第 5 章

视 频 基 础

相信读者对视频一定不陌生，平时也经常浏览各大视频网站，甚至偶尔还会把视频缓存到本地，保存成.mkv、.avi 等格式的文件。这就涉及常说的网络流媒体和本地视频文件。本地视频文件常见格式有 MP4、MKV、AVI 等。在流媒体网站上看视频常用的协议有 HTTP、RTSP、RTMP、HLS 等。视频技术比较复杂，包括视频封装、视频编解码、视频播放、视频转码等。

5.1 视频基础概念

人们目前所处的时代是移动互联网时代，也可以说是视频时代。日常生活中正在被越来越多的视频元素所影响，例如从爱奇艺到快手，从“三生三世”到“三十而已”，到处都充斥着各种各样的视频。几个比较流行的视频 App，如图 5-1 所示。

图 5-1 视频相关的 App

而这一切离不开视频拍摄技术的不断升级，还有视频制作产业的日益强大，也离不开通信技术的飞速进步。试想一下，如果还是采用多年前的 56k Modem 拨号上网，或者使用 2G 手机上网，读者还能享受到现在动辄 1080P 甚至 4K 的视频体验吗？

注意：1080P 是指视频分辨率为 1920×1080 逐行扫描。

大众能享受到视频带来的便利和乐趣，除了视频拍摄工具和网络通信技术升级之外，还有一个重要因素就是视频编码技术的突飞猛进。视频编码技术涉及的内容太过专业和庞杂，市面上的书籍或博客多数只是枯燥地罗列技术概念，对于新手来讲读完后依然糊里糊涂，笔者打算借此机会专门整理成关于音视频、编解码及流媒体直播的零基础科普系列图书。

5.1.1　图像与像素

图像是人类视觉的基础，是自然景物的客观反映，是人类认识世界和人类本身发展的重要源泉。“图”是物体反射或透射光的分布，“像”是人的视觉系统所接收的图在人脑中所形成的印象或认识。图像是客观对象的一种相似性的、生动性的描述或写真，是人类社会活动中最常用的信息载体，或者说图像是客观对象的一种表示，它包含了被描述对象的有关信息。广义上，图像就是所有具有视觉效果的画面，包括纸介质的、底片或照片上的、电视上的、投影仪或计算机屏幕上的画面。在计算机领域，与图像相关的概念非常多，包括像素、PPI、图像位深度等。

1. 像素

在讲解图像之前，需要先了解一下像素。图像本质上是由很多“带有颜色的点”组成的，而这些点，就是“像素”。像素的英文叫 Pixel，缩写为 PX。这个单词是由图像（Picture）和元素（Element）这两个单词所组成的。图像与像素的关系，如图 5-2 所示。

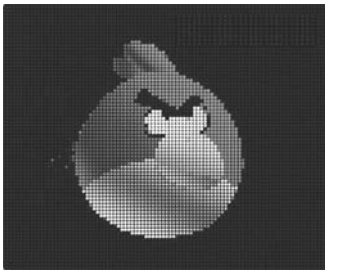


图 5-2　图像与像素的关系

像素是图像显示的基本单位，分辨率是指一张图片的宽度和高度的乘积，单位是像素。通常说一张图片的大小，例如 1920×1080 像素，是指宽度为 1920 像素，高度为 1080 像素。它们的乘积是 1920×1080＝2 073 600，也就是说这张图片是两百万像素的。1920×1080，也被称为这幅图片的分辨率，例如计算机显示器的分辨率如图 5-3 所示。

图 5-3　计算机显示器的分辨率

注意：分辨率是显示器的重要指标。

2. PPI

每英寸的像素数(Pixels Per Inch,PPI)是手机屏幕(或计算机显示器)上每英寸面积到底能放下多少“像素”。这个值越高,图像就越清晰细腻。像素与 PPI 的关系,如图 5-4 所示。

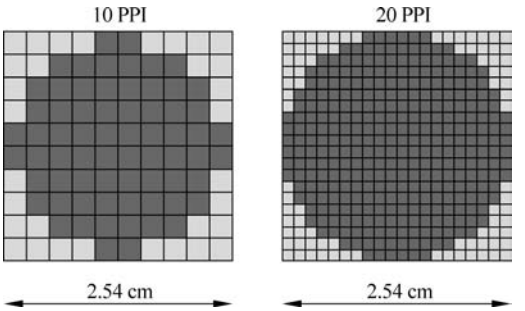


图 5-4　像素与 PPI

以前的功能机,屏幕 PPI 都很低,有很强烈的颗粒感,如图 5-5 所示。

图 5-5　低 PPI 与功能机

后来苹果公司开创了史无前例的“视网膜”(Retina)屏幕,PPI 值高达 326,画质清晰,视觉上也没有了颗粒感,如图 5-6 所示。

图 5-6　高 PPI 与智能机

综上所述,PPI 是像素的密度单位,PPI 值越高,画面的细节就越丰富,所以数码相机拍出来的图片因品牌或生产时间不同可能有所不同,常见的有 72PPI、180PPI 和 300PPI。DPI(Dots Per Inch)是指输出分辨率,是针对输出设备而言的,一般的激光打印机的输出分辨率为[300DPI,600DPI],印刷的照排机可达到[1200DPI,2400DPI],常见的冲印一般为[150DPI,300DPI]。

3. 颜色

像素必须有颜色才能组成缤纷绚丽的图像。如果像素只有黑色和白色,那么图像就是黑白色的。在日常生活中颜色可以有无数种类别,例如常见的口红就有很多种不同的颜色。

这种颜色又该如何表示呢？在计算机系统中,不是用文字来表述颜色的,而是用数字来

表述颜色的，这就涉及“彩色分量数字化”的概念。以前在美术课学过，任何颜色都可以通过红色(Red)、绿色(Green)、蓝色(Blue)按照一定比例调制出来。这 3 种颜色被称为“三原色(RGB)”，如图 5-7 所示。

在计算机里，R、G、B 也被称为“基色分量”。它们的取值，分别为 0～255，一共 256 个等级(256 是 2 的 8 次方)。任何颜色都可以用 R、G、B 这 3 个值的组合表示。例如 RGB＝(183，67，21)所表示的就是一个自定义的颜色值，如图 5-8 所示。

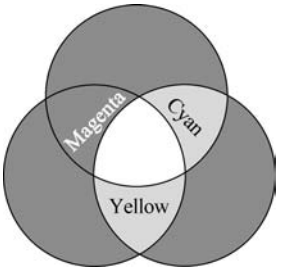


图 5-7　三原色：RGB

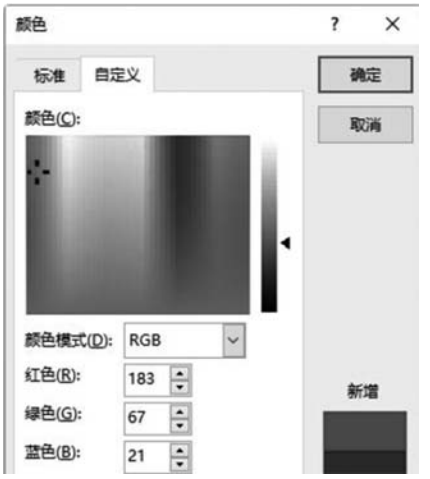


图 5-8　自定义 RGB

通过这种方式，一共能表达多少种颜色呢？RGB 为 3 个分量，假如每个分量占 8b，取值分别为 0～255。那么这 3 个分量的组合取值为 256×256×256＝16 777 216 种，因此也简称为 1600 万色。这种颜色范围已经超过了人眼可见的全部色彩，所以又叫真彩色。更高的精度，对于人眼来讲，已经没有意义了，完全识别不出来。RGB 这 3 色，每色有 8b，这种方式表达出来的颜色，也被称为 24 位真彩色。

4. 像素位深度

像素位深度是指每像素所用的位数，它决定了彩色图像的每像素可能有的颜色数，或者确定灰度图像的每像素可能有的灰度级数。例如一幅彩色图像的每像素用 R、G、B 这 3 个分量来表示，若每个分量用 8b 表示，则每像素共用 24b 表示，即像素的深度为 24b，每像素可以是 16 777 216(千万级)种颜色中的一种。

通常把像素位深度说成图像深度。表示每像素的位数越多，它所能表达的颜色数目就越多，而它的深度也就越深。虽然像素位深度或图像深度可以很深，但由于设备本身的限制，加上人眼自身分辨率的局限，一般情况下，一味追求特别深的像素深度没有实际意义。因为像素深度越深，数据量就越大，所需要的传输带宽及存储空间也就越大。相反，如果像素深度太浅，则会影响图像的质量，图像看起来让人觉得很粗糙而不自然。

5.1.2 色彩空间

RGB是三原色，可以混合出所有的颜色。常见的色彩空间有RGB、YUV、HSV。

1. RGB

RGB色彩空间（又称为颜色模型、颜色空间），由红、绿、蓝三原色组成，广泛用于BMP、TIFF、PPM等。任何颜色都可以通过按一定比例混合三原色产生。如果每个色度成分用8b表示，则取值范围为[0，255]。RGB色彩空间的主要目的是在电子系统中检测、表示和显示图像，例如电视和计算机，但是在传统摄影中也有应用。它是最常见的面向硬件设备的彩色模型，它是与人的视觉系统密切相连的模型，根据人眼结构，所有的颜色都可以看作3种基本颜色——红（R）、绿（G）、蓝（B）的不同比例的组合。国际照度委员会（CIE）规定的红、绿、蓝3种基本色的波长分别为700nm、546.1nm、435.8nm。RGB模型空间是一个正方体，如图5-9所示。

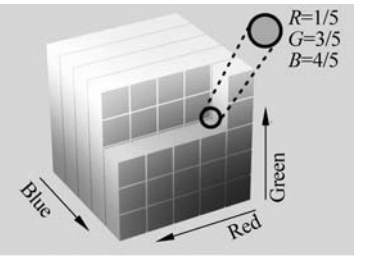


图5-9 RGB正方体模型空间

2. YUV

YUV色彩空间是一种亮度与色度分离的色彩格式。早期的电视都是黑白的，只有亮度值，即Y。有了彩色电视以后，加入了U和V两种色度，形成现在的YUV。YUV能更好地反映HVS的特点。YUV是被欧洲电视系统所采用的一种颜色编码方法。在现代彩色电视系统中，通常采用三管彩色摄像机或彩色CCD摄影机进行取像，然后把取得的彩色图像信号经分色、分别放大校正后得到RGB，再经过矩阵变换电路得到亮度信号Y和两个色差信号R-Y（即U）、B-Y（即V），最后发送端将亮度和两个色差总共3个信号分别进行编码，用同一信道发送出去。这种色彩的表示方法就是所谓的YUV色彩空间表示。采用YUV色彩空间的重要性是它的亮度信号Y和色度信号U、V是分离的。

YUV主要用于优化彩色视频信号的传输，使其向后兼容老式黑白电视。与RGB视频信号传输相比，它最大的优点在于只需占用极小的频宽（RGB要求3个独立的视频信号同时传输），其中Y表示亮度（Luminance或Luma），也就是灰阶值，而U和V所表示的则是色度（Chrominance或Chroma），其作用是描述影像色彩及饱和度，用于指定像素的颜色。亮度是透过RGB输入信号来建立的，其方法是将RGB信号的特定部分叠加到一起。色度则定义了颜色的两个方面（色调与饱和度），分别用Cr和Cb表示。其中，Cr反映了RGB输入信号红色部分与RGB信号亮度值之间的差异，而Cb反映的是RGB输入信号蓝色部分

与 RGB 信号亮度值之同的差异。

YUV 的发明是由于彩色电视与黑白电视的过渡时期，黑白视频只有 Y 视频，也就是灰阶值。到了彩色电视显示标准的制定，则是以 YUV 的格式来处理彩色电视图像，把 U 和 V 视作表示彩度的 C，如果忽略 C 信号，则剩下的 Y 信号就跟之前的黑白电视信号相同，这样一来便解决了彩色电视机与黑白电视机的兼容问题。YUV 最大的优点在于只需占用极小的带宽，因为人眼对亮度敏感，而对色度不敏感，因此减少部分 U 和 V 的数据量，对人眼来说感知不到。YUV 的每个分量如下。

(1) Y：亮度，也就是灰度值。除了表示亮度信号外，还含有较多的绿色通道量。

(2) U：蓝色通道与亮度的差值。

(3) V：红色通道与亮度的差值。

3. HSV

RGB 和 YUV 颜色模型都是面向硬件的，而 HSV(Hue Saturation Value)颜色模型则是面向用户的。HSV 颜色模型的三维表示从 RGB 立方体演化而来。设想从 RGB 沿立方体对角线的白色顶点向黑色顶点观察，就可以看到立方体的六边形外形。六边形边界表示色调，水平轴表示饱和度，明度沿垂直轴测量。HSV 的每个分量如下。

(1) 色调 H：色调用角度度量，取值范围为 0°～360°，从红色开始按逆时针方向计算，红色为 0°、绿色为 120°、蓝色为 240°，如图 5-10 所示。它们的补色分别是青色(为 180°)、品红(为 300°)和是黄色(为 60°)。

(2) 饱和度 S：饱和度表示颜色接近光谱色的程度。一种颜色可以看成某种光谱色与白色混合的结果，其中光谱色所占的比例愈大，颜色接近光谱色的程度就愈高，颜色的饱和度也就愈高。饱和度高，颜色则深而艳。光谱色的白光成分为 0，饱和度达到最高。通常取值范围为 0%～100%，值越大，颜色越饱和。

(3) 明度 V：明度表示颜色明亮的程度，对于光源色，明度值与发光体的光亮度有关。对于物体色，此值和物体的透射比或反射比有关。通常取值范围为 0%(黑)～100%(白)。HSV 的模型结构如图 5-11 所示。

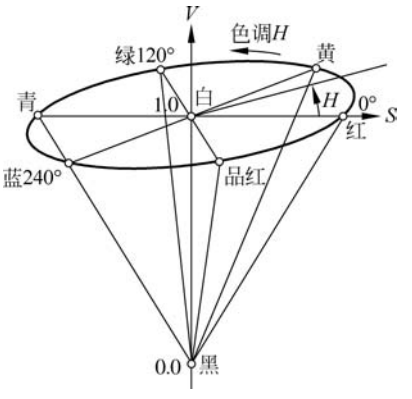


图 5-10　HSV 的色调 H 用角度度量

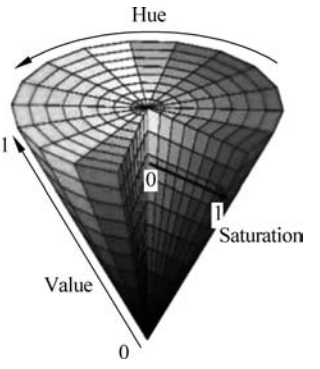


图 5-11　HSV 的模型结构

5.1.3 数字视频

数字视频就是以数字形式记录的视频，是和模拟视频相对的。数字视频有不同的产生方式、存储方式和播出方式。例如通过数字摄像机直接产生数字视频信号，存储在数字带、P2卡、蓝光盘或者磁盘上，从而得到不同格式的数字视频，然后通过特定的播放器将视频播放出来。

