清华大学

综 合 论 文 训 练

题目：可伸缩视频编码在流媒体系统中的应用

系　　别：　自动化系

专　　业：　自动化

姓　　名：　孟胜彬

指导教师：　陆文凯　研究员

2011 年 6 月 6 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____ 导师签名：_____ 日　期：_____

中文摘要

异构网络信道质量的波动和播放终端运算及显示能力的差异使得传统的视频编码难以满足不同场景和用户的需要。由最新的视频编码标准 H.264/AVC 扩展而来的可伸缩视频编码（Scalable Video Coding，SVC）能够通过丢弃一些数据来提供空间、时间和图像质量等方面的伸缩性，从而应对了网络化的视频应用所面临的挑战。本文研究了可伸缩视频编码在流媒体系统中的应用。在对 SVC 关键特性和现有的流媒体技术做介绍之后，我们设计并实现了支持 SVC 的流媒体服务器和播放客户端，组成了一个 SVC 视频点播系统。此外，为保证在嵌入式设备上的解码速度，还对 SVC 解码器的实现进行了针对 ARM 平台的优化。最后，对整个系统的测试表明我们成功地在流媒体视频播放中加入了伸缩特性，完成了一个支持 SVC 的流媒体系统；而且通过优化加快了解码速度。对 SVC 应用的研究将继续进行并逐渐改变人们目前的视频体验。

关键词：可伸缩视频编码；流媒体；应用；ARM 优化

ABSTRACT

Traditional coding technology can no longer fulfill the requirement of today's video application, considering the variety in 1) channel quality of heterogeneous network and 2) computing or displaying capacity of devices. Based on the newest video coding standard H.264/AVC, a technology called Scalable Video Coding (SVC) can provide temporal, spatial and quality scalability by discarding data in a coded video bit-stream, and will meet the challenge in network-oriented video applications. This paper is a research on applying SVC in the streaming media system. After introducing the key features of SVC and the current streaming media technology, we design and implement a streaming server and a player that both support SVC videos, so accomplish an SVC-supported VOD system. We still optimize the SVC decoder to speed it up in ARM platform. Test for the system indicates that we have successfully added scalability into the stream media playing and archived faster decoding after optimization .The research on SVC will continue and finally change people's video experience.

Keywords： Scalable Video Coding (SVC); streaming media; application; ARM optimization

目录

主要符号表

AU：Access Unit，访问单元

AVC：Advanced Video Coding，高级视频编码

DID：Dependency Identifier，依赖层（空间层）标识

DSS：Darwin Streaming Server，达尔文流媒体服务器

JVT：Joint Video Team，联合视频小组（由 ISO 和 ITU 的相关专家组成）

NAL：Network Abstract Layer，网络抽象层

NALU：NAL Unit，NAL 单元

QID：Quality Identifier，质量层标识

RTCP：Real-time Transport Control Protocol，实时传输控制协议

RTP：Real-time Transport Protocol，实时传输协议

RTSP：Real-Time Streaming Protocol，实时流协议

SDP：Session Describe Protocol，会话描述协议

SNR：Signal Noise Ratio，信噪比

SVC：Scalable Video Coding，可伸缩视频编码

TID：Temporal Identifier，时间层标识

VCL：Video Coding Layer，视频编码层

VOD：Video On Demand，视频点播

第一章 引言

1.1 研究背景与现状

视频已经成为人们生产和生活不可缺少的一部分。而在计算机技术和网络技术的推动下，视频应用场景也变得非常复杂。尤其是在网络占主导的今天，在线或实时视频的随时随地观看、通过视频进行的随时随地通信，大大方便了人们的信息获取和交流沟通。然而，这种复杂的应用场景也为视频编码技术提出了新的挑战。见图 1.1。

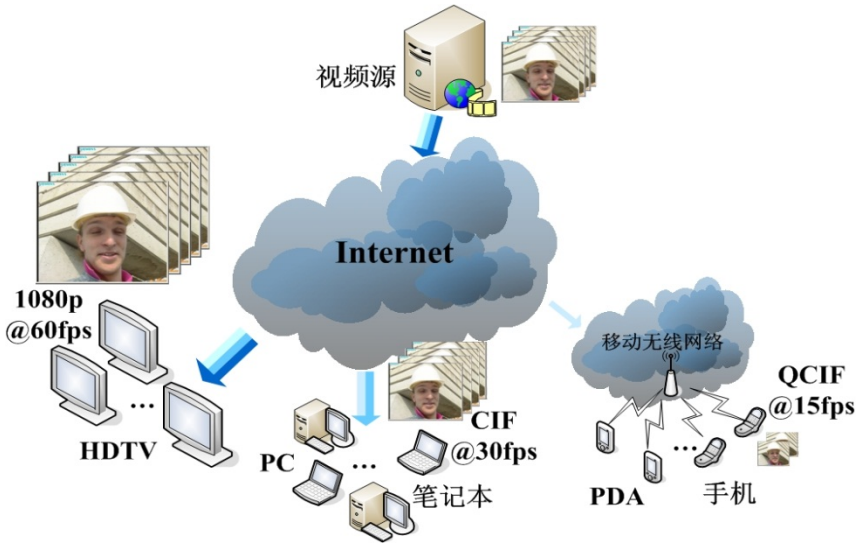


图 1.1：视频应用的复杂场景

一方面，从网络状况来看，目前既有几十 kb/s 速度的拨号上网，又有几十 Mb/s 速度的局域网或光纤接入方式，而且随着移动通信网和广播电视网与 Internet 的融合，无线与有线网络混联，使得带宽和信道质量更加不确定。在这样异构且波动的网络环境下，确定的视频流很难高效传输，也很难满足不同接入用户的需求。另一方面，从用户终端来看，运行视频应用的既可能是具有较强计算能力的 PC 机，也可以是只能进行低负荷运算的手机或 PDA 设备。显示装置既可能是高清的液晶显示器，也可能是只有 QCIF 大小（176×144）的手机屏幕。在这种情况下，传统编码技术得到的单一的视频流，很难满足不同用户

的视频体验。例如，当视频码率过高的时候，手机用户根本无法正常播放；而较低的视频码率又会使得具有较强处理能力的 PC 用户不够满意。综上，只有视频流本身具有自适应性或可伸缩性，才能满足多样化的需求。

以前的一些视频编码标准，如 MPEG-2、H.263、MPEG-4，在其高级特性中都提供了对可伸缩性的支持，但都将极大降低编码效率并增加解码复杂度。它们都没有被广泛应用。与之相反，一种称之为“联播”(Simulcast)的技术却在上述需求的推动下被许多视频网站（如优酷网）采用。这种技术原理很简单，就是预先编好不同码率的多个码流放在服务器上，根据网络和用户终端的情况选取其中合适的一个发送。这种方式提供了一定程度的适应性，但其缺点也是明显的。第一，编码、储存、传输多个码流使得时间和空间开销成倍增加；第二，这种方式一般只能提供很小的伸缩空间，如优酷网只提供了“普通”、“高清”的选择。

2003 年，国际电信联盟（ITU）和国际标准化组织（ISO）共同制定的新一代视频编码标准 H.264/AVC（Advanced Video Coding）获得通过[1]。这一新标准不仅大大提高了编码效率，同时也强化了对网络的友好性，以顺应视频业务网络化的趋势。 H.264/AVC 出台后，ITU 和 ISO 共同成立的联合视频专家组（Joint Video Team）又开始着手新一代可伸缩视频编码（Scalable Video Coding，SVC）的标准化工作[2]。2007 年，JVT 的成果被作为 H.264/AVC 的扩展纳入标准化体系[3]。由于 JVT 制定的可伸缩编码框架依托于最新的 H.264 标准，其编码效率和实用性相比之前的可伸缩编码技术有了很大提高。下文提到 SVC，一般特指 H.264 标准的 SVC 扩展。

SVC 提出以来，学术界对其进行了多方面的研究，工业界也在积极推进基于 SVC 的应用。2007 年以来的大量论文不仅对 SVC 编码过程中的层间预测技术、码率分配和控制、率失真优化（Rate-Distortion Optimization）等作了深入研究，也对 SVC 在 IP 和移动网络上的传输、文件格式与系统接口等方面进行了探讨。在工业界，各种视频产品和应用在完全跟随 H.264 的潮流之后，已经开始向 SVC 提供的可伸缩性靠拢。大多数采用传统编码的视频业务，如数字电视、视频电话、视频会议、基于流媒体技术的在线视频观看等，都将从 SVC 中得到改善。鉴于视频流媒体系统的普及，SVC 进入这一应用领域的速度也将更快。

流媒体（Streaming Media）是指采用流式传输的方式在网络上播放的媒体格式。与传统媒体播放不同，流媒体在播放前不需要下载整个文件内容，而是

将数据分包，在传送的同时就可播放，只在开始时有缓冲延迟。流媒体系统的关键技术在于流式传输，即服务器能够将多媒体文件分割成包，按特定的顺序发给播放客户端，而客户端能够正确组合收到的数据流并解码出媒体数据。目前，流媒体技术已经得到了广泛的应用，除了互联网上常见的在线视频点播（VOD）、直播，流媒体也出现在远程教育和医疗、网络广告、实时视频会议、网络视频监控等诸多信息服务领域。

流媒体系统具备三个核心组件，即播放器（包括解码器）、服务器、编码器。其中编码器主要用于将原始音视频数据压缩编码成能用于流式传输的文件，它对整个系统的性能起着关键的作用。从 H.264 和 SVC 标准公布后对视频编码的影响程度不难推测，流媒体应用中的编码器也将采用更灵活高效的新的编码技术并向支持可伸缩的方向发展，终将被 SVC 取代。流媒体服务器负责将数据发送至客户端，目前一些服务器已经能支持自动调整发送速度，但在视频源换用 SVC 编码后，流媒体服务器必须重新设计来实现对 SVC 可伸缩性数据发送的支持。不难预料，引入 SVC 特性之后，流媒体系统对网络条件和终端能力的适应性和鲁棒性将大大提高，这将是未来的必然趋势。

1.2 本文工作

本文研究可伸缩编解码在流媒体系统中的应用，并设计实现了一套支持 SVC 的视频点播系统。目前市场上几乎没有支持 SVC 的视频应用，本文的工作是对在流媒体系统中加入 SVC 进行创新性的探索和尝试。首先改造了现有的流媒体服务器软件，加入对 SVC 编码视频的支持，其次设计实现了支持 SVC 的播放终端。在系统完成之后，对其可用性做了测试和分析。

后文内容如下组织：第二章对可伸缩视频编码技术进行概述；第三章在介绍现有流媒体技术的基础上构建支持 SVC 的流媒体服务器；第四章实现了支持 SVC 解码的播放终端，即基于 Android 系统的手机播放器和电视机顶盒，并探讨了解码器优化问题；第五章对成果进行测试和分析；最后一章进行总结和展望。

第二章 可伸缩视频编码（SVC）技术概述

可伸缩视频编码（Scalable Video Coding，SVC），简言之，就是在视频编码时生成一个包含不同层次信息的码流。从这个单一码流中，我们能够方便地丢弃一些数据，得到不同码率的多个码流，从而解码出在空间大小、帧率、画面质量等方面可伸缩的视频。

正如引言中所说，可伸缩视频编码在学术和商业领域都得到了广泛的重视。这是由网络异构性、终端差异性所引发的对视频服务的多样化需求所推动的。首先，随着无线通信技术和接入技术的发展，广播电视网和移动通信网逐渐与互联网无缝衔接，走向“三网融合”，这导致目前的网络环境十分多样，带宽和服务质量可能有很大差异。因此，固定码率的传输往往无法得到保证。其次，接入网络的终端设备也越来越丰富，如普通 PC、掌上电脑、移动电话、电视机顶盒等等。这些终端具有不同的显示和处理能力，传统编码的视频不能良好地适应这些变化。虽然能够通过预先编好多个码流的方法来满足上述需求，但对同一视频多次编码显然大大增加了时间和存储上的成本。于是，“一次编码，多层利用”的可伸缩编码就成为众望所归。

基于 H.264/AVC 扩展而来的 SVC 在传统编码的基础上引入了“分层编码”的概念。即对数据源一次编码得到的码流中，含有一个或多个的“层”。其中最低分辨率、最低帧率和最差图像质量的部分称为基本层，除此之外为增强层。增强层对应着更高的分辨率、帧率和图像质量。这样的码流能够提供在空间、时间、质量（或 SNR，即信噪比）等方面的伸缩性。当终端没有能力播放增强层或者网络无法承担高码率时，可伸缩码流的增强层部分会从比特流中移除，只有基本层数据被传送到终端解码器。SVC 编码标准得益于 H.264/AVC 灵活的编码逻辑结构，也继承了 H.264/AVC 中许多良好设计的编码工具。SVC 不仅在编码效率和复杂性上与单层编码增加不多[4]，并且完全兼容 H.264/AVC。可以预见，它将在未来的视频应用中成为主流。

以下结合传统编码框架，在 H.264/AVC 的基础上，对 SVC 可伸缩性的实现原理进行概述。

2.1 H.264/AVC 简介

SVC 是在 H.264/AVC 的基础上扩展而成的，它利用了 H.264/AVC 的大部分设计特性和编码工具（事实上，SVC 被标准化也是作为 H.264/AVC 文档附录的形式）。为了分析 SVC 编码技术，有必要先对视频编码标准 H.264/AVC 进行介绍。

H.264/AVC 的设计目标有两方面，一是显著提高编码效率，二是在网络化的趋势下满足对灵活性和可定制性的要求。为此，这一新标准提出了两个概念：视频编码层（Video Coding Layer，VCL）用于高效的表示视频内容，而网络抽象层（Network Abstract Layer，NAL）则封装视频的 VCL 表示并提供头部信息，使之适用于多样化的传输层次和存储介质（参见图 2.1）。

