수행한다.
- <16> 이 같이 8개의 뷰가 동시에 복호화가 되므로 매우 많은 계산량을 요구하고 있다. 본 발명의목적은 선택된 뷰 이외의 뷰는 품질을 매우 떨어뜨려서 계산량과 전송량을 줄이는 것이다. 예를 들어, 8개의 CIF 영상(352X288)을

$$\frac{352}{16} \times \frac{288}{16} \times 30 \times 8 = 95,040$$

프레임을 30Hz로 복호화한다면, 1초에

개의 매크로블럭을 복

호화해야 하지만, 8개의 뷰는 QCIF 영상(176X144)로 하고, 하나의 뷰만 CIF 영상으로 한다면

$$\left(\frac{176}{16} \times \frac{144}{16} \times 8 + \frac{352}{16} \times \frac{288}{16} \right) \times 30 = 35,640$$

개의 매크로블럭만 복호화하므로

계산량이 약 3/8으로 줄어든다.

발명이 이루고자 하는 기술적 과제

- <17> 본 발명의 목적은 디스플레이용으로 수신측에서 선택된 뷰 이외의 뷰는 품질을 매우 떨어뜨려서 계산량과 전송량을 줄이는 것이다. 예를 들어, 8개의 CIF 영상(352X288)을 프레임율 30Hz로 복호화한다면, 1초에

$$\frac{352}{16} \times \frac{288}{16} \times 30 \times 8 = 95,040$$

(176X144)로 하고, 하나의 뷰만 CIF 영상으로 한다면

$$\left(\frac{176}{16} \times \frac{144}{16} \times 8 + \frac{352}{16} \times \frac{288}{16} \right) \times 30 = 35,640$$

개의 매크로블럭만 복호화하므로 계산량이 약 3/8으로 줄어든다. 즉, 3개의 CIF급 비디오를 복호화하는 것과 계산량이 비슷하다. 만일 기본 계층을 QCIF로 하고, 향상계층을 4CIF로 한다면 계산량은 약 3/16으로 줄어들어서 4CIF 하나를 복호화 하는 것에 서 50%만 계산량을 추가하면 된다.

발명의 구성 및 작용

- <18> 본 발명에 의하면 기본계층과 향상계층간 계층화 방식은 공간적 계층화 (spatial scalability), 시간적 계층화 (temporal scalability), 품질 계층화 (quality scalability), 또는 이들 방법을 조합한 방법 모두 가능하다. 그러나, 본 발명에서는 설명을 간단하게 하기 위해서 일 실시예로 하나의 공간적 계층화 계층을 사용하는 경우에 대해서만 설명한다.
- <19> 다시점 비디오 서비스에서 동시에 복호화되고 있는 뷰는 다수이지만 디스플레이되는 뷰는 하나라는 점에 착안한다. 랜덤 액세스(뷰 스위칭) 지연을 얼마나 허용하는가에 따라 계산량, 버퍼량 등 자원을 절약할 수 있는 방법이 있을 수 있다.
- <20> 예를 들어 도 2에서는 8개의 QCIF와 하나의 CIF가 동시에 복호화되고 있는 경우를 보여준다. 이때 8개의 QCIF는 MVC 방식에 의해 복호화되고, 그중 하나의 QCIF는 CIF의 공간적 계층 복호화의 기본 계층이 된다. CIF 영상이 복원되는 뷰가 디스플레이 되며 display_view라고 정의한다. 이로써 동시에 8개의 CIF를 MVC 복호화하는 것에 비해서 자원을 절약할 수 있지만 랜덤액세스 지연, 복잡도의 증가 등 문제가 발생한다. 본 발명에서는 이 같은 문제를 최소화하는 부호화, 복호화 방식을 제시하고, 필요한 선택스를 규정한다. 이와 같이 다시점 비디오 복호화와 계층화 비디오 복호화를 동시에 이용하는 복호화 방식을 HMSVC(Hybrid Multiview and Scalable Coding)이라고 정의한다.
- <21> 만일 display_view를 복호화하는데 필요한 최소의 데이터만 전송한다면 자원을 절약할 수 있을 것이다. 랜덤 액세스시 도 3과 같이 4개의 QCIF 스트림과 하나의 CIF가 복호화되는 경우 QCIF간에는 MVC로 동작하고 있다고 하자. 참조관계는 그림에서와 같이 S1과 S2가 S0를 참조하고, S3는 S2를 참조한다고 하자. CIF는 디스플레이 되는 뷰, 즉, display_view에서만 복호화된다고 하자. 이때, 뷰 2가 display_view이면 S0, S2의 QCIF 데이터와 뷰 2의 CIF 데이터만 전송되어 복호화되면 된다. 이로써 계산량, 전송율, 복호화 버퍼메모리 등 자원을 절약할 수 있다. 만일 뷰 2에서 뷰 1으로 스위칭을 사용자가 요청하였을 때, 뷰 1을 복호화하고 있지 않았기 때문에 다음 앵커까지 (다음 GOP 시작까지) 복호화가 불가능하다.
- <22> 응용 서비스는 다음과 같은 항목별로 분류할 수 있다.
- <23> 1. 실시간으로 동시에 부호화/복호화되는가? (예) 화상전화, 화상회의, 스포츠중계
- <24> 2. 멀티캐스트되므로 개개인의 피드백 요구를 서버가 들어줄 수 없는가? (예) 스포츠중계
- <25> 3. 이미 인코딩되어있는 콘텐츠를 스트리밍하는 경우인가? (예) VOD
- <26> 랜덤 액세스 지연, 영상 품질, 자원의 절약은 서로 밀접한 관계를 가지고 있으며 이러한 관계는 응용 서비스의 형태에 따라 다르게 나타날 수 있다.
- <27> 1. 실시간 부호화/복호화 (realtime encoding/decoding)
- <28> 실시간으로 동시에 부호화/복호화되며 1:1 송신인 경우는 단순히 하나의 AVC 또는 SVC로 구현할 수 있다. 뷰 스위칭도 단순히 부호화기의 입력영상이 다른 뷰에서 입력되면 된다. 스위칭시에도 영상간 참조모드가 차별없이 적용될 수 있다. 따라서 이 경우에는 MVC가 사실상 필요없게 된다.
- <29> 2. 스트리밍 (Streaming encoded video)
- <30> 이미 부호화된 다시점 시퀀스를 스트리밍하는 경우이다. 실시간 부호화/복호화하면서 멀티캐스트하는 경우에도