1. 图像与视频

图像是人对视觉感知的物质的再现。三维自然场景的对象包括深度、纹理和亮度信息。二维图像主要包括纹理和亮度信息。二维图像示意如图5-12所示。

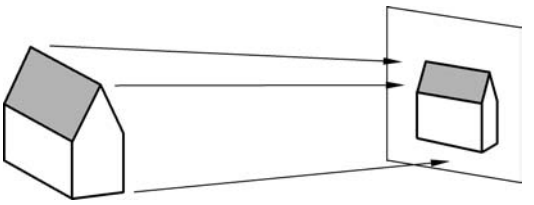


图5-12 二维图像示意图

视频本质上是连续的图像。视频由多幅图像构成，包含对象的运动信息，又称为运动图像。总之，视频是由多幅连续图像组成的，如图5-13所示。

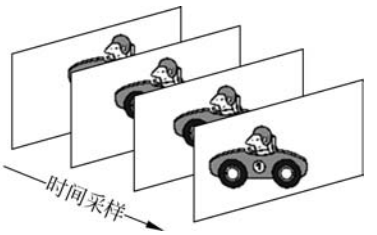


图5-13 视频是由多幅连续图像组成的

2. 数字视频

数字视频可以理解为自然场景空间和时间的数字采样表示。空间采样的主要技术指标为解析度(Resolution)，空间采样与像素的关系如图5-14所示。

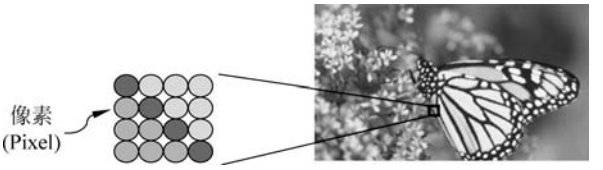


图5-14 空间采样与像素

时间采样的主要技术指标为帧率，时间采样与帧率的关系如图 5-15 所示。

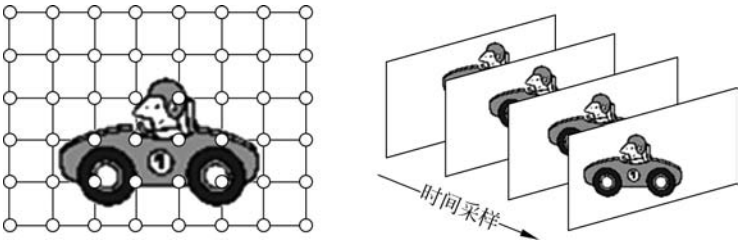


图 5-15　时间采样与帧率

数字视频的系统流程包括采集、处理、显示这 3 个步骤，如图 5-16 所示。

(1) 采集：通常使用照相机或摄像机。

(2) 处理：包括编解码器和传输设备。

(3) 显示：通常用显示器进行数字视频的渲染。

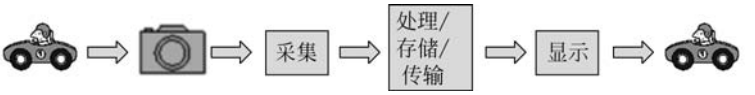


图 5-16　数字视频系统的原理流程示意图

3. HVS

人类视觉系统(Human Visual System，HVS)是由眼睛、神经、大脑构成的。人眼类似于一个光学信息处理系统，但它不是一个普通的光学信息处理系统，从物理结构看，人类视觉系统由光学系统、视网膜和视觉通路等组成。

注意：这里的 HVS 与上文的 HSV 色彩空间是完全不同的两个概念。

人类通过 HVS 获取外界图像信息，当光辐射刺激人眼时，将会引起复杂的生理和心理变化，这种感觉就是视觉。HVS 作为一种图像处理系统，它对图像的认知是非均匀和非线性的。人眼对于图像的视觉特性的特点，主要包括对亮度信号比对色度信号敏感、对低频信号比对高频信号敏感、对静止图像比对运动图像敏感、对图像水平线条和垂直线条比对斜线条敏感等。HVS 的研究包括光学、色度学、视觉生理学、视觉心理学、解剖学、神经科学和认知科学等许多科学领域。

针对 HVS 的特点，数字视频系统的设计应该考虑的因素包括以下几项：

(1) 丢弃高频信息，只编码低频信息。

(2) 提高边缘信息的主观质量。

(3) 降低色度的解析度。

(4) 对感兴趣区域(ROI)进行特殊处理。

在机器视觉、图像处理中，从被处理的图像中以方框、圆、椭圆、不规则多边形等方式勾勒出需要处理的区域，称为感兴趣区域，如图 5-17 所示。

图 5-17　ROI：感兴趣区域

4. YUV 采样格式

YUV 图像可以根据 HVS 的特点，对色度进行分量采样，这样可以降低视频数据量。根据亮度和色度分量的采样比率，YUV 图像的采样格式通常包括 4∶4∶4、4∶2∶2、4∶2∶0，如图 5-18 所示。

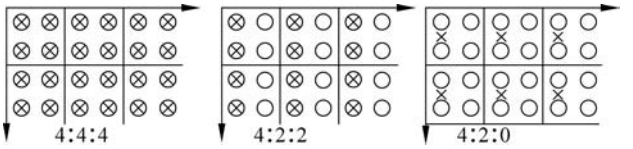


图 5-18　YUV 采样格式

5. YUV 亮度分辨率

根据 YUV 图像的亮度，定义了几种常用的分辨率，如表 5-1 所示。

表 5-1　亮度分辨率

格 式 名 称	亮度分辨率
SQCIF	128×96
QCIF	176×144
CIF	352×288
4CIF	704×576
SD	720×576
HD	1280×720

5.1.4　视频的基础概念

根据人眼视觉暂留原理，每秒超过 24 帧的图像变化看上去是平滑连续的，这样的连续画面叫视频。所谓视频其实就是由很多的静态图片组成的。由于人类眼睛的特殊结构，在画面快速切换时，画面会有残留，所以静态图片快速切换的时候感觉起来就是连贯的动作，这就是视频的基本原理。

1. 帧

帧(Frame)就是视频或者动画中的每张画面，而视频和动画特效就是由无数张画面组

合而成的，每张画面都是一帧。视频帧又分为 I 帧、P 帧和 B 帧。I 帧，即帧内编码帧，大多数情况下 I 帧就是关键帧，也就是一个完整帧，无须任何辅助就能独立完整显示的画面。P 帧，即前向预测编码帧，是一个非完整帧，通过参考前面的 I 帧或 P 帧生成画面。B 帧，即双向预测编码帧，参考前后图像帧编码生成，通过参考前面的 I/P 帧或者后面的 P 帧来协助形成一个画面。只有 I 帧和 P 帧的视频序列，如 I1P1P2P3P4I2P5P6P7P8。包括 I 帧、P 帧和 B 帧的序列，如 I1P1P2B1 P3P4B2 P5I2B3 P6P7。

注意：解码器是有缓存的，通常以 GOP 为单位，所以 B 帧可以参考其后续的 P 帧。

2. 帧和场

平常人们看的电视采用的是每秒播放 25 帧，即每秒更换 25 张图像，由于视觉暂留效应，所以人眼不会感到闪烁。每帧图像又是分为两场进行扫描的，这里的扫描是指电子束在显像管内沿水平方向一行一行地从上到下扫描，第一场先扫奇数行，第二场扫偶数行，即常说的隔行扫描，扫完两场即完成一帧图像。当场频为 50Hz、帧频为 25Hz 时，奇数场和偶数场扫描的是同一帧图像，除非图像静止不动，否则相邻两帧图像不同。计算机显示器与电视机显像管的扫描方式是一样的，所以一帧图像包括两场，即顶场和底场，如图 5-19 所示。

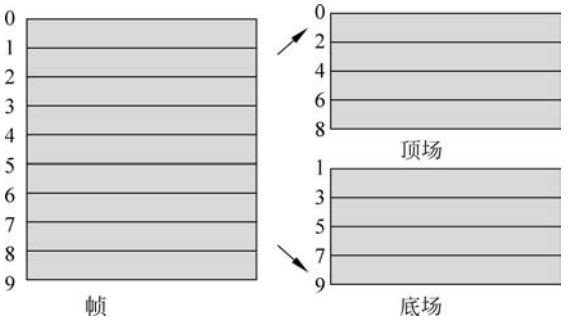


图 5-19　帧与顶场、底场

3. 逐行与隔行扫描

每帧图像由电子束顺序地一行接着一行连续扫描而成，这种扫描方式称为逐行扫描。把每帧图像通过两场扫描完成则是隔行扫描，在两场扫描中，第一场（奇数场）只扫描奇数行，依次扫描 1、3、5 行等，而第二场（偶数场）只扫描偶数行，依次扫描 2、4、6 行等。隔行扫描技术在传送信号带宽不够的情况下起了很大作用。逐行扫描和隔行扫描的显示效果主要区别在稳定性上面，隔行扫描的行间闪烁比较明显，逐行扫描克服了隔行扫描的缺点，画面平滑自然无闪烁。在电视的标准显示模式中，i 表示隔行扫描，p 表示逐行扫描。

逐行扫描图像是指一帧图像的两场在同一时间得到，ttop＝tbot。隔行扫描图像是指一帧图像的两场在不同时间得到，ttop≠tbot。逐行与隔行扫描图像如图 5-20 所示。

4. 帧数

帧（Frames）即帧的数量，可以解释为静止画面的数量。

图 5-20　逐行扫描与隔行扫描

5. 帧率

帧率(Frame Rate)是用于测量显示帧数的量度,单位为 f/s(Frames Per Second)或赫兹(Hz),即每秒显示的帧数。帧率越高,画面越流畅、逼真,对显卡的处理能力要求也越高,数据量也越大。前面提到每秒超过 24 帧的图像变化看上去是平滑连续的,这是针对电影等视频而言的,但是对游戏来讲 24f/s 是不流畅的。常见的帧率包括以下几种。

(1) 10～12f/s:由于人类眼睛的特殊生理结构,如果所看画面的帧率高于每秒 10～12 帧的时候,就会认为是连贯的,此现象称为视觉暂留。

(2) 24f/s:一般电影的拍摄及播放帧数是每秒 24 帧。

(3) 60f/s:这个帧率对人眼识别来讲已经具备较高的平滑度。

(4) 85f/s:人类大脑处理视频的极限,人眼无法分辨更高频率的差异。

注意:在做页面性能优化时,常用 60f/s 作为一个基准,所以需要尽量让每帧的渲染控制在 16ms 内,这样才能达到一秒 60 帧的流畅度。

6. 刷新率

屏幕每秒画面被刷新的次数,分为垂直刷新率和水平刷新率,一般提到的刷新率是指垂直刷新率,以赫兹(Hz)为单位,刷新率越高,图像就越稳定,图像显示也就越自然清晰。目前,大多数显示器根据其设定按 30Hz、60Hz、120Hz 或者 144Hz 的频率进行刷新。其中最常见的刷新频率是 60Hz,这样做是为了继承以前电视机刷新频率为 60Hz 的设定。

7. 分辨率

分辨率即视频、图片的画面大小或尺寸。分辨率是以横向和纵向的像素数量来衡量的,表示平面图像的精细程度。视频精细程度并不只取决于视频分辨率,还取决于屏幕分辨率。例如 1080P 的 P 指逐行扫描(Progressive Scan),即垂直方向像素,也就是“高”,所以 1920×1080 叫 1080P,而不叫 1920P。当 720P(1280×720P)的视频在 1080P 屏幕上播放时,需要将图像放大,放大操作也叫上采样。

8. 码率

码率即比特率,是指单位时间内播放连续媒体(如压缩后的音频或视频)的比特数量。在不同领域有不同的含义,在多媒体领域,指单位时间播放音频或视频的比特数,可以理解成吞吐量或带宽。码率的单位为 b/s,即每秒传输的数据量,常用单位有 b/s、kb/s 等。比特率越高,带宽消耗得就越多。通俗一点理解码率就是取样率,取样率越大,精度就越高,

图像质量就越好，但数据量也越大，所以要找到一个平衡点，用最低的比特率达到最小的失真。

在一个视频中，不同时段画面的复杂程度是不同的，例如高速变化的场景和几乎静止的场景所需的数据量也是不同的，若都使用同一种比特率是不太合理的，所以引入了动态比特率。动态比特率(Variable Bit Rate，VBR)，是指比特率可以随着图像复杂程度的不同而随之变化。图像内容中简单的片段采用较小的码率，图像内容中复杂的片段采用较大的码率，这样既保证了播放质量，又兼顾了数据量的限制。例如 RMVB 视频文件，其中的 VB 就是指 VBR，表示采用动态比特率编码方式，达到播放质量与数据量兼得的效果。同理，静态比特率(Constant Bit Rate，CBR)是指比特率恒定，此时图像内容复杂的片段质量不稳定，图像内容简单的片段质量较好。除 VBR 和 CBR 外，还有 CVBR(Constrained Variable Bit Rate)、ABR(Average Bit Rate)等。

9. CPU & GPU

中央处理器(Central Processing Unit，CPU)，包括算术逻辑部件(Arithmetic Logic Unit，ALU)和高速缓冲存储器(Cache)及实现它们之间联系的数据(Data)、控制及状态的总线(Bus)。

GPU 即图形处理器(Graphics Processing Unit)，专为执行复杂的数学和几何计算而设计的，拥有二维或三维图形加速功能。GPU 相比于 CPU 具有更强大的二维、三维图形计算能力，可以让 CPU 从图形处理的任务中解放出来，执行其他更多的系统任务，这样可以大大提高计算机的整体性能。

硬件加速(Hardware Acceleration)就是利用硬件模块来替代软件算法以充分利用硬件所固有的快速特性，硬件加速通常比软件算法的效率要高。例如将与二维、三维图形计算相关的工作交给 GPU 处理，从而释放 CPU 的压力，是属于硬件加速的一种。

5.1.5 视频格式

视频格式非常多，包括视频文件格式、视频封装格式、视频编码格式等。例如常见的视频文件格式有 MP4、RMVB、MKV、AVI 等，常见的视频编码格式有 MPEG-4、H.264、H.265 等。下面详细介绍这 3 个概念。

1. 视频文件格式

Windows 系统中的文件名都有后缀，例如 1.doc、2.wps、3.psd 等。Windows 设置后缀名的目的是为了让系统中的应用程序来识别并关联这些文件，让相应的文件由相应的应用程序打开。例如双击 1.doc 文件，它会知道让 Microsoft Office 去打开，而不会用 Photoshop 去打开这个文件。常见的视频文件格式如 1.avi、2.mpg，这些都叫作视频的文件格式，它与计算机上安装的视频播放器关联，通过后缀名来判断格式。

2. 视频封装格式

视频封装格式(也叫容器)就是将已经编码并压缩好的视频轨和音频轨按照一定的格式放到一个文件中，也就是说仅是一个外壳，或者把它当成一个放视频轨和音频轨的文件夹也

可以。说得通俗点，视频轨相当于饭，而音频轨相当于菜，封装格式相当于一个碗，或者一个锅，是一个用来盛放饭菜的容器。例如 AVI、MPEG、VOB 是视频封装格式，相当于存储视频信息的容器。视频封装格式一般是由相应的公司开发出来的，可以在计算机上看到的 1.avi、2.mpg、3.vob 等这些视频文件格式的后缀名，即采用相应的视频封装格式的名称。常用的视频封装格式与对应的视频文件格式如表 5-2 所示。

表 5-2 常用的视频封装格式与对应的视频文件格式

视频封装格式名称	视频文件格式
AVI(Audio Video Interleave)	.avi
WMV(Windows Media Video)	.wmv
MPEG(Moving Picture Experts Group)	.mpg/.vob/.dat/.mp4
Matroska	.mkv
Real Video	.rm
QuickTime	.mov
Flash Video	.flv

下面介绍几种常用的视频封装格式。

(1) 音频视频交错(Audio Video Interleaved，AVI)格式，后缀为.avi，于 1992 年被微软公司推出。这种视频格式的优点是图像质量好。由于无损 AVI 可以保存 alpha 通道，所以经常被使用。但缺点太多，例如体积过于庞大，而且压缩标准不统一。

(2) DV 格式(Digital Video Format)，是由索尼、松下、JVC 等多家厂商联合提出的一种家用数字视频格式。数字摄像机就是使用这种格式记录视频数据的。它可以通过计算机的 IEEE 1394 端口将视频数据传输到计算机，也可以将计算机中编辑好的视频数据回录到数码摄像机中。这种视频格式的文件扩展名也是.avi。电视台采用录像带记录模拟信号，通过 EDIUS 由 IEEE 1394 端口采集卡从录像带中采集出来的视频就是这种格式。

(3) MOV 格式是美国苹果公司开发的一种视频格式，默认的播放器是苹果的 QuickTime，具有较高的压缩比率和较完美的视频清晰度等特点，并可以保存 alpha 通道。

(4) MPEG 格式，文件后缀可以是.mpg、.dat、.vob、.mp4 等。制定该标准的专家组于 1988 年成立，专门负责为 CD 建立视频和音频标准，而成员都是视频、音频及系统领域的技术专家。MPEG 文件格式是运动图像压缩算法的国际标准。MPEG 格式目前有 3 个压缩标准，分别是 MPEG-1、MPEG-2 和 MPEG-4。MPEG-1、MPEG-2 目前已经使用较少，下面着重介绍 MPEG-4，其制定于 1998 年，MPEG-4 是为了播放流式媒体的高质量视频而专门设计的，以求使用最少的数据获得最佳的图像质量。目前 MPEG-4 最有吸引力的地方在于它能够保存接近于 DVD 画质的小型视频文件。读者可能注意到了，怎么没有 MPEG-3 编码呢？因为这个项目原本目标是为高分辨率电视(HDTV)而设计的，随后发现 MPEG-2 已足够满足 HDTV 应用，故 MPEG-3 的研发便终止了。

