【FFmpeg实战】H264, H265硬件编解码基础及码流分析

Linux百里
赞同 1
1 人赞同了该文章

1. 概览

1.1. 为什么要编码

众所周知,视频数据原始体积是巨大的,以720P 30fps的视频为例,一个像素大约3个字节,如下所得,每秒钟产生87MB,这样计算可得一分钟就将产生5.22GB.

数据量/每秒=1280*720*33*3/1024/1024=87MB

因此,像这样体积重大的视频是无法在网络中直接传输的.而视频编码技术也就因运而生.关于视频编码原理的技术可以参考本人其他文章,这里不做过多描述.

1.2. 编码技术

经过很多年的开发迭代,已经有很多大牛实现了视频编码技术,其中最主流的有H.264编码,以及新一代的H.265编码,谷歌也开发了VP8,VP9编码技术.对移动端而言,苹果内部已经实现了如H.264,H.265编码,我们需要使用苹果提供的VideoToolbox框架来实现它.

1.3. 编码分类

- 软件编码(简称软编)：使用CPU进行编码。
- 硬件编码(简称硬编)：不使用CPU进行编码，使用显卡GPU,专用的DSP、FPGA、ASIC芯片等硬件进行编码。

优缺点

- 软编：实现直接、简单，参数调整方便，升级易，但CPU负载重，性能较硬编码低，低码率下质量通常比硬编码要好一点。
- 硬编：性能高，低码率下通常质量低于硬编码器，但部分产品在GPU硬件平台移植了优秀的软编码算法（如X264）的，质量基本等同于软编码。

iOS系统中的硬编码
苹果在iOS 8.0系统之前，没有开放系统的硬件编码解码功能，不过Mac OS系统一直有，被称为Video ToolBox的框架来处理硬件的编码和解码，终于在iOS 8.0后，苹果将该框架引入iOS系统。

1.4. 编码原理

对视频执行编码操作后,原始视频数据会被压缩成三种不同类型的视频帧: I帧,P帧,B帧.

- I帧:关键帧.完整编码的帧.可以理解成是一张完整画面,不依赖其他帧
- P帧:参考前面的I帧或P帧,即通过前面的I帧与自己记录的不同的部分可以形成完整的画面.因此,单独的P帧无法形成画面.
- B帧:参考前面的I帧或P帧以及后面的P帧

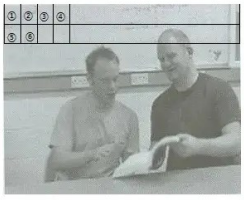
> 补充: I帧的压缩率是7（跟JPG差不多），P帧是20，B帧可以达到50. 但是iOS中一般不开启B帧,因为B帧的存在会导致时间戳同步较为复杂.

两种核心算法

- 帧内压缩

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。

如下图:我们可以通过第 1、2、3、4、5 块的编码来推测和计算第 6 块的编码，因此就不需要对第 6 块进行编码了，从而压缩了第 6 块，节省了空间



帧内预测.png

- 帧间压缩: P帧与B帧的压缩算法

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。

如下图:可以看到前后两帧的差异其实是很小的，这时候用帧间压缩就很有意义。

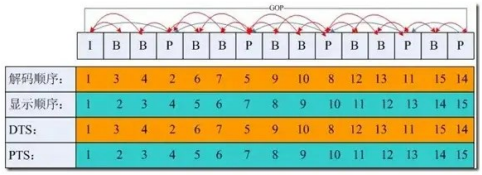
帧间压缩11.jpg

有损压缩与无损压缩

- 有损压缩: 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
- 无损压缩: 压缩前和解压缩后的数据完全一致.优化数据的排列等.

DTS和PTS

- DTS:主要用于视频的解码,在解码阶段使用.
- PTS:主要用于视频的同步和输出.在渲染的时候使用.在没有B frame的情况下.DTS和PTS的输出顺序是一样的。



如上图：I帧的解码不依赖于任何的其它的帧.而P帧的解码则依赖于其前面的I帧或者P帧.B帧的解码则依赖于其前的最近的一个I帧或者P帧 及其后的最近的一个P帧.

