

清 华 大 学

综 合 论 文 训 练

题目：多视点（Multi-view Coding）
视频的并行实时解码

系 别：计算机科学与技术系

专 业：计算机科学与技术

姓 名：卿培

指导教师：孙立峰 副教授

2010 年 6 月 22 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

视频编解码的并行在近几年 CPU 向多核方向发展之后成为一个热门的研究领域。最近成为标准的多视点视频由于其数据量庞大、编解码运算复杂, 很难依靠单核的处理器达到实用的解码速度。因此, 多核并行解码多视点视频势在必行。

我们以现有的多视点视频解码器为基础, 通过对解码器函数级的优化, 包括函数逻辑、减少循环内部的计算量、使用汇编实现部分函数, 以及一个可以稳定运行的并行解码框架的实现, 最终使得多视点视频的解码可以在主流 PC 上实现两路标清的实时解码。

本文的主要贡献是:

- 首次实现了非商业化的解码器在 PC 上的两路多视点视频实时解码;
- 使用庞一等在 2009 年提出的多视点视频编解码并行调度框架^[1] 在解码器中实现了稳定的调度器。

关键词：多视点；并行；实时；解码

ABSTRACT

With the multi-core trend of processor design and production, research efforts on the parallelization of video coding have been strengthened. Multi-view coding (MVC), which has been standardized recently, demands massive space for storage and an enormous amount of computation to encode and decode and therefore is almost impossible to be decoded in realtime with a single processor core. A parallel decoder for multi-view coding is highly in demand.

On the basis of an available decoder, we apply multiple ways of optimization on function level, including rewriting the logic structure, decreasing the amount of computations inside loops and replacing some of the function body with assembler code. Meanwhile, a stable parallel framework for scheduling and decoding is implemented. All the optimizations add up to reach the performance guideline of realtime decoding of dual-view standard definition (SD) video on mainstream PC platforms.

The main contributions of this paper are

- to realize the first noncommercial decoder capable of decoding dual-view MVC video in real-time;
- to implement a stable version of scheduler and decoder with the framework^[1] put forward by Pang, et.al, in 2009.

Key words: Multi-view Coding(MVC); Parallel; Realtime; Decoding

目 录

第 1 章 引言	1
1.1 选题背景	1
1.2 已有研究	2
1.3 本文的任务	5
1.4 本文的结构	5
第 2 章 MVC 解码器原理与实现	7
2.1 参考软件 JMVC 介绍.....	7
2.2 自主开发的 MVC 解码器	9
2.3 小结	10
第 3 章 解码器性能优化方案	11
3.1 软硬件平台说明.....	11
3.2 解码器性能分析.....	12
3.2.1 VS2008 内置性能分析	12
3.2.1.1 性能分析步骤.....	12
3.2.1.2 性能分析结果.....	12
3.2.2 VTune 分析	13
3.2.2.1 性能分析步骤.....	13
3.2.2.2 性能分析结果.....	14
3.2.3 性能分析结果说明	14
3.3 优化方案	15
3.4 优化目标	15
3.5 小结	16

第 4 章 解码器单核性能优化	17
4.1 函数逻辑优化	17
4.2 循环优化	19
4.3 汇编优化	20
4.4 其它优化	21
4.5 小结	21
第 5 章 解码器并行优化	23
5.1 对解码器并行性的分析	23
5.2 并行框架介绍	26
5.3 问题与解决	26
5.4 小结	27
第 6 章 解码器性能优化实验结果	29
6.1 测试数据的生成	29
6.2 实验结果及说明	32
6.3 小结	35
第 7 章 3D 播放器设计与实现	39
7.1 实验平台	39
7.2 调研阶段	39
7.3 3D 播放器设计	41
7.3.1 静态 3D 图像的显示	41
7.3.2 立体视频（图像序列）的显示	42
7.4 播放器测试	44
7.5 小结	45
第 8 章 结论和展望	47
8.1 本文工作总结	47
8.2 存在的问题	47
8.3 未来工作	48
插图索引	49

表格索引	50
参考文献	52
致 谢	58
声 明	59
附录 A 外文资料的书面翻译	61
A.1 摘要	61
A.2 简介	61
A.3 多视点视频、深度和关联数据编码	62
A.3.1 传统立体视频编码	62
A.3.2 视频加深度数据的压缩	64
A.3.3 多视点视频编码	65
A.3.4 结论和未来研究方向	68
附录 B 实验原始数据	69
附录 C 实验中使用的部分代码和配置文件	76
附录 D 综合论文训练记录表	80

主要符号对照表

CABAC	基于上下文的自适应二进制算数编码 (context-based adaptive binary arithmetic coding): CABAC 的设计概念对于发生机率 > 0.5 的事件有效地编码, 改进了传统霍夫曼编码法需要大量的乘法运算的问题, 而在效能与压缩效率上取得相当大的改善空间。。
CAVLC	基于上下文的自适应变长编码 (context-based adaptive variable-length code): 适用于 DCT 转换后的整系数矩阵, 经过 zig-zag 顺序扫描之后, 在最高层的系数通常为 +1/-1; 又取得以 zig-zag 顺序扫描时, 连续出现的 0, 或非零系数的总数、最后一个非零系数前零的数目等参数, 作为查表时的坐标。CAVLC 针对不同的块大小设计了不同的查找表, 对各种不同的上下文, 使用不同的查找表进行编码, 有效缩短输出比特流长度。。
GOP	group of pictures: 一个 GOP 中所有帧的参有固定的参考结构; 下一个 GOP 中的帧与上一个 GOP 中的帧的参考结构一样。
MB	宏块 (macroblock): 一个 16×16 的亮度块采样和对应的两个色度块采样。
MVC	多视点视频编解码 (multi-view video coding)。
NAL unit	网络抽象层 (network abstract layer) 单元: 一个语法结构, 包含后续数据的类型指示和所包含的字节数, 数据以 RBSP 形式出现, 必要时其中还散布有防伪字节。

第 1 章 引言

1.1 选题背景

21 世纪的最初十年是计算机技术与互联网应用发展极为迅速的时期。在此期间，人们见证了 CPU 主频步入 GHz 时代，内存的容量和吞吐率按照摩尔定律稳定地翻番，硬盘容量进入了 TB 级，人们探索互联网的接入方式也从曾经的 56kbps 甚至更慢的 modem 换成了带宽以 Mbps 计的宽带接入。可以说，这些都为数字多媒体技术在计算机和互联网上的应用奠定了硬件基础。

我们从十年前看 VCD，五年前看 DVD，到现在拥有了 720p 到 1080p 的高清电影资源；从 8bit 的音频到 24bit、192kHz 采样的无损音乐格式，无不计算能力更强和存储容量更大的计算机来处理。信息化的大步跨越让多媒体技术在我们日常生活中扮演越来越重要的角色。各种新的应用也接踵而来。

3D 电视经过多年的发展，已经逐渐走向成熟。在 CES(Consumer Electronics Show) 2010 的展厅里，“3D”成了最大的赢家，备受关注。Panasonic 的 VT25 平板 3D 电视也获得了这届 CES 的“Best in Show Award”。无论是各大家电厂商展台上的 3D 电视，还是可以用在现有系统上的 NVIDIA 3D Vision 套装，都令消费者们感受到了 3D 时代即将来临。其实，3D 电视的起源并不比 2D 电视晚很长时间^[2]，但是由于其对计算、存储和传输的资源消耗比普通 2D 电视高出一倍甚至数倍，所以一直以来未能普及。

为了满足资源有限条件下的 3D 视频应用，各种编码技术陆续诞生，这些编码使用一定量的比特率来描述多视频信号，达到一定的压缩率，同时尽可能减小失真。在这个过程中，国际标准化组织应运而生。目前国际上有两个音视频编码标准化组织：

- 一是国际电信联盟标准化组 (ITU-T) 下属的视频编码专家组 (VCEG: Video Coding Expert Group)。制定了一系列视频通信协议和标准，包括 H.261、H.263、H.263+、H.264 等。主要应用在视频通信领域，如视频会议等。
- 二是国际标准化组织 (ISO) 和国际电工委员会 (IEC) 下属的运动图像

编码专家组 (MEPG: Motion Picture Expert Group)。自 1988 年成立以来,研究和开发了多种多媒体压缩标准,包括 MPEG-1、MPEG-2、MPEG-4、MPEG-7 等。主要应用在存储媒介 (DVD)、广播电视网、有流媒体传输等。

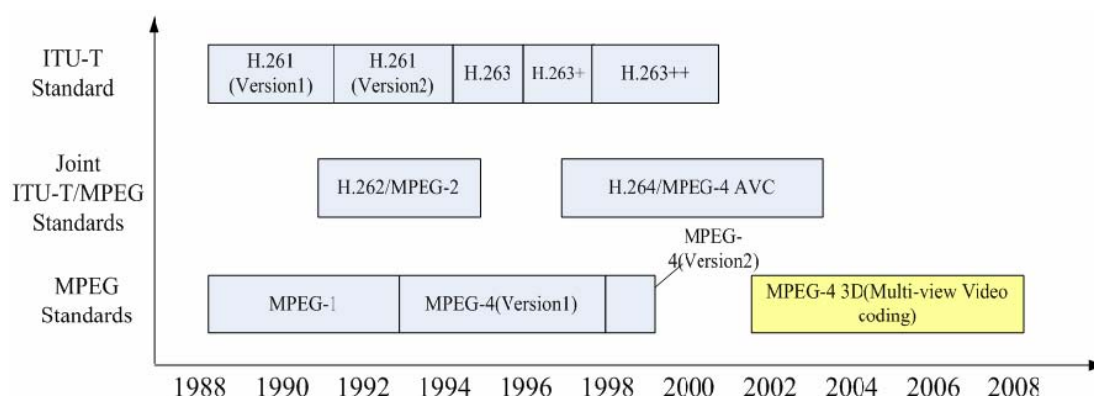


图 1.1 MPEG、VCEG、JVT 视频编解码标准的发展

在 2001 年 6 月,经过评估发现,H.26L 编码技术基本能够满足 MPEG 的标准需求,因此 MPEG 与 VCEG 的成员共同成立了一个新的工作组 JVT (Joint Video Team),来推动和管理 H.26L 的最后标准化开发,并在 2003 年形成了视频标准,同时被 MPEG 和 VCEG 采纳:MPEG 将其定为 MPEG-4 part 10,VCEG 将其定为 H.264。H.264/MPEG-4 part 10 标准称为高级视频编码 (AVC: Advanced Video Coding),扩展性较好。用于 3D 视频的多视点视频编解码 (MVC: Multi-view Video Coding) 就是其一个扩展。MPEG 与 VCEG 及 JVT 制定的主要视频标准如图 1.1 所示。

1.2 已有研究

在 Multi-view Video Coding 方向的研究在很久前就开始了。3D 视频的一个特点是:传输给用户的各个视角的视频描述的是同一个场景。各个视角的视频在统计意义上的相关性很大,这些相关性就可以用来做预测,如图 1.2 所示^①。综合利用时间序列以及视角间的图像预测结构,能够极大地提高编码效率^[3,4]。[5] 中提及了多视点图像编码的一些前沿研究。

① <http://mpeg.chiariglione.org/technologies/mpeg-4/mp04-mvc/image004.jpg>

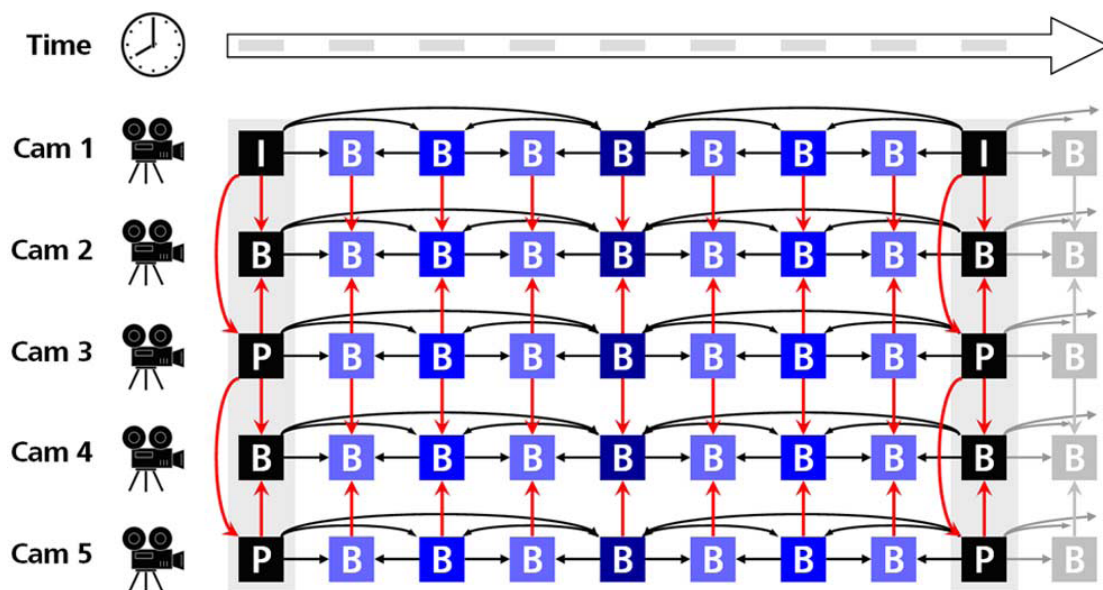


图 1.2 MVC 的时间和视角间预测结构

对于视角间和时间序列上的图像预测结构，有不少文献提出了不同的方法。其中，H.264/AVC 时间和视角间预测的基于 B 帧的算法^[4]在 MPEG 标准测试下性能表现最好^[6-9]。在实验中，各种指标都显示 MVC 远远超越了独立压缩每个视角视频的方法。不过，能够获得的性能提升对参数有一定依赖，如相机间距、帧率、内容复杂度（运动和纹理）等属性。对于一些数据集，信噪比峰值（PSNR）的增益在 0.5dB 以下，而最大的增益能达到 3dB。