(5) WMV(Windows Media Video)格式，后缀为.wmv 或.asf，也是微软公司推出的一种采用独立编码方式并且可以直接在网上实时观看视频节目的文件压缩格式。WMV 格式

的主要优点包括本地或网络回放、丰富的流间关系及扩展性等。WMV 格式在网站上播放时，需要安装 Windows Media Player(简称 WMP)，很不方便，现在已经几乎没有网站采用这种格式了。

(6) Flash Video 格式，是由 Adobe Flash 延伸出来的一种流行网络视频封装格式，后缀为.flv。随着 H5 视频标准的普及，Flash 正在逐步被淘汰。

(7) Matroska 格式，是一种新的多媒体封装格式，后缀为.mkv。这种封装格式可把多种不同编码的视频及 16 条或以上不同格式的音频和不同语言的字幕封装到一个 Matroska Media 文档内。它也是其中一种开放源代码的多媒体封装格式，还可以提供非常好的交互功能，比 MPEG 更方便、更强大。

3. 视频编码格式

视频编码格式是指能够对数字视频进行压缩或者解压缩的程序或者设备，也可以指通过特定的压缩技术，将某种视频格式转换成另一种视频格式。通常这种压缩属于有损数据压缩。即使是同一种视频文件格式，如 *.MPG，又分为 MPEG-1、MPEG-2、MPEG-4 几种不同的视频封装格式；就算是同一种视频封装格式，如 MPEG-4，又可以使用多种视频编码方式。视频的编码格式才是一个视频文件的本质所在，不要简单地通过文件格式和封装形式来区分视频。常见的视频编码格式有几大系列，包括 H.26X 系列、MPEG 系列及其他系列。

(1) H.26x 系列是由 ITU 主导的，主要包括 H.261、H.262、H.263、H.264、H.265 等。H.261 主要在较早的视频会议和视频电话产品中使用。H.263 主要用在视频会议、视频电话和网络视频上。H.264 即 H.264/MPEG-4 第十部分，或称 AVC，是一种视频压缩标准，也是一种被广泛使用的高精度视频的录制、压缩和发布格式。H.265 及高效率视频编码是一种视频压缩标准，是 H.264/MPEG-4 AVC 的继任者。HEVC 被认为不仅提升了图像质量，同时也能达到 H.264/MPEG-4 AVC 的 2 倍之压缩率，等同于同样画面质量下比特率减少了 50%，可支持 4K 分辨率甚至超高画质电视，最高分辨率可达到 8192×4320(8K 分辨率)，这是目前发展的趋势。

(2) MPEG 系列是由 ISO 下属的 MPEG 开发的，视频编码方面主要包括几部分。MPEG-1 第二部分主要使用在 VCD 上，有些在线视频也使用这种格式，该编解码器的质量大致上和原有的 VHS 录像带相当。MPEG-2 第二部分(等同于 H.262)，使用在 DVD、SVCD 和大多数数字视频广播系统和有线分布系统中。MPEG-4 第二部分可以使用在网络传输、广播和媒体存储上，比起 MPEG-2 和第一版的 H.263，它的压缩性能有所提高。MPEG-4 第十部分和 ITU-T 的 H.264 采用的是相同的标准。这两个编码组织合作，诞生了 H.264/AVC 标准。ITU-T 给这个标准命名为 H.264，而 ISO/IEC 称它为 MPEG-4 高级视频编码。

(3) 其他系列的视频编码格式包括 AMV、AVS、Bink、CineForm、Cinepak、Dirac、DV、RealVideo、RTVideo、SheerVideo、Smacker、Sorenson Video、VC-1、VP3、VP6、VP7、VP8、VP9、WMV 等，因为这些编码方式不常用，此处不详细介绍。

5.2 音视频封装

常见的 AVI、RMVB、MKV、ASF、WMV、MP4、3GP、FLV 等其实只能算是一种封装标准。一个完整的视频文件是由音频和视频两部分组成的，例如 H.264、Xvid 等就是视频编码格式，MP3、AAC 等就是音频编码格式。例如将一个 Xvid 视频编码文件和一个 MP3 音频编码文件按 MP4 封装标准进行封装以后，就可以得到一个 MP4 为后缀的视频文件，这个就是常见的 MP4 视频文件。

5.2.1 数据封装和解封装

数据封装(Data Encapsulation)就是把业务数据映射到某个封装协议的净荷中，然后填充对应协议的包头，就可以形成封装协议的数据包，并完成速率适配。数据解封装就是封装的逆过程，拆解协议包，处理包头中的信息，取出净荷中的业务信息数据封装，这与封装是一对逆过程。数据的封装与解封装如图 5-21 所示。

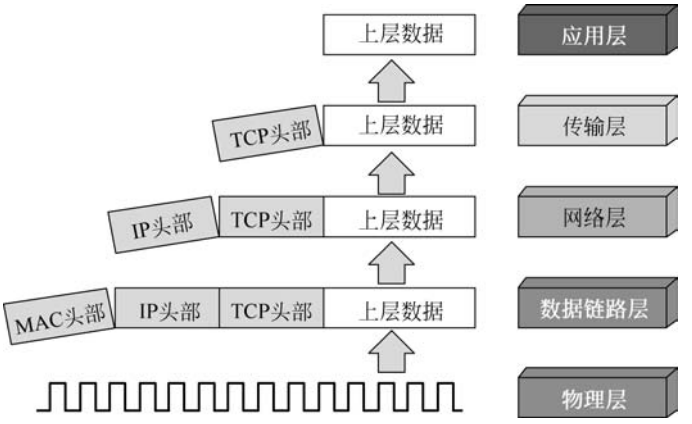


图 5-21 数据封装与解封装

5.2.2 音视频的封装

对于任何一部视频来讲，如果只有图像，而没有声音，肯定是不行的，所以视频编码后，需要加上音频编码，然后一起进行封装。封装是指封装格式，简单来讲就是将已经编码压缩好的视频轨和音频轨按照一定的格式放到一个文件中。再通俗点讲，视频轨相当于饭，而音频轨相当于菜，封装格式相当于一个饭盒，是用来盛放饭菜的容器，如图 5-22 所示。目前市面上主要的视频容器分为 MPG、VOB、MP4、3GP、ASF、RMVB、WMV、MOV、Divx、MKV、FLV、TS/PS 等。封装之后的视频，就可以传输了，也可以通过视频播放器进行解码观看。

封装格式也称多媒体容器，它只是为多媒体编码提供了一个“外壳”，也就是将所有的处理好的视频、音频或字幕都包装到一个文件容器内以便呈现给观众，这个包装的过程就叫封

装，如图 5-23 所示。其实封装就是按照一定规则把音视频、字幕等数据组织起来，包含编码类型等公共信息，播放器可以按照这些信息来匹配解码器、同步音视频。

图 5-22　音视频的容器封装格式与编码方式

图 5-23　音视频的封装

5.2.3　封装格式

封装格式即音视频容器，例如经常看到的视频后缀名. mp4、. rmvb、. avi、. mkv、. mov 等，这些就是音视频的容器，它们将音频和视频甚至字幕一起打包进去，封装成一个文件，用来存储或传输编码数据，可以理解成一个容器。

所谓封装格式，就是以怎样的方式将视频轨、音频轨、字幕轨等信息组合在一起。不同的封装格式所支持的视频、音频编码格式是不一样的，例如 MKV 格式支持得比较多，RMVB 则主要支持 Real 公司的视频、音频编码格式。常见的封装格式包括 AVI、VOB、WMV、RM、RMVB、MOV、MKV、FLV、MP4、MP3、WebM、DAT、3GP、ASF、MPEG、OGG 等。视频文件的封装格式并不影响视频的画质，影响视频画面质量的是视频的编码格式。一个完整的视频文件是由音频和视频两部分(有的也包括字幕)组成的。

MPG 是 MPEG 编码所采用的容器，具有流的特性，里面又分为 PS 和 TS，PS 主要用于 DVD 存储，TS 主要用于 HDTV。VOB 是 DVD 采用的容器格式，支持多视频多音轨多字幕等。MP4 是 MPEG-4 编码所采用的容器，基于 QuickTime MOV 开发，具有许多先进特性。AVI 是音视频交互存储，是最常见的视频音频容器，支持的视频音频编码也是最多的。ASF 是 Windows Media 所采用的容器，能够用于流传送，还能包容脚本等。3GP 是 3GPP 视频所采用的格式，主要用于流媒体传送。RM 是 RealMedia 所采用的容器，用于流传送。

MOV 是 QuickTime 所采用的容器，几乎是现今最强大的容器，甚至支持虚拟现实技术、Java 等，它的变种 MP4、3GP 的功能都没这么强大。MKV 能把 Windows Media Video、RealVideo、MPEG-4 等视频音频融为一个文件，而且支持多音轨，支持章节、字幕等。OGG 是 Ogg 项目所采用的容器，具有流的特性，支持多音轨、章节、字幕等。OGM 是 OGG 容器的变种，能够支持基于 DirectShow 的视频音频编码，支持章节等特性。WAV 是一种音频容器，常说的 WAV 就是没有压缩的 PCM 编码，其实 WAV 中还可以包括 MP3 等其他 ACM 压缩编码。下面重点介绍几种常见的音视频封装格式。

1. MP4

MP4(MPEG-4 Part 14)是一种常见的多媒体容器格式，它是在 ISO/IEC 14496-14 标准文件中定义的，属于 MPEG-4 的一部分。MP4 是一种较为全面的容器格式，被认为可以在其中嵌入任何形式的数据，不过常见的大部分的 MP4 文件存放的是 AVC(H.264)或 MPEG-4 Part 2 编码的视频和 AAC 编码的音频。MP4 格式的官方文件后缀名是.mp4，还有其他的以 MP4 为基础进行的扩展格式，如 M4V、3GP、F4V 等。

1) box 结构树

MP4 文件中所有数据都装在 box 中，也就是说 MP4 由若干 box 组成，每个 box 有类型和长度，包含不同的信息，可以将 box 理解为一个数据对象块。box 中可以嵌套另一个 box，这种 box 称为 container box。MP4 文件 box 以树状结构的方式组织，一个简单的 MP4 文件由以下 box 组成(可以使用 mp4info 工具查看 MP4 文件结构)，MP4 文件的 box 树状结构，如图 5-24 所示。根节点(ROOT)之下，主要包含以下 3 个 box 节点，即 ftyp(File Type Box)文件类型、moov(Movie Box)文件媒体信息和 mdat(Media Data Box)媒体数据。

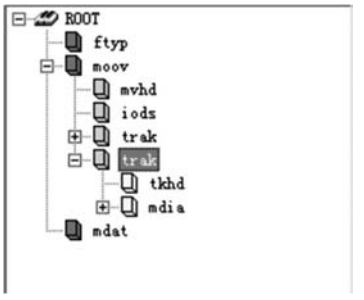


图 5-24　MP4 的 box 树状结构图

2) ftyp

一个 MP4 文件有且仅有一个 ftyp 类型的 box，作为 MP4 格式的标识并包含一些关于文件的信息。ftyp 是 MP4 文件的第 1 个 box，包含了视频文件使用的编码格式、标准等。ftyp box 通常放在文档的开始，通过对该 box 解析可以让软件(播放器、demux、解析器)知道应该使用哪种协议对该文档解析，这是后续解读的基础。下面是一段定义 MP4 文件头的结构体。

```
typedef struct{
    unsigned int length;                          //box长度为28,包括这4字节本身
    unsigned char name[4];                        //4个字符:ftyp
    unsigned char majorBrand[4];
    unsigned int minorVersion;
    unsigned char compatibleBrands[12];
}FtypBox;
```

下面是一段十六进制的 MP4 文件数据。

```
00 00 00 1C 66 74 79 70 6D 70 34 32 00 00 00 01          ; …ftypmp42…
6D 70 34 31 6D 70 34 32 69 73 6F 6D                      ; mp41mp42isom
```

具体解析如下。

(1) 00 00 00 1C,即 box 长度为 28,包括这 4 字节本身和后面的 24b 的 box body。

(2) 66 74 79 70,即 ftyp,代表 box type。

(3) 6D 70 34 32,即 mp42,代表 major brand。

(4) 00 00 00 01,即 1,代表 minor version。

(5) 6D 70 34 31 6D 70 34 32 69 73 6F 6D,以及 Compatible brands、mp41(0X6D703431)、mp42(0X6D703432)、isom(0X69736F6D),说明本文档遵从(或称兼容)MP41、MP42、ISON 这 3 种协议。

3) moov

ftyp box 之后会有一个 moov 类型的 box,是一种 container box,子 box 中包含了媒体的 metadata 信息。该 box 包含了文档媒体的 metadata 信息,moov 是一个 container box,具体内容信息由子 box 诠释。同 File Type Box 一样,该 box 有且只有一个,并且只被包含在文档层。一般情况下,moov 会紧随 ftyp 出现,moov 中会包含一个 mvhd 和两个 trak(一个音频和一个视频)。其中 mvhd 为 header box,一般作为 moov 的第一个子 box 出现(对于其他 container box 来讲,header box 都应作为首个子 box 出现)。trak 包含了一个 track 的相关信息,是一个 container box。

4) mdat

MP4 文件的媒体数据包含在 mdat 类型的 box(Media Data Box)中,该类型的 box 也是 container box,可以有多个,也可以没有(当媒体数据全部引用其他文件时),媒体数据的结构由 metadata 进行描述。

2. AVI

音频视频交错(Audio Video Interleaved,AVI)格式,是一门成熟的老技术,尽管国际学术界公认 AVI 已经属于被淘汰的技术,但是简单易懂的开发 API,还在被广泛使用。AVI 符合 RIFF(Resource Interchange File Format)文件规范,使用四字符码(Four-Character Code,FOURCC)表征数据类型。AVI 的文件结构分为头部、主体和索引 3 部分。主体中图

像数据和声音数据是交互存放的，从尾部的索引可以索引到想放的位置。AVI 本身只提供了这么一个框架，内部的图像数据和声音数据格式可以是任意的编码形式。因为索引放在了文件尾部，所以在播放网络流媒体时已力不从心。例如从网络上下载 AVI 文件，如果没有下载完成，则很难正常播放出来。

1）基本数据单元

AVI 中有两种最基本的数据单元，一个是 Chunks，另一个是 Lists，结构体如下：

```
//Chunks
typedef struct{
    DWORD dwFourcc;
    DWORD dwSize;                    //data
    BYTE data[dwSize];               //contains headers or audio/video data
} CHUNK;

//Lists
typedef struct {
    DWORD dwList;
    DWORD dwSize;                    //dwFourCC + data
    DWORD dwFourCC;
    BYTE data[dwSize - 4];           //contains Lists and Chunks
} LIST;
```

由如上代码可知，Chunks 数据块由一个四字符码、4B 的 data size(指下面的数据大小)及数据组成。Lists 由 4 部分组成，包括 4B 的四字符码、4B 的数据大小(指后面列的两部分数据大小)、4B 的 list 类型及数据组成。与 Chunks 数据块不同的是，Lists 数据内容可以包含字块(Chunks 或 Lists)。

2）RIFF 简介

RIFF 是一种复杂的格式，其目的在于适用于多媒体应用的各种类型的数据。RIFF 是微软公司为了 Windows GUI 而创建的一种多媒体文件格式。RIFF 本身并没有定义任何存储数据的新方法，但是 RIFF 定义了一个结构化的框架，它包含了现有的数据格式。由此可知，用户可以创建由两种或更多现有文件格式组成的新的复合格式。

多媒体应用需要存储和管理各种数据，包括位图、音频数据、视频数据和外围设备控制信息。RIFF 提供了一种很好的方式来存储所有这些不同类型的数据。RIFF 文件中包含的数据类型可以通过文件的后缀名来确认，例如常见的文件后缀名包括. avi、. wav、. rdi、. rmi 和. bnd 等。

注意：AVI 是现在 RIFF 规范中使用最全面的一种类型，WAV 虽然也常被使用，但是非常简单，WAV 开发者通常使用旧的规范来构建它们。

因为 RIFF 涉及各种多媒体文件的名字，所以 RIFF 经常被它所包含的媒体文件所描述，例如 AVI 格式，而不是作为一种格式名字，因此对于刚开始接触 RIFF 文件的用户来讲特别容易混淆这些概念。例如包含音频和视频数据的文件一般称为 AVI 文件，而不是