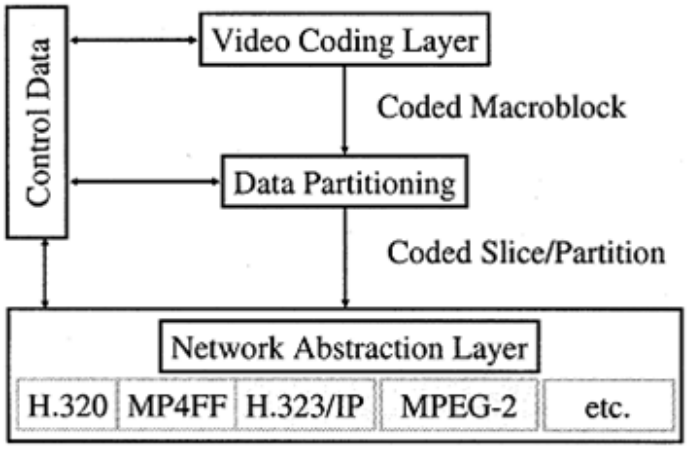


图 2.1：H.264/AVC 编码标准的结构[1]

图 2.1 中，H.320 是国际电信联盟制定的关于在 ISDN（综合数字交换网）上进行视频会议的标准，H.323 则是其 IP 电话方面的标准，MPEG-2 是 ISO 关于视听系统的标准，其中规定了音视频文件存储格式等。可见，经过 NAL 层的抽象，H.264/AVC 编码得到的码流可以灵活适应各种应用。下面，分别对 VCL 和 NAL 进行介绍。

2.1.1 视频编码层 VCL

VCL 层的目标是高效率地表示视频内容。与其他视频编码标准一样，H.264/AVC 的 VCL 层采用了基于块的混合视频编码框架。参见图 2.2。

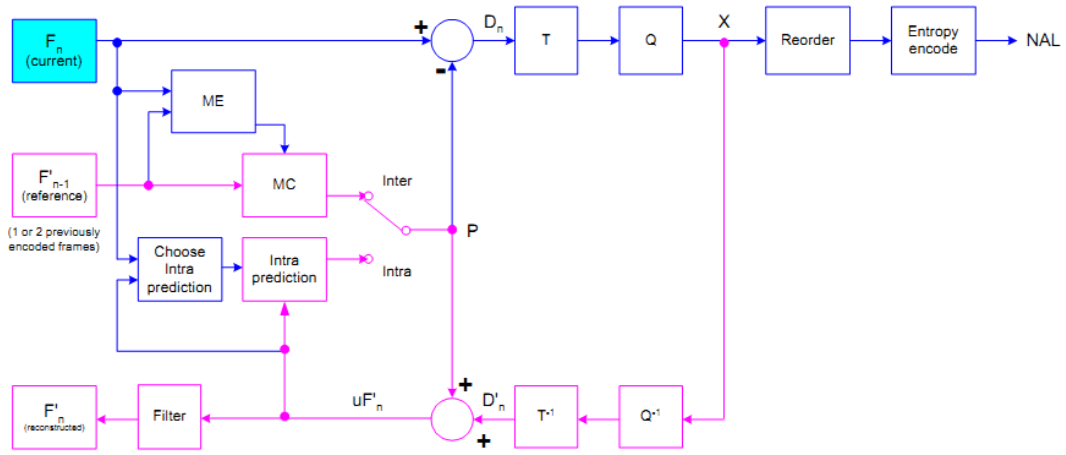


图 2.2：H.264/AVC VCL 层编码框架

所谓“基于块”，是指将原始样本数据分成 16*16 大小的宏块（Macro Block，MB），接下来的编码过程将以宏块为基本单位。对每一个宏块，先进行时间或空间预测；预测得到的残差数据进行类似于 DCT 的整数变换，然后对变换系数进行量化、重排、熵编码，最终形成比特流。

虽然这样的编码流程与之前的编码标准并无二致，但由于 H.264/AVC 对各个阶段进行了精心设计，并将灵活性和适应性贯穿于整个编码过程，使得其编码效率大大提高。以下仅着重分析 H.264/AVC 相对于之前标准的改进之处。

预测分为帧内预测和帧间预测，目的分别是去除空间和时间冗余。在 H.264/AVC 中，预测时可将一个宏块进一步分为小的子块进行，最小可以分成 4*4 的小块。标准还提供了多达 9 种的预测模式，从而能够更精细地利用局域特性。而对帧间预测，H.264/AVC 除支持单向和双向预测（P 片和 B 片）之外，每个方向还允许选用多个参考帧。在运动向量的准确度上，新的标准允许精确到 1/4 像素，而且还允许跨越边界的值（采用外推插值技术）。

对预测残差进行变换可以进一步去除空间相关性。考虑到精细的预测模式已经很好的去除了相关性，H.264 一改之前标准以宏块为单位的 DCT 变换，采用了更小的变换块（4*4）。而且对于相对平滑的色度值（YCbCr/YUV 中的 U、V），还对每个宏块中的 4 个 4*4 子块变换后的直流系数又做了一次哈达玛变换。小尺寸块的变换大大减小了计算复杂性。当然，H.264 也支持将变换块扩展成 16*16 大小。

H.264/AVC 提供了两种熵编码方法 CAVLC（Context Adaptive Variable Length Coding）和 CABAC（Context-based Adaptive Binary Arithmetic Coding）

来去除量化后系数（以及其他一些参数）的统计冗余。它们都是上下文自适应的，前者为可变长编码，后者为二进制算术编码，后者计算较复杂但压缩率高，前者反之，可根据应用场景配置不同的算法。

以上只选取 VCL 的部分特性进行了简述。除了所提到的方面，H.264/AVC 还包含更多可由编码器选择的配置。这样不仅能够更有针对性地去除冗余、提高编码效率，而且也增加了该标准的普适性、灵活性和可定制性。正因为如此，能够在其基础上方便地扩展得到可伸缩视频编码——SVC。

VCL 编码得到的比特流送去 NAL 层进行封装。

2.1.2 网络抽象层 NAL

NAL 的设计目标是提供网络友好性，使得 VCL 数据在不同系统中的使用可以简单而高效地定制。我们后面将看到，SVC 能够应用于流媒体系统并与普通的 H.264 视频无缝兼容，很大程度上得益于这种网络抽象层机制。

网络抽象层的基本概念和语法结构是 NAL 单元（NAL Unit，NALU）。这是一种由整数个字节组成的数据包。一个 NALU 以 1 字节的头信息起始，之后是不定长度的载荷。参见表 2.1。

表 2.1：NAL 单元结构

NALU header byte			NALU payload
F	NRI	TYPE	……

NALU 字节头包含禁止位(F，1bit) 、重要性指示位(NRI，2bit)、NALU 类型(TYPE，5bit) 三方面信息。其中禁止位规定为 0，重要性指示位在一定意义上表示该 NALU 的重要程度（或传输时的优先级），而 NALU 类型则表明该 NALU 的载荷类型。

在 H.264 标准中，NALU 类型的 0~12 已用，13~23 被保留用于以后可能的扩展（如 SVC 扩展用到了其中的值），24~31 不做规定，可根据应用场景定制。按照 NAL 单元中载荷的不同，将其划分为两类：VCL 单元和 non-VCL 单元。前者主要包含来自 VCL 的图像样本值编码数据，后者主要含有图像或序列参数集、补充增强信息（Supplemental Enhancement Information，SEI）等解码多个 VCL 单元所共用的一些重要数据。这种机制不仅能节省开销，也便于将重要数据通过更可靠的信道传输。

将解码数据以 NAL 单元的形式组织，既为不同系统中的应用提供了灵活性，也使得该标准易于扩展，是 H.264 的一大特色。通常来说，在基于分包传输的应用中（如 RTP/UDP），一个 NAL 单元可以构成一个包；而在字节流形式的应用中（如文件存储），则不同 NAL 单元之间需要插入特定分割符。后者不在本文范围之内，故不做讨论；对如何使用 RTP 协议传输 H.264 编码数据，后文会有涉及。

2.2 扩展 H.264 实现可伸缩编码

视频可伸缩性一般包括三个方面：时间可伸缩，即帧率的变化；空间可伸缩，即图像大小（分辨率）的变化；质量可伸缩，即 SNR 的变化。SVC 标准可同时支持这三个方面的伸缩特性。

2.2.1 时间可伸缩

具备时间可伸缩的码流能够根据需要动态改变视频帧率。在 H.264 标准中，把用于解码一帧的多个 NAL 单元称为一个 AU（Access Unit）。SVC 就是通过丢弃 AU 来降低帧率。具有 3 个时间层的可伸缩实例如图 2.3 所示，其中从上到下虚框表示的 AU（即视频帧）逐渐被丢弃，从而帧率以 1/2 倍递减（这只是一种简单情况，事实上，SVC 支持不限于 2 倍的伸缩特性）。

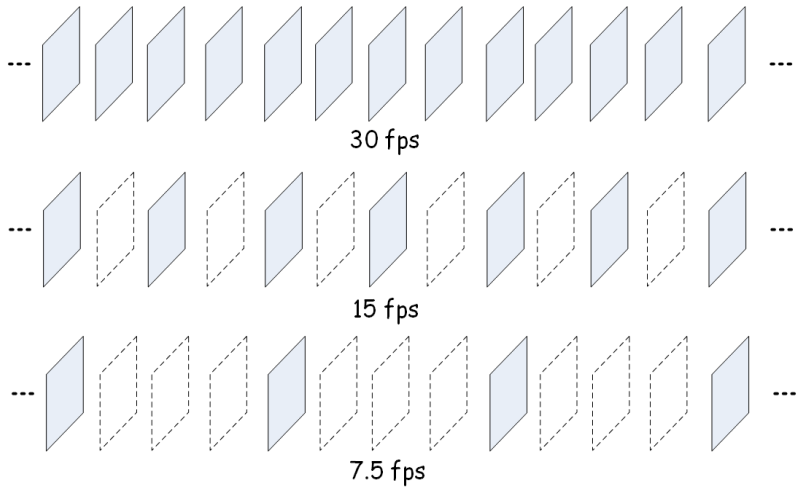


图 2.3：时间可伸缩图示

时间可伸缩的实现不需要对 H.264 做任何增加，只需采用一种称为“层次

化预测结构”的技术。

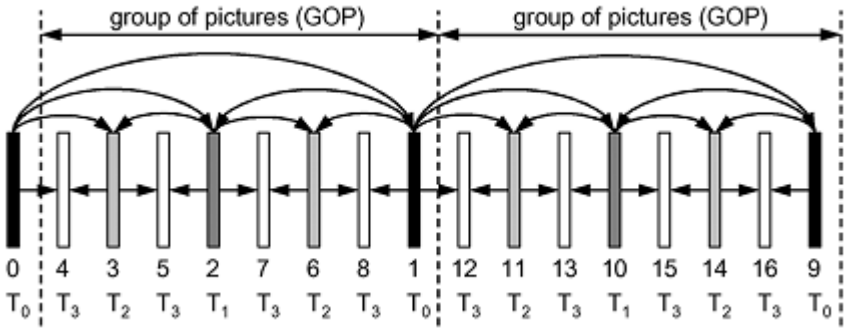


图 2.4：层次化预测实现时间可伸缩

图 2.4 说明了通过层次化预测来实现时间可伸缩的原理。图中一个 AU 下面的数字表示其编码顺序，数字下面的符号 T_i 表示其属于第 i 时间层。增强层（i>0）的帧被编码为 B 帧（即参考其前后的帧进行双向预测），而且只参考较低层的帧。这样的结构下，低层的 AU 可以不依赖于高层，即任一较高层及其以上各层的 AU 可以被丢弃而不影响解码，以此实现帧率的伸缩性。

需要说明的是，此例中这种“层次化 B 帧”的结构会带来编解码时延，且仅限于 1:2 伸缩比，这些不足均可通过仔细设计层次化预测结构而改善。

2.2.2 空间可伸缩

空间可伸缩是指视频大小的改变。图 2.5 形象地展示了空间可伸缩的效果和应用场景。

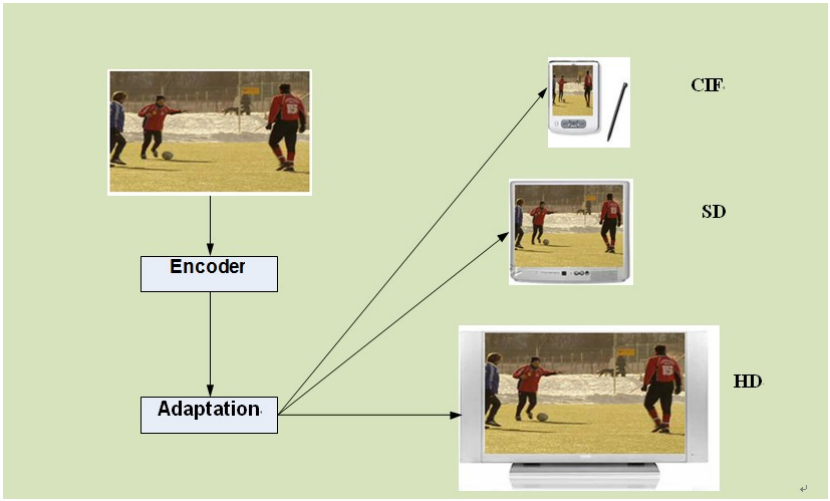


图 2.5：空间可伸缩的效果

注意这里的图像大小改变并非在解码后进行了缩放，而是传送给终端的码流本身就是分辨率可变的。空间可伸缩的实现比较复杂，需要在 H.264 中引入分层编码。每个分辨率都对应一个 H.264 编解码层，称之为“dependency layer”，由 DID（Dependency ID）标记。每个层都需要不同尺寸的输入图像，这一般由最高分辨率的图像下采样得到。每层的运动补偿和帧间预测都和单层 H.264 编码一样，然而由于表示的是同一图像，各层之间必然有很强的相关性。为了提高编码效率，SVC 规定了各层之间的预测机制，称之为“层间预测”。图 2.6 展示了空间可伸缩编码的复杂结构。可见，每个空间层都是一个 H.264 编码器，它们之间通过层间预测相联系。

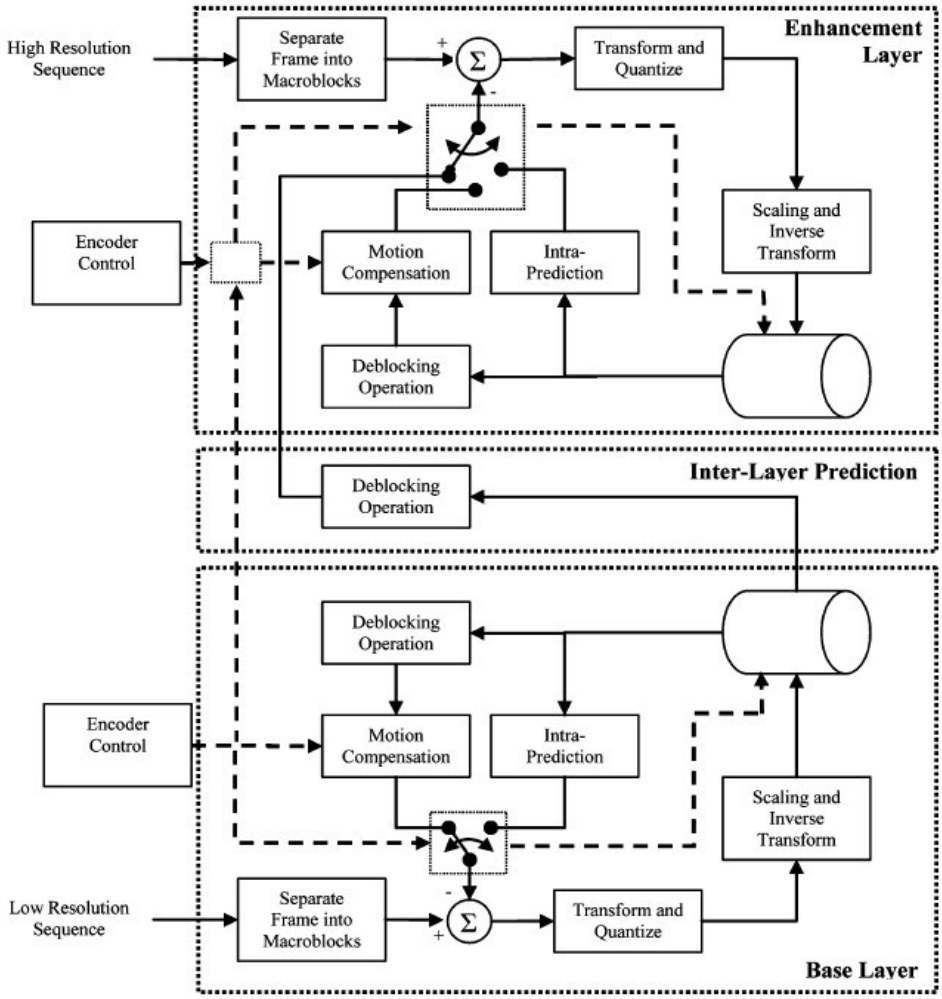


图 2.6：空间可伸缩编码结构[5]

2.2.3 质量可伸缩

质量可伸缩顾名思义就是视频各帧的图像质量可变。这是通过丢弃一个AU中的一些质量精细化NAL单元来实现的。具体操作时，给一个AU内的每个NAL单元打上质量层标记QID（Quality ID），可以将QID高于某个值的所有NAL单元丢弃，剩下的即可组成较低质量的码流。

2.2.4 对NAL的扩展

以上分析的主要是在VCL层实现可伸缩编码的原理，可以看到多处涉及对特定NAL单元的取舍。这就需要对H.264的NAL层进行扩展从而能够在码流中标记和传送特定的可伸缩性信息与数据。

在前面介绍H.264/AVC时曾提到，NALU头字节中的NAL类型（nal_type）在13~23的取值留作扩展；SVC就增加了一些扩展的NAL类型。nal_type=20的单元为新增的VCL单元，包含的是SVC中增强层的编码数据；nal_type=14为新增的SVC前缀单元，它可能是VCL的也可能是的non-VCL的，取决于紧接着它到来的NALU。此外，还有nal_type=15的NALU来传送SVC特有的图像参数集。关于这些新增类型的具体分析可以参见文献[6]。

普通H264解码器遇到nal_type大于12的单元会忽略之，但能成功解码基本层，因此SVC可与之兼容。而支持SVC的解码器将对类型为14、20的NALU进行利用，实现可伸缩性。与普通H.264的NALU不同，类型为14、20的NALU并非只有1个头字节，而是扩展为4字节的头。如图2.7所示。

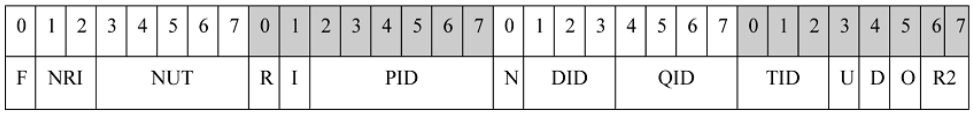


图2.7：4字节的SVC NAL单元头结构[6]

可以看出，其中除了与普通NALU一致的第一字节，还包含了空间、时间和质量层的标志（DID、TID、QID）以及其他一些用于提供可伸缩性的信息。

2.3 SVC编码器程序

根据以上分析，可在标准H.264编码器的基础上扩展得到SVC编码器。无

论 H.264 还是 SVC，在标准文档中都只对码流的语法语义和解码过程做了规定，换句话说：对编码的过程不做限制，只要得到符合标准的码流即可。因此编码器的程序实现可以多种多样。对 SVC 编码器程序的详细讨论不在本文范围之内。制定 SVC 标准的联合视频小组（JVT）提供有 SVC 编码器的参考软件，可以在此基础上实现自己的编码器程序。

第三章 构建支持 SVC 的流媒体服务器

3.1 流媒体技术现状

本节先对已有的流媒体技术做介绍。目前的基于流媒体的视频应用已经有很多，典型的为在线视频观看，包括点播和直播。但现有的应用中尚未采用 SVC 所提供的特性。本文正是在现有技术的基础上进行改进，将 SVC 应用于流媒体系统。

3.1.1 流媒体系统基本原理

流媒体系统之所以能够在不必下载完整音视频文件的情况下进行播放，主要是因为采用了流式传输。流式传输的对象是存放在服务器上的流式文件。生成流式文件是流媒体系统的起点。流式文件经过了特殊编码，使其适合在网络上边下载边播放。对原始的音视频数据，首先会经过压缩处理（即标准媒体文件的压缩编码），之后将数据分成适当大小的分组并考虑差错恢复功能，此外一般还要加入特定的时间戳和索引（hint 或 index）信息，用于提供如快进、后退和随机访问等控制功能。上述过程称为媒体文件的流化。流化得到的流式文件存放在服务器上等待传输。

流式传输是流媒体系统的关键。通常流式传输可以分为两种：顺序流式传输（progressive streaming）和实时流式传输（real-time streaming）。顺序流式传输虽然也能边下边播，但用户只能观看已下载的部分，不能跳到未下载的前面部分。这种流式传输方式通过 HTTP 即可实现，不需其他特殊协议，但也因功能和特性太少而逐渐被取代。目前广泛应用的是实时流式传输。这种方式允许用户对媒体流的发送进行更多的控制，可随机访问前后内容。相应地，这种传输方式也需要特定的服务器和特殊的协议，如 RTP 和 RTSP。关于服务器和流媒体协议的介绍留待后文，下面先给出典型的流媒体视频点播系统的基本框架。如图 3.1。

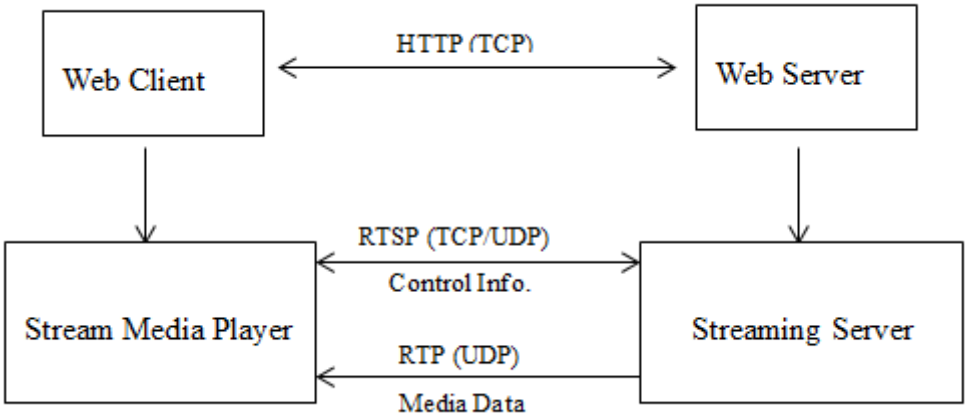


图 3.1：典型的流媒体应用框架

通过流式传输播放在线视频的一般过程如下：

- 用户检索和选择视频；在 Web 浏览器与 Web 服务器之间通过 HTTP/TCP 进行通信。
- Web 浏览器利用从服务器得到的参数对客户端的流媒体播放起进行初始化；这些参数包括流媒体服务器地址、目录和媒体文件信息等。
- 播放器与流服务器之间运行实时流控制协议（RTSP），进行 A/V 数据传输的控制，如准备、开始、暂停、后退等。
- 流服务器使用实时传输协议（RTP）将媒体数据传送给客户端，数据抵达时即可解码播放。

流媒体是面向网络的应用，既需要终端软件，也需要传输和控制协议。本节剩余部分中，3.1.2 对流媒体用到的主要协议进行介绍，3.1.3 介绍服务器和客户端软件。

3.1.2 流媒体中的传输与控制协议

（一）实时流协议（RTSP）

实时流协议（Real-Time Streaming Protocol，RTSP）是一种应用层协议，其目的是为了在 IP 网络中有效传输流媒体数据。RTSP 本身并不发送媒体数据内容，它主要起到建立流和控制流的作用。RTSP 类似于 HTTP，在客户端和服务器之间发送请求和回应。一般会有以下交互：

- OPTIONS: 客户端询问服务器有哪些操作，服务器进行回应，列出可

用的方法。

- DISCRIBE：客户端向服务器请求描述信息，服务器以会话描述协议（Session Describe Protocol，SDP）的形式返回该信息，包括音视频的编码类型、参数，控制通道等等。关于 SDP，后文介绍 H.264/SVC 流时会具体涉及。
- SETUP：客户端在得到媒体信息后建立流；对音频、视频需要分别建立。
- PLAY（类似的有 PAUSE、SCALE 等）：开始播放媒体（或进行控制）。服务器在收到 PLAY 后会开始以 RTP 协议发送媒体数据包。
- TEARDOWN：客户端请求关闭流。

可见，RTSP 只在播放流媒体之前和结束时发挥作用。实际传输数据采用的是 RTP。

（二）实时传输协议（RTP）

实时传输协议（Real-time Transport Protocol，RTP）负责传输媒体数据。它是单向的，将流媒体内容以包的形式从服务器发送至客户端。RTP 包由头信息和载荷组成[7]。包头中提供了版本号、有效载荷类型、时间戳、序列号等信息，并且允许扩展。包的载荷则是实际的音视频编码数据。接收端解码时必须知道 RTP 载荷中数据的编码方式，包头中的有效载荷类型即标识了这一信息。有效载荷类型域共 7bit，可表示多达 128 种载荷。除了标准中指定的类型外，还可以进行扩展。

可以用指定有效载荷类型的 RTP 包来表示所传送数据的编码方式为 H.264。此时，RTP 载荷（payload）将呈现为 H.264 定制的结构[8]。H.264 的一个 NAL 单元大小不定，有时会超过一个 RTP 包的容量（受限于 RTP 下层协议的包大小，如 UDP）；有时又较小，可将多个 NAL 单元封装在一个 RTP 包中。为此，针对 H.264 的 RTP 载荷将第一个字节作为头字节来标记上述的分拆和组合。事实上，H.264 的 RTP 载荷的第一字节采用的结构与 NAL 单元的头字节相同，并且各个比特的意义也一致。其中后 5bit 表示该 RTP 包中的 NAL 单元的类型，若为 0~23 则表示 H.264 标准的一个完整 NAL 单元，大于 24 的被用来表示拆分或组合 NAL 单元。

RTP 为包括 H.264 在内的不同标准流媒体数据提供了有效的传输支持。但它却是不可靠的，它本身没有任何错误检测或拥塞控制机制，因此无法提供 QoS

（服务质量）保证。RTP 底层可以使用 UDP 也可以使用 TCP，大多采用的是 UDP。而且 RTP 一般与下面提到的 RTCP 配合使用。

（三）实时传输控制协议（RTCP）

实时传输控制协议（Real-time Transport Control Protocol，RTCP）被设计用来监测流媒体传输质量并在 RTP 会话参与者之间传递信息。RTCP 包可携带不同类型的信息，其中最典型的是接收者报告。这种 RTCP 包由接收者向服务器发送，报告其收到的 RTP 包数目、丢包数、包的抖动、延时情况等。这相当于提供一种反馈信息，发送端应用程序可以利用这些信息做出调整，如改变发送速率等。一般流媒体软件都会利用 RTCP 来控制和保证传输质量。

3.1.3 服务器端和客户端软件

实用的流媒体系统离不开服务器端软件和客户端播放软件。在流媒体服务器上运行的程序负责响应连接请求，从存储系统中取出音视频数据并以流式传输的方式发送给客户端。完整的流媒体服务器平台需提供会话服务、内容服务、流服务、媒体数据存储管理等（图 3.2）。

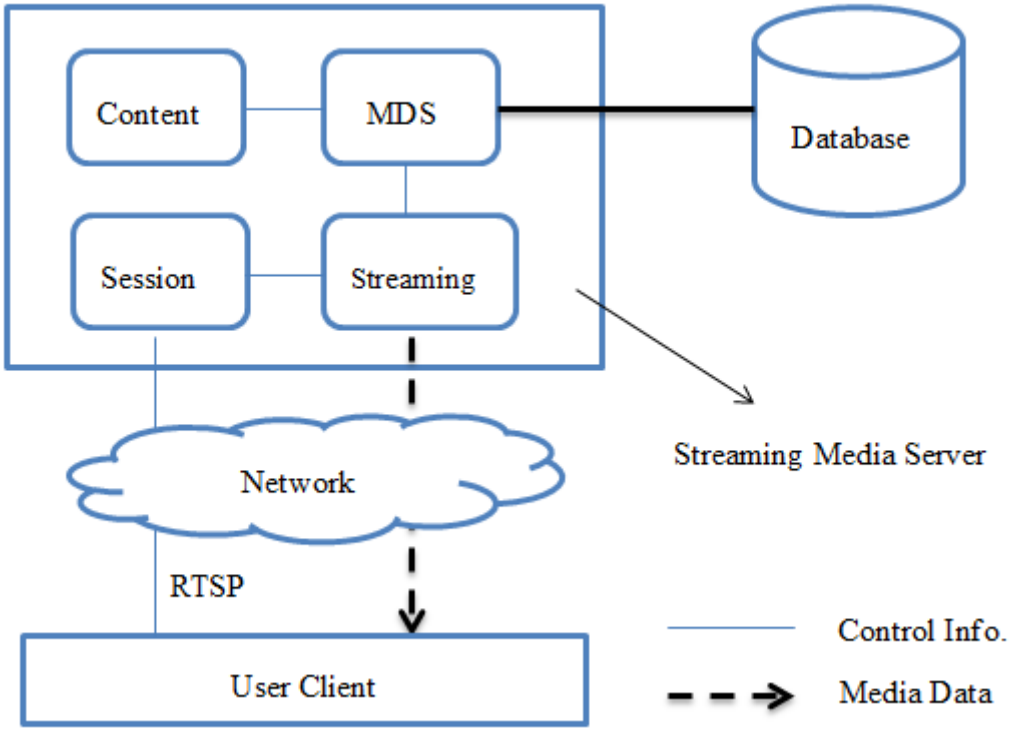


图 3.2：流媒体服务组件

用户端软件主要是支持播放流媒体（复杂的客户端还可能有节目查询等），为此除了普通媒体播放功能，还需要实现 RTSP、RTP 等网络协议。由于仅流媒体服务器中的流服务组件和支持 SVC 的媒体播放属本文关注内容，因此对其他组件不再涉及。

这里指出，目前市场上应用较多的流媒体技术和平台主要有三种：RealNetworks 公司的 RealSystem，微软公司的 Windows Media，以及苹果公司的 QuickTime。它们分别有各自的流媒体服务器和播放器。苹果公司的流服务器为 QuickTime Streaming Server，它同时提供开源版本，称为 Darwin Streaming Server（DSS）[11]。本文所实现的支持 SVC 的流媒体系统服务器端就是在 DSS 基础上修改而成。

3.2 在现有流媒体服务器中加入对 SVC 的支持

Darwin 流媒体服务器包含多个工程，其中负责流发数据的主要集中在 StreamingServer 工程中。在这里，由 RTSP 协议与客户端建立连接会话，并在客户端的 SETUP 命令下建立多个音频和视频的流，然后由 RTP 协议向客户端发送数据。每个会话对应一个 RTPSession 类，每个流对应一个 RTPStream 类，RTPStream 类的 Write 函数起到实际的发送数据功能。实现对 SVC 的支持主要是在此类中改变向流中写数据的逻辑。当然，前提是数据的来源必须是 SVC 的。

Darwin 服务器处理的媒体数据来源于特定的 MP4 文件。正如前文提到的，与本地播放的普通 MP4 文件不同，用于流式传输的 MP4 必须经过 hint 处理（即流化）。流化后的 MP4 文件不仅包括对应音、视频的 audio track 和 video track，还包括一个 hint track，其中信息用于流式传输。SVC 的 MP4 文件区别于非 SVC 的地方就是在 hint track 中有 SVC 伸缩层的数目信息，因此要采用特定的 SVCCreator 工具来打 hint。SVCCreator 在打 hint 时会将 SVC 视频流所支持的空间、时间、质量层数目写入 hint track。这样流化得到的 MP4 文件就是支持 SVC 的，可送给 Darwin 服务器。改造 Darwin 服务器支持 SVC 的工作主要包括在会话描述信息中加入伸缩层数目，能自动调整发送的层，以及能由客户端指定发送层的上限。

3.2.1 在描述信息中加入伸缩层数目

Darwin 服务器在接受到客户端的 DESCRIBE 命令（以 RTSP 发送）后，会解析 MP4 文件中的相应媒体信息，以 SDP 的形式发给客户端。SDP 是一种描述协议，其中包含了音视频的参数，其中可以任意添加参数项。通常采用 RTP 作为媒体传输方式的系统都要用到 SDP[9]。下面是一个 SDP 内容的实例：

```
v=0
o=StreamingServer 3516069823 1303912527000 IN IP4 172.31.203.21
s=\TheFoundingOfARepublicHD.mp4
u=http:///
e=admin@
c=IN IP4 0.0.0.0
t=0 0
a=control:*
a=range:npt=0-8440.36800

m=video 0 RTP/AVP 96
a=control:trackID=3
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=676400;
    sprop-parameter-sets=Z2QAKqxWJAUARbARAAZdOwExLQCPGjWK
    gA==,b1YAKksViQFAEWwEQAGXTsBMS0Ajxo1ipUg=,b1YAKmsVi
    QFAEWwEQAGXTsBMS0Ajxo1ipUg=,aOie8sA=,aEqJ4byw,aG6J4M8s;
    packetization-mode=1
a=svclayercount: 2/0/2

m=audio 0 RTP/AVP 97
a=control:trackID=4
a=rtpmap:97 MPA/48000
a=mpeg4-esid:2
```

我们在描述视频的参数中加入 svclayercount 这一项，把从 hint track 读到的空间、时间、质量层数目告诉客户端。上面示例中的“a=svclayercount: 2/0/2”表明这一视频除了基本层外，还有两个时间增强层和两个质量增强层。这就使得

客户端可以主动设置想要接收的层。这只是提供一种支持，其实客户端也可以完全不主动设置，服务器会根据网络质量和客户端解码能力自动调整发送的增强层数据。

3.2.2 自动调整发送的 SVC 层

SVC 的最大优势就在于一次编码得到的码流含有不同的层，能够根据需要选择其中某个层传输和播放。支持 SVC 的流媒体系统必须实现这一点。

Darwin 服务器本身具有一定的自动调整功能，能通过判断发送的 RTP 包的延时、丢包率等反馈信息，来决定减少或增加发送数据。其原理是：如果延时较大或是丢包较多，则认为网络情况较差；这样文件读取模块在读取 MP4 文件的时候，就跳过一些不影响继续播放的数据（视频质量有所牺牲）。然而，对于非 SVC 编码的数据源，这种调整是很有限的，因为视频码流本省的伸缩性就很小。对于 SVC，这样的自动调整就是基本层和不同增强层间的选择，视频码率可以有很大改变。

由于 SVC 基于 H.264，以 NAL 来组织编码数据，因此调整发送数据主要是对 NAL 单元的取舍。与 Darwin 原有机制不同，我们读 MP4 文件时把所有数据都读到，而在向流发送时舍弃某些高层的 NAL 单元。具体到操作，即在 RTPStream 的 Write 函数中增加 SVCQualityFilter。SVCQualityFilter 根据当前激活的是哪个层来对 NAL 单元进行过滤。因为每次写入流的对象是一个 RTP 包，故 SVCQualityFilter 的处理对象也是一个 RTP 包。前面提到，一个 RTP 包可以含有多个或不到一个 NAL 单元，因此 SVCQualityFilter 需要针对不同情况分别处理。其代码逻辑如可由图 3.3 所示的流程图来表示。

至于如何确定当前应该激活那个层，我们可以继承 Darwin 原有的机制，即根据发送的 RTP 包的延时、丢包率等反馈信息来判断。客户端向服务器发送的 RTCP 协议包正是为了提供这种反馈信息。当然 Darwin 原有的“质量层”（参数 QualityLevel）被 SVC 中的时间、空间、质量的各个增强层所替代。

此外，除了服务器自主确定该发送哪个层的数据，我们还允许在客户端由用户进行限制。

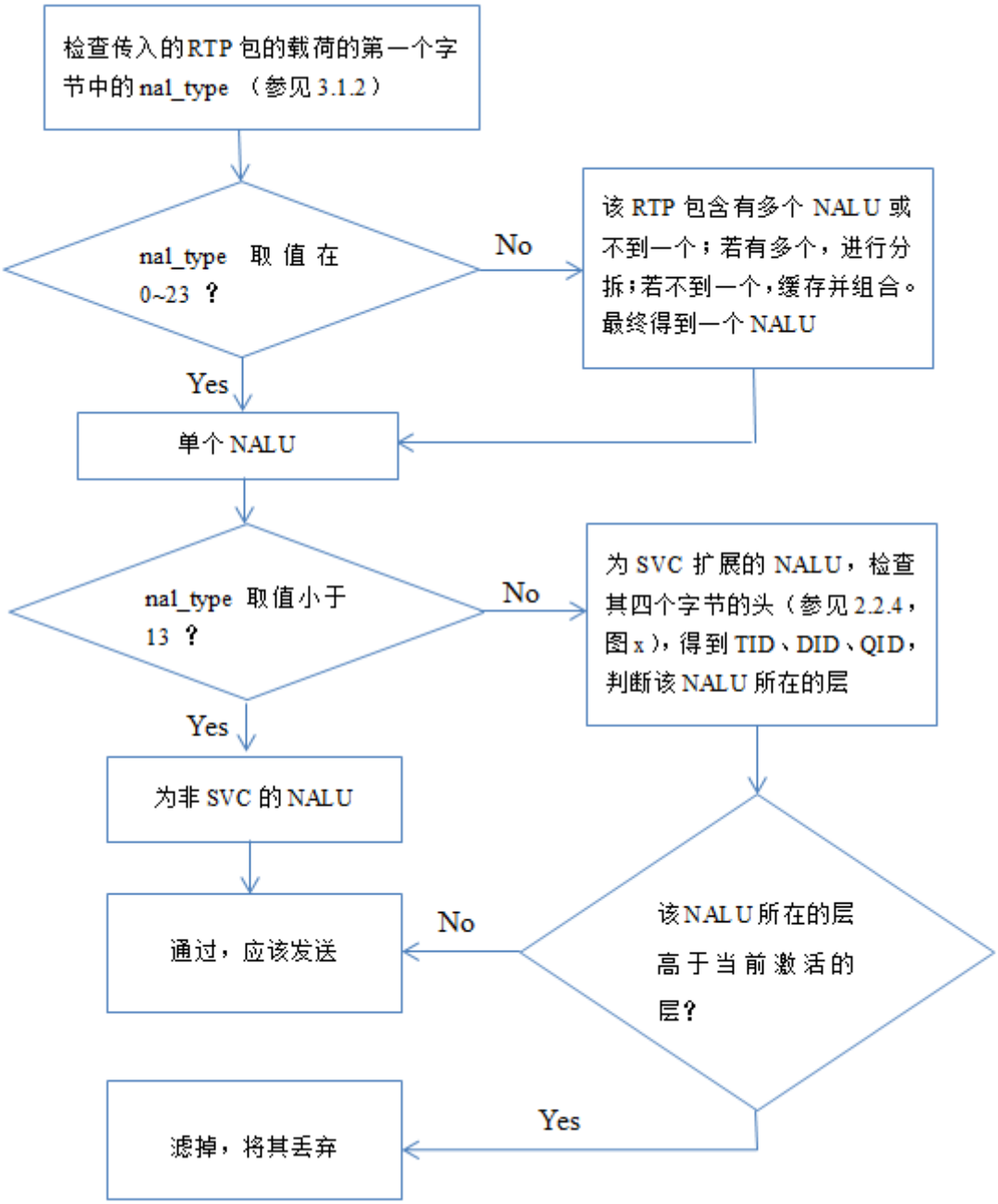


图 3.3：SVC 流媒体服务器对 NAL 单元进行过滤的流程图

3.2.3 支持客户端对发送的层进行设置

考虑这样一个场景：对一个手机用户（或其他使用网络流量套餐的用户）来说，每月的流量是有限的。他可能希望以节省流量的码率看较低质量的视频，而不是获得最好的观看体验。即使其所处网络状况很好且终端播放能力足够，也不希望服务器以较高增强层发送数据。为此，我们允许用户对发送的层数进

行指定。

3.2.1 中已经在媒体描述信息中加入了增强层的数目，终端播放器可以获取，然后指定它希望播放的层。在服务器端，我们要做的是接收客户端的指示并据此过滤 NAL 单元的发送。这可借助 RTSP 协议中 PLAY 命令来实现（参见 3.1.2 中内容）。按照 RTSP 协议，客户端与服务器间以 SETUP 命令建立起流（stream）之后，由客户端发送 PLAY 命令开始播放。发送 PLAY 时，协议允许附加一些参数。服务器实现为可以接收 x-Max-SVC-Quality-Level 参数来表示指定的 SVC 最高增强层，参数值取“tid/sid/qid”的形式。tid、sid、qid 依次表示所指定的时间层、空间层、质量层（对应 NALU 头信息中的 TID、DID、QID）。服务器在解析 PLAY 命令时记录该参数值，将其作为过滤 NAL 包时激活哪些层的依据。

以下为客户端发送 PLAY 命令的一个实例：

PLAY
rtsp://59.108.48.21:9108/TheDetectiveConanTheLostShipinTheSky.mp4/ RTSP/1.0

CSeq: 11

Accept: application/sdp

x-Max-SVC-Quality-Level: 2/0/1 joint

x-prebuffer: maxtime=5.000000

Session: 19421842134141

User-Agent: 7DLive RTSP 1.0

其中“x-Max-SVC-Quality-Level: 2/0/1 joint”表示客户端希望传输的视频码流中含有两个时间增强层和一个质量增强层（0/0/0 表示只发送基本层）。客户端可以在播放的过程中多次发送 PLAY 命令来传递 x-Max-SVC-Quality-Level 参数。服务器只会把第一次当作开始播放的命令，之后的都仅作为指定层数的命令。

以上基于开源的 DSS 实现了支持 SVC 的流媒体服务器端软件（同时保持对非 SVC 流媒体的兼容）。该服务器能够根据网络情况自主调整发送不同的视频增强层数据，也允许用户对发送的层数进行设置。有了这样的服务器，只需再有支持 SVC 视频播放的终端，即可组成一个完整可用的 SVC 流媒体系统。

第四章 支持 SVC 的播放终端设计与实现

4.1 概述

目前大多数播放器都支持流媒体播放。最新的视频编码标准 H.264/AVC 出台以后，用 H.264 编码的 MP4 文件越来越流行。不仅播放器软件中加入了 H.264 的解码核心，而且各芯片厂商纷纷实现了 H.264 的硬件解码。可以说，H.264 已经迅速得到普及。

但由于标准的 SVC 扩展才制定不久，目前尚无支持 SVC 的播放设备。可伸缩视频编码天然就是面向网络的，所以其应用也必将从网络视频，即流媒体系统入手。本文前面已经构建了支持 SVC 的流媒体服务器，本章设计并实现能够解码 SVC 码流的播放终端。

考虑到目前智能手机越来越流行，在手机上随时随地观看在线视频正成为一种时尚。手机上的流媒体应用对一般人来说都要受流量的限制，而且手机的网络状况也比较多样，传输质量无法保证。在这种情况下， SVC 恰可以体现其码率可变、适应不同网络条件的优势。同时，现在 IPTV 也逐渐走向家庭，在家里通过电视机加机顶盒观看网络视频也成为很多人的选择，这也可以成为 SVC 的用武之地。总之，对 SVC 播放器的需求会越来越多。

Google 公司于 2007 年末推出了其基于 Linux 内核的 Android 操作系统，开始进军智能手机领域。目前该系统不仅在智能手机上应用广泛，而且逐渐成为平板电脑（上网本）、电视机顶盒、车载 PDA 等嵌入式产品的软件平台。Android 操作系统开放源代码，可以被定制，适合在其基础上开发自己的产品软件。

经过上述调研，我们确定了设计并实现基于 Android 的 SVC 播放终端的计划。在手机和机顶盒上均可采用 Android 操作系统（目前市场上这样的产品越来越多），这就使得为 Android 手机编写的程序可以方便地用于机顶盒。相对于手机来说，电视机顶盒要涉及比较复杂的硬件系统，其对 SVC 的支持目前尚在开发中；但从软件上看，由于都基于 Android 平台，二者是类似的。本文以下只对所实现的 Android 手机 SVC 播放器做详细介绍。

4.2 Android 手机播放器 SVPlayer

Android 程序开发使用 Google 公司提供的软件开发包（SDK）。Android 系统底层基于 Linux 内核，使用 C 和 C++语言；顶层则是虚拟机架构，使用 Java 语言。这之间的联系采用 Java 平台中的 JNI 技术。JNI 是 Java Native Interface 的缩写，意为 Java 本地接口，它的目的是允许 Java 与其他本地语言（如 C/C++）之间互相调用。关于其具体介绍可参考相关文档，这里仅说明其基本用法：将 C/C++语言编写的代码按照 JNI 的特定要求编译为动态库（Linux 下为.so 文件），在 Java 程序中载入该动态库，即可调用其中的方法。为方便开发者，Google 公司专门发布了 Native Development Kit（NDK）。它与 SDK 搭配，可以将 Java 语言和 C 语言结合起来编写高效的 Android 程序。一般 Java 适合编写程序界面，而对效率要求高的计算过程（如音视频解码）需要用 C 语言实现。

本文所开发的 SVC 播放器称为 SVPlayer，其解码核心以及播放和同步均用 C 语言编写，程序界面用 Java 实现。

4.2.1 SVC 解码核心

虽然 Android 设备几乎都提供了标准 H.264 的解码硬件，可以不耗 CPU 地播放标准编码 H.264 编码的 MP4 文件，但 SVC 的解码只能通过软件实现。

SVC 标准由 ITU 与 ISO 的专家组成的联合视频小组（JVT）制定。该组织在标准形成后还发布了一个官方的参考软件，称为 JSVM（Joint Scalable Video Model）。该软件完整实现了 SVC 的解码过程，但是拿它来编写实用软件是不适当的。首先，该参考软件比较庞大，采用面向对象的方法，囊括了 SVC 标准中的几乎所有特性，有很多在实际系统中很难用到。其次，该软件没有考虑任何优化，速度很慢，根本无法完成实时解码。虽然如此，该程序清晰的逻辑和对 SVC 全面的支持却适合学习和参考。本文中 SVPlayer 的 SVC 解码核心就是参考 JSVM 的逻辑，对开源编解码库 FFmpeg 中的 H264 解码器改造得到。

FFmpeg 是一个被广泛应用的编解码库。它是开放源代码的，由 C 语言实现，支持大多数流行的编解码器。FFmpeg 提供了一套音视频编码、转码、解码的完整方案，不仅用于很多 PC 软件，还被移植到各种嵌入式设备中。在 Android 手机上编写播放器时，首选的解码框架就是 FFmpeg。

FFmpeg 由 libavcodec、libavformat、libavutil、libswscale 等几个不同的库构成。其中最重要的是 libavcodec 和 libavformat（libavutil、libswscale 提供一些辅助函数以及对图像进行缩放的方法，）。就解码而言，前者包含各种音视频解码器的实现，负责实际解码过程；后者则包含解复用器（demuxer）和网络协议

（如播放流媒体时的 RTSP/RTP 等）的实现，用于解析文件格式，获取和拆分数据。当然它们也有互相依赖之处。SVC 解码核心主要是在 libavcodec 库中实现。

当前的 FFmpeg libavcodec 中已经加入了对最新视频编码标准 H.264 的支持。在 FFmpeg 中，各种编解码的实现被称为一个个的 CODEC（COder & DECoder）。在符合 FFmpeg 程序框架的条件下可以注册加入第三方 Codec。由于 SVC 是 H.264 的扩展，为方便起见，我们可以直接在 FFmpeg 中的 H.264 decoder 基础上进行修改，加入 SVC 的解码逻辑即可。当然，更佳的方法是将 SVC 特性封装成一个新的 decoder 进行注册，这样符合 FFmpeg 的要求，也便于配置。

FFmpeg 的视频解码流程一般是这样的：打开输入文件（本地文件或网络流）后，会根据文件提供的头信息寻找相应的解复用器和解码器。它通过解复用器把文件解析成一个一个的 packet(储存在结构体 AVPacket 中)，然后调用解码器提供的 decode_frame 函数将 AVPacket 解码成 AVFrame，即得到以 YUV 表示的一帧。因此，解码器主要实现的接口为 decode_frame 函数。对于 H.264 解码器，该函数在 h264.c 文件中。这个文件包括了 H.264 解码所需的全部逻辑。在 decode_frame 中，先解析出 NAL 单元，将其送给 decode_nal_units；decode_nal_units 函数中判断 NALU 头字节中的 nal_type，据此分别进行处理。标准 H.264 只关心 nal_type 取值 1~12 的 NALU，而对其进行扩展主要就是在这里增加分支，对 nal_type=14/20/15 的 SVC NALU 进行处理。如对 nal_type=20 的 SVC 图像片(slice) NALU，添加 decode_svc_slice 函数；等等。当然，实现空间可伸缩引入的“层间预测”等技术也要求对标准 H.264 相应的预测代码进行修改。

在对 H.264 解码器的改动中，只是增加了对 SVC 扩展的处理，原有解码逻辑并没有改变。这样得到的解码器核心不仅支持 SVC 视频流，对标准 H.264 码流也能够前向兼容。

得到 SVC 解码核心后，即可利用它编写支持 SVC 的视频播放器。

4.2.2 音视频播放与同步

本文开发的 SVPlayer 是基于 FFmpeg 框架的播放器，不仅支持 SVC 播放，对已有的几乎所有格式（FFmpeg 所支持的）都能使用。

图 4.1 是音视频播放程序的结构模块图。

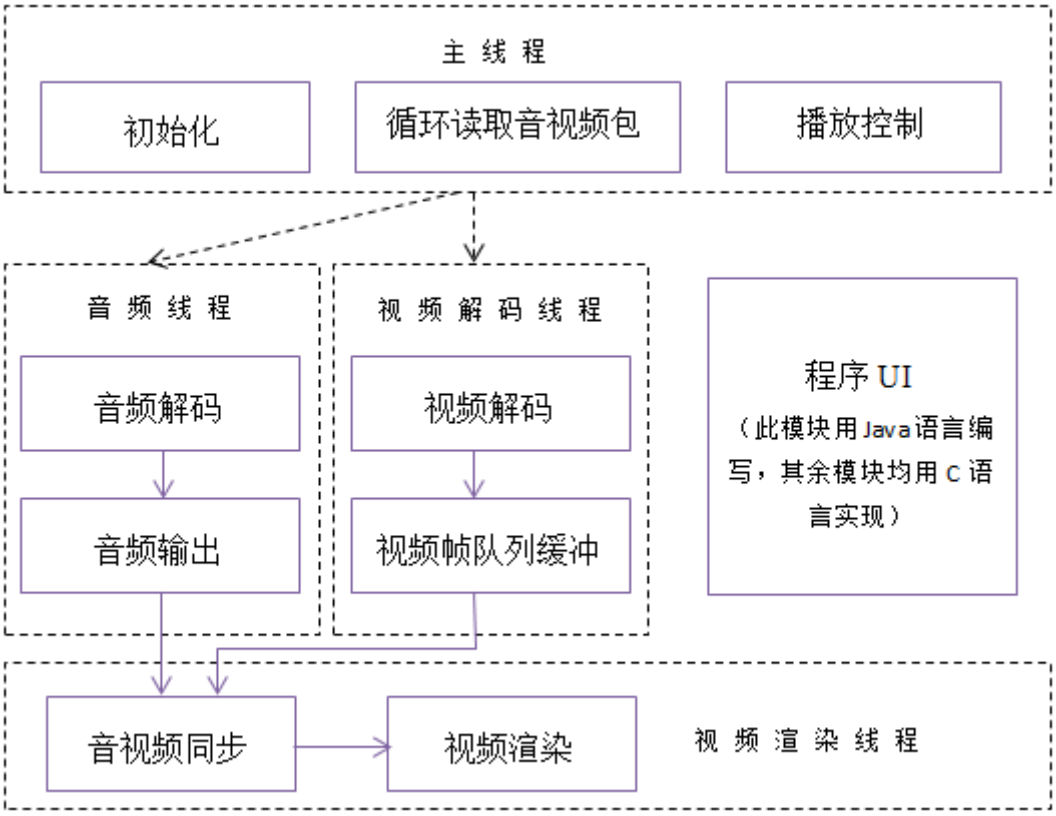


图 4.1：音视频播放程序模块图

UI 设计主要利用 Android 提供的界面机制，在 Android 开发文档中有大量资料，本文仅在 4.2.4 中简单介绍 SVPlayer 的实现结果，不再涉及细节。下面对音视频播放和同步方案稍作展开。

在对数据和音视频输出设备进行初始化后，程序会开始从数据源（本地文件或网络流）循环读取包，FFmpeg 中的 demuxer 会将音视频包解析开来。音视频数据包被分别存入各自的 packet 队列。音频解码线程从音频包队列中取包进行解码，解出音频 sample 后直接输出。因为音频是有采样率的，设备按此频率播放声音，就是实际速度，故音频其实可以不考虑时间戳。而对视频就要考虑时间戳和同步问题。为此，视频部分分为两个线程：一个负责解码，得到视频帧后存放在 frame 队列；另一个负责视频渲染，从 frame 队列中取出一帧后，在适当的时间输出。虽然视频也有帧率，但按照帧率播放往往是不可靠的，所以一般视频文件中每帧都有一个时间戳，称为 PTS(Presentation Time Stamp)，播放时按此显示每一帧。但这个值有时会不准确，而且可能渐渐与音频播放脱

节。所以需要进行同步处理。本程序采取将视频同步到音频时钟的方案。

在音频输出时，一定的数据量对应一定的音频播放时间。因为音频有固定采样率，这个对应是非常准确的（如采样率为 44.1KHz、样本大小用两字节 16bit 表示的双声道音频，每秒声音对应的数据量为 44100*16*2=1411.2Kbit）。我们统计输出的音频数据量，即可维护一个准确的音频时钟。在播放视频时，对于每一帧，将其 PTS 与当前音频时钟对比，若 PTS 晚于音频时钟，则增加该帧的输出延迟；反之，若 PTS 早于音频时钟，就立即输出，甚至跳过（丢掉）该帧。实际表明，这种方案能够满足要求，在人眼可分辨的范围内实现了较好的同步。

播放控制中的暂停主要是对线程的挂起和恢复（播放流媒体时还包括发送 RTSP 的 PAUSE 命令），快进和快退主要借助于 FFmpeg libavformat 库提供的 av_seek_frame 函数，不再详细说明。

4.2.3 对 SVC 视频层的设置

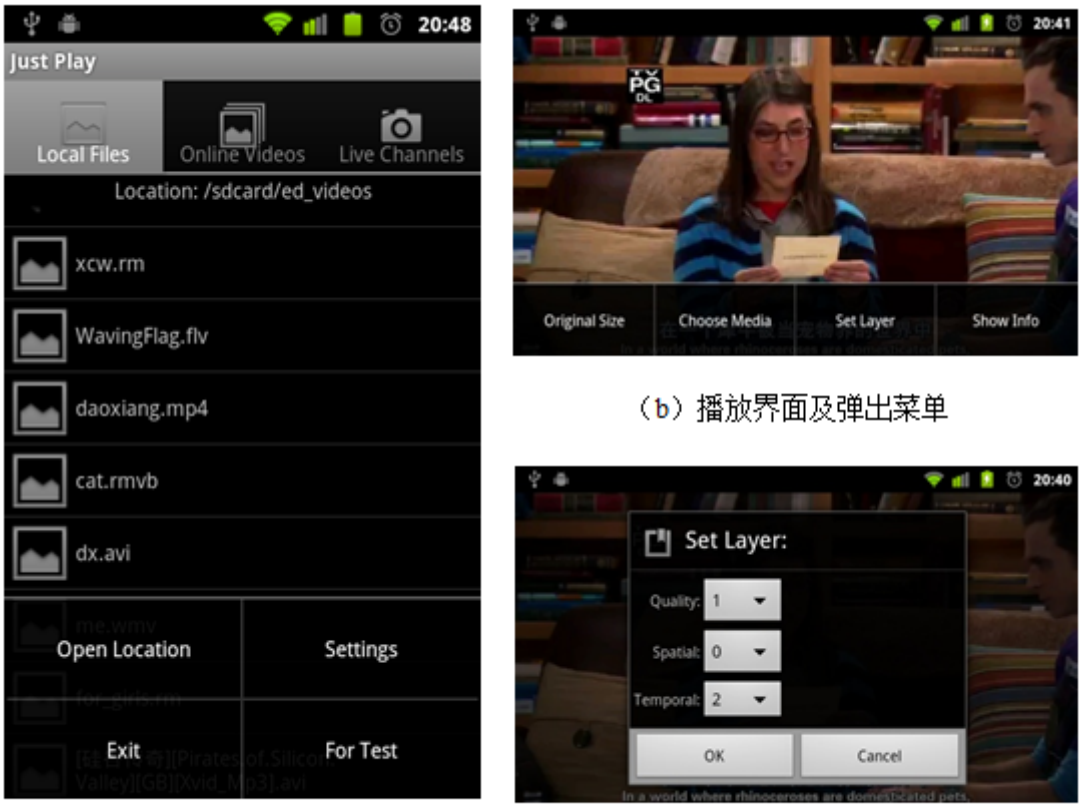
第三章设计流媒体服务器时指出允许客户端在播放 SVC 视频时对增强层数据进行设置。SVPlayer 对此进行了支持。此功能是通过 RTSP 协议完成的。在 FFmpeg 的 libavformat 库中有对 RTSP 协议的实现代码。我们在播放器的播放控制模块中添加 svcLayerSet 函数，该函数发送 RTSP 协议中的 PLAY 命令，命令含有协定的“x-Max-SVC-Quality-Level”参数（见 3.2.3），服务器接到该参数后，即按照其指定的层数进行发包。

4.2.4 软件功能与界面

SVPlayer 主要目的在播放 SVC 流媒体，但同时也支持普通媒体文件的播放（本地的或是网络的）。其主要功能包括：

- 能从服务器读取并显示点播视频列表和电视直播频道列表（对直播的服务器端支持尚在进行中），供用户选择
- 用户观看 SVC 网络视频时能根据网络情况自动调整帧率和画面质量，保证播放流畅（这主要是服务器端支持）
- 用户可以通过“Set Layer”菜单指定要接收的 SVC 码流中的增强层数目
- 可以浏览并播放本地音视频文件
- 可以通过“Open Location”菜单打开由 URL 指定的音视频文件
- 其他设置和查看工具，包括允许用户查看当前的下载码率、解码速度、播放帧率的统计值等

图 4.2 是 SVPlayer 部分界面截图。



（a）主界面（浏览媒体）及弹出菜单

（b）播放界面及弹出菜单

（c）播放中设置希望播放的层

图 4.2：SVPlayer 部分界面截图

4.3 针对 ARM 平台的优化

目前手机等嵌入式设备所采用的 CPU 90%以上都是精简指令集（RISC）的 ARM 架构（区别于复杂指令集的 x86 架构）。在完成通用的播放器软件 SVPlayer 以后，为了加快程序在嵌入式设备中的执行速度，我对其进行了针对 ARM 平台的优化。

对多媒体应用来说，除代码逻辑之外程序员可以进行的优化主要有两种：一是数据级优化，即对于数组拷贝等进行大量数据处理的地方，尽量提高其并行性；二是线程级优化，即将计算任务分配于不同线程，最大限度利用多核 CPU 的线程并行性。由于 SVPlayer 主要针对 Android 手机开发，而现在绝大多数手

机都采用单核 CPU，所以目前来说线程级优化的意义并不大。所以本文进行的优化主要是数据级的。

在视频解码时，有大量的数据拷贝、赋值操作，这些操作可以通过一种称为 SIMD（Single Instruction Multiple Data， 单指令多数据操作）的技术进行加速。所谓单指令多数据，即 CPU 在执行一条指令时，同时操作多个数据单元。它是借助于 CPU 中的加长型寄存器实现的，需要特定的处理器支持。不同处理器厂商可能推出自己的 SIMD 指令集。如 Intel 的 MMX（Multi Media eXtension，多媒体扩展）指令集，ARM 公司推出的向量处理指令集 NEON。本文所做的优化即 NEON 优化。

NEON 是 ARM 公司的 Cortex-A 系列处理器的扩展功能，适合于多媒体处理中的向量运算[10]。其基本思想是把多个数据作为一个向量装载到 128bit 的长寄存器中，然后一条指令集可对其进行加法、乘法等运算，大大提高了数据运算的并行性。图 4.3 说明了 NEON 中的向量寄存器及可能的数据组织。

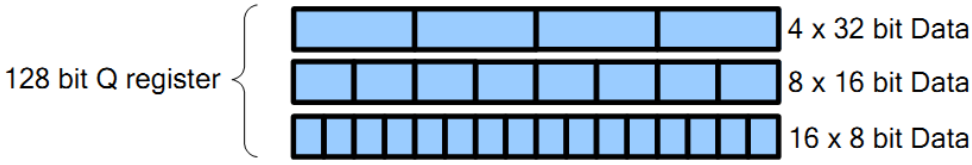
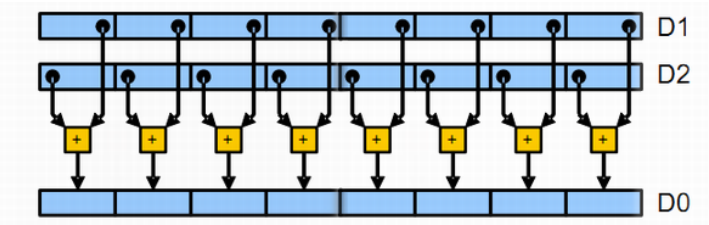


图 4.3：NEON 中的向量寄存器

在进行数组处理时，这种向量计算的优势十分明显，可以显著减少循环次数。在 SVC 解码核心中，有许多数组操作可以通过 NEON 优化来提速。表 4.1（在下页）对比了 add_coeffs 函数中的主要代码优化前后的版本及其对程序速度的影响。

add_coeffs 函数逻辑十分简单，其主要计算量是把两个长度为 48*8 的整形数组相加。用常规方法就需要循环 48*8 次；而采用 NEON 向量运算，只需循环 48 次：每次将连续 8 个 16bit 整数转载进 CPU 的 128bit 寄存器，然后用一条指令即可完成 8 个整数的加法运算（参见图 4.4）。这样的速度几乎是常规方法的 8 倍。经测量，频繁调用 add_coeffs 的 svc_decode_slice 函数优化后执行时间从 10ms 降到了 4ms。加速效果还是比较明显的。



（Di 表示 128bit 的长寄存器；可以同时计算 8 个 16bit 整数）

图 4.4：NEON 向量加法图示

表 4.1：NEON 优化前后代码及程序速度对比

优化前版本	NEON 优化后版本

int16_t *a=s->coeffs[mb_y*s->mb_width + mb_x];
int16_t *b=h->mb;
for(i = 0; i < 48 * 8; i ++)
{
a[i] += b[i];
}
	int16_t *a=s->coeffs[mb_y*s->mb_width + mb_x];
	int16_t *b=h->mb;
	for(i = 0; i < 48 ; i++)
	{
	int16x8_t a8 = vld1q_s16 (a);
	int16x8_t b8 = vld1q_s16 (b);
	a8 = vaddq_s16 (a8, b8);
	vst1q_s16(a, a8);
	a += 8;
	b += 8;
	}

频繁调用 add_coeffs 的 svc_decode_slice 函数执行时间： 约 10ms	频繁调用 add_coeffs 的 svc_decode_slice 函数执行时间： 约 4ms

综上，通过 ARM 平台上的 NEON 技术进行优化，能显著提高代码速度。在对整个 SVPlayer 进行优化后能使其在计算能力有限的嵌入式设备上发挥更大的作用。目前我们设计的 SVC 播放终端是基于 Android 的，而 Android 最初就是针对 ARM 平台的。因此，针对 ARM 的优化有很强的实用性。目前的 NEON 优化采用的还是 gcc 针对 ARM NEON 的内置 C 函数（经过 gcc 编译形成对应的 NEON 指令），为进一步提速，可以直接使用 NEON 汇编指令进行更细致的优化。

第五章 系统测试和结果分析

5.1 系统功能测试

在完成 SVC 流媒体的服务器和客户端之后，我们实际测试了整个系统。在服务器端，我们放上了不同分辨率和不同码率的 SVC 编码视频。测试视频源汇总于表 5.1。

表 5.1：用于测试的 SVC 视频源

视频文件	分辨率（w × h）	最低码率（kbit/s）	最高码率（kbit/s）
TheBigBangTheoryS04E01.mp4	480×256	80	320
TheBigBangTheoryS04E02.mp4	320×176	80	320
TheBigBangTheoryS04E03.mp4	800×448	80	320
TheBigBangTheoryS04E04.mp4	800×448	120	480

客户端使用的是运行 Android2.3 系统的三星 i9023 手机，其处理器为 ARM Cortex-A8 系列中主频 1GHz 的 Hummingbird CPU。网络环境包括 Wi-Fi 和联通 3G 移动网络。

对不同大小和不同分辨率的视频分别进行测试，发现均能正常播放。对于分辨率较小的 TheBigBangTheoryS04E01.mp4 和 TheBigBangTheoryS04E02.mp4，可以通过调节选择观看基本层和所有增强层；而对于分辨率为 800×448（即 WVGA 或标清水平的）的视频，基本层可以流畅观看，但较高码率的增强层却因终端运算能力的限制而影响观看（详情在后面 5.2 中介绍）。这也正说明了进行优化的意义。

图 5.1 比较了只有基本层和含有质量增强层这两种设置下的播放情况。这是视频文件 TheBigBangTheoryS04E03.mp4 在同一场景的截图。不同的是，图 5.1(a)设置为仅播放基本层，即“tid/sid/qid”参数组取值为“0/0/0”；而图 5.1(b)设置为加入一个质量增强层，即“tid/sid/qid”参数组取值为“0/0/1”。从统计并显示的数据可以看到，两种情况下的码率分别为 114 kbit/s 和 259 kbit/s。二

者的视频质量也有可见的差异，这在放大的部分看来更加明显。图 5.1(b)中的图像要比图 5.1(a)中的看起来清晰，尤其放大图中女人的脸和沙发的纹理。

由此，该 SVC 流媒体系统中视频的可伸缩特性示例性地体现了出来。另外，对于时间伸缩层的设置可以由播放帧率的变化得到体现，实际实验中可观察到帧率从 12 帧每秒到 25 帧每秒的变化。由于空间可伸缩用于显示屏幕不同的场景，在固定大小的手机上实验意义相对不大，故不做讨论。

（a）只有基本层时的播放效果

（b）含有质量增强层时的播放效果

图 5.1：只有基本层和含有质量增强层的视频播放情况对比

对程序其他方面的功能也进行了不同程度的测试。最终结论是 SVPlayer 和所搭建的流媒体服务器组成了一个可用的 SVC 视频点播系统，达到了设计目的和应用需求。

5.2 优化前后解码速度测试和对比

上面提到，所用的四个测试视频中，大小为 800×448 的两个在设置较高增

强层时会因设备解码速度不够而不能流畅观看。这也正说明了对程序进行优化工作的重要性。

在 4.3 中论述了针对 ARM 平台的 NEON 优化，这里我们对优化前后的解码速度进行测试和对比。测试参数如表 5.2 所示。

表 5.2：解码速度测试的参数

终端处理器	ARM　Cortex-A8 系列单核 CPU，主频 1GHZ
操作系统	Android 2.3.4
测试视频	TheBigBangTheoryS04E04.mp4 分辨率大小：800×448 测试的 SVC 层：基本层(0/0/0)和最高增强层(2/0/2) 相应的码率：120 kbit/s 和 480 kbit/s 相应的帧率：12 FPS 和 24 FPS

实验中分别使用优化前和优化后的程序播放同一视频的两个 SVC 码流：第一个码率为 120kbit/s，表示帧率为 12FPS 的基本层视频，记为码流 I；第二个码率为 480 kbit/s，表示帧率为 24FPS 的增强层视频，记为码流 II。统计平均解码速度的结果如表 5.3 所示。

表 5.3：优化前后解码速度对比

	码流 I	码流 II
优化前程序	13 帧每秒 （刚刚能不影响播放）	10 帧每秒 （远远达不到播放要求）
优化后程序	29 帧每秒 （完全不会影响播放）	18 帧每秒 （离播放要求有些差距）

注：①码流 I 的码率为 120kbit/s，正常播放帧率为 12FPS。　②码流 II 的码率为 480kbit/s，正常播放帧率为 24FPS。

图 5.2 为优化前后的程序播放码流 I 时同一场景的截图，其中显示的数据为程序输出的统计信息（3 秒更新一次），可以看出解码帧率（“Decode FPS”）的差异。

（a）优化前

（b）优化后

图 5.2：程序优化前后播放 12FPS 视频的对比

从上述结果可以看出，优化前后的程序都能顺利播放基本层，而对最高的增强层都无法流畅播放。虽然如此，二者的解码速度差别还是很大的，优化后比优化前速度增加了一倍多。而测试选用的是 SVC 码流中最低码率的基本层和最高码率的增强层；容易推测，如果是处于两个极端中间的码率，那么优化前无法流畅播放时优化后的仍然能够胜任。

总之，针对 ARM 平台的优化使程序有了很大提速。但选用最高增强层的例子也想指出，嵌入式设备的 CPU 计算能力毕竟有限，对大的视频无法实时解码。在这种情况下一方面对程序的优化非常重要，另一方面，设计硬件解码芯片才是根本解决之道——这也将是本文 SVC 播放终端的一个改进方向。

第六章 总结与展望

本文主要研究内容为可伸缩视频编码在流媒体系统中的应用。可伸缩编码SVC 作为最新视频编码标准 H.264/AVC 的扩展，一经推出就被认为有很大的应用前景，而目前广泛普及的流媒体系统中还尚未加入对 SVC 的支持。本文工作的创新之处就是弥补了这一空白。在分析 SVC 技术和现有流媒体现状的基础上，本文设计并实现了支持 SVC 的流媒体服务器和客户端。该流媒体服务器是基于开源的 Darwin 流媒体服务器，在其中加入了对 SVC NAL 单元进行过滤的机制，使之能够自适应地发送不同的视频增强层数据，实现可伸缩；另外还支持用户对发送的层进行设置。支持 SVC 播放的客户端则是 Android 手机上的SVPlayer，它不仅可以与 SVC 服务器配合构成一个可用的 SVC 流媒体系统、播放可伸缩性的在线视频，而且还同时具备播放传统媒体文件的功能。此外，本文还讨论了对 ARM 平台上的 SVC 解码和播放器的优化。这对 SVC 在以 ARM 为主的嵌入式领域中的应用也是很有意义的。

可以预见，SVC 将在未来的视频应用中占据越来越重要的地位。但总体来看，实际生活中的 SVC 产品还并不很多，这一方面是因为 SVC 标准出现不久，需要一个市场接受期；另一方面也可能是对其理论和应用的研究还不够充分。本文虽然设计实现了基于 SVC 的流媒体系统，但离产品化还有一定的距离。另外，本文仅讨论了手机上的 SVC 播放器，没有涉及其它终端上的工作进展。进一步的努力方向包括以下几个方面。

一是扩展支持 SVC 的终端类型。除手机之外，电视机顶盒、PC、平板电脑等都可以作为 SVC 视频播放的终端。本文写作时，支持 SVC 的电视机顶盒正在开发中。

二是设计集成芯片，实现 SVC 硬件解码。由前文可见，虽然经过了优化，但手机上的软件解码能力仍然有限。嵌入式系统中的多媒体应用大多是有硬件支持，SVC 也将不例外。本文作者正积极探索与硬件厂商合作开发 SVC 解码芯片，用于机顶盒等设备。

三是进一步完善和改进目前的 SVC 流媒体系统。目前采用的是 Client/Server 架构，可以加入 P2P 支持等。

最后指出，SVC 与流媒体结合的系统不仅限于点播、直播等在线视频观看，

而是在视频会议，网络视频监控，远程教育等诸多领域都有广阔的应用空间。这些都可以作为将来进一步探索的方向。而 SVC 相关的其他方面的理论研究和应用开发也是值得关注和重视的领域。

插图索引

表格索引

参考文献

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard”. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.13, pp.560-576, July. 2003.

[2] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the scalable video coding extension of the H.264/AVC standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, pp. 1103–1120, September 2007.

[3] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC 1, Version 8 (including SVC extension), July 2007.

[4] M. Wien, H. Schwarz, and T. Oelbaum, “Performance analysis of SVC”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1194–1203, September 2007.

[5] A. Segall and G. J. Sullivan, “Spatial Scalability Within the H.264/AVC Scalable Video Coding Extension,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1121–1135, Sep. 2007.

[6] Y.-K. Wang, M. M. Hannuksela, S. Pateux, A. Eleftheriadis, and S.Wenger, “System and transport interface of SVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1149–1163, Sep. 2007.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” IETF STD 0064, RFC3550, Jul. 2003.

[8] S. Wenger, M. M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer, “RTP payload format for H.264 video,” IETF RFC 3984,Feb. 2005.

[9] S. Wenger, Y.-K. Wang, and T. Schierl, “Transport and signaling of SVC in IP networks,” *IEEE Trans. Circuits Syst. Video Technol*., vol.17, no. 9, pp. 1164–1173, Sep. 2007.

[10] *ARM Ltd*, “NEON Support in the ARM Compiler” [Online], Available: *White Paper PDF Holding Page* http://www.arm.com/11848.php?language=zh .

[11] 黄拔峰，钟明，杨传钧，张家钰．Darwin Streaming Server 的研究与应用[J]．计算机工程：多媒体技术与应用，2004，30（19），134-135

致 谢

在本科学位论文完成之际，我想对如下团体和个人表示感谢。首先感谢清华大学和自动化系在大学本科四年中对我的培养，这使我在知识水平和为人素养上都有很大提高。其次，感谢我本科的班主任和论文指导教师陆文凯，在他的关心和督促下我顺利完成了本科学业和此篇论文。另外，在本文工作的进展过程中，北大计算机科学与技术研究所的郭宗明和孙俊两位老师，以及其他一些师长和同学都给予了热心的指导和帮助，在此一并表示感谢。

声 明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____　　日　期：_____

附录 A 外文资料的书面翻译

H.264/AVC 视频编码标准概述（节选）

摘要：H.264/AVC 是 ITU-T 视频编码专家组（VCEG）和 ISO/IEC 运动图像专家组（MPEG）提出的最新视频编码标准。该标准的主要目标在于增强压缩性能，并且提供一个“网络友好”的视频表示形式，以满足“传统”（如视频通话）和“非传统”（存储、广播、流媒体）应用需求。相比之前的标准，H.264/AVC 在率失真效率方面有了显著的提高。本文对 H.264/AVC 的技术特征进行了概论，描述了该标准的档次和应用，并简述了这一标准的形成历史。

关键词：AVC，H.263，H.264，JVT，MPEG-3，MPEG-4，标准，视频

第一章 引言

H.264/AVC 是视频编码的最新国际标准。本文发表之时，它应该已被国际电信联盟（ITU-T）批准为 H.264 建议书，并被国际标准化组织/国际电气委员会（ISO/IEC）批准为国际标准 14496-10（MPEG-4 part10）高级视频编码（AVC）。

MPEG-2 视频编码标准（又被称为 ITU-T H.262）于十年前在 MPEG-1 的基础上扩展而来，增加了对隔行扫描视频编码的支持，使数字电视系统得以在世界范围内推广。它广泛用于标清和高清电视信号通过卫星、电缆、地面发射塔的传输，以及高质量标清视频在 DVD 上的存储。

然而，高清晰电视的服务种类和流行度的增加，对编码效率提出了更高的要求。此外，其他一些传输介质，如线缆调制解调、xDSL、UMTS，能提供的数据传输速率要比广播信道低很多。增强的编码效率既可以允许更多的视频传输信道，又可以在已有传输容量上得到更高质量的视频。

电信应用领域的视频编码随着 ITU-T H.261、H.262（MPEG-2）、H.263（以及随后被称为 H.263+ 和 H.263++的 H.263 增强版）等标准的发展而不断进步，而且除 ISDN、T1/E1 之外，又逐渐支持 PSTN、移动无线网络、局域网/因特网。在这些进步中，一方面尽力最大化编码效率，另一方面努力处理网络类型的多样以及它们对丢包/错误鲁棒性的需求。

最近 MPEG-4 视觉标准（MPEG-4 part 2）已经开始在原来编码标准的领

域获得广泛应用。它提供了对视频形状编码的支持，也在致力于不断扩宽数字视频的使用环境。

早在 1998 年，ITU-T SG16 Q.6 的*视频编码专家组*（VCEG）发起了一个被称为 H.26L 的项目，旨在加倍既有视频编码标准的编码效率（这意味着对特定的清晰度码率将减小一半）。新标准的初稿于 1999 年 10 月发布。2001 年 12 月，VCEG 和*运动图像专家组*（MPEG）ISO/IEC JTC 1/SC 29/WG 11 形成了一个*联合视频小组*（JVT），最终使新的视频编码标准 H.264/AVC 于 2003 年 3 月正式通过。

这一标准的范围如图 A-1 所示。该图显示了典型的视频编解码链（不包括视频信号的传输和存储）。和之前所有的 ITU-T 及 ISO/IEC 视频编码标准一样，只有核心解码器被标准化，即：对码流和语法作了严格限制，对语法元素的解码过程作了定义，从而使得只要给定一个符合标准的码流，所有符合标准的解码器都会产生同样的输出。这种对标准范围的限制能够为根据特定应用进行优化实现（例如在压缩质量、实现开销、产品上市时间中寻求平衡）提供最大的自由。然而，该标准对端到端的重现质量没有任何保证，因为它甚至允许暴力编码技术——只要符合标准即可。

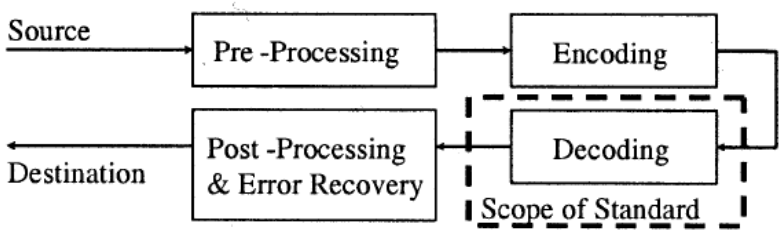


图 A-1：视频编码标准的范围

这篇文章结构如下。第二部分对 H.264/AVC 的应用做整体概述，并指出能改善这些应用的设计的技术特点。第三部分论述了网络抽象层（NAL）以及 H.264/AVC 编码后数据的整体结构。视频编码层（VCL）在第四部分叙述。第五部分解释了 H.264/AVC 所支持的档次及该标准的一些潜在应用领域。

第二章 应用和设计亮点

新的标准是为一些应用领域的技术解决方案而设计。这些领域至少包括：

- 通过电缆、卫星、线缆调制解调、DSL、地面发射站等途径的广播；

- 在光学介质、磁介质，及 DVD 等上的交互式或串行存储；
- 基于 ISDN、以太网、LAN、DSL、无线和移动网络、调制解调器等或是这些方式混合的的传统服务；
- 基于 ISDN、线缆调制解调、DSL、 LAN、无线网络等的视频点播和流媒体服务；
- 基于 ISDN、DSL、以太网、 LAN、无线和移动网络等的多媒体信息服务（MMS）；

此外，现有和未来的网络上可能会衍生新的应用。这就提出了一个如何应对应用和网络形式如此多样的问题。

为了满足这种对灵活性和可定制性的需求，H.264/AVC 的设计提出了两个概念：VCL 用于高效的表示视频内容，而 NAL 则封装视频的 VCL 表示并提供头部信息，使之适用于多样化的传输层次和存储介质（参见图 A-2）。

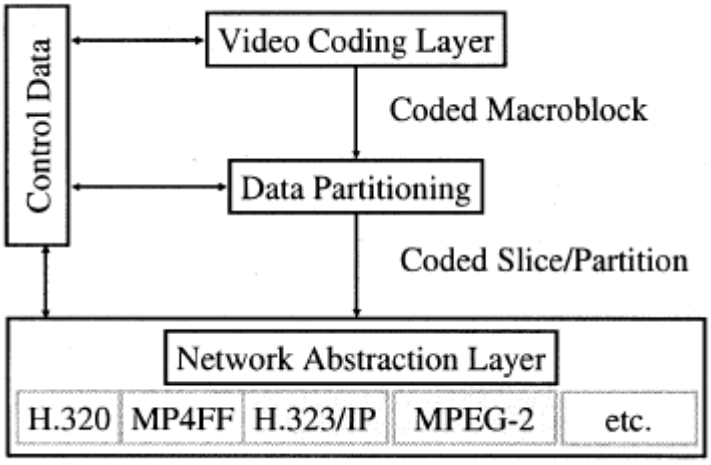


图 A-2：H.264/AVC 编码器结构

相比于之前的视频编码方法（比如在 MPEG-2 视频部分中的那些），一些新的设计亮点增强了编码效率，包括以下更好预测将要编码的图片的方法。

- **小尺寸的可变块大小运动校正：**这一标准在选择运动校正块的尺寸和形状方面比之前的标准具有更大的灵活性，其最小的亮度运动校正块尺寸可以是 4×4。
- **四分之一像素精度的运动校正：**大多数之前的标准都采用顶多二分

之一像素的运动向量精度。新的设计对此有所改善，增加了四分之一像素的运动向量精度。这最早出现于 MPEG-4 视觉（第 2 部分）标准的一个高级档次中。相比于旧标准，插值运算的复杂度进一步减小。

- **跨越图像边界的运动向量：**在 MPEG-2 及其之前的标准中，运动向量只能指向处于已解码的参考图像之内的区域。图像边缘外插技术作为可选特征第一次出现在 H.263 中，它也被包含在了 H.264/AVC 中。
- **多参考帧的运动校正：** 在 MPEG-2 及其之前的标准中，预测编码帧（称为“P”帧）只使用前一帧的图像来预测当前图像的值。新的设计扩展了出现在 H.263++中的参考图像选择技术，允许编码器从已解码的众多图像中选择用于运动校正的参考，以此提高编码效率。同样，在双向预测帧（称为“B”帧）中也引入了这一扩展的参考选择能力（而在 MPEG-2 中只能使用特定的两帧，即显示顺序中的前一个 I 或 P 帧和显示顺序中的后一个 I 或 P 帧）。
- **参考顺序和显示顺序的解耦合：**在之前的标准中，在用于运动校正的图像顺序和用于显示的图像顺序之间有这严格的依赖。而在 H.264/AVC 中，这一依赖几乎全被去掉，编码器可以非常自由的选择参考和显示顺序，只要不超过内存界限。
- **图像表示方法和参考能力的解耦合：**在之前的标准中，用一些方法编码的图像（如双向预测编码的图像）不能够作为序列中其他图像的预测参考。新的标准去除了这样的限制，为解码器提供了更大的灵活性，而且在大多数情况下，使得编码器可以选取与被预测图像更接近的参考。
- **带权重预测：**H.264/AVC 的一个创新之处是允许编码器对运动校正的预测信号进行加权或是偏置一定量。这对含有阴影的场景可以显著提高编码效率，而且也可以灵活用于其它目的。
- **改进的“跳过”或“直接”运动推断：**在之前的标准中，预测编码图像的“跳过”区域不能在场景中运动。这对那些含有全局运动的视频会产生负面影响。于是新的 H.264/AVC 设计转而在“跳过”区域中进行运动推断。对于双向预测编码的区域（称为 B 片），H.264/AVC 还包含一种增强的运动推断方法，称为“直接”运动校正。这是对 H.263+和 MPEG4 Visual 中“直接”预测设计的改进。
- **帧内编码的方向性空间预测：**在帧内编码（即没有参考其他帧的）

的图像区，采用了一种新的技术：对本帧图像已解码部分的边缘进行外推。这可以提高预测信号的质量，也使得能够从那些不是帧内编码的区域进行预测（这在采用 H.263+和 MPEG4 Visual 中的变换域预测方法时是不可以的）。

- **循环内去块滤波：**基于块的视频编码会产生“块效应”。这既可能来自解码过程的预测阶段也可能来自残差编码阶段。采用自适应的去块滤波是常见的提高解出的视频质量的方法。如果设计得好，去块滤波既可以提高客观质量，又可以提高主观质量。H.264/AVC 对 H.263+某个可选特性中的一个概念进行了改进，把去块滤波放在了运动校正的预测循环中。这样去块滤波对视频质量的改进就可以用于帧间预测，从而提高预测其他帧的能力。

除了预测方法的改进，H.264 在其他部分也有其他提高编码效率的设计。包括以下这些。

- **小尺寸块变换：**之前所有主要的视频编码标准都采用 8x8 的变换块，而新的 H.264/AVC 却主要基于 4x4 的块进行变换。这使得编码器能够以更加局部自适应的方式来表示信号，从而减少常见的“振铃效应”。（另外，更小的块尺寸从以下两点来看也是合理的：一是增加了用上面提到的方法更好地预测视频内容的能力，二是能够提供边界对应于那些最小的预测区域的变换区）
- **分层的块变换：**虽然大多数情况下使用 4x4 的变换块都是有利的，但仍有一些信号相关性很强，可以采用基函数更长的表示方法。H.264/AVC 提供了两种方式满足这种需要：1）采用分层的变换，可以让那些低频的色度信号的有效变换块大小扩展到 8x8；2）允许编码器为帧内编码选择一种特殊的类型，从而使低频的亮度信号的变换块大小扩展到 16x16，方式与 1）中用于色度信号的类似。
- **短字长的变换：**所有之前的标准都导致编解码器在进行变换计算时使用复杂的 32 位数据处理，而 H.264/AVC 则只需要 16 位的算术运算。
- **无失真的逆变换：**在之前的视频编码标准中，获取与原数据完全一致的逆变换结果是不现实的，因此都只要求视频的变换表示不超过某个误差限即可。这样一来，每个解码器都可能解出略微不同的结果视频，即在解码器和编码器之间产生“漂移”（drift），降低了有效视频质量。基于 H.263++的一个可选特性，H.264/AVC 是第一个实现所有解码器

解出的视频完全一致的标准。

- **算术熵编码：**H.264/AVC 引用了一种被称为算术编码的更高级熵编码方法。算术编码原来是 H.263 的一个可选特性，在 H.264/AVC 中这种技术被更有效地利用，产生了一个非常强大的熵编码方法，即上下文自适应二进制算术编码（CABAC）。
- **上下文自适应的熵编码：**H.264/AVC 中的两种熵编码方法，CAVLC 和 CABAC 都采用了基于上下文的适应性来提高性能。

H.264/AVC 还引入了许多新的设计概念，从而增强了对数据错误和丢失的鲁棒性以及用于不同网络环境的灵活性。包括以下这些亮点。

- **参数集结构：**参数集的设计使得头信息能够更加鲁棒和有效地传输。考虑到在之前的标准中，重要信息（如序列和图像的头信息）内很少几个关键比特的丢失就会对解码过程产生严重负面影响，H.264/AVC 设计为将关键信息分离出来，以更灵活特殊的方式单独**处理**。
- **NAL 单元句法结构：**H.264/AVC 中的每一个句法结构都被放在称为“NAL 单元”的逻辑数据包中。与之前的标准强制使用指定的比特流接口不同，NAL 单元句法结构允许对携带视频内容的方法有更强的定制性，使得对每个特定网络都能有适合的方式。
- **可变的片大小：**MPEG-2 中网状的片结构增加了头数据的数量且降低了预测的效果，从而降低了编码效率；而在 H.264/AVC 中，片的大小是高度可变的，正如在更早的 MPEG-1 中那样。
- **可变的宏块排序（FMO）：**一种把整帧图像分割成“片组”的方法被提出，使得每个片都成为某个“片组”的一个可独立解码的子集。如果使用恰当，可变宏块排序可以通过管理每个片中编码区域的关系来显著增强对数据丢失的鲁棒性。（FMO 也可以被用于其他目的）
- **任意的片排序（ASO）：**由于图像中的每个片都可以（近似）独立于其他片而解码，H.264/AVC 的设计能够以任意顺序发送和接收图像中的片。这种功能最早在 H.263+的可选特性中出现，可以改善实时应用中的端到端时延，尤其是在不保序传输的网络中（例如 IP 网络）。
- **冗余图像：**为了增强对数据丢失的鲁棒性，H.264/AVC 的设计方案允许解码器发送图像中区域的冗余表示。这使得在该图像区域的主表示在数据传输中丢失的情况下，解码器还能得到一个（一般来说相对降级的）可用表示。

- **数据划分：**一些用于表示每个区域的编码信息（例如运动向量及其他一些预测信息）比其他用于表示视频内容的信息要更重要或更有价值。因此，H.264/AVC 允许每个片的句法根据句法元素的分类划分成至多三个不同的部分来传输。这部分设计是对 MPEG-4 和 H.263++某个可选内容的继承和改进。新标准中，设计被简化为让每个包含同一句法部分的单一句法都受控于特定的句法元素类别。
- **SP/SI 同步/切换帧：**H.264/AVC 设计方案包含了一个由图片类型组成的新特性，它允许一些解码器的解码过程同步到一个由其他解码器产生的视频流上去却避免使所有解码器遭受由发送 I 帧导致的性能损失。这就使一个解码器能够在视频内容的数据速率各异的不同表示之间切换，而且能够从数据丢失和错误中恢复，甚至还使得一些特殊模式成为可能，如快进、快退等。

以上提到的关键特性还会在第三、四两章中有更详细的描述。

第三章 NAL

NAL 的设计目的是提供“网络友好性”，以使 VCL 能在不同的系统中方便而高效地使用。

NAL 提供了把 H.264/AVC 的 VCL 数据映射到以下传输层的能力：

- RTP/IP，用于各种实时有线和无线因特网服务的；
- 文件格式，比如 ISO MP4，用于储存和多媒体服务（MMS）；
- H.32X，用于各种有线和无线通讯服务；
- MPEG-2 系统，用于广播服务；等等。

如何具体定制视频内容使之满足每个特定应用的需要不再 H.264/AVC 的标准化的范围之内；但是，NAL 的设计却涉及了这些方面的内容。网络抽象层（NAL）的一些基本概念包括：NAL 单元，参数集，以及访问单元。以下仅简单介绍这些概念；更详细的描述以及错误恢复可以参考文献[6]和[7]。

A．NAL 单元

编码后的视频数据被组织成 NAL 单元。每个 NAL 单元是一个包含整数字节的有意义的数据包。NAL 单元的第一字节成为头字节，它包含的信息用来指明该 NAL 单元中数据的种类；余下的字节则含有头字节所指定类型的载荷数据。

必要的时候，NAL 单元中的载荷数据中穿插有“*竞争防止字节*”。插入的这些字节是一些特定的值，目的是防止意外生成与“*起始前缀码*”相同的数据模式。

NAL 单元的结构定义采取通用的格式，使其既能用于面向数据包的传输系统，又能用于面向比特流的系统。编码器生成的一系列 NAL 单元被称为一个 NAL 单元流。

B．NAL 单元在字节流格式中的运用

一些系统（如 H.320 和 MPEG-2/H.222.0 系统）需要将整个或部分 NAL 单元流以字节或比特的有序流的形式传送。此时，NAL 单元流中的 NAL 单元的边界就要能够与编码数据本身中的模式区分开来。

对这样的系统，H.264/AVC 规范定义了一个字节流的格式。在符合规范的字节流中，每个 NAL 单元前面都加上了称为“起始前缀码”的三个字节。NAL 单元的边界就可以通过搜寻特定的起始字节码找到。而竞争防止字节的应用确保了起始前缀码能够唯一标识出新 NAL 单元的开始（即它们不会出现在一个 NAL 单元的内部）。

此外还有少量的附加数据（每帧视频图像一个字节）用于数据对齐。如果工作在一个对比特流不提供字节边界对齐功能的系统上的解码器，这些数据被用于从数据流中恢复必要的对齐。

如果需要，还可以向字节流格式中加入更多的数据，用来扩展发送的数据量，或是帮助加快字节对齐恢复的速度。

C．NAL 单元在数据包传输系统中的应用

在其他一些系统中（例如 IP/RTP 系统），编码后的数据由系统传输协议所分组的包来携带。这些分组包中 NAL 单元的边界就不需要前缀字节码也可以确定。这这些系统中，包含前缀字节码将是一种浪费，因此 NAL 单元将被装载于数据包而免去前缀字节码。

D．VCL 类型和非 VCL 类型的 NAL 单元

NAL 单元被划分为 VCL 和非 VCL 两类。VCL 的 NAL 单元含有表示视频帧中采样点的数据，而非 VCL 的 NAL 单元则包含其他所有相关的附加信息，如参数集（被大量 VCL 型 NAL 单元所有重要头数据）和补充增强信息（计时信息和其他一些补充数据，用于增强解码后视频的可用性，而非在解码视频帧

中的样本时所必需）。

E．参数集

参数集含有的一般是那些在解码大量 VCL 型 NAL 单元时很少变化的信息，其类型有两种：

- 序列参数集，用于被称为编码视频序列的一系列连续的编码视频图像。
- 图像参数集，用于编码视频序列中一个或多个单独图像的解码。

序列和图像参数集的机制使得不常变化的信息的传输与视频图像样点的编码表示的传输分离开来。每个 VCL 型 NAL 单元都含有一个标记，指向与它有关的图像参数集的内容；而每个图像参数集也含有一个标记，指向与它有关的序列参数集的内容。在这种方式下，少量信息（即标记）被用来参考较大量的信息（即参数集），避免了较大量信息在每个 VCL 型 NAL 单元中的重复传输。

序列和图像参数集可以被相当早地发送（相对于 VCL 型 NAL）。而且它们还可以被重复已增强对数据丢失的鲁棒性。在某些应用中，参数集和 VCL 型 NAL 单元在同样的信道中发送（称为“带内传输”）。而在另外一些应用中（参见图 A-3），“带外传输”将更有优势，即用比视频信道更可高的的传输机制来运送参数集。

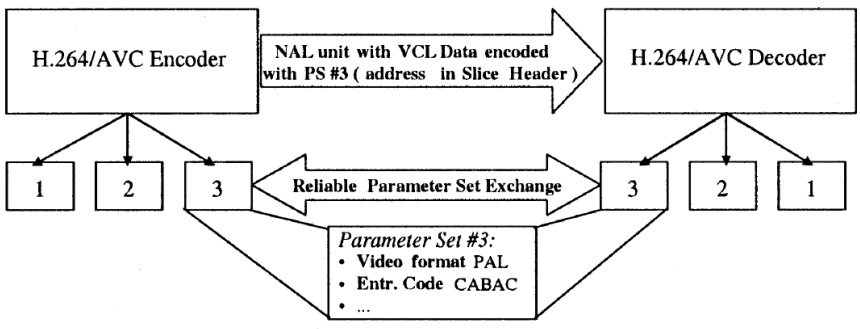


图 A-3：参数集以更可靠的信道进行“带外传输”

F．访问单元

特定形式的 NAL 单元的集合被称为一个访问单元（Access Unit）。每个访问单元能解码出一帧图像。访问单元的格式如图 A-4 所示。

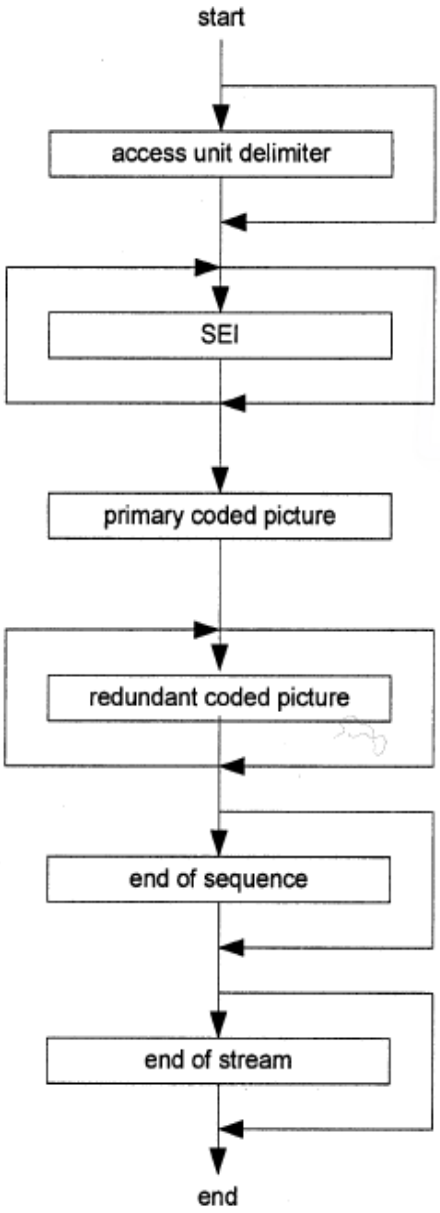


图 A-4：一个访问单元的结构

每个访问单元含有一组 VCL 型 NAL 单元，这些 NAL 单元共同组成一个*主编码图像*。访问单元还可能有一个称为*访问单元分隔符*的前缀来帮助定位它的起始。在主编码图像之前，还可能出现含有如图像计时之类数据的*补充增强信息*。

主编码图像由一组 VCL 型 NAL 单元组成，它们又由表示视频图像样本点的*片*或*片段数据划分*组成。

在主编码图像之后可能出现一些附加的 VCL 型 NAL 单元，它们含有同一视频图像区域的冗余表示。这被称为*冗余编码图像*，可以在主编码图像中的数据丢失或损坏的情况下被解码器做用作恢复。即使冗余编码图像出现，也并不要求解码器一定要解码它们。

最后，如果该图像是一个编码视频序列（可以独立解码，并且只使用一个序列参数集的一系列图像）的最后一帧，那么一个*序列结束* NAL 单元就会出现，表示该序列的结束。而如果该图像是整个 NAL 单元流的最后一帧，那么类似地，会有一个*流结束* NAL 单元出现。

G．编码视频序列

一个编码视频序列由 NAL 单元流中一系列连续的访问单元组成，而且只使用一个序列参数集。在已知必要的参数集信息（既可以“带内”又可以“带外”传送）的条件下，每个编码视频序列可以被独立于任何其他视频解码序列而解码。在编码视频序列的开始是一个*“立即解码刷新”*（*instantaneous decoding refresh，IDR*）访问单元。IDR 访问单元包含一个*帧内*图像——可以在不解码 NAL 单元流中任何之前图像的条件下解码的图像。IDR 访问单元的出现意味着该 NAL 单元流中接下来的图像都不再需要参考它所含的帧内图像之前的图像了。

一个 NAL 单元流可能由一个或多个编码视频序列组成。

（原文第三章结束。本翻译节选至此。——译者注）

书面翻译对应的原文索引

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard”. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.13, pp.560-576, July. 2003.