这个方法在提高视频编码压缩率的同时，增加了编解码的复杂度，要求处理更大量的数据和进行更多的计算。对于 MVC 编解码的研究大多使用参考软件 JMVC 进行^①。参考软件的编写比较注重对应多视点视频编码的原理，对于学习 MVC 编解码很有帮助，但也因此损失了极大的性能。在实验所使用的机器上对 720×576 的两路视频进行编码，19 帧需要大约半个小时，这远远不够实际应用的需求。诺基亚研究中心（Nokia Research Center）针对其运行 Maemo 的平台开发了一款 MVC 实时解码器^②，针对单核心做了不少优化。由于其主要利用汇编优化，而处理器并不是 x86 兼容的，所以不易移植到 PC 上。

至此，我们认为有必要进行一个能够在 PC 上进行两路到更多路 MVC 视频的解码器，进行客户端的实时解码。这样的解码器是 3D 视频应用普及的一个

① 由 Heiko Schwarz、Tobias Hinz、Karsten Suehring 等人开发。

② <http://research.nokia.com/research/mobile3D>

重要前提。

由于 MVC 是 H.264/AVC 的一个扩展，所以我们首先想到 H.264 的并行编解码问题。这个方向近几年有一些文献：[10] 提出的算法使用一个处理器进行动量估计以外的运算，其余处理器进行运动向量的估计。[11] 做了 H.264 解码器的宏块并行分析，[12] 给出了 H.264 解码器宏块并行算法。而更进一步针对多视点视频的并行解码的研究则相对较少：[13] 提出了一种超空间 (hyper-space) 的方法，[14] 通过对参考帧施加一定限制条件使得解码器等待参考帧传入的时间大幅降低，从而提升并行解码速度。

庞一等在 2009 年提出了一种多核架构上并行处理多视点视频的启发式调度框架^[1]。不同于以往的帧并行或宏块并行的研究，他们提出了通过优化解码任务调度来提高解码器并行性能的观点，并通过实验证明性能的大幅提升。该方法实现一个如图 1.3 所示的框架，通过重新安排帧解码顺序，最小化多路视频解码时因为等待参考帧传入和其解码结果的等待时间，从而加快解码速度。

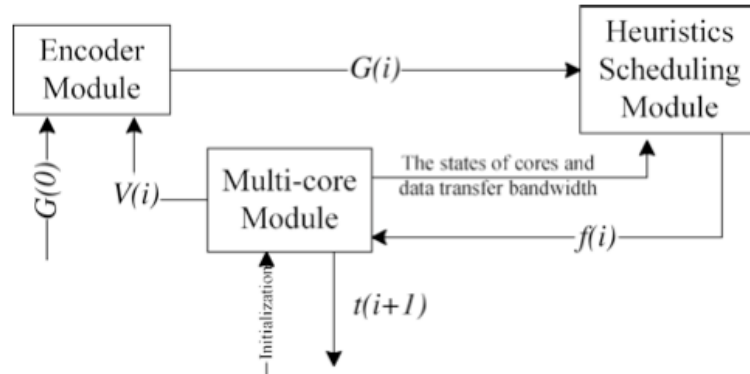


图 1.3 包含调度的 MVC 编解码框架

在此基础上，我们已经实现了一个成型的多视点视频编解码系统，包括编码器和解码器。编码器将若干 YUV 格式的视频编码为一个 MVC 视频。解码器将一个 MVC 视频解码为 N 个独立的 YUV 视频，N 为该 MVC 视频包含的视角数量。编解码结果符合 MVC 的标准^[15]。在第 2.2 节中将进一步介绍解码器部分。

1.3 本文的任务

目前已有的 MVC Decoder 工程速度较慢，不能满足实时解码播放的要求。本文主要通过针对单核和多核结构的优化，使现有的 MVC 解码器能够在 CPU 上做到至少两路分辨率为 720×576 的视频的实时解码。

此外，由于 MVC 视频的标准^[15]发布尚不足一年，硬件也尚未普及，还没有一种播放器能够支持各种硬件平台上的 MVC 视频播放。我们的实验环境有 Bolod 的 3D 平板电视和 nVidia 的 3D Vision 眼镜套装，为了实现一个从采集到播放的完整系统，除了解码器的优化以外，我们还需要设计实现一个 3D 播放器，让用户能够看到三维的播放效果。

1.4 本文的结构

本文第 2 章介绍需要优化的多视点视频解码器的原理和实现。第 3 章说明我们进行解码器性能优化的基本思路。第 4 章和第 5 章分别详述单核心性能优化和多核处理的并行优化过程。第 6 章介绍实验结果并进行了一定的分析。同时进行的 3D 播放器设计与实现部分与解码器优化是两个独立的工作，所以单列在第 7 章。最后在第 8 章总结解码器优化和 3D 播放器的工作，并对下一步的工作有所展望。

第 2 章 MVC 解码器原理与实现

自 2006 年起，胡伟栋等开始着手编写一套视频编解码系统。至 2010 年初基本完成，编解码的过程基本符合 MVC 标准^[15]的描述。编写过程中，主要参考了《H.264 and MPEG-4 video compression, video coding for next-generation multimedia》^[16]、T264 项目^①和 JMVC 参考软件。代码参考了 JMVC 的框架设置。

2.1 参考软件 JMVC 介绍

JVT 在发布 MVC 草案的同时还发布了 MVC 编解码的参考软件 JMVM (Joint Multi-view Video Model)。MVC 标准确立之后，JMVM 改名为 JMVC (Joint Multi-view Video Coding)，陆续有新版本发布。我们的编解码器起初参考的是 JMVC 4.2 版，今年由于新发布的 JMVC 7.2 编解码结果与旧版并不兼容，我们又针对新版做了更正。

JMVC 包括以下几个工程：

- H264AVCCommonLibStatic
- H264AVCDecoderLibStatic
- H264AVCEncoderLibStatic
- H264AVCVideoIoLibStatic
- MVCBitStreamAssembler
- MVCBitStreamExtractor
- DownConvertStatic
- PSNRStatic
- H264AVCDecoderLibTestStatic
- H264AVCEncoderLibTestStatic

其中，DownConvertStatic 是为了前向兼容；PSNRStatic 是进行编码质量分析，与 MVC 编解码实现无关，不做介绍。

H264AVCCommonLibStatic 包含编解码时都需要的一些函数，例如对读取

① <http://sourceforge.net/projects/t264/>

一帧的属性、各类 Filter 函数、量化与反量化函数、宏块的变换与反变换操作等等。H264AVCDecoderLibStatic 就是解码器的函数库，H264AVCEncoderLibStatic 则是编码器函数库。H264AVCVideoIoLibStatic 是进行视频文件读写操作的函数库。

由于 MVC 视频文件可能包含多路 H.264 视频，所以需要对码流进行打包，MVCBitStreamAssembler 就是这个模块；相应的，MVCBitStreamExtractor 就是从码流中抽取各路视频的模块。

除了函数库之外，JMVC 还提供了两个工程演示调用库函数进行 MVC 编码和解码的过程，分别在 H264AVCEncoderLibTestStatic 和 H264AVCDecoderLibTestStatic 中。

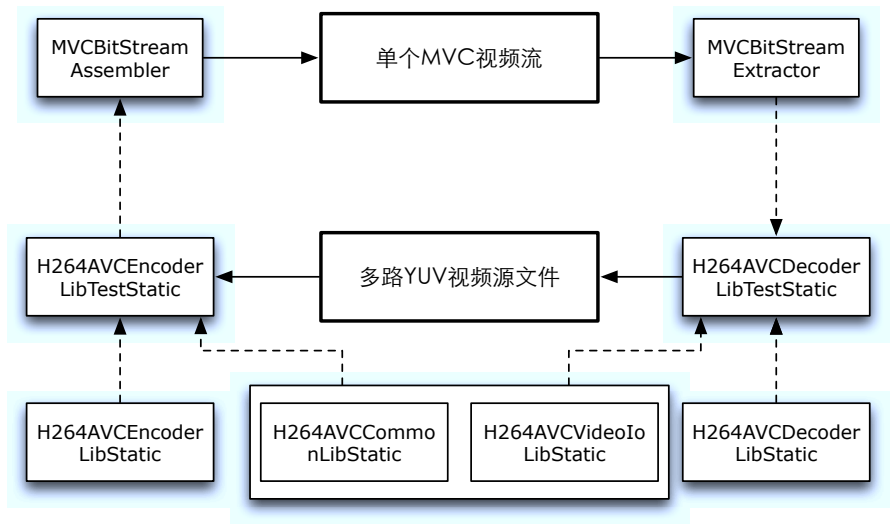


图 2.1 JMVC 编解码框架

利用 JMVC 进行编解码的框架如图 2.1 所示。

编码器的输入为若干路 YUV 视频，一般是同一场景从若干不同角度拍摄的视频，有相同的分辨率并且经过时间同步和校准。YUV 文件要求是 4:2:0 格式，每 $2 \times 2 = 4$ 个像素由一个 2×2 的亮度矩阵和两个 1×1 的色度矩阵组成。亮度矩阵顾名思义表示了 4 个像素的亮度，两个色度矩阵表示像素红蓝两色的平均色度，如图 2.2 所示。在编解码器开发期间，庞一、胡伟栋、张凤研等对编解码在帧级别的并行性进行了深入研究^[17,18]，并在编码器和解码器中都采用了帧并行的算法。

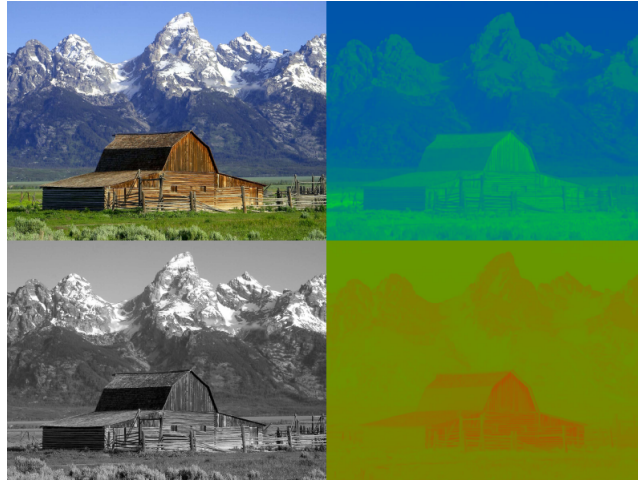


图 2.2 原始图像 (左上) 和其 Y' (左下)、 U (右上)、 V (右下) 分量

2.2 自主开发的 MVC 解码器

我们根据参考软件和 [16] 中的描述，实现了 MVC 的编解码器。其中，编码器的处理过程如图 2.3^① 所示。解码器简单来说就是编码器的逆过程，其流程如图 2.4^② 所示。

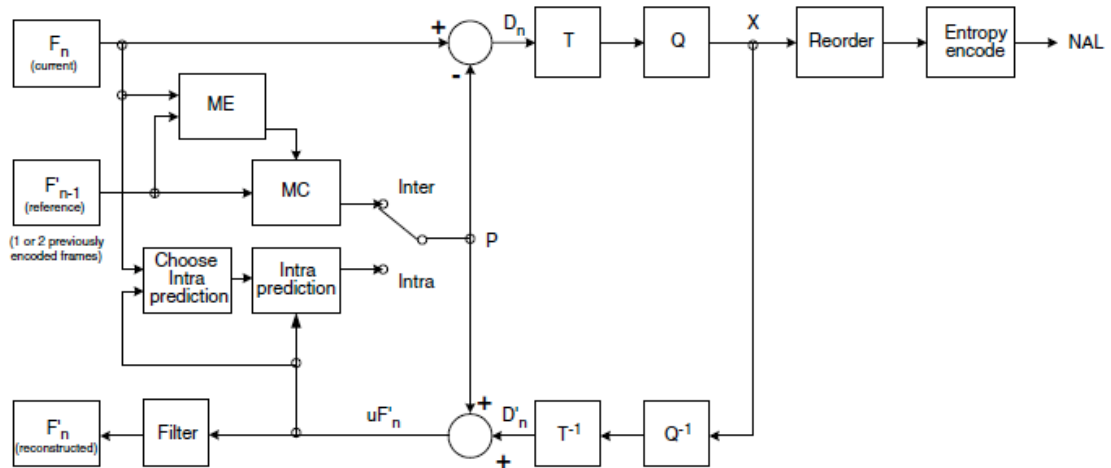


图 2.3 MVC Encoder 流程图

解码器中，在解码一帧之前，需要获得解码该帧所需要的全部参考帧，之后才能进行解码。所以说，帧的解码顺序与输入顺序是不一致的。在我们的解

① 《H.264 and MPEG-4 video compression, video coding for next-generation multimedia》^[16]p.160

② 《H.264 and MPEG-4 video compression, video coding for next-generation multimedia》^[16]p.161

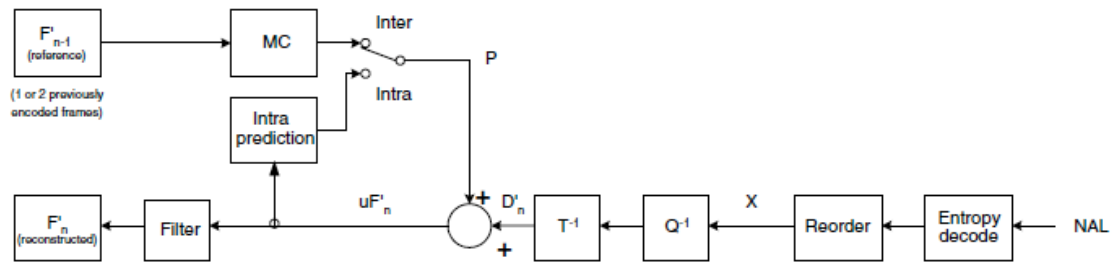


图 2.4 MVC Decoder 流程图

码器中，通过一个根据 [1] 实现的调度器来安排解码顺序。我们的解码器有一个线程进行解码源数据的读取，在实验中，是读文件流；相应的，有一个线程进行解码输出的写操作，在实验中是写文件流；此外，调度器会根据参数配置，同时启用 1 个到 N 个线程执行解码任务，N 不会大于设定的最大线程数。