RIFF 文件。只有编程开发人员才会知道或留意其实 AVI 文件和 RIFF 文件是同一种文件格式。另外一个问题就是，很容易将 RIFF 和 TIFF(Tag Image File Format)文件弄混淆。这两种格式都可以添加或删除文件的数据结构，但是它们内部的数据结构却相差非常大。与 TIFF 不同的是 RIFF 文件格式是基于 Electronic Arts 交换文件格式(IFF)的结构。虽然这两种格式都使用相同的数据存储概念，但它们在设计上却是不兼容的。

RIFF 是一种包含多个嵌套数据结构的二进制文件格式，RIFF 文件中的每个数据结构都称为块。块在 RIFF 文件中没有固定位置，因此标准偏移值不能用于定位它们的字段，换句话说，块是由用户来定义的，所以每个块没有统一的位置。块包含数据，如数据结构、数据流或称为子块的其他块。每个 RIFF 块都具有以下基本结构：

```
typedef struct _Chunk
{
    DWORD ChunkId;                      /* Chunk ID marker */
    DWORD ChunkSize;                    /* Size of the chunk data in Bytes */
    BYTE ChunkData[ChunkSize];          /* The chunk data */
} CHUNK;
```

注意：RIFF 是以小端字节顺序写入的。字节顺序指占内存多于一个字节类型的数据在内存中的存放顺序，通常有小端、大端两种字节顺序。小端字节顺序指低字节数据存放在内存低地址处，高字节数据存放在内存高地址处；大端字节顺序指高字节数据存放在低地址处，低字节数据存放在高地址处。基于 x86 平台的 PC 是小端字节顺序的。

其中 ChunkId 包含 4 个 ASCII 字符，用于标识块包含的数据。例如字符 RIFF 用于识别包含 RIFF 数据的块。如果 ID 小于 4 个字符，则使用空格(ASCII 32)在右侧填充。ChunkSize 表示存储在 ChunkData 字段中的数据的长度，不包括添加到数据中的任何填充数据(为了数据对齐会在结尾添加 0，但是添加的 0 不算长度)。

注意：结构体中 ChunkId 的 4B 和 ChunkSize 本身的 4B 的长度均不在 ChunkSize 的数据长度中。

ChunkData 中的数据是以 4B 对齐的，如果实际数据是奇数，就在数据末尾添加 0，ChunkSize 的值是不包含末尾为了对齐所添加的 0 的。子块(Subchunk)也具有与块(Chunk)相同的结构，一个子块就是包含在另一个块中的任何块。只有 RIFF 块和列表块(LIST Chunk)包含子块，其他的块都只包含数据。一个 RIFF 文件本身就是一个完整的 RIFF 块，文件中的所有其他块和子块都包含在该块中。如果要读取 RIFF 文件，则应该忽略不能识别的或不使用的块。如果是在写一个 RIFF 文件，则应该将所有未知的或不使用的块全部写入文件中，不要丢掉它们，一般根据不同需求写入。

用于存储音频和视频信息的 RIFF 文件称为 AVI 文件。RIFF AVI 文件格式通常只包含一个 AVI 块，但是其他类型的块也可能出现。解析一个 AVI 文件可以忽略不需要的块，同时要确保文件中存储了一个 AVI 块。虽然微软使用标准的表示法来描述 RIFF 文件内部数据结构，但是使用 C 语法来说明 RIFF AVI 文件中块和子块的位置是更清晰的，示例

码如下：

```
struct _RIFF                                        /* RIFF */
{
    struct _AVICHUNK                                /* AVI */
    {
        struct _LISTHEADERCHUNK                     /* hdrl */
        {
            AVIHEADER AviHeader;                    /* avih */
            struct _LISTHEADERCHUNK                 /* strl */
            {
                AVISTREAMHEADER  StreamHeader;      /* strh */
                AVISTREAMFORMAT  StreamFormat;      /* strf */
                AVISTREAMDATA  StreamData;          /* strd */
            }
        }
        struct _LISTMOVIECHUNK                      /* movi */
        {
            struct _LISTRECORDCHUNK                 /* rec */
            {
                /* Subchunk 1 */
                /* Subchunk 2 */
                /* Subchunk N */
            }
        }
        struct _AVIINDEXCHUNK                       /* idx1 */
        {
            /* Index data */
        }
    }
}
```

以上代码结构表示仅包含一个 AVI 块的 RIFF 文件的内部数据布局，该块遵循先前描述的块数据结构的格式，每个块的 ChunkId 已经在注释中列出。AVI 块由 4 个字符的块标识符 AV 标识(注意最后的空白字符)。AVI 块包含两个强制 LIST 子块(hdrl 和 movi 这两个 LIST 是必须有的)，它们指示存储在文件中的数据流的格式和数据。

3) AVI 文件结构

AVI 文件采用 RIFF 文件结构方式，使用四字符码(FOURCC)表征数据类型，例如 RIFF、AVI、LIST 等，通常将四字符码称为数据块 ID。RIFF 文件的基本单元叫作数据块(Chunk)，由数据块四字符码(数据块 ID)、数据长度、数据组成。整个 RIFF 文件可以看成一个数据块，其数据块 ID 为 RIFF，称为 RIFF 块，一个 RIFF 文件中只允许存在一个 RIFF 块。RIFF 块中包含一系列其他子块，其中 ID 为 LIST，称为 LIST 块，LIST 块中可以再包含一系列其他子块，但除了 LIST 块外的其他所有的子块都不能再包含子块。

下面介绍一下标准 RIFF 文件结构，如图 5-25 所示，一个 AVI 通常包含几个子块。ID 为 hdrl 的 LIST 块，包含了音视频信息，以及描述媒体流信息。ID 为 info 的 LIST 块，包含编码该 AVI 的程序信息。ID 为 junk 的 chunk 数据块，是无用数据，用于填充 ID 为 movi 的 LIST 块，包含了交错排列的音视频数据。ID 为 idxl 的 Chunk 块，包含音视频排列的索引数据。

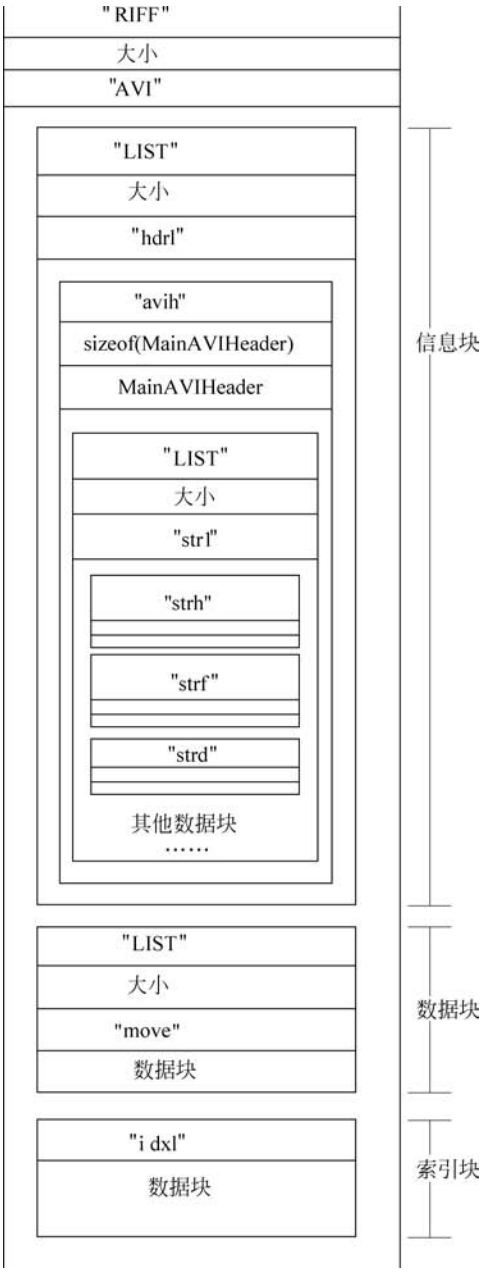


图 5-25　AVI 文件的 RIFF 结构

3. FLV

FLV(Flash Video)是现在非常流行的流媒体格式,由于其视频文件占用空间较小、封装及播放简单等特点,使其很适合在网络上进行应用,目前主流的视频网站无一例外地使用了 FLV 格式,但是当前浏览器已经不提倡使用 Flash 插件了,通过 video.js 和 flv.js 可以扩展 Flash 功能。FLV 是常见的流媒体封装格式,可以将其数据看为二进制字节流。总体上看,FLV 包括文件头(File Header)和文件体(File Body),其中文件体由一系列的 Tag 及 Tag Size 对组成。FLV 格式的 Tag 结构如图 5-26 所示。

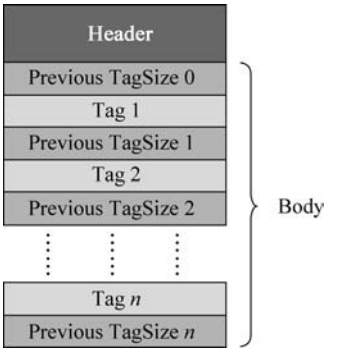


图 5-26 FLV 格式的 Tag 结构

其中,PreviousTagSize 紧跟在每个 Tag 之后,占 4B,表示这是一个 UI32 类型的数值,代表前面一个 Tag 的大小。需要注意的是,PreviousTagSize ≠0 的值总为 0。Tag 类型包括视频、音频和 Script,并且每个 Tag 只能包含一种类型的数据。

FLV 格式的十六进制数据分析主要是 header、body 及 tag。例如下面是一段 FLV 格式的 MV 视频,推荐使用 Binary Viewer 或 Ultra Edit 的二进制查看工具,如图 5-27 所示。

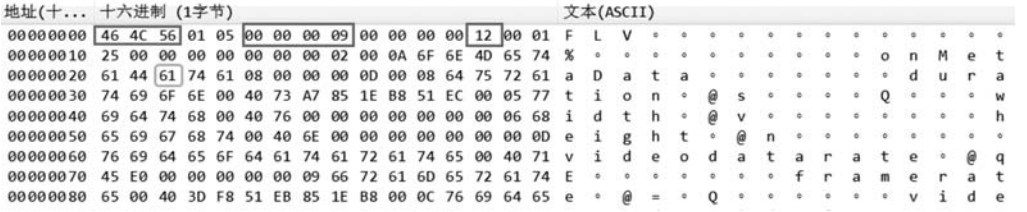


图 5-27 FLV 格式的十六进制数据分析

FLV 的 Header 头部分由几部分组成,分别是 Signature(3B)、Version(1B)、Flags(1B)、DataOffset(4B)。其中 Signature 占 3B,固定为 FLV 这 3 个字符作为标识。一般发现前 3 个字符为 FLV 时就认为是 FLV 文件。Version 占 1B,标识 FLV 的版本号,此处为 1。Flags 占 1B,其中第 0 位和第 2 位分别表示 video 与 audio 存在的情况(1 表示存在,0 表示不存在)。如图 5-27 所示,看到的是 0x05,也就是 00000101,代表既有视频,也有音频。DataOffset 占 4B,表示 FLV 的 header 长度,固定为 9B。

FLV 的 body 部分是由一系列的 back-pointers＋tag 构成，back-pointers 固定为 4B，表示前一个 tag 的 size。tag 分为 3 种类型，包括 video、audio 和 scripts。

FLV 的 tag 部分是由 tag type、tag data size、Timestamp、TimestampExtended、stream ID、tag data 组成的。其中 type 占 1B、8 代表 Audio、9 代表 Video、18 代表 scripts。tag data size 占 3B，表示 tag data 的长度，从 stream ID 后算起。Timestreamp 占 3B，即时间戳。TimestampExtended 占 1B，即时间戳扩展字段。stream ID 占 3B，总是 0。tag data 是数据部分。

4. TS

MPEG-2-TS 是一种标准容器格式，用于传输与存储音视频、节目与系统信息协议数据，广泛应用于数字广播系统，日常数字机顶盒接收的就是 TS 流。在 MPEG-2 标准中，有两种不同的码流输出到信道，一种是 PS 流，适用于没有传输误差的场景。另一种是 TS 流，适用于有信道噪声的传输场景。节目流设计用于合理可靠的媒体，如光盘（如 DVD），而传输流设计用于不太可靠的传输，即地面或卫星广播。此外，传输流可以携带多个节目。MPEG-2 System（编号 13818-1）是 MPEG-2 标准的一部分，该部分描述了多个视频、音频和数据多种基本流（ES）合成传输流（TS）和节目流（PS）的方式。

1）基本概念

首先需要分辨与 TS 传输流相关的几个基本概念。

（1）ES 即基本流（Elementary Stream），直接从编码器出来的数据流，可以是编码过的音频、视频或其他连续码流。

（2）PES 即打包的基本流（Packetized Elementary Streams），是 ES 流经过 PES 打包器处理后形成的数据流，在这个过程中完成了将 ES 流分组、加入包头信息（如 PTS、DTS）等操作。PES 流的基本单位是 PES 包，PES 包由包头和负载（Payload）组成。

（3）节目流（Program Stream，PS），由 PS 包组成，而一个 PS 包又由若干 PES 包组成。一个 PS 包由具有同一时间基准的一个或多个 PES 包复合而成。

（4）传输流（Transport Stream，TS），由固定长度（188B）的 TS 包组成，TS 包是对 PES 包的另一种封装方式，同样由具有同一时间基准的一个或多个 PES 包复合而成。PS 包是不固定长度的，而 TS 包为固定长度的。

（5）节目特定信息（Program Specific Information，PSI），用来描述传输流的组成结构。PSI 信息由 4 种类型的表组成，包括节目关联表（PAT）、节目映射表（PMT）、条件接收表（CAT）、网络信息表（NIT）。PAT 与 PMT 两张表帮助找到该传输流中的所有节目与流，PAT 用于指示该 TS 流由哪些节目组成，每个节目的节目映射表 PMT 的 PID 是什么，而 PMT 用于指示该节目由哪些流组成，每一路流的类型与 PID 是什么。

（6）节目关联表（Program Association Table，PAT），该表的 PID 是固定的 0x0000，它的主要作用是指出该传输流的 ID，以及该路传输流中所对应的几路节目流的 MAP 表和网络信息表的 PID。

（7）节目映射表（Program Map Table，PMT），该表的 PID 是由 PAT 提供给出的。通

过该表可以得到一路节目中包含的信息。例如，该路节目由哪些流构成和这些流的类型(视频、音频、数据)，指定节目中各流对应的 PID，以及该节目的 PCR 所对应的 PID。

(8) NIT 即网络信息表(Network Information Table)，该表的 PID 是由 PAT 提供给出的。NIT 的作用主要是对多路传输流的识别，NIT 提供多路传输流、物理网络及网络传输相关的一些信息，如用于调谐的频率信息及编码方式，以及调制方式等参数方面的信息。

(9) 条件访问表(Conditional Access Table，CAT)，该表的 PID 为 0x0001。

PAT 用于描述有多少路节目，每路节目的 PMT 表的 PID 是多少，PMT 则描述了本节目有多少流，每路流的类型与 PID 是多少。举个例子，找到一个 TS 包，它的 PID 是 0，说明它的负载内容是 PAT 信息，解析 PAT 信息，发现节目 1 的 PMT 表的 PID 是 0x10，接着，在比特流中寻找一个 PID 为 0x10 的 TS 包，它的负载内容是节目 1 的 PMT 表信息，解析该 PMT 信息可以发现第一路流是 MPEG-2 音频流，PID 号为 0x21，第二路流是 MPEG-2 视频流，PID 号是 0x22，第三路流是 DVB 字幕流，PID 号是 0x23，解析完毕，凡是比特流中 PID 号为 0x22 的 TS 包，所负载的内容为 MPEG-2 视频流，把这些包一个一个找出来，把其中的有效码流一部分一部分拼接起来，然后送给解码器去解码。注意，就一般的视频流而言，只要拼接成一个完整的 PES 包，就可以送出去给解码器，然后继续拼接下一个 PES 包。

除了上述的几种表外，MPEG-2 还提供了私有字段，用于实现对 MPEG-2 的扩充。为便于传输，实现时分复用，基本流 ES 必须打包，也就是将顺序连续、传输连续的数据流按一定的时间长度进行分割，分割的小段叫作包，因此打包也被称为分组。在 MPEG-2 标准中，有两种不同的码流可以输出到信号，包括 PS 和 TS。PS 流包结构的长度可变，一旦某个 PS 包的同步信息丢失，接收机就无法确认下一个包的同步位置，从而导致信息丢失，因此 PS 流适用于合理可靠的媒体，如光盘，PS 流的后缀名一般为 vob 或 evo，而 TS 传输流不同，TS 流的包结构为固定长度(188B)，当传输误码破坏了某个 TS 包的同步信息时，接收机可在固定的位置检测它后面包中的同步信息，从而恢复同步，避免信息丢失，因此 TS 可适用于不太可靠的传输，即地面或卫星传播，TS 流的后缀一般为 ts、mpg、mpeg。由于 TS 码流具有较强的抵抗传输误码的能力，因此目前在传输媒体中进行传输的 MPEG-2 码流基本上采用 TS。

