H264 Scalable Video Coding 学习报告

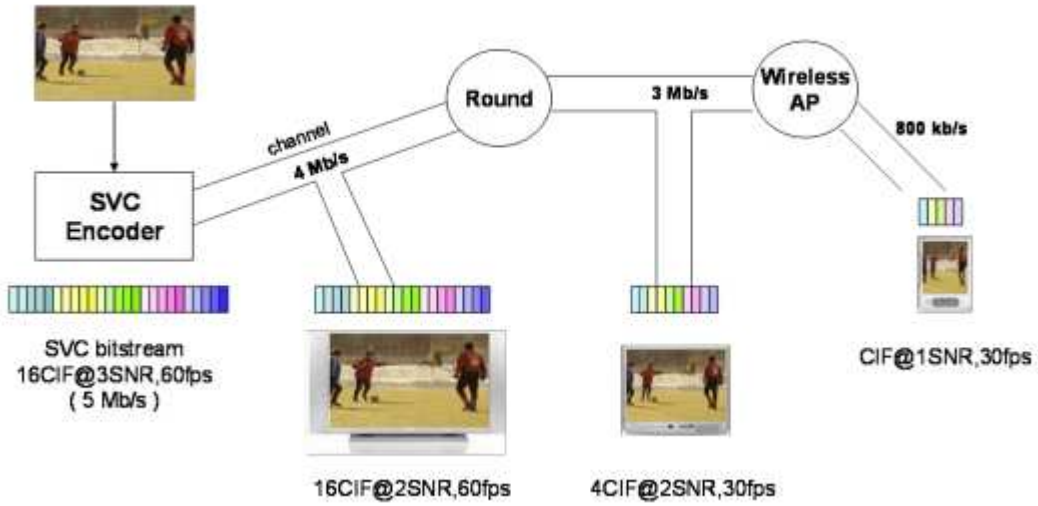
- by shane, 2009-9-25

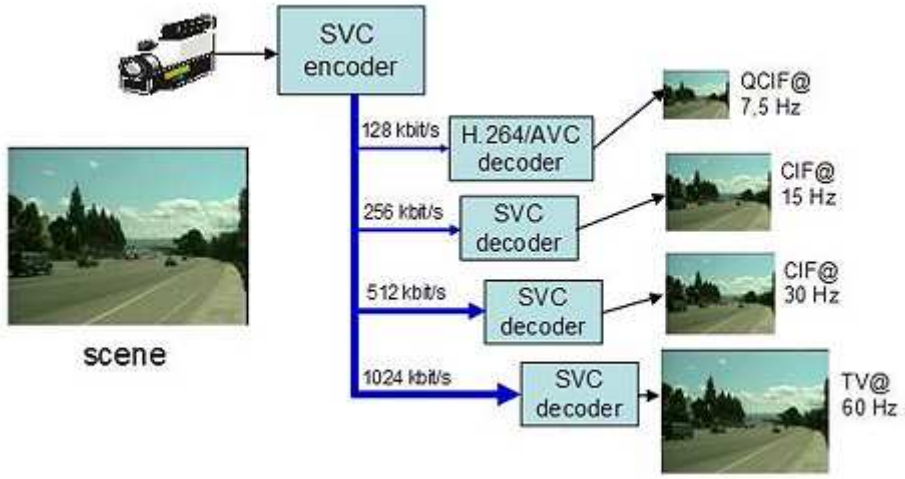
1. Scalable Video Coding – 可分级视频编码

- 为什么需要可分级视频编码技术（SVC）

实际应用中，存在不同的网络和不同的用户终端，各种情况下对视频质量的需求不一样。例如，在利用网络传输视频信息时，由于网络带宽限制了数据传输，因此要求当网络带宽较小的时候，只传输基本的视频信号，并根据实际网络的状况决定是否传输增强的视频信息，使视频的质量得到加强。

在这样的背景下，利用可分级视频编码技术实现一次性编码产生具有不同帧率、分辨率的视频压缩码流，然后根据不同网络带宽、不同的显示屏幕和终端解码能力选择需要传输的视频信息量，以此实现视频质量的自适应调整。





- 可分级视频编码（SVC）是什么

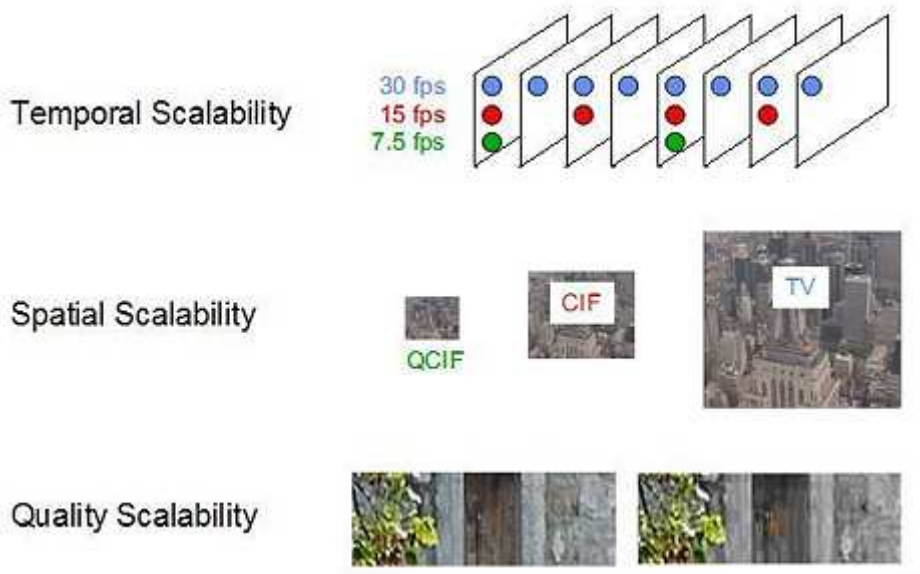
为了能够实现从单一码流中解码得到不同帧率（时间可分级）、分辨率（空间可分级）和图像质量（SNR 可分级）的视频数据的编码技术。

H.264 SVC 以 H.264 AVC 视频编解码器标准为基础，利用了 AVC 编解码器的各种高效算法工具，在编码产生的编码视频时间上（帧率）、空间上（分辨率）可扩展，并且是在视频质量方面可扩展的，可产生不同帧速率、分辨率或质量等级的解码视频。

2. H264 Scalable Video Coding Extension

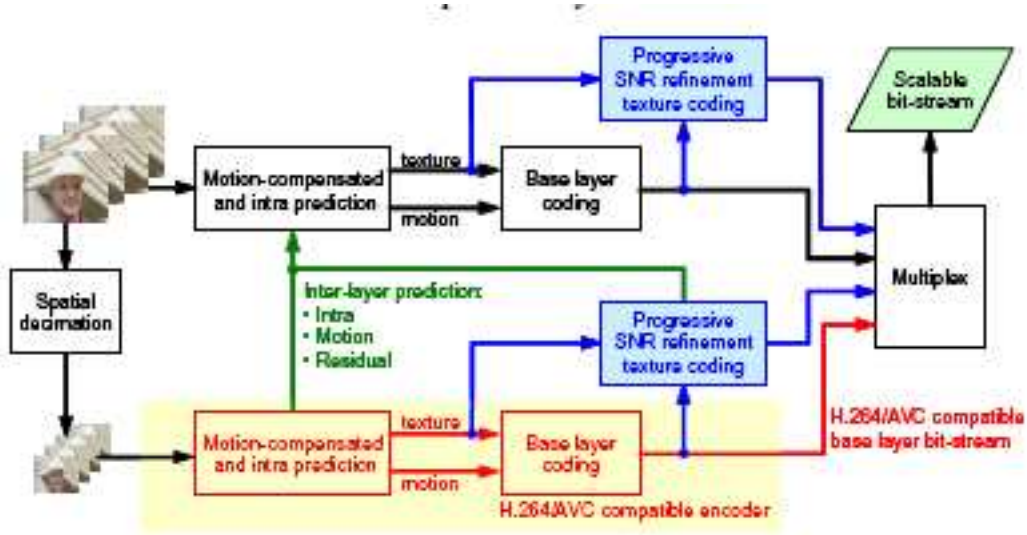
H.264 SVC（H.264 可分级编码）作为 H.264 标准的一个扩展最初由 JVT 在 2004 年开始制定，并于 2007 年 7 月获得 ITU 批准。

H264 可分级视频编码是 H264/AVC 标准的扩展 (Annex G , 2007)，它是以 H264/AVC 标准为基础，通过各种编码工具，形成分层编码(multi-layer coding)方式，以提供时间(Temporal)上，空间(Spatial)上和图像质量(Quality)上的可调整特性，可产生不同帧率、图像分辨率或图像质量等级的解码视频。