2.3 小结

本章主要介绍了参考软件 JMVC 的模块划分、功能以及我们自主开发的 MVC 解码器的基本结构和解码流程。

第 3 章 解码器性能优化方案

毕设工作的主体是解码器性能优化，本章主要介绍性能优化的一些准备工作并提出优化方案。

第 3.1 节介绍了实验所用的软硬件平台。第 3.2 节介绍了对解码器进行性能分析的方法与结果，第 3.3 节以此为基础给出了性能优化的基本方案，最后在第 3.4 节中说明了优化的目标。

3.1 软硬件平台说明

我们的解码器支持多种硬件平台，包括 x86、CELL 等处理器；多种操作系统，包括 Linux 和 Windows。本文所做的优化如无特殊说明，皆与软硬件平台无关，可以直接在其他平台上应用。为了方便实验，实验主要在以下环境下进行：

- 硬件平台
 - Intel Core2 Quad Q9400 @ 2.66GHz ^①
 - 4GB DDR2/800 Memory
 - NVIDIA GeForce GTS 250 with 512MB
- 软件平台
 - Microsoft Windows XP Professional SP3 ^②
 - NVIDIA WHQL Driver v197.13
 - DirectX 9.0c
 - Microsoft Visual Studio 2008 SP1 ^③
 - CUDA SDK 3.0
 - Intel VTune Performance Analyzer 9.1 Build 385 ^④
 - Intel C++ Compiler Professional 11.1.038 ^⑤

① 禁止了 SpeedStep 功能

② Lenovo ThinkCenter M6100T 预装操作系统

③ 获取自：清华大学校园正版软件服务

④ 30 天评估版，序列号为 VVVC-BDGCWFJC

⑤ 授权同 VTune

3.2 解码器性能分析

对解码器进行优化的过程中，首先需要对其现有的性能表现有一个细致的了解。对于每一个函数的被调用次数、运行时间、运行时间百分比等指标都需要有较为精确的测量，才能更好地进行接下来的优化。

对此，我使用了两个性能分析工具来分析 MVC Decoder 工程。分别在接下来两节中说明。

3.2.1 VS2008 内置性能分析

Visual Studio 2008 Team System 版内置了一个 Analyze 功能，可以对目标程序进行性能分析。性能分析有两种：

- 一种是不改变编译结果，通过运行时采样，得到每个采样时刻正在运行的函数，经过汇总后得到函数在采样上的分布情况，由此估计每个函数运行时间占总运行时间的百分比。这种方式称为 **Sampling** 方式。
- 另一种是在编译时加入一些辅助代码，运行时通过这部分代码来标志进入和推出一个函数的时间，借助这些隐藏的输出信息得到总运行时间在各个函数内部的分布。这种方式称为 **Instrumental Code** 方式。

3.2.1.1 性能分析步骤

在 VS2008 的菜单中有一个“Analyze”项，选择“New Performance Wizard”打开一个性能分析向导。

选择要分析的项目、分析的方式之后就可以开始分析了。

成功执行植入了 **instrumental code** 的可执行程序之后，**profiling** 工具收集到大量的数据。经过一段时间（在我的机器上大约两分钟）的处理，会给出一个性能分析报告。

3.2.1.2 性能分析结果

性能分析报告中的 **FunctionList** 视图给出了各个函数（包括系统调用）占用的时间、时间的百分比以及被调用次数等等数据。我们关心的是其中的“**Exclusive Time**”、“**Number of Calls**”两列。当然，“**Exclusive Time %**”也被我们选中，因为百分比相对于毫秒数能让我们更直观地了解函数占用的时间。

表 3.1 VS2008 性能分析报告（前 10 项记录）

Function Name	Exclusive Time ^①	Exclusive Time %	Number of Calls ^②
macroblockPredGetDataUV	341.01	14.41	2,632,478
macroblockPredGetDataY	321.37	13.58	1,316,239
idct4x4_c	316.86	13.39	5,028,960
macroblockGetPred_axb	181.29	7.66	1,316,239
macroblockGetPred_axb_Bi	133.42	5.64	571,944
macroblockGetHalfPel	132.47	5.6	148,552
Filter	101.92	4.31	6,215,308
addMacroblockdata	85.36	3.61	184,171
iquant4x4_c	84.48	3.57	5,028,960
macroblockPBPrediction	70.95	3	184,171

① 不包括函数体内调用其他函数的净运行时间

② 函数被调用次数

分析报告中仅仅 MVCDecoder.exe 自身的函数就有近 200 条记录，而大部分都占用不足 0.05% 的时间，优化价值不大。在表 3.1 中我们给出了时间占用排在前 10 的记录。更详细的记录参见附录 B 中的表 B.1。

3.2.2 VTune 分析

在实验过程中，我们发现大部分函数运行总时间没有明显差别，与我们进行项目合作的北京世纪鼎点软件有限公司的罗翰先生指出，VS2008 的性能分析可能不够准确，使用 Intel 的 VTune 或许能够得到更准确的性能分析结果。

关于 Intel VTune 如何精确地进行性能分析，文 [19–22] 中有介绍。

3.2.2.1 性能分析步骤

打开 VTune Performance Analyzer 之后，我们新建一个工程，选择采集 call graph 数据。配置好要运行的可执行文件和执行路径就可以开始分析了。与 VS2008 的分析一样，经过一段时间的执行和数据处理，我们获得一个性能分析报告。

表 3.2 VTune 性能分析报告 (前 10 项记录)

Function	Calls	Self Time	% in function
macroblockPredGetDataUV	2632478	774863	0.61
macroblockPredGetDataY	1316239	702575	0.58
idct4x4_c	5028960	398076	1
macroblockInterDecode16x16_y	184171	328834	0.46
macroblockGetPred_axb	1316239	311730	0.11
FilterMB	210600	203506	0.21
Filter	6215308	191937	1
iquant4x4_c	5028960	157459	1
macroblockInterDecode_uv	184171	154697	0.48
macroblockPBPrediction	184171	150609	0.05

3.2.2.2 性能分析结果

性能分析报告包含一个函数调用关系图以及函数运行时间表，我们关注的是后者。表中大部分列与 VS2008 的分析结果是公共的，比如函数名、总时间、净时间等等。我们采集了与此前同样意义的几列数据，包括“Function”，“Calls”，“Self Time”和“% in function”。我们在表 3.2 中给出前 10 条记录。更详细的记录参见表 B.2。

3.2.3 性能分析结果说明

我们观察两个分析结果表 3.1 和表 3.2 可以发现：虽然两个性能分析工具在结果上存在一些差别，但函数运行时间的排序大体上是一致的，耗时多的函数都排在表格靠上的位置，表中列出的前 10 位的函数有 7 个在两个表中都出现了。我们认为，可以根据性能分析的结果来确定函数优化的大体方向，优化的主要对象就是表中排在前列的函数。

对这些函数进行进一步观察，我们可以将它们分类：

简单函数多次调用 典型的例子是 `idct4x4_c` 和 `iquant4x4_c`。这两个函数的调用次数都在 5028960 次，这仅仅是解码 65 帧两路视频所需要的次数。对于这类函数，哪怕是性能上的一点点提升，都会因为巨大的调用次数而对总解码时间产生较大的影响。

复杂函数 典型的例子是 `macroblockGetHalfPel`，这个函数仅仅执行 148552 次，

单次执行时间是 `idct4x4c` 的 14 倍，`iquant4x4c` 的 53 倍。这样的函数需要有很大幅度的优化才会在总时间中有所体现。

同时，我咨询了对整个系统十分熟悉的胡伟栋，他指出：耗时排在前列的函数大多是根据 JMVC 参考软件的逻辑来编写的，其考虑了一些工程上扩展的需要而非性能优先，因此有一定的提升空间。

3.3 优化方案

在经过几次组会讨论之后，我们确定了如下的几个优化方案，分别对不同类型的函数进行重写逻辑、优化循环、汇编优化和 CUDA 优化。从重写逻辑开始，一步步优化解码器性能，直到符合优化目标（见第 3.4 节）为止。

重写函数逻辑 解码器中有一些函数的逻辑判断有多处重复，降低了性能。如果能够理清楚函数逻辑，修改函数内部执行的顺序，就能够加快一些速度。这样的优化在第 4.1 节中有例子及说明。

循环的优化 对帧和宏块的处理常常有两层 `for` 循环便利一个 $width \times height$ 的像素矩阵，逐像素进行操作。对于这样的函数，首先考虑是否能够将多次循环内的操作合并到少量循环，再考虑循环结构对 `cache` 的友好程度。这样的优化在第 4.2 节中有例子及说明。

汇编优化 在项目讨论中，罗翰先生提出 `ffmpeg` 等开源的视频项目中可能会有些变换的汇编优化版本，如果能够应用汇编优化，那么这些函数的性能将会有大幅度的提升。就此，我们考虑对一些函数进行汇编优化。第 4.3 节中进一步说明了汇编优化。

CUDA 优化 视频处理中很常见的就是并行性无处不在，解码器的调度器保证了帧解码具有一定的并行性，而对帧和宏块内部的处理往往还有可以并行的 `for` 循环逐像素操作，利用 GPU 的大量核心进行简单操作可以加快这类操作的速度。

3.4 优化目标

我们进行 MVC 解码器优化的目标是为了能让普通用户使用 PC 作为终端能够收看 3D 视频。在显卡尚未内置 MVC 硬解码器的情况下，目前所有的解码任务都交给 CPU 来完成。我们将优化的目标设定在用户使用主流 CPU 能够进行

两路标清视频^①的实时解码。

量化的指标就是，用 CPU 进行两路分辨率为 720×576 的视频，每一路都能达到 30 帧/秒，总计 60fps 的解码速率。

达到上述目标之后，我们的解码器就能够用来在双目 3D 显示平台下开展实际应用了。如果想要使用我们同时拥有的 Bolod 生产的裸眼观看的 3D 电视，则需要输出 8 路信号。这在目前的 CPU 软解码算法上较难实现，目前有两种解决方案，一种是直接利用 GPU 加速解码过程，软解 8 路信号；另一种是 CPU 解码出 2 路信号，再用 GPU 通过 2 路立体信号合成出 8 路需要的信号。前一种方式已经在 NVIDIA 的蓝光播放器中应用了，不过其解决方案并不开源；后一种两路信号合成八路信号的项目，清华大学媒体所的李化常师兄正在进行中。在 2010 年 3 月已经实现了合成算法，将两帧画面合成出八帧大约耗时 1 秒，目前正在进行算法的优化工作。

3.5 小结

本章主要介绍了用 Visual Studio 内置的 profiler 以及 Intel VTune 对解码器进行性能分析的过程，给出了分析结果，列举了耗时最长的 10 个函数并对这些结果进行了一定的说明。针对性能分析结果，提出了优化方案，并明确了性能优化要达到的目标。

^① 关于视频分辨率，有标清SDTV、增强型标清EDTV和高清HDTV等制式，我们所说的标清指的是国内广泛使用的EDTV中的PAL制式视频，分辨率为 720×576 。

第 4 章 解码器单核性能优化

按照第 3.3 节中提到的优化方案，我们对解码器的单核性能进行了若干优化。本章中列举其中具代表意义的部分。

第 4.1 节详细介绍了对函数逻辑的优化，第 4.2 节介绍了对循环的优化并举例说明，第 4.3 节介绍了汇编优化，第 4.4 节提及了其它一些优化。

4.1 函数逻辑优化

在 MVCCCommonLib 模块中，PBPrdict.cpp 文件中包含了一些宏块预测相关的函数。此前的性能分析结果表 3.1 中显示，其中的 macroblockGetHalfPel() 函数耗时较多。这是一个使用 Finite Impulse Response (FIR) 计算一个 Luma 块的 half-pel 的函数。具体算法参考了《H.264 and MPEG-4 video compression, video coding for next-generation multimedia》^[16] 第 173 页起的描述。

函数内的一部分源代码如下：

代码 4.1 macroblockGetHalfPel() 函数片段（优化前）

```
1 ...  
2 for (int i = 0; i < hw; ++i){  
3     for (int j = 2; j < hh - 3; ++j){  
4         pif->vdata_half[0][i][j] = ...; //some expression of vdata_half[][][]  
           and vdata[][]  
5     }  
6 }  
7 for (int i = 0; i < hw; ++i){  
8     for (int j = 2; j < hh - 3; ++j){  
9         tmp[i][j] = ...; //some expression of vdata[][]  
10    }  
11 }  
12 ...
```

检查两次循环中对变量的访问，第二次循环需要访问 **vdata** 数组，而第一次循环只修改 **vdata_half** 数组，对 **vdata** 没有做任何改动，所以两个循环可以合并成一个。

最后形成的代码如下：

代码 4.2 macroblockGetHalfPel() 函数片段（优化后）

```

1 ...
2 for (int i = 0; i < hw; ++i)
3     for (int j = 2; j < hh - 3; ++j){
4         pif->vdata_half[0][i][j] = ...; //some expression of vdata_half[][][]
           and vdata[][]
5         tmp[i][j] = ...; //some expression of vdata[][]
6     }
7 ...

