YUV

# 1.概述

与RGB类似，YUV也是图像的存储方式。

YUV一般还有几种表述方式：YCbCr，YPbPr；

亮度信号被称作Y；色度信号由两个相互独立的信号组成，一般称作UV，也有称作CbCr或者PbPr；

这些不同的称谓，是由不同的编码方式产生的，但他们的概念基本相同。

YUV格式有两大类：packed和planar。

Planar的YUV格式，是按行存储的，先连续存储Y，紧接着存储所有U，最后存储所有V；packed的YUV格式，Y、U、V是连续存储的。

YUV码流的存储方式，主流的有4中：YUV444, 422, 420, 411。概述如下：

* YUV444采样，每一个Y共用一组UV分量；
* YUV422采样，每两个Y共用一组UV分量；
* YUV420采样，每四个Y共用一组UV分量；

按照我现在的理解，一般说的YUV444等代表的是packed格式，YUV444P代表的是planar格式。Yuv422和420同理。

Yuv420除了packed和planar，还有semiPlanar，一般简写为yuv420sp，详见第二章。

# 2.各个不同格式

## 2.1.YUV444

YUV444，一个Y对应一组UV分量，其格式可概述为：

* Packed格式(YUV444)，为YUVYUVYUV........；
* Planar模式(YUV444P)，为YYY....UUU....VVV...；

Y、U、V的数量是相同的，如果分辨率为w\*h，那么yuv文件的size就是：w\*h\*3;

## 2.2.YUV422

YUV422有几种不同的格式划分方式：YUYV，UYVY，YUV422P等；

* YUYV是yuv422--packed的一种，其格式为：YUYVYUYVYUYV.......；
* UYVY是yuv422--packed的一种，其格式为：UYVYUYVYUYVY.......；
* Yuv422p是planar模式，其格式为：YYYY....UU....VV....；

Y、U、V数量不同，两个Y对应一组UV，yuv文件的size就是：w\*h\*2；

## 2.3.YUV420

YUV420分为YUV420P和YUV420SP两种：

* YUV420p，其格式为:YYYY....UU..VV..;
* YUV420sp，其格式为:YYYY....UVUV.....;

Yuv文件的size，是：w\*h\*3/2；

YUV420是使用最为广泛的一种yuv格式，其精细划分也最多，概述如下：

* I420：即标准意义上的YUV420P，存储方式为：YYYY....UU..VV..；
* YU12：与I420一致，是安卓的存储方式；
* YV12：属于YUV420P的一种，存储格式为：YYYY....VV..UU..;
* NV12：属于yuv420sp的一种，存储格式为：yyyy....uvuv....，简单说，就是先存放Y，之后UV交替存储，U在前；
* NV21：属于yuv420sp的一种，存储格式为：yyyy....vuvu....，与NV12不同的仅仅是U和V的顺序不同，V在前；

## 2.4.YUV411

TODO

# 3.转换规则

## 3.1.packed和planar的转换

同种格式的packed和planar模式的转换更为简单一些，以yuv444为例说明：

yuv444packed模式下，yuvyuv.....的格式是其文件格式；yuv444planar模式下，yyyy....uuuu....vvvv....是其文件格式。因此所谓的转换，只是将各个位置的y、u、v分量，重新排列。

## 3.2.YUV444、YUV422、YUV420的转换

H264

# 1.VCL NAL

H264协议分为两层：VCL, NAL；

VCL，Video coding layer，视频编码层；NAL，Network abstraction layer，网络抽象层；

VCL数据表示编码处理的输出，表示被压缩后的视频数据序列。NAL单元用来格式化数据并提供头信息，保证数据适合各种信道和存储介质上的传输。

# 2.SODB RBSP EBSP

SODB：string of data bits，最原始的编码数据，无任何附加内容；

RBSP：在SODB基础上增加了rbsp\_stop\_ont\_bit(bit值为1)，并在其后用0按字节进行了补齐；

EBSP：Encapsulation byte sequence packets，在RBSP基础上，增加防止伪起始码字节0x03；

EBSP存在的原因，是对于H264来说，一般会应用0x00000001/0x000001作为一个有效的NALU的起始码，如果这时候存在原始的编码数据，其中有0x0000的数据部分，就会使用0x03作一次转换，防止与起始码冲突。例如：

0x000000 -> 0x00000300

0x000001 -> 0x00000301

等到解码的时候，会将0x03删除掉，称之为“脱壳”。

# 3.NALU

NAL单元，简称NALU，其构成有两部分：NAL header和RBSP等数据部分；

NAL头部占用1个字节，构成为：

|  |  |  |
| --- | --- | --- |
| Forbiden | 1bit | H264规定这一位必须是0，forbidden\_zero\_bit |
| NalPriority | 2bit | nal\_ref\_idc，0-3，表示这个nalu的优先级 |
| Type | 5bit | nalu的类型 |

对于NALU的type，H264规范定义如下：

|  |  |  |
| --- | --- | --- |
| TypeValue | 类型 | C |
| 0 | 未使用 |  |
| 1 | 不分区，非IDR图像的片 | 2，3，4 |
| 2 | 片分区A | 2 |
| 3 | 片分区B | 3 |
| 4 | 片分区C | 4 |
| **5** | **IDR图像中的片，I帧** | **2，3** |
| **6** | **补充增强信息单元，SEI** | **5** |
| **7** | **序列参数集，SPS** | **0** |
| **8** | **图像参数集，PPS** | **1** |
| 9 | 分界符 | 6 |
| 10 | 序列结束 | 7 |
| 11 | 码流结束 | 8 |
| 12 | 填充 | 9 |
| 13 - 23 | 保留 |  |
| 24 - 31 | 未使用 |  |

如上的NALU的type规范，是我们解析H264的重要依据；

# 4.起始码

H264规范定义了起始码(startcode)的概念，用来判断是否是一个新的NALU的开始。

如果NALU对应的slice是一帧的开始，则用0x00000001作为起始码；否则用0x000001作为起始码；

# 5.SPS PPS SEI

SPS和PPS是初始化解码器必须要有的数据，没有这两部分数据，视频数据无法解析并播放。

H264文件，有一些是每个IDR前面都存在相应的SPS和PPS，有一些是只有在文件开头才有，文件中就没有了。一般来说:

如果是直播的话，每个IDR前面都要加上sps和pps，因为有的观众会中途进来观看；

如果是本地稳定文件，开头有sps和pps就足够了，也可以每个IDR前面都加入sps和pps，要看具体需求了。

SEI作为辅助增强信息，supplemental enhancement infomation，并非总是存在。

# 6.IDR和I帧

IDR，instantaneous decoding refresh，即时解码刷新；

I帧和IDR帧都是帧内预测的。

IDR的作用是立刻刷新，从IDR开始重新计算一个新的序列开始编码，会导致DPB(decoding picture buffer，参考帧列表)被清空；

I帧不具备这个强制清空DPB的能力。

IDR一定是I帧，但I帧不一定是IDR帧。

IDR帧之后的所有帧，都不能够引用IDR帧之前的帧的内容；但普通I帧则不同，气候的B帧和P帧是可以参考普通I帧之前的I帧的。

# 编码级别

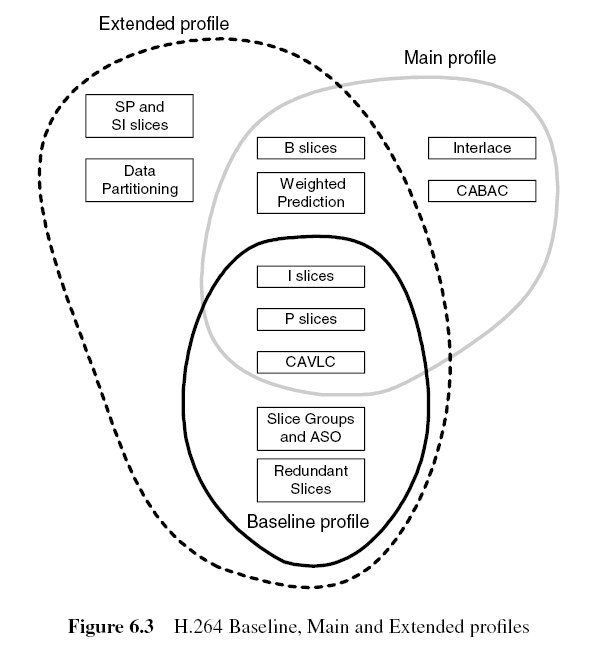
H264支持几种编码级别：baseline profile，main profile，extended profile；

baseline profile是基本档，利用I片和P片支持帧内核帧间编码，支持利用基于上下文的自适应的变长编码进行的熵编码--CAVLC。主要用于可视电话、会议电视、无线通信等实时视频通信；

main profile是主要档，支持隔行视频，采用B帧的帧间编码和加权预测的帧间编码；支持利用基于上下文的自适应的算术编码--CABAC。主要用于数字广播电视和数字视频存储；

extended profile是扩展档，支持吗刘志坚有效的切换(SP和SI片)、改变误码性能，但不支持隔行视频和CABAC。

详见下图。



MP4

MPEG4技术于1998年11月公布，1999年1月投入使用。该标准不仅是针对一定比特率下的音视频编码，更加注重多媒体系统的交互性和灵活性。MP4封装格式对应的国际标准是ISO/IEC 14496-12(信息技术 视听对象编码，第12部分，ISO 基本媒体文件格式)。

MP4视频文件封装格式是基于QuickTime容器格式的，因此参考QuickTime的编码格式对于MP4文件格式有帮助。MP4是一个开放的容器格式，几乎可以用来描述所有的媒体结构。MP4文件中的媒体描述(metaData)和媒体数据(mediaData)是分开描述的，而且媒体数据的组织自由度很高，并不要求一定要按照时间顺序排列，甚至媒体数据可以直接引用其他文件。MP4现在被广泛用于封装H264和AAC音频，是高清音视频的代表。

# 1.概述

MP4文件中的所有数据以box的形式组织起来，box就是QuickTime中的atom的定位。可以这么说，MP4文件由若干个box组成，每个box有类型和长度，可以将box理解为一个数据对象块。

box可以包含另外一个或多个box，这种box称为container box。

一个mp4文件一般的组成结构如下：

1. 首先是一个“ftyp”类型的box，这个box作为mp4格式的标志，并且包含了关于文件的一些基本信息；
2. 紧随其后的是一个“moov”类型(movie box)的box，这是一个container box，子box中包含了媒体的所有metadata信息；
3. 之后是所有的媒体数据，存储在“mdat”类型(media data box)的box中，这也是一个continer box，可以有多个，也可以没有(如果媒体数据全部引用自其他文件，就不存在这个box)；

## 1.1.术语

track：表示一些sample的集合，对于媒体数据来说，track就是一个视频或者音频序列。

hint track：一种特殊的track，并不包含媒体数据，只是包含了一些将其他track打包成流媒体的指示信息。

sample：如果是hint track，sample定义一个或多个流媒体包的格式。如果是正常track，video sample就是一帧视频或一组连续的视频帧；audio sample是一段连续的音频帧；

sample table：指明sample的时序和物理布局的表；

chunk：一个track的几个sample所组成的单元。

# 2.Box

要注意，Box的字节序是网络字节序，也就是大端字节序(Big-endian)，简单说就是一个32位的4字节整数，存储方式是高位字节在内存中的低端。

Box由header和body两部分组成。header统一指明box的大小和类型，body根据类型有不同的意义和格式。

标准的box header的前两部分如下：

1. 开始的4字节是boxSize，这个size包括了box header和box body的整个box的大小；这里有一些特殊情况：
   1. 如果size是1，说明这个Box的大小超出了4个字节能表示范围，是一个large size，需要到largesize域中才能找到；一般情况下只有mdat类型的box才会用到largesize，因为媒体数据本身的size有可能比较大；
   2. 如果size是0，表示这个box是文件的最后一个box，这个box的结尾就是文件结尾了。
2. 随后紧跟的是32位(4字节)的box type，其值代表了不同的含义：ftyp，moov，mdat是已经介绍过的，有一个特殊的，就是uuid，如果是该类型说明是用户自定义的类型。如果是其他未定义的类型，可以直接忽略这个box。

# 3.ftyp -- File type box

ftyp类型的box在一个mp4文件中，有且只有1个。而且这个box必须包含在文件层，不能被其他box包含。一般来说，这个box应该存在于mp4文件的最开始，指示这个mp4文件的相关信息。

ftyp的header由两部分组成：boxSize和boxType；

ftyp的body由以下几部分组成：

* 4字节的major brand；
* 4字节的minor brand；
* 一个compatible brands，这是一个数组，数组中每个元素占用4个字节，每个元素代表了这个mp4文件的一种兼容格式；

如下图所示：



前四个字节是box size，值是0x18，也就是十进制的24，说明占用了前24个字节；

紧跟着的是box type，是ftyp；

之后是box body，8个字节，分别是major brand和minor brand；

最后的8个字节是能兼容的格式，mp42和mp41；

# 4.moov -- movie box

moov包含的是文件的metadata信息。

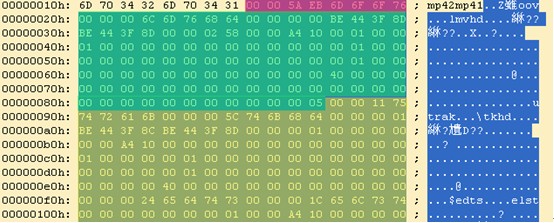
moov是一个container box，具体内容信息由它的子box来诠释。与ftyp相同，moov只有一个，而且只能包含在文件层。一般情况下，moov会紧随ftyp出现。

通常情况下：

moov的boxHeader包含了boxSize和boxType，boxSize是整个moov的size，包含所有的mvhd和trak的size；boxType值为“moov”；

moov的boxBody会包含一个mvhd和若干个trak：

* mvhd作为header部分存在，一般作为moov的第一个子box出现。
* trak包含了一个track的相关信息，是一个container box。



如上图，红色区域是moov的boxHeader，包含了boxSize和boxType；绿色部分是mvhd，也是一个box；后面的内容是若干个track；

## 4.1.mvhd -- movie header box

mvhd的结构如下表所示：

|  |  |  |
| --- | --- | --- |
| **字段** | **字节数** | **含义** |
| box size | 4 | size。只是这个movie header的大小，不包含后续trak的size |
| box type | 4 | box类型，“mvhd” |
| version | 1 | box版本，0或者1，以下各个字段所占的字节数都是按照version==0来说的，因为version一般都是0. |
| flags | 3 |  |
| creation time | 4 | 创建时间。相对于UTC时间19040101零点的秒数。 |
| modification time | 4 | 修改时间。 |
| time scale | 4 | 文件媒体在1秒内的刻度值。可以理解为1秒长度的时间单元数 |
| duration | 4 | 这个track的时间长度。  使用duration和time scale可以计算这个track的时长，例如：  audio track的time scale=8000，duration=560128，那么时长=duration / time scale = 70.016；  video track的time scale=600， duration=42000，那么时长=duration / time scale = 70； |
| rate | 4 | 推荐的播放速度，高16位和低16位分别是浮点数的整数部分和小数部分，也就是[16.16]格式，该值为1.0表示正常向前播放； |
| volume | 2 | 与rate类似，是[8.8]的格式，1.0表示使用最大音量 |
| reserved | 10 | 保留位 |
| matrix | 36 | 视频变换矩阵 |
| pre-defined | 24 |  |
| next track id | 4 | 下一个track使用的id号 |

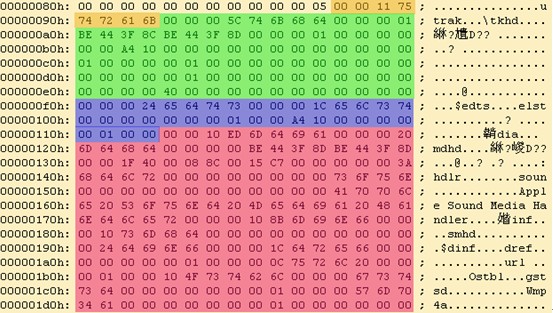
## 4.2.trak -- track box

“trak”也是一个container box，其子box包含了该track的媒体数据引用和描述(hint track除外)。trak包含boxHeader和boxBody，boxHeader包含两部分：boxSize和boxType；boxBody包含了所有的tkhd和mdia；

一个mp4文件中的媒体，可以包含多个track，但至少要有一个track。这些track之间互相独立，有各自的时间和空间信息。“trak”必须包含一个“tkhd”和一个“mdia”，此外还有很多可选的box。

tkhd：track header box；

mdia：media box；



如上图，黄色部分是trak的boxSize+boxType；绿色部分是tkhd的内容；蓝色部分是edts，是一种可选的box；红色为一部分的mdia数据。

### 4.2.1.tkhd -- track header box