- H264 SVC 结构

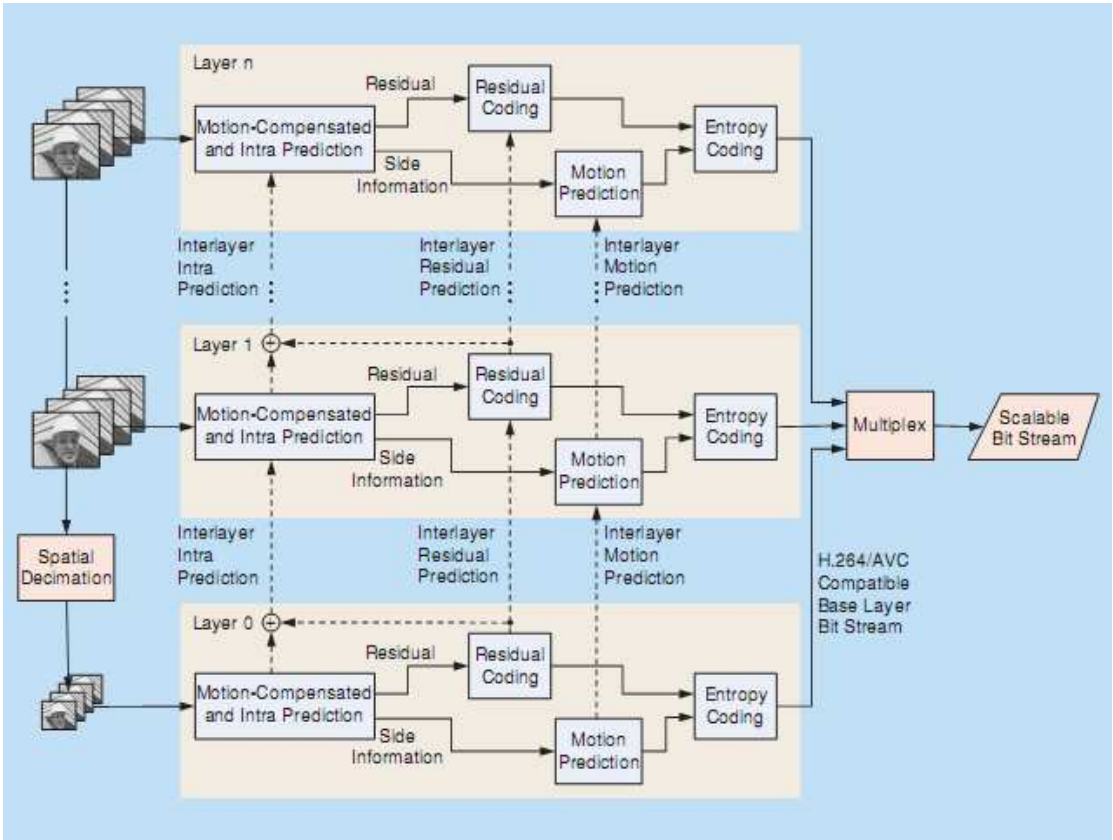
H264 可分级视频编码采用分层编码方式实现，由一个基本层（Base Layer）和多个增强层（Enhancement Layer）组成，增强层依赖基本层的数据来解码。其中，基本层(base layer)编码了基本的视频信息，实现了最低图像分辨率、帧率，并且基本层的编码是兼容 H264/AVC 编码标准的，能够采用 H264/AVC 解码器进行解码。



H264 SVC 以不同分辨率的图像为基础形成分层结构**(inter-layer prediction)**，并在此基础上每一层用 **Hierarchical B-picture 来实现 Temporal Scalablity，用 MGS 编码来实现 Quility Scalability (SNR Scalability)**

每一层内的编码都使用了 H264/AVC 中帧内和帧间预测编码工具，而相邻层之间使用了 SVC 独有的层间预测编码工具。

EL（Enhancement Layer）的参考，可以是来源于同一层的其他帧，也可以是低层上采样的帧。但是不可以是更高层的帧，那样的话丢弃高层的 nalu 会导致低层无法解码。

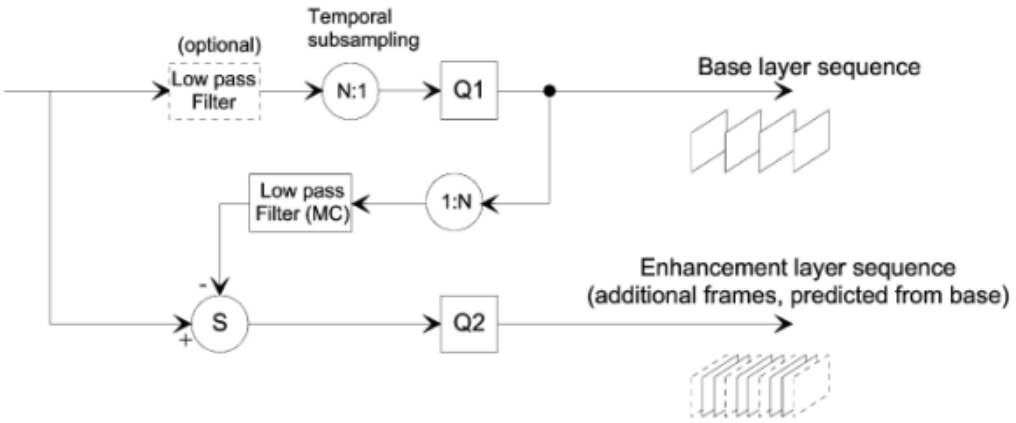


- Temporal Scalablity – 时间可分级编码工具

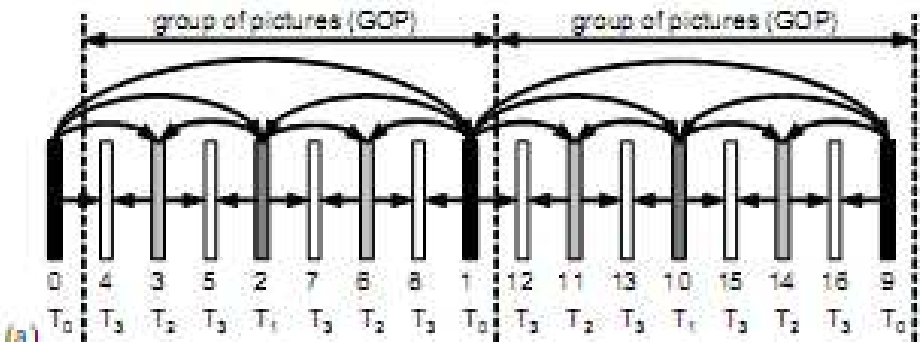
利用 **Hierarchical B-picture**(层次化Ｂ帧)实现时间可分级，使每一层具有不同的帧率。
（note: MTCF 由于复杂性最终没有被 SVC 标准采用）

在原始的 JSVM 設計中，採用了 MCTF 來達到 Temporal Scalability 與產生更高解析度的加強層，但是在標準的制訂過程中，是否要使用 MCTF 這種 Lifting 的架構一直飽受爭議，雖然要達到解碼端的畫面完美重建勢必無法關閉 Lifting 架構中的 Updating Stage，但是如果不關閉 Updating Stage 在低位元率的時候，反而會因為量化誤差而使兩兩畫面間的畫質差異過大，且 Lifting 架構屬於 Open Loop 的編碼架構，會發生飄移誤差的問題，再加上使用 MCTF 的編碼架構會使得編碼複雜度大大的提高。在諸多考量下，後來放棄了納入 MCTF 的編碼架構，僅只保留 Hierarchical B pictures 的編碼方式來達成 Temporal Scalability，但嚴格來說 Hierarchical B pictures 的編碼方式也是一個模仿 MCTF 的編碼架構。

Hierarchical B-picture 的概念来自于传统的 B picture 编码方式，但与之不同的是，在 Hierarchical B-picture 中产生的 B picture 可以被其他 B picture 参考



分层预测结构

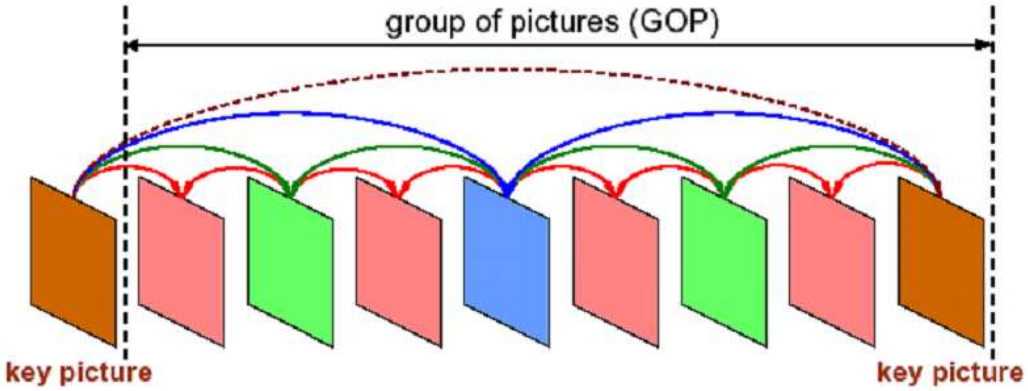


每一层对应不同的帧率：

			0			1			9	lay 0
		0		2		1		10	9	lay 1
	0	3	2		1		11	10	14	9 lay 2
Full-frame-rate:	0	4	3	5	2	7	6	8	1	12 11 13 10 15 14 16 9 lay 3