```

这个优化使程序运行时间^①缩短了 1016ms^②。

同样在 MVCCCommonLib 模块中的 PBPrdict.cpp 里，macroblockPredGetDataY() 和 macroblockPredGetDataUV() 函数也是运行耗时很多的函数，事实上，不论是 Visual Studio 还是 VTune 的分析结果，这两个函数都是占用时间最多的（见表 3.1和表 3.2）。

这两个函数大部分的时间耗费在 memset 和 memcpy 上。对于这些系统 api 的调用，我们无法作修改，但是仔细观察其逻辑，我们发现函数的逻辑判断结构可以做一定优化。

源代码如下：

代码 4.3 macroblockPredGetDataY() 函数片段（优化前）

```

1 ...
2 if (x<0) x = 0;
3 if (x >= pif->sizeh) x = pif->sizeh - 1;
4 ...
5 if (le < 0) le = 0;
6 if (le > 0)
7     memcpy(pif->vdata[i]+dl, linef + sl, le);
8 ...

```

由于 pif->sizeh 的物理意义决定了它必然是非负的，当 x=0 的时候不可能出现 x>=pif->sizeh 的情况，所以每次调用 macroblockPredGetDataY() 的时候，如果 x<0，都做了多余判断。

le 是 int 类型，所以当 le<0 时，顺序执行语句不可能出现 le>0 的情况，也存在多余判断问题。事实上，第7行的 memcpy 语句只有在 le>=1 的情况下才会

① 这里所说的程序运行时间是指用 release 版本的解码器解码一个长度为 65 帧，分辨率为 720×576，路数为 2 的 Multi-view 视频所耗费的时间。下同。

② 原始的版本总运行时间为 26344ms。

执行，而当 $le < 1$ 时， le 都会被赋值为 0。

经过优化后的代码如下：

代码 4.4 macroblockPredGetDataY() 函数片段（优化后）

```
1 ...  
2 if (x < 0) x = 0;  
3 else if (x >= pif->sizeh) x = pif->sizeh - 1;  
4 ...  
5 if (le < 1) le = 0;  
6 else  
7     memcpy(pif->vdata[i]+dl, linef + sl, le);  
8 ...
```

对 macroblockPredGetDataUV() 的优化同理。这两个函数优化后，程序运行时间再次减少了 437ms。

在 macroblockGetPred_axb() 中，原始的程序一共用了超过 20 个 if 语句来决定如何进行预测。根据我在《数字逻辑设计》以及《计算机组成原理》课程中的实验经验，这样的多重 if 语句用 switch 语句替换不但能使代码逻辑更加清晰，而且能提高性能。这相当于是把许多串联的 2 位的 MUX 改写成一个多位的 MUX。

此处优化使程序运行时间缩短了 47ms。

4.2 循环优化

macroblockGetPred_axb() 函数耗时在表 3.1 和表 3.2 中分别列在第 4 和第 5 位，也是耗时极多的函数。除了上一节提到的 if 语句改写成 switch 语句之外，我们还对其进行了循环内部的优化。

首先是循环变量的声明，根据我们的经验，编译器会对循环变量的声明放在循环体内部的写法做优化，至少在 linux 平台下的 gcc 会这样做。再加上将循环变量的声明放在循环内部可以避免变量在作用域之外被使用，所以我们将所有的循环变量声明全部移到循环体内部。

其次，我们观察代码发现有多处如代码 4.5 所示的循环。

代码 4.5 macroblockGetPred_axb() 函数片段（优化前）

```
1 ...  
2 for (i = 0; i < size_x; ++i)
```

```

3   for (j = 0; j < sizey; ++j)
4       pred->y[i+off_in_x][j+off_in_y] = pif->vdata_half[halfid][i+borderT][
        j+borderL];
5   ...

```

这些循环内的操作有一个特点：使用包含循环变量的表达式作为数组访问的下标。

以代码 4.5 中第 4 行为例，在整个循环中，计算了 $size_x \times size_y$ 次 $i + off_in_x$ 和 $i + borderT$ 。实际上可以在内层循环外就进行计算，这样就只用计算 $size_x$ 次，极大地减少了加法操作，同时大幅减少了变量访问次数：本来要访问两个变量，放在寄存器中，再做加法；现在只用访问一个计算好的变量即可。

优化后的代码如下：

代码 4.6 macroblockGetPred_axb() 函数片段（优化后）

```

1   ...
2   for (int i = 0; i < size_x; ++i){
3       int ioffset_x = i+off_in_x;
4       int iboarterT = i+borderT;
5       for (int j = 0; j < size_y; ++j)
6           pred->y[ioffset_x][j+off_in_y] = pif->vdata_half[halfid][iboarterT][j+
            borderL];
7   }
8   ...

```

经过上述优化，程序运行时间缩短了 1031ms。

4.3 汇编优化

我们查找了一些文献，发现可以用 MMX 和 SSE 等扩展指令集来加速一些矩阵变换函数，例如 `idct8x8`。Intel Application Note AP-922^[23] 说明了如何用 MMX 和 SSE 加速 DCT 变换。AP-945^[24] 说明了如何用 SSE2 加速 IDCT 变换。

实际操作中，我们采用了 Laurent Aimar、Loren Merritt、Holger Lubitz 和 Min Chen 编写的 SSE2-optimized H.264 iDCT 代码替换 C++ 实现的 iDCT 代码。

由于我们测试用的样例视频对 `idct8x8` 的使用远不及 `idct4x4` 频繁，所以这部分优化在程序运行时间上并没有明显的体现。这是由于我们目前主要测试的是 H.264 的 Base Profile，当我们的解码器将来支持 High Profile 的时候，就会大量调用 8x8 的变换了。

4.4 其它优化

除了以上优化之外，我们还进行了一些细微的优化。

在每一路 H.264 码流的解码过程中，我们经常需要判断一些标志位的值，根据这些标志位来选择做什么样的操作。这往往是通过一系列的 `if-else` 判断实现的。这就出现了很多的选择分支，进入每个分支的概率是不同的。我们通过解码大量不同的视频文件，记录这些分支的选择情况，对判断的顺序进行了微调。将大概率的事件优先判断，执行相应的操作，省去了一些在进入分支之前的无效判断。

这样的优化对所有的测试样例平均提高 1% 的性能。

我们还尝试了更换编译器，使用 Intel C++ Compiler Professional（见第 3.1 节）来替换 Visual Studio 包含的编译器。

用 ICC 编译的代码比 VS 默认编译器编译出来的代码快 5% 到 6%。

4.5 小结

本章主要介绍了对解码器进行优化，包括逻辑重构、循环优化、汇编优化以及其它（例如编译器）的优化。这些优化主要提高了单核解码性能。尽管有了这些优化，解码器的单核性能仍然不能满足需求，所以还需要做并行化，这一部分在下一章中展开。

第 5 章 解码器并行优化

经过第 4 章的优化之后，我们的解码器性能仍然没有达到第 3.4 节的要求。所以我们继续进行并行优化。本章主要从三个方面进行展开，首先第 5.1 节分析解码器的并行性，然后第 5.2 节介绍我们所实现的并行框架，最后第 5.3 节说明并行中遇到的问题与解决方法。

5.1 对解码器并行性的分析

在第 1.2 节中提到了 H.264 并行方向的一些研究^[10–14]。虽然他们提出的具体方法我们没有采用，但是其思想与我们进行并行解码所做的思考是互通的。

解码过程中可能的并行性来自于两个方面：帧和宏块。

在解码帧的过程中，需要考虑参考结构。我们常常使用如图 5.1^①所示的参考结构。

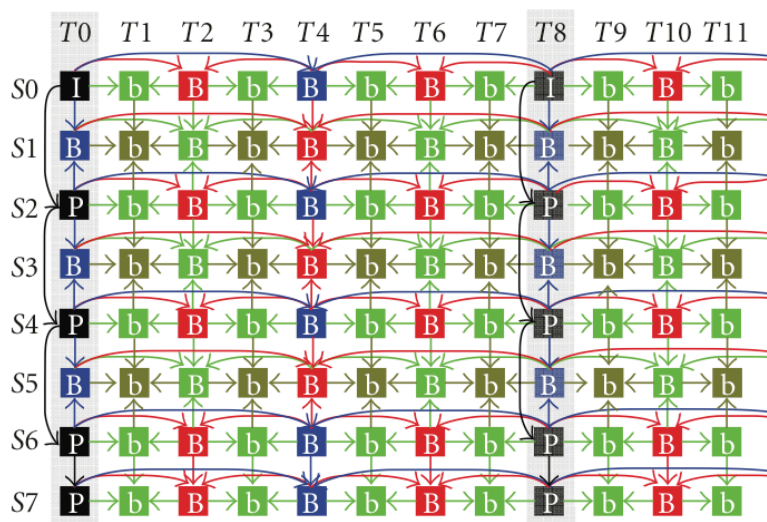


图 5.1 典型的 MVC 参考结构

解码其中的 I 帧时，不需要参考任何帧，只要 I 帧的 NAL Unit 全部传入就可以开始对该帧解码。而对其中的 P 帧和 B 帧，都有箭头指向该帧，表示

① 图片选自 [7] 中的 Section 3

对这一帧的解码需要参考别的帧。我们用 (S_n, T_n) 表示在 T_n 时刻， S_n 这一路码流对应的帧。 (S_0, T_1) 就需要参考 (S_0, T_0) 和 (S_0, T_2) ，而 (S_0, T_2) 又要参考 (S_0, T_0) 和 (S_0, T_4) ， (S_0, T_4) 则要参考 (S_0, T_0) 和 (S_0, T_8) ，每一帧的解码都必须走过一条路径才能满足其参考帧都已经完成解码的条件。这样就形成了一个分层的图，从一个空的根节点向下，第 i 层的节点是第 1 到第 $i-1$ 层解码完毕后立刻可以解码的帧。这样的图中，每一层的帧都可以并行地分别解码。如果将所有的视点 S_0 到 S_8 都考虑进来，情况就更加复杂。庞一等提出的多视点视频并行调度框架^[1]就给出了这样的图的计算过程，对于图 5.2 所示的预测结构，通过他们的算法能够得到如图 5.3 所示的解码调度图^①。

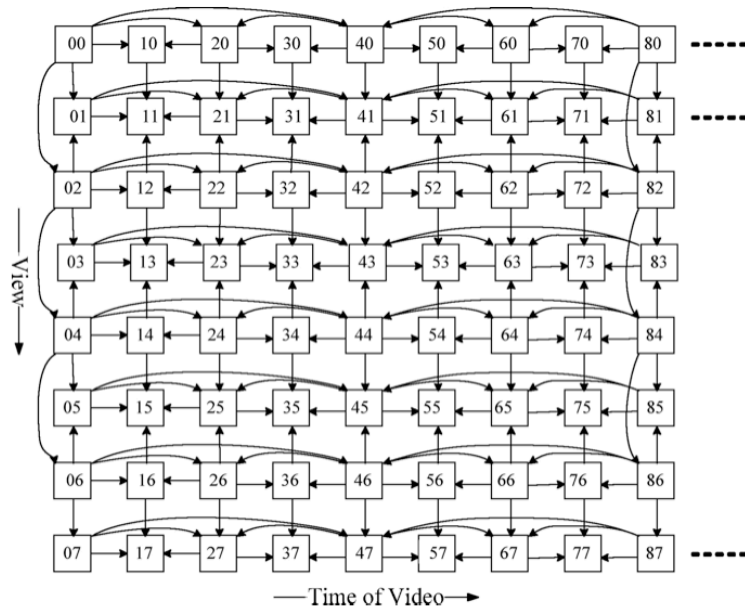


图 5.2 GOP 预测结构

在解码宏块的过程中，也有相似参考结构，如图 5.4^② 所示。在解码 $MB(4,1)$ 的时候，需要参考 $MB(3,0)$ 、 $MB(3,1)$ 、 $MB(4,0)$ 的解码结果。从 T_1 时刻解码左上角 $MB(0,0)$ 起， T_2 时刻可以解码 $MB(1,0)$ ， T_3 时刻可以同时解码 $MB(2,0)$ 和 $MB(0,1)$ ， T_4 可以同时解码 $MB(3,0)$ 和 $MB(1,1)$ ，以此类推。如果硬件资源足够同时解码 3 个宏块，那么全部 16 个宏块只要 13 个时间片就可以解码完成。

