**音视频基础知识**

[一 音频 1](#_Toc1643)

[1 名词解释 1](#_Toc20069)

[1.1 什么是声音 1](#_Toc29925)

[1.2 声音AD转化 1](#_Toc22601)

[1.3 什么是CODEC 4](#_Toc29410)

[1.4 音频采样率 4](#_Toc14457)

[1.5 位深度/采样大小/采样精度 5](#_Toc15491)

[1.6 通道数 5](#_Toc13097)

[1.7 数据带宽 5](#_Toc856)

[1.8 音频帧率 6](#_Toc277)

[2 音频数据裸流-PCM 7](#_Toc23800)

[2.1 PCM数据格式 7](#_Toc10799)

[2.2 存储方式 8](#_Toc16213)

[· 回顾：字节序 10](#_Toc27437)

[3 音频压缩编码标准 12](#_Toc16567)

[3.1 WAVE音频文件格式 13](#_Toc9236)

[1 RIFF文件与WAV文件 13](#_Toc16511)

[2 WAV文件头 14](#_Toc102)

[二 视频 16](#_Toc1832)

[1 常见接口 16](#_Toc19889)

[2 视频播放器原理 18](#_Toc7962)

[3 流媒体协议 21](#_Toc23428)

[4 视频文件格式/封装格式 22](#_Toc30027)

[6 视频压缩编码标准 24](#_Toc22548)

[· MPEG-1 24](#_Toc1995)

[· MPEG-2 25](#_Toc30497)

[· MPEG-4 27](#_Toc22943)

[· MPEG-7 29](#_Toc23763)

[· MPEG-21 Multimedia Framework 30](#_Toc24225)

[· WMV（WINDOWS MEDIA VIDEO） 32](#_Toc21669)

[· H.261 33](#_Toc26434)

[· H.263 33](#_Toc23712)

[· H.263+ 34](#_Toc4949)

[· H.263++ 35](#_Toc3954)

[· H.264／MPEG-4 AVC 36](#_Toc19545)

[· AVS(数字音视频编解码技术标准/MPEG中国代表团) 41](#_Toc3102)

[7 视频(图像)颜色编码 43](#_Toc2769)

[· RGB 43](#_Toc12383)

[·RGB格式 46](#_Toc1970)

[··网页格式 46](#_Toc25332)

[··RGB555 46](#_Toc16880)

[··RGB565 47](#_Toc21864)

[··RGB24 48](#_Toc16808)

[··RGB32 48](#_Toc8479)

[· YUV 49](#_Toc3461)

[·YUV采样方式 51](#_Toc420)

[·YUV存储格式： 51](#_Toc11335)

[·· YUYV 格式 （属于YUV422） 52](#_Toc16012)

[·· UYVY 格式 （属于YUV422） 52](#_Toc7070)

[·· YUV422P（属于YUV422） 52](#_Toc10615)

[·· YV12，YU12(I420)格式（属于YUV420P） 53](#_Toc11953)

[·· NV12、NV21（属于YUV420） 53](#_Toc30950)

[·· YV12和YU12/I420的区别 53](#_Toc6666)

[·· YUV420P和YUV420SP的区别 54](#_Toc20232)

[·YUV文件大小计算 54](#_Toc24512)

[· YUV和RGB互相转换 56](#_Toc29613)

[·· 转换公式 56](#_Toc9752)

[·· 转换方法及转换效率 56](#_Toc17141)

[· 视频质量系数PSNR 57](#_Toc5101)

[8 BMP图像数据格式 58](#_Toc7816)

[1 简介 58](#_Toc27042)

[2 BMP格式结构 58](#_Toc23878)

[· BMP文件头 59](#_Toc625)

[· BMP信息头 60](#_Toc21980)

[· BMP调色板 61](#_Toc2378)

[# 知识扩展 65](#_Toc17056)

[FFmpeg 65](#_Toc17834)

[1 概念 65](#_Toc29085)

[2 项目结构 65](#_Toc18436)

[3 命令集合 66](#_Toc2491)

**一 音频**

1 名词解释

* 1. **什么是声音**

能量波，有频率有振幅，

**频率高低**就是**音调**，**振幅大小**就是**音量**；**采样率**是**对频率采样**，**采样精度**是**对幅度采样**

人耳能听到的频率范围是**200-20KHz**，

声音的特性可由**三个要素**来描述，即**响度**、**音调**和**音色**。

**响度**（loudness）：**又称音量；**人耳对声音强弱的主观感觉称为响度。响度和声波振动的**幅度**有关。一般说来，声波振动幅度越大则响度也越大。当我们用较大的力量敲鼓时，鼓膜振动的幅度大，发出的声音响；轻轻敲鼓时，鼓膜振动的幅度小，发出的声音弱。

**音调**（Pitch）：人耳对声音高低的感觉称为音调。音调主要与**声波的频率**有关。声波的频率高，则音调也高。当我们分别敲击一个小鼓和一个大鼓时，会感觉它们所发出的声音不同。小鼓被敲击后振动频率快，发出的声音比较清脆，即音调较高；而大鼓被敲击后振动频率较慢，发出的声音比较低沉，即音调较低。

**音色**（Timbre）：**亦称音品；**音色是人们区别具有同样响度、同样音调的两个声音之所以不同的特性，或者说是人耳对各种频率、各种强度的声波的综合反应。音色与声波的振动波形有关，或者说与声音的频谱结构有关。

* 1. **声音AD转化**

自然界中的声音非常复杂，波形极其复杂，通常我们采用的是**脉冲代码调制编码**。即**PCM编码**。PCM通过**抽样、量化、编码**三个步骤将连续变化的模拟信号转换为数字编码。

A: 抽样

**声音**其实是一种**能量波**，因此也有频率和振幅的特征，**频率对应于时间轴线**，**振幅对应于电平轴线**。波是无限光滑的，弦线可以看成由无数点组成，由于存储空间是相对有限的，数字编码过程中，必须对弦线的点进行采样。采样的过程就是抽取某点的频率值，很显然，在一秒中内抽取的点越多，获取得**频率信息**更丰富，为了复原波形，一次振动中，必须有2个点的采样，人耳能够感觉到的最高频率为20kHz，因此要满足人耳的听觉要求，则需要至少每秒进行40k次采样，用40kHz表达，这个40kHz就是采样率。我们常见的CD，采样率为44.1kHz。

B：量化

采集过程中**视频和音频同步**是非常重要的，光有频率信息是不够的，我们还必须获得该频率的能量值并量化，用于表示信号强度。量化电平数为**2的整数次幂**，我们常见的CD位**16级**的**采样大小**，即2的4次方。

采样大小相对采样率更难理解，因为要显得抽象点，举个简单例子：假设对一个波进行8次采样，采样点分别对应的能量值分别为A1-A8，但我们只使用2bit的采样大小，结果我们只能保留A1-A8中4个点的值而舍弃另外4个。如果我们进行3bit的采样大小，则刚好记录下8个点的所有信息。**采样率**和**采样大小**的**值越大**，记录的波形更接近原始信号。

C: 编码

根据采样率和采样大小可以得知，相对自然界的信号，音频编码最多只能做到无限接近，至少目前的技术只能这样了，相对自然界的信号，任何数字音频编码方案都是有损的，因为无法完全还原。在计算机应用中，能够达到**最高保真水平**的就是**PCM编码**，被广泛用于素材保存及音乐欣赏，CD、DVD以及我们常见的WAV文件中均有应用。因此，PCM约定俗成了**无损编码**，因为PCM代表了数字音频中最佳的保真水准，并不意味着PCM就能够确保信号绝对保真，PCM也只能做到最大程度的无限接近。我们而习惯性的把MP3列入有损音频编码范畴，是相对PCM编码的。强调编码的相对性的有损和无损，是为了告诉大家，要做到真正的无损是困难的，就像用数字去表达圆周率，不管精度多高，也只是无限接近，而不是真正等于圆周率的值

**采样频率**：即取样频率,指**每秒**钟取得**声音样本**的**次数**。

采样频率越高,声音的质量也就越好,声音的还原也就越真实，但同时它占的资源比较多。由于人耳的分辨率很有限,太高的频率并不能分辨出来。在16位声卡中有22KHz、44KHz等几级,其中，22KHz相当于普通FM广播的音质，44KHz已相当于CD音质了，目前的常用采样频率都不超过48KHz。

**采样位数**：即**采样值**或**取样值**（就是将采样样本幅度量化）。

它是用来衡量声音波动变化的一个参数，也可以说是**声卡**的**分辨率**。它的数值越大，分辨率也就越高，所发出声音的能力越强。

**声道数**：很好理解，有单声道和立体声之分。

单声道的声音只能使用一个喇叭发声（有的也处理成两个喇叭输出同一个声道的声音），立体声的PCM 可以使两个喇叭都发声（一般左右声道有分工） ，更能感受到空间效果。

**时长**： 采样时长

* 1. **什么是CODEC**

音频压缩CO+解压缩DEC，CODEC就是多媒体数字信号编解码器，主要负责DAC和ADC。

不管是音频加速器好，还是I/O控制器好，他们输入输出的都是**纯数字信号**，我们要使用声卡上的Line Out插孔输出信号的话，信号就必须经过声卡上的CODEC的转换处理。可以说，声卡模拟输入输出的品质和CODEC的转换品质有着重大的关系，音频加速器或I/O控制器决定了声卡内部数字信号的质量，而CODEC则决定了模拟输入输出的好坏。

* 1. **音频采样率**

音频采样率是指录音设备在一秒钟内对声音信号的采样次数，

采样频率越高声音的还原就越真实越自然。

在当今的主流采集卡上，采样频率一般共分为11025Hz、22050Hz、24000Hz、44100Hz、48000Hz五个等级，

11025Hz能达到AM调幅广播的声音品质，而22050Hz和24000HZ能达到FM调频广播的声音品质，44100Hz则是理论上的CD音质界限，48000Hz则更加精确一些。

在数字音频领域，常用的采样率有：

8,000 Hz - 电话所用采样率, 对于人的说话已经足够

11,025 Hz - AM调幅广播所用采样率

22,050 Hz/24,000 Hz - FM调频广播所用采样率

32,000 Hz - miniDV 数码视频 camcorder、DAT (LP mode)所用采样率

44,100 Hz - 音频 CD, 也常用于 MPEG-1音频（VCD, SVCD, MP3）所用采样率

47,250 Hz - 商用 PCM 录音机所用采样率

48,000 Hz - miniDV、数字电视、DVD、DAT、电影和专业音频所用的数字声音所用采样率

50,000 Hz - 商用数字录音机所用采样率

96,000 /192,000 Hz - DVD-Audio、一些 LPCM DVD 音轨、BD-ROM（蓝光盘）音轨、和 HD-DVD （高清晰度 DVD）音轨所用所用采样率

2.8224 MHz - Direct Stream Digital 的 1 位 sigma-delta modulation 过程所用采样率。

* 1. **位深度/采样大小/采样精度**

每采集一次声音数据的大小，是位深度，那么单位肯定是 bit了.

* 1. **通道数**

和硬件参数有关，采集声音源的设备有几个.

* 1. **数据带宽**

采样率48000,位深度(采样大小) 16bit ,通道数2；

设备1秒内可以采集到多少音频数据是:

48000 \* 16 \* 2 = 1536000 位

48000 \* 16 \* 2 / 8 = 192000 字节.

也就是我的设备在一秒内可以采集192000 Byte

**则数据带宽 192000B/s**

* 1. **音频帧率**

音频的帧率，怎么理解呢：

每秒内采集48000次，这个是总的采集次数。

也许我们要分为100次，每次也就采集480，或者分为50次，每次采集960,。

这个就要看具体的硬件呢，所以硬件肯定会开放一个接口的.

让你获取一个minbufsize,意思就是这个,每一次提取多少字节.