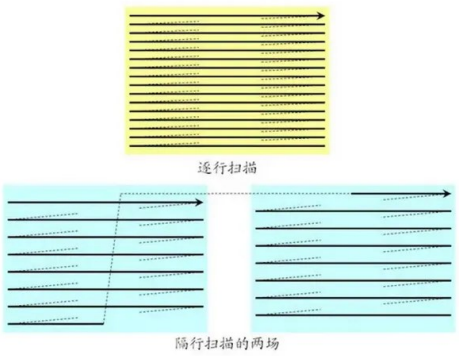
2. 编码数据码流结构

在我们的印象中，一张图片就是一张图像，视频就是很多张图片的集合.。但是因为我们要做音视频编程,就需要更加深入理解视频的本质.

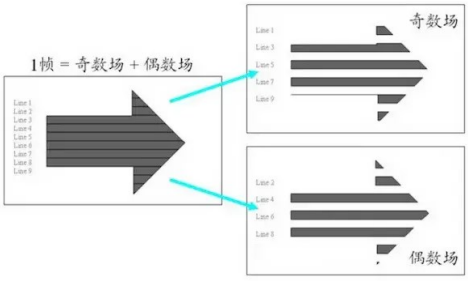
2.1 刷新图像概念.

在编码的码流中图像是个集合的概念,帧、顶场、底场都可以称为图像,一帧通常就是一幅完整的图像.

- 逐行扫描:每次扫描得到的信号就是一副图像，也就是一帧. 逐行扫描适合于运动图像
- 隔行扫描:扫描下来的一帧图像就被分为了两个部分，这每一部分就称为「场」，根据次序分为：「顶场」和「底场」.适合于非运动图像



逐行扫描与隔行扫描



帧与场

2.2. 重要参数

- 视频参数集VPS（Video Parameter Set）

VPS主要用于传输视频分级信息，有利于兼容标准在可分级视频编码或多视点视频的扩展。

（1）用于解释编码过的视频序列的整体结构，包括时域子层依赖关系等。HEVC中加入该结构的主要目的是兼容标准在系统的多子层方面的扩展，处理比如未来的可分级或者多视点视频使用原先的解码器进行解码但是其所需的信息可能会被解码器忽略的问题。

（2）对于给定视频序列的某一个子层，无论其SPS相不相同，都共享一个VPS。其主要包含的信息有：多个子层或操作点共享的语法元素；档次和级别等会话关键信息；其他不属于SPS的操作点特定信息。

（3）编码生成的码流中，第一个NAL单元携带的就是VPS信息

- 序列参数集SPS（Sequence Parameter Set）

包含一个CVS中所有编码图像的共享编码参数。

（1）一段HEVC码流可能包含一个或者多个编码视频序列，每个视频序列由一个随机接入点开始，即IDR/BLA/CRA。序列参数集SPS包含该视频序列中所有slice需要的信息。

（2）SPS的内容大致可以分为几个部分：1、自引ID；2、解码相关信息，如档次级别、分辨率、子层数等；3、某档次中的功能开关标识及该功能的参数；4、对结构和变换系数编码灵活性的限制信息；5、时域可分级信息；6、VUI。

- 图像参数集PPS（Picture Parameter Set）

包含一幅图像所用的公共参数，即一幅图像中所有片段SS（Slice Segment）引用同一个PPS。

（1）PPS包含每一帧可能不同的设置信息，其内容同H.264中的大致类似，主要包括：1、自引信息；2、初始图像控制信息，如初始QP等；3、分块信息。

（2）在解码开始的时候，所有的PPS全部是非活动状态，而且在解码的任意时刻，最多只能有一个PPS处于激活状态。当某部分码流引用了某个PPS的时候，这个PPS便被激活，称为活动PPS，一直到另一个PPS被激活。

参数集包含了相应的编码图像的信息。SPS包含的是针对一连续编码视频序列的参数（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）。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。

通常，SPS 和PPS 在片的头信息和数据解码前传送至解码器。每个片的头信息对应一个pic_parameter_set_id，PPS被其激活后一直有效到下一个PPS被激活；类似的，每个PPS对应一个seq_parameter_set_id，SPS被其激活以后将一直有效到下一个SPS被激活。

参数集机制将一些重要的、改变少的序列参数和图像参数与编码片分离，并在编码片之前传送至解码端，或者通过其他机制传输。

扩展知识点：档次（Profile）、层（Tier）和级别（Level）

- 档次: 主要规定编码器可采用哪些编码工具或算法。
- 级别: 指根据解码端的负载和存储空间情况对关键参数（最大采样率、最大图像尺寸、分辨率、最小压缩比、最大比特率、解码缓冲区DPB大小等）加以限制。

考虑到应用可根据最大的码率和CPB大小来区分，因此有些级别定义了两个层Tier：主层和高层，主层用于大多数应用，而高层用于那些最严苛的应用。

2.3. 原始码流

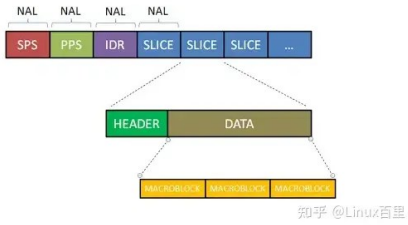
- IDR

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。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。

- 结构

由一个接一个的 NALU 组成的，而它的功能分为两层，VCL(视频编码层)和 NAL(网络提取层).

下图以h264的码流结构为例,如果是h265则在sps前还有vps.



H264码流

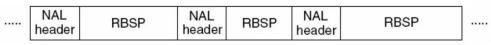
- 组成

NALU (Nal Unit) = NALU头 + RBSP 在 VCL

数据传输或存储之前，这些编码的 VCL 数据，先被映射或封装进 NAL 单元(以下简称 NALU，Nal Unit) 中。每个 NALU 包括一个原始字节序列负荷(RBSP, Raw Byte Sequence Payload)、一组 对应于视频编码的 NALU 头部信息。RBSP 的基本结构是:在原始编码数据的后面填加了结尾 比特。一个 bit“1”若干比特“0”，以便字节对齐。

2.3.1. H.264码流

一个原始的H.264 NALU 单元常由 [StartCode] [NALU Header] [NALU Payload] 三部分组成



NALU组成

- StartCode : Start Code 用于标示这是一个NALU 单元的开始，必须是“00 00 00 01” 或“00 00 01”
- NALU Header

下表为 NAL Header Type

F	1bit	forbidden_zero_bit, H.264定义此位必须为0
NRI	2bit	nal_ref_idc, 0~3, 标识这个NALU的重要性(3最高)
Type	5bit	nal_unit_type, NALU单元的类型
		0 未使用
		1 未使用Data Partitioning、非IDR图像的Slice
		2 使用Data Partitioning、且为Slice A
		3 使用Data Partitioning、且为Slice B
		4 使用Data Partitioning、且为Slice C
		5 IDR图像中的Slice
		6 补充增强信息单元(SEI)
		7 序列参数集(Sequence Parameter Set, SPS)
		8 图像参数集(Picture Parameter Set, PPS)

		9	分界符
		10	序列结束
		11	码流结束
		12	填充
		13...23	保留
		24...31	未使用

NAL Header Type

例如,下面幅图分别代表IDR与非IDR帧具体的码流信息:

2.IDR

在一个NALU中，第一个字节（即NALU header）用以表示其包含数据的类型及其他信息。我们假定一个头信息字节为0x67作为例子：

十六进制	二进制
0x67	0 11 00111

如表所示，头字节可以被解析成3个部分，其中：

1>. forbidden_zero_bit = 0：占1个bit，禁止位，用以检查传输过程中是否发生错误，0表示正常，1表示违反语法；

2>. nal_ref_idc = 3：占2个bit，用来表示当前NAL单元的优先级。非0值表示参考字段/帧/图片数据，其他不那么重要的数据则为0。对于非0值，值越大表示NALU重要性越高

3>. nal_unit_type = 7：最后5位用以指定NALU类型，NALU类型定义如上表

从表中我们可以获知,NALU类型1-5为视频帧，其余则为非视频帧。在解码过程中，我们只需要取出NALU头字节的后5位，即将NALU头字节和0x1F进行与计算即可得知NALU类型，即：

NALU类型 = NALU头字节 & 0x1F

> 注意: 可以将start code理解为不同nalu的分隔符,header是某种类型的key,payload是该key的value.

码流格式

H.264标准中指定了视频如何编码成独立的包，但如何存储和传输这些包却未作规范，虽然标准中包含了一个Annex附件，里面描述了一种可能的格式Annex B，但这并不是一个必须要求的格式。

为了针对不同的存储传输需求，出现了两种打包方法。一种即Annex B格式，另一种称为AVCC格式。

- Annex B

从上文可知，一个NALU中的数据并未包含他的大小（长度）信息，因此我们并不能简单的将一个个NALU连接起来生成一个流，因为数据流的接收端并不知道一个NALU从哪里结束，另一个NALU从哪里开始。

Annex B格式用起始码（Start Code）来解决这个问题，它在每个NALU的开始处添加三字节或四字节的起始码0x000001或0x00000001。通过定位起始码，解码器就可以很容易的识别NALU的边界。

当然，用起始码定位NALU边界存在一个问题，即NALU中可能存在与起始码相同的数据。为了防止这个问题，在构建NALU时，需要将数据中的0x000000,0x000001,0x000002,0x000003中插入防竞争字节（Emulation Prevention Bytes)0x03，使其变为：

0x000000 = 0x0000 03 00
0x000001 = 0x0000 03 01
0x000002 = 0x0000 03 02
0x000003 = 0x0000 03 03

解码器在检测到0x000003时，将0x03抛弃，恢复原始数据。

由于Annex B格式每个NALU都包含起始码，所以解码器可以从视频流随机点开始进行解码，常用于实时的流格式。在这种格式中通常会周期性的重复SPS和PPS，并且经常时在每一个关键帧之前。

- AVCC

AVCC格式不使用起始码作为NALU的分界，这种格式在每个NALU前都加上一个指定NALU长度的大端格式表示的前缀。这个前缀可以是1、2或4个字节，所以在解析AVCC格式的时候需要将指定的前缀字节数的值保存在一个头部对象中，这个都通常称为extradata或者sequence header。同时，SPS和PPS数据也需要保存在extradata中。

H.264 extradata语法如下：

bits	line by byte	remark	
8	version	always	0x01
8	avc profile	sps0	
8	avc compatibility	sps0	
8	avc level	sps0	
6	reserved	all bits on	

2	NALULengthSizeMinusOne		
3	reserved	all bits on	
5	number of SPS NALUs usually	1	
16	SPS size		
N	variable SPS NALU data		
8	number of PPS NALUs usually	1	
16	PPS size		
N	variable PPS NALU data		

其中第5字节的后2位表示的就是NAL size的字节数。需要注意的是，这个NALULengthSizeMinusOne是NALU前缀长度减一，即，假设前缀长度为4，那么这个值应该为3。

这里还需要注意的一点是，虽然AVCC格式不使用起始码，但防竞争字节还是有的。

AVCC格式的一个优点在于解码器配置参数在一开始就配置好了，系统可以很容易的识别NALU的边界，不需要额外的起始码，减少了资源的浪费，同时可以在播放时调到视频的中间位置。这种格式通常被用于可以被随机访问的多媒体数据，如存储在硬盘的文件。

2.3.2. H.265码流

HEVC全称High Efficiency Video Coding(高效率视频编码，又称H.265)，是比H.264更优秀的一种视频压缩标准。HEVC在低码率视频压缩上，提升视频质量、减少容量即节省带宽方面都有突出表现。

H.265标准围绕H.264编码标准,保留原有的某些技术，同时对一些技术进行改进，编码结构大致上和H.264的架构类似。这里着重讲一下两者编码格式的区别。

同H.264一样，H.265也是以NALU的形式组织起来。而在NALU header上，H.264的HALU header是一个字节，而H.265则是两个字节。我们同样假定一个头信息为0x4001作为例子:

十六进制	二进制
0x4001	0 100000 000000 001

如表所示，头信息可以被解析成4个部分，其中:

- forbidden_zero_bit = 0：占1个bit，与H.264相同，禁止位，用以检查传输过程中是否发生错误，0表示正常，1表示违反语法;
- nal_unit_type = 32：占6个bit，用来用以指定NALU类型
- nuh_reserved_zero_6bits = 0：占6位，预留位，要求为0，用于未来扩展或3D视频编码
- nuh_temporal_id_plus1 = 1：占3个bit，表示NAL所在的时间层ID

对比H.264的头信息，H.265移除了nal_ref_idc，此信息被合并到了nal_unit_type中，H.265NALU类型规定如下:

nal_unit_type	NALU类型	备注
0	NAL_UNIT_CODE_SLICE_TRAIL_N	非关键帧
1	NAL_UNIT_CODED_SLICE_TRAIL_R	
2	NAL_UNIT_CODED_SLICE_TSA_N	
3	NAL_UINT_CODED_SLICE_TSA_R	
4	NAL_UINT_CODED_SLICE_STSA_N	
5	NAL_UINT_CODED_SLICE_STSA_R	
6	NAL_UNIT_CODED_SLICE_RADL_N	
7	NAL_UNIT_CODED_SLICE_RADL_R	
8	NAL_UNIT_CODED_SLICE_RASL_N	
9	NAL_UNIT_CODE_SLICE_RASL_R	
10 ~ 15	NAL_UNIT_RESERVED_X	保留
16	NAL_UNIT_CODED_SLICE_BLA_W_LP	关键帧
17	NAL_UNIT_CODE_SLICE_BLA_W_RADL	
18	NAL_UNIT_CODE_SLICE_BLA_N_LP	
19	NAL_UNIT_CODE_SLICE_IDR_W_RADL	
20	NAL_UNIT_CODE_SLICE_IDR_N_LP	
21	NAL_UNIT_CODE_SLICE_CRA	
22 ~ 31	NAL_UNIT_RESERVED_X	保留
32	NAL_UNIT_VPS	VPS(Video Paramater Set)
33	NAL_UNIT_SPS	SPS
34	NAL_UNIT_PPS	PPS
35	NAL_UNIT_ACCESS_UNIT_DELIMITER	
36	NAL_UNIT_EOS	
37	NAL_UNIT_EOB	
38	NAL_UNIT_FILLER_DATA	
39	NAL_UNIT_SEI	Prefix SEI
40	NAL_UNIT_SEI_SUFFIX	Suffix SEI
41 ~ 47	NAL_UNIT_RESERVED_X	保留
48 ~ 63	NAL_UNIT_UNSPECIFIED_X	未规定
64	NAL_UNIT_INVALID	

具体type含义可以参考这篇文档type类型

H.265的NALU类型是在信息头的第一个字节的第2到7位，所以判断H.265NALU类型的方法是将NALU第一个字节与0x7E进行与操作并右移一位，即：

NALU类型 = （NALU头第一字节 & 0x7E）>> 1

与H.264类似，H.265码流也有两种封装格式，一种是用起始码作为分界的Annex B格式，另一种则是在NALU头添加NALU长度前缀的格式，称为HVCC。在HVCC中，同样需要一个extradata来保存视频流的编解码参数，其格式定义如下：

bits	line by byte	remark
8	configurationVersion	always 0x01
2	general_profile_space	
1	general_tier_flag	
5	general_profile_idc	
32	general_profile_compatibility_flags	
48	general_constraint_indicator_flags	
8	general_level_idc	
4	reserved	‘1111’ b
12	min_spatial_segmentation_idc	
6	reserved	‘111111’ b
2	parallelismType	
6	reserved	‘111111’ b
2	chromaFormat	
5	reserved	‘11111’ b
3	bitDepthLumaMinus8	
5	reserved	‘11111’ b
3	bitDepthChromaMinus8	
16	avgFrameRate	
2	constantFrameRate	
3	numTemporalLayers	
1	tmporalIdNested	
2	lengthSizeMinusOne	
8	numOfArrays	

Repeated of Array(VPS/SPS/PPS)

1| array_completeness

1| reserved| ‘0’ b

6| NAL_unit_type

16| numNalus

16| nalUnitLength

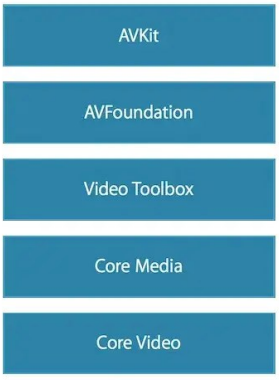
N| NALU data

从上表可以看到，在H.265的extradata后半段是一段格式重复的数组数据，里面需要包含的除了与H.264相同的SPS、PPS外，还需多添加一个VPS。

VPS（Video Parament Set,视频参数集),在H.265中类型为32。VPS用于解释编码过的视频的整体结构，包括时域子层依赖关系等，主要目的在于兼容H.265标准在系统的多子层方面的扩展。

3. iOS中的视频编解码

iOS中视频处理框架分层



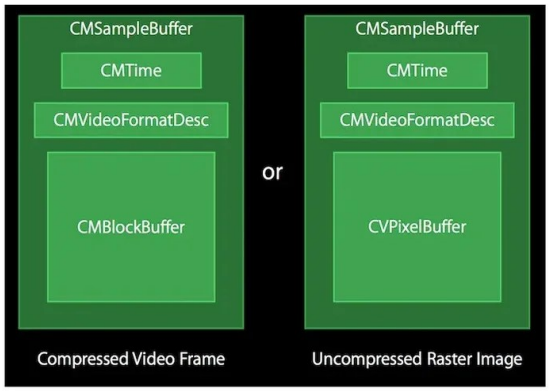
3.1. 框架的选择

对于 AVKit、AVFoundation 和 VideoToolbox 来说，他们的功能和可定制性越来越强，但相应的使用难度也越大，因此你应该根据实际需求合理的选择使用哪个层级的接口。事实上，即使你使用 AVFoundation 或 AVKit 依然可以获得硬件加速的效果，你失去的只是直接访问硬编解码器的权限。对于 VideoToolbox 来说，你可以通过直接访问硬编解码器，将 H.264 文件或传输流转换为 iOS 上的 CMSampleBuffer 并解码成 CVPixelBuffer，或将为压缩的 CVPixelBuffer 编码成 CMSampleBuffer：

3.2. 视频数据的容器

调用 AVCaptureSession 拍摄输出的每一帧图像都会被包装成 CMSampleBuffer 对象，通过这个 CMSampleBuffer 对象你就可以获取到未压缩的 CVPixelBuffer 对象；如果读取 H.264 文件你也可以获取数据生成压缩的 CMBlockBuffer 对象并创建一个 CMSampleBuffer 对象给 VideoToolbox 来解码。

也就是说，CMSampleBuffer 既可以作为 CVPixelBuffer 对象的容器，也可以作为 CMBlockBuffer 对象的容器，CVPixelBuffer 可以说是未压缩的图像数据容器，而 CMBlockBuffer 则是压缩图像数据容器。



3.3. 编码数据裸流

NALU. 对于一个 H.264 裸流或者文件来说，它是由一个一个的 NALU(Network Abstraction Layer Unit) 单元组成，每个 NALU 既可以表示图像数据，也可以表示处理图像所需要的参数数据。它主要有两种格式：Annex B 和 AVCC。也被称为 Elementary Stream 和 MPEG-4 格式，Annex B 格式以 0x000001 或 0x00000001 开头，AVCC 格式以所在的 NALU 的长度开头。

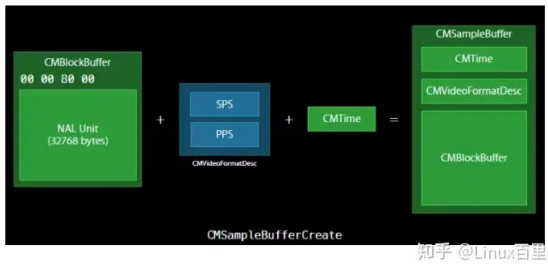
3.4. VideoToolBox中常用数据结构

- CMSampleBuffer: 存放编解码前后的视频图像的容器数据结构
- CVPixelBuffer: 编码前和解码后的图像数据结构
- CMBlockBuffer: 编码后图像的数据结构
- CMVideoFormatDescription: 图像存储方式，编解码器等格式描述
- pixelBufferAttributes: : CFDictionary对象，可能包含了视频的宽高，像素格式类型（32RGBA, YCbCr420），是否可以用于OpenGL ES等相关信息
- CMTime: 时间戳相关。时间以 64-big/32-bit形式出现。 分子是64-bit的时间值，分母是32-bit的时标(time scale)

3.5. 为编码的数据添加start code.

在iOS中使用vtCompressionSession编码后的数据不包含start code.需要我们自行添加,为什么要添加start code? 如上文所说,为了区分每个NALU及以后提取vps,sps,pps等关键信息.

如下图,我们在编码回调中可以拿到编码后的 `CMSampleBuffer` 数据.如果是h265, `CMVideoFormatDesc` 中还有vps.而这段数据在推流前必须寻找到关键数据vps,sps,pps以及添加start code.因为在 `CMBlockBufferRef` 中可能还含有一个或多个NALU,所以我们需要通过遍历它的内存地址找到在每个NALU的分割点将数据替换为start code.具体代码操作参考实战篇.



CMSampleBufferCreate

接下来，我们就需要处理这一帧视频的图像数据了。通过 CMSampleBufferGetDataBuffer 和 CMBlockBufferGetDataPointer 我们可以获取视频数据的内存地址。VTCompressionSession 编码出来的视频帧为 AVCC 格式，因此我们可以读取头部 4 个字节数据来获取当前 NALU 的长度。这里有一个需要注意的是，AVCC 格式使用大端字节序，它可能跟当前使用的系统字节序不一样，事实上，iOS 系统使用小端字节序，因此我们需要先将这个长度数据转换为 iOS 系统使用的小端字节序：

NALUnitLength = CFSwapInt32BigToHost(NALUnitLength)

> 硬编码基本上是硬解码的一个逆过程。解析出参数集SPS和PPS，加上开始码后组装成NALU。提取出视频数据，将长度码转换成开始码，组长成NALU。将NALU发送出去。

3.6. 将H.264裸流解码为CMSampleBuffer

如上图,我们知道，CMSampleBuffer = CMTime + FormatDesc + CMBlockBuffer . 需要从H264的码流里面提取出以上的三个信息。最后组合成CMSampleBuffer，提供给硬解码接口来进行解码工作。

在H.264, H.265的语法中，有一个最基础的层，叫做Network Abstraction Layer, 简称为NAL。编码数据正是由一系列的NAL单元(NAL Unit, 简称NAUL)组成的。

NALU

编码码流由NALU单元组成,一个NALU可能包含有:

- 视频帧,视频帧也就是视频片段，具体有 P帧, I帧，B帧

H264的码流

- 编码属性合集-FormatDesc(包含VPS,SPS和PPS)

流数据中，属性集合可能是这样的：(如果是H.265码流,SPS前面还有VPS)

属性集合

经过处理之后，在Format Description中则是:

Format Description

要从基础的流数据将SPS和PPS转化为Format Desc中的话，需要调用 `CMVideoFormatDescriptionCreateFromH264ParameterSets`,`CMVideoFormatDescriptionGetHEVCParameterSetAtIndex` 方法

- NALU header

对于流数据来说，一个NAUL的Header中，可能是0x00 00 01或者是0x00 00 00 01作为开头(两者都有可能，下面以0x00 00 01作为例子)。0x00 00 01因此被称为开始码(Start code).

NALU header

总结以上知识，我们知道编码码流由NALU单元组成，NALU单元包含视频图像数据和编码器的参数信息。其中视频图像数据就是CMBlockBuffer，而编码器参数信息则可以组合成FormatDesc。具体来说参数信息包含VPS,SPS（Sequence Parameter Set）和PPS（Picture Parameter Set）.如下图显示了一个H.264码流结构：

H.264码流

提取sps和pps生成FormatDesc

- 每个NALU的开始码是0x00 00 01，按照开始码定位NALU
- 通过类型信息找到sps和pps并提取，开始码后第一个byte的后5位，7代表sps，8代表pps
- 使用CMVideoFormatDescriptionCreateFromH264ParameterSets函数来构建CMVideoFormatDescriptionRef

提取视频图像数据生成CMBlockBuffer

- 通过开始码，定位到NALU
- 确定类型为数据后，将开始码替换成NALU的长度信息（4 Bytes）
- 使用CMBlockBufferCreateWithMemoryBlock接口构造CMBlockBufferRef

根据需要，生成CMTime信息。（实际测试时，加入time信息后，有不稳定的图像，不加入time信息反而没有，需要进一步研究，这里建议不加入time信息）

根据上述得到CMVideoFormatDescriptionRef、CMBlockBufferRef和可选的时间信息，使用CMSampleBufferCreate接口得到CMSampleBuffer数据这个待解码的原始的数据。如下图所示的H264数据转换示意图。

CMSampleBufferCreate

3.7. 将 CMSampleBuffer渲染到界面

显示的方式有两种：

将CMSampleBuffers提供给系统的AVSampleBufferDisplayLayer 直接显示

- 使用方式和其它CALayer类似。该层内置了硬件解码功能，将原始的CMSampleBuffer解码后的图像直接显示在屏幕上面，非常的简单方便。

利用OPenGL自己渲染
通过VTDecompression接口来，将CMSampleBuffer解码成图像，将图像通过UIImageView或者OpenGL上显示。

作者：小东邪啊
链接：jianshu.com/p/868bb8738 ...

音视频开发 视频教程：ke.qq.com/course/320213 ...

音视频开发学习资料、教学视频，免费分享有需要的可以自行添加学习交流群：739729163 领取

发布于 2023-06-29 21:35 · IP 属地湖南

软硬件 H.264 (MPEG-4 高级视频编码) 硬件研发

写下你的评论...

还没有评论，发表第一个评论吧

文章被以下专栏收录

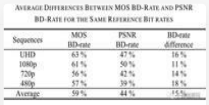
音视频开发
分享音视频开发技术知识

推荐阅读

H264--1--编码原理以及I帧B帧P帧、pts&dts
来源于：https://blog.csdn.net/bingqing... ---------- 前言 ---------- H264是新一代的编码标准，以高压缩高质量和支持多种网络...
xiaofu

H265与ffmpeg改进开发
H265与ffmpeg改进开发 1. IntroductionKSC265是集编码、解码于一体的H.265编解码软件，完全遵循H.265协议标准。符合H.265编码规范的视频都可以通过KSC265进行解码，通过KSC265编码的视频...
吴建明wujianming

H264和X264究竟有什么区别？
零声Github分享官



H265比H264压缩性能提升50%？
codec2021