① 图 5.2 和图 5.3 分别选自 [1] 的 Section III 与 Section IV

② 图片选自 [25] 中的 Section 4

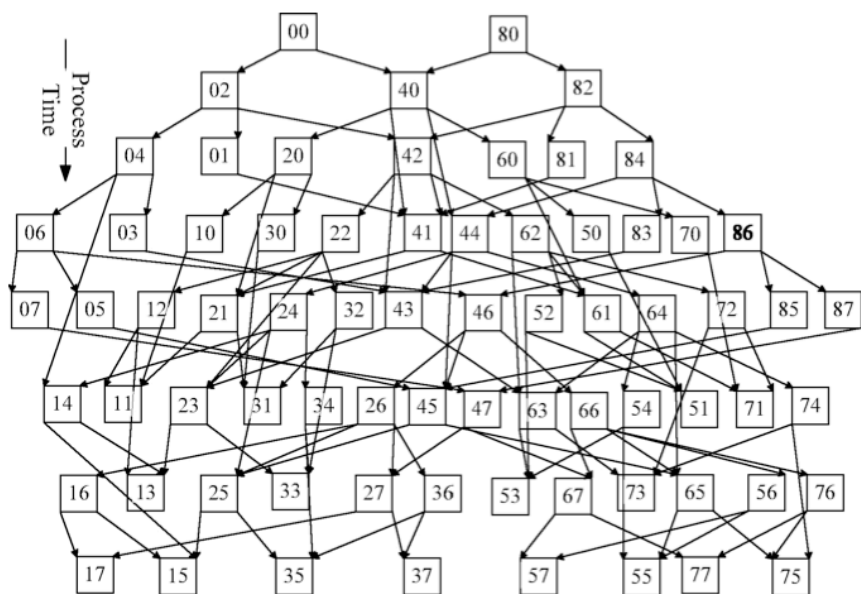
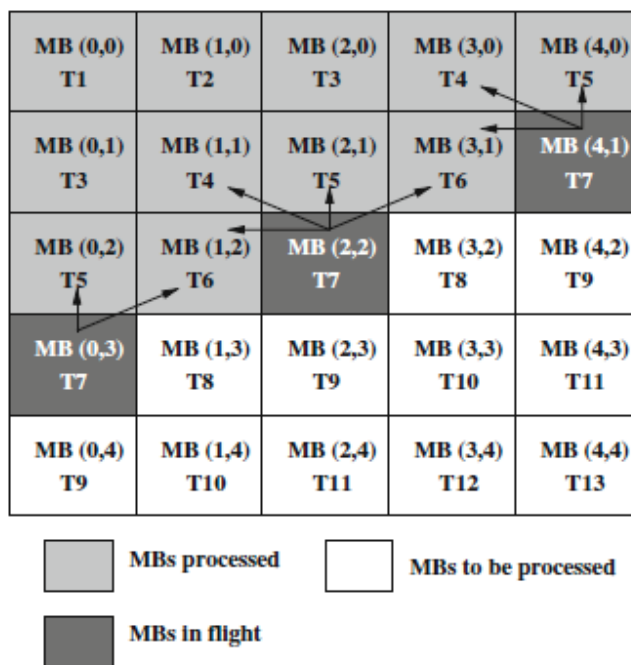


图 5.3 多视点视频中的 GOP 调度



5.2 并行框架介绍

我们的解码器的框架如图 5.5 所示。有一个线程作为控制器控制整个解码器的工作，一个 `BitStreamReader` 线程负责读取源码流，一个 `RawDataWriter` 线程负责写解码后输出的码流。调度器就是 [1] 中的算法实现，控制着 1 个到多个解码线程进行解码操作。可以看出，即使将解码线程的并行程度设为最多同时 1 个线程，整个解码器进程实际还是启用了多个线程的。

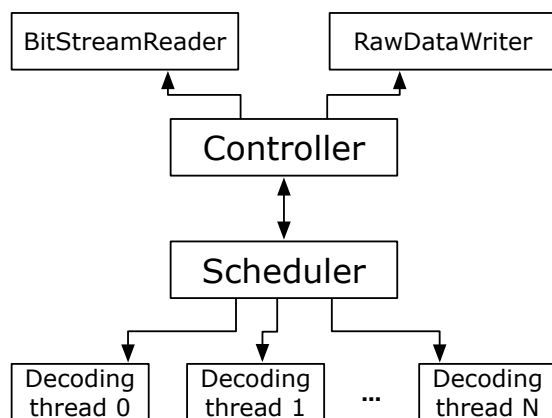


图 5.5 解码器模块关系

5.3 问题与解决

测试多线程解码的过程中，我们发现起初实现的版本在解码线程大于 2 的情况下，有不小的概率进入死锁状态，停滞在某一帧不前。这个概率随着解码线程数的增加而增大，在解码线程数为 2 的时候也有较小的概率进入这样的状态。这就令我们的并行解码实现失去了实用价值，我们不可能要求用户不断地尝试解码直到某次成功。

经过跟踪调试，我们找到了问题所在：在一些切换线程的位置上，我们使用了 `sleep(0)` 来暂停当前线程，希望回到控制线程。正是 `sleep(0)` 导致了程序停止后续任务的执行。过程如下

1. 读入一些 NAL Unit，切换到解码线程
2. 解码线程发现数据不足以进行下一步解码，希望切回控制线程要求进一步数据
3. （这时应该置一个标志位通知控制线程做相应动作）

4. 但是此时没有同一优先级的线程希望获得资源，由于 `sleep(0)` 在其他线程不请求资源的情况下立即返回当前函数，而标志位尚未改变，于是就出现了死锁。

将所有的 `sleep(0)` 改为 `sleep(1)` 后，问题解决。经过查阅资料，我们发现，在 Linux 环境下，`sleep(0)` 的确常被用于将 `cpu` 交给其它线程，但是在 Windows 下，经过测试发现 `sleep(0)` 会使 `cpu` 占用率接近 100%，所以实际上是一个死循环，没有其它任务的时候空消耗系统资源。

5.4 小结

本章介绍了解码中存在的并行性，对帧并行和宏块并行都有所解释；之后说明了我们实现的解码器并行框架；最后提到了并行解码尝试中发现的一个问题和解决方法。

第 6 章 解码器性能优化实验结果

本章主要介绍对解码器优化结果的测试方法及结果。首先说明了生成测试数据的过程，然后使用这些数据对优化前后的解码器进行测试，对比结果。

6.1 测试数据的生成

测试数据采用第 2.1 节中提到的参考软件 JMVC 和 ffmpeg^① 配合生成。ffmpeg 用来将多种格式（如 AVI、MP4）的视频转换为 YUV 格式。JMVC 中的 H264AVCEncoderLibTestStatic 用来对 N 路 YUV 视频压缩成 H.264 流，MVCBitStreamAssemblerStatic 用来将多路 H.264 打包成一个 MVC 视频流。我们测试的视频分为 3 类：一路、两路和八路的多视点视频。他们的生成方法较为类似，我们以八路视频为例，说明生成的具体方法：

1. 首先要编写一个描述视频压缩参数的配置文件，我们实际使用的配置文件之一见代码 C.1。其中比较关键的项为

InputFile 输入的文件名格式，golf1 表示输入的各路视频为 golf1_0.yuv、golf1_1.yuv、...、golf1_7.yuv。

OutputFile 输出的文件名格式，与 InputFile 类似。输出的文件扩展名为 .264。

SourceWidth 待编码视频的水平分辨率^②，如 320。

SourceHeight 待编码视频的垂直分辨率，如 240。

FrameRate 待编码视频的帧率，如 30.0。

FramesToBeEncoded 从 YUV 文件的第一帧算起，一共需要编码的帧数，如 623。

GOPSize 参考结构中的 GOP 大小的上限，如 4。对这个参数的设置影响参考结构的复杂程度。

NumViewsMinusOne 编码视频的路数减 1，如 7。

ViewOrder 各路视频的编码顺序，如 0-2-1-4-3-6-5-7。

① <http://www.ffmpeg.org/>

② 由于 YUV 视频的内容只是视频每一帧的实际数据，不像我们常使用的一些被包装在容器中的视频格式，并没有存储视频的分辨率、帧率等参数。

其它还有一些对每一路视频的参考结构的限定，我们采用与 JMVC 的 demo 中相同的参考结构。

2. 有了上述的配置文件 MVC.cfg^① 之后就可以使用如下的命令来编码了

```
H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 0
```

这个命令使用 MVC.cfg 中定义的参数对 View 0 进行编码。想要编码全部 8 路视频需要严格按照 MVC.cfg 中 ViewOrder 定义的顺序进行编码。

3. 编码生成了 8 个 H.264 码流之后，我们需要将其打包成一个文件。MVCBitStreamAssemblerStatic 就是这样的一个人 packer，它也需要一个配置文件，样例见代码 C.2。其中定义了输入输出的文件名以及视频的路数。这里对输入文件的列举顺序没有要求，只要文件与视角对应即可。使用如下命令可以将码流打包：

```
MVCBitStreamAssemblerStatic.exe -vf assembler.cfg
```

4. 由于 JMVC 的编码器性能极其低下^②，我们推荐使用批处理来依次执行这些编码命令，免去每隔十几或几十分钟需要人工进行下一步编码命令的输入操作。一个批处理的样例见代码 C.3。

我们能够在网上获取的用于研究的 YUV 格式多视点样例视频片段大多是 QVGA 的分辨率，这一方面由于在我们进行多视点视频解码器优化之前，已有的编解码器性能很差；另一方面是由于 YUV 格式对数据几乎没有压缩，需要大量的存储空间^③。我们的测试目标针对的是分辨率至少为 720×576 的视频，这样的 YUV 源很难获得，需要自己生成。

值得庆幸的是，NVIDIA 为了推广其 3D 产品，在网站上提供了几段高清的两路视频片段。我们使用 ffmpeg 将其中分辨率为 1440×1080 的一个 WMV 视频转换成两路独立的 YUV 文件。

1. 我们首先使用了 MPEG Streamclip^④ 将这个 2880×1080 的 WMV 裁剪成左右两个 1440×1080 的部分。分别存为 left_1440x1080.mp4 和 right_1440x1080.mp4。
2. 使用如下命令将视频转为目标分辨率（如 1280×720 ）的 YUV 视频：

```
ffmpeg.exe -y -i left_1440x1080.mp4 -ss 41 -vframes 65 -s 1280x720 dual-
```

① 文件名可以取任意合法的文件名，对扩展名没有要求。

② 我们编码这个分辨率为 320×240 ，一共 623 帧的 golf623.mvc 总共耗费了近四个小时。

③ 我们测试用的 1280×720 的两路视频，仅仅两秒左右（65 帧）就需要 $85.7\text{MB} \times 2$ 的空间。

④ <http://www.squared5.com/>

test_1280x720_65f_0.yuv

```
ffmpeg.exe -y -i right_1440x1080.mp4 -ss 41 -vframes 65 -s 1280x720 dual-  
test_1280x720_65f_1.yuv
```

这里我们转换了从第 41 帧起的 65 帧画面。

3. 对这两个生成的 YUV 视频用前文提到的 JMVC 编码的方法就可以编码并打包成两路的 MVC 测试视频了。

我们还使用了一段电影 *Avatar* 的花絮 *The World of Pandora* 中的片段生成一路的测试视频。

使用如下命令生成 YUV 文件：

```
ffmpeg.exe -y -i The_World_of_Pandora.mov -ss 15 -vframes 65 -s 1280x720 sin-  
gletest_1280x720_65f_0.yuv
```

再用 JMVC 编码打包成一路 MVC 视频。

至此我们生成了全部用于测试的 MVC 视频，进行测试用到的所有视频的属性如表 6.1 所示：

表 6.1 性能测试中使用的所有视频一览

Clip Name	Resolution	Num of Frames	Num of Ways
dual144	176 × 144	65	2
dual288	352 × 288	65	2
dual576	720 × 576	65	2
dual720	1280 × 720	65	2
golf25	320 × 240	25	8
golf25dual	320 × 240	25	2
golf623	320 × 240	623	8
race81	320 × 240	81	8
single144	176 × 144	65	1
single288	352 × 288	65	1
single576	720 × 576	65	1
single720	1280 × 720	65	1

6.2 实验结果及说明

我们用解码器解码表 6.1 中的每一个视频。优化前的基准性能见表 B.3, Visual Studio 自带的编译器编译出的解码器的性能见表 B.4, ICC 编译出来的解码器性能见表 B.5。

参考《 \LaTeX and the GNUPLOT Plotting Program》^[26] 和《gnuplot 4.4: An Interactive Plotting Program》^[27], 用 gnuplot^① 对其中部分数据制图。

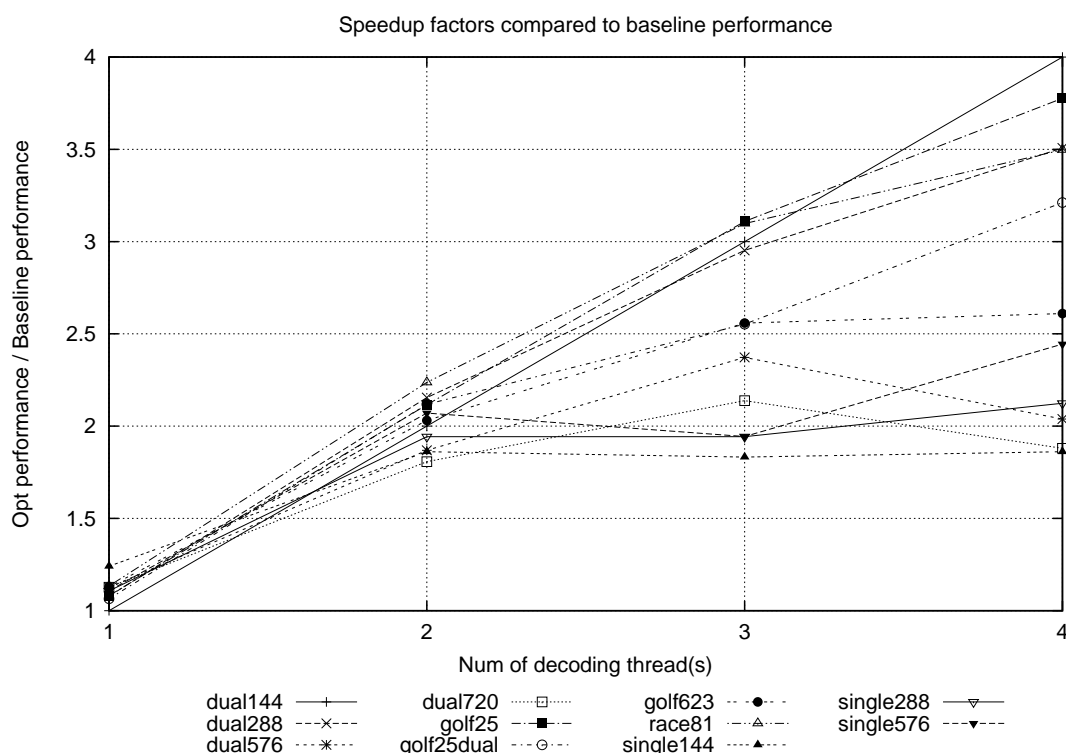


图 6.1 解码不同视频的加速比 (使用 MS 编译器)