다음과 같은 시나리오가 적용된다. 선택된 뷰에 대해서만 향상계층이 전송된다. 기본 계층은 모든 뷰가 다 복호화되며, display_view에 대해서만 향상계층이 전송된다. 뷰 스위칭시에 기본 계층은 바로 뷰 스위칭이 일어나지만, 향상 계층에서는 인트라 리프레쉬 (Intra 또는 Enhanced Intra)가 일어나야 뷰 스위칭이 가능하다.

- <31> 품질의 저하가 없어야 하고, 뷰스위칭 지연은 허용되면, 도 6과 같이 향상 계층 인트라 리프레쉬 이전까지 그전 뷰를 계속 디스플레이한다. 뷰 스위칭 요구와 실제 스위칭 중간 기간동안 이전 프레임의 향상 계층 데이터가 전송되어야 한다.
- <32> 만일 품질의 저하를 허용하고 뷰 스위칭 지연을 허용하지 않으면 도 5와 같이 기본 계층을 바로 스위칭하고, 향상 계층은 기본 계층을 업샘플링하여 디스플레이 한다. 뷰 스위칭 요구와 실제 스위칭 중간 기간동안 그전 뷰의 향상 계층 데이터는 전송될 필요없다.
- <33> 인트라 리프레쉬를 자주하면 코딩효율이 떨어지므로, 도 7과 같이 중간중간에 SI (Switching Intra) 또는 SP (Switching Predictive)를 사용하여 코딩 효율을 어느 정도 유지하면서 뷰 스위칭 지연을 줄이는 것이 바람직하다.
- <34> HMSVC 복호화 과정
- <35> 1. MVC 복호화
- <36> 기본계층에서는 기존의 MVC에서 바뀌는 사항이 없다. 다만, 현재 디스플레이 되는 뷰를 인식하는 정보가 복호화측에서 필요하다. 일 실시예로 display_view라고 하자. 부호화기에서 부호화해야하는 측 또는 전송단에서도 display_view를 알면 display_view를 복호화하는데 필요한 데이터만 선택적으로 복호화측에 제공할 수 있다. 예를 들어, 도 1에서 뷰 4가 display_view라면 뷰 1과 뷰 3은 부호화할 필요가 없고 부호화하였더라도 전송할 필요없다. 그러나, GOP 단위로 랜덤엑세스하는 것을 허용하지 않고, 만일 완전히 즉각적인 랜덤엑세스를 원한다면 모든 뷰를 동시에 복호화하여야 한다.
- <37> 레퍼런스 픽처 리스트 관리 : 기존의 MVC 뷰간 예측구조를 따를 경우, 레퍼런스 픽처 리스트 관리는 기존의 방법과 동일하다
- <38> 2. SVC 복호화
- <39> 여러 개의 뷰중에서 선택된 뷰만 향상계층 디코딩을 수행한다. 향상 계층은 1개 또는 전부를 선택할 수 있다. 향상 계층간에 뷰간 참조는 기본 계층의 뷰간 참조와 구조를 달리 할 수 있다. 일 실시예로 향상 계층에서는 뷰간 참조를 하지 않는 것이 바람직하다.
- <40> 뷰 스위칭 : 뷰 스위칭시 향상계층은 가장 가까운 인트라 리프레쉬 (intra refresh) 이후부터 복호화 가능하다. 향상 계층의 뷰 스위칭(랜덤엑세스) 지연을 줄이기 위해서는 SP(switching predictive) 또는 SI(switching intra)방식을 사용하는 것이 바람직하다.
- <41> 레퍼런스 픽처 리스트 관리 : 다른 뷰의 기본계층의 복호화된 영상을 향상 계층의 참조 영상으로 사용할 수 있다. 이를 위하여 레퍼런스 픽처 구조에 대한 선택스가 필요하다.
- <42> HMSVC를 위한 선택스
- <43> 1. NAL 헤더
- <44> 수신측에서 하나의 뷰만 SVC 향상 계층이 사용되는 경우에는 display_view를 정하면, MVC 헤더를 전송하지 않더라도 자동적으로 display_view의 SVC 향상 계층임을 인식할 수 있게된다. 그러나, 여러 개의 뷰에 대해서 SVC NALU을 전송하는 경우에는 뷰와 계층을 각각 인식할 수 있어야 한다.
- <45> ■ **Concatenated MVC/SVC NAL header**
- <46> MVC와 SVC의 NAL 헤더를 붙여서 헤더를 만드는 방식이다. SVC NALU는 선택적으로 존재하므로 기본적으로 MVC를 가지고 있는 것이 좋다. 이 경우에 MVC 기본 계층 NALU (NAL Unit)인지 SVC NALU인지 구분은 MVC NAL 헤더에서 이루어져야 한다. 이를 위하여 일 실시예로 MVC NAL 헤더의 마지막 비트를 사용할 수 있다, SVCE (SVC Extension)이라고 이름을 붙이자. SVCE=1이면, HMSVC NAL 헤더이고, SVCE=0이면 SVC NAL 헤더가 된다.
- <47> 첫 바이트에 있는 H.264/AVC NAL 헤더는 붙여지는 SVC NAL 헤더에서 첫 바이트와 같이 하는 것이 바람직하다. 이때, 두 번째 바이트에 있는 temporal_level(3)은 현 SVC의 기본 계층의 그것이고, 다섯 번째 바이트에 있는

temporal_level(3)은 현 SVC NALU의 그것이다.