Picture 0/9 – Key Frame, coded by I/P frame

Qulity Scalability(MGS)也使用了 Key Frame



- Spatial Scalability – 空间可分级编码工具

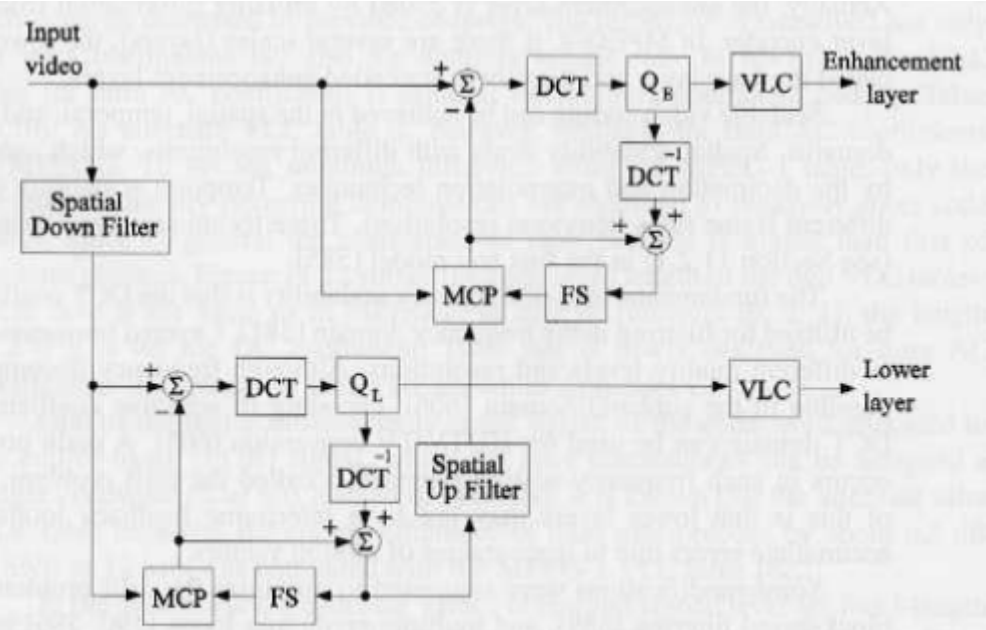
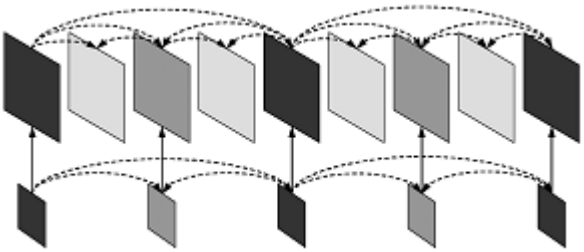


Figure 11.23 An example of a two-layer coder structure for spatial scalability (MCP: motion compensated prediction; FS: frame store).



利用 down/up sampling filter(上/下采用滤波)、inter-layer prediction(层间预测)实现时间可分级，使每一层具有不同的图像分辨率。

由于低层是由高层通过下采样得到的，因此在相同的时域分辨率情况下，每一个高层帧都有与之相对应的低层帧，两者之间存在着显而易见的相关性，我们称之为层间冗余。显然，为了获得更高的压缩效率，有必要通过各种层间预测技术来消除层间冗余，这也是空域可伸缩性技术的关键所在。设计层间预测算法的目的是尽可能利用基本层的信息提高增强层的编码效率。

几个相关语法元素

- base mode flag：指明一种宏块类型。1 表示当前宏块（enhancement layer 中）只编码残差信息，帧内编码模式或运动相关信息都有相应的参考层中块推导出来。EL 中的宏块无论 inter 还是 intra 都可以 base mode flag=1。

- motion prediction flag: 作用于宏块中每个 partition 的 reference list，指明 reference index、motion vector prediction 是否由 reference layer 中相应的块推导出来。

- residual prediction flag：只要当前宏块是 inter，无论 base mode flag 是否为 1，都可以采用下述第三种 inter-layer prediction：**inter-layer residual prediction**。

H264 SVC 提供三种层间预测（**inter-layer prediction**）：

- **inter-layer intra prediction**

base mode flag 为 1，且相应的 reference layer 中的 8x8 块是帧内编码，参考层中的 4x4 块（4 个）被重构，经过去块滤波操作后上采样得到预测信号。**亮度上采样采用的是 4-tap 的 FIR 滤波器，色度上采样采用的是 bilinear 滤波器。**而后，ehencement layer 传送残差系数，经反变换后加到预测信号上。

在 upsampling 之前，需要对 reference layer 的重构信息进行去块滤波（deblocking）。为了保证 sigle loop 的解码，需要避免在 reference layer 进行 motion compensation，即 Constrained Inter-layer Intra-prediction。

- inter-layer motion prediction

base mode flag 为 1，且相应的参考层中的 8x8 块是帧间编码，Block partition 信息、reference index、motion vectors 由 reference layer 中相应的 8x8 块推导出来。此时 block partition 信息由 reference layer 中相应的 8x8 块的 partition 信息上采样得到，同时得到与 partition 相应的 reference index、motion vectors（需要先被 scale）。

base mode flag 为 0 时，还可以有一种对 motion 的 interlayer prediction，开关为 motion prediction flag：

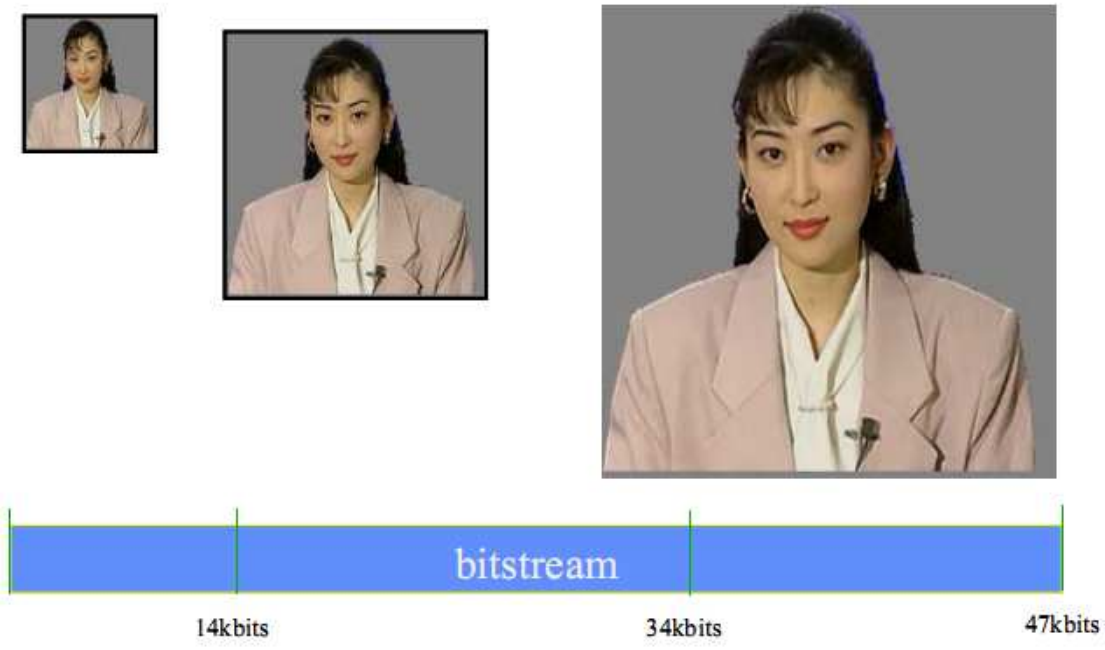
- **motion prediction flag 为 1**，则与此 reference list 相关的 reference index、motion vector prediciton 由 reference layer 中相应的块推导出来。这里与 base mode flag 为 1 时有点区别，这里只是 MVP 由 reference layer 推导出来，所以 EL 还要传送 MVD。别忘了 motion vectors 需要被 scale。

- **motion prediction flag 为 0**，传统的帧间宏块：运动矢量的预测与 h.264 标准相同。

- inter-layer residual prediction

在 enhencement layer 中的 inter-coded 宏块，无论是采用 base mode flag 还是传统的帧间宏块，都可以采用这一方式。

residual prediction flag 为 1，则 reference layer 中相应的 8x8 块的残差经过 **bilinear 上采样**，作为 enhencement layer 宏块的残差预测，enhencement layer 中传送“残差的残差”。需要注意的是这里的上采样不要跨越 reference layer 的变换块边界，否则会降低视觉效果，具体的处理办法是对边界进行重复外拓。



目前的 JSVM 是一個多次執行的程式，為了要達到 Spatial Scalability 的目的，在每次開始執行編碼器之前，需要一個額外的程式將原始的視訊內容調整到所要的解析度大小。例如，原始的視訊畫面大小為 4CIF 大小的畫面，如果要增加對畫面大小 CIF 與 QCIF 的支援的話，在 JSVM 裡面提供了一個名為 DownConvert 的程式讓我們可以將原始畫面轉換到任意的畫面

解析度大小。事實上，DownConvert 只不過是一個傳統的 Spatial domain 的轉碼器，這個程序被稱為 Spatial Decimation。當完成畫面轉換之後，每個解析度大小的畫面將各自被送進獨自的編碼程序中。

- Quility Scalability (SNR Scalability) – 图像质量可分级

一般是通过越来越小的量化步长来进行量化而进行的
CGS
MGS (SVC 中采用)
MGS 需要考虑的是压缩效率和 drift 效应的折中，H264 SVC 中是通过引入 key picture 达到较好的效果。 Key picture 即时域最基本层的图像。这些 key picture 的重构是通过 base quality layer 实现的。因此 higher quality layer 的丢弃造成的 drift 仅仅只能作用在当前 key picture 到下一个 key picture 之间，实现了对 drift 效应限制。与此不同，时域扩展层的图像则是通过参考层的最高可用 quality 运动补偿得到的，从而提升了压缩效率。要实现压缩效率和 drift 效应的最佳折中，可以通过调整 GOP size 或者 hierarchy 的阶数。

FGS

要達到能夠調整畫質的條件，是每一層都使用不同的量化參數(Quantization parameter)，想要高的畫質，就選用較低的量化參數，反之亦然。SVC 提供了兩種可調整畫質的方法，一種是 Coarse Grain Scalability(CGS)，它採用與空間可調性相同的技術。另一方面，在預估過後的殘留值使用類似 MPEG-4 精細可調的位元層編碼(FGS Bit-plane Coding)做壓縮，使得每個解析度層都有一定的畫質可調範圍。FGS 的技術與以前 MPEG-4 或是 MPEG-2 不同的地方是，它可以動態調整是要由強化層(enhancement layer)或是基本層作為預測的參考，這樣可以兼顧編碼效率，以及考慮到若是強化層的資料被砍掉時，不會有太大的預測錯誤產生。

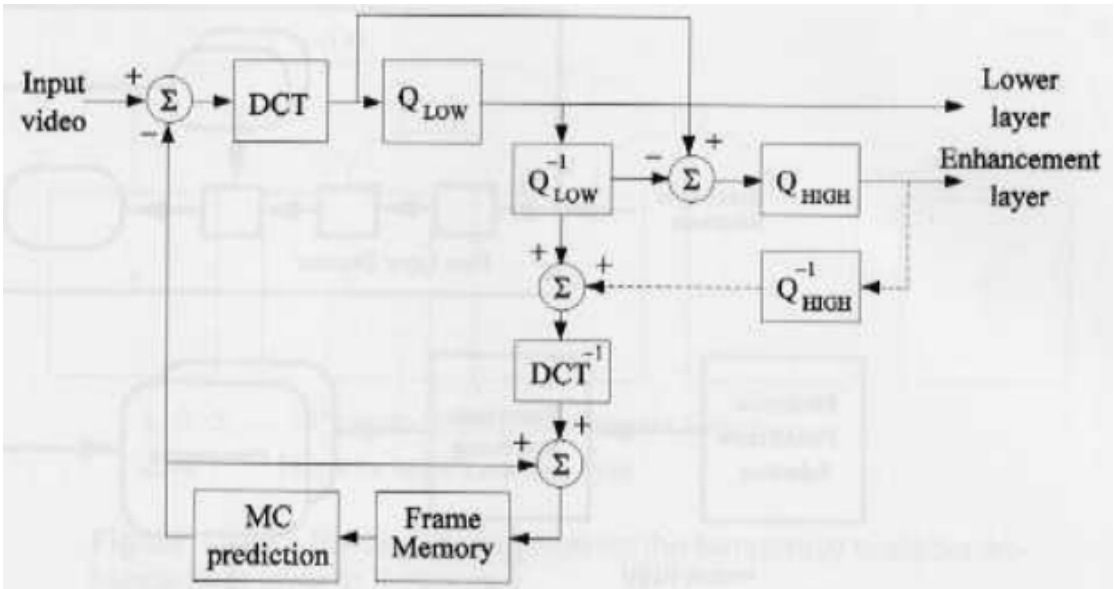


Figure 11.27 SNR scalable encoder by quantizer refinement and feedback in the prediction loop.

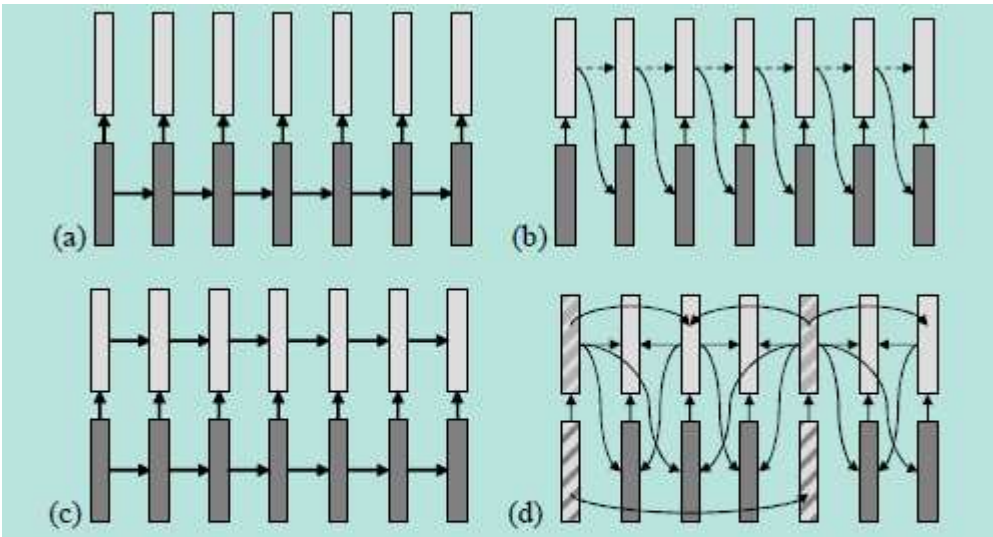
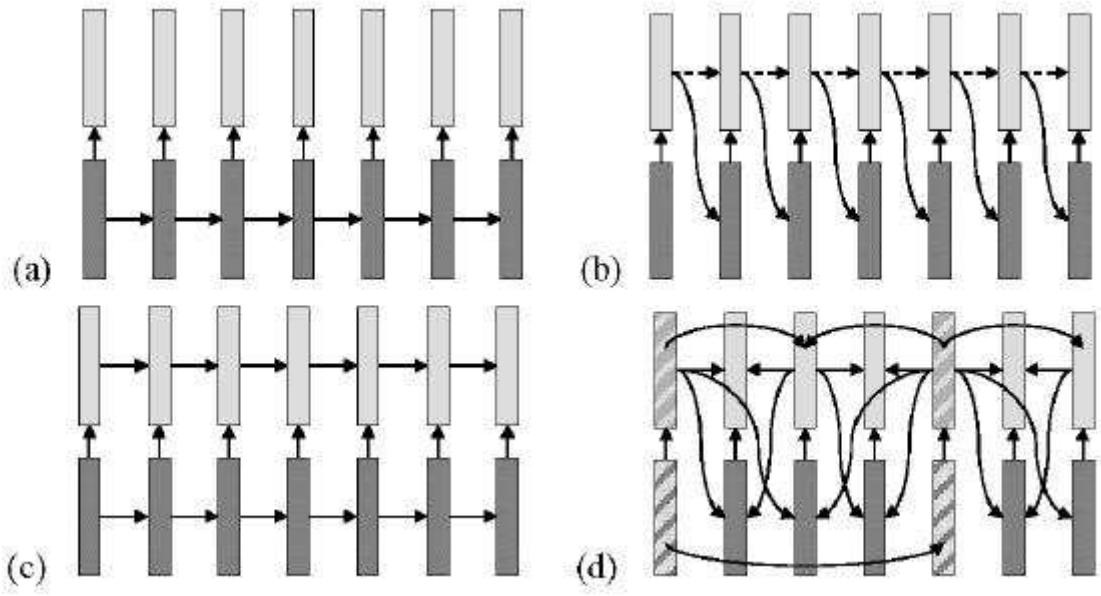


Fig. 8. Various concepts for trading off enhancement layer coding efficiency and drift for packet-based quality scalable coding: (a) base layer only control, (b) enhancement layer only control, (c) two-loop control, (d) key picture concept of SVC for hierarchical prediction structures, where key pictures are marked by the hatched boxes.