经过优化后的解码器支持大于 1 的解码线程数。用 Visual Studio 内置的微软编译器编译的解码器用 1 到 4 个线程解码各个测试样例的性能与优化前的基准性能相比的加速比如图 6.1 所示。用 ICC 编译器编译的解码器的加速比如图 6.2 所示。更清晰的版本附在本章末尾的图 6.4 和图 6.5。

可以看出, 几条折线的趋势都证明增加线程数能够加快解码速度, 而几条折线在线程数为 3 和 4 时的变化又有所不同:

① <http://www.gnuplot.info/>

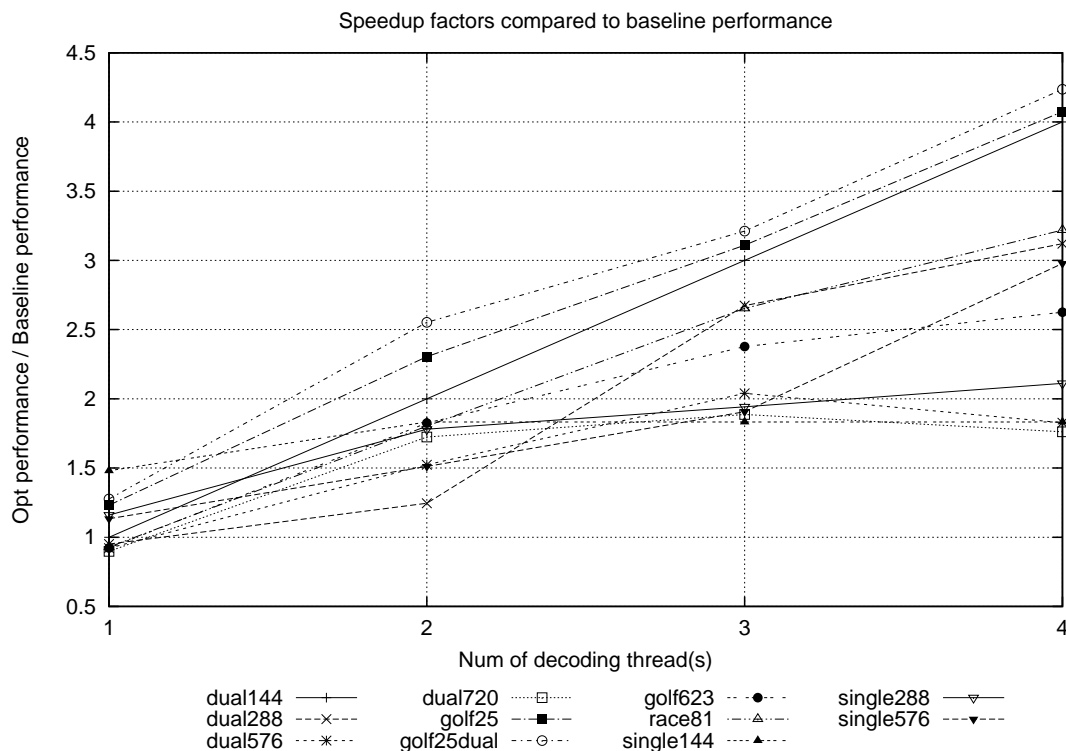


图 6.2 解码不同视频的加速比 (使用 ICC 编译器)

- dual144 的加速比正好为线程数，这是一个巧合。不过也能部分说明当视频的并行性足够时，增加的线程资源能够被充分利用。
- golf25、golf25dual、dual288 和 race81 这四个视频也能充分利用额外的资源。
- single144、single288、single576、single720 这些单路的视频加速比都没有超过 2.6，这是由于一路视频的参考结构比较简单，没有视角间的参考。调度器很难安排出能够并行处理的大于 2 个任务。
- golf623 的加速比低于预期，我们推测原因在于编码时的 GOPSize 设成 4 对于 8 路视频偏小了，造成并行程度不够。

对部分测试视频的解码帧率如图 6.3。这里计算的帧率是同时解码 N 路时的帧率，也就是观看者实际看到的等效帧率。对两路视频而言，要达到 30fps 的等效帧率，必须能够实际解码速率达到 60fps。

从图中可以看到，对于实验目标针对的 720×576 的两路视频，只要线程数超过一个，就已经达到并超过了设定的实验目标。

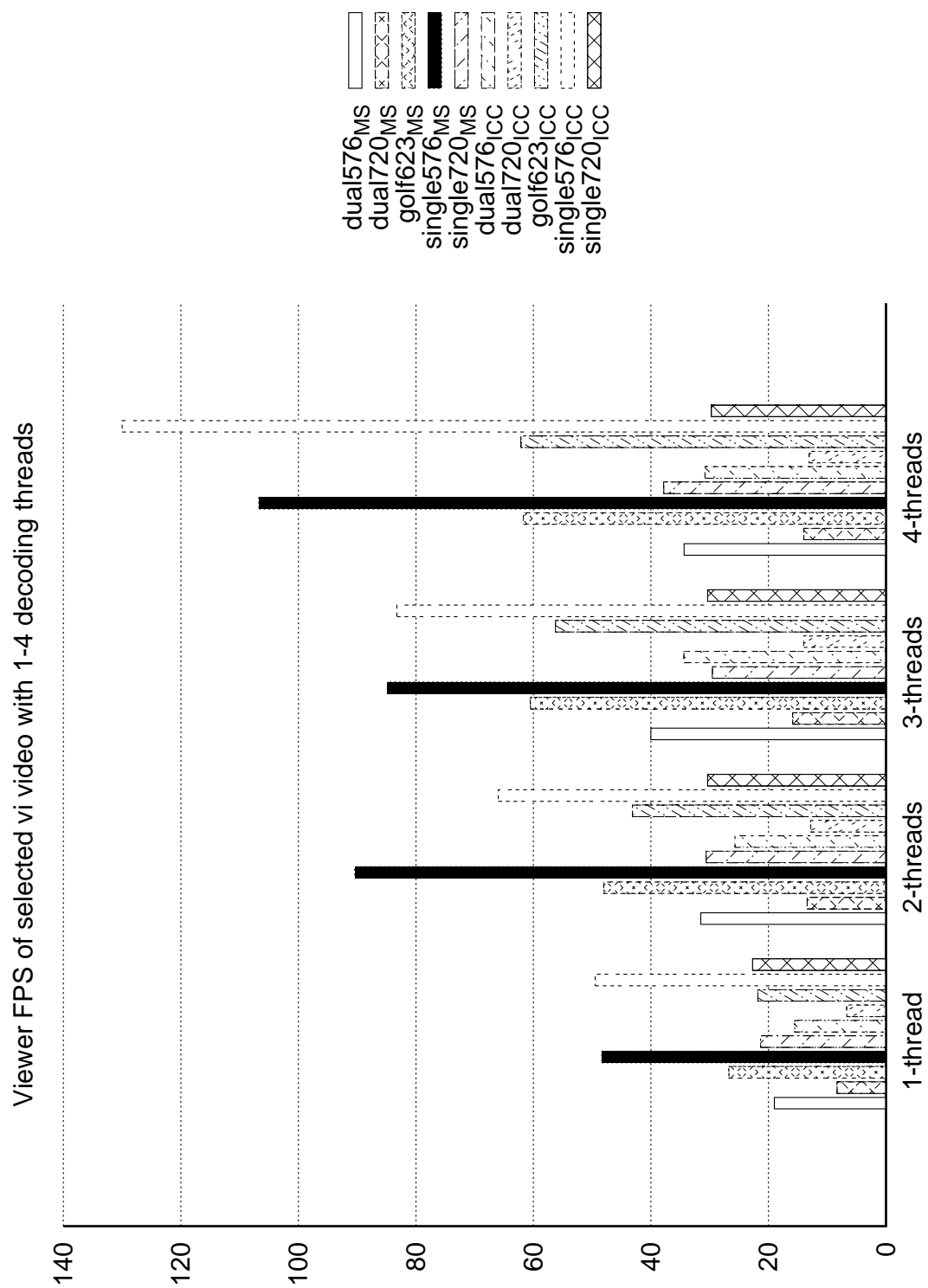


图 6.3 部分视频解码的等效帧率

6.3 小结

本章详细介绍了测试解码器性能的视频准备工作；对比了优化后与优化前的解码器性能，不同的测试视频得到了不同的结果，但是总的趋势是符合逻辑的；数据表明优化后的解码器达到了先前设定的目标。

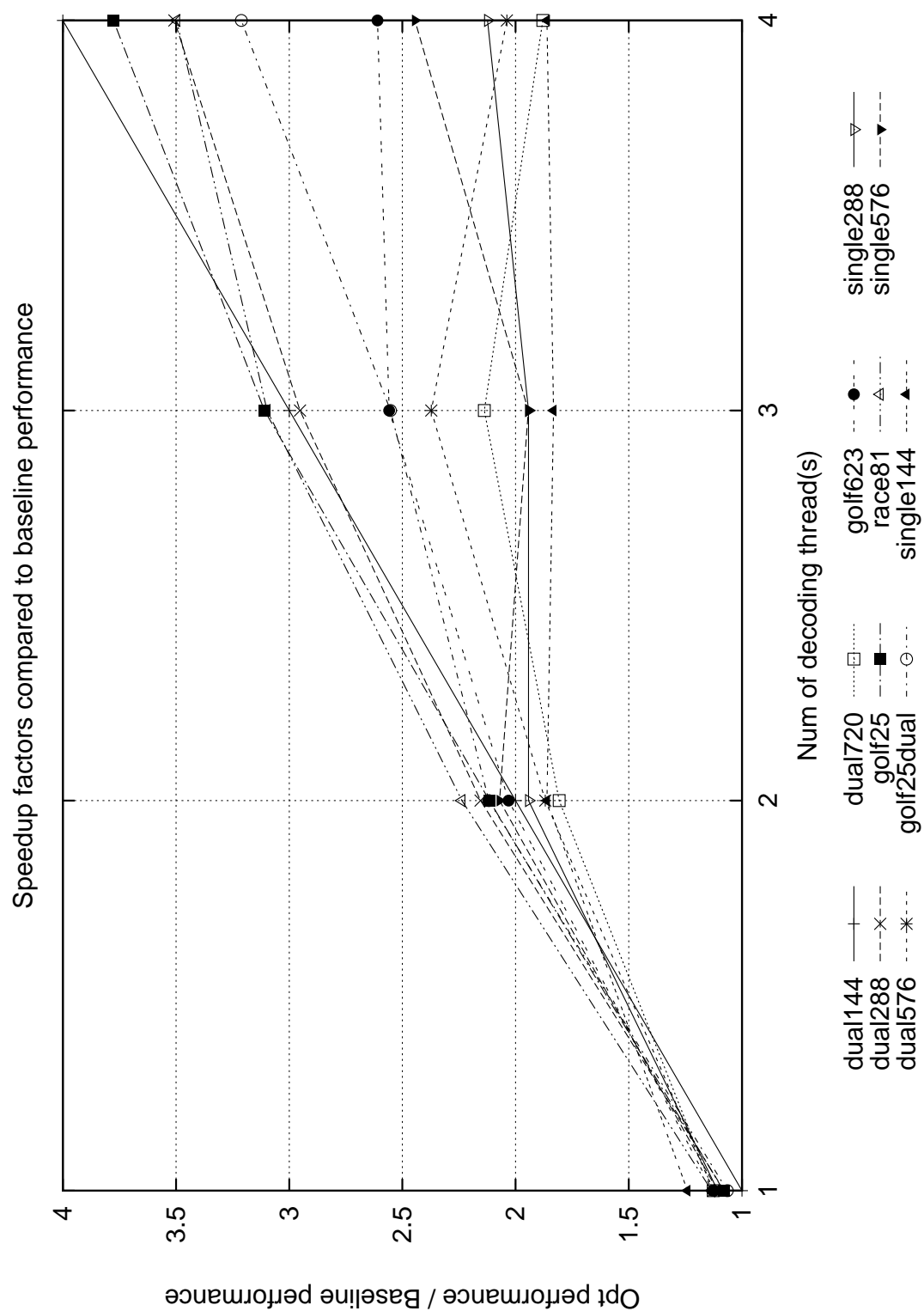


图 6.4 解码不同视频的加速比 (使用 MS 编译器)