<48> **MVC NAL 헤더를 이용한 제한적 SVC NAL 헤더**

<49> SVC 계층의 수를 매우 제한적으로 하는 경우에는 SVC에 대한 정보를 View_id (10) 또는 resv (6)를 이용하여 표현할 수 있다. 이 경우 MVC NAL 헤더의 자유도가 줄어들지만 NAL 헤더의 크기를 줄일 수 있다. HMSVC의 경우에는 1가지 종류의 SVC를 허용한다면, 1비트, 4가지 이하 종류의 SVC만을 허용한다면 2비트를 사용하여 SVC를 표시할 수 있다. 일 실시예로 view_id(10)을 view_id(8)+SVC_id(2)로 이용할 수 있다. 이를 사용하는 예로 MVC는 QCIF 15Hz로 하고, SVC는 CIF 15Hz, 4CIF 15Hz, 4CIF 30Hz가 가능하다. 이때, 뷰의 개수는 256개까지 가능하다. 일 실시예로, 다음과 같이 할 수 있다. 표 1은 제한적 SVC NAL 헤더를 위한 계층화 정보 테이블을 나타낸다.

표 1

<50>

	QCIF 15Hz (base)	CIF 15Hz	CIF 30Hz	4CIF 30Hz
SVC_id	00	01	10	11

<51> SVC_id(1)을 사용하는 경우에도 항상 계층은 매우 다양하게 설정할 수 있다, 일 실시예로 다음과 같이 SVC_id=0은 QCIF 15Hz로 하고, SVC_id=1은 4CIF 30Hz로 할 수 있다. 이때, 뷰의 개수는 512개까지 가능하다. View_id(10)을 사용할 수 없다면 SVC_id는 resv(6)에서 설정할 수 있다. 즉, resv(6)에서 1비트 또는 2비트를 사용하여 SVC 계층을 인식할 수 있다.

<52> 허용하는 SVC는 SPS (Sequence Parameter Set) 또는 PPS (Picture Parameter Set) 메시지를 이용해서 설정할 수 있다. SVC_id를 한 비트로 하느냐, 2비트로 하느냐를 선택적으로 할 수 있다면 이에 대한 선택도 SPS 또는 PPS 메시지를 통해 전달되어야 한다.

<53> **● 기타 방식**

<54> ■ slice header 이용 : Overhead가 너무 크기 때문에 현실적이지 못하다

<55> ■ UDP port를 뷰별로 규정

<56> 서버 클라이언트 구조의 HMSVC 스트리밍 서비스를 구현할 경우, 기본계층에 해당하는 MVC 비트열을 하나의 UDP 포트로 전송하고, 고급계층에 해당하는 SVC 비트열의 경우, 각 뷰 별로 UDP 포트를 할당하여 전송한다. 이러한 경우, 기본계층의 비트열들은 기존의 MVC 복호화 방법으로 복호를 실시하고, 고급계층에 한해서 전송되는 포트 로 뷰를 구분할 수 있다. 이는 MVC, SVC NAL header에서의 선택스 변화는 필요하지 않게 된다.

<57> ■ 라벨을 이용하는 방식

<58> 모든 NAL 헤더를 2바이트로 하고, 첫 번째 바이트는 H.264/AVC의 그것과 같이 하고, 두 번째 바이트는 라벨로 하여, 하나의 시퀀스에 대해서 구별해야하는 모든 종류의 NAL 헤더 리스트로 만들고 각각에 대해 라벨을 부여하는 방식이다. 라벨과 해당 NAL 헤더는 SPS 메시지를 통해 전달된다. SI/SP를 위한 NAL 헤더도 따로 있어야 한다. H.264/AVC의 NAL 헤더만 사용하면 인식이 가능한 SPS나 PPS에 대해서는 라벨이 필요없다.

<59> 라벨 방식은 1바이트 라벨방식으로 하면 256가지의 서로다른 NAL 헤더를 규정할 수 있고, 처음 7비트는 라벨로 사용하고 마지막 비트는 연장여부를 정하는 방식으로 하면 무한개의 NAL 헤더를 인식할 수 있게된다. 다음의 표에서와 같이 하나의 뷰에 대해서 기본 계층을 포함해서 3개의 SVC의 계층이 있고, 각각 시간적 레벨이 3개 인 경우에는 하나의 뷰에 대해서 9개의 라벨이 필요하다. 만일 8개의 뷰가 존재한다면, 총 72개의 라벨이 필요하다. SP나 SI를 사용하는 경우 라벨로 구분하려면 뷰마다 9개의 라벨이 더 필요하고, 슬라이스 헤더에서 구분한다면 따로 라벨이 필요 없다. 표 2는 라벨을 이용하는 HMSVC NAL 헤더를 위한 계층화 정보 테이블을 나타낸다.

표 2

<60>

Label	Temporal_ID (3)	Anchor (1)	View_id(8)	SVC(2)
0	0	1	0	00
1	1	0	0	00

2	2	0	0	00
3	0	1	0	01
4	1	0	0	01
5	2	0	0	01
6	0	1	0	10
7	1	0	0	10
8	2	0	0	10
9	0	1	1	00
10	1	0	1	00
11	2	0	1	00
12	0	1	1	01
13	1	0	1	01

<61> 앵커인지 여부는 temporal_id와 중복적이면 anchor(1)은 필요없다. 또한, view_level(3)은 view_id에서 유추하여 알 수 있다면, 3개의 비트는 필요없게 된다. 또한, temporal_id(3)과 SVC 종류가 내용이 중복되면, 부별 필요한 라벨 수는 더 줄어들게 된다. 다만, 서버, MANE, 복호기에서 필터링여부를 판단하기 위해 중복적인 정보를 구체화하기 위해 NALU마다 약간의 계산이 필요하다. MANE에서는 각 사용자별로 또는 액세스 네트워크별로 필터링해야하는 라벨의 리스트만 알려주면 된다. 구체적으로 어떤 라벨이 어떤 계층이나 뷰에 속하는지는 알려줄 필요없다.