2 音频数据裸流-PCM

**PCM**(Pulse Code Modulation，**脉冲编码调制**)音频数据是**未经压缩**的**音频采样数据裸流**，它是由**模拟信号**经过**采样、量化、编码**转换成的标准数字音频数据。

描述PCM数据的6个参数：

**Sample Rate :**

采样频率。8kHz(电话)、44.1kHz(CD)、48kHz(DVD)。

**Sample Size**

量化位数。通常该值为16-bit。

**Number of Channels**

通道个数。常见的音频有立体声(stereo)和单声道(mono)两种类型，立体声包含左声道和右声道。

另外还有环绕立体声等其它不太常用的类型。

**Sign**

表示样本数据是否是有符号位，比如用一字节表示的样本数据，

有符号的话表示范围为-128 ~ 127，无符号是0 ~ 255。

**Byte Ordering**

字节序。字节序是little-endian还是big-endian。通常均为little-endian。

**Integer Or Floating Point**

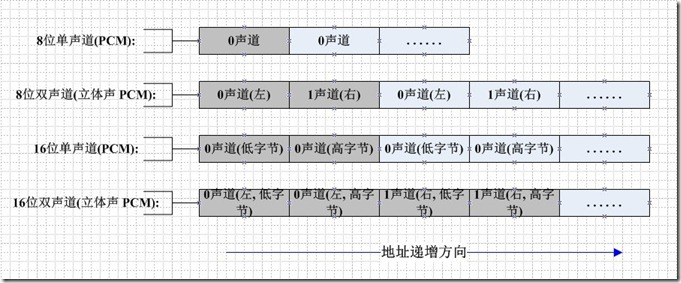
整形或浮点型。大多数格式的PCM样本数据使用整形表示，

而在一些对精度要求高的应用方面，使用浮点类型表示PCM样本数据。

**2.1 PCM数据格式**

如果是单声道的音频文件，采样数据按时间的先后顺序依次存入（有的时候也会采用LRLRLR方式存储，只是另一个声道的数据为0），如果是双声道的话就按照LRLRLR的方式存储，存储的时候与**字节序**有关。

big-endian模式如下图所示：



**2.2 存储方式**

对于双声道音频来说，Packed方式为两个声道的数据交错存储；Planar方式为两个声道分开存储。假设一个L/R为一个采样点，数据存储的方式如下所示：

Packed: L R L R L R L R

Planar: L L L L R R R R

说明：

· Planar模式是ffmpeg内部存储模式，我们实际使用的音频文件都是Packed模式的。

· FFmpeg解码不同格式的音频输出的音频采样格式不是一样。

测试发现，其中AAC解码输出的数据为浮点型的 AV\_SAMPLE\_FMT\_FLTP 格式， MP3解码输出的数据为 AV\_SAMPLE\_FMT\_S16P 格式（使用的mp3文件为16位深）。

具体采样格式可以查看解码后的AVFrame中的format成员或解码器的AVCodecContext中的sample\_fmt成员。

· Planar或者Packed模式直接影响到保存文件时写文件的操作，操作数据的时候一定要先检测音频采样格式。

**· 回顾：字节序**

谈到字节序的问题，必然牵涉到两大CPU派系。那就是Motorola的PowerPC系列CPU和Intel的x86系列CPU。

PowerPC系列采用big endian方式存储数据，而x86系列则采用little endian方式存储数据。

那么究竟什么是big endian，什么又是little endian？

**big endian**是指**低地址**存放**最高有效字节**（MSB，Most Significant Bit），

**little endian**则是**低地址**存放**最低有效字节**（LSB，Least Significant Bit）。

下面用图像加以说明。

比如数字0x12345678在两种不同字节序CPU中的存储顺序如下所示：

Big Endian

低地址 高地址

----------------------------------------------------------------------------->

| 12 | 34 | 56 | 78 |

Little Endian

低地址 高地址

----------------------------------------------------------------------------->

| 78 | 56 | 34 | 12 |

所有网络协议都是采用big endian的方式来传输数据的。所以也把big endian方式称之为**网络字节序**。当两台采用不同字节序的主机通信时，在发送数据之前都必须经过字节序的转换成为网络字节序后再进行传输。

3 音频压缩编码标准

**音频编码**的主要作用是将**音频采样数据**（**PCM**等）**压缩**成为**音频码流**，从而**降低**音频的**数据量**。

音频编码也是互联网视音频技术中一个重要的技术。

但是一般情况下音频的数据量要远小于视频的数据量，因而即使使用稍微落后的音频编码标准，而导致音频数据量有所增加，也不会对视音频的总数据量产生太大的影响。高效率的音频编码在同等的码率下，可以获得更高的音质。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 推出机构 | 推出时间 | 适用领域 | 文件格式 |
| AAC | MPEG | 1997 | 各个领域（新） |  |
| AC-3 | Dolby Inc. | 1992 | 电影 |  |
| MP3 | MPEG | 1993 | 各个领域（旧） | .mp3 |
| WMA | Microsoft Inc. | 1999 | 微软平台 |  |
| WAVE |  |  | 微软 | .wav |
| Ogg |  |  |  |  |
| RealAudio |  |  | 在线播放 |  |
|  |  |  |  |  |

由表可见，近年来并未推出全新的音频编码方案，可见音频编码技术已经基本可以满足人们的需要。音频编码技术近期绝大部分的改动都是在MP3的继任者——AAC的基础上完成的。

AAC+ > MP3PRO > AAC> RealAudio > WMA > MP3

**3.1 WAVE音频文件格式**

WAVE声音文件格式是目前Windows最直接保存声音数据的文件格式.在涉及声音信号处理时大多是对WAV文件直接操作,有必要搞清楚所研究声音的文件格式.

**1 RIFF文件与WAV文件**

在Windows环境下,大部分多媒体文件都依循着一种结构来存放信息,称为**资源互换文件格式**(Resources Interchange File Format),简称RIFF。

比如声音的WAV文件,视频的AVI文件,动画的MMM文件等均是由此结构衍生出来的. 所以,要掌握多媒体文件格式,首先得认识RIFF的结构.

**RIFF**是一种**树状结构**,其基本组成**单位**是**chunk**(即块),每个chunk由辨识码,数据大小和数据组成。

如下图：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 辨识码 | | | | 数据大小 | | | | 数据 | | | | | | | | | | | | | | | | | | | | | |

可以看出,一个chunk的长度,就是数据的大小加上8Byte.

一般而言,**chunk**本身不允许**内部再包含**chunk,

但有**两个例外**:以"**RIFF**"和以"**UST**"为**辨识码**的chunk。

针对这两种chunk,RIFF又从原先的"裸数据"中切出4Byte作为"格式辨别码",

如下图所示.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | I | F | F |  |  |  |  | W | A | V | E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 辨识码 | | | | 数据大小 | | | | 格式辨识码 | | | | 数据 | | | | | | | | | | | | | | | |
| 辨识码 | | | | 数据大小 | | | | 数据 | | | | | | | | | | | | | | | | | | | |

对RIFF的树状结构有所了解之后,可以知道它相当于一个根目录,而格式辨识码则相当于具体的盘符如C:,D:等等.

Windows下的各种**多媒体文件格式**就如同在磁盘机下规定只能存放怎样的目录,而在该目录下仅能存放何种数据.

**2 WAV文件头**

顾名思义,WAV就是波形音频文件(Wave Audio),是Windows中用来表示数字化声音的一种标准格式,

其: **文件扩展名**为.wav,是一种非常简单的RIFF文件,

**格式辨识码**为"WAVE".整个**WAV文件**分成**两部分**:文件头和**数据块**.

struct **RIFF\_HEADER**

{

char szRiffID[4]; // 'R','I','F','F'

DWORD dwRiffSize; //

char szRiffFormat[4]; // 'W','A','V','E'

};

struct **WAVE\_FORMAT**

{

WORD wFormatTag;

WORD wChannels;

DWORD dwSamplesPerSec;

DWORD dwAvgBytesPerSec;

WORD wBlockAlign;

WORD wBitsPerSample;

};

struct **FMT\_BLOCK**

{

char szFmtID[4]; // 'f','m','t',' '

DWORD dwFmtSize;

**WAVE\_FORMAT** wavFormat;

};

struct **FACT\_BLOCK**

{

char szFactID[4]; // 'f','a','c','t'

DWORD dwFactSize; // 4

};

typedef struct **WAVE\_DATA** {

char fccID[4];

unsigned long dwSize;

}**WAVE\_DATA**;

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 字段 | | | | 大小 | | 内容 |  | | |  | |
| RIFF\_HEAD | 辨识码 | | | | 4B | | “RIFF” |  | | |  | |
| 数据大小 | | | | 4B | | Size(RIFF\_DATA) |  | | |  | |
|  |  | | | |  | |  |  | | |  | |
| RIFF\_DATA | 格式辨识码 | | | | 4B | | “WAVE” |  | | |  | |
|  |  | |  | |  | |  |  | |  | |
| fmt  子块 | chunk\_name | | 4B | | “fmt” | |  |  | |  | |
| Size 子块数据大小 | | 4B | | Size(fmt\_data) value16 or 18 | |  | |  | |
| fmt\_data | FormatTag | | 2B | | 编码方式，  一般为0x0001(WAVE—FORMAT\_PCM) |  | |  | |  | |
| nChannels | | 2B | | 声道数目，1--单声道；2--双声道 |  | |  | |  | |
| SamplesPerSec | | 4B | | 采样频率 |  | |  | |  | |
| AvgBytesPerSec | | 4B | | 每秒所需字节数 ===> WAVE\_FORMAT |  | |  | |  | |
| BlockAlign | | 2B | | 数据块对齐单位(每个采样需要的字节数) |  | |  | |  | |
| BitsPerSample | | 2B | | 每个采样需要的bit数 |  | |  | |  | |
|  | | 2B | | 附加信息（可选，通过Size来判断有无 |  | |  | |  | |
|  | |  | |  |  | |  | |  | |
| 为了产生出能够正确读出的WAV文件,必须严格注意以下几个分量间的特定关系  否则产生出的文件将无法正常播放:  **nAvgBytesPerSec = nSamplesPerSec\*nChannels\*wBitsPerSamp／8**  **nBlockAlign = nChannels\*wBitsPerSample／8** | | | | |  | |  | |  | |
|  | |  | |  | |  |  | |  | |
|  |  | |  | |  | |  |  | |  | |
| Fact  子块 | Chunk name / ID | | 4B | | “fact” | |  |  | |  | |
| 子块数据大小 | | 4B | | Val = 4 | |  | |  | |
| Data | | 4B | |  | |  | |  | |
| Fact Chunk是可选字段,一般当wav文件由某些软件转化而成,则包含该Chunk | | | | | |  | |  | |  | |
|  |  | | | | | |  | |  | |  | |
|  |  | |  | |  | |  |  | |  | |
| Data子块 | Chunk name / ID | | 4B | |  | |  |  | |  | |
| 子块数据大小 | | 4B | | 这里的大小为 PCM 数据的大小 | |  | |  | |
|  | 子块数据 | |  | | PCM数据 | |  | |  | |

**二 视频**

模拟视频的数字化主要包括色彩空间的转换、光栅扫描的转换以及分辨率的统一。

1 常见接口

**HDMI：**传输的是数字信号，**非压缩**视频信号+音频信号，使用面广，高清电视上有



**DP（Display Port）**：传输的是数字信号**，非压缩**视频信号+音频信号

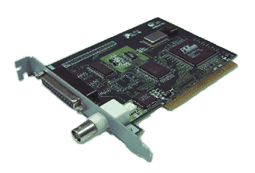
（和HDMI比有一个角是直角），使用面广，有些笔记本上就有



**SDI：**传输的是数字信号，**非压缩**视频信号（没有音频），传输距离远，用于广播电视领域



**ASI：**传输的是数字信号，**压缩**视频信号（例如MPEG2-TS（里面是H.264码流）），用于广播电视领域



**DVI：**传输的是数字信号，**非压缩**视频信号（没有音频），用于电脑显示器



**VGA：**传输的是模拟信号，比较老，用于电脑显示器



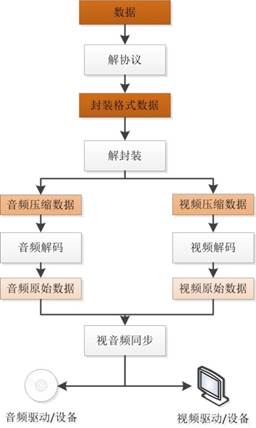
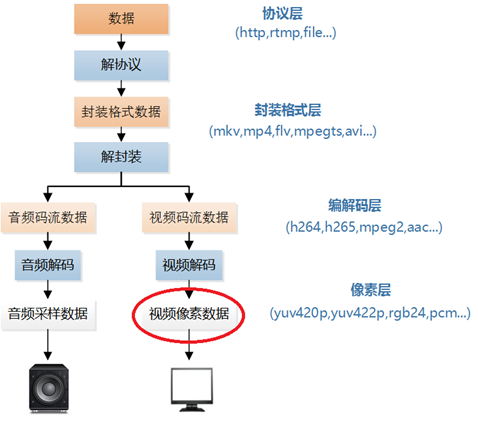
2 视频播放器原理

视音频技术主要包含以下几点：**封装技术**，**视频压缩编码技术**以及**音频压缩编码技术**。如果考虑到网络传输的话，还包括**流媒体协议技术**。

视频播放器播放一个互联网上的视频文件，需要经过以下几个步骤：**解协议**，**解封装**，**解码视音频**，**视音频同步**。

如果播放本地文件则不需要解协议，为以下几个步骤：**解封装**，**解码视音频**，**视音频同步**。

他们的过程如图所示：

**解协议**的作用，就是将**流媒体协议**的**数据**，**解析**为标准的相应的**封装格式数据**。

视音频在网络上传播的时候，常常采用各种**流媒体协议**，例如HTTP，RTMP，或是MMS等等。这些协议在传输视音频数据的同时，也会传输一些信令数据。这些信令数据包括对播放的控制（播放，暂停，停止），或者对网络状态的描述等。

解协议的过程中会去除掉信令数据而只保留视音频数据。

例如，采用RTMP协议传输的数据，经过解协议操作后，输出FLV格式的数据。

**解封装**的作用，就是将输入的**封装格式**的**数据**，**分离**成为**音频流压缩编码数据**和**视频流压缩编码数据**。

**封装格式**种类很多，例如MP4，MKV，RMVB，TS，FLV，AVI等等，它的作用就是将已经压缩编码的视频数据和音频数据按照一定的格式放到一起。

例如，FLV格式的数据，经过解封装操作后，输出H.264编码的视频码流和AAC编码的音频码流。

**解码**的作用，就是将**视频/音频压缩编码数据**，**解码**成为**非压缩的视频/音频原始数据**。

**音频**的**压缩编码标准**包含AAC，MP3，AC-3等等，**视频**的**压缩编码标准**则包含H.264，MPEG2，VC-1等等。

解码是整个系统中最重要也是最复杂的一个环节。通过解码，压缩编码的视频数据输出成为**非压缩的颜色数据**，例如YUV420P，RGB等等；压缩编码的音频数据输出成为**非压缩的音频抽样数据**，例如PCM数据。

**视音频同步**的作用，就是**根据解封装模块处理**过程中获取到的**参数信息**，**同步解码**出来的**视频和音频数据**，并将视频音频数据送至系统的显卡和声卡播放出来。

3 流媒体协议

**流媒体协议**是服务器与客户端之间**通信**遵循的**规定**。

当前网络上主要的流媒体协议如表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 推出机构 | 传输层协议 | 客户端 | 适用领域 |
| RTSP+RTP | IETF | TCP+UDP | VLC . WMP | IPTV |
| RTMP | Adobe Inc. | TCP | Flash | 互联网直播 |
| RTMFP | Adobe Inc. | UDP | Flash | 互联网直播 |
| MMS | Microsoft Inc. | TCP/UDP | WMP | 互联网直播+点播 |
| HTTP | WWW+IETF | TCP | Flash | 互联网点播 |

**RTSP+RTP**经常用于IPTV领域。

因为其采用UDP传输视音频，支持组播，效率较高。

但其缺点是网络不好的情况下可能会丢包，影响视频观看质量。

因为互联网网络环境的不稳定性，RTSP+RTP较少用于互联网视音频传输。

互联网视频服务通常采用TCP作为其流媒体的传输层协议，因而像RTMP，MMS，HTTP这类的协议广泛用于互联网视音频服务之中。这类协议不会发生丢包，因而保证了视频的质量，但是传输的效率会相对低一些。此外**RTMFP**是一种比较新的流媒体协议，特点是**支持P2P**。

4 视频文件格式/封装格式

**视频文件格式**是指视频保存的一种格式，视频是现在电脑中多媒体系统中的重要一环。

为了适应储存视频的需要，人们设定了不同的**视频文件格式**来把视频和音频**放在**一个文件中，以方便同时回放。

视频文件格式有不同的分类，如：

微软视频 ：wmv、asf、asx

Real Player ：rm、 rmvb

MPEG视频 ：mp4

手机视频 ：3gp

Apple视频 ：mov、m4v

其他常见视频：avi、dat、mkv、flv、vob

|  |  |
| --- | --- |
| 视频文件格式 | 视频文件后缀 |
| AVI（Audio Video Interleaev） | .avi |
| WMV ( Windows Media Video) | .wmv .asf .asx |
| MPEG(Mocing Picture Experts Group) | ,mp4 .mpg .mpeg .vob .dat .3gp |
| Martoska | .mkv |
| Real Video (RealNetworks) | .rm .rmvb |
| QuickTime File Format (Apple) | .mov |
| Flash Video | .flv |

**封装格式**的主要**作用**是把视频码流和音频码流按照一定的格式存储在一个文件中。

现如今流行的封装格式如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 推出机构 | 流媒体 | 支持的视频编码 | 支持的音频编码 | 使用领域 |
| **AVI** | Microsoft Inc. | 不支持 | 几乎所有的编码 | 几乎所有的编码 | BT下载影视 |
| **MP4** | MPEG | 支持 | MPEG-2, MPEG-4, H.264, H.263 | AAC, MPEG-1 Layers I, II, III, AC-3 | 互联网视频网站 |
| **TS** | MPEG | 支持 | MPEG-1, MPEG-2, MPEG-4, H.264 | MPEG-1 Layers I, II, III, AAC, | IPTV，数字电视 |
| **FLV** | Adobe Inc. | 支持 | Sorenson, VP6, H.264 | MP3, ADPCM, Linear PCM, AAC | 互联网视频网站 |
| **MKV** | CoreCodec Inc. | 支持 | 几乎所有的编码 | 几乎所有的编码 | 互联网视频网站 |
| **RMV8** | Real Networks Inc. | 支持 | RealVideo 8, 9, 10 | AAC, Cook Codec, RealAudio Lossless | BT下载影视 |

由表可见，除了AVI之外，其他封装格式都支持**流媒体**，即可以“**边下边播**”。有些格式更“万能”一些，支持的视音频编码标准多一些，比如MKV。而有些格式则支持的相对比较少，比如说RMVB。

6 视频压缩编码标准

**视频编码**的主要作用是将**视频像素数据**（RGB，YUV等）**压缩**成为**视频码流**，从而**降低**视频的**数据量**。

如果视频不经过压缩编码的话，体积通常是非常大的，一部电影可能就要上百G的空间。 **视频编码**是视音频技术中最重要的技术之一。视频码流的数据量占了视音频总数据量的绝大部分。高效率的视频编码在同等的码率下，可以获得更高的视频质量。

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 推出机构 | 推出时间 | 适用领域 |
| HEVC(H.265) | MPEG/ITU-T | 2013 |  |
| H.264 | MPEG/ITU-T | 2003 | 各个领域 |
| MPEG4 | MPEG | 2001 |  |
| MPEG2 | MPEG | 1994 | 数字电视 |
| VP9 | Google | 2013 |  |
| VP8 | Google | 2008 | 不普及 |
| VC-1 | Microsoft inc. | 2006 | 微软平台 |

HEVC > VP9 > H.264> VP8 > MPEG4 > H.263 > MPEG2。

**· MPEG-1**

类型：Audio&Video

制定者：MPEG(Moving Picture Expert Group)

所需频宽：2Mbps

特性：对动作不激烈的视频信号可获得较好的图像质量，但当动作激烈时，图像就会产生马赛克现象。它没有定义用于额外数据流进行编对码的格式，因此这种技术不能广泛推广。它主要用于家用VCD，它需要的存储空间比较大。

优点：对动作不激烈的视频信号可获得较好的图像质量。

缺点：当动作激烈时，图像就会产生马赛克现象。它没有定义用于额外数据流进行编对码的格式，因此这种技术不能广泛推广。

应用领域：Mixer

版权方式：Free

备注：MPEG-1即俗称的VCD。MPEG是ISO/IEC JTC1 1988年成立的运动图像专家组（Moving Picture Expert Group）的简称，负责数字视频、音频和其他媒体的压缩、解压缩、处理和表示等国际技术标准的制定工作。MPEG-1制定于1992年，它是将视频数据压缩成1~2Mb/s的标准数据流。对于清晰度为352×288的彩色画面，采用25帧/秒，压缩比为50：1时，实时录像一个小时，经计算可知需存储空间为600MB左右，若是8路图像以每天录像10小时，每月30天算，则要求硬盘存储容量为1440GB，则显然是不能被接受的。

**· MPEG-2**

类型：Audio&Video

制定者：MPEG(Moving Picture Expert Group)

所需频宽：视频上4.3Mbps，音频上最低的采样率为16kHz

特性：编码码率从每秒3兆比特～100兆比特，是广播级质量的图像压缩标准，并具有CD级的音质。MPEG-2的音频编码可提供左、右、中及两个环绕声道，以及一个加重低音声道，和多达7个伴音声道。作为MPEG-1的兼容性扩展，MPEG-2支持隔行扫描视频格式和其它先进功能，可广泛应用在各种速率和各种分辨率的场合。但是MPEG-2标准数据量依然很大，不便存放和传输。

优点：MPEG-2的音频编码可提供左、右、中及两个环绕声道，以及一个加重低音声道，和多达7个伴音声道，具有CD级的音质。可提供一个较广的范围改变压缩比，以适应不同画面质量、存储容量以及带宽的要求。支持隔行扫描视频格式和其它先进功能，可广泛应用在各种速率和各种分辨率的场合。

缺点：压缩比较低，数据量依然很大，不便存放和传输，如用于网络方面则需要较高的网络带宽，因此不太适合用于Internet和VOD点播方面。

应用领域：Mixer

版税方式：按个收取(最初的收费对象为解码设备和编码设备，中国DVD制造商每生产一台DVD需要交纳专利费16.5美元。向解码设备和编码设备收取的专利授权费每台2.5美元)

备注：MPEG-2是其颁布的（活动图像及声音编码）国际标准之一，制定于1994年，是为高级工业标准的图像质量以及更高的传输率而设计，为了力争获得更高的分辨率（720×486），提供广播级视频和CD级的音频，它是高质量视频音频编码标准。在常规电视的数字化、高清晰电视 HDTV、视频点播VOD，交互式电视等各个领域中都是核心的技术之一。

由于MPEG-2在设计时的巧妙处理，使得大多数MPEG-2解码器也可播放 MPEG-1格式的数据，如VCD。

MPEG-2的音频编码可提供左、右、中及两个环绕声道，以及一个加重低音声道，和多达7个伴音声道。我们平时所说的 DVD就是采用MPEG-2编码压缩，所以可有8种语言的配音。

除了作为DVD的指定标准外，MPEG-2的应用前景非常的广阔，MPEG-2还可用于广播、有线电视网、电缆网络以及卫星直播 (Direct Broadcast Satellite) 提供广播级的数字视频。

MPEG-2的另一特点是可提供一个较广的范围改变压缩比，以适应不同画面质量、存储容量以及带宽的要求。对于最终用户来说，由于现存电视机分辨率限制，MPEG-2所带来的高清晰度画面质量在电视上效果并不明显，不过其音频特性非常引人注目，如加重低音，多伴音声道等。

**· MPEG-4**

类型：Video

制定者：MPEG(Moving Picture Expert Group)

所需频宽：128Kbps~38.4Mbps(600kb/s左右)

特性：支持对象型态编码及合成图像的压缩、适用于高阶交互功能与特殊视频制作、容错性编码技术及细微式可调性编码技术，可适用于频宽变化剧烈的网络，更适于交互AV服务以及远程监控。

MPEG-4是第一个使你由被动变为主动(不再只是观看，允许你加入其中，即有交互性)的动态图象标准；它的另一个特点是其综合性；从根源上说，MPEG-4试图将自然物体与人造物体相溶合(视觉效果意义上的)。 MPEG-4的设计目标还有更广的适应性和可扩展性。MPEG4 试图达到两个目标：

1.低比特率下的多媒体通信；

2.是多工业的多媒体通信的综合。据此目标，MPEG4 引入AV 对象（Audio/Visaul Objects），使得更多的交互操作成为可能。

MPEG-4标准是面向对象的压缩方式，根据图像内容，将其中的对象（物体、人物、背景）分离出来分别进行帧内、帧间编码压缩，并允许在不同的对象之间灵活分配码率，对重要的对象分配较多的字节，对次要的对象分配较少的字节，从而大大提高了压缩比，使其在较低的码率下获得较好的效果。

优点：压缩率高，质量优，容错性好，视频质量分辨率比较高，而数据速率相对较低，采用面向对象的压缩方式。

缺点：专利收费不合理。

应用领域：Mixer

版税方式：按个收取(分别向运营商和终端用户收费,消费者使用解码设备，除购买设备时需要缴纳的一次性专利费外，还将按使用时间进行收费)，每台解码设备需要交给MPEG-LA 0.25美元, 编码/解码设备还需要按时间交费（4美分/天=1.2美元/月=14.4美元/年）

备注： MPEG-4是为交互式多媒体通讯制定得压缩标准。MPEG4于1998 年11 月公布，原预计1999 年1月投入使用的国际标准MPEG4不仅是针对一定比特率下的视频、音频编码，更加注重多媒体系统的交互性和灵活性。MPEG专家组的专家们正在为 MPEG-4的制定努力工作。

MPEG-4标准主要应用于视像电话(Video Phone)，视像电子邮件(Video Email)和电子新闻(Electronic News)等，其传输速率要求较低，在4800-6?000bits/sec之间，分辨率为176X144。MPEG-4利用很窄的带宽，通过帧重建技术，压缩和传输数据，以求得最少的数据获得最佳的图象质量。

MPEG-4的视频质量分辨率比较高，而数据速率相对较低。主要原因在于，MPEG-4采用ACE（高级译码效率）技术，它是一套首次使用于 MPEG-4的编码运算规则。

与ACE有关的目标定向可以启用很低的数据率。它与MPEG-2相比，可节省90%的储存空间。MPEG-4还可以在声频与视频流中广泛的升级。当视频在5kb/s与10Mb/s之间变化时，声频信号可以在2kb/s与24kb/s之间进行处理。特别要强调的是MPEG-4标准是面向对象的压缩方式，不是像MPEG-1和MPEG-2简单地将图像分为一些像块，而是根据图像内容，将其中的对象（物体、人物、背景）分离出来分别进行帧内、帧间编码压缩，并允许在不同的对象之间灵活分配码率，对重要的对象分配较多的字节，对次要的对象分配较少的字节，从而大大提高了压缩比，使其在较低的码率下获得较好的效果。

MPEG-4的面向对象的压缩方式也使图像探测功能和准确性更充分体现，该图像探测功能使硬盘录像机系统具有较好的视频移动报警功能。总之MPEG-4是一种崭新的低码率、高压缩比的视频编码标准，传输速率为4.8~6?kbit/s，使用时占用的存储空间比较小，例如：对于清晰度352×288的彩色画面，其每帧占用空间为1.3KB时，选25帧/秒，则每小时需120KB、每天10小时、每月30天，则每路每月需 36GB。若是8路则需288GB，这显然是能接受的。

**· MPEG-7**

（Multimedia Content Description Interface，多媒体内容描述接口)

类型：Video

制定者：MPEG(Moving Picture Expert Group)

所需频宽：-

特性：MPEG并不对应用标准化，但可利用应用来理解需求并评价技术，它不针对特定的应用领域，而是支持尽可能广泛的应用领域。 MPEG-7是针对存储形式(在线、脱机)或流形式(如 Internet上的广播、推送模型)的应用而制定的，并且可以在实时和非实时环境中操作。一个实时环境意味着当采集资料时，信息是与内容相关的。像其他 MPEG家族成员一样，MPEG-7是满足特定需求的视听信息的标准表示。MPEG-7建立在其他标准表示的基础之上，例如PCM、 MPEG-1、MPEG-2和MPEG-4。因此，MPEG-7会引用部分现有标准。但MPEG-7描述子将不依赖于被描述内容的编码和存储方式。 MPEG-7可以独立于其他MPEG标准使用，在MPEG-4 中定义的表示方式也非常适合MPEG-7标准的建立。

优点：广泛的多媒体运用，可以在存储形式、流形式，实时或非实时中运用。

缺点：-

应用领域：Mixer

版税方式：-

备注：MPEG-7是为互联网视频检索制定的压缩标准。

国际标准化组织（ISO）在制定MPEG-1、MPEG-2及MPEG-4的标准基础上，推出了新的标准MPEG-7，该标准的正式名称为“多媒体内容描述接口”（Multimedia Content Description Interface）, 其目标就是产生一种描述多媒体内容数据的标准，满足实时、非实时以及推-拉应用的需求,它既不同于基于波形和基于压缩的表示方式如MPEG-1和MPEG -2,又不同于基于对象的表示方式如MPEG-4，而是将对各种不同类型的多媒体信息进行标准化描述，并将该描述与所描述的内容相联系，以实现快速有效的搜索。

MPEG-7将扩展现有标识内容的专用方案及有限的能力，包含更多的多媒体数据类型。换句话说，它将规范一组“描述子”，用于描述各种多媒体信息，也将对定义其他描述子以及结构(称为“描述模式”)的方法进行标准化。这些“描述”(包括描述子和描述模式)与其内容关联，允许快速有效地搜索用户感兴趣的资料。

MPEG-7将标准化一种语言来说明描述模式，即“描述定义语言”。带有MPEG-7数据的 AV资料可以包含静止图像、图形、3D模型、音频、语音、视频，以及这些元素如何在多媒体表现中组合的信息。这些通用数据类型的特例可以包含面部表情和个人化特性。

MPEG-7的功能与其他MPEG标准互为补充。MPEG-1、 MPEG-2和MPEG-4是内容本身的表示，而MPEG-7是有关内容的信息，是比特的比特。

**· MPEG-21 Multimedia Framework**

类型：Video

制定者：MPEG(Moving Picture Expert Group)

所需频宽：

特性：MPEG－21 Multimedia Framework是致力于在大范围的网络上实现透明的传输和对多媒体资源的充分利用。MPEG－21致力于为多媒体传输和使用定义一个标准化的开放框架。这种框架将在开放的市场中为内容提供商和业务提供商创造同等的机会。同时，这将在一种互操作的模式下为用户提供更丰富的信息，用户将因此而受益。

MPEG－21景象可以总结如下：一个多媒体框架，它可以在广阔的范围里，为不同的网络用户提供透明的和可不断扩展的多媒体资源。 MPEG－21基于两个基本概念：分布和处理基本单元DI（the Digital Item）以及DI与用户间的互操作。 MPEG－21也可表述为：以一种高效、透明和可互操作的方式支持用户交换、接入、使用甚至操作DI的技术。

优点：

①将不同的协议、标准、技术等有机地融合在一起；

②制定新的标准；

③将这些不同的标准集成在一起。

MPEG-21标准其实就是一些关键技术的集成，通过这种集成环境就对全球数字媒体资源进行透明和增强管理，实现内容描述、创建、发布、使用、识别、收费管理、产权保护、用户隐私权保护、终端和网络资源抽取、事件报告等功能。

缺点：-

应用领域：Mixer

版税方式：-

备注：MPEG（ISO/IEC JTCI SC29 WGII）从2000年6月开始着手定义21世纪多媒体应用的标准化技术：MPEG－21 “Multimedia Framework”。MPEG－21是一个可互操作和高度自动化的框架，而且这个框架还考虑到了DRM（digital rights management）的要求、对象化的多媒体接入以及使用不同网络和终端进行传输等问题。DI是MPEG－21框架中，一个具有标准表示、身份认证和相关元数据的数字对象。这个实体是框架中分布和处理的基本单元。在MPEG－21中，一个用户是指与MPEG－21进行环境交互或者使用DI的任何实体。这些用户包括个人、消费者、社团、组织、公司和政府部门。从单纯技术的角度来说，MPEG－21认为“内容提供商”和“使用者（consumer）”之间没有分别——他们都是用户。一个单独的实体可以以几种方式使用网络的内容，同时所有这些与MPEG－21交互的实体都被平等对待。然而，一个用户可以根据与之交互的其他用户的不同来承担特定的角色，发挥不同的作用。在最基本的层次上，MPEG－21可以被看成是提供用户间交互的一个框架。

**· WMV（WINDOWS MEDIA VIDEO）**

类型：Video

制定者：微软公司

所需频宽：128Kbps~38.4Mbps(600kb/s左右)

特性：一种流媒体格式，WMV格式的体积非常小，适合在网上播放和传输。

优点：在同种视频质量的条件下，WMV的文件非常小

缺点：非开放性标准，时延非常大。

应用领域：media

版税方式：按个收费

备注：WMV是微软推出的一种流媒体格式，它是在“同门”的ASF（Advanced Stream Format）格式升级延伸来得。在同等视频质量下，WMV格式的体积非常小，因此很适合在网上播放和传输。由于微软本身的局限性其WMV的应用发展并不顺利。第一, WM9是微软的产品它必定要依赖着Windows，Windows 意味着解码部分也要有PC, 起码要有PC机的主板。这就大大增加了机顶盒的造价，从而影响了视频广播点播的普及。第二，WMV技术的视频传输延迟非常大，通常要10几秒钟，正是由于这种局限性，目前WMV也仅限于在计算机上浏览WM9视频文件。

**· H.261**

类型：Video

制定者：CCITT(即以后的ITU-T)

所需频宽：6?kbps至1.92Mbps

特性：最初是针对在ISDN上实现电信会议应用特别是面对面的可视电话和视频会议而设计的。实际的编码算法类似于MPEG算法，但不能与后者兼容。H.261在实时编码时比MPEG所占用的CPU运算量少得多，此算法为了优化带宽占用量，引进了在图像质量与运动幅度之间的平衡折中机制，也就是说，剧烈运动的图像比相对静止的图像质量要差。因此这种方法是属于恒定码流可变质量编码而非恒定质量可变码流编码。

优点：在实时编码时比MPEG所占用的CPU运算量少得多。

缺点：剧烈运动的图像比相对静止的图像质量要差

应用领域：media

版权方式：Free

备注：于1990年完成和批准了CCITT推荐书 H.261，用于电视会议、可视电话。

**· H.263**

类型：Video

制定者：ITU-T

所需频宽：低达20K到24Kbps带宽

特性：灵活性、节省带宽和存储空间、安装方便、可方便的进行二次开发。H.263与H.261相比采用了半象素的运动补偿，并增加了4种有效的压缩编码模式。

优点：能提供更好的图像质量、更低的速率、安装方便、可方便的进行二次开发。H.263与H.261相比采用了半象素的运动补偿，并增加了4种有效的压缩编码模式。

缺点：限制了其应用的图像输入格式，仅允许5种视频源格式。

应用领域：media(IP视频通信方面)

版税方式：按个收取

备注：1996年ITU-T完成了H.263编码标准。H.263使用户可以扩展带宽利用率，可以低达128Kbps的速率实现全运动视频(每秒30 帧)。H.263以其灵活性以及节省带宽和存储空间的特性，具有低总拥有成本并提供了迅速的投资回报。H.263是为以低达20K到 24Kbps带宽传送视频流而开发的，基于H.261编解码器来实现。但是，原则上它只需要一半的带宽就可取得与H.261同样的视频质量。

**· H.263+**

类型：Video

制定者：ITU-T

所需频宽：低达20K到24Kbps带宽

特性：允许更大范围的图像输入格式，自定义图像的尺寸，从而拓宽了标准使用的范围，使之可以处理基于视窗的计算机图像、更高帧频的图像序列及宽屏图像。采用先进的帧内编码模式；增强的PB-帧模式改进了H.263的不足，增强了帧间预测的效果；去块效应滤波器不仅提高了压缩效率，而且提供重建图像的主观质量。增加了时间分级、信噪比和空间分级，对在噪声信道和存在大量包丢失的网络中传送视频信号很有意义；另外，片结构模式、参考帧选择模式增强了视频传输的抗误码能力。

优点：允许更大范围的图像输入格式，增强的PB-帧模式，增强了帧间预测的效果，去块效应滤波器不仅提高了压缩效率，增加了时间分级、信噪比和空间分级，片结构模式、参考帧选择模式。

缺点：

应用领域：media

版税方式：按个收取

备注：ITU-T在H.263发布后又修订发布了H.263标准的版本2，非正式地命名为H.263+标准。它在保证原H.263标准核心句法和语义不变的基础上，增加了若干选项以提高压缩效率或改善某方面的功能。

**· H.263++**

类型：Video

制定者：ITU-T

所需频宽：低达20K到24Kbps带宽

特性：H263++在H263+基础上增加了3个选项，主要是为了增强码流在恶劣信道上的抗误码性能，同时为了提高增强编码效率。

优点：提高了抗误码性能，增强编码效率。

缺点：

应用领域：media

版税方式：按个收取

备注：H263++在H263+基础上增加了3个选项，主要是为了增强码流在恶劣信道上的抗误码性能，同时为了提高增强编码效率。这3个选项为：

①选项U——称为增强型参考帧选择，它能够提供增强的编码效率和信道错误再生能力(特别是在包丢失的情形下)，需要设计多缓冲区用于存贮多参考帧图像；

②选项V——称为数据分片，它能够提供增强型的抗误码能力(特别是在传输过程中本地数据被破坏的情况下)，通过分离视频码流中DCT的系数头和运动矢量数据，采用可逆编码方式保护运动矢量；

③选项W——在H263+的码流中增加补充信息，保证增强型的反向兼容性，附加信息包括：指示采用的定点IDCT、图像信息和信息类型、任意的二进制数据、文本、重复的图像头、交替的场指示、稀疏的参考帧识别。

**· H.264／MPEG-4 AVC**

类型：Video

制定者：ITU-T VCEG 和ISO/IEC MPEG

所需频宽：1Mbps的频宽

特性：H264标准使运动图像压缩技术上升到了一个更高的阶段，H.264压缩能力比H.263更强（在相同的重建图像质量下，H.264比 H.263+和MPEG-4(SP)减小50%码率），但相对H.263有更惊人的运算量，连带影响省电、散热等议题。在较低带宽上提供高质量的图像传输是H.264应用亮点。

H.264推广应用对视频终端、网守、网关、MCU等系统的要求较高，将有力地推动视频会议软、硬件设备在各个方面的不断完善。但是现在的处理器还没有足够的运算能力处理H.264影片，象Athlon 6? FX、双核心的Athlon 6? X2、Pentium 4甚至双核心Pentium D都力不从心。同时，要播放H.264影片，需要有强劲的显卡。ATI即推出的R520是目前唯一能搞定这一高难度任务的显卡。而NVIDIA的G70、 Geforce 7800 GTX都无法达到播放H.264影片的要求。

优点：高压缩比、高图像质量、良好的网络适应性，在较低带宽上提供高质量的图像传输。

①在相同的重建图像质量下，H.264比H.263+和MPEG-4(SP)减小50%码率；

②对信道时延的适应性较强，既可工作于低时延模式以满足实时业务，如会议电视等；又可工作于无时延限制的场合，如视频存储等；

③提高网络适应性，采用“网络友好”的结构和语法，加强对误码和丢包的处理，提高解码器的差错恢复能力；

④在编/解码器中采用复杂度可分级设计，在图像质量和编码处理之间可分级，以适应不同复杂度的应用；

⑤相对于先期的视频压缩标准，H.264引入了很多先进的技术，包括4×4整数变换、空域内的帧内预测、1/4象素精度的运动估计、多参考帧与多种大小块的帧间预测技术等。新技术带来了较高的压缩比，同时大大提高了算法的复杂度。

缺点：对视频终端、网守、网关、MCU等系统的要求较高，在小型手持装置市场的发挥空间，恐怕不如H.263。

应用领域：media

备注：H.264是为新一代交互视频通讯制定的标准。该标准也被称为AVC（Advanced Video Coding）标准，是MPEG-4的第10部分。

H264码流详细说明

h264是类似于mpeg4的一种压缩算法，只不过h264更加高级，压缩率更高，使用更方便、更灵活。h264具有这些优点得益于他的设计。

h264是分层设计，分为**VCL层**和**NAL层:**

其中:

**VCL（视频编码层）**是为了表示视频数据的内容，

**NAL（网络提取层）**层则是为了格式化数据，方便存取。

VCL输出的是原始数据比特流（SODB，String of data bits），表示H.264的语法元素编码完成后的实际的原始二进制码流。

SODB通常不能保证字节对齐,故需要补齐为原始字节序列负荷（RBSP，Raw Byte Sequence Payload）。

**RBSP = SODB + RBSP trailing bits**

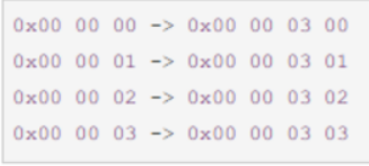
**NALU = NAL header(1 byte) + RBSP**

**H.264 = Start Code Prefix(3 bytes) + NALU + Start Code Prefix(3 bytes) + NALU +…**

起始码

每个NALU之**间**由起始码（Start Code Prefix）分隔，起始码分为两种：0x000001(3 bytes) or 0x00000001(4 bytes).

如果NALU 对应的Slice(切片) 为一帧的开始，则用4 字节表示，即0x00000001；否则用3 字节表示，0x000001.NALU针对起始码设计了防止冲突机制，如果出现连续的0x000000，0x000001,0x000002,0x000003时，会在两个0之间插入03，如下：



一个NAL单元如下：

|  |  |  |  |
| --- | --- | --- | --- |
| Header(1Byte) | | | Body(NByte) |
| Forbidden\_zero\_bit(1bit) | Nal\_ref\_idc(2bit) | Nal\_unit\_type(5bit) | RBSP |

前一个字节为NAL头，头之后是NAL体（RBSP部分），即压缩后的视频数据。

**RBSP**(raw byte sequence payload):原始字节序列载荷：一个语法结构，包含整数个封装于NAL单元中的字节。

For\_bidden\_zero\_bit (禁止位)

禁止位在编码中默认值为0，当网络识别此单元中存在比特错误时，可将其设为1，以便接收方丢掉该单元，主要用于适应不同种类的网络环境（比如有线无线相结合的环境）。

Nal\_ref\_idc

取 00 ~ 11, 指示这个 NALU 的重要性, 用于在重构过程中标记一个NAL单元的重要性，值越大，越重要。值为0表示这个NAL单元没有用于预测，因此可被解码器抛弃而不会有错误扩散；值高于0表示此NAL单元要用于无漂移重构，且值越高，对此NAL单元丢失的影响越大。

**不等于0**时，规定NAL单元的内容包含一个**序列参数集**，或一个**图像参数集**，或一个**参考图像片**，或一个**参考图像片的数据分割块**。

如果**一个NAL单元(**包含 一个片 或 片分区) 的nal\_ref\_idc 等于 0 时，该 片 或者 片分区 是 一个 非参考图像 的一部分。

对于序列参数集 或者 序列参数集扩展 或者 图像参数集 的NAL单元，nal\_ref\_idc不应等于0.应该为11

IDR NAL单元的 nal\_ref\_idc 不应等于0，即nal\_unit\_type 等于5的时候。

**所有nal\_unit\_type 等于6，9，10，11，12的NAL单元其nal\_ref\_idc都应等于0**

Nal\_unit\_type是指包含在NAL单元中的RBSP数据结构的类型。

VCL NAL 单元是指那些nal\_unit\_type值等于1到5（包含1，5）的NAL单元。所有其他的NAL单元都称作非VCL NAL单元

对于每个NAL中body是什么数据的内容，通过header中nal\_unit\_type(5bit)决定，详见下表：

|  |  |  |
| --- | --- | --- |
| Nal\_unit\_type | NAL单元和RBSP语法结构的内容 | C |
| 0（0 0000 B） | 未指定 |  |
| 1（0 0001 B） | 非IDR图像的片 slice\_layer\_without\_partitioning\_rbsp() | 2,3,4 |
| 2（0 0010 B） | 片数据分割块A slice\_data\_partition\_a\_layer\_rbsp() | 2 |
| 3（0 0011 B） | 片数据分割块B slice\_data\_partition\_b\_layer\_rbsp() | 3 |
| 4（0 0100 B） | 片数据分割块C slice\_data\_partition\_c\_layer\_rbsp() | 4 |
| 5（0 0101 B） | IDR图像的片 slice\_layer\_without\_partitioning\_rbsp() | 2，3 |
| 6（0 0110 B） | 辅助增强信息(SEI) sei\_rbsp() | 5 |
| 7（0 0111 B） | 序列参数集(SPS) seq\_parameter\_set\_rbsp() | 0 |
| 8（0 1000 B） | 图像参数集(PPS) pic\_parameter\_set\_rbsp() | 1 |
| 9（0 1001 B） | 访问单元分隔符 access\_unit\_delimiter\_rbsp() | 6 |
| 10（0 1010 B） | 序列结尾 end\_of\_seq\_rbsp() | 7 |
| 11（0 1011 B） | 流结尾 end\_of\_stream\_rbsp() | 8 |
| 12（0 1100 B） | 填充数据 filler\_data\_rbsp() | 9 |
| 13（0 1101 B） | 序列参数集扩展 seq\_parameter\_set\_extension\_rbsp() | 10 |
| 14~18 | 保留 |  |
| 19（1 0011 B） | 未分割的辅助编码图像的编码条带  slice\_layer\_without\_partitioning\_rbsp() | 2,3,4 |
| 20~23 | 保留 |  |
| 24~31 | 未指定 |  |

这用来标识NAL单元中的RBSP数据类型，其中，nal\_unit\_type为1， 2， 3， 4， 5的NAL单元称为VCL的NAL单元，其他类型的NAL单元为非VCL的NAL单元

一个h264文件是由NAL序列组成的，为了区分单个的NAL，在生成NAL序列中，往**每个**NAL单元前添加一个**起始编码前缀**'**0x00 00 00 01**'。

**· AVS(数字音视频编解码技术标准/MPEG中国代表团)**

类型：Audio&Video

制定者：数字音频编解码技术标准工作组

所需频宽：码率覆盖几十kbps的低带宽通信到数十Mbps的高清晰度电视广播特性：适应面十分广阔，包括数字电视、激光视盘、网络流媒体、无线流媒体、数字音频广播、视频监控等等领域。码率覆盖几十 kbps的低带宽通信到数十Mbps的高清晰度电视广播，可以支持低延迟模式的视频会议应用，也支持高压缩效率的视频存储应用等。与MPEG-2等现有压缩标准相比，在相同的视觉质量下，码率至少可以降低50%。编码效率比传统的MPEG-2国际标准提高近3倍。

优点：编码效率高、实现复杂度低、专利收费合理

缺点：实现上有一定难度，要实现由国家标准转为国际标准还需要时间。

应用领域：Mixer

版税方式：按个收取(每台终端1元人民币)

备注：属于信源编码技术，和信道编码及显示技术一起构成数字电视的技术体系，用以解决海量音视频数据的压缩问题。AVS是一个最基础的标准，不仅对数字电视产业至关重要，也广泛应用于激光视盘机、多媒体通信、互联网流媒体等数字音视频产业，它的编码效率比传统的MPEG-2国际标准提高近3 倍。

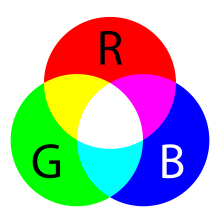
7 视频(图像)颜色编码

对一种颜色进行编码的方法统称为“**颜色空间**”或“**色域**”。

用最简单的话说，世界上任何一种颜色的“颜色空间”都可定义成一个固定的数字或变量。

**· RGB**

对RGB，并不陌生，从初中开始接触的**色光**的**三原色**，告诉我们我们可以看到的光可以由这三种颜色按一定的比例去混合得到；后来在HTML以及Android开发中设置元素/控件的颜色时，可以通过一串数字，得到某个特定的颜色。这就是RGB的应用。

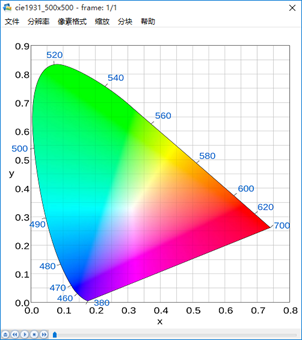


RGB 模型是目前常用的一种彩色信息表达方式，它使用红、绿、蓝三原色的亮度来定量表示颜色。该模型也称为加色混色模型，是以RGB三色光互相叠加来实现混色的方法，因而适合于显示器等发光体的显示。

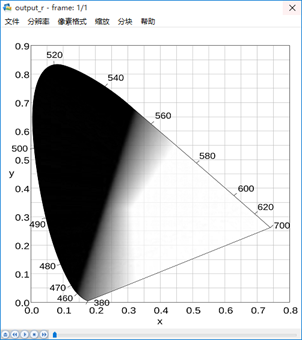
RGB 颜色模型可以看做三维直角坐标颜色系统中的一个单位正方体。任何一种颜色在RGB 颜色空间中都可以用三维空间中的一个点来表示。

在**RGB 颜色空间**上，当**三种基色**的**亮度值**为**零**时，即在原点处，就显示为**黑色**。当**三种基色**都达到**最高亮度**时，就表现为**白色**。在**连接黑色与白色**的**对角线**上，是**亮度等量**的三基色混合而成的**灰色**，该线称为**灰色线**。

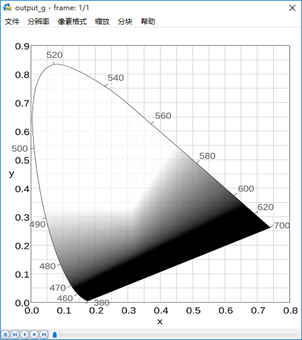
一张标准的CIE 1931色度图。该色度图右下为红色，上方为绿色，左下为蓝色，如下所示。



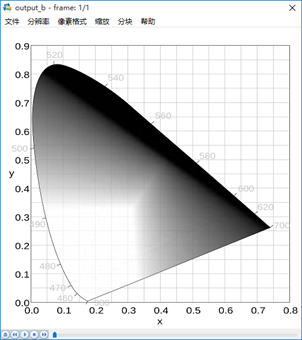
R数据图像如下所示。



G数据图像如下所示。



B数据图像如下所示。



什么是BGR：与RGB类似，只是存储时B位与R位的位置进行调换。

**·RGB格式**

RGB（红、绿、蓝）只是众多颜色空间的一种。采用这种编码方法，每种颜色都可用三个变量来表示-红色绿色以及蓝色的强度。记录及显示彩色图像时，RGB是最常见的一种方案。但是，它缺乏与早期黑白显示系统的良好兼容性。因此，许多电子电器厂商普遍采用的做法是，将RGB转换成YUV颜色空间，以维持兼容，再根据需要换回RGB格式，以便在电脑显示器上显示彩色图形。

**··网页格式**

由于网页(WEB)是基于计算机浏览器开发的媒体，所以颜色以光学颜色RGB（红、绿、蓝）为主。

**网页颜色**是以**16进制代码**表示，一般格式为**#DEFABC**（字母范围从A-F,数字从0-9 ）；如黑色，在网页代码中便是：#000000(在css编写中可简写为#000)。

当颜色代码为#AABB11时，可以简写为#AB1表示，如#135与#113355表示同样的颜色。

**RGB1**、**RGB4**、**RGB8**都是调色板类型的**RGB格式**，在描述这些媒体类型的格式细节时，通常会在BITMAPINFOHEADER数据结构后面跟着一个调色板（定义一系列颜色）。它们的图像数据并不是真正的颜色值，而是当前像素颜色值在调色板中的索引。

以RGB1（2色位图）为例，比如它的调色板中定义的两种颜色值依次为0x000000（黑色）和0xFFFFFF（白色）…（每个像素用1位表示）表示对应各像素的颜色为：黑黑白白黑白黑白黑白白白…。

**··RGB555**

RGB555是另一种**16位**的RGB格式，RGB分量都用5位表示（剩下的1位不用）。使用一个字读出一个像素后，这个字的各个位意义如下：

高字节 低字节

X R R R R R G G G G G B B B B B （X表示不用，可以忽略）

可以组合使用屏蔽字和移位操作来得到RGB各分量的值：

#define RGB555\_MASK\_RED 0x7C00

#define RGB555\_MASK\_GREEN 0x03E0

#define RGB555\_MASK\_BLUE 0x001F

R = (wPixel & RGB555\_MASK\_RED) >> 10;  // 取值范围0-31

G = (wPixel & RGB555\_MASK\_GREEN) >> 5;  // 取值范围0-31

B = wPixel & RGB555\_MASK\_BLUE;  // 取值范围0-31

**··RGB565**

RGB565使用16位表示一个像素，这16位中的5位用于R，6位用于G，5位用于B。程序中通常使用一个字（WORD，一个字等于两个字节）来操作一个像素。当读出一个像素后，这个字的各个位意义如下：

高字节 低字节

R R R R R G G G G G G B B B B B

可以组合使用屏蔽字和移位操作来得到RGB各分量的值：

#define RGB565\_MASK\_RED 0xF800

#define RGB565\_MASK\_GREEN 0x07E0

#define RGB565\_MASK\_BLUE 0x001F

R = (wPixel & RGB565\_MASK\_RED) >> 11; // 取值范围0-31

G = (wPixel & RGB565\_MASK\_GREEN) >> 5; // 取值范围0-63

B = wPixel & RGB565\_MASK\_BLUE; // 取值范围0-31

#define RGB(r,g,b) (unsigned int)( (r|0x08 << 11) | (g|0x08 << 6) | b|0x08 )

#define RGB(r,g,b) (unsigned int)( (r|0x08 << 10) | (g|0x08 << 5) | b|0x08 )

该代码可以解决24位与16位相互转换的问题

**··RGB24**

RGB24使用24位来表示一个像素，RGB分量都用8位表示，取值范围为0-255。注意在**内存中**RGB各分量的排列顺序为：**BGR BGR BGR…**。

通常可以使用RGBTRIPLE数据结构来操作一个像素，它的定义为：

**typedef** **struct** tagRGBTRIPLE {

**BYTE** rgbtBlue;  // 蓝色分量

**BYTE** rgbtGreen;  // 绿色分量

**BYTE** rgbtRed;  // 红色分量

} RGBTRIPLE;

**··RGB32**

RGB32使用32位来表示一个像素，RGB分量各用去8位，剩下的8位用作Alpha通道或者不用。（ARGB32就是带Alpha通道的RGB24。）注意在内存中RGB各分量的排列顺序为：BGRA BGRA BGRA…。通常可以使用RGBQUAD数据结构来操作一个像素，它的定义为：

**typedef** **struct** tagRGBQUAD {

**BYTE** rgbBlue;  // 蓝色分量

**BYTE** rgbGreen;  // 绿色分量

**BYTE** rgbRed;  // 红色分量

**BYTE** rgbReserved; // 保留字节（用作Alpha通道或忽略）

} RGBQUAD。

**· YUV**

YUV是编译true-color颜色空间（color space）的种类。

**Y′UV, YUV, YCbCr, YPbPr**所指涉的范围，常有混淆或重叠的情况。从历史的演变来说，其中YUV和Y'UV通常用来编码**电视**的**模拟**信号，而**YCbCr**则是用来描述**数字视频信号**，适合视频与图片压缩以及传输，例如MPEG、JPEG。但在现今，YUV通常已经在电脑系统上广泛使用。

YUV也是一种**颜色编码方法**，主要用于电视系统以及模拟视频领域，它将亮度信息（Y）与色彩信息（UV）分离，没有UV信息一样可以显示完整的图像，只不过是黑白的，这样的设计很好地解决了彩色电视机与黑白电视的兼容问题。并且，YUV不像RGB那样要求三个独立的视频信号同时传输，所以用YUV方式传送**占用极少的频宽**。、

**“Y”**表示**明亮度**（Luminance或Luma），也就是**灰阶值**，只有Y信号为黑白图像。

**“U”和“V”**表示的则是**色度**（Chrominance或Chroma），**作用**是描述**影像**色彩及饱和度，用于指定像素的颜色。UV的值越高，代表该像素会有更饱和的颜色。

**YCbCr**有的时候会被写作：YCBCR，是色彩空间的一种，通常会用于影片中的影像连续处理，或是数字摄影系统中。Y为颜色的亮度（luma）成分、而**CB**和**CR**则为**蓝色**和**红色**的**浓度偏移量**成分。

如果想把**YUV格式像素数据**变成**灰度图像**，只需要将**U、V分量设置**成**128**即可。

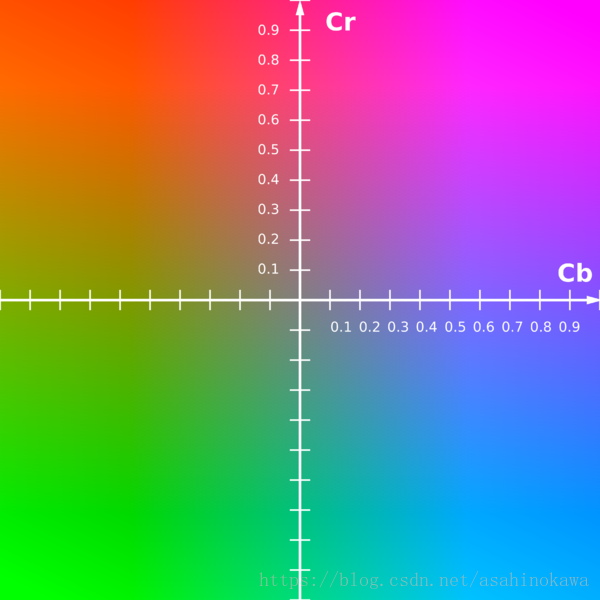
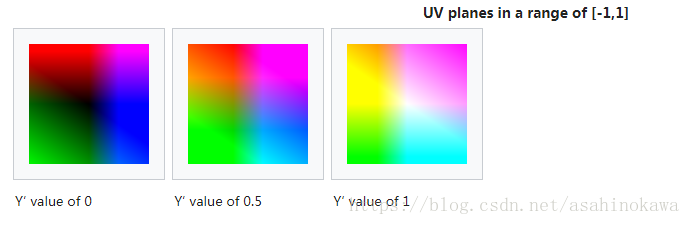
这是因为U、V是图像中的**经过偏置处理**的**色度分量**。**色度分量**在**偏置处理前**的取值范围是**-128**至**127**，这时候的**无色**对应的是“0”值。经过**偏置后**色度分量取值变成了**0**至**255**，因而此时的**无色**对应的就是128了。

对于YUV模型，实际上很多时候，我们是把它和YIQ / YCrCb模型混为一谈的。

实际上,**YUV模型**用于**PAL制式**的电视系统，Y表示亮度，UV并非任何单词的缩写。

**YIQ模型**与YUV模型类似，用于**NTSC制式**的电视系统。YIQ颜色空间中的I和Q分量相当于将YUV空间中的UV分量做了一个33度的旋转。

**YCbCr**颜色空间是由YUV颜色空间派生的一种颜色空间，主要用于**数字电视系统**中。从RGB到YCbCr的转换中，输入、输出都是8位二进制格式。

Y表示亮度，CbCr表示颜色。怎么表示颜色，可以看下面这幅坐标图：   
   
 Y要如何表示亮度呢，下面是Y在不同的情况下的表现：   
 

**·YUV采样方式**

为节省带宽起见，大多数YUV格式平均使用的**每像素位数(bpp)**都**少于24位**。

主要的抽样（subsample）格式有YCbCr 4:2:0、YCbCr 4:2:2、YCbCr 4:1:1和YCbCr 4:4:4。YUV的表示法称为A:B:C表示法：

YUV 4:4:4采样,每一个Y对应**一组**UV分量 [Y0U0V0 Y1U1V1...Y7U7V7 24bit/每组]

YUV 4:2:2采样,每两个Y共用**一组**UV分量 [Y0U0Y1V0 Y2U1Y3V1 ... Y10U5Y11V5]

YUV 4:2:0采样,每四个Y共用**一组**UV分量。 [Y0U0Y1 Y2U1Y3 Y4U2Y5 Y6U3Y7]

**·YUV存储格式**：

YUV码流的**存储格式**其实与其**采样的方式**密切相关，主流的采样方式有三种，YUV4:4:4，YUV4:2:2，YUV4:2:0，这里我想强调的是如何根据其采样格式来从码流中还原每个像素点的YUV值，因为只有正确地还原了每个像素点的YUV值，才能通过YUV与RGB的转换公式提取出每个像素点的RGB值，然后显示出来。

**紧缩格式**（packed formats）：将Y、U、V值存储成MacroPixels数组，和RGB的存放方式类似。

紧缩格式（packedformat）中的YUV是混合在一起的，对于YUV4:4:4格式而言，用紧缩格式很合适的，因此就有了UYVY、YUYV等。

**平面格式**（planar formats）：将Y、U、V的三个分量分别存放在不同的矩阵中。

平面格式（planarformats）是指每Y分量，U分量和V分量都是以独立的平面组织的，也就是说所有的U分量必须在Y分量后面，而V分量在所有的U分量后面，此一格式适用于采样（subsample）。平面格式（planarformat）有I420（4:2:0）、YV12、IYUV等。

**·· YUYV 格式 （属于YUV422）**

Start+0 Y00 U00 Y01 V00 Y02 U01 Y03 V01

Start+8 Y10 U10 Y11 V10 Y12 U11 Y13 V11

Start+16 Y20 U20 Y21 V20 Y22 U21 Y23 V21

Start+24 Y30 U30 Y31 V30 Y32 U31 Y33 V31

YUYV为YUV422采样的存储格式中的一种，相邻的两个Y共用其相邻的两个U、V，分析，对于像素点Y’00、Y’01 而言，其U、V的值均为 U00、V00，其他的像素点的YUV取值依次类推。

**·· UYVY 格式 （属于YUV422）**

UYVY格式也是YUV422采样的存储格式中的一种，只不过与YUYV不同的是UV的排列顺序不一样而已，还原其每个像素点的YUV值的方法与上面一样。

**·· YUV422P（属于YUV422）**

Start+0 Y00 Y01 Y02 Y03

Start+4 Y10 Y11 Y12 Y13

Start+8 Y20 Y21 Y22 Y23

Start+12 Y30 Y31 Y32 Y33

Start+16 U00 U01

Start+18 U10 U11

Start+20 U20 U21

Start+22 U30 U31

Start+24 V00 V01

Start+26 V10 V11

Start+28 V20 V21

Start+30 V30 V31

YUV422P也属于YUV422的一种，它是一种Plane模式，即平面模式，并不是将YUV数据交错存储，而是先存放所有的Y分量，然后存储所有的U（Cb）分量，最后存储所有的V（Cr）分量，如上图所示。其每一个像素点的YUV值提取方法也是遵循YUV422格式的最基本提取方法，即两个Y共用一个UV。比如，对于像素点Y’00、Y’01 而言，其Cb、Cr的值均为 Cb00、Cr00。

**·· YV12，YU12(I420)格式（属于YUV420P）**

Start+0 Y00 Y01 Y02 Y03

Start+4 Y10 Y11 Y12 Y13

Start+8 Y20 Y21 Y22 Y23

Start+12 Y30 Y31 Y32 Y33

Start+16 U00 U01

Start+18 U10 U11

Start+20 V00 V01

Start+22 V10 V11

YU12和YV12属于**YUV420格式**，也是一种Plane模式，将Y、U、V分量分别打包，依次存储。其每一个像素点的YUV数据提取遵循YUV420格式的提取方式，即4个Y分量共用一组UV。

注意，上图中，Y’00、Y’01、Y’10、Y’11共用Cr00、Cb00，其他依次类推。

**·· NV12、NV21（属于YUV420）**

Start+0 Y00 Y01 Y02 Y03

Start+4 Y10 Y11 Y12 Y13

Start+8 Y20 Y21 Y22 Y23

Start+12 Y30 Y31 Y32 Y33

Start+16 U00 V00 U01 V01

Start+18 U10 V10 U11 V11

NV12和NV21属于YUV420格式，是一种two-plane模式，即Y和UV分为两个Plane，但是UV（CbCr）为交错存储，而不是分为三个plane。其提取方式与上一种类似，即Y’00、Y’01、Y’10、Y’11共用Cr00、Cb00

**·· YV12和YU12/I420的区别**

一般来说，**直接采集**到的**视频数据**是**RGB24**的格式。

RGB24一帧的大小size＝width×heigth×3 Bit，RGB32的size＝width×heigth×4， 如果是I420（即YUV标准格式4：2：0）的数据量是 size＝width×heigth×1.5 Bit。 在采集到RGB24数据后，需要对这个格式的数据进行第一次压缩。即将图像的颜色空间由**RGB TO YUV**。因为，X264在进行编码的时候需要标准的YUV（4：2：0）。但是这里需要注意的是，虽然YV12也是（4：2：0），但是YV12和I420的却是不同的，在存储空间上面有些区别。

如下：

YV12 ： 亮度（行×列） ＋ V（行×列/4) + U（行×列/4）

I420 ： 亮度（行×列） ＋ U（行×列/4) + U（行×列/4）

可以看出，YV12和I420基本上是一样的，就是**UV的顺序**不同。

继续我们的话题，经过第一次数据压缩后RGB24－>YUV（I420）。这样，数据量将减少一半。同样，如果是RGB24－>YUV（YV12），也是减少一半。但是，虽然都是一半，如果是YV12的话效果就有很大损失。然后，经过X264编码后，数据量将大大减少。将编码后的数据打包，通过RTP实时传送。到达目的地后，将数据取出，进行解码。完成解码后，数据仍然是YUV格式的，所以，还需要一次转换，这样windows的驱动才可以处理，就是**YUV2RGB24**。

**·· YUV420P和YUV420SP的区别**

**YUV420P**，Y，U，V三个分量都是**平面格式**，分为I420和YV12。I420格式和YV12格式的不同 在U平面 和 V平面 的位置不同。在I420(YU12)格式中，U平面紧跟在Y平面之后，然后才是V平面（即：YUV）；但YV12则是相反（即：YVU）。

**YUV420SP**, Y分量平面格式，UV打包格式, 即NV12。 NV12与NV21类似，U 和 V 交错排列,不同在于UV顺序。

YU12/I420: YYYYYYYY UU VV => YUV420P

YV12: YYYYYYYY VV UU => YUV420P

NV12: YYYYYYYY UVUV => YUV420SP

NV21: YYYYYYYY VUVU => YUV420SP

**·YUV文件大小计算**

以720×480大小图象YUV420 planar为例，

其存储格式是： 共大小为(720×480×3>>1)字节，

分为三个部分:Y, U和V

Y分量： (720×480)个字节

U(Cb)分量： (720×480>>2)个字节 （ n>>2 <=> n/4 ）

V(Cr)分量： (720×480>>2)个字节

三个部分内部均是行优先存储，三个部分之间是Y,U,V 顺序存储。 即 ：

0 ~ 720×480字节 是Y分量值存储空间，

720×480 ~ 720×480×5/4字节 是U分量值存储空间，

720×480×5/4 ~ 720×480×3/2字节 是V分量值存储空间。

在YUV420中，一个像素点对应一个Y，一个4X4的小方块对应一个U和V。对于所有YUV420图像，它们的Y值排列是完全相同的，因为只有Y的图像就是灰度图像。YUV420sp与YUV420p的数据格式它们的UV排列在原理上是完全不同的。420p它是先把U存放完后，再存放V，也就是说UV它们是连续的。而420sp它是UV、UV这样交替存放的。(见下图) 有了上面的理论，我就可以准确的计算出一个YUV420在内存中存放的大小。

width \* hight =Y（总和）

U = Y / 4

V = Y / 4。

所以YUV420 数据在内存中的长度是 width \* hight \* 3 / 2，

假设一个分辨率为8X4的YUV420图像，它们的格式如下图：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Y01** | **Y02** | **Y03** | **Y04** | **Y05** | **Y06** | **Y07** | **Y08** |
| **Y09** | **Y10** | **Y11** | **Y12** | **Y13** | **Y14** | **Y15** | **Y13** |
| **Y17** | **Y18** | **Y19** | **Y20** | **Y21** | **Y22** | **Y23** | **Y24** |
| **Y25** | **Y26** | **Y27** | **Y28** | **Y29** | **Y30** | **Y31** | **Y32** |
| **U01** | **V01** | **U02** | **V02** | **U03** | **V03** | **U04** | **V04** |
| **U05** | **V05** | **U06** | **V06** | **U07** | **V07** | **U08** | **V08** |

YUV420P数据格式如下图

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Y01** | **Y02** | **Y03** | **Y04** | **Y05** | **Y06** | **Y07** | **Y08** |
| **Y09** | **Y10** | **Y11** | **Y12** | **Y13** | **Y14** | **Y15** | **Y13** |
| **Y17** | **Y18** | **Y19** | **Y20** | **Y21** | **Y22** | **Y23** | **Y24** |
| **Y25** | **Y26** | **Y27** | **Y28** | **Y29** | **Y30** | **Y31** | **Y32** |
| **U01** | **U02** | **U03** | **U04** | **U05** | **U06** | **U07** | **U08** |
| **V01** | **V02** | **V03** | **V04** | **V05** | **V06** | **V07** | **V08** |

**· 常见的色彩空间**

图像格式定义.

bpp (bit per pixer) -- 每像素色彩位数

**YUV420P planar YUV 4:2:0, 12bpp, ( 1 Cr & Cb sample per 2x2 Y samples )**

**YUYV422 packed YUV 4:2:2, 16bpp, ( Y0 Cb Y1 Cr )**

**UYVY422 packed YUV 4:2:2, 16bpp, ( Cb Y0 Cr Y1 )**

**YUV422P planar YUV 4:2:2, 16bpp, ( 1 Cr & Cb sample per 2x1 Y samples )**

**YUV444P planar YUV 4:4:4, 24bpp, ( 1 Cr & Cb sample per 1x1 Y samples )**

**YUV410P planar YUV 4:1:0, 9bpp, ( 1 Cr & Cb sample per 4x4 Y samples )**

**YUV411P planar YUV 4:1:1, 12bpp, ( 1 Cr & Cb sample per 4x1 Y samples )**

**GRAY8 Y 8bpp**

**MONOWHITE Y, 1bpp, 0 is white, 1 is black, in each byte pixels are ordered from the msb to the lsb \*/**

**MONOBLACK Y, 1bpp, 0 is black, 1 is white, in each byte pixels are ordered from the msb to the lsb \*/**

**NV12 planar YUV 4:2:0, 12bpp 1 plane for Y and 1 plane for the UV components, which are interleaved (first byte U and the following byte V)**

**NV21 planar YUV 4:2:0 12bpp as above, but U and V bytes are swapped \*/**

**RGB24 packed RGB 8:8:8, 24bpp, RGBRGB...**

**BGR24 packed RGB 8:8:8, 24bpp, BGRBGR...**

**ARGB packed ARGB 8:8:8:8, 32bpp, ARGBARGB...**

**RGBA packed RGBA 8:8:8:8, 32bpp, RGBARGBA...**

**ABGR packed ABGR 8:8:8:8, 32bpp, ABGRABGR...**

**BGRA packed BGRA 8:8:8:8, 32bpp, BGRABGRA...**

**RGB565BE packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), big-endian**

**RGB565LE packed RGB 5:6:5, 16bpp, (msb) 5R 6G 5B(lsb), little-endian**

**RGB555BE packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), big-endian,**

**most significant bit to 0**

**RGB555LE packed RGB 5:5:5, 16bpp, (msb)1A 5R 5G 5B(lsb), little-endian,**

**most significant bit to 0**

**BGR565BE packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), big-endian**

**BGR565LE packed BGR 5:6:5, 16bpp, (msb) 5B 6G 5R(lsb), little-endian**

**BGR555BE packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), big-endian,**

**most significant bit to 1**

**BGR555LE packed BGR 5:5:5, 16bpp, (msb)1A 5B 5G 5R(lsb), little-endian,**

**most significant bit to 1**

**0RGB packed RGB 8:8:8, 32bpp, 0RGB0RGB...**

**RGB0 packed RGB 8:8:8, 32bpp, RGB0RGB0...**

**0BGR packed BGR 8:8:8, 32bpp, 0BGR0BGR...**

**BGR0 packed BGR 8:8:8, 32bpp, BGR0BGR0...**

**BGGR8 bayer BGBG..(odd line), GRGR..(even line), 8-bit samples**

**RGGB8 bayer RGRG..(odd line), GBGB..(even line), 8-bit samples**

**GBRG8 bayer GBGB..(odd line), RGRG..(even line), 8-bit samples**

**GRBG8 bayer GRGR..(odd line), BGBG..(even line), 8-bit samples**

**· YUV和RGB互相转换**

**·· 转换公式**

YUV(256 级别) 可以从8位 RGB 直接计算：

Y = 0.299 R + 0.587 G + 0.114 B

U = - 0.1687 R - 0.3313 G + 0.5 B + 128

V = 0.5 R - 0.4187 G - 0.0813 B + 128

反过来，RGB 也可以直接从YUV (256级别) 计算:

R = Y + 1.402 (Cr-128)

G = Y - 0.34414 (Cb-128) - 0.71414 (Cr-128)

B = Y + 1.772 (Cb-128)

YUV与RGB相互转换的公式如下（RGB取值范围均为0-255）：

Y = 0.299R + 0.587G + 0.114B

U = -0.147R - 0.289G + 0.463B

V = 0.615R - 0.515G - 0.100B

R = Y + 1.14V

G = Y - 0.39U - 0.58V

B = Y + 2.03U

// 用整型算法代替浮点算法，提高云算速度

y = (unsigned char)( ( 66\*r + 129\*g + 25\*b + 128) >> 8 ) + 16 ;

u = (unsigned char)( ( -38\*r - 74\*g + 112\*b + 128) >> 8 ) + 128 ;

v = (unsigned char)( ( 112\*r - 94\*g - 18\*b + 128) >> 8 ) + 128 ;

**·· 转换方法及转换效率**

|  |  |
| --- | --- |
| **Methed** | **Time(ms)** |
| YV12ToBGR24\_Native | 83.7263 |
| YV12ToBGR24\_Table | 54.2376 |
| YV12ToBGR24\_OpenCV | 26.0529 |
| YV12ToBGR24\_FFmpeg | 3.41499 |
| YV12ToBGR24\_Pinknoise | 14.1215 |

由上述表格可以看出，基于FFmpeg的格式转换效率最高，利用查找表法可以提高转换效率，但基于查找表法的转换效率有待进一步优化提升。

后续应关注于FFmpeg其格式转换的算法实现，抽取其实现代码，使格式转换不依赖于FFmpeg库。

**· 视频质量系数PSNR**

对于8bit量化的像素数据来说，PSNR的计算公式如下所示。

IMG_256

上述公式中mse的计算公式如下所示。

IMG_256

其中M，N分别为图像的宽高，xij和yij分别为两张图像的每一个像素值。PSNR通常用于质量评价，就是计算受损图像与原始图像之间的差别，以此来评价受损图像的质量。

PSNR取值通常情况下都在**20-50**的范围内，取值**越高**，代表两张图像越接近，反映出受损图像**质量越好**。

8 BMP图像数据格式

**1 简介**

**BMP(Bitmap-File)图形文件**是Windows采用的**图形文件格式**，在Windows环境下运行的所有图象处理软件都支持BMP图象文件格式。

Windows系统内部各图像绘制操作都是以BMP为基础的。

Windows 3.0以前的BMP图文件格式与显示设备有关，因此把这种BMP图象文件格式称为设备相关位图DDB(device-dependent bitmap)文件格式。

Windows 3.0以后的BMP图象文件与显示设备无关，因此把这种BMP图象文件格式称为设备无关位图DIB(device-independent bitmap)格式。

（注：Windows 3.0以后，在系统中仍然存在DDB位图，象BitBlt()这种函数就是基于DDB位图的，只不过如果你想将图像以BMP格式保存到磁盘文件中时，微软极力推荐你以DIB格式保存），目的是为了让Windows能够在任何类型的显示设备上显示所存储的图象。BMP位图文件默认的文件扩展名是BMP或者bmp（有时它也会以.DIB或.RLE作扩展名）。

**2 BMP格式结构**

BMP文件的数据按照从文件头开始的先后顺序分为四个部分：

◆ **位图文件头**(bmp file header)： 提供文件的格式、大小等信息

◆ **位图信息头**(bitmap information)：提供图像数据的尺寸、位平面数、压缩方式、颜色索引等信息

◆ **调色板**(color palette)：**可选**，如使用索引来表示图像，调色板就是索引与其对应的颜色的映射表

◆ **位图数据**(bitmap data)：图像数据区

BMP图片文件数据表如下：

|  |  |
| --- | --- |
| 数据段名称 | 大小（byte） |
| 位图文件图 | 14 |
| 位图信息头 | 40 |
| 调色板 | 由biBitCount决定 |
| 位图数据 | 图片大小决定 |

**· BMP文件头**

BMP文件头结构体定义如下：

typedef struct tagBITMAPFILEHEADER

{

UINT16 bfType; //2Bytes，必须为"BM"，即0x424D 才是Windows位图文件

DWORD bfSize; //4Bytes，整个BMP文件的大小

UINT16 bfReserved1; //2Bytes，保留，为0

UINT16 bfReserved2; //2Bytes，保留，为0

DWORD bfOffBits; //4Bytes，文件起始位置到图像像素数据的字节偏移量

} **BITMAPFILEHEADER**;

BMP文件头数据表如下：

|  |  |  |
| --- | --- | --- |
| 变量名 | 大小/Byte | 作用说明 |
| bfType | 2 | 文件标识符  必须为"BM"，即0x424D 才是Windows位图文件  ‘BM’：Windows 3.1x, 95, NT,…  ‘BA’：OS/2 Bitmap Array  ‘CI’：OS/2 Color Icon  ‘CP’：OS/2 Color Pointer  ‘IC’：OS/2 Icon  ‘PT’：OS/2 Pointer  因为OS/2系统并没有被普及开，所以在编程时，你只需判断第一个标识“BM”就行 |
| bfSize | 4 | 整个BMP文件的大小（以位B为单位） |
| bfReserved1 | 2 | 保留，必须设置为0 |
| bfReserved2 | 2 | 保留，必须设置为0 |
| bfOffBits | 4 | 说明：  从文件头0000h开始到图像像素数据的字节偏移量（以字节Bytes为单位），以为位图的调色板长度根据位图格式不同而变化，可以用这个偏移量快速从文件中读取图像数据 |

**· BMP信息头**

BMP信息头结构体定义如下：

typedef struct \_tagBMP\_INFOHEADER

{

DWORD  biSize;      //4Bytes，INFOHEADER结构体大小，存在其他版本I NFOHEADER，用作区分

LONG   biWidth;     //4Bytes，图像宽度（以像素为单位）

LONG   biHeight;     //4Bytes，图像高度，+：图像存储顺序为Bottom2Top，-：Top2Bottom

WORD   biPlanes;     //2Bytes，图像数据平面，BMP存储RGB数据，因此总为1

WORD   biBitCount;          //2Bytes，图像像素位数

DWORD  biCompression;     //4Bytes，0：不压缩，1：RLE8，2：RLE4

DWORD  biSizeImage;        //4Bytes，4字节对齐的图像数据大小

LONG   biXPelsPerMeter;    //4 Bytes，用象素/米表示的水平分辨率

LONG   biYPelsPerMeter;    //4 Bytes，用象素/米表示的垂直分辨率

DWORD  biClrUsed;          //4 Bytes，实际使用的调色板索引数，0：使用所有的调色板索引

DWORD biClrImportant;      //4 Bytes，重要的调色板索引数，0：所有的调色板索引都重要

}**BMP\_INFOHEADER**;

BMP信息头数据表如下：

|  |  |  |
| --- | --- | --- |
| 变量名 | 大小 | 作用说明 |
| biSize | 4Byte | BNP信息头即BMP\_INFOHEADER结构体所需要的字节数（以字节为单位） |
| biWidth | 4Byte | 说明图像的宽度（以像素为单位） |
| biHeight | 4Byte | 说明图像的高度（以像素为单位）  这个值还有一个用处，指明图像是正向的位图还是倒向的位图，  该值是正数说明图像是倒向的 即图像存储是由下到上；  该值是负数说明图像是倒向的 即图像存储是由上到下。  大多数BMP位图是倒向的位图，所以此值是正值。 |
| biPlanes | 2Byte | 为目标设备说明位面数，其值总设置为1 |
| biBitCount | 2Byte | 说明一个像素点占几位（以比特位/像素位单位），其值可为1,4,8,16,24或32 |
| biCompression | 4Byte | 说明图像数据的压缩类型，取值范围为：  0 BI\_RGB 不压缩（最常用）  1 BI\_RLE8 8比特游程编码（BLE），只用于8位位图  2 BI\_RLE4 4比特游程编码（BLE），只用于4位位图  3 BI\_BITFIELDS比特域（BLE），只用于16/32位位图 |
| biSizeImage | 4Byte | 说明图像的大小，以字节为单位。当用BI\_RGB格式时，总设置为0 |
| biXPelsPerMeter | 4Byte | 说明水平分辨率，用像素/米表示，有符号整数 |
| biYPelsPerMeter | 4Byte | 说明垂直分辨率，用像素/米表示，有符号整数 |
| biClrUsed | 4Byte | 说明位图实际使用的调色板索引数，0：使用所有的调色板索引 |
| biClrImportant | 4Byte | 说明对图像显示有重要影响的颜色索引的数目，如果是0，表示都重要。 |

**· BMP调色板**

BMP调色板结构体定义如下：

typedef struct \_tagRGBQUAD

{

BYTE rgbBlue; // 指定蓝色强度

BYTE rgbGreen; // 指定绿色强度

BYTE rgbRed; // 指定红色强度

BYTE rgbReserved; // 保留，设置为0

} **RGBQUAD**;

1，4，8位图像才会使用调色板数据，16,24,32位图像不需要调色板数据，即调色板最多只需要256项（索引0 - 255）。

颜色表的大小根据所使用的颜色模式而定：

2色图像为8字节；16色图像位64字节；256色图像为1024字节。

其中，每4字节表示一种颜色，并以B（蓝色）、G（绿色）、R（红色）、alpha（32位位图的透明度值，一般不需要）。即首先4字节表示颜色号1的颜色，接下来表示颜色号2的颜色，依此类推。

颜色表中RGBQUAD结构数据的个数有biBitCount来确定，当biBitCount=1,4,8时，分别有2,16,256个表项。

当biBitCount=1时，为2色图像，BMP位图中有2个数据结构RGBQUAD，一个调色板占用4字节数据，所以2色图像的调色板长度为2\*4为8字节。

当biBitCount=4时，为16色图像，BMP位图中有16个数据结构RGBQUAD，一个调色板占用4字节数据，所以16像的调色板长度为16\*4为64字节。

当biBitCount=8时，为256色图像，BMP位图中有256个数据结构RGBQUAD，一个调色板占用4字节数据，所以256色图像的调色板长度为256\*4为1024字节。

当biBitCount=16，24或32时，没有颜色表。

**· BMP图像数据区**

位图数据记录了位图的每一个像素值，记录顺序是在扫描行内是从左到右,扫描行之间是从下到上。位图的一个像素值所占的字节数:

当biBitCount=1时，8个像素占1个字节;

当biBitCount=4时，2个像素占1个字节;

当biBitCount=8时，1个像素占1个字节;

当biBitCount=24时,1个像素占3个字节;

Windows规定一个扫描行所占的字节数必须是4的倍数(即以long为单位),不足的以0填充，

一个扫描行所占的字节数计算方法:

DataSizePerLine= (biWidth\* biBitCount+31)/8;

// 一个扫描行所占的字节数

DataSizePerLine= DataSizePerLine/4\*4; // 字节数必须是4的倍数

位图数据的大小(不压缩情况下):

DataSize= DataSizePerLine\* biHeight;

颜色表接下来位为位图文件的图像数据区，在此部分记录着每点像素对应的颜色号，其记录方式也随颜色模式而定，既2色图像每点占1位（8位为1字节）；16色图像每点占4位（半字节）；256色图像每点占8位（1字节）；真彩色图像每点占24位（3字节）。所以，整个数据区的大小也会随之变化。

究其规律而言，可的出如下计算公式：

图像数据信息大小=（图像宽度\*图像高度\*记录像素的位数）/8。

**· RGB像素数据封装BMP图像**

1)将RGB数据前面加上文件头和信息头。

2)将RGB数据中每个像素的“B”和“R”的位置互换。

BMP采用的是小端（Little Endian）存储方式。

这种存储方式中“RGB24”格式的像素的分量存储的先后顺序为B、G、R。

由于RGB24格式存储的顺序是R、G、B，所以需要将“R”和“B”顺序作一个调换再进行存储。

# 知识扩展

**FFmpeg**

**1 概念**

**FFmpeg**是一套可以用来**记录**、**转换数字音频**、**视频**，并能将其**转化为流**的开源计算机程序。采用**LGPL**或**GPL**许可证。

它提供了录制、转换以及**流化音视频**的完整解决方案。它包含了非常先进的音频/视频编解码库libavcodec，为了保证高可移植性和编解码质量，libavcodec里很多code都是从头开发的。

FFmpeg在Linux平台下开发，但它同样也可以在其它操作系统环境中编译运行，包括Windows、Mac OS X等。这个项目最早由Fabrice Bellard发起，2004年至2015年间由Michael Niedermayer主要负责维护。许多FFmpeg的开发人员都来自MPlayer项目，而且当前FFmpeg也是放在MPlayer项目组的服务器上。项目的名称来自MPEG视频编码标准，前面的"FF"代表"Fast Forward"。

**2 项目结构**

**libavformat**：用于各种音视频封装格式的生成和解析，包括获取解码所需信息以生成解码上下文结构和读取音视频帧等功能；

**libavcodec**：用于各种类型声音/图像编解码；

**libavutil**：包含一些公共的工具函数；

**libswscale**：用于视频场景比例缩放、色彩映射转换；

**libpostproc**：用于后期效果处理；

**ffmpeg**：该项目提供的一个工具，可用于格式转换、解码或电视卡即时编码等；

**ffsever**：一个 HTTP 多媒体即时广播串流服务器；

**ffplay**：是一个简单的播放器，使用ffmpeg 库解析和解码，通过SDL显示；

**3 命令集合**

1.获取视频的信息

ffmpeg -i video.avi

2.将图片序列合成视频

ffmpeg -f image2 -i image%d.jpg video.mpg

上面的命令会把当前目录下的图片（名字如：image1.jpg. image2.jpg. 等...）合并成video.mpg

3.将视频分解成图片序列

ffmpeg -i video.mpg image%d.jpg

上面的命令会生成image1.jpg. image2.jpg. ...

支持的图片格式有：PGM. PPM. PAM. PGMYUV. JPEG. GIF. PNG. TIFF. SGI

4.为视频重新编码以适合在iPod/iPhone上播放

ffmpeg -i source\_video.avi input -acodec aac -ab 128kb -vcodec mpeg4 -b 1200kb -mbd 2 -flags +4mv+trell -aic 2 -cmp 2 -subcmp 2 -s 320x180 -title X final\_video.mp4

说明：

\* 源视频：source\_video.avi

\* 音频编码：aac

\* 音频位率：128kb/s

\* 视频编码：mpeg4

\* 视频位率：1200kb/s

\* 视频尺寸：320 X 180

\* 生成的视频：final\_video.mp4

5.为视频重新编码以适合在PSP上播放

ffmpeg -i source\_video.avi -b 300 -s 320x240 -vcodec xvid -ab 32 -ar 24000 -acodec aac final\_video.mp4

说明：

\* 源视频：source\_video.avi

\* 音频编码：aac

\* 音频位率：32kb/s

\* 视频编码：xvid

\* 视频位率：1200kb/s

\* 视频尺寸：320 X 180

\* 生成的视频：final\_video.mp4

6.从视频抽出声音.并存为Mp3

ffmpeg -i source\_video.avi -vn -ar 44100 -ac 2 -ab 192 -f mp3 sound.mp3

说明：

\* 源视频：source\_video.avi

\* 音频位率：192kb/s

\* 输出格式：mp3

\* 生成的声音：sound.mp3

7.将wav/ogg/aac文件转成Mp3

ffmpeg -i audio.wav -acodec libmp3lame audio.mp3

ffmpeg -i audio.ogg -acodec libmp3lame audio.mp3

ffmpeg -i audio.ac3 -acodec libmp3lame audio.mp3

ffmpeg -i audio.aac -acodec libmp3lame audio.mp3

8.将.avi视频转成.mpg

ffmpeg -i video\_origine.avi video\_finale.mpg

9.将.mpg转成.avi

ffmpeg -i video\_origine.mpg video\_finale.avi

10.将.avi转成gif动画（未压缩）

ffmpeg -i video\_origine.avi gif\_anime.gif

11.合成视频和音频

ffmpeg -i son.wav -i video\_origine.avi video\_finale.mpg

12.将.avi转成.flv

ffmpeg -i video\_origine.avi -ab 56 -ar 44100 -b 200 -r 15 -s 320x240 -f flv video\_finale.flv

13.将.avi转成dv

ffmpeg -i video\_origine.avi -s pal -r pal -aspect 4:3 -ar 48000 -ac 2 video\_finale.dv

或者：

ffmpeg -i video\_origine.avi -target pal-dv video\_finale.dv

14.将.avi压缩成divx

ffmpeg -i video\_origine.avi -s 320x240 -vcodec msmpeg4v2 video\_finale.avi

15.将Ogg Theora压缩成Mpeg dvd

ffmpeg -i film\_sortie\_cinelerra.ogm -s 720x576 -vcodec mpeg2video -acodec mp3 film\_terminate.mpg

16.将.avi压缩成SVCD mpeg2

NTSC格式：

ffmpeg -i video\_origine.avi -target ntsc-svcd video\_finale.mpg

PAL格式：

ffmpeg -i video\_origine.avi -target pal-svcd video\_finale.mpg

17.将.avi压缩成VCD mpeg2

NTSC格式：

ffmpeg -i video\_origine.avi -target ntsc-vcd video\_finale.mpg

PAL格式：

ffmpeg -i video\_origine.avi -target pal-vcd video\_finale.mpg

18.多通道编码

ffmpeg -i fichierentree -pass 2 -passlogfile ffmpeg2pass fichiersortie-2

19.从flv提取mp3

ffmpeg -i source.flv -ab 128k dest.mp3