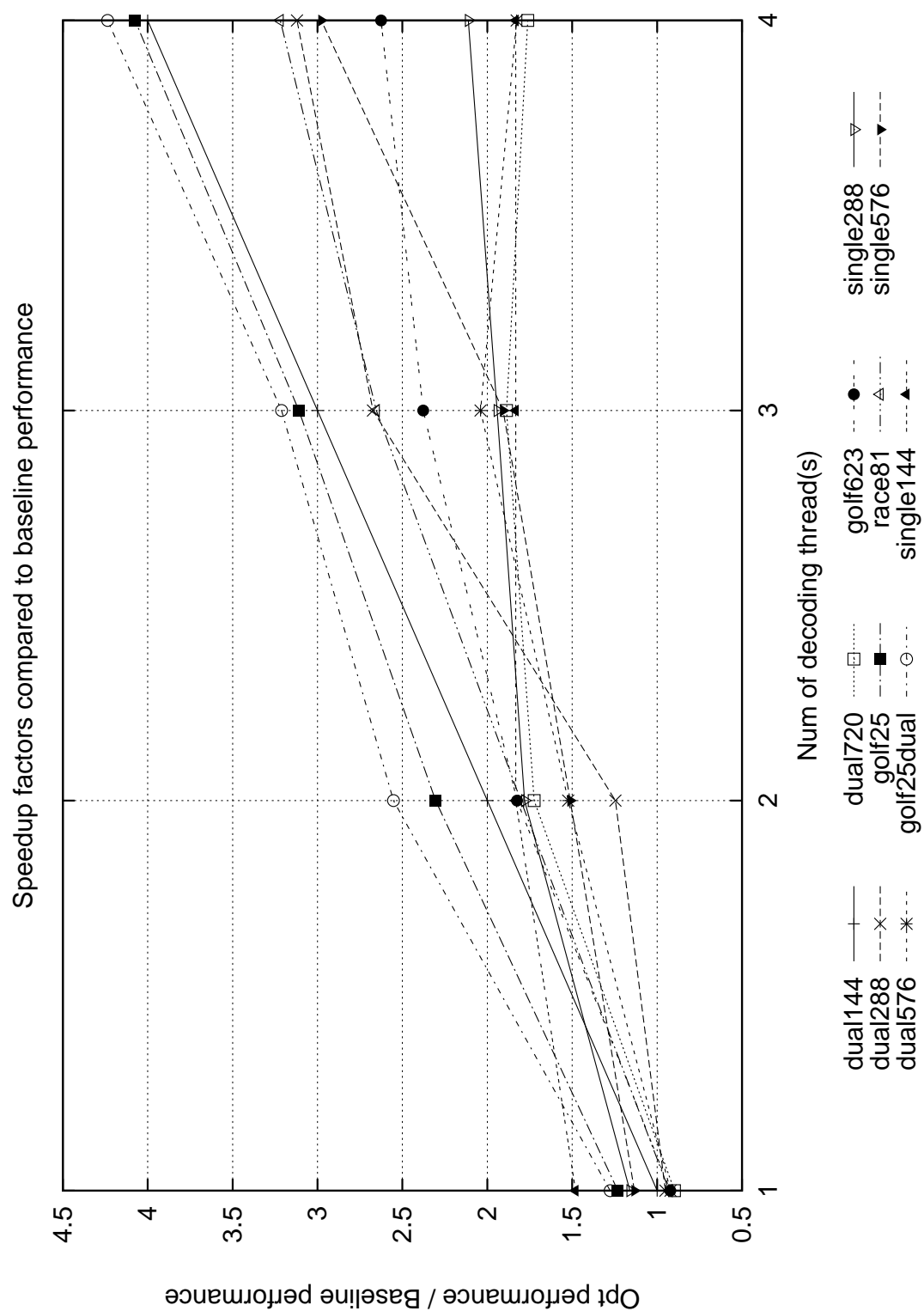


图 6.5 解码不同视频的加速比 (使用 ICC 编译器)

第 7 章 3D 播放器设计与实现

7.1 实验平台

我们希望利用 NVIDIA 3D Vision 套装进行 3D 播放器的立体显示。由于 3D Vision 只支持 Windows Vista 和 Windows 7，所以实验的操作系统有限制。实验主要在以下环境下进行：

- 硬件平台
 - Intel Core2 Quad Q9400 @ 2.66GHz ^①
 - 4GB DDR2/800 Memory
 - NVIDIA GeForce GTS 250 with 512MB
- 软件平台
 - Microsoft Windows 7 Enterprise ^②
 - NVIDIA 3D Vision Driver v1.23
 - DirectX 11.0
 - Microsoft Visual Studio 2008 SP1 ^③

7.2 调研阶段

想要实现 3D 播放器就需要调用硬件资源，至少要能让 3D Vision 的眼镜工作起来。对此，我们进行了一段时间的调研，希望找到一些资料，了解如何调用这些资源。

NVIDIA 的 3D Vision 驱动盘上有一个 3D 播放器程序，可以用来播放 NVIDIA 官方网站上提供下载的一些 3D 视频片段。可惜这个播放器不是开源的，我们猜想其内部是利用了 NVIDIA 提供的 API 函数的，于是去 NVIDIA 的开发站点上下载了 NVAPI^④。在 NVAPI 中，我们找到了和立体显示相关的部分接口函数如代码 7.1：

① SpeedStep 功能关闭

② 获取自：清华大学校园正版软件服务

③ 获取自：清华大学校园正版软件服务

④ http://developer.download.nvidia.com/NVAPI/NVAPI_May2009.zip

代码 7.1 NVAPI 中立体显示的部分函数

```
1 ...
2 NVAPIINTERFACE NvAPI_Stereo_Enable(void);
3 NVAPIINTERFACE NvAPI_Stereo_Disable(void);
4 NVAPIINTERFACE NvAPI_Stereo_CreateHandleFromIUnknown(IUnknown *pDevice,
5     StereoHandle *pStereoHandle);
6 NVAPIINTERFACE NvAPI_Stereo_Activate(StereoHandle stereoHandle);
7 NVAPIINTERFACE NvAPI_Stereo_Deactivate(StereoHandle stereoHandle);
8 NVAPIINTERFACE NvAPI_Stereo_CaptureJpegImage(StereoHandle stereoHandle,
9     NvU32 quality);
10 ...
```

其中的确有 **Stereoscopic3D** 子项，但是只有一些状态查询、保存当前图像等函数，并没有我们所需要的能够实现在显示图像或图形时能控制 3D 眼镜开启的接口。对此，我们认为是 **public** 版本的 **NVAPI** 不完整，没有包含这部分比较新的接口。我们尝试了注册 **NVIDIA** 的注册开发人员^① (**registered developer**)，期望能够获得完整版的 **NVAPI**，但是申请一直没有得到答复。这个方向的调研到此中断。

由于 **3D Vision** 的原理是将左右两个视点的图像交替显示，然后通过同步器控制眼镜快门使得每个时刻都只有一只眼睛能看到为其显示的图像，以此来实现双目立体显示。而在 **NVAPI** 中，我们又发现了名为 **NvAPI_Stereo_Enable()** 和 **NvAPI_Stereo_Activate()** 的两个函数，继而提出了一个问题：如果我们人工地控制程序交替显示两个视点的视频，显式地调用这两个函数（或其中之一），**NVIDIA** 的驱动能否自动地让 3D 眼镜工作？

我们为此写了一个测试程序，可惜结果并不令人满意，我们看到的就是交替渲染的有重影的图像，而 3D 眼镜根本没有工作。

我们一直在 **NVIDIA** 的开发人员论坛以及各类 3D 相关的开发论坛上不断寻找关于 **3D Vision** 调用的尝试。尽管 **NVIDIA** 的开发人员论坛上有十几个活跃的 **thread** 是与此相关的，但是长期以来没有人回复可行的解决方案。经过一段时间的搜索，我们终于在 **mtbs3d.com** 上找到了一个帖子^② 声称成功地调用 **3D Vision** 显示了立体图像。有人根据该贴的说明进行了尝试表示同样获得成功，并给出了一个 **Demo**，显示一些在屏幕上进行 **XYZ** 三个方向上平移的立方体。我们下载了该 **Demo** 程序编译运行证实可行。这个 **Demo** 是基于 **Direct3D** 的，

① http://developer.nvidia.com/page/registered_developer_program.html

② <http://www.mtbs3d.com/phpBB/viewtopic.php?f=7&t=5072>



图 7.1 静态 3D 显示测试的左眼图像



图 7.2 静态 3D 显示测试的右眼图像

其中没有明文调用 NVIDIA 的 API，所以我们认为：NVIDIA 的驱动对 Direct3D 程序能够自动启用 3D 眼镜。

7.3 3D 播放器设计

7.3.1 静态 3D 图像的显示

有了一次成功调用 3D 眼镜的经历，我们找到了一些头绪，并就此开始设计 3D 播放器。

我们希望 3D 播放器能够接收一系列的图像，这些图像分为两个序列，分别是左眼和右眼视角的图像。然后播放器以一个预设的帧率交替播放这两个序列中的图像。如果 NVIDIA 的驱动能够成功开启 3D 眼镜，此时我们通过 3D 眼镜就能看到立体图像了。

经过不断尝试，我们终于实现了一帧静态 3D 图像的显示。做法如下：

1. 首先我们创建一个 `IDirect3DSurface9`，这是 Direct3D 中用于渲染的表面。假设我们要显示的图像的宽度和高度分别为 *imgWidth* 和 *imgHeight*。这个表面的宽度就应该是 $2 \times \text{imgWidth}$ ，这一点较容易理解，因为有左右两个视角的图像。但是这个表面的高度必须设为 *imgHeight* + 1，这个多出来的 1 像素是上文提及的帖子中指出的关键点之一，算是一个 **magic number**。实验中我们把高度设为 *imgHeight* 得到的就不是立体图，而是两幅横向并列显示的拉伸过的图像。
2. 有了这个表面之后，我们需要为其添加内容。以左上角 (0,0) 为原点，调用 `D3DXLoadSurfaceFromFile()` 读入左眼图像（图 7.1），再以 (*imgWidth*,0)

为原点再次调用函数读入右眼图像（图 7.2）。

3. 在渲染之前，我们还要为这个待显示的结构写一个头，叫做 *LPNVSTEREOIMAGEHEADER*，其中要指定宽度、高度、颜色深度以及一个 signature，这个名为 *NVSTEREO_IMAGE_SIGNATURE* 的 signature 需要赋一个特定的值 0x4433564e。见代码 7.2。

代码 7.2 LPNVSTEREOIMAGEHEADER 的结构

```
1 // Stereo Blitdefines
2 #define NVSTEREO_IMAGE_SIGNATURE 0x4433564e //NV3D
3 typedef struct _Nv_Stereo_Image_Header {
4     unsigned int dwSignature;
5     unsigned int dwWidth;
6     unsigned int dwHeight;
7     unsigned int dwBPP;
8     unsigned int dwFlags;
9 } NVSTEREOIMAGEHEADER, *LPNVSTEREOIMAGEHEADER;
```

4. 经过如上的操作，利用 Direct3D 的 renderer 渲染输出到显示器的图像就是立体图了 (图 7.3)。注意实际透过眼镜是看不到重影的，两帧图像交替显示，眼镜快门相应地轮流遮挡视线，只能看到各自视角的图像。

实现了一帧静态立体图的显示，就有了显示立体视频的基础，显示动态图像的部分在下一节阐述。

7.3.2 立体视频（图像序列）的显示

视频的显示可以看做以一个预设的帧率 FPS（frames per second）显示一个图像序列中的一张张图像。这里的帧率根据视频的制式不同而改变，一般都在 24 或 30 左右。

视频中的 FPS 与游戏中的 FPS 有所不同。在游戏中，往往追求硬件所能处理的最高的 FPS，尽可能使画面更加流畅。而在视频回放的时候，有一个固定的帧率，必须要尽可能让实际帧率接近这个预设的帧率，才能满足视频的每一帧出现在时间轴的正确位置上。然而，视频播放和游戏的渲染又有一个共同点，当硬件资源很有限的情况下，会将帧率降到很低，出现跳帧，这也是为了保证画面的时间正确性，避免出现像“慢动作回放”一样的效果。



图 7.3 渲染后输出的 3D 图像

综合以上的特点，我们就需要有一个控制帧率上限的机制，保证帧率在资源足够的时候不会超过预设值；同时要支持在资源受限时自动跳帧。

为了实现这样的机制，我们需要获得较为精确的时间，来控制每一帧持续的时长。ANSI C++ 中提供的 `getTicker()` 函数是 ms 级的，比较粗糙，无法满足一些常见的（如 23.97）标准帧率的控制需要。对此，我们查阅了 MSDN 的文档后选用了 Windows API 中提供的 `QueryPerformanceCounter()` 和 `QueryPerformanceFrequency()` 配合实现 us 级的计时器。

我们实现的帧率上限控制的机制如下：在开始播放第 0 帧的时刻记下当前时间 t_0 。在渲染完第 i 帧 F_i 之后，获得当前时间 t_n ，然后准备进行第 $(t_n - t_0) \times FPS$ 的渲染。通过比较当前已渲染的帧 F_i 和待渲染的帧 F_n 是否一致来决定是否进行渲染操作。这个机制的核心就是式 7-1。

$$F_n = (t_n - t_0) \times FPS \quad (7-1)$$

其中 FPS 为根据播放内容而预设的一个帧率， t_n 为当前时刻， t_0 为播放视频第 0 帧的起始时刻， F_n 为当前时刻应该渲染的帧的序号。

帧率上限控制算法的伪代码见Algorithm 1。

Algorithm 1 Frame rate cap

```
1:  $\_start \leftarrow initialTime$ 
2:  $bufferd = FALSE$ 
3:  $lastRenderedFrame \leftarrow -1$ 
4: while  $TRUE$  do
5:    $\_current \leftarrow currentTime - \_start$ 
6:    $frameToRender \leftarrow \_current * FPS$ 
7:   if  $frameToRender + 1 > totalFrames$  then
8:     break {break when reaching the last frame}
9:   end if
10:  if  $bufferd = FALSE$  then
11:    load  $Frame_{frameToRender+1}$  to memory
12:     $bufferd \leftarrow TRUE$ 
13:  end if
14:  if  $frameToRender \neq lastRenderedFrame$  then
15:    render  $Frame_{frameToRender}$ 
16:     $lastRendered \leftarrow frameToRender$ 
17:     $bufferd \leftarrow FALSE$ 
18:  end if
19: end while
```

7.4 播放器测试

我们实现的播放器具有如下功能：

输入 左右两个视点的图像序列文件名格式。实验中采用了“前缀+序号+后缀”的方式。前缀为文件相对路径，图像的文件名包含其对应的帧序号，后缀为文件扩展名。

图像序列的总帧数 $totalFrames$ ，见Algorithm 1的第 7 行。

播放时的帧率 FPS ，见Algorithm 1的第 6 行。

输出 在显示器上输出如图 7.3 效果的立体图序列，帧率为 *FPS*。通过 3D 眼镜可以看到立体视频。

我们使用 720×576 的分辨率对播放器进行测试，选用了几段不同的图像序列。在不断测试过程中，我们发现在图像包含告诉运动的物体时能够看出跳帧现象。于是我们输出了每一帧的 **load** 和 **render** 情况，发现实际帧率在 19fps 到 21fps 不等。我们希望寻找性能瓶颈所在，进行了以下排查：