<62> 2. SPS(Sequence Parameter Set)

<63> SPS는 부호화, 복호화기 모두가 공통적으로 가지고 있어야 하는 정보들을 담고 있다. 본 발명에서는 기존의 MVC, SVC의 SPS가 부호화, 복호화기 양쪽 모두에 필요하게 된다. 또한 MVC NAL 헤더를 이용한 제한적 SVC NAL 헤더 및 라벨을 이용하는 HMSVC NAL헤더를 이용하여 본 발명의 장치를 구현하게 될 경우 SPS의 수정이 필요하게 된다.

<64> ■ Concatenated MVC/SVC NAL header

<65> Concatenated MVC/SVC 구조를 사용할 경우, HMSVC NAL header에서 해당 슬라이스의 계층화 정보를 담고 있으므로, MVC, SVC를 동시에 지원할 수 있도록 MVC, SVC의 SPS가 모두필요하게 되며, 이때 각각의 MVC, SVC SPS는 기존의 방식을 따른다.

<66>

<67> ■ MVC NAL 헤더를 이용한 제한적 SVC NAL 헤더

<68> MVC NAL 헤더를 이용한 제한적 SVC NAL 헤더의 경우, SPS는 MVC NAL 헤더의 view_id에 해당하는 10비트 중 상위(또는 하위) 몇 비트를 이용하여 계층화를 표현할 것인지 명시하여야 한다. 또한 view_id에서 취한 계층화 정보의 내용을 부호화, 복호화기 간에 통일 시켜야 한다. 따라서 SPS 선택스의 변화가 필요하다

<69> 3. PPS(Picture Parameter Set) : 변화 없음

<70> **본 발명이 적용되는 서비스** : 본 발명은 멀티캐스트 서비스에서 유용하다. 피드백 요구의 전달 가능성은 상관없다. 여기서 피드백 요구란 사용자(client)가 현재 어떤 뷰를 시청하고 싶어하는가를 알려주는 것이다. 만일 소수의 사용자만 있는 경우에는 사용자가 시청하고 있는 뷰를 조사하여 아무도 시청하지 않는 경우에는 전송하지 않음으로써 자원을 절약할 수 있다. 그리고, 사용자별로 가용한 대역폭을 알 수 있는 경우(예를 들어, IEEE 802.16e)에는 사용자에게 따라 항상 계층을 전송할지 하지 않을지 여부를 선택적으로 결정할 수 있으면 바람직할 것이다. 이러한 결정은 사용자 그룹에 가까이 있는 MANE (Media Aware Network Element)에서 이루어지거나, 서버에서 이루어지는 것이 바람직하다.

<71> 서비스 품질은 또한 사용자의 단말기에 따라 결정되거나 사용자의 선택에 의해 결정될 수도 있다. 만일 수신 패킷별로 요금이 책정되는 경우에는 사용자가 품질을 선택할 수 있는지 여부가 매우 중요하다. 사용자가 매우 많아서 개별적인 제어가 어려운 경우에는 전송대역폭 줄이는 효과보다는 단말기 자원을 절약하는 효과를 위해 본 발명이 사용될 수 있다.

<72> 4개의 뷰가 있는 서비스에서 뷰 1, 2, 3이 뷰 0을 참조한다고 하자. 단말기 (1)과 (2)는 모든 뷰의 모든 데이터

가 브로드캐스팅되는 망에서 필요한 스트림을 선택하여 디코딩하여 디스플레이한다. 뷰 1을 선택한 단말기 (1)은 디스플레이 화면 크기가 작고, CPU 성능이 충분하지 않으므로 뷰 0과 뷰 1의 기본 계층만 복호화한다. 뷰 0은 뷰 1의 참조 뷰로 필요하다. 뷰 2를 선택한 단말기 (2)에서는 디스플레이 화면 크기가 크고, CPU 성능이 충분하므로 SVC 향상 계층을 수신하여 복호화한다. 단말기 (3)에서는 뷰 1을 선택하였고 1:1 VOD 서비스이므로 MANE에서는 단말기 (3)에서 오는 피드백 정보에 의거하여 필요없는 스트림은 필터링하여 전송한다.

<73> NAL 헤더와 SPS의 신택스가 필요하다. 여기서 NAL 헤더는 결합형 (concatenated)과 SVC 제한형 (restricted SVC) 두 가지에 대하여 명시한다.

<74> ■ HMSVC 패킷의 NAL 헤더 구조

<75> 1. 결합형 NAL 헤더 (Concatenated MVC/SVC NAL header)

<76> 표 3a 및 표 3b는 결합형 NAL 헤더를 나타내는 것으로서, 표 3b는 표 3a에 이어지는 부분이다.

표 3a

nal_unit_header_svc_mvc_extension() {	C	Descriptor
svc_mvc_flag	All	u(1)
if (!svc_mvc_flag) {		
reserved_zero_one_bit	All	u(1)
priority_id	All	u(6)
temporal_level	All	u(3)
dependency_id	All	u(3)
quality_level	All	u(2)
reserved_zero_one_bit	All	u(1)
layer_base_flag	All	u(1)
use_base_prediction_flag	All	u(1)
discardable_flag	All	u(1)
fragmented_flag	All	u(1)
last_fragment_flag	All	u(1)
fragment_order	All	u(2)
} else {		
temporal_level	All	u(3)
view_level	All	u(3)
anchor_pic_flag	All	u(1)
view_id	All	u(10)

<77>

표 3b

reserved_zero_five_bits	All	u(5)
Hybrid_mvc_svc_extension_flag	All	u(1)
}		
nalUnitHeaderBytes += 3		
if(!Hybrid_mvc_svc_extension & !svc_mvc_flag) {		
reserved_zero_two_bits	All	u(2)
priority_id	All	u(6)
temporal_level	All	u(3)
dependency_id	All	u(3)
quality_level	All	u(2)
reserved_zero_one_bit	All	u(1)
layer_base_flag	All	u(1)
use_base_prediction_flag	All	u(1)
discardable_flag	All	u(1)
fragmented_flag	All	u(1)
last_fragment_flag	All	u(1)
fragmented_order	All	u(2)
nalUnitHeaderBytes += 3		
}		

<78>

<79>

3. SVC 제한형 (restricted SVC) NAL 헤더를 위한 SPS, MVC NAL 헤더 구조

<80>

MVC NAL header의 view_id(u(10))의 상/하위의 비트를 이용하여 제한적인 계층화 정보를 나타낸다. 이 때, view_id의 몇 비트를 이용할지는 SPS를 이용하여 설정한다.

<81>

표 4는 SVC 제한형 NAL 헤더를 위한 SPS 메시지 처리 방식을 나타낸다.

표 4

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
nalUnitHeaderBytes = 1		
if(nal_unit_type == 7) {		
seq_parameter_set_rbsp() /* specified in Annex G */		
}		
NumBytesInRBSP = 0		
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
}		
}		

<82>

<83>

표 5a, 표 5b, 및 표 5c는 SPS 처리 루틴에서 profile_idc값(XX)에 따라서seq_parameter_set_mvc_extenstion() 수행하는 것을 나타낸다.

표 5a

seq_parameter_set_rbsp() {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 83) {		
seq_parameter_set_svc_extension() /* specified in Annex G */		
}		
if(profile_idc == XX) {		
seq_parameter_set_mvc_extension() /* specified in Annex G */		
}		
if(profile_idc == 100 profile_idc == 110 profile_idc == 122 profile_idc == 144 profile_idc == 83 profile_idc == XX) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc == 3)		
residual_colour_transform_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)

<84>

표 5b

if(seq_scaling_matrix_present_flag)		
for(i = 0; i < 8; i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
Else		
scaling_list(ScalingList8x8[i - 6], 64, UseDefaultScalingMatrix8x8Flag[i - 6])	0	
}		
}		
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)

<85>

표 5c

direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters()	0	
rbp_trailing_bits()	0	
}		

<86>

<87> 표 6a 및 표 6b는 NAL MVC SPS 처리 루틴을 나타낸다.

표 6a

seq_parameter_set_mvc_extension() {	C	Descriptor
num_views_minus_1		ue(v)
for(i = 0; i <= num_views_minus_1; i++) {		
num_anchor_refs_l0[i]		ue(v)
for(j = 0; j < num_anchor_refs_l0[i]; j++)		
anchor_ref_l0[i][j]		ue(v)
num_anchor_refs_l1[i]		ue(v)
for(j = 0; j < num_anchor_refs_l1[i]; j++)		
anchor_ref_l1[i][j]		ue(v)
}		
for(i = 0; i <= num_views_minus_1; i++) {		
num_non_anchor_refs_l0[i]		ue(v)
for(j = 0; j < num_non_anchor_refs_l0[i]; j++)		
non_anchor_ref_l0[i][j]		ue(v)
num_non_anchor_refs_l1[i]		ue(v)
for(j = 0; j < num_non_anchor_refs_l1[i]; j++)		
non_anchor_ref_l1[i][j]		ue(v)
}		
restricted_bits		ue(v)
dependency_id		ue(v)
temporal_level		ue(v)
quality_level		ue(v)

<88>

표 6b

for(i = 0; i < dependency_id * temporal_level * quality_level; i++) {		
for(j = 0; j < dependency_id; j++) {		
svc_table[i] = svc_table[i] & j		
svc_table[i] = svc_table[i] << 3		
for(k = 0; k < temporal_level; k++) {		
svc_table[i] = svc_table[i] & k		
svc_table[i] = svc_table[i] << 3		
for(l = 0; l < quality_level; l++) {		
svc_table[i] = svc_table[i] & l		
svc_table[i] = svc_table[i] << 2		
}		
}		
}		
}		
}		

<89>

<90> 표 7은 nal_unit_header_svc_mvc_extension 처리 루틴을 나타낸다.

표 7

nal_unit_header_svc_mvc_extension() {	C	Descriptor
svc_mvc_flag	All	u(1)
if (!svc_mvc_flag) {		
reserved_zero_one_bit	All	u(1)
priority_id	All	u(6)
temporal_level	All	u(3)
dependency_id	All	u(3)
quality_level	All	u(2)
reserved_zero_one_bit	All	u(1)
layer_base_flag	All	u(1)
use_base_prediction_flag	All	u(1)
discardable_flag	All	u(1)
fragmented_flag	All	u(1)
last_fragment_flag	All	u(1)
fragment_order	All	u(2)
} else {		
temporal_level	All	u(3)
view_level	All	u(3)
anchor_pic_flag	All	u(1)
view_id	All	u(10)
svc_id = restricted_bits & MSB(view_id)		
view_id = 11111 & LSB(view_id)		
reserved_zero_six_bits	All	u(6)
}		
nalUnitHeaderBytes += 3		

<91>

<92>

3. Label을 이용한 HMSVC NAL unit 처리

<93>

A. SPS를 읽고 복호화기 설정

<94>

i.NAL header로부터 SPS 처리 루틴으로 이동(표 8 참조)

표 8

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
nalUnitHeaderBytes = 1		
if(nal_unit_type == 7) {		
seq_parameter_set_rbsp() /* specified in Annex G */		
}		
NumBytesInRBSP = 0		
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
}		
}		

<95>

<96>

ii. SPS 처리 루틴에서 profile_idc 값에 따라서(XXX) seq_parameter_set_mvc_label_extenstion() 수행(표 9a, 9b, 및 9c 참조)

표 9a

seq_parameter_set_rbsp() {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 83) {		
seq_parameter_set_svc_extension() /* specified in Annex G */		
}		
if(profile_idc == XX) {		
seq_parameter_set_mvc_extension() /* specified in Annex G */		
}		
if(profile_idc == XXX) {		
seq_parameter_set_mvc_label_extension()		
}		
if(profile_idc == 100 profile_idc == 110 profile_idc == 122 profile_idc == 144 profile_idc == 83 profile_idc == XX profile_idc == XXX) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc == 3)		
residual_colour_transform_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)

<97>

표 9b

seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < 8; i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
Else		
scaling_list(ScalingList8x8[i - 6], 64, UseDefaultScalingMatrix8x8Flag[i - 6])	0	
}		
}		
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		

<98>

표 9c

mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters()	0	
rbsp_trailing_bits()	0	
}		

iii. seq_parameter_set_mvc_label_extension()를 통한 label 설정(표 10 참조)

표 10

seq_parameter_set_mvc_label_extension() {	C	Descriptor
num_views_minus_1		ue(v)
temporal_level		ue(v)/u(3)
dependency_id		ue(v)/u(3)
quality_level		ue(v)/u(2)
for(i = 0; i <= num_views_minus_1*temporal_level*dependency_id*quality_level; i++) {		
for(j = 0; j <= temporal_level; j++) {		
label[i] = label[i] & temporal_level		
label[i] = label[i] << 3		
for(k = 0; k <= dependency_id; k++) {		
label[i] = label[i] & dependency_id		
label[i] = label[i] << 3		
for(l = 0; l <= quality_level; l++) {		
label[i] = label[i] & quality_level		
label[i] = label[i] << 2		
}		
}		
}		
}		
}		

B. Hybrid MVC SVC NAL unit type code 정의(표 11a 및 11b 참조)

표 11a

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
14	Coded slice of an Hybrid-MVC-SVC picture slice_layer_in_HMSVC_label_extension()	2, 3, 4

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
nalUnitHeaderBytes = 1		
if(nal_unit_type == 14) {		
nal_unit_header_HMSVC_label_extension()		
}		
NumBytesInRBSP = 0		
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
}		
}		

표 11b

nal_unit_header_HMSVC_label_extension() {	C	Descriptor
label_number	All	u(7)
next_label_bit_flag	All	u(1)
if(!next_label_bit_flag) {		
nal_unit_header_HMSVC_label_extension()		
}		
}		

■ Semantics

● Concatenated MVC SVC NAL header

■ nal_unit_header_svc_mvc_extension()

Hybrid_mvc_svc_extension_flag 해당 비트가 1(또는 0)으로 설정되어 있는 경우, 이후 존재하는 3바이트는 SVC header로 인식하여 처리하도록 하는 플래그 비트

● Restricted SVC NAL header

■ seq_parameter_set_mvc_extension()

restricted_bits HMSVC NAL header가 입력될 경우, view_id에 해당하는 10bits 중 상위(또는 하위) 몇 비트를 이용하여 계층화 정보를 판단하는 지 나타낸다.

dependency_id HMSVC 부호/복호기가 지원 할 수 있는 공간적 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.

- <113> **temporal_level** HMSVC 부호/복호기가 지원 할 수 있는 시간적 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.
- <114> **quality_level** HMSVC 부호/복호기가 지원 할 수 있는 화질 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.
- <115> ■ **nal_unit_header_svc_mvc_extension()**
- <116> **view_id** seq_parameter_set_mvc_extension()의 restricted_bits를 이용하여 view_id의 상위(또는 하위) 비트로부터 계층화 정보를 나타낼 svc_id를 추출한다. 추출된 svc_id는 계층화 정보 테이블을 이용하여 해당되는 계층화 정보를 표현한다.
- <117> ● Label을 이용한 HMSVC NAL header
- <118> ■ **seq_parameter_set_mvc_label_extension()**
- <119> seq_parameter_set_mvc_label_extension()은 부호/복호기에서 지원 가능한 계층화 종류를 규정 짓는다. 이를 이용하여 부호/복호기는 계층화 테이블(label)을 작성하여 유지한다.
- <120> **num_views_minus_1** 전체 뷰의 개수를 표현
- <121> **temporal_level** HMSVC 부호/복호기가 지원 할 수 있는 시간적 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.
- <122> **dependency_id** HMSVC 부호/복호기가 지원 할 수 있는 공간적 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.
- <123> **quality_level** HMSVC 부호/복호기가 지원 할 수 있는 화질 계층화 정도를 나타낸다. 해당 값을 이용하여, 부호/복호기는 계층화 정보 테이블을 작성하고, HMSVC NAL header가 입력되는 경우에 계층화 정보를 판단한다.
- <124> ■ **nal_unit_header_HMSVC_label_extension()**
- <125> 새로운 nal_type 정의(Coded slice of an Hybrid-MVC-SVC picture)에 따라, HMSVC_label NAL header가 입력되는 경우, 처리하는 루틴.
- <126> **label_number** seq_parameter_set_mvc_label_extension()에 의하여 부호/복호기에 설정된 계층화 label 테이블의 label 번호를 나타낸다.
- <127> **next_label_bit_flag** label 번호가 7비트로서 표현이 불가능한 경우, 연장을 위하여 사용되는 플래그 비트로서 그 값이 1(또는 0)으로 설정되어 있는 경우, 이후 존재하는 1바이트를 nal_unit_header_HMSVC_label_extension()으로 처리하도록 한다.
- <128> 4. HMSVC를 이용한 자원 효율적인 다시점 스트리밍 장치
- <129> 다음의 그림(도 10)은 HMSVC 부호화, 복호화기, MANE를 이용하는 자원 효율적인 다시점 스트리밍 장치의 블록도를 나타낸다. 다시점 영상을 위하여 하나 이상의 카메라를 입력으로 하여 기본계층을 위한 입력은 다운 샘플링을 실시하고, 고급계층을 위한 입력은 입력 영상을 그대로 하여 각각 부호화기에 입력한다. 이 때, 기본계층에 해당하는 MVC 부호화기는 기존 MVC 부호화기의 예측구조 등을 그대로 사용하도록 한다. 각각의 기본계층을 참조하여, 고화질의 고급계층을 부호화 한다. 이때 생성되는 고급계층의 비트열에는 본 발명에서의 기술인 Concatenated MVC/SVC NAL header 또는 SPS, MVC NAL header를 이용한 Restricted SVC NAL header를 이용한다. 각각의 부호화기에서 생성된 비트열은 다중화기를 이용하여 하나 또는 다수의 비트열로 합쳐져서 MANE로 전송되고, 이 중 기본계층은 그 모두를 전송하고, 고급계층은 사용자의 명령(view change command)에 따라, 사용자가 시청하기를 원하는 뷰를 선택적으로 전송한다. 복호기에 도착한 비트열은 역 다중화기를 통하여, 기본계층 비트열과 고급계층 비트열로 나뉘어 각각 해당하는 복호기로 입력되어 복호화 된다. 이 때, 고급계층의 비트열은 현재 고화질로 시청되어야 할 뷰에 해당하며, 기본계층의 동일 뷰(그림에서 S2)를 참조하여 복호화 한다. 시청 도중 사용자의 요구에 의한 뷰 전환이 있을 경우, 뷰 전환 명령이 MANE로 전달되어 지고, 필터는 해당되는 뷰의 고급계층 비트열을 전송한다. 복호화기에 위치하는 스위치는 뷰전환이 있을 경우, 고급계층의 참조 방향을 변환시켜주는 역할을 수행한다. 도 11은 HMSVC를 이용한 스트리밍 시스템의 예를 보여준다.

발명의 효과

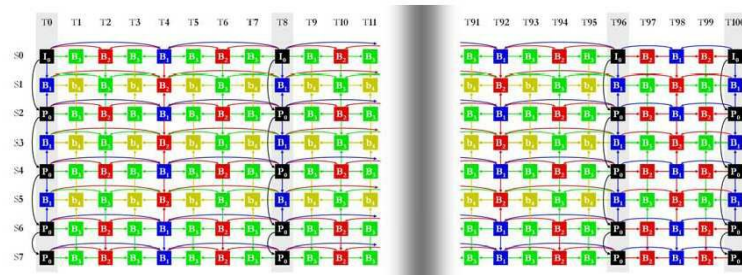
- <130> MVC의 적용범위가 크게 확대될 수 있다. 현재는 모든 뷰를 다 수신하고 복호화하여야 하므로 대역폭, CPU 성능, 메모리 자원이 크게 필요하다. 그러나, 디스플레이할 뷰와 요구되는 품질에 따라 전송하는 스트림을 선택적으로 조절할 수 있게 되었다. 다음과 같은 조건에서 적용범위가 넓어진다.
- <131> 1. 사용자가 디스플레이할 뷰와 품질을 선택할 수 있음.
- <132> 2. 전송가능한 대역폭의 범위가 더욱 확대됨. (즉, 가용 대역폭이 적은 경우에도 서비스 할 수 있게됨)
- <133> 3. 사용할 수 있는 단말기의 범위가 확대됨. (CPU의 성능이 더 낮은 경우와 디스플레이 크기가 더 작은 경우에 적용할 수 있게 되었음)

도면의 간단한 설명

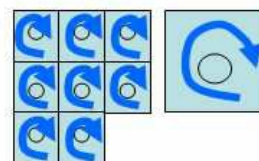
- <1> 도 1. 다시점 비디오 코딩의 예측 구조의 예
- <2> 도 2. QCIF 8개와 CIF 한 개가 병렬적으로 (parallel) 복호화되는 HMSVC 방식
- <3> 도 3. QCIF 4개와 CIF 한 개가 포함된 HMSVC 방식
- <4> 도 4. 실시간 부호화/복호화되는 1:1 대화형 서비스
- <5> 도 5. 스트리밍시 지연되지만 화질저하 없는 랜덤 액세스
- <6> 도 6. 화질 저하는 되지만 지연없이 랜덤 액세스
- <7> 도 7. SI/SP를 이용한 랜덤 액세스
- <8> 도 8. Concatenated MVC/SVC NAL header
- <9> 도 9. 본 발명의 사용 시나리오
- <10> 도 10. HMSVC 부호화 / 복화화기, MANE를 이용하는 스트리밍 시스템의 블록도
- <11> 도 11. HMSVC를 이용한 스트리밍 시스템의 예

도면

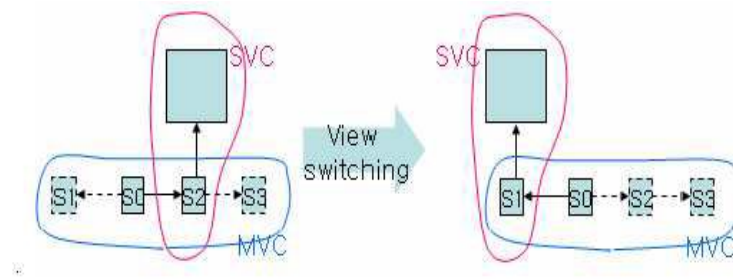
도면1



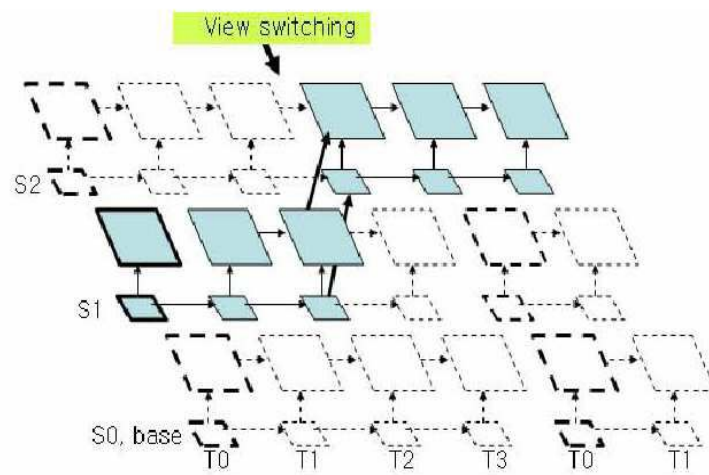
도면2



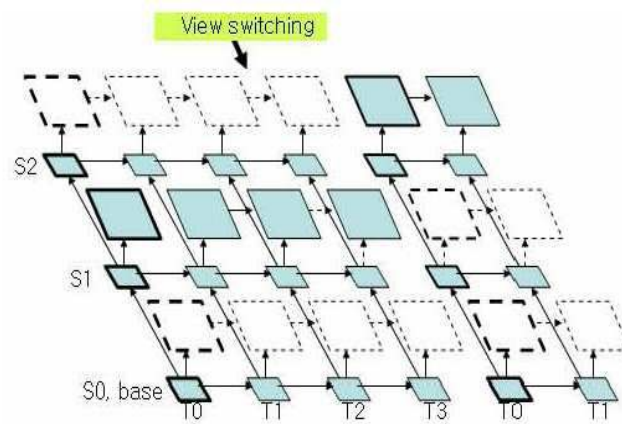
도면3



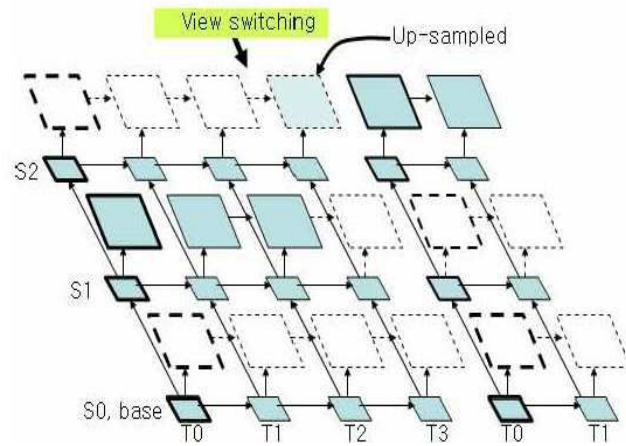
도면4



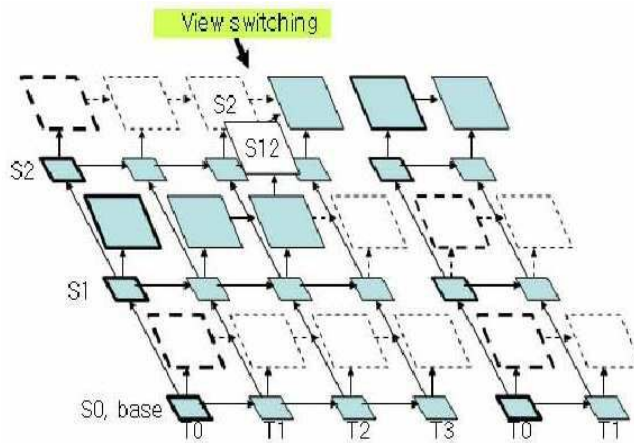
도면5



도면6



도면7

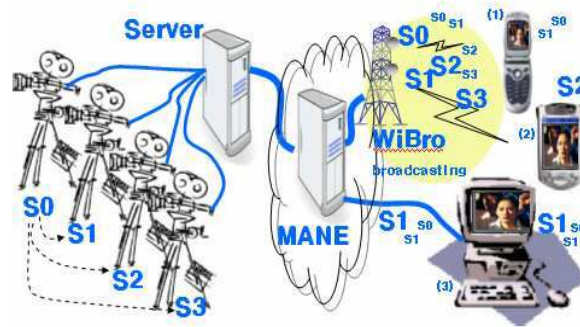


도면8

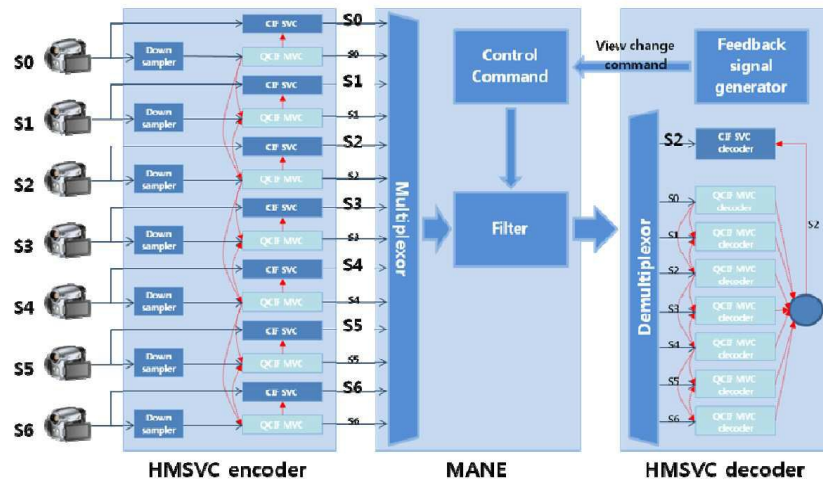
F (1)	NRI (2)	NAL types (5)	svc_mvc (1)	Temporal_level (3)	View_level (3)	Anchor_flag (1)	View_id (10)	Resv (5)	SVCE (1)
----------	------------	------------------	----------------	-----------------------	-------------------	--------------------	-----------------	-------------	-------------

svc_mvc (1)	resv (1)	priority id (6)	temporal level (3)	dependency id (3)	quality level (2)	resv (1)	layer base (1)	base prediction (1)	discard able (1)	frag ment (1)	last fragment (1)	order (1)
----------------	-------------	-----------------------	--------------------------	-------------------------	-------------------------	-------------	----------------------	---------------------------	------------------------	---------------------	-------------------------	--------------

도면9



도면10



도면11