2) TS 流形成过程

TS 流的形成过程大体上分为 3 个步骤，如图 5-28 所示。

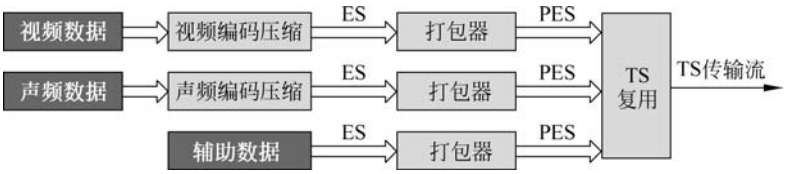


图 5-28　TS 流的形成过程

下面以电视数字信号为例来说明 TS 流的形成过程。第一步通过原始音视频数据经过压缩编码得到 ES 流，生成的 ES 基本流比较大，并且只是 I、P、B 这些视频帧或音频取样信

息。第二步对 ES 基本流进行打包生成 PES 流，通过 PES 打包器，首先对 ES 基本流进行分组打包，在每个包前加上包头就构成了 PES 流的基本单位，即 PES 包，对视频 PES 来讲，一般是一帧一个包，音频 PES 一般一个包不超过 64kB。PES 包头信息中加入了 PTS、DTS 信息，用于音视频的同步。第三步使同一时间基准的 PES 包经过 TS 复用器生成 TS 传输包。

PES 包的长度通常远大于 TS 包的长度，一个 PES 包必须由整数个 TS 包来传送，没装满的 TS 包由填充字节填充。PES 包进行 TS 复用时，往往一个 PES 包会分存到多个 TS 包中。将 PES 包内容分配到一系列固定长度的 TS 传输包(TS Packet)中，TS 流中 TS 传输包头加入了 PCR(节目参考时钟)与 PSI(节目专用信息)，其中 PCR 用于解码器的系统时钟恢复。TS 包的基本结构，如图 5-29 所示。每个包的固定长度为 188B，其中包头为 4B。PCR 时钟的作用非常重要，编码器中有一个系统时钟用于产生指示音视频正确显示和解码的时间标签(如 DTS、PTS)。解码器在解码时首先利用 PCR 时钟重建与编码器同步的系统时钟，再利用 PES 流中的 DTS、PTS 进行音视频的同步，这也是音视频同步的基本原理。

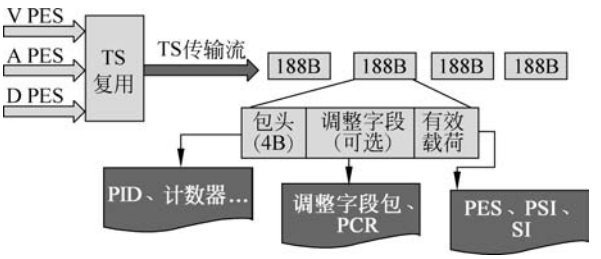


图 5-29　TS 包的基本结构

TS 流的形成过程如下。

(1) 将原始音视频数据压缩之后，由压缩结果组成一个基本码流 ES。

(2) 对 ES 进行打包形成 PES。

(3) 在 PES 包中加入时间戳信息 PTS/DTS。

(4) 将 PES 包内容分配到一系列固定长度的 TS 包中。

(5) 在传输包中加入定时信息 PCR。

(6) 在传输包中加入节目专用信息 PSI。

(7) 连续输出传输包形成具有恒定比特率的 MPEG-TS 流。

TS 流的解析过程，可以说是生成的逆过程如下。

(1) 从复用的 MPEG-TS 流中解析出 TS 包。

(2) 从 TS 包中获取 PAT 及对应的 PMT(PSI 中的表格)。

(3) 从而获取特定节目的音视频 PID。

(4) 通过 PID 筛选出特定音视频相关的 TS 包，并解析出 PES。

(5) 从 PES 中读取 PTS/DTS，并从 PES 中解析出基本码流 ES。

(6) 将 ES 交给解码器，获得压缩前的原始音视频数据。

5. M3U8

M3U8 是 Unicode 版本的 M3U，采用 UTF-8 编码。M3U 和 M3U8 文件都是苹果公司使用的 HTTP Live Streaming（HLS）协议格式的基础，这种协议格式可以在 iPhone 和 Macbook 等设备播放。M3U8 文件其实是 HLS 协议的部分内容，而 HLS 是一个由苹果公司提出的基于 HTTP 的流媒体网络传输协议。

1）HLS 简介

HTTP 直播流协议（HTTP Live Streaming，HLS），其工作原理是把整个流分成一个一个小的基于 HTTP 的文件来下载，每次只下载一部分。当媒体流正在播放时，客户端可以选择从许多不同的备用源中以不同的速率下载同样的资源，允许流媒体会话适应不同的数据速率。在开始一个流媒体会话时，客户端会下载一个包含元数据的 Extended M3U（M3U8）Playlist 文件，用于寻找可用的媒体流。HLS 只请求基本的 HTTP 报文，与实时传输协议（RTP）不同，HLS 可以穿过任何允许 HTTP 数据通过的防火墙或者代理服务器，也可以很容易地使用内容分发网络来传输媒体流。

简而言之，HLS 是新一代流媒体传输协议，其基本实现原理为将一个大的媒体文件进行分片，将该分片文件资源路径记录于 M3U8 文件内，其中附带一些额外描述，例如该资源的多带宽信息，主要用于提供给客户端。客户端依据该 M3U8 文件即可获取对应的媒体资源，进行播放，因此客户端获取 HLS 文件，主要就是对 M3U8 文件进行解析操作。

2）M3U8 文件格式

M3U8 文件实质是一个播放列表（Playlist），其可能是一个媒体播放列表（Media Playlist），也可能是一个主播放列表（Master Playlist）。但无论是哪种播放列表，其内部文字使用的都是 UTF-8 编码。当 M3U8 文件作为媒体播放列表（Media Playlist）时其内部信息记录的是一系列媒体片段资源，顺序播放该片段资源即可完整展示多媒体资源。M3U8 的播放列表内容片段如下。

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXTINF:9.009,
http://media.example.com/first.ts
#EXTINF:9.009,
http://media.example.com/second.ts
#EXTINF:3.003,
http://media.example.com/third.ts
```

对于点播来讲，客户端只需按顺序下载上述片段资源，依次进行播放，而对于直播来讲，客户端需要定时重新请求该 M3U8 文件，看一看是否有新的片段数据需要进行下载并播放。

当 M3U8 作为主播放列表（Master Playlist）时，其内部提供的是同一份媒体资源的多份流列表资源，其格式如下：

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.
40.2"
http://example.com/low/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=240000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.
40.2"
http://example.com/lo_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.
40.2"
http://example.com/hi_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.
40.2"
http://example.com/high/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=64000,CODECS="mp4a.40.5"
http://example.com/audio/index.m3u8
```

该备用流资源指定了多种不同码率，以及不同格式的媒体播放列表，并且该备用流资源也可同时提供不同版本的资源内容，例如不同语言的音频文件、不同角度拍摄的视频文件等。客户可以根据不同的网络状态选取合适码流的资源，并且最好根据用户喜好选择合适的资源内容。

5.3　视频压缩编码

编码这一概念在通信与信息处理领域中被广泛使用，其基本原理是将信息按照一定规则使用某种形式的码流表示与传输。常用的需要编码的信息主要有文字、语音、视频和控制信息等。

5.3.1　视频编码基础知识

视频编码涉及的基础知识非常复杂，包括图像、视频、压缩原理等。

1. 视频和图像的关系

之前已经介绍了像素、图像、视频，本节重点讲解图像和视频的关系。例如把一本动画书放到手里，然后快速翻动，这样就形成了一种动画效果，也就是视频。视频本质上是由大量的连续图片组成的。在视频中，一帧就是指一幅静止的画面。衡量视频很重要的一个指标是帧率，帧率越高，视频就越逼真、越流畅。不同帧率与视频的流畅度如图 5-30 所示。

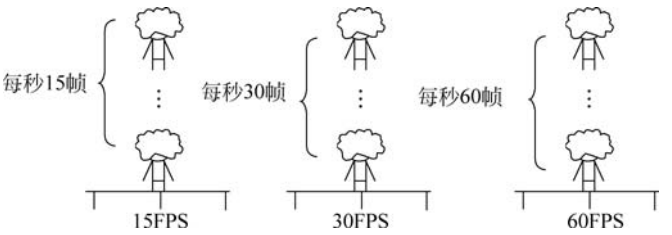


图 5-30　帧率与视频的流畅度

2. 视频编码的理由

视频为什么需要编码？关键就在于一个原始视频，如果未经编码，体积是非常庞大的。以一个分辨率为 1920×1080，并且帧率为 30 的视频为例，共有 1920×1080＝2 073 600 像素，每像素占用 24b(假设采取 RGB24)。也就是每幅图片占用 2 073 600×24＝49 766 400b。8b(位)＝1B(字节)，所以，49 766 400b＝6 220 800B≈6.22MB。这才是一幅 1920×1080 图片的原始大小(6.22MB)，再乘以帧率 30。也就是说，1s 视频的大小是 186.6MB，1 分钟大约是 11GB，一部 90 分钟的电影，约为 1000GB(约 1TB)。

视频产生之后涉及两个问题，包括存储和传输。如果按照 100Mb/s 的网速(12.5MB/s)，下载刚才的那部电影，大约需要 22 小时。为了看一部电影，需要等待 22 小时，这是绝大部分用户所不能接受的。正因为如此，专业的视频工程师就提出，必须对视频进行压缩编码。

3. 视频编码

数据编码是指按指定的方法将信息从一种格式转换成另一种格式。视频编码就是将一种视频格式转换成另一种视频格式。视频编码和解码是互逆的过程，如图 5-31 所示。

图 5-31　视频编解码

编码的终极目的就是为了压缩，市面上各种各样的视频编码方式都是为了让视频变得更小，有利于存储和传输。视频从录制到播放的整个过程，如图 5-32 所示。

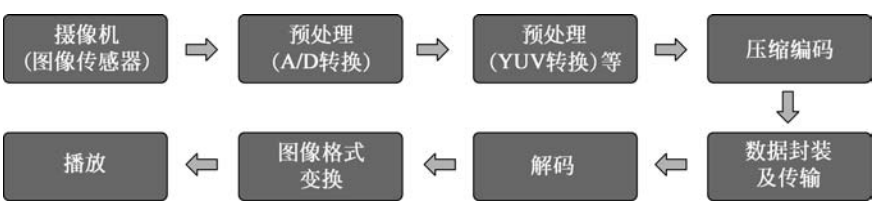


图 5-32　视频从录制到播放的整个过程

首先是视频采集，通常会使用摄像机、摄像头进行视频采集，如图 5-33 所示。采集了视频数据之后就要进行模数转换，将模拟信号转变成数字信号。当然现在很多摄像机或摄像头可以直接输出数字信号。信号输出之后，还要进行预处理，将 RGB 信号变成 YUV 信号，然后进行压缩编码，包括音频和视频，然后进行音视频封装，形成有利于存储或传输的格式，以便可以通过网络传输出去。

图 5-33　视频采集

4. YUV 成像原理

前面介绍过 RGB 信号和 YUV 信号，YUV 本质上是一种颜色数字化表示方式。视频通信系统之所

以要采用 YUV，而不是 RGB，主要是因为 RGB 信号不利于压缩。在 YUV 这种格式中，加入了亮度这一概念。视频工程师发现，眼睛对于亮和暗的分辨要比对颜色的分辨更精细一些，也就是说，人眼对色度的敏感程度要低于对亮度的敏感程度，所以在视频存储中，没有必要存储全部颜色信号，可以把更多带宽留给黑白信号（亮度），将稍少的带宽留给彩色信号（色度），这就是 YUV 的基本原理，*Y* 是亮度，*U* 和 *V* 则是色度。Y′CbCr 是 YUV 的压缩版本，不同之处在于 Y′CbCr 用于数字图像领域，而 YUV 用于模拟信号领域，MPEG、DVD、摄像机中常说的 YUV 其实就是 Y′CbCr。YUV 的成像过程如图 5-34 所示。

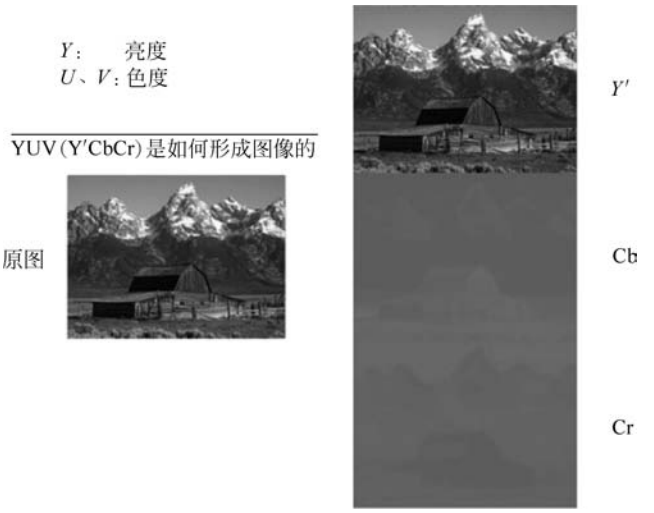


图 5-34　YUV 是如何形成图像的

另外 YUV 格式还可以很方便地实现视频压缩，YUV 的存储格式与其采样方式密切相关。采样的原理非常复杂，通常采用的是 YUV 4∶2∶0 采样方式，能获得 1/2 的压缩率。这些预处理做完后，就可以正式进行编码了。

5. IPB 帧

I 帧是指帧内编码帧（Intra Picture），采用帧内压缩去掉空间冗余信息。P 帧是指前向预测编码帧（Predictive-Frame），通过将图像序列中前面已经编码的帧的时间冗余信息来压缩传输数据量的编码图像。参考前面的 I 帧或者 P 帧。B 帧是指双向预测内插编码帧（Bi-Directional Interpolated Prediction Frame），既考虑源图像序列前面的已编码帧，又顾及源图像序列后面的已编码帧之间的冗余信息，以此来压缩传输数据量的编码图像，也称为双向编码帧。参考前面的一个 I 帧或者 P 帧及其后面的一个 P 帧。

注意：B 帧有可能参考它后面的 P 帧，解码器一般有视频帧缓存队列（以 GOP 为单位）。

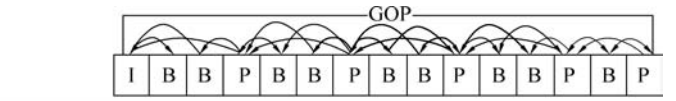
6. PTS 和 DTS

解码时间戳（Decoding Time Stamp，DTS）用于标识读入内存中比特流在什么时候开始送入解码器中进行解码，也就是解码顺序的时间戳。展示时间戳（Presentation Time Stamp，PTS）用于度量解码后的视频帧什么时候被显示出来。在没有 B 帧的情况下，DTS

和 PTS 的输出顺序是一样的，一旦存在 B 帧，PTS 和 DTS 则会不同，也就是显示顺序的时间戳。

7. GOP 简介

图像组(Group OF Picture，GOP)，指两个 I 帧之间的距离。Reference 即参考周期，指两个 P 帧之间的距离。一个 I 帧所占用的字节数大于一个 P 帧，一个 P 帧所占用的字节数大于一个 B 帧，所以在码率不变的前提下，GOP 值越大，P、B 帧的数量就会越多，平均每个 I、P、B 帧所占用的字节数就越多，也就更容易获取较好的图像质量。Reference 越大，B 帧的数量就越多，同理也更容易获得较好的图像质量。IPB 帧的字节大小为 I＞P＞B。GOP 解码顺序和显示顺序如图 5-35 所示。



解码顺序：	1	3	4	2	6	7	5	9	10	8	12	13	11	15	14
显示顺序：	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DTS:	1	3	4	2	6	7	5	9	10	8	12	13	11	15	14
PTS:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

图 5-35　GOP 解码顺序和显示顺序

5.3.2　视频压缩

对原始视频进行压缩的目的是为了去除冗余信息，这些冗余信息包括以下几个方面。

(1) 空间冗余，即图像相邻像素之间有较强的相关性。

(2) 时间冗余，即视频序列的相邻图像之间内容相似。

(3) 编码冗余，即不同像素值出现的概率不同。

(4) 视觉冗余，即人的视觉系统对某些细节不敏感。

(5) 知识冗余，即规律性的结构可由先验知识和背景知识得到。

数据压缩主要分为无损压缩和有损压缩。无损压缩(Lossless)即压缩前、解压缩后图像完全一致 X=X′，压缩比一般比较低(2∶1～3∶1)，典型的压缩格式有 Winzip、JPEG-LS 等。有损压缩(Lossy)即压缩前与解压缩后图像不一致 X≠X′，压缩比一般都比较高(10∶1～20∶1)，典型格式有 MPEG-2、H. 264/AVC、AVS 等。数据压缩与解压缩的概要流程如图 5-36 所示。

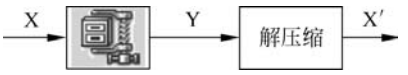


图 5-36　数据压缩与解压缩

无损压缩也称为可逆编码，重构后的数据与原数据完全相同，适用于磁盘文件的压缩等。主要采用熵编码方式，包括香农编码、哈夫曼编码和算术编码等。香农编码采用信源符号的累计概率分布函数来分配码字，效率不高，实用性不大，但对其他编码方法有很好的理论指导意义。霍夫曼(哈夫曼)

编码完全依据出现概率来构造异字头的平均长度最短的码字，先对图像数据扫描一遍，计算出各种像素出现的概率，按概率的大小指定不同长度的唯一码字，由此得到一张该图像的哈夫曼码表。编码后的图像数据记录的是每像素的码字，而码字与实际像素值的对应关系记录在码表中。算术编码是用符号的概率和编码间隔两个基本参数来描述的，在给定符号集和符号概率的情况下，算术编码可以给出接近最优的编码结果。使用算术编码的压缩算法通常先要对输入符号的概率进行估计，然后编码，估计越准，编码结果就越接近最优的结果。

有损压缩也称为不可逆编码，重构后的数据与原数据有差异，适用于任何允许有失真的场景，例如视频会议、可视电话、视频广播、视频监控等。常用的有损编码方式包括预测编码、变换编码、量化编码、混合编码等。

5.3.3 视频编码原理

视频编码是指通过特定的压缩技术，将某个视频格式的文件转换成另一种视频格式。视频数据在时域和空域层面都有极强的相关性，这也表示有大量的时域冗余信息和空域冗余信息，压缩编码技术就是去掉数据中的冗余信息。

去除时域冗余信息的主要方法包括运动补偿、运动表示、运动估计等。运动补偿是通过先前的局部图像来预测、补偿当前的局部图像，可有效减少帧序列冗余信息。运动表示是指不同区域的图像使用不同的运动矢量来描述运动信息，运动矢量通过熵编码进行压缩(熵编码在编码过程中不会丢失信息)。运动估计是指从视频序列中抽取运动信息，通用的压缩标准使用基于块的运动估计和运动补偿。

去除空域冗余信息的主要方法包括变换编码、量化编码和熵编码。变换编码是指将空域信号变换到另一正交矢量空间，使相关性下降，数据冗余度减小。量化编码是指对变换编码产生的变换系数进行量化，控制编码器的输出位率。熵编码是指对变换、量化后得到的系数和运动信息进行进一步的无损压缩。

原始未压缩的视频数据存储空间大，一个 1080P 的 1s 视频大约需要 160MB。原始未压缩的视频数据传输占用带宽大，20Mb/s 的带宽传输上述视频大约需要 64s，这样肯定无法满足实时视频的需求，而经过 H.264 编码压缩之后，视频只有大约 2MB，20Mb/s 的带宽仅仅约需要 1s，基本可以满足实时传输的需求。如果再加一些编码优化，或者降低分辨率，还可以进一步压缩视频，从而降低传输时间，所以从视频采集传感器采集来的原始视频必须经过视频编码。

注意：笔者抛砖引玉，以上参数不一定很精确，读者可以自行枚举更多的案例。

如上所述，原始未压缩视频所需的存储空间是非常大的。那为什么巨大的原始视频可以编码成很小的视频呢？这其中的技术又是什么呢？核心思想就是去除冗余信息。视频一般有 5 种冗余信息，包括空间冗余、时间冗余、编码冗余、视觉冗余和知识冗余。

视频本质上是一系列图片进行连续快速播放，最简单的压缩方式就是对每帧图片进行压缩，例如比较古老的 MJPEG 编码所采用的就是这种编码方式，这种编码方式只有帧内编码，利用空间上的取样预测来编码。形象的比喻就是把每帧都作为一张图片，采用 JPEG 的

编码格式对图片进行压缩，这种编码只考虑了一张图片内的冗余信息压缩，如图 5-37 所示，绿色的部分就是当前待编码的区域，灰色的部分就是尚未编码的区域，绿色区域可以根据已经编码的部分进行预测（绿色的左边、下边、左下）。

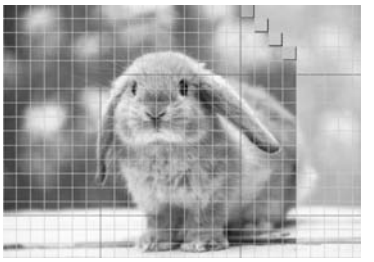


图 5-37　帧间预测编码

但是帧和帧之间存在时间上的相关性，后续逐步开发出了一些比较高级的编码器，可以采用帧间编码。简单点说就是通过搜索算法选定了帧上的某些区域，然后通过计算当前帧和前后参考帧的向量差进行编码的一种形式，如图 5-38 所示，滑雪的同学是向前位移的，但实际上是雪景在向后位移，P 帧通过参考帧（I 或其他 P 帧）就可以进行编码了，编码之后的大小会变得非常小，压缩比非常高。

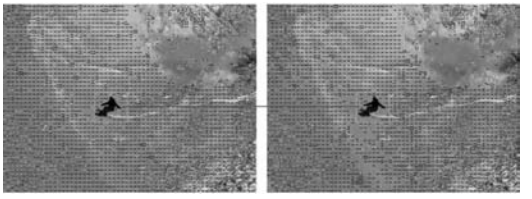


图 5-38　帧间预测编码

I 帧采取帧间预测编码，主要流程包括帧间预测、变换、量化、熵编码等，如图 5-39 所示。

P 帧采取帧间预测编码，主要流程包括帧间预测、变换、量化、熵编码等，如图 5-40 所示。

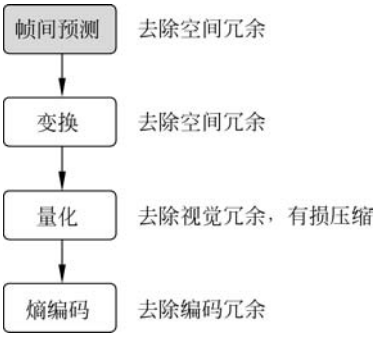


图 5-39　I 帧的帧间预测流程图

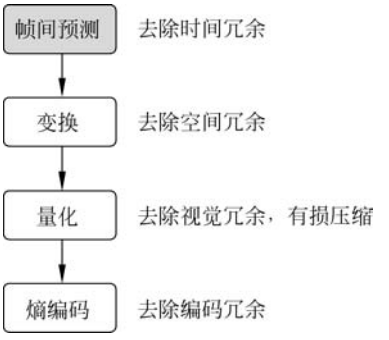


图 5-40　P 帧的帧间预测流程图

5.3.4　视频编码的关键技术

编解码器包括编码器和解码器。编码器（Encoder）是指压缩信号的设备或程序；解码器（Decoder）是指解压缩信号的设备或程序；编解码器（Codec）是指编解码器对。编码的主要流程包括预测、变换、量化、熵编码，解码的流程与之互逆，如图 5-41 所示。编解码器中涉及的关键技术主要包括预测、变换、量化、熵编码，下面详细讲述这几方面的知识。

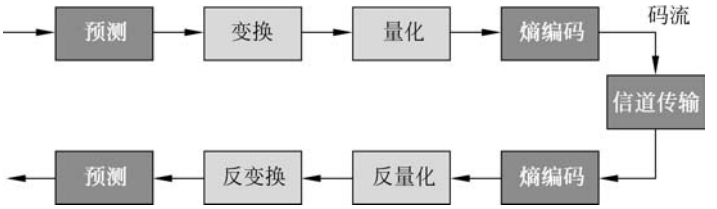


图 5-41　编解码的关键流程

第一，为什么要变换？变换可以去除图像像素之间的空间相关性。变换是一种线性运算，可以将图像从空间域转换到变换域或者频率域。空间域图像的能量往往分布相对比较均匀，经过变换后，变换域中图像的变换系数间近似是统计独立的，基本去除了相关性，并且能量集中在直流和低频率的变换系数上，高频率变换系数的能量很小，甚至大部分高拟系数能量接近于零，所以在变换域进行滤波、进行与视觉特性相匹配的量化及熵编码，可以实现图像数据的有效压缩。变换的核心点是找到一个完美的正交矩阵，那什么样的正交变换矩阵才算是完美的呢？最佳的变换矩阵应该是变换矩阵中每行或每列的基矢量和图像的统计特性相匹配，即应和图像本身的特征矢量相匹配。K-L 变换是在均方误差最小准则下，失真最小的一种变换，但 K-L 变换的变换矩阵是由图像协方差矩阵的特征矢量组成的，不同图像的变换矩阵是不同的，每次都要进行计算，并且计算烦琐，很难满足实时处理要求，而在各种正交变换中，以自然图像为编码对象时与 K-L 变换性能最接近的是离散余弦变换 DCT。理论与实验表明：对于标准清晰度的自然图像，像素块尺寸选 8×8 或 16×16 是最适宜的。

第二，为什么要量化？通常变换离不开量化，因为图像从空间域矩阵变换到变换域的变换系数矩阵，其系数个数并未减少，数据量也不会减少，因此并不能直接压缩数据。为了压缩码率，还应当根据图像信号在变换域中的统计特性进行量化和熵编码。按照量化的维数，量化可分为标量量化和矢量量化。标量量化是一维量化，是一个幅度对应一个量化结果，而矢量量化是二维甚至多维量化，两个或两个以上的幅度决定一个量化结果。在图像和视频编码中，普遍采用标量量化。量化过程存在量化误差，这种误差称为量化噪声。量化器设计时将信号幅度划分为若干个量化级，一般为 2 的整数次幂。把落入同一级的样本值归为一类，并给定一个量化值。量化级数越多，量化误差就越小，质量就越好。量化级数取决于量化步长和信号强度。理想的标量量化器应该采用非线性量化，也就是每个量化区间间距应该不相等，每个量化区间的划分取决于信号的概率密度函数分布。在视频编码中，残差图像块的变换系数近似认为服从广义高斯分布或拉普拉斯分布，可以根据这个概率分布函数，确定非线性标量量化。残差图像的变换系数通常主要分布在零频率分量附近，而偏离零频率的分量，出现概率呈指数下降。在不影响人眼观看的条件下，量化器的作用是降低系数的精度来消除可以忽略的系数。人眼视觉系统对不同频率分量信号的失真感知能力是不一样的，人眼对高频系数相对不敏感，对色度相对也不敏感。为了得到好的编码效果，应该根据系数块中的不同位置设计量化器，对图像进行 DCT 变换后直接对 DCT 系数进行符合人眼视觉特性的非均匀量化。通常通过一个归一化加权量化矩阵实现，而目前 JPEG 和 MPEG

标准中均采用了加权量化矩阵实现基于人眼视觉特性的自适应量化控制。

第三，为什么要采用熵编码？经 DCT 变换后，能量主要集中在直流和较低频率系数上，而大部分变换系数为 0 或接近 0。再加上视觉加权处理和量化，有更多的 0 产生，这些 0 往往连在一起而成串出现。连续 0 的个数叫作零游程，于是在编码时，不对单个的 0 编码，而对零游程编码，这样就会提高编码效率。为了制造更长的零游程，在编码之前，对变换系数矩阵采用 Z 字形扫描读取数时进行重新排列，很多像块经变换后，变换系数经过 ZigZag 排列，排在队尾的很长一串系数全是 0。游程编码通常采用一种称为 Run-Level 的编码技术，这里 Run 是两个非零系数之间连续 0 的个数，而 Level 是非 0 数的绝对值。经过 ZigZag 扫描后的系数序列可以转化为若干个 Run-Level 数据对。由于统计编码是信息保持编码，所以信息量是怎样度量的呢？在信息论中，信息量是用不确定性的量度来定义的。一条消息出现的概率越小，则信息量越大；若出现的概率越大，则信息量越小。从信息论看，熵是对一条消息进行编码时最小平均比特率的理论值。这个结果虽然没有说明如何设计码字的方法，但却很有用。假如设计的码字平均比特率和熵一样，那么这个码字就是最佳的。由此得出结论，熵提供了一种可以测试编码性能的参考基准。利用信息熵原理的编码方法有若干种，如哈夫曼编码方法利用概率分布特性、算术编码利用概率分布的编码方法和游程编码利用相关性等。

5.3.5 视频编解码流程

视频的编解码流程分为编码和解码，是互逆的过程。编码过程包括运动估计、运动补偿、DCT、量化与熵编码等，如图 5-42 所示。

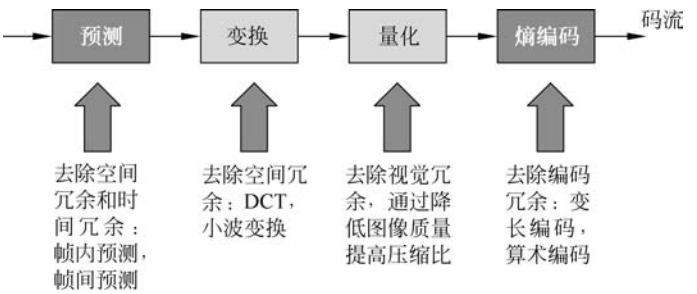


图 5-42 编码流程

(1) 运动估计(Motion Estimate)是指从前几帧中寻找匹配的宏块，有多种不同的搜索算法，编码获得的质量和速度也不相同。其中快速算法的编码质量比全搜索算法低不了太多，但是速度却高了很多倍。得到运动矢量的过程被称为运动估计。

(2) 运动补偿(Motion Compensate)是通过先前的局部图像来预测、补偿当前的局部图像，它是减少帧序列冗余信息的有效方法，包括全局运动补偿和分块运动补偿两类。运动补偿的结果分为两份，一份被当前帧做参考求出差值 Dn，另一份用于生成 Fn 的对应参考帧。

(3) 离散余弦变换(Discrete Cosine Transform,DCT),主要用于对数据或图像进行压缩,能够将空域的信号转换到频域上,具有良好的去相关性的性能。图像经过 DCT 变换后,其频率系数的主要成分集中于比较小的范围,且主要位于低频部分。DCT 变换本身是无损的,但是在图像编码等领域给接下来的量化、哈弗曼编码等创造了很好的条件,同时,由于 DCT 变换是对称的,所以,可以在量化编码后利用 DCT 反变换,在接收端恢复原始的图像信息。DCT 是从 DFT(离散傅里叶变换)推导出来的另一种变换,因此许多 DFT 的属性在 DCT 中仍然被保留下来。

(4) 量化(Quant)是指除指定的数值(有损压缩)。量化的结果分为两份,一份做进一步处理,另一份经过反量化(Rescale)/反 DCT(IDCT)变化,结合第 2 步的运动补偿生成 Fn 对应的参考帧,供后续参考。

(5) 重排(Reorder)是指对量化后的数据进行重排序,常用算法为游程编码等。

(6) 熵编码(Entropy Encode)进行最后编码,使用上一步结果及 Vectors 和 Headers 的内容。

解码是编码的逆操作,过程如图 5-43 所示。

(1) 熵解码(Entropy Decode)是指对数据进行熵解码,填充 Vectors 和 headers。

(2) 重排(Reorder)是指对熵编码后的数据记性重排序,常用游程解码等。

(3) 反量化(Rescale)是指乘指定的比例。

(4) 反离散余弦变换(IDCT)即反 DCT 变换,得到 D'_n。

(5) 运动补偿对上一帧的参考帧做运动补偿处理。

(6) 重构(Reconstructed)是指结合第 5 步的运动补偿结果和第 3 步的反量化结果,重构 F'_n参考帧。

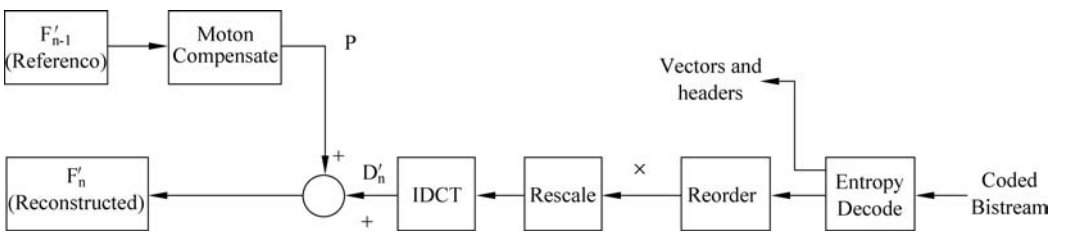


图 5-43 解码流程

5.4 视频播放原理

绝大多数的视频播放器,如 VLC、MPlayer、Xine,包括 DirectShow,在播放视频的原理和架构上是非常相似的。视频播放器播放一个互联网上的视频文件需要经过几个步骤,包括解协议、解封装、音视频解码、音视频同步、音视频输出。

5.4.1 视频播放器简介

视频播放器播放本地视频文件或互联网上的流媒体文件大概需要解协议、解封装、解码、同步、渲染等几个步骤，如图 5-44 所示。

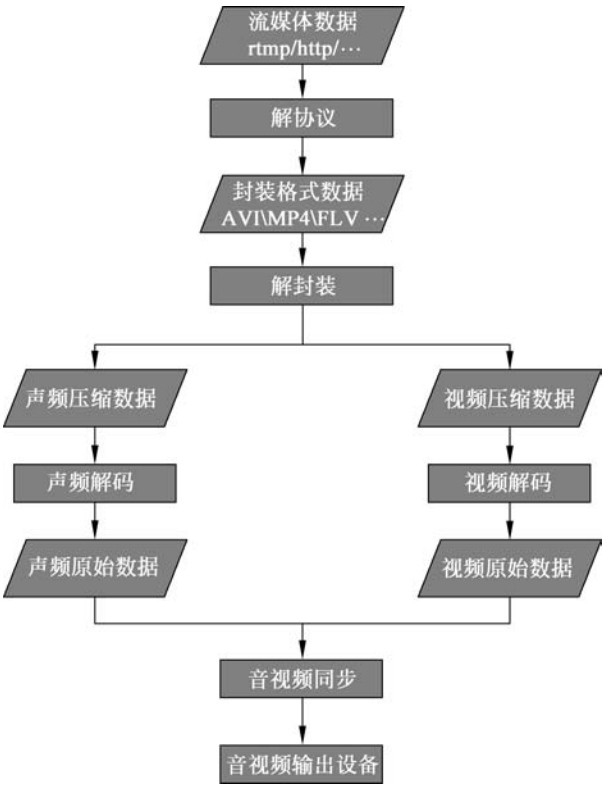


图 5-44 视频播放流程图

1. 解协议

解协议是指将流媒体协议的数据，解析为标准的封装格式数据。音视频在网络上传播的时候，常采用各种流媒体协议，例如 HTTP、RTMP、RTMP、MMS 等。这些协议在传输音视频数据的同时，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。在解协议的过程中会去除信令数据而只保留音视频数据。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。

注意：“文件”本身也是一种“协议”，常见的流媒体协议有 HTTP、RTSP、RTMP 等。

2. 解封装

解封装是指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。封装格式种类有很多，例如 MP4、MKV、RMVB、TS、FLV、AVI 等，其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。

3. 解码

解码是指将视频/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，视频的压缩编码标准则包含 H.264、MPEG-2、VC-1 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。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。

4. 音视频同步

根据解封装模块在处理过程中获取的参数信息，同步解码出来的视频和音频数据，被送至系统的显卡和声卡播放出来。为什么需要音视频同步呢？媒体数据经过解复用流程后，音频/视频解码便是独立的，也是独立播放的，而在音频流和视频流中，其播放速度是由相关信息指定的，例如视频是根据帧率，音频是根据采样率。从帧率及采样率即可知道视频/音频播放速度。声卡和显卡均是以一帧数据来作为播放单位，如果单纯依赖帧率及采样率进行播放，在理想条件下，应该是同步的，不会出现偏差。

下面以一个 44.1kHz 的 AAC 音频流和 24f/s 的视频流为例来说明。如果一个 AAC 音频 frame 每个声道包含 1024 个采样点，则一个 frame 的播放时长为(1024/44 100)×1000ms≈23.27ms，而一个视频 frame 播放时长为 1000ms/24≈41.67ms。理想情况下，音视频完全同步，但实际情况下，如果用上面那种简单的方式，慢慢地就会出现音视频不同步的情况，要么是视频播放快了，要么是音频播放快了。可能的原因包括：一帧的播放时间难以精准控制；音视频解码及渲染的耗时不同，可能造成每一帧输出有一点细微差距，长久累计，不同步便越来越明显；音频输出是线性的，而视频输出可能是非线性的，从而导致有偏差；媒体流本身音视频有差距(特别是 TS 实时流，音视频能播放的第 1 个帧起点不同)，所以解决音视频同步问题引入了时间戳，它包括几个特点：首先选择一个参考时钟(要求参考时钟上的时间是线性递增的)，编码时依据参考时钟给每个音视频数据块都打上时间戳。播放时，根据音视频时间戳及参考时钟来调整播放，所以视频和音频的同步实际上是一个动态的过程，同步是暂时的，不同步则是常态。

5.4.2 FFmpeg 播放架构与原理

ffplay 是使用 FFmpeg API 开发的功能完善的开源播放器。在 ffplay 中各个线程如图 5-45 所示，扮演角色如下：read_thread 线程扮演着图中 Demuxer 的角色；video_thread 线程扮演着图中 Video Decoder 的角色；audio_thread 线程扮演着图中 Audio Decoder 的角色。主线程中的 event_loop 函数循环调用 refresh_loop_wait_event 则扮演着视频渲染的角色。回调函数 sdl_audio_callback 扮演着图中音频播放的角色。VideoState 结构体变量则扮演着各个线程之间的信使。

(1) read_thread 线程负责读取文件内容，将 video 和 audio 内容分离出来后生成 packet，将 packet 输出到 packet 队列中，包括 Video Packet Queue 和 Audio Packet Queue，不考虑 subtitle。

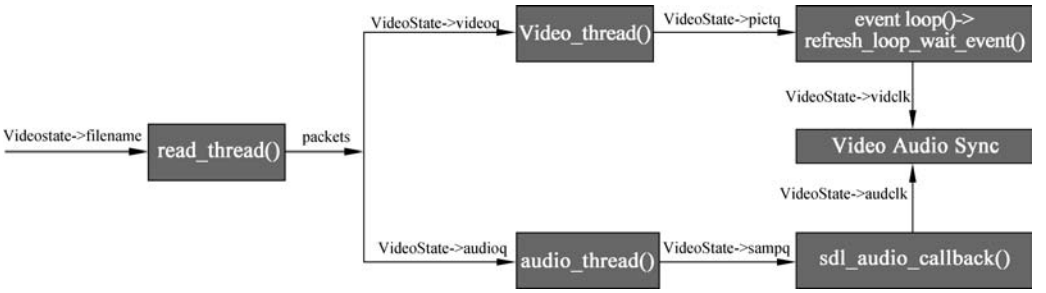


图 5-45 ffplay 基本架构图

(2) video_thread 线程负责读取 Video Packets Queue 队列,将 video packet 解码得到 Video Frame,将 Video Frame 输出到 Video Frame Queue 队列中。

(3) audio_thread 线程负责读取 Audio Packets Queue 队列,将 audio packet 解码得到 Audio Frame,将 Audio Frame 输出到 Audio Frame Queue 队列中。

(4) 主线程→event_loop→refresh_loop_wait_event 负责读取 Video Frame Queue 中的 video frame,调用 SDL 进行显示,其中包括了音视频同步控制的相关操作。

(5) SDL 的回调函数 sdl_audio_callback 负责读取 Audio Frame Queue 中的 audio frame,对其进行处理后,将数据返回给 SDL,然后 SDL 进行音频播放。

FFmpeg 解码流程涉及几个重要的数据结构和 API,如图 5-46 所示。

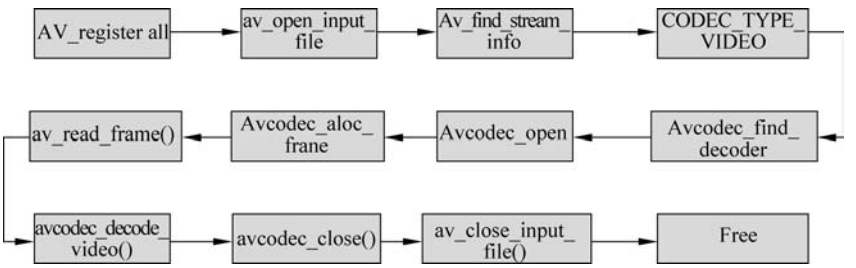


图 5-46 FFmpeg 解码流程图

(1) 注册所有容器格式和 CODEC,使用 av_register_all,最新版本中无须调用该函数。

(2) 打开文件 av_open_input_file,最新版本为 avformat_open_input。

(3) 从文件中提取流信息 av_find_stream_info。

(4) 枚举所有流,查找的种类为 CODEC_TYPE_VIDEO。

(5) 查找对应的解码器 avcodec_find_decoder。

(6) 打开编解码器 avcodec_open。

(7) 为解码帧分配内存 avcodec_alloc_frame。

(8) 不停地从码流中提取帧数据 av_read_frame。

(9) 判断帧的类型,对于视频帧则调用 avcodec_decode_video。

(10) 解码完后,释放解码器 avcodec_close。

(11) 关闭输入文件 av_close_input_file。

注意:该流程图为 FFmpeg 2.x 的版本,最新的 FFmpeg 4.x 系列的流程图略有改动。

5.4.3 VLC 播放原理

VLC 播放一个视频分为 4 个步骤,第一步是 access 访问,或者理解为接收、获取、得到;第二步是 demux 解复用,就是把通常合在一起的音频和视频分离(可能还有字幕);第三步是 decode 解码,包括音频和视频的解码;第四步是 output 输出,也分为音频和视频的输出(aout 和 vout)。

例如播放一个 UDP 组播的 MPEG TS 流,access 部分负责从网络接收组播流,放到 VLC 的内存缓冲区中,access 模块关注 IP 协议,如是否为 IPv6、组播地址、组播协议、端口等信息;如果检测出来的是 RTP 协议(RTP 协议在 UDP 头部简单地加上了固定 12B 的信息),还要分析 RTP 头部信息。access 模块包括很多具体的协议,如 file、http、dvd、ftp、smb、tcp、dshow、mms、v4l 等。

demux 部分首先要解析 TS 流的信息。TS 格式是 MPEG-2 协议的一部分,概括地说,TS 通常是固定 188B 的一个 packet,一个 TS 流可以包含多个 program,一个 program 又可以包含多个视频、音频和文字信息的 ES 流;每个 ES 流会由不同的 PID 标示,而又为了可以分析这些 ES 流,TS 有一些固定的 PID,用来间隔发送 program 和 es 流信息的表格:PAT 和 PMT 表。之所以需要 demux,是因为音视频在制作的时候实际上都是独立编码的,得到的是分开的数据,为了传输方便必须用某种方式合并起来,这就有了各种封装格式,也就有了 demux。

demux 分解出来的音频流和视频流分别送往音频解码器和视频解码器。因为原始的音视频都占用大量空间,而且是冗余度较高的数据,通常在制作的时候会进行某种压缩。这就是常见的音视频编码格式,包括 MPEG-1、MPEG-2、MPEG-4、H.264、rmvb 等。音视频解码器的作用就是把这些压缩了的数据还原成原始的音视频数据。VLC 解码 MPEG-2 使用了一个独立的库 libmpeg2。VLC 关于编解码的模块都放在/modules/codec 目录下,其中包括著名的庞大的 FFmpeg。解码器,例如视频解码器输出的是一张一张的类似位图格式的图像,但是要让人从屏幕上看得到,还需要一个视频输出的模块。可以像一个 Win32 窗口程序那样直接把图像画到窗口 DC 上,其实 VLC 的一个输出模块 WinGDI 就是这么实现的,但是通常这种实现方式太慢了,而且会消耗大量的 CPU 资源。在 Windows 下比较好的办法是用 DirectX 的接口,这样会自动调用显卡的加速功能。

这样的功能分解使模块化更容易一点,每个模块只需要专注于自己的事情,从整体来讲功能强大而且灵活。

5.4.4 现代播放器架构

通常来讲,一个典型的播放器可以分解成几个部分,包括应用层、视图层和内核层。现

代播放器的通用架构如图 5-47 所示。

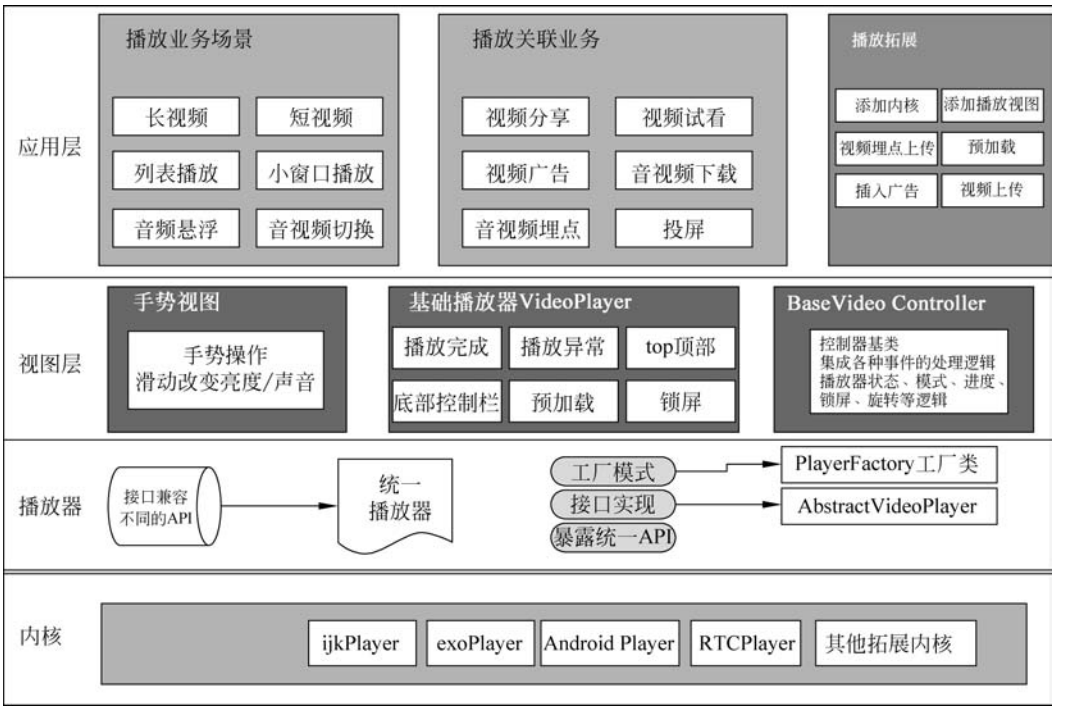


图 5-47　现代播放器通用架构图

1. 架构简介

用户界面(UI)定义了终用户端的观看体验,包括“皮肤”(播放器的外观设计)、所有可自定义的特性(如播放列表和社交分享等)及业务逻辑部分(如广告、设备兼容性逻辑及认证管理等)。

播放器内核是最核心的部件,播放器最底层的部分是内核,如解码器等,这层的功能直接调用操作系统暴露出来的 API。解码器的主要功能在于解码并渲染视频内容,DRM 管理器则通过解密过程来控制是否有权播放。DRM 即数字版权管理,是指数字内容,如音视频节目内容、文档、电子书籍等在生产、传播、销售、使用过程中进行的权利保护、使用控制与管理的技术。

2. 用户界面

UI 层处于播放器架构的上层,它控制了用户所能看到和交互的东西,同时也可以使用自己的品牌来将其定制,为用户提供独特的用户体验。这一层最接近于前端开发部分。在 UI 内部,也包含了业务逻辑组件,这些组件构成了播放体验的独特性,虽然终用户端无法直接和这部分功能进行交互。UI 部分主要包含 3 大组件。

(1)“皮肤”是对与播放器视觉相关部分的统称,包括进度控制条、按钮和动画图标等。和大部分设计类的组件一样,这部分组件也可以使用 CSS 实现,设计师或者开发者可以很

方便地用来集成。

(2) UI 逻辑部分,此部分定义了播放过程中和用户交互方面所有可见的交互,包括播放列表、缩略图、播放频道的选择及社交媒体分享等。基于预期达到的播放体验,还可以往这部分加入很多其他的功能特性,其中有很多以插件的形式存在。除了传统的 UI 元素之外,还有一个非常有趣的特性,在用户观看流媒体的时候,直播以小视窗的形式展示,观众可以通过这个小窗口随时回到直播中。由于布局或者 UI 和多媒体引擎完全独立,这些特性在 HTML5 中使用 dash.js 只需几行代码就能实现。对于 UI 部分来讲,最好的实现方式是让各种特性都以插件/模块的形式添加到 UI 核心模块中。

(3) 业务逻辑部分,除了上面所介绍的两部分功能特性之外,还有一个不可见的部分,这部分构成了业务的独特性,包括认证和支付、频道和播放列表的获取,以及广告等。这里也包含了一些与技术相关的东西,例如用于 A/B 测试模块,以及和设备相关的配置,这些配置用于在多种不同类型的设备之间选择多个不同的媒体引擎。这部分的最大特点在于,无论使用什么样的底层引擎,在上层都可以使用相同的 JavaScript 或者 CSS 来定制 UI 或者业务逻辑。

3. 多媒体引擎

近年来,多媒体引擎更是以一种全新独立的组件的形式出现在播放器架构中。在 MP4 时代,平台处理了所有与播放相关的逻辑,而只将一部分与多媒体处理相关的特性(仅仅是播放、暂停、拖曳和全屏模式等功能)开放给开发者,然而,新的基于 HTTP 的流媒体格式需要一种全新的组件来处理和控制新的复杂性,包括解析声明文件、下载视频片段、自适应码率监控及决策指定等甚至更多。起初,ABR 的复杂性被平台或者设备提供商处理了,然而,随着主播控制和定制播放器需求的递增,在一些新的播放器中慢慢也开放了一些更为底层的 API,如 Web 上的 Media Source Extensions、Flash 上的 Netstream 及 Android 平台的 Media Codec,并迅速吸引来了很多基于这些底层 API 的强大而健壮的多媒体引擎。

4. 解码器和 DRM 管理器

出于解码性能和安全考虑,解码器和 DRM 管理器与操作系统平台密切绑定,其工作流程如图 5-48 所示。

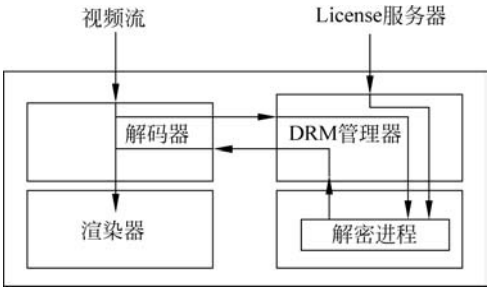


图 5-48　解码器、渲染器和 DRM 工作流程图

解码器用于处理与最底层播放相关的逻辑，它将不同封装格式的视频进行解包，并将其内容解码，然后将解码后的视频帧交给操作系统进行渲染，最终让终用户端看到。由于视频压缩算法变得越来越复杂，解码过程是一个需要密集计算的过程，并且为了保证解码性能和流畅的播放体验，解码过程需要强依赖于操作系统和硬件。现在的大部分解码依赖于 GPU 加速解码的帮助，这也是为什么免费而更强大的 VP9 解码器没有赢得 H.264 市场地位的原因之一。如果没有 GPU 的加速，解码一个 1080P 的视频就会占去 70%左右的 CPU 计算量，并且丢帧率还可能很严重。在解码和渲染视频帧的基础之上，管理器也提供了一个原生的 buffer，多媒体引擎可以直接与该 buffer 进行交互，实时了解它的大小并在必要的时候刷新它。前面提到，每个平台都有它自己的渲染引擎和相应的 API，例如 Flash 平台有 Netstream，Android 平台有 Media Codec API，而 Web 上则有标准的 Media Sources Extensions。MSE 越来越吸引开发者的眼球，将来可能会成为继浏览器之后其他平台上的事实标准。

DRM 管理器在传输工作室生产的付费内容的时候是必要的。这些内容必须防止被盗，因此 DRM 的代码和工作过程都向终端用户和开发者屏蔽了。解密过的内容不会离开解码层，因此也不会被拦截。

5.5 视频转码原理

视频转码(Video Transcoding)是指将已经压缩编码的视频码流转换成另一种视频码流，以适应不同的网络带宽、不同的终端处理能力和不同的用户需求。视频转码本质上是一个先解码，再编码的过程，因此转换前后的码流可能遵循相同的视频编码标准，也可能不遵循相同的视频编码标准。视频转码技术就是通过某种手段改变现有视频数据的编码方式。视频转码技术使用的目的不同，其实现的手段也各不相同。

5.5.1 视频转码

不同编码格式之间的数据转码可通过转码方法改变视频数据的编码格式。通常这种数据转码会改变视频数据的现有码流和分辨率。例如可以将基于 MPEG-2 格式的视频数据转换为 DV、MPEG-4 或其他编码格式，同时根据其转码目的，指定转码产生视频数据的码流和分辨率。可以将 MPEG-2 全 I 帧 50Mb/s 的视频源数据转换为 25Mb/s 码流的 DV 格式数据，用于笔记本移动编辑系统，同时产生一个 300×200 低分辨率的 MPEG-4 文件，使用 REAL 或者微软的 WMV 格式进行封装，通过互联网络传输至主管领导处用于审看。这种转码方式设计的算法较为复杂，其实质上是一个重新编码的过程，涉及的算法复杂度和系统开销是由转码所需图像质量要求及转码前后两种编码方式的相关度所决定的。

相同编码格式的数据转码指不改变压缩格式，只通过转码手段改变其码流或头文件信息。根据其使用目的，可分为改变码流和不改变码流两种。如可以将 MPEG-2 全 I 帧 50Mb/s 码流的视频数据转码为 MPEG-2 IBBP 帧 8Mb/s 码流的视频数据，直接用于播出服务器以便播出。或者将基于 SONY 视频服务器头文件封装的 MPEG-2 全 I 帧 50Mb/s

码流的视频文件，改变其头文件和封装形式，使之可以在给予 MATROX 板卡的编辑系统上直接编辑使用。这种转码方式的复杂度要小于不同编码格式转码的复杂度，而且对视频工程而言，更加具有可操作性。

视频数据不同编码之间的相互转化有很多算法可以实现，许多运动图像专家对此也进行了深入研究，针对不同的编码方式提出了相当多可行的方案。这些方案的共同特点就是充分利用所需相互转换编码之间的共同特征，尽量减少编解码所带来的图像质量损失，同时达到时间和资源消耗的平衡。如将一个 MPEG-2 的视频数据转换成 MPEG-4 的视频数据，当然可以采用的方法是先将 MPEG-2 的视频解压缩成单帧的图像序列，再将其重新压缩编码使其成为 MPEG-4 的视频数据。

5.5.2 非线性编辑

非线性编辑是借助计算机进行数字化制作，绝大多数的工作在计算机里完成的，不再需要那么多的外部设备，对素材的调用也是瞬间实现的，不用反反复复地在磁带上寻找，突破单一的时间顺序编辑限制，可以按各种顺序排列，具有快捷简便、随机的特性。非线性编辑只要上传一次就可以多次编辑，信号质量始终不会变低，所以节省了设备、人力，从而提高了效率。非线性编辑需要专用的编辑软件、硬件，现在绝大多数的电视/电影制作机构采用了非线性编辑系统。

传统线性视频编辑是按照信息记录顺序，从磁带中重放视频数据并进行编辑，需要较多的外部设备，如放像机、录像机、特技发生器、字幕机，工作流程十分复杂。非线性编辑系统是指把输入的各种音视频信号进行 A/D 转换，采用数字压缩技术将其存入计算机硬盘中。非线性编辑没有采用磁带，而是使用硬盘作为存储介质，记录数字化的音视频信号，由于硬盘可以满足在 1/25s 内完成任意一副画面的随机读取和存储，因此可以实现音视频编辑的非线性。

任何非线性编辑的工作流程都可以简单地看成输入、编辑、输出这样 3 个步骤。当然由于不同软件功能的差异，其使用流程还可以进一步细化。以 Premiere Pro 为例，其使用流程主要分成如下 5 个步骤，Premiere Pro 的主编辑界面如图 5-49 所示。

(1) 素材采集与输入就是利用 Premiere Pro 将模拟视频、音频信号转换成数字信号并存储到计算机中，或者将外部的数字视频存储到计算机中，成为可以处理的素材。输入主要是把其他软件处理过的图像、声音等，导入 Premiere Pro 中。

(2) 素材编辑就是设置素材的入点与出点，以选择最合适的部分，然后按时间顺序组接不同素材的过程。

(3) 特技处理特别重要，对于视频素材，特技处理包括转场、特效、合成叠加等；对于音频素材，特技处理包括转场、特效等。令人震撼的画面效果就是在这一过程中产生的，而非线性编辑软件功能的强弱，往往也体现在这方面。配合某些硬件，Premiere Pro 还能够实现特技播放功能。

(4) 字幕是节目中非常重要的部分，包括文字和图形两个方面。在 Premiere Pro 中制作字幕很方便，几乎没有无法实现的效果，并且还有大量的模板可供选择。

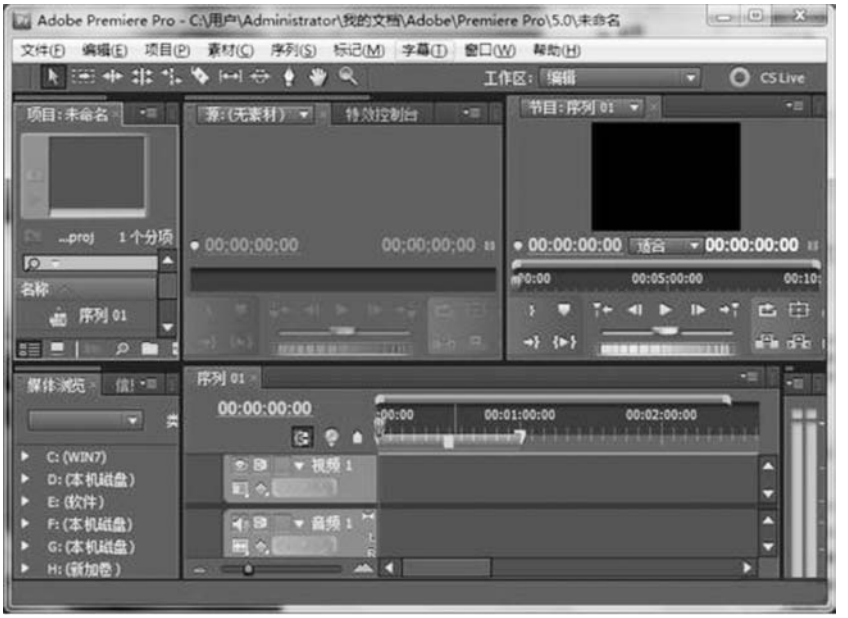


图 5-49　Premiere Pro 主编辑界面

(5) 节目编辑完成后就可以输出并回录到录像带上,也可以生成视频文件,发布到网上,或者刻录 VCD 和 DVD 等。

5.6　短视频技术

短视频技术主要涉及短视频拍摄端、播放端及合成、上传、转码、分发、加速、播放等操作,短视频的架构流程如图 5-50 所示。

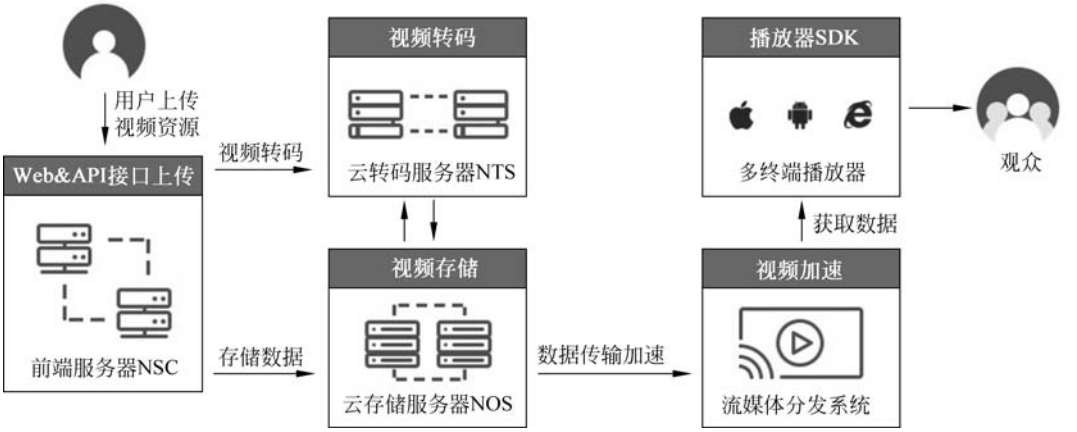


图 5-50　短视频架构流程

断点续拍指在拍摄过程中可以进行分段式拍摄，并将最终拍摄的所有内容合成一个视频的功能。通过断点续拍功能可以轻松实现不同的视频内容拼接，快速打造出视频拼接功能，进一步强化平台高质量的内容输出。在技术上为了实现断点续拍功能，需要在进行断点拍摄时直接调用系统 API 或第三方的相机库，将每段 mov 等格式的视频片段作为临时文件保存，存放到沙盒目录中，然后维护视频片段地址列表，等到拍摄结束后，将地址列表中指向的每个视频片段进行合成处理。如果不需实现回删功能，则可不必维护视频地址列表，读取所有单次录制时存储的临时视频片段，进行合成即可。最后生成一部完整的短视频。

重拍指当对拍摄的短视频部分内容不满意时可以直接剪切掉这部分内容，进行重拍。通过重拍功能配合断点续拍，可以对短视频进行多次剪辑、拍摄，以便增强视频制作时的灵活性，方便性。重拍功能需要维护一个视频片段地址列表，进行回删操作时只需删除视频列表中相对应的视频片段地址。最终只合成地址列表中指向的视频片段，合成完成后，删除该次录制的所有临时视频片段即可。

滤镜和水印技术，这两项功能作为短视频美化最核心的功能，由于它们的实现方式大同小异，所以将它们整合到了一起。滤镜是指拍摄的短视频可以选择不同的场景滤镜，并且进行美化程度调整。水印是指短视频拍摄完成后，可以在短视频上添加不同的水印，保护短视频的版权。

短视频使用到的底层技术主要可以分为两大类，一类来自于视频处理的性能需求，另一类来自于网络传输视频的带宽质量需求。视频的相关处理是非常耗性能的，例如 720p@30f/s 的原始视频对应了 82 944 000B/s 的数据量，或者说 663 552 000b/s。先不谈压缩传输，单单这些数据量在内存与 CPU 之间传输也不是一个小数目。除了可以服务器内对短视频进行高性能处理，CDN 对用户体验也至关重要。CDN 是一种将服务器放在全国各地的服务器集群，使用户在看视频时可以自动选取最近的服务器，实现最好的速度和用户体验。对广义的视频网站，从视频的上传到观看并不是实时的，中间存在视频转码、人工审核、CDN 同步 3 个步骤。视频直播则是另一种思路，讲求的是“秒开”，即用户从单击直播画面到看到直播视频要尽量快。通过在 CDN 服务器上使用缓存首段视频等方式，使“秒开”已经达到了几百毫秒的水平，极大地提升了用户体验。

虽然以技术积累为底气，但随着云服务技术的发展让技术易于获得，技术的商业价值如何实现，对于企业的发展更加至关重要，短视频也是如此。当前火热的短视频背后也隐藏着一些所谓的“黑科技”技术。“黑科技”通常是指人工智能之类的高新技术，但对于抖音、快手、秒拍这样的业务来讲，例如美颜是锦上添花、雪中送炭的技术，还有就是视频云体系和内容审核体系。前者决定能否上车，后者决定车能开多远。这里说的视频云体系，包括核心的视频采集、编解码、传输优化，包括 CDN、分布式存储等，不仅要让业务能快速上线，还要保证用户体验，保证不同终端、不同码率、不同格式、不同网络的自适应，以及对高并发的支持都是必需的。