在文中出现的“参考帧”这个名词时，都是特指 inter-frame prediction 的参考，即 P，B 的参考，而不是 inter-layer prediction 的参考。

1. 认识 MGS 结构

理解 MGS 结构，先要理解下面这个图（fig.8 in Overview of the Scalable Video Coding Extension of the H.264/AVC Standard），该图描述了在 SNR 分级时，inter prediction 可能的几种选择：



(a)Base layer only，非常简单，所有的 inter prediction 全都是在 SNR BL 层完成。这样做的缺点就是编码的效率会不好。因为所有的参考帧都是 SNR BL，也就是最不“清楚”的图像。参考帧品质差，那么残差就会大，进而编出的 bit 会多。

(b)Enhancement layer only，克服了(a)的缺点，所有参考帧都用最“清楚”的图像，那么编码效率最好。这种结构仍然有问题，一旦解码端只收到了 SNR BL，而 SNR EL 都丢失了。那么很显然会造成解码出问题，并且这种问题会一直传播下去直到下一个 IDR，即 drift 效应。

(c)Two-loop，很自然可以想到用这种结构来克服(a)和(b)的问题。但很明显这样的结构仍然会有 drift 效应。

(d)Key picture，SVC 标准使用的方法，实际是一种 a)和(b)的折中方案。key picture 就是 temporal id 为 0 的 access unit（图中用条纹标明）。**对 key picture 而言，它们只能采用 SNR BL 层来实现 inter prediction，这和图(a)是完全一致的；而 non key picture，则一概使用最“清楚”的图像作为参考帧，这个图(b)是完全一样的。**

简单分析一下，编码效率角度来看由于大部分图像都是使用最“清楚”的图像作参考，因此编码效率较好；drift 控制角度来看任何由于 SNR EL 的丢弃造成的错误都不可能转播到下一个 key picture 之后，因此也不错。

2. 概念 reference base picture

出发点明确了以后，再看 SVC 标准是如何实现这样的 MGS 结构的。首先看标准上的一个概念 reference base picture：

G.3.46 reference base picture：A reference picture that is obtained by decoding a base quality layer representation with the nal_ref_idc syntax element not equal to 0 and the store_ref_base_pic_flag syntax element equal to 1 of an access unit and all layer presentations of the access unit that are referred to by inter-layer prediction of the base quality layer representation. A reference base picture is

not an output of the decoding process, but the samples of a reference base picture may be used for inter prediction in the decoding process of subsequent pictures in decoding order. Reference base picture is a collective term for a reference base field or a reference base frame.

简单说，就是 QId 等于 0 的参考图像。它可能并不是最终的输出图像（因为最终的输出图像可能是 SNR BL + SNR EL），但它在整个解码的过程中，会在某些场合下作为参考帧。结合下面的一个例子来理解

QId = 1: I1 b B1 b P1 ...

QId = 0: I0 b B0 b P0 ... (example-1)

哪些 picture 是 reference base picture？很明显按照 reference base picture 的定义 I0、P0、B0 都是，I1、P1、B1 不是（QId 不为 0），b 也不是（不是参考帧）。也很容易理解，我们最后需要输出的图像是 SNR BL + EL 的最“清楚”图像，所以 I0，P0，B0 并不会被输出。但是它们有可能会被用作参考。什么时候用作参考？这时候就和 MGS key picture 有关系了。结合上面的图(d),可以看到解码 P0 的时候 I0 是要被当作参考帧的。这就是 reference base picture 的含义。

3. 语法元素 store_ref_base_pic_flag

更进一步说，**key picture 中 reference base picture 会充当参考帧的角色，尽管不会最终输出,它们仍需要被存入 DPB 中;而 non key picture(例如 B0)的 reference base picture，它在整个解码的过程中并不会充当参考，因此不应该被放入 DPB 中占用多余资源**。那么这里引入标准中的一个语法元素 store_ref_base_pic_flag：

store_ref_base_pic_flag equal to 1 specifies that, when the value of dependency_id as specified in the NAL unit header is equal to the maximum value of dependency_id for the VCL NAL units of the current access unit, an additional representation of the coded picture that may or may not be identical to the decoded picture is marked as "used for reference". This additional representation is also referred to as reference base picture and may be used for inter prediction of following pictures in decoding order, but it is not output.

简单说，这个语法元素就是指定当前 nalu 包含的 reference base picture 是不是要被存入 DPB 中用作将来的参考。再观察可以发现，store_ref_base_pic_flag 这个语法元素一般情况下只会在 Qid=0 的 NALU 中。那么回到上面的例子，显然对于 I0 和 P0，store_ref_base_pic_flag 应该为 1；B0 的 store_ref_base_pic_flag 应该为 0。

4. 语法元素 use_ref_base_pic_flag

接下来我们再看另一个语法元素 use_ref_base_pic_flag：

use_ref_base_pic_flag equal to 1 specifies that reference base pictures are used as reference pictures for the inter prediction process. use_ref_base_pic_flag equal to 0 specifies that decoded pictures are used as reference pictures during the inter prediction process.

该语法元素属于 nal_unit_header_svc_extenstion，它的作用就是表明当前的 nalu 使用什

么作为参考。需要注意的是对于 Dependency ID 相同的 nalu，use_ref_base_pic_flag 都得是相同的。再拿一个例子来说明

QId = 1: I1 B1 P1 ...

QId = 0: I0 B0 P0 ... (example-2)

假设解码端现在 BL 和 EL 都能收到。当解码 P0 时，P0 有两个参考帧可以选用，1) I0，即 **reference base picture**；2) I0+I1，即 **decoded picture**。如果我们使用的是 MGS，那么 P0 显然是 key picture，它必须使用 I0 做参考。因此此时 P0 的 use_ref_base_pic_flag 就应该是 1。遵循标准，P1 的 use_ref_base_pic_flag 也是 1。

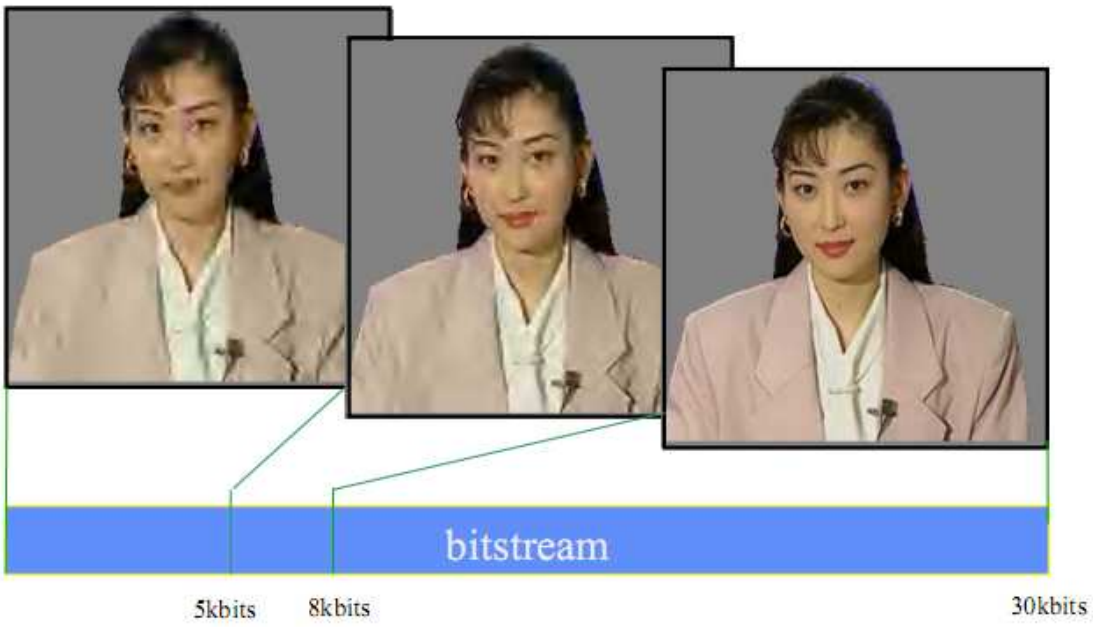
现在来看**解码 B0，根据 MGS 的结构，B0 应该采用 I0+I1 和 P0+P1**，即 decoded pictures 来当作参考，因此 B0 的 use_ref_base_pic_flag 应该是 0。同样道理，B1 的 use_ref_base_pic_flag 也是 0。下面来说明一下**use_ref_base_pic_flag 等于 0,即 decoded pictures 来当作参考**的具体含义。在上面的例子 example-2 里我们假设了 BL 和 EL 都在解码端收到了，那如果 EL 丢掉了呢？再看此时的 B0 解码，它会使用仅有的 I0，P0 来当作参考帧来解码。这就是“decoded pictures”的含义。显然这样解码的 B0、B1 会有错误，但是这里的 drift 效应不会扩散到 P0 之后。

这里有个额外的话题，我们可以发现**对于 I picture，它们的 use_ref_base_pic_flag 是没有意义的，因为它们不需要任何参考帧**。那为什么这个语法元素还要出现在 I NALU 中增加 1 bit 的冗余呢？原因我想可能是这样的：前面已经说过 use_ref_base_pic_flag 是 nal_unit_header_svc_extenstion 的一个语法元素。在 SVC 码流在传输的过程中网络中会有一些 adapting 节点，用来做 bitstream extracting。它们会 parse 每个 nalu 的 header，然后判断哪些 NALU 要被丢掉。这样的节点通常要处理大量的数据包，因此我们希望 adapting 的实现最简单经济。因此显然一致的 nalu header 结构是最适合的。如果对 I nalu 去掉这个语法元素，那么 adapting 节点还要多一个额外的判断。

5. 实例

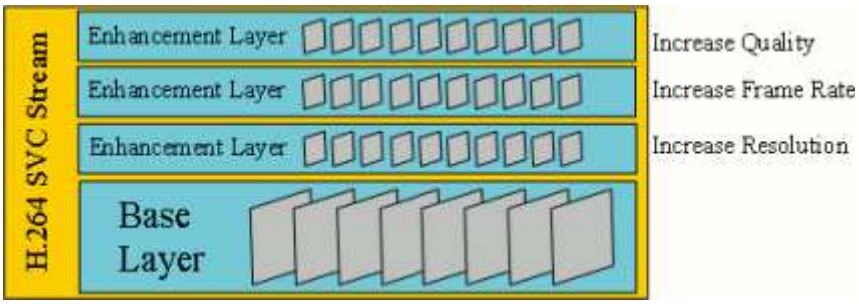
到这里基本上 MGS 的结构和标准的实现方法已经介绍完了。下面给出一个例子：
以下是从编码 trace 文件中取出的结果：
Structure:
Q2: I2 b B2 b P2
Q1: I1 b B1 b P1
Q0: I0 b B0 b P0

Picture	DTQ	use_ref_base_pic_flag	store_ref_base_pic_flag
I0	000	1(无意义)	1
I1	001	1(无意义)	无
I2	002	1(无意义)	无
P0	000	1	1
P1	001	1	无
P2	002	1	无
B0	010	0	0
B1	011	0	无
B2	012	0	无



- Bitstream 结构与 Extractor

在傳統的可調適性視訊編碼中，例如 MPEG-4 FGS，編碼器會分別產生一個基礎層與加強層的 Bitstream。如果我們可以得知目前的網路頻寬變化，只要切割加強層的資訊到目前可用的頻寬大小就可以達到位元率的調適，可是這樣的調適只在於 SNR Scalability，如果要改變畫面大小或畫面更新率時就會遇到不少問題要處理，但是**在目前的 Scalable Extension of H.264/AVC 中引入了「Layer」的概念，Bitstream 是由許多的 Layer 所組成，每個 Layer 有可能是 Temporal Layer、Spatial Layer 或 SNR Layer。而當發生網路變化或者是在不同用戶播放裝置的環境中，Extractor 則去抽取對應的 Layer 來因應各種可能的變化**，例如，偵測到目前網路頻寬是 256K，且用戶端播放裝置要 CIF 大小的畫面，且畫面更新率為 15 的時候，則 Extractor 只要讀取 Bitstream 中每個 NAL Unit 的標頭，然後挑出屬於 CIF 大小的畫面，且畫面更新率為 15 的 Layer 出來，然後看是否仍然低於要求的頻寬要求，如果是則加入更多的 SNR Layer 來滿足頻寬大小。



SVC 是为了在比特流级别上实现可伸缩性，为了这个目的，SVC 增加了 **nal_unit_header_svc_extension**，其中包含了 D(Spatial 层次号), Q(quality 层次号)以及 T(Temporal 层次号)及其他有用信息，用于方便 Bit-stream 的提取。

In order to attach SVC related information to non-SVC NAL units, **prefix_nal_unit_rbsp** are

introduced.

- Access Unit of H264 SVC

在 H.264/AVC 中，Access Unit 是一个很好理解的概念，就是若干 NAL unit 的组合（包括 VCL NALU 和相关的 non-VCL NAL unit）能解出一副完整图像，就认为是一个 access unit，标准中给出的定义为：

access unit: A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures*, one *auxiliary coded picture*, or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.

然而当进入 H.264/SVC 时代，由于多个不同 layer 的存在使得 nalu 的组合更为复杂，这个概念变得不好理解起来。而标准对此却没有相应的说明（至少目前对 JVT-x201 观察没有发现）。这篇笔记的目的就是总结对 SVC 中 access uni 概念的理解。

在“System and Transport Interface of SVC，Ye-Kui Wang,”这篇 paper 中有这样一段说明：In this paper seven logical entities are described that may consist of more than one NAL unit:

1. layer representation：对于一幅图像，具有相同 dependency_id 和 quality_id 的所有 coded slices 的组合。
2. dependency representation：对于一幅图像，具有相同 dependency_id 的所有 coded slices 组合。
3. access unit：对于一幅图像，所有的 dependency representation 再加上相关的 non-VCL NAL units。
4. scalable layer：具有相同 dependency_id、quality_id 和 temporal_id 的所有 layer representation 的组合。
5. dependency layer：具有相同 dependency_id 的 scalable layer 组合。
6. base layer：dependency_id、quality_id 都为 0 的 layer representation。
7. coded video sequence：不多说

首先需要理解，前三个概念（representation 和 access unit）都是针对某一原始图像而言的；后三个概念都是多个图像集合（layer）的概念。因此对 access unit 可以有下图说明，这里表达的是一幅原始图像在 SVC 编码中的呈现关系：

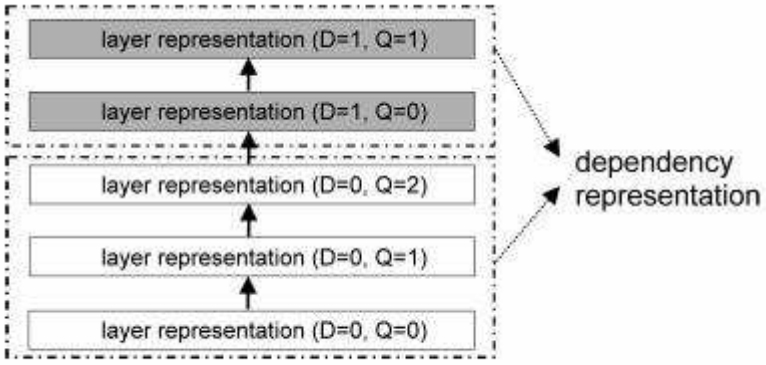
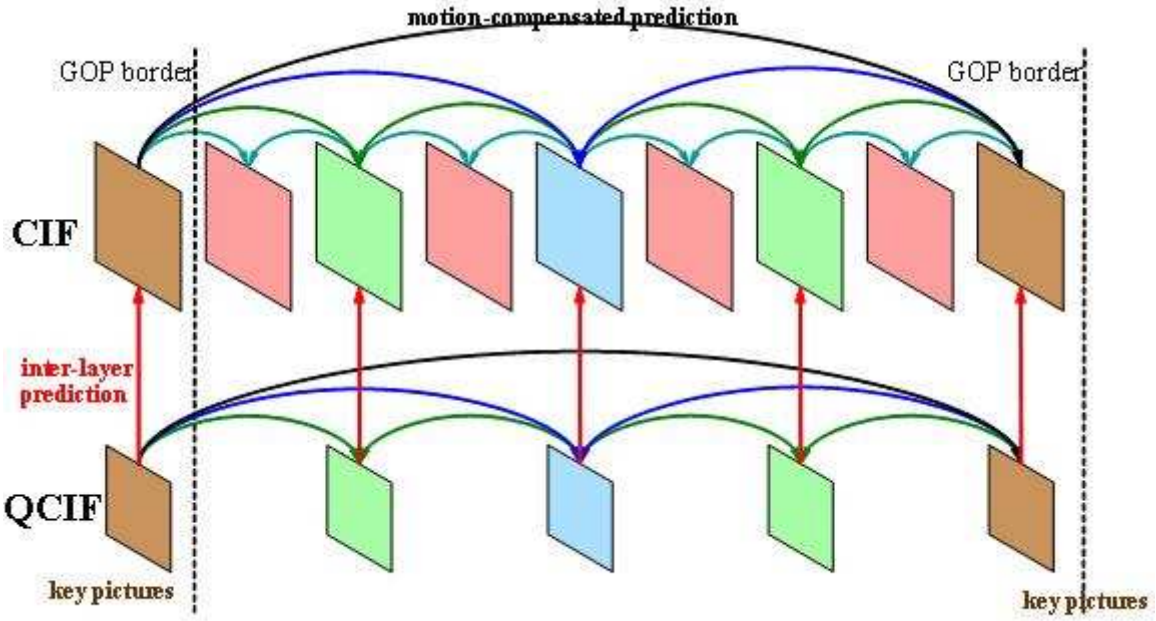
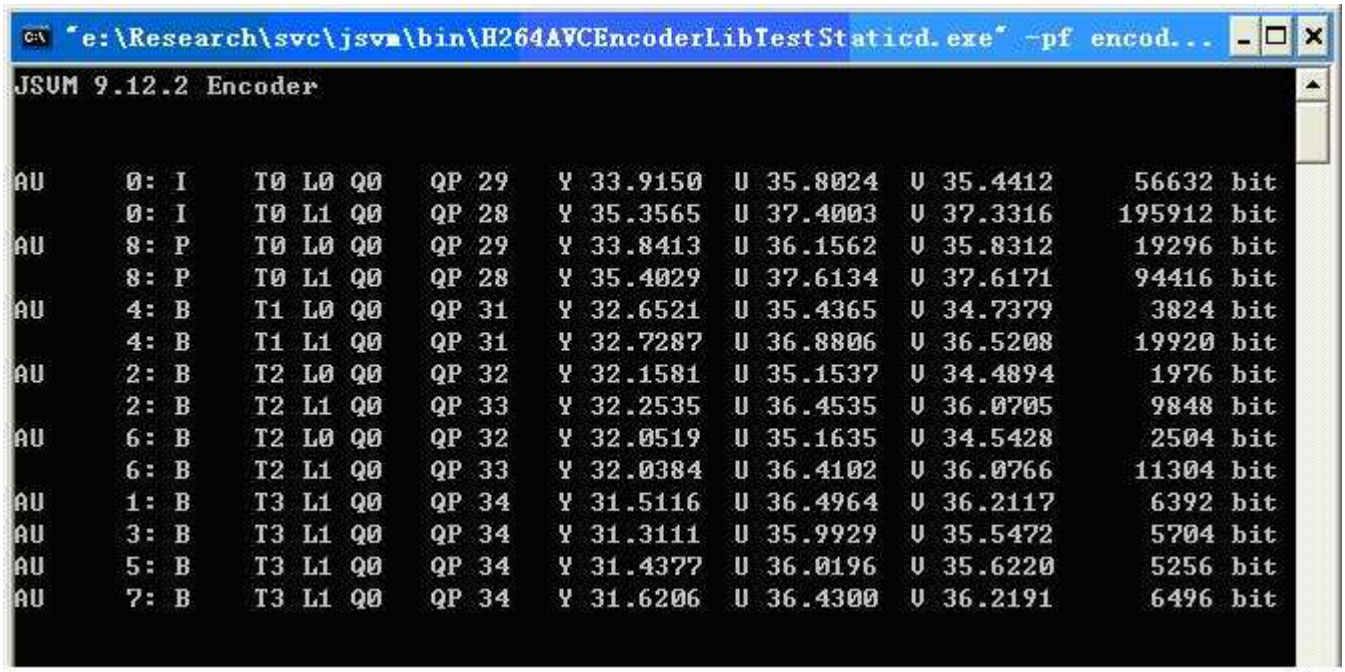


Fig. 3. Example of an access unit. D: depdency_id. Q: quality_id.

这里再举一个例子，使用 jsvm 依照下图的方式编码：



一共有几个 access unit 呢？依照上面的理解，显然共有 9 个，褐色、绿色、蓝色代表的 5 个是由两个 dependency representation 构成的 access unit，红色的 4 个则是由一个 dependency representation 构成的 access unit。当然这里没有画出与之相关的一些 non-VCL NAL unit，实际编码中是有的，比如第一帧时有 SPS、PPS、SEI，后续每帧相应的 Prefix、SEI 等。可以通过 jsvm 编码过程的信息来验证，如下图：



到这里，SVC 中的 access unit 的概念已经基本弄清楚了。

3. H264 SVC spec 及参考代码情况

目前，我们手头上有 H264 SVC spec（2007）的电子版，且为草案，没有发现有正式版的 H264 SVC spec。

- 目前关于 H264 SVC 方面的参考代码比较少，主要的就是 JVSM
- 开源软件中有发现一个项目是关于 H264 SVC 的 – Open SVC
- 初步来看，这些参考代码的复杂度都比较高，且可读性也很差，从中获取有用信息可能要花费较多时间

由于 H264 SVC 是在 H264/AVC 基础上发展起来的，具有较高的复杂度，所以除了 H264 本身编解码复杂度之外，还额外包括了 H264 SVC 特有的层间预测编解码复杂度，因此在实现上存在一定的难度。

4. H264 SVC 中增加的语法（24 个）

- 除了增加的语法，其他语法和 H264/AVC 相同

1. NAL unit header SVC extension （3 字节）

   SVC 是为了在比特流级别上实现可伸缩性，为了这个目的，SVC 增加了 nal_unit_header_svc_extension，其中包含了 D(Spatial 层次号), Q(quality 层次号)以及 T(Temporal 层次号)及其他有用信息，用于方便 Bit-stream 的提取

G.7.3.1.1 NAL unit header SVC extension syntax

nal_unit_header_svc_extension( ) {	C	Descriptor
reserved_one_bit	All	u(1)
idr_flag	All	u(1)
priority_id	All	u(6)
no_inter_layer_pred_flag	All	u(1)
dependency_id	All	u(3)
quality_id	All	u(4)
temporal_id	All	u(3)
use_ref_base_pic_flag	All	u(1)
discardable_flag	All	u(1)
output_flag	All	u(1)
reserved_three_2bits	All	u(2)
}		

2. Subset sequence parameter set RBSP

G.7.3.2.1.3 Subset sequence parameter set RBSP syntax

subset_seq_parameter_set_rbsp( ) {	C	Descriptor
seq_parameter_set_data( )	0	
if(profile_idc == 83 || profile_idc == 86) {		
seq_parameter_set_svc_extension( )	0	
svc_vui_parameters_present_flag	0	u(1)
if(svc_vui_parameters_present_flag == 1)		
svc_vui_parameters_extension( )	0	
}		
additional_extension2_flag	0	u(1)
if(additional_extension2_flag == 1)		
while(more_rbsp_data( ))		
additional_extension2_data_flag	0	u(1)
rbsp_trailing_bits( )	0	
}		

3. Sequence parameter set SVC extension

G.7.3.2.1.4 Sequence parameter set SVC extension syntax

seq_parameter_set_svc_extension( ) {	C	Descriptor
inter_layer_deblocking_filter_control_present_flag	0	u(1)
extended_spatial_scalability	0	u(2)
if(ChromaArrayType == 1 || ChromaArrayType == 2)		
chroma_phase_x_plus1_flag	0	u(1)
if(ChromaArrayType == 1)		
chroma_phase_y_plus1	0	u(2)
if(extended_spatial_scalability == 1) {		
if(ChromaArrayType > 0) {		
seq_ref_layer_chroma_phase_x_plus1_flag	0	u(1)
seq_ref_layer_chroma_phase_y_plus1	0	u(2)
}		
seq_scaled_ref_layer_left_offset	0	se(v)
seq_scaled_ref_layer_top_offset	0	se(v)
seq_scaled_ref_layer_right_offset	0	se(v)
seq_scaled_ref_layer_bottom_offset	0	se(v)
}		
seq_tcoeff_level_prediction_flag	0	u(1)
if(seq_tcoeff_level_prediction_flag) {		
adaptive_tcoeff_level_prediction_flag	0	u(1)
}		
slice_header_restriction_flag	0	u(1)
}		

4. Prefix NAL unit RBSP

In order to attach SVC related information to non-SVC NAL units, prefix_nal_unit_rbsp are introduced.

G.7.3.2.12 Prefix NAL unit RBSP syntax

prefix_nal_unit_rbsp( ) {	C	Descriptor
if(nal_ref_idc != 0) {		
store_ref_base_pic_flag	2	u(1)
if((use_ref_base_pic_flag || store_ref_base_pic_flag) && !idr_flag)		
dec_ref_base_pic_marking( )	2	
prefix_nal_unit_additional_extension_flag	2	u(1)
if(prefix_nal_unit_additional_extension_flag == 1)		
while(more_rbsp_data( ))		
prefix_nal_unit_extension_flag	2	u(1)
rbsp_trailing_bits( )	2	
}		
}		

5. Slice layer in scalable extension RBSP

G.7.3.2.13 Slice layer in scalable extension RBSP syntax

slice_layer_in_scalable_extension_rbsp( ) {	C	Descriptor
slice_header_in_scalable_extension( )	2	
if(!slice_skip_flag)		
slice_data_in_scalable_extension( )	2 | 3 | 4	
rbsp_slice_trailing_bits( )	2	
}		

6. Slice header in scalable extension

G.7.3.3.4 Slice header in scalable extension syntax

slice_header_in_scalable_extension( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
if(separate_colour_plane_flag == 1)		
colour_plane_id	2	u(2)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(idr_flag == 1)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type == 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type == 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0]	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		

7. Decoded reference base picture marking

G.7.3.3.5 Decoded reference base picture marking syntax

dec_ref_base_pic_marking( ) {	C	Descriptor
adaptive_ref_base_pic_marking_mode_flag	2 | 5	u(1)
if(adaptive_ref_base_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2 | 5	ue(v)
if(memory_management_control_operation == 1)		
difference_of_pic_nums_minus1	2 | 5	ue(v)
if(memory_management_control_operation == 2)		
long_term_pic_num	2 | 5	ue(v)
} while(memory_management_control_operation != 0)		
}		

8. Slice data in scalable extension

G.7.3.4.1 Slice data in scalable extension syntax

slice_data_in_scalable_extension( ) {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != EI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i = 0; i < mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && ((CurrMbAddr % 2) == 0 || ((CurrMbAddr % 2) == 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer_in_scalable_extension( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != EI)		
prevMbSkipped = mb_skip_flag		

9. Macroblock layer in scalable extension

G.7.3.6 Macroblock layer in scalable extension syntax

macroblock_layer_in_scalable_extension( ) {	C	Descriptor
if(in_crop_window(CurrMbAddr) && adaptive_base_mode_flag)		
base_mode_flag	2	u(1) | ae(v)
if(!base_mode_flag)		
mb_type	2	ue(v) | ae(v)
if(!base_mode_flag && mb_type == I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	3	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i **]**	3	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i **]**	3	u(v)
} else {		
if(! base_mode_flag) {		
noSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) == 4) {		
sub_mb_pred_in_scalable_extension(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type [mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type == I_NxN)		
transform_size_8x8_flag	2	u(1) | ae(v)

10. Macroblock prediction in scalable extension

G.7.3.6.1 Macroblock prediction in scalable extension syntax

mb_pred_in_scalable_extension(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) == Intra_4x4 || MbPartPredMode(mb_type, 0) == Intra_8x8 || MbPartPredMode(mb_type, 0) == Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) == Intra_4x4)		
for(luma4x4BlkIdx = 0; luma4x4BlkIdx < 16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) == Intra_8x8)		
for(luma8x8BlkIdx = 0; luma8x8BlkIdx < 4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(ChromaArrayType != 0)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
if(in_crop_window(CurrMbAddr) && adaptive_motion_prediction_flag) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx **]**	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx **]**	2	u(1) | ae(v)
}		

11. Sub-macroblock prediction in scalable extension

G.7.3.6.2 Sub-macroblock prediction in scalable extension syntax

sub_mb_pred_in_scalable_extension(mb_type) {	C	Descriptor
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
if(in_crop_window(CurrMbAddr) && adaptive_motion_prediction_flag) {		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Direct && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx]	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Direct && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx]	2	u(1) | ae(v)
}		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1 && !motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0 && !motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		

12. Scalability information SEI message

G.13.1.1 Scalability information SEI message syntax

scalability_info(payloadSize) {	C	Descriptor
temporal_id_nesting_flag	5	u(1)
priority_layer_info_present_flag	5	u(1)
priority_id_setting_flag	5	u(1)
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i]	5	ue(v)
priority_id[i]	5	u(6)
discardable_flag[i]	5	u(1)
dependency_id[i]	5	u(3)
quality_id[i]	5	u(4)
temporal_id[i]	5	u(3)
sub_pic_layer_flag[i]	5	u(1)
sub_region_layer_flag[i]	5	u(1)
iroi_division_info_present_flag[i]	5	u(1)
profile_level_info_present_flag[i]	5	u(1)
bitrate_info_present_flag[i]	5	u(1)
frm_rate_info_present_flag[i]	5	u(1)
frm_size_info_present_flag[i]	5	u(1)
layer_dependency_info_present_flag[i]	5	u(1)
parameter_sets_info_present_flag[i]	5	u(1)
bitstream_restriction_info_present_flag[i]	5	u(1)
exact_inter_layer_pred_flag[i]	5	u(1)
if(sub_pic_layer_flag[i] || iroi_division_info_present_flag[i])		

13.

G.13.1.2 Layers not present SEI message syntax

layers_not_present(payloadSize) {	C	Descriptor
num_layers	5	ue(v)
for(i = 0; i < num_layers; i++) {		
layer_id[i]	5	u(8)
}		
}		

14.

G.13.1.3 Layer dependency change SEI message syntax

layer_dependency_change (payloadSize) {	C	Descriptor
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i]	5	u(8)
layer_dependency_info_present_flag[i]	5	u(1)
if (layer_dependency_info_present_flag[i]) {		
num_directly_dependent_layers[i]	5	ue(v)
for (j = 0; j < num_directly_dependent_layers[i]; j++)		
directly_dependent_layer_id_delta_minus1[i][j]	5	ue(v)
} else {		
layer_dependency_info_src_layer_id_delta_minus1[i]	5	ue(v)
}		
}		
}		

15.

G.13.1.4 Sub-picture scalable layer SEI message syntax

sub_pic_scalable_layer(payloadSize) {	C	Descriptor
layer_id	5	ue(v)
}		

16.

G.13.1.5 Non-required layer representation SEI message syntax

non_required_layer_rep(payloadSize) {	C	Descriptor
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
entry_dependency_id[i]	5	u(3)
num_non_required_layer_rep_minus1[i]	5	ue(v)
for(j = 0; j <= num_non_required_layer_rep_minus1[i]; j++) {		
non_required_layer_rep_dependency_id[i][j]	5	u(3)
non_required_layer_rep_quality_id[i][j]	5	u(4)
}		
}		
}		

17.

G.13.1.6 Priority layer information SEI message syntax

priority_layer_info(payloadSize) {	C	Descriptor
pr_dependency_id	All	u(3)
num_priority_ids	5	u(4)
for(i = 0; i < num_priority_ids; i++) {		
alt_priority_id[i **]**	5	u(6)
}		
}		

18.

G.13.1.7 Scalable nesting SEI message syntax

scalable_nesting(payloadSize) {	C	Descriptor
all_layer_representations_in_au_flag	5	u(1)
if(all_layer_representations_in_au_flag = = 0) {		
num_layer_representations_minus1	5	ue(v)
for(i = 0; i <= num_layer_representations_minus1; i++) {		
sei_dependency_id[i **]**	5	u(3)
sei_quality_id[i **]**	5	u(4)
}		
sei_temporal_id	5	u(3)
}		
while(!byte_aligned( ))		
sei_nesting_zero_bit /* equal to 0 */	5	f(1)
do		
sei_message( )	5	
while(more_rbsp_data( ))		
}		

19.

G.13.1.8 Base layer temporal HRD SEI message syntax

base_layer_temporal_hrd (payloadSize) {	C	Descriptor
num_of_temporal_layers_in_base_layer_minus1	5	ue(v)
for(i = 0; i < num_of_temporal_layers_in_base_layer_minus1; i++){		
temporal_id[i]	5	u(3)
timing_info_present_flag[i]	5	u(1)
if(timing_info_present_flag[i]) {		
num_units_in_tick[i]	5	u(32)
time_scale[i]	5	u(32)
fixed_frame_rate_flag[i]	5	u(1)
}		
nal_hrd_parameters_present_flag[i]	5	u(1)
if(nal_hrd_parameters_present_flag[i])		
hrd_parameters( )		
vcl_hrd_parameters_present_flag[i]	5	u(1)
if(vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )		
if(nal_hrd_parameters_present_flag[i] || vcl_hrd_parameters_present_flag[i])		
low_delay_hrd_flag[i]	5	u(1)
pic_struct_present_flag[i]	5	u(1)
}		
}		

20.

G.13.1.9 Quality layer integrity check SEI message syntax

quality_layer_integrity_check(payloadSize) {	C	Descriptor
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
entry_dependency_id[i]	5	u(3)
quality_layer_crc[i]	5	u(16)
}		
}		

21.

G.13.1.10 Redundant picture property SEI message syntax

redundant_pic_property(payloadSize) {	C	Descriptor
num_dId_minus1	5	ue(v)
for(i = 0; i <= num_dId_minus1; i++) {		
dependency_id[i]	5	u(3)
num_qId_minus1[i]	5	ue(v)
for(j = 0; j <= num_qId_minus1[i]; j++) {		
quality_id[i][j]	5	u(4)
num_redundant_pics_minus1[i][j]	5	ue(v)
for(k = 0; k <= num_redundant_pics_minus1[i][j]; k++) {		
redundant_pic_cnt_minus1[i][j][k]	5	ue(v)
pic_match_flag[i][j][k]	5	u(1)
if(!pic_match_flag[i][j][k]) {		
mb_type_match_flag[i][j][k]	5	u(1)
motion_match_flag[i][j][k]	5	u(1)
residual_match_flag[i][j][k]	5	u(1)
intra_samples_match_flag[i][j][k]	5	u(1)
}		
}		
}		
}		
}		

22.

G.13.1.11 Temporal level zero dependency representation index SEI message syntax

tl0_dep_rep_index(payloadSize) {	C	Descriptor
tl0_dep_rep_idx	5	u(8)
effective_idr_pic_id	5	u(16)
}		

23.

G.13.1.12 Temporal level switching point SEI message syntax

tl_switching_point(payloadSize) {	C	Descriptor
delta_frame_num	5	se(v)
}		

24. SVC VUI parameters extension

G.14.1 SVC VUI parameters extension syntax

svc_vui_parameters_extension( ) {	C	Descriptor
num_layers_minus1	0	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
dependency_id[i]	0	u(3)
quality_id[i]	0	u(4)
temporal_id[i]	0	u(3)
timing_info_present_flag[i]	0	u(1)
if(timing_info_present_flag[i]) {		
num_units_in_tick[i]	0	u(32)
time_scale[i]	0	u(32)
fixed_frame_rate_flag[i]	0	u(1)
}		
nal_hrd_parameters_present_flag[i]	0	u(1)
if(nal_hrd_parameters_present_flag[i])		
hrd_parameters( )		
vcl_hrd_parameters_present_flag[i]	0	u(1)
if(vcl_hrd_parameters_present_flag[i])		
hrd_parameters( )		
if(nal_hrd_parameters_present_flag[i] || vcl_hrd_parameters_present_flag[i])		
low_delay_hrd_flag[i]	0	u(1)
pic_struct_present_flag[i]	0	u(1)
}		
}		