- 我们首先考虑是否因为图像序列的格式为 JPEG，在读取的时候需要时间解码。于是将图像序列转化为 BMP 格式的位图，测试发现帧率进一步降至 14fps 左右。测试用图存为 JPEG 时每帧平均大小 50KB，BMP 则超过 300KB。
- 有了 JPEG 和 BMP 的对比，我们认为磁盘 I/O 可能是一个瓶颈。对此，我们划分了 500MB 的内存作为一个 RAMDISK。用 HD Tune 测试这个分区的读写速度和随机访问时间等指标如下：

Transfer Rate (Min) 2751.7MB/sec

Transfer Rate (Max) 3363.2MB/sec

Transfer Rate (Avg) 3239.8MB/sec

Access Time (Avg) 0.0ms

Burst Rate 3670.7MB/sec

CPU Usage 0.0%

将图像序列存放在该 RAMDISK 分区上再次进行测试，帧率依然在 21fps 左右。所以磁盘 IO 被排除在瓶颈之外。

- 此时我们决定用性能分析工具分析我们的程序，找出其中时间耗费最多的部分，将该部分打散成多个小的函数再次分析。如此循环了几次，终于找到性能瓶颈所在：我们调用 D3D 中的 `D3DXLoadSurfaceFromFile()` 函数两次，分别将左右眼的图像绘制到渲染的表面上。而这两个函数调用占用了整个程序运行总时间的 95% 以上。

这个 API 的性能我们暂时未解决，尚在寻找可行的替代方法。

7.5 小结

本章介绍了 3D 播放器设计实现的全部过程，从调研阶段遇到的各种问题、种种尝试，到发现一种可能的解决方案，然后经过一些尝试，最终借助

Direct3D 实现了支持 3D Vision 套装的立体视频播放器。不过依然存在一定的性能瓶颈，有待进一步解决。

第 8 章 结论和展望

8.1 本文工作总结

视频编解码的并行在近几年 CPU 向多核方向发展之后成为一个热门的研究领域。最近成为标准的多视点视频由于其数据量庞大、解码运算复杂，很难依靠单核的处理器达到实用的解码速度。多核并行解码多视点视频势在必行。我们以现有的多视点视频解码器为基础，希望通过一定的优化和并行处理，使普通的 PC 能够进行立体视频的实时解码。

总体来说，目前达到了实验初期预定的性能优化目标。通过对解码器函数级的优化，包括函数逻辑、减少循环内部的计算量、使用汇编实现部分函数，以及一个可以稳定运行的并行解码框架的实现，最终使得多视点视频的解码可以在主流 PC 上实现两路标清的实时解码。

在解码器达到性能要求之后，我们着手设计实现了一个基于 NVIDIA 3D Vision 套装的立体视频播放器，播放两路立体视频时能够通过 3D 眼镜看到立体效果。

8.2 存在的问题

目前的解码器主要存在以下问题：

1. 多线程解码视频时的加速比不够稳定。对于部分视频会出现增加线程反而降低解码性能的情形。
2. 对 MVC 标准的支持尚不完整。
3. 对一些裸眼立体显示设备要求的八路视频解码性能还达不到实时。

3D 播放器主要存在以下问题：

1. 播放的图像序列以磁盘文件而非内存中一段数据的形式存在，将来可能成为性能瓶颈。
2. 播放时的帧率达不到流畅观看的要求，这主要是使用的一个 D3D API 函数调用造成的瓶颈。

8.3 未来工作

针对前文提出的问题，解码器在将来还需要进行以下工作：

1. 对于加速比反常下降的视频，我们希望实现一个智能的判断机制，在不超过线程上限的范围内，自动使用性能最优的线程数进行解码，避免性能损失。
2. 增加对 MVC 标准中 Main Profile 和 Stereo High Profile 的支持，增强解码器的通用性。
3. 借助 GPU 进行部分计算，实现两路高清视频的实时解码和八路视频的实时解码。

3D 播放器尚需要实现以下改进：

1. 自行实现对渲染表面的填充函数，替换 D3D API 调用，突破现有的性能瓶颈，保证播放的画面流畅。
2. 对播放器的输入进行修改，使其满足一个自定义的接口，方便将播放器和解码器的输出对接，做到实时解码并播放。

插图索引

图 1.1	MPEG、VCEG、JVT 视频编解码标准的发展	2
图 1.2	MVC 的时间和视角间预测结构	3
图 1.3	包含调度的 MVC 编解码框架	4
图 2.1	JMVC 编解码框架	8
图 2.2	原始图像 (左上) 和其 Y' (左下)、U (右上)、V (右下) 分量	9
图 2.3	MVC Encoder 流程图	9
图 2.4	MVC Decoder 流程图	10
图 5.1	典型的 MVC 参考结构	23
图 5.2	GOP 预测结构	24
图 5.3	多视点视频中的 GOP 调度	25
图 5.4	宏块参考结构	25
图 5.5	解码器模块关系	26
图 6.1	解码不同视频的加速比 (使用 MS 编译器)	32
图 6.2	解码不同视频的加速比 (使用 ICC 编译器)	33
图 6.3	部分视频解码的等效帧率	34
图 6.4	解码不同视频的加速比 (使用 MS 编译器)	36
图 6.5	解码不同视频的加速比 (使用 ICC 编译器)	37
图 7.1	静态 3D 显示测试的左眼图像	41
图 7.2	静态 3D 显示测试的右眼图像	41
图 7.3	渲染后输出的 3D 图像	43
图 A.1	H.262/MPEG-2 视频多视点类 (multiview profile) 的预测示意	63
图 A.2	MVC 的时间和视角间预测结构	66

表格索引

表 3.1	VS2008 性能分析报告 (前 10 项记录)	13
表 3.2	VTune 性能分析报告 (前 10 项记录)	14
表 6.1	性能测试中使用的所有视频一览	31
表 B.1	VS2008 性能分析报告 (前 50 项记录)	69
表 B.2	VTune 性能分析报告 (前 50 项记录)	70
表 B.3	不同视频解码速度一览 (基准性能)	72
表 B.4	不同视频解码速度一览 (MS 编译器)	72
表 B.5	不同视频解码速度一览 (ICC 编译器)	74

代码索引

4.1	macroblockGetHalfPel() 函数片段（优化前）	17
4.2	macroblockGetHalfPel() 函数片段（优化后）	18
4.3	macroblockPredGetDataY() 函数片段（优化前）	18
4.4	macroblockPredGetDataY() 函数片段（优化后）	19
4.5	macroblockGetPred_axb() 函数片段（优化前）	19
4.6	macroblockGetPred_axb() 函数片段（优化后）	20
7.1	NVAPI 中立体显示的部分函数	40
7.2	LPNVSTEREOIMAGEHEADER 的结构	42
C.1	JMVC 编码八路视频配置文件样例	76
C.2	JMVC 打包八路视频配置文件样例	78
C.3	JMVC 打包八路视频脚本样例	79

参考文献

- [1] Pang Y, Sun L, Wen J, et al. A framework for heuristic scheduling for parallel processing on multicore architecture: a case study with multiview video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 2009, 19(11):1658–1666
- [2] Smolic A, Müeller K, Stefanoski N, et al. Coding algorithms for 3DTV—a survey. *IEEE transactions on circuits and systems for video technology*, 2007, 17(11):1606–1620
- [3] Merkle P, Müeller K, Smolic A, et al. Statistical evaluation of spatiotemporal prediction for multiview video coding. *Proc. ICOT 2005*, 2005. 27–28
- [4] Kaup A, Fecker U. Analysis of multi-reference block matching for multi-view video coding. *Proceedings of Proc. 7th Workshop Digital Broadcasting*. Citeseer, 2006. 33–39
- [5] Magnor M, Ramanathan P, Girod B, et al. Multi-view coding for image-based rendering using 3-D scene geometry. *IEEE Transactions on circuits and systems for video technology*, 2003, 13(11):1092–1106
- [6] Flierl M, Mavlankar A, Girod B. Motion and Disparity Compensated Coding for Multiview Video. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 2007, 17(11)
- [7] Merkle P, Müeller K, Smolic A, et al. Efficient compression of multi-view video exploiting inter-view dependencies based on H. 264/MPEG4-AVC. *Proceedings of 2006 IEEE International Conference on Multimedia and Expo*, 2006. 1717–1720
- [8] Müeller K, Merkle P, Schwarz H, et al. Multi-view video coding based on H. 264/AVC using hierarchical B-frames. *Proceedings of Picture Coding Symposium*, 2006
- [9] Merkle P, Smolic A, Müeller K, et al. Efficient prediction structures for multiview video coding. *IEEE Transactions on circuits and systems for video technology*, 2007, 17(11):1461–1473
- [10] Li P, Veeravalli B, Kassim A. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 2005, 15(9):1098–1112
- [11] Seitner F, Schreier R, Bleyer M, et al. A macroblock-level analysis on the dynamic behaviour of an H. 264 decoder. *Proceedings of IEEE International Symposium on Consumer Electronics*, 2007. ISCE 2007, 2007. 1–5
- [12] Mesa M, Ramírez A, Azevedo A, et al. Scalability of Macroblock-level Parallelism for H. 264 Decoding. *Proceedings of 2009 15th International Conference on Parallel and Distributed Systems*. IEEE, 2009. 236–243

- [13] Yang Y, Jiang G, Yu M, et al. Parallel process of hyper-space-based multiview video compression. Proceedings of 2006 IEEE International Conference on Image Processing, 2006. 521–524
- [14] Ugur K, Liu H, Lainema J, et al. Parallel encoding-decoding operation for multiview video coding with high coding efficiency. Proceedings of 3DTV Conference, 2007, 2007. 1–4
- [15] JVT. Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding. ISO/IEC 14496-10:2009, 2009
- [16] Richardson I. H. 264 and MPEG-4 video compression: video coding for next-generation multimedia. The Robert Gordon University, Aberdeen, UK: John Wiley & Sons Inc, 2003
- [17] Pang Y, Hu W, Sun L, et al. Adaptive data-driven parallelization of multi-view video coding on multi-core processor. Science in China Series F: Information Sciences, 2009, 52(2):195–205
- [18] Yi P, Weidong H, Lifeng S, et al. Parallelized multiview video coding on cell broadband engine blade. Proceedings of The Proceedings of the Workshop on Cell Systems and Applications (ISCA WCSA), 2008. 82–89
- [19] Levinthal D. OS Performance Analysis on Intel® Core™2 Duo Processors. Technical report, Intel, 2007. http://softwarecommunity.intel.com/isn/downloads/softwareproducts/pdfs/os_performance.pdf
- [20] Levinthal D. Introduction to Performance Analysis on Intel® Core™ 2 Duo Processors. Technical report, Intel, 2006. http://softwarecommunity.intel.com/isn/downloads/softwareproducts/pdfs/performance_analysis.pdf
- [21] Levinthal D. Analyzing and Resolving multi-core non scaling on Intel® Core™2 processors. Technical report, Intel, 2008. http://softwarecommunity.intel.com/isn/downloads/softwareproducts/pdfs/non_scaling.pdf
- [22] Levinthal D. Cycle Accounting Analysis on Intel® Core™2 Processors. Technical report, Intel, 2008. http://softwarecommunity.intel.com/isn/downloads/softwareproducts/pdfs/cycle_accounting.pdf
- [23] A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMXTM Instructions. Application Note AP-922 Order Number: 742474-001, Intel Corp., 1999
- [24] Using Streaming SIMD Extensions (SSE2) to Implement an Inverse Discrete Cosine Transform. Application Note AP-945 Order Number: 248670-001, Intel Corp., 2000
- [25] Meenderinck C, Azevedo A, Juurlink B, et al. Parallel scalability of video decoders. Journal of Signal Processing Systems, 2009, 57(2):173–194
- [26] Kotz D. L^AT_EX and the Gnuplot Plotting Program. GNUPLOT L^AT_EX Tutorial Version, 1991, 3
- [27] Williams T, Kelley C, et al. gnuplot 4.4: An Interactive Plotting Program. User Manual, 2009.

- [28] Smolic A, Mueller K, Merkle P, et al. 3D video and free viewpoint video-technologies, applications and MPEG standards. Proceedings of 2006 IEEE International Conference on Multimedia and Expo, 2006. 2161–2164
- [29] Onural L, Sikora T, Smolic A. An overview of a new european consortium: Integrated three-dimensional television-capture, transmission and display (3DTV). Proceedings of Proceedings of European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology (EWIMT). Citeseer, 2004
- [30] Smolic A, McCutchen D. 3DAV exploration of video-based rendering technology in MPEG. IEEE transactions on circuits and systems for video technology, 2004, 14(3):348–356
- [31] Smolic A, Kauff P. Interactive 3-D video representation and coding technologies. Proceedings of the IEEE, 2005, 93(1):98–110
- [32] Onural L, Sikora T, Ostermann J, et al. An assessment of 3DTV technologies. Proceedings of Proc. of NAB Broadcast Engineering Conf, 2006. 456–467
- [33] Alatan A, Yemez Y, Gudukbay U, et al. Scene representation technologies for 3DTV—a survey. IEEE transactions on circuits and systems for video technology, 2007, 17(11):1587–1605
- [34] Tanimoto M. Free viewpoint television—FTV. Proceedings of Proc. PCS, 2004
- [35] Fujii T, Tanimoto M. Free viewpoint TV system based on ray-space representation. Proceedings of Proceedings of SPIE, volume 4864, 2002. 175
- [36] Matsuyama T, Wu X, Takai T, et al. Real-time dynamic 3-D object shape reconstruction and high-fidelity texture mapping for 3-D video. IEEE Transactions on Circuