H264 传输/NAL层

H.264码流第一个 NALU 是 SPS（序列参数集Sequence Parameter Set）

H.264码流第二个 NALU 是 PPS（图像参数集Picture Parameter Set）

H.264码流第三个 NALU 是 IDR（即时解码器刷新）

H.264 NALU语法结构

NALU

NAL网络提取层 里面放一些与网络相关的信息，定义片级以上的语法级别（如序列参数集和图像参数集，针对网络传输），

同时支持以下功能：独立片解码，起始码唯一保证，SEI以及流格式编码数据传送，

NAL层将SODB打包成RBSP然后加上NAL头，组成一个NALU（NAL单元）

(??MediaCodec编码出来的数据 已经是NAL??)

(??看数据封装RBSP EBSP就可知道， NAL负责格式化数据并提供头信息，以保证数据适合各种信道和存储介质上的传输 ??)

VCL 视频编码层 对核心算法引擎，块，宏块及片的语法级别的定义，他最终输出编码完的数据 SODB

VCL NAL 单元是指那些nal\_unit\_type 值等于 1 到 5(包括 1 和 5)的 NAL 单元

(1：非IDR图像的编码条带/编码slice

2：编码条带/编码slice的数据划分A，

3：…数据划分B

4:…数据划分C

5.IDR图像的编码条带 )，

这些单元都包含了视频数据。所有其他的 NAL 单元都称作非 VCL NAL 单元，PPS和SPS都是非VCL NAL单元。

(??VCL层不会给出编码出一个SPS PPS单元，只会有图像数据，NAL层要获得SPS/PPS参数封装成NAL单元发出去??)

VCL层出来的是编码完的视频帧数据，这些帧可能是I、B、P帧，而且这些帧可能属于不同的序列(GOP/帧组 IDR分开)

再者同一个序列还有相对应的一套序列参数集和图片参数集等等，

       所以要完成视频的解码，不仅需要传输VCL层编码出来的视频帧数据，还需要传输序列参数集、图像参数集等数据。

参数集 包括序列参数集 SPS  和图像参数集 PPS  
       SPS

包含的是针对一连续编码视频序列的参数，如标识符 seq\_parameter\_set\_id、帧数及 POC 的约束、参考帧数目、解码图像尺寸和帧场编码模式选择标识等等。(??MP4文件/容器中可以从sps中获取总帧数 然后除以时长duration 就是帧率??)(??帧场编码模式 如果是隔行扫描 e.g 1080i 就可以知道先发过来的是奇偶场??)

       PPS

对应的是一个序列中某一幅图像或者某几幅图像，其参数如标识符 pic\_parameter\_set\_id、可选的 seq\_parameter\_set\_id、熵编码模式选择标识、片组数目、初始量化参数和去方块滤波系数调整标识等等。

数据分割：(数据划分”片段/slice/编码条带”)组成片的编码数据存放在 3 个独立的 DP（数据分割，A类、B类、C类）中，各自包含一个编码片的子集。

       分割Ａ包含片头和片中每个宏块头数据。

       分割Ｂ包含帧内和 SI 片宏块的编码残差数据。

       分割 C包含帧间宏块的编码残差数据。

       每个分割可放在独立的 NAL 单元并独立传输。

Slice是片的意思/片段/编码条带，H264中把图像分成一帧（frame）或两场（field），而帧又可以分成一个或几个片（Slilce）；片由宏块（MB）组成。宏块是编码处理的基本单元

除了流开头的字节流NAL单元，大多字节流NAL单元的开头没有leading\_zero\_8bits （一个字节的0）

所有字节流NAL的单元的开头，都会存在 start\_code\_prefix\_one\_3bytes （三个000001字节）。所以，如果不考虑流的开头，没有特殊性的字节流NAL单元，都是只有3字节的起始码

一个访问单元的第一个NAL单元是指：

在基本编码图像的最后一个 VCL NAL 单元之后的第一个任何下列 NAL 单元代表了一个新的访问单元的开始：  
— 访问单元分隔 NAL 单元(存在时)  
— 序列参数集 NAL 单元(存在时)  
— 图像参数集 NAL 单元(存在时)  
— SEI NAL 单元(存在时)  
— nal\_unit\_type 值在 14-18 之间(包括)的 NAL 单元  
— 基本编码图像的第一个 VCL NAL 单元(总是存在)

片分为I片、B片、P片和其他一些片。

I片只包含I宏块，P片可包含P和I宏块，而B片可包含B和I宏块。

I宏块利用从当前片中已解码的像素作为参考进行帧内预测。

P宏块利用前面已编码图象作为参考图象进行~~帧内预测~~ 帧间预测。

B宏块则利用双向的参考图象（前一帧和后一帧）进行~~帧内预测~~帧间预测。

宏块

一个编码图像通常划分成若干宏块组成

一个宏块由一个16×16亮度像素和附加的一个8×8 Cb和一个8×8 Cr彩色像素块组成。(YUV420???)

H264结构中，

一个视频图像编码后的数据叫做一帧，

一帧由一个片（slice）或多个片组成，

一个片由一个或多个宏块（MB）组成，

一个宏块由16x16的yuv数据组成。宏块作为H264编码的基本单位。

H264编码过程中的三种不同的数据形式：(??从编码后的数据 到 可网络传输的数据??)

SODB 数据比特串 ----＞最原始的编码数据，即VCL数据；

RBSP　 原始字节序列载荷 ----＞在SODB的后面填加了结尾比特，RBSP trailing bits　一个bit“1”，若干比特“0”,以便字节对齐；

EBSP　 扩展字节序列载荷 ---- > 在RBSP基础上填加了仿校验字节（0X03）

它的原因是：　在NALU加到Annexb上时，需要添加每组NALU之前的开始码StartCodePrefix,如果该NALU对应的slice为一帧的开始则用4位字节表示，0x00000001,否则用3位字节表示0x000001（是一帧的一部分）。另外，为了使NALU主体中不包括与开始码相冲突的，在编码时，每遇到两个字节连续为0，就插入一个字节的0x03。解码时将0x03去掉。也称为脱壳操作。

(也就是编码数据VCL不适用于网络传输，必须经过处理EBSP，把编码数据中两个连续0x00字节加入0x03，这样就可以区分0x00 00 00 03 0x00 00 03的startCodePrefix还是0x00 00 03 编码数据)

NAL的开始和结束  
  
编码器将每个NAL各自独立、完整地放入一个分组，因为分组都有头部，解码器可以方便地检测出NAL的分界，并依次取出NAL进行解码。

每个NAL前有一个起始码 0x00 00 01（或者0x00 00 00 01），解码器检测每个起始码，作为一个NAL的起始标识，当检测到下一个起始码时，当前NAL结束。

同时H.264规定，当检测到0x000000时，也可以表征当前NAL的结束。那么NAL中数据(VCL编码数据)出现0x000001或0x000000时怎么办？H.264引入了防止竞争机制，如果编码器检测到NAL数据存在0x000001或0x000000时，编码器会在最后个字节前插入一个新的字节0x03，(??? EBSP 扩展字节序列载荷 ??? )这样：

0x0000 00－>0x0000 03 00  
0x0000 01－>0x0000 03 01  
0x0000 02－>0x0000 03 02  
0x0000 03－>0x0000 03 03 （也就是遇到连续两个字节都是0x0就在后面加上0x03）  
解码器检测到0x000003时，把03抛弃，恢复原始数据（脱壳操作）。  
  
 NALU的顺序要求   
  
 H.264/AVC标准对送到解码器的NAL单元顺序是有严格要求的，如果NAL单元的顺序是混乱的，必须将其重新依照规范组织后送入解码器，否则解码器不能够正确解码。

1.序列参数集NAL单元       SPS (非VCL的NAL单元 不是编码数据的NAL单元)

必须在传送所有以此参数集为参考的其他NAL单元之前传送，但允许这些NAL单元中间出现重复的序列参数集NAL单元。

所谓重复的详细解释为：序列参数集NAL单元都有其专门的标识，如果两个序列参数集NAL单元的标识相同，就可以认为后一个只不过是前一个的拷贝，而非新的序列参数集。

2.图像参数集NAL单元      PPS (非VCL的NAL单元不是编码数据的NAL单元)

必须在所有以此参数集为参考的其他NAL单元之前传送，但允许这些NAL单元中间出现重复的图像参数集NAL单元，这一点与上述的序列参数集NAL单元是相同的。

3.不同基本编码图像中的片段（slice）单元和数据划分片段（data partition）单元在顺序上不可以相互交叉，即不允许属于**某一基本编码图像**的一系列片段（slice 非IDR图像的编码条带nal\_type=1 IDR图像的编码条带nal\_type=5）单元和数据划分片段（data partition 非IDR图像的编码条带的A/B/C类数据分段 ）单元中忽然出现**另一个基本编码图像**的片段（slice）单元片段和数据划分片段（data partition）单元。

4.参考图像的影响：如果**一幅图像**以另一幅图像为参考，则属于**前者**的**所有片段（slice）单元和数据划分片段（data partition**）单元必须在属于后者的片段和数据划分片段之后，无论是基本编码图像还是冗余编码图像都必须遵守这个规则。

5.**基本编码图像**的所有片段（slice）单元和数据划分片段（data partition）单元必须在属于相应**冗余编码图像**的片段（slice）单元和数据划分片段（data partition）单元之前。

6.如果数据流中出现了连续的无参考基本编码图像，则图像序号小的在前面。

7.如果arbitrary\_slice\_order\_allowed\_flag置为1，**一个基本编码图像**中的**片段（slice 编码条带）**单元和**数据划分片段（data partition）**单元的**顺序是任意的**，如果arbitrary\_slice\_order\_allowed\_flag置为0，则要按照片段中第一个宏块的位置来确定**片段的顺序（编码条带）**，若使用数据划分，则**A类**数据划分片段在**B类**数据划分片段之前，B类数据划分片段在**C类**数据划分片段之前，而且对应不同片段的数据划分片段不能相互交叉，也不能与没有数据划分的片段相互交叉。

8.如果存在SEI（补充增强信息）单元的话，它必须在它所对应的基本编码图像的片段（slice）单元和数据划分片段（data partition）单元之前，并同时必须紧接在上一个基本编码图像的所有片段（slice）单元和数据划分片段（data partition）单元后边。假如SEI属于多个基本编码图像，其顺序仅以第一个基本编码图像为参照。

9.如果存在**图像分割符**的话，它必须在所有SEI 单元、基本编码图像的所有片段slice）单元和数据划分片段（data partition）单元之前，并且紧接着上一个基本编码图像那些NAL单元。

10.如果存在**序列结束符（GOP）**，且序列结束符后还有图像，则该图像**必须是IDR（即时解码器刷新）图像**。序列结束符的位置应当在属于这个IDR图像的分割符、SEI 单元等数据之前，且紧接着前面那些图像的NAL单元。如果序列结束符后没有图像了，那么它的就在比特流中所有图像数据之后。

11.**流结束符**在比特流中的最后。

对于 H.264中出现的一些概念从大到小排序依次是：**序列、图像、片组、片、NALU、宏块、亚宏块、块、像素**。

一幅图像由 1～N个片组组成，而每一个片组又由一个或若干个片组成一个片由一个NALU或三个NALU（假如有数据分割）组成。图像解码过程中总是按照片进行解码，然后按照片组将解码宏块重组成图像。从这种意义上讲，片实际是最大的解码单元。

在 H.264协议中**图像是个集合概念**，**顶场、底场、帧都可以称为图像**（本文图像概念时都是集合概念）

**帧、P 帧、B帧等等，实际上都是我们把图像这个概念具体化和细小化了**。我们**在 H.264里提到的“帧”通常就是指不分场的图像**

如果**不采用FMO（灵活宏块排序）** 机制，则**一幅图像只有一个片组**

如果**不使用多个片，则一个片组只有一个片**

如果**不采用DP**（**数据分割 Data partition**）机制，则**~~一个片就是一个NALU~~** ~~DP~~，**~~一个 NALU 也就是一个片~~ 一个片就最终包含在一个NALU中**

 否则，**~~一个片由 三个 NALU 组成~~** 一个片由三个DP组成，每个DP分别包含在一个NALU中，共3个NALU（即nal\_unit\_type 值为2、3、4 的**三个** NALU 属于 一个片，为片下的三个DP）

nal\_unit\_type **= 2 编码条带数据分割块A  slice\_data\_partition\_a\_layer\_rbsp()**  
nal\_unit\_type **= 3 编码条带数据分割块B slice\_data\_partition\_b\_layer\_rbsp( )**  
nal\_unit\_type **= 4 编码条带数据分割块C slice\_data\_partition\_c\_layer\_rbsp( )**

?? 片又分为 I B P SI PI 等slice\_type （注：IDR图像时，slice\_type为2,4,7,9） （??实际只有 P B I slice和 SP SI slice??）???

|  |  |
| --- | --- |
| **slice\_type** | **name** |
| 0 | P slice |
| 1 | B slice |
| 2 | I slice |
| 3 | SP slice |
| 4 | SI slice |
| 5 | P slice |
| 6 | B slice |
| 7 | I slice |
| 8 | SP slice |
| 9 | SI slice |

**~~一幅图像根据组成它的片类型来分，可以分为标准“表7-5”中的 8种类型。我们平常应用中所最常见到的其实是这些类型的特例。例如：我们平常所谓的“I帧”和“IDR 帧”，其实是 primary\_pic\_type 值为 0的图像，我们平常所谓的“P帧”其实是 primary\_pic\_type 值为 1的图像的特例，我们平常所谓的“B帧”其实是 primary\_pic\_type 值为 2的图像的特例。~~**

**一幅图像根据概念来分可以分为两种：IDR 图像和非 IDR图像。一幅图像是否是 IDR 图像是由组成该图像的 NALU决定的，如果组成该图像的 NALU 为标准“表7-1”中 nal\_unit\_type 值为 5 的NALU，则该图像为 IDR 图像，否则为非 IDR图像。这里也有几点值得说明：**

**（1）、nal\_unit\_type 值为 5 的 NALU 只会出现在 IDR 图像中，而 IDR图像中的所有 NALU 都是nal\_unit\_type 值为 5 的NALU；**

**（2）、我们以组成一幅图像的片的类型(slice\_type)来区分该图像是否是IDR 图像是错误的。**

**例如：一幅图像中的所有片都是 I 片并不代表这幅图像就是 IDR 图像。因为 I片也可以从属于 nal\_unit\_type 值为 1 的 NALU 也即非IDR图像有可能全部包含I片。只不过我们平常最常见到的形式是：所有片都是 I片的图像就是 IDR 图像。其实这个时候 IDR 图像的概念也被我们具体化和细小化了。但IDR图像必定全部包含I片或SI片，不过只有用NALU的类型才能判断是不是IDR图像**

RTP协议的H.264视频传输

NALU H.264 的基本流（elementary stream,ES）的结构分为两层，包括视频编码层（VCL）和网络适配层（NAL）。

视频编码层负责高效的视频内容表示，而网络适配层负责以网络所要 求的恰当的方式对数据进行打包和传送。

H.264 的基本流由一系列NALU （Network Abstraction Layer Unit ）组成，不同的NALU数据量各不相同。

在每个NALU 前添加起始码：0x000001(起始码不属于NALU，但在NALU header之前，用来隔开每个NALU)，用来指示一个 NALU的起始和终止位置。

每个 NALU单元由一个字节的 NALU头（NALU Header）和若干个字节的载荷数据（RBSP）组成

(上面是一个字节 0 对应的是byteBuffer的最高位 header[4]&0x80 )

( spyDroid中，会根据 MediaCodec::dequeueOutputBuffer buffer[0,1,2,3] 4个字节 确认是否一个NAL开始 00 00 00 01

然后根据 第5个字节 buffer[4] && 0x1F 取出 NALU的类型 ，如果是7/8 代表负载数据是 sps pps 等 )

NALU Header 一般:

0x67 0 11 0 0111 nal\_unit\_type为低5位， 0 0111 = 7 nal\_ref\_idc=3 => SPS\_FRAME

0x68 0 11 0 1000 nal\_unit\_type为低5位， 0 1000 = 8 nal\_ref\_idc=3 => PPS\_FRAME

0x65 0 11 0 0101 nal\_unit\_type为低5位， 0 0101 = 5 nal\_ref\_idc=3 => I\_FRAME

0x41 0 10 0 0001 nal\_unit\_type为低5位， 0 0001 = 1 nal\_ref\_idc=2 => P\_FRAME

F：forbidden\_zero\_bit.1 位，禁止位

如果有语法冲突，则为 1。当网络识别此单元存在比特错误时，可将其设为 1，以便接收方丢掉该单元。

NRI：nal\_ref\_idc.2 位，参考级别

用来指示该NALU 的重要性等级。值越大，表示当前NALU越重要。具体大于0 时取何值，没有具体规定。

解码器在解码处理不过来的时候，可以丢掉重要性为0的NALU

Type：5 位，帧的类型

指出NALU 的类型。具体如表1 所示：

H264定义的类型 values for nal\_unit\_type

typedef enum {

NALU\_TYPE\_SLICE = 1,

NALU\_TYPE\_DPA = 2,

NALU\_TYPE\_DPB = 3,

NALU\_TYPE\_DPC = 4,

NALU\_TYPE\_IDR = 5,

NALU\_TYPE\_SEI = 6,

NALU\_TYPE\_SPS = 7,

NALU\_TYPE\_PPS = 8,

NALU\_TYPE\_AUD = 9,

NALU\_TYPE\_EOSEQ = 10,

NALU\_TYPE\_EOSTREAM = 11,

NALU\_TYPE\_FILL = 12,

#if (MVC\_EXTENSION\_ENABLE)

NALU\_TYPE\_PREFIX = 14,

NALU\_TYPE\_SUB\_SPS = 15,

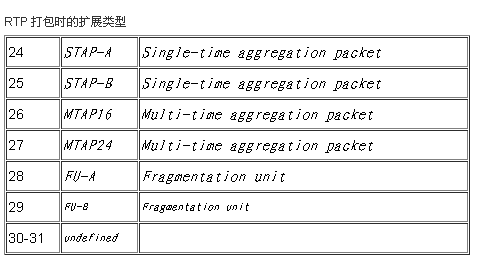
NALU\_TYPE\_SLC\_EXT = 20,

NALU\_TYPE\_VDRD = 24 // View and Dependency Representation Delimiter NAL Unit

#endif

} NaluType;

未指定/不保留的NAL类型，在RTP中会被使用，RTP扩展了NAL类型如下：



不同类型的NALU的重要性指示如下表所示：

( 所谓参考帧，就是在其他帧解码时需要参照的帧。比如一个I帧可能被一个或多个B帧参考，一个B帧可能被某个P帧参考。)



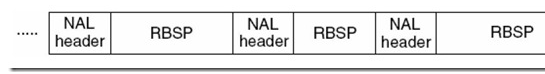
NRI 值为 7 和 8 的NALU 分别为序列参数集（sps）和图像参数集（pps）。参数集是一组很少改变的，为大量VCL NALU 提供解码信息的数据。

其中序列参数集作用于一系列连续的编码图像，

而图像参数集作用于编码视频序列中一个或多个独立的图像。

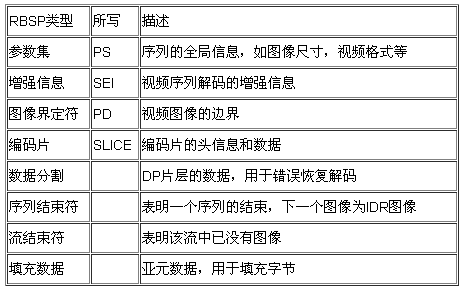
如果没能正确接收到这两个参 数集，那么其他NALU 也是无法解码的。因此它们一般在发送其它 NALU 之前发送(SpyDroid中使用RTSP先发送)，并且使用不同的信道或者更加可靠的传输协议（如TCP）进行传输，也可以重复传输。

H264在网络传输的是NALU，NALU的结构是：NAL头+RBSP （?? 为什么不是NAL Header + EBSP ??）



NALU头用来标识后面的RBSP是什么类型的数据，他是否会被其他帧参考以及网络传输是否有错误

RBSP数据是下表中的一种：

参数集分PPS SPS 数据分割分DP A B C ( data partition 数据划分片段)

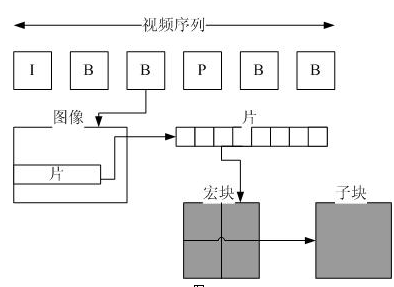
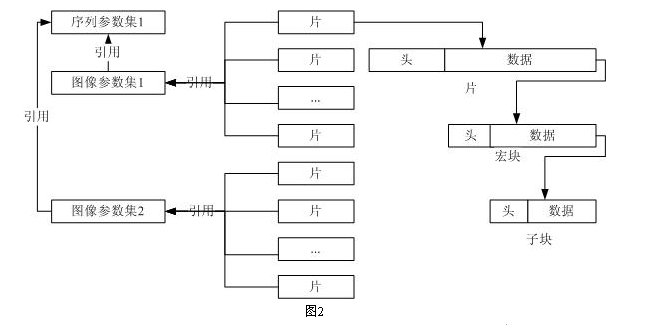
在实际的H264数据帧中，往往帧前面带有00 00 00 01 或 00 00 01分隔符，一般来说编码器编出的首帧数据为PPS与SPS，接着为I帧…



pts/dts怎样实现音视频同步

H264编解码/VCL

H264/AVC 的分层结构

分层结构由五层组成，分别是序列参数集、图像参数集、片（Slice）、和宏块和子块。

参数集是一个独立的数据单位，不依赖于参数集外的其它

一个参数集不对应某一个特定的图像或序列，同一序列参数集可以被多个图像参数集引用，同理，同一个图像参数集也可以被多个图像引用。只在编码器认为需要更新参数集的内容时，才会发出新的参数集。

在H.264中，图像以序列(GOP 帧组)为单位进行组织。一个序列的第一个图像叫做IDR图像，IDR图像都是I帧，H.264引入IDR图像为了解码的同步，当解码器解码到IDR图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果前一个序列出现重大错误，在这里可以获得重新同步的机会。IDR图像之后的图像永远不会使用IDR之前的图像的数据来解码。

IDR是I帧，但I帧不一定是IDR。I帧之后的图像有可能会使用I帧之前的图像做运动参考(??但不会跨过IDR帧，也就是一个GOP帧组，两个IDR帧之间，可以存在非IDR的I帧，帧组中P帧可以使用之前N个的I帧或P帧作为参考??)。

<http://blog.sina.com.cn/s/blog_8fb8cd4801018yyo.html>

1. 参照一段时间内图像的统计结果表明，在相邻几幅图像画面中，一般有差别的像素只有10%以内的点,亮度差值变化不超过2%，而色度差值的变化只有1%以内

2. 当某个图像与之前的图像变化很大，无法参考前面的帧来生成，那我们就结束上一个序列，开始下一段序列，也就是对这个图像生成一个完整帧

H264协议 三种帧：

a. 完整编码的帧叫I帧

b. 参考之前的I帧生成的只包含差异部分编码的帧叫P帧

c. 参考前后的帧编码的帧叫B帧

 H264采用的核心算法是帧内压缩和帧间压缩，帧内压缩是生成I帧的算法，帧间压缩是生成B帧和P帧的算法

a. I、B、P各帧是根据压缩算法的需要,是人为定义的,它们都是实实在在的物理帧。

b. 一般来说，I帧的压缩率是7（跟JPG差不多），P帧是20，B帧可以达到50。可见使用B帧能节省大量空间，节省出来的空间可以用来保存多一些I帧，这样在相同码率下，可以提供更好的画质。

三种帧说明：

I帧:帧内编码帧 ，I帧表示关键帧 这一帧画面的完整保留 解码时只需要本帧

I帧特点:

1.它是一个全帧压缩编码帧。它将全帧图像信息进行JPEG压缩编码及传输;

2.解码时仅用I帧的数据就可重构完整图像;

3.I帧描述了图像背景和运动主体的详情;

4.I帧不需要参考其他画面而生成;

5.I帧是P帧和B帧的参考帧(其质量直接影响到同组中以后各帧的质量);

6.I帧是帧组GOP的基础帧(第一帧),在一组中只有一个I帧;

7.I帧不需要考虑运动矢量;

8.I帧所占数据的信息量比较大。

P帧:前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧（或P帧）的差别，解码时需要用之前缓存的画面叠加上本帧定义的差别，生成最终画面。（也就是差别帧，P帧没有完整画面数据，只有与前一帧的画面差别的数据）

P帧的预测与重构:P帧是以I帧为参考帧,在I帧中找出P帧“某点”的预测值和运动矢量,取预测差值和运动矢量一起传送。在接收端根据运动矢量从I帧中找出P帧“某点”的预测值并与差值相加以得到P帧“某点”样值,从而可得到完整的P帧。

P帧特点:

1.P帧是I帧后面相隔1~2帧的编码帧;

2.P帧采用运动补偿的方法传送它与前面的I或P帧的差值及运动矢量(预测误差);

3.解码时必须将I帧中的预测值与预测误差求和后才能重构完整的P帧图像;

4.P帧属于前向预测的帧间编码。它只参考前面最靠近它的I帧或P帧;

5.P帧可以是其后面P帧的参考帧,也可以是其前后的B帧的参考帧;

6.由于P帧是参考帧,它可能造成解码错误的扩散;

7.由于是差值传送,P帧的压缩比较高。

B帧:双向预测内插编码帧。B帧是双向差别帧，也就是B帧记录的是本帧与前后帧的差别（具体比较复杂，有4种情况，但我这样说简单些），换言之，要解码B帧，不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高，但是解码时CPU会比较累。

B帧的预测与重构：B帧以前面的I或P帧和后面的P帧为参考帧,“找出”B帧“某点”的预测值和两个运动矢量,并取预测差值和运动矢量传送。接收端根据运动矢量在两个参考帧中“找出(算出)”预测值并与差值求和,得到B帧“某点”样值,从而可得到完整的B帧。

B帧特点

1.B帧是由前面的I或P帧和后面的P帧来进行预测的;

2.B帧传送的是它与前面的I或P帧和后面的P帧之间的预测误差及运动矢量;

3.B帧是双向预测编码帧;

4.B帧压缩比最高,因为它只反映丙参考帧间运动主体的变化情况,预测比较准确;

5.B帧不是参考帧,不会造成解码错误的扩散。

--------------------------------

压缩算法的说明

--------------------------------

h264的压缩方法:

1.分组:把几帧图像分为一组(GOP，也就是一个序列，帧组),为防止运动变化,帧数不宜取多。

2.定义帧:将每组内各帧图像定义为三种类型,即I帧、B帧和P帧;

3.预测帧:以I帧做为基础帧,以I帧预测P帧,再由I帧和P帧预测B帧;

4.数据传输:最后将I帧数据与预测的差值信息进行存储和传输。

帧内（Intraframe）压缩也称为空间压缩（Spatial compression）。当压缩一帧图像时，仅考虑本帧的数据而不考虑相邻帧之间的冗余信息，这实际上与静态图像压缩类似。帧内一般采用有损压缩算法，由于帧内压缩是编码一个完整的图像，所以可以独立的解码、显示。帧内压缩一般达不到很高的压缩，跟编码jpeg差不多。

帧间（Interframe）压缩的原理是：相邻几帧的数据有很大的相关性，或者说前后两帧信息变化很小的特点。也即连续的视频其相邻帧之间具有冗余信息,根据这一特性，压缩相邻帧之间的冗余量就可以进一步提高压缩量，减小压缩比。帧间压缩也称为时间压缩（Temporal compression），它通过比较时间轴上不同帧之间的数据进行压缩。帧间压缩一般是无损的。帧差值（Frame differencing）算法是一种典型的时间压缩法，它通过比较本帧与相邻帧之间的差异，仅记录本帧与其相邻帧的差值，这样可以大大减少数据量。

顺便说下有损（Lossy ）压缩和无损（Lossy less）压缩。无损压缩也即压缩前和解压缩后的数据完全一致。多数的无损压缩都采用RLE行程编码算法。有损压缩意味着解压缩后的数据与压缩前的数据不一致。在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或音频信息,而且丢失的信息不可恢复。几乎所有高压缩的算法都采用有损压缩,这样才能达到低数据率的目标。丢失的数据率与压缩比有关,压缩比越小，丢失的数据越多,解压缩后的效果一般越差。此外,某些有损压缩算法采用多次重复压缩的方式,这样还会引起额外的数据丢失。

序列/IDR/GOP：

1. 在H264中图像以序列为单位进行组织，一个序列是一段图像编码后的数据流，以I帧开始，到下一个I帧结束。

2. (??GOP 以IDR帧开始 中间可能有I帧，但是只有IDR帧才会丢掉之前的参考帧队列，IDR帧后面的P/B帧不会参考IDR前面的帧来解码，两个IDR帧之间的P/B帧会参照之前的N个I/P/B帧来解码??)

3. 一个序列就是一段内容差异不太大的图像编码后生成的一串数据流。当运动变化比较少时，一个序列可以很长，因为运动变化少就代表图像画面的内容变动很小，所以就可以编一个I帧，然后一直P帧、B帧了。当运动变化多时，可能一个序列就比较短了，比如就包含一个I帧和3、4个P帧。

4. 一个序列的第一个图像叫做 IDR 图像（立即刷新图像），IDR 图像都是 I 帧图像。

5. H.264 引入 IDR 图像是为了解码的重同步，当解码器解码到 IDR 图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果前一个序列出现重大错误，在这里可以获得重新同步的机会。IDR图像之后的图像永远不会使用IDR之前的图像的数据来解码。

（1）解析函数（获取信息）：

ff\_h264\_decode\_nal()：解析NALU Header。  
ff\_h264\_decode\_seq\_parameter\_set()：解析SPS。  
ff\_h264\_decode\_picture\_parameter\_set()：解析PPS。  
ff\_h264\_decode\_sei()：解析SEI。  
ff\_h264\_decode\_slice\_header()：解析Slice Header。

[I 帧和 IDR 帧的区别](http://blog.csdn.net/chenchong_219/article/details/44870903)

<http://blog.csdn.net/chenchong_219/article/details/44870903>

有如下帧序列： IPPPP **I P**PPP ……（我们程序没有 B 帧，所以帧序列简单些，但道理是一样的）。按照 3 个参考帧编码。

因为“按照 3 个参考帧编码”，所以参考帧队列长度为 3 。

遇到绿色的 I 时，并不清空参考帧队列，把这个 I 帧加入参考帧队列（当然 I 编码时不用参考帧。）。再检测到红色的 P 帧时，用到的就是 PPI 三帧做参考了。

不怕自己罗嗦（好记性不如烂笔头），再强调一个： 一个参考帧，就是参考当前帧的前面的那帧（因为没涉及到 B 帧，所以“前面的那帧”既是播放顺序的，也是编码顺序的）。多个参考帧是一个道理 。 （ 我以前一直误解为从前面的几帧中找到最合适的一个参考帧）

最后，“ 但是收到 IDR 帧时，解码器另外需要做的工作就是：把所有的 PPS 和 SPS 参数进行更新。由此可见，在编码器端，每发一个 IDR ，就相应地发一个 PPS&SPS\_nal\_unit ”应该是对的吧。先这样认为：

IDR frame：I和IDR帧都使用帧内预测，在编码解码中为了方便，首个I帧要和其他I帧区别开，把第一个I帧叫IDR，这样方便控制编码和解码流程，所以IDR帧一定是I帧，但I帧不一定是IDR帧；IDR帧的作用是立刻刷新,使错误不致传播,从IDR帧开始算新的序列开始编码。I帧有被跨帧参考的可能,IDR不会。

语法元素

H.264编码器输出的Bit流中，每个Bit都隶属于某个语法元素

代码分析中用”语法”代表某个说明，元素名称由小写字母和一系列下划线组成(比如下面的nal\_unit\_type )，变量名称是大小写字母组成，中间没有下划线(比如下面的NumBytesInRBSP)



还有资料：

从Slice\_Header学习H.264（一）--片头语法元素介绍 ----- ??讲述一些学习H264的方法，而且会对句法比如slice\_header有深入解析??

从Slice\_Header学习H.264（三.1）--相关细节之 POC的计算

从Slice\_Header学习H.264（三.2）--相关细节之 参考图像列表

分层结构由五层组成，分别是序列参数集、图像参数集、片（Slice）、和宏块和子块，下面对个层进行分析：

每个NAL前有一个起始码 0x000001，解码器检测每个起始码，作为一个NAL的起始标识，当检测到下一个起始码时，当前NAL结束。

(H.264引入防止竞争机制，解码器检测到0x000003时，把03抛弃，恢复原始数据)

解码器在解码时，首先逐个字节读取NAL的数据，统计NAL的长度，然后再开始解码。

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| nal\_nuit(NumBytesInNALunit){/\* NumBytesInNALunit为统计出来的数据长度\*/ |  |  |
| forbidden\_zero\_bit /\* 等于0 \*/语法元素 | All | f(1)  读进连续的1个bit |
| nal\_ref\_idc/\* 当前NAL的优先级，取值范围0-3 \*/语法元素 | All | u(2)  读进连续2个bits 并作为无符号整数 |
| nal\_unit\_type /\* NAL类型，见表2描述 \*/ 语法元素 | All | u(5) |
| NumBytesInRBSP=0 变量名称 |  |  |
| for(i=1;i<NumBytesInNALunit;i++){ |  |  |
| if(i+2<NumBytesInNALunit && next\_bits(24)==0x000003{ |  |  |
| /\* 0x000003伪起始码，需要删除0x03这个字节 \*/ |  |  |
| rbsp\_byte[NumBytesInRBSP++] 语法元素 | All | b(8)  读进连续的8个bit |
| rbsp\_byte[NumBytesInRBSP++] | All | b(8) |
| i+=2/\* 取出前两个0x00后，跳过0x03 \*/ |  |  |
| emulation\_prevention\_three\_byte/\* equal to 0x03 \*/语法元素 | All | f(8) |
| }else{ |  |  |
| rbsp\_byte[NumBytesInRBSP++] /\* 继续读取后面的字节 \*/ | All | b(8) |
| } |  |  |
| } |  |  |

SPS序列参数集 分析，(NAL单元类型是SPS 非VCL的NAL单元)

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| seq\_parameter\_set\_rbsp(){ |  |  |
| profile\_idc/\* 语法元素 指明所用的Profile \*/  66 Baseline  77 Main  88 Extended  100 High (FRExt)  110 High 10 (FRExt)  122 High 4:2:2 (FRExt)  144 High 4:4:4 (FRExt)  100是High (FRExt) | 0 | u(8) |
| constraint\_set0\_flag | 0 | u(1) |
| constraint\_set1\_flag | 0 | u(1) |
| constraint\_set1\_flag | 0 | u(1) |
| reserved\_zero\_5bits /\* equal to 0 \*/ | 0 | u(5) |
| level\_idc /\* 语法元素 指明所用的Level \*/ | 0 | u(8) |
| seq\_parameter\_set\_id /\* 语法元素 指明本序列参数集的id号，0-31，被图像集引用，编码需要产生新的序列集时，使用新的id，而不是改变原来参数集的内容 ???每个序列参数集都有自己的唯一ID??? \*/ | 0 | ue(v) |
| log2\_max\_frame\_num\_minus4/\* 为读取元素frame\_num服务，frame\_num标识图像的解码顺序，frame\_num的解码函数是ue(v)，其中v=log2\_max\_frame\_num\_minus4+4，该元素同时指明frame\_num的最大值MaxFrameNum=2( log2\_max\_frame\_num\_minus4+4**)**\*/ ???可以从sps中知道帧数??? | 0 | ue(v) |
| pic\_order\_cnt\_type /\* 指明poc的编码方法，poc标识图像的播放顺序，poc可以由frame\_num计算，也可以显示传送。poc共三种计算方式 \*/ | 0 | ue(v) |
| if(pic\_order\_cnt\_type==0) |  |  |
| log2\_max\_pic\_order\_cnt\_lsb\_minus4 /\* 指明变量MaxPicOrderCntLsb的值， MaxPicOrderCntLsb＝2(log2\_max\_pic\_order\_cnt\_lsb\_minus4+4) \*/ | 0 | ue(v) |
| else if(pic\_order\_cnt\_type==1){ |  |  |
| delta\_pic\_order\_always\_zero\_flag /\* 等于1时，元素delta\_pic\_order\_cnt[0]和delta\_pic\_order\_cnt[1]不在片头中出现，并且它们的默认值是0，等于0时，上述两元素出现的片头中 \*/ | 0 | u(1) |
| offset\_for\_non\_ref\_pic /\* 用来计算非参考帧或场的poc，[-231,231-1] \*/ | 0 | se(v) |
| offset\_for\_top\_to\_bottom\_field/\* 计算帧的底场的poc \*/ | 0 | se(v) |
| num\_ref\_frames\_inpic\_order\_cnt\_cycle /\* 用来解码poc,[0.255] \*/ | 0 | ue(v) |
| for(i=0;i<num\_ref\_frames\_inpic\_order\_cnt\_cycle;i++) |  |  |
| offset\_for\_ref\_frame[i]/\* 用来解码poc，对于循环中的每个元素指定一个偏移 \*/ | 0 | se(v) 有符号指数 哥伦布编码 |
| } |  |  |
| num\_ref\_frames /\* 参考帧队列可达到的最大长度，[0,16] \*/ | 0 | ue(v) |
| gaps\_in\_frame\_num\_value\_allowed\_flag /\* 为1，允许slice header中的frame\_num不连续 \*/ | 0 | u(1) |
| pic\_width\_inmbs\_minus1 /\* 本元素加1，指明以宏块为单位的图像宽度PicWidthInMbs=pic\_width\_in\_mbs\_minus1+1 \*/ | 0 | ue(v) |
| pic\_height\_in\_map\_units\_minus1 /\* 本元素加1，指明以宏块为单位的图像高宽度 PicHeightInMapUnitsMbs=pic\_height\_in\_map\_units\_minus1+1 \*/ | 0 | ue(v)  无符号指数 哥伦布编码 |
| frame\_mbs\_only\_flag /\* 等于0表示本序列中所有图像均为帧编码；等于1，表示可能是帧，也可能场或帧场自适应，具体编码方式由其它元素决定。结合前一元素：FrameHeightInMbs=(2-frame\_mbs\_only\_flag)\*PicHeightInMapUnits \*/ | 0 | ue(v) |
| if(frame\_mbs\_only\_flag) |  |  |
| mb\_adaptiv\_frame\_field\_flag /\* 指明本序列是否是帧场自适应模式：  frame\_mbs\_only\_flag=1，全部是帧  frame\_mbs\_only\_flag=0， mb\_adaptiv\_frame\_field\_flag=0，帧场共存  frame\_mbs\_only\_flag=0， mb\_adaptiv\_frame\_field\_flag=1，帧场自适应和场共存\*/ | 0 | u(1) |
| direct\_8x8\_inference\_flag /\* 用于指明B片的直接和skip模式下的运动矢量的计算方式 \*/ | 0 | u(1) |
| frame\_cropping\_flag /\* 解码器是否要将图像裁剪后输出，如果是，后面为裁剪的左右上下的宽度 \*/ | 0 | u(1) |
| if(frame\_cropping\_flag){ |  |  |
| frame\_crop\_left\_offset | 0 | ue(1) |
| frame\_crop\_right\_offset | 0 | ue(1) |
| frame\_crop\_top\_offset | 0 | ue(1) |
| frame\_crop\_bottom\_offset | 0 | ue(1) |
| } |  |  |
| vui\_parameters\_present\_flag /\* 指明vui子结构是否出现在码流中，vui子结构在附录中指明，用于表征视频格式的信息 \*/ | 0 | u(1) |
| if(vui\_parameters\_present\_flag) |  |  |
| vui\_parameters() | 0 |  |
| rbsp\_trailing\_bits() | 0 |  |
| } |  |  |

片层句法

片层句法（不分区） nal\_type = 1 非IDR图像的编码条带 slice ???nal\_type = 5 IDR图像的编码条带

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| slice\_layer\_without\_partitioning\_rbsp(){ |  |  |
| slice\_header() | 2 |  |
| slice\_data() /\* all categories of slice\_data() syntax \*/ | 2|3|4 |  |
| rbsp\_slice\_trailing\_bits() | 2 |  |
| } |  |  |

片层A分区句法 nal\_type = 2 非IDR图像编码条带的A类型数据划分(data partition)

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| slice\_data\_partition\_a\_layer\_rbsp(){ |  |  |
| slice\_header() | 2 |  |
| slice\_id | 2 | ue(v) |
| slice\_data() /\* only category 2 parts of slice\_data() syntax \*/ | 2 |  |
| rbsp\_slice\_trailing\_bits() | 2 |  |
| } |  |  |

片层B分区句法 nal\_type = 3 非IDR图像编码条带的B类型数据划分(data partition)

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| slice\_data\_partition\_b\_layer\_rbsp(){ |  |  |
| slice\_id | 3 | ue(v) |
| if(redundant\_pic\_cnt\_present\_flag) |  |  |
| redundant\_pic\_cnt | 3 |  |
| slice\_data() /\* only category 3 parts of slice\_data() syntax \*/ | 3 | ue(v) |
| rbsp\_slice\_trailing\_bits() | 3 |  |
| } |  |  |

片层C分区句法

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| slice\_data\_partition\_c\_layer\_rbsp(){ |  |  |
| slice\_id | 4 | ue(v) |
| if(redundant\_pic\_cnt\_present\_flag) |  |  |
| redundant\_pic\_cnt | 4 |  |
| slice\_data() /\* only category 4 parts of slice\_data() syntax \*/ | 4 | ue(v) |
| rbsp\_slice\_trailing\_bits() | 4 |  |
| } |  |  |

 拖尾（trailing bits）句法

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| rbsp\_trailing\_bits(){ |  |  |
| rbsp\_stop\_one\_bit /\* equal to 1 \*/ | All | f(1) |
| while(!byte\_aligned()) |  |  |
| rbsp\_alignment\_zero\_bit /\* equal to 0 \*/ | All | f(1) |
| } |  |  |

 片头（Slice header）句法

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| slice\_header(){ // ??? 只有片层和片层A分区才会有slice\_header ???? |  |  |
| first\_mb\_in\_slice /\* 片中的第一个宏块地址，片通过这个句法元素来标定它自己的地址。在帧场自适应模式下，宏块都是成对出现，这时本句法元素表示的是第几个宏块对，对应的第一个宏块的真实地址应该是：2\*first\_mb\_in\_slice \*/ | 2 | ue(v) |
| slice\_type /\* 片的类型 可以用来区分I P B帧，见表12 \*/ | 2 | ue(v) |
| pic\_parameter\_set\_id /\* 引用的图像集索引 ????用哪个PPS作为解码这一片???? \*/ | 2 | ue(v) |
| frame\_num /\* 每个参考帧都有一个连续的frame\_num作为它们的标识，它指明了各图像的解码顺序。非参考帧也有，但没有意义。 \*/ | 2 | u(v) |
| if(!frame\_mbs\_only\_flag){ |  |  |
| field\_pic\_flag  /\* 片层中标识图像编码模式的唯一一个元素，详细描述，见图3 \*/ | 2 | u(1) |
| if(filed\_pic\_flag) |  |  |
| bottom\_field\_flag /\* 1:底场，0:顶场 \*/ | 2 | u(1) |
| } |  |  |
| if(nal\_unit\_type==5) |  |  |
| idr\_pic\_id /\* IDR图像标识，不同的IDR图像有不同的IDR值。场模式下，IDR帧的两个场有相同的idr\_pic\_id值，[0..65535] \*/ | 2 | ue(v) |
| if(pic\_order\_cnt\_type==0){ |  |  |
| pic\_order\_cnt\_lsb/\* 在poc的第一种算法中，显示传递poc的值，u(v)中，v=log2\_max\_pic\_order\_cnt\_lsb\_minus4+4 \*/ | 2 | u(v) |
| if(pic\_order\_present\_flag && !field\_pic\_flag) |  |  |
| delta\_pic\_order\_cnt\_bottom /\* 如果是在场模式下，场对中的两个场都各自被构造为一个图像，它们有各自的poc的计算方法来分别计算两场的poc，也就是一个场对拥有一对poc值；而在帧模式或帧场自适应模式下，一个图像只能根据片头中的元素计算出一个poc。在frame\_mbs\_only\_flag不为1时，每个帧或场自适应的图像在解码时，帧或帧场自适应中包含的两个场也必须有各自的poc值，通过本元素，可以在已经解码的帧或帧场自适应图像的poc基础上新映射一个poc，并把它赋给底场。\*/ | 2 | se(v) |
| } |  |  |
| if(pic\_order\_cnt\_type==1 && !delta\_pic\_order\_always\_zero\_flag){ |  |  |
| delta\_pci\_order\_cnt[0] /\* poc的第二和第三种算法是从frame\_num中映射得来，本元素用于帧编码下的底场和场编码方式的场 \*/ | 2 | se(v) |
| if(pic\_order\_present\_flag && !field\_pic\_flag) |  |  |
| delta\_pic\_order\_cnt[1] /\* 用于帧编码下的顶场\*/ | 2 | se(v) |
| } |  |  |
| if(redundant\_ic\_cnt\_present\_flag) |  |  |
| redundant\_pic\_cnt /\* 冗余片的id号 \*/ | 2 | ue(v) |
| if(slice\_type==B) |  |  |
| direct\_spatial\_mv\_pred\_flag /\* B图像在直接预测模式下，1:空间预测，0:时间预测 \*/ | 2 | u(1) |
| if(slice\_type==P||slice\_type==SP||slice\_type==B){ |  |  |
| num\_ref\_idx\_active\_override\_flag /\* 重载PPS中的参考帧队列中实际可用的参考帧的数目。 \*/ | 2 | u(1) |
| if(num\_ref\_idx\_active\_override\_flag){ |  |  |
| num\_ref\_idx\_10\_active\_minus1 /\* 重载值 \*/ | 2 | ue(v) |
| if(slice\_type==B) |  |  |
| num\_ref\_idx\_11\_active\_minus1 /\* 重载值 \*/ | 2 | ue(v) |
| } |  |  |
| } |  |  |
| ref\_pic\_list\_reordering() /\* 见第7节描述 \*/ | 2 |  |
| if((weighted\_pred\_flag && (slice\_type==P || slice\_type==SP)) || (weighted\_bipred\_idc==1 && slice\_type==B)) |  |  |
| pred\_weight\_table() /\* 见第8节描述 \*/ | 2 |  |
| if(nal\_ref\_idc!=0) |  |  |
| dec\_ref\_pic\_marking() /\* 见第9节描述 \*/ | 2 |  |
| if(entropy\_coding\_mode\_flag && slice\_type!=1 && slice\_type != SI) |  |  |
| cabac\_init\_idc  /\* 给出cabac初始化时表格的选择，[0..2] \*/ | 2 | ue(v) |
| slice\_qp\_delta /\* 指出用于当前片的所有宏块的量化参数的初始值。SliceQPY=26+pic\_init\_qp\_minus26+slice\_qp\_delta，[0..51] \*/ | 2 | se(v) |
| if(slice\_type==SP || slice\_type==SI){ |  |  |
| if(slice\_type==SP) |  |  |
| sp\_for\_switch\_flag /\* 指出SP帧中的p宏块的解码方式是否是switching模式 \*/ | 2 | u(1) |
| slice\_qs\_delta /\* 与slice\_qp\_delta的语义相似，用于SP和SI，QSY=26+pic\_init\_qs\_minus26+slice\_qs\_delta，[0..51] \*/ | 2 | se(v) |
| } |  |  |
| if(deblocking\_flter\_control\_present\_flag){ |  |  |
| disable\_deblocking\_filter\_idc /\* H.264指定了一套算法可以在解码器端独立地计算图像中各边界的滤波强度进行滤波。除了解码器独立计算之外，编码器也可以传递句法元素来干涉滤波强度，该元素指定了在块的是否使用滤波，同时批明那个块的边界不用滤波 \*/ | 2 | ue(v) |
| if(disable\_deblocking\_filter\_idc != 1){ |  |  |
| slice\_alpha\_c0\_offset\_div2 /\* 增强alpha时的偏移值，FilterOffsetA=slice\_alpha\_c0\_offset\_div2 << 1 \*/ | 2 | se(v) |
| slice\_beta\_offset\_div2 /\* 增强beta的偏移值，FilterOffsetB=slice\_beta\_offset\_div2<<1 \*/ | 2 | se(v) |
| } |  |  |
| } |  |  |
| if(num\_slice\_group\_minus1 > 0 && slice\_group\_map\_type>=3 && slice\_group\_map\_type <= 5) |  |  |
| slice\_group\_change\_cycle /\* 片组类型是3.4.5时，由该元素可以获取片组中映射单元的数目 \*/ | 2 | u(v) |
| } |  |  |

表11

|  |  |
| --- | --- |
| **slice\_type** | **name** |
| 0 | P slice |
| 1 | B slice |
| 2 | I slice |
| 3 | SP slice |
| 4 | SI slice |
| 5 | P slice |
| 6 | B slice |
| 7 | I slice |
| 8 | SP slice |
| 9 | SI slice |

表12（注：IDR图像时，slice\_type为2,4,7,9） （??实际只有 P B I slice和 SP SI slice??）

参考帧重排序

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| ref\_pic\_list\_reordering(){ /\* 每个参考图像都有frame\_num，但编码器要指定当前图像的参考图像时，使用是ref\_id，frame\_num->PicNum->ref\_id \*/ |  |  |
| if(slice\_type!=I && slice\_type!=SI){ |  |  |
| ref\_pic\_list\_reordering\_flag\_l0 /\* 指明List0是否进行重排序 \*/ | 2 | u(1) |
| if(ref\_pic\_list\_reordering\_flag\_l0) |  |  |
| do{ |  |  |
| reordering\_ofpic\_nums\_idc /\* 执行哪种重排序操作，见表14描述 \*/ | 2 | ue(v) |
| if(reordering\_ofpic\_nums\_idc==0 || reordering\_ofpic\_nums\_idc==1) |  |  |
| abs\_diff\_pic\_num\_minus1 /\* 对短期参考帧重排序时指明重排序图像与当前的差，见表14 \*/ | 2 | ue(v) |
| elseif(reordering\_ofpic\_nums\_idc==2) |  |  |
| long\_term\_pic\_num /\* 对长期参考帧得排序时指明重排序图像 \*/ | 2 | ue(v) |
| }while(reordering\_of\_pic\_nums\_idc!=3) |  |  |
| } |  |  |
| if(slice\_type==B){ |  |  |
| ref\_pic\_list\_reordering\_flag\_l1 /\* 指明List1是否进行重排序 \*/ | 2 | u(1) |
| if(ref\_pic\_list\_reordering\_flag\_l1) |  |  |
| do{ |  |  |
| reordering\_ofpic\_nums\_idc /\* 执行哪种重排序操作，见表14描述 \*/ | 2 | ue(v) |
| if(reordering\_ofpic\_nums\_idc==0 || reordering\_ofpic\_nums\_idc==1) |  |  |
| abs\_diff\_pic\_num\_minus1 /\* 对短期参考帧重排序时指明重排序图像与当前的差，见表14 \*/ | 2 | ue(v) |
| elseif(reordering\_ofpic\_nums\_idc==2) |  |  |
| long\_term\_pic\_num /\* 对长期参考帧得排序时指明重排序图像 \*/ | 2 | ue(v) |
| }while(reordering\_of\_pic\_nums\_idc!=3) |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| **reordering\_of\_pic\_nums\_idc** | **操作** |
| 0 | 短期参考帧重排序，abs\_diff\_pic\_num\_minus1会出现在码流中，从当前图像的PicNum减去(abs\_diff\_pic\_num\_minus1+1)后指明需要重排序的图像 |
| 1 | 短期参考帧重排序，abs\_diff\_pic\_num\_minus1会出现在码流中，从当前图像的PicNum加上(abs\_diff\_pic\_num\_minus1+1)后指明需要重排序的图像 |
| 2 | 长期参考帧重排序，long\_term\_pic\_num会出现在码流中，指明需要重排序的图像。 |
| 3 | 结束循环，退出重排序操作。 |

加权预测的语义

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| pred\_weight\_table(){ |  |  |
| luma\_log2\_weight\_denom /\* 给出参考帧列表中参考图像所有亮度的加权系数，[0..7] \*/ | 2 | ue(v) |
| chroma\_log2\_weight\_denom /\* 给出参考帧列表中参考图像所有色度的加权系数，[0..7] \*/ | 2 | ue(v) |
| for(i=0;i<=num\_ref\_idx\_l0\_active\_minus1;i++){ |  |  |
| luma\_weight\_10\_flag /\* 1:在参考帧序列0中的亮度的加权系数存在 \*/ | 2 | u(1) |
| if(luma\_weight\_l0\_flag){ |  |  |
| luma\_weight\_l0[i] /\* 用参考序列0预测亮度值时，所用的加权系数。如果luma\_weight\_l0\_flag=0，luma\_weight\_l0[i]=2luma\_log2\_weight\_denom \*/ | 2 | se(v) |
| luma\_offset\_l0[i] /\* 用参考序列0预测亮度值时，所用的加权系数的偏移，[-128-..127]，如果luma\_weight\_l0\_flag=0，该值0 \*/ | 2 | se(v) |
| } |  |  |
| chroma\_weight\_l0\_flag /\* 同Luma相似，但用于色度 \*/ | 2 | u(1) |
| if(chroma\_weight\_l0\_flag){ |  |  |
| for(j=0;j<2;j++) |  |  |
| chroma\_weight\_l0[i][j] | 2 | se(v) |
| chroma\_offset\_l0[i][j] | 2 | se(v) |
| } |  |  |
| } |  |  |
| if(slice\_type==B) |  |  |
| for(i=0;i<=num\_ref\_idx\_l1\_active\_minus1;i++){ |  |  |
| luma\_weight\_l1\_flag | 2 | u(1) |
| if(luma\_weight\_l1\_flag){ |  |  |
| luma\_weight\_l1[i] | 2 | se(v) |
| luma\_offset\_l1[i] | 2 | se(v) |
| } |  |  |
| chroma\_weight\_l1\_flag | 2 | u(1) |
| if(chroma\_weight\_l1\_flag) |  |  |
| for(j=0;j<2;j++){ |  |  |
| chroma\_weight\_l1[i][j] | 2 | se(v) |
| chroma\_offset\_l1[i][j] | 2 | se(v) |
| } |  |  |
| } |  |  |
| } |  |  |

参考图像序列标记

|  |  |  |
| --- | --- | --- |
| **句法** | **C** | **Desc** |
| dec\_ref\_pic\_marking(){ /\* 前文介绍的重排序操作是对参考帧队列重新排序，而标记操作负责将参考图像移入或移出参考帧队列 \*/ |  |  |
| if( nal\_unit\_type ==5){ |  |  |
| no\_output\_of\_prior\_pics\_flag  /\* IDR时，1:将前面已经解码的图像全部输出 \*/ | 2|5 | u(1) |
| long\_term\_reference\_flag  /\* IDR时，1:使用长期参考，并且每个IDR图像解码后自动成为长期参考帧，0:IDR图像解码后自动成为短期参考帧 \*/ | 2|5 | u(1) |
| }else{ |  |  |
| adaptive\_ref\_pic\_marking\_mode\_flag /\* 0:FIFO，使用滑动窗的机制，先入先出，在这种模式下，无法对长期参考帧进行操作，1:自适应标记，后续码流中会有一系列句法元素显式指明操作的步骤。 \*/ |  | u(1) |
| if(adaptive\_ref\_pic\_marking\_mode\_flag){ |  |  |
| do{ |  |  |
| memory\_management\_control\_operation /\*自适应模式下，指明本次操作的具体内容，详细描述见表17 \*/ | 2|5 | ue(v) |
| if(memory\_management\_control\_operation == 1 ||memory\_management\_control\_operation==3) |  |  |
| difference\_of\_pic\_nums\_minus1/\* 通过该元素可以计算出需要操作的图像在短期参考队列中的序号。 \*/ | 2|5 | ue(v) |
| if(memory\_management\_control\_operation == 2) |  |  |
| long\_term\_pic\_num /\* 得到所要操作的长期参考图像的序号 \*/ | 2|5 | ue(v) |
| if(memory\_management\_control\_operation == 3 ||memory\_management\_control\_operation == 6) |  |  |
| long\_term\_frame\_idx /\* 分配一个长期参考帧的序号给一个图像 \*/ | 2|5 | ue(v) |
| if(memory\_management\_control\_operation == 4) |  |  |
| max\_long\_tewrm\_frame\_idx\_plus1/\* 此元素减1，指明长期参考队列的最大数目，[0..num\_ref\_frames] \*/ | 2|5 | ue(v) |
| }while(memory\_management\_control\_operation != 0) |  |  |
| } |  |  |
| } |  |  |

|  |  |
| --- | --- |
| memory\_management\_control\_operation | **操作** |
| 0 | 结束循环 |
| 1 | 将一个短期参考图像标记为非参考图像，也即将一个短期参考图像移出参考帧队列 |
| 2 | 将一个长期参考图像标记为非参考图像，也即将一个长期参考图像移出参考帧队列 |
| 3 | 将一个短期参考图像转为长期参考图像 |
| 4 | 指明长期参考帧的最大数目 |
| 5 | 清空参考帧队列，将所有参考图像移参考帧队列，并禁用长期参考机制 |
| 6 | 将当前图像存存为一个长期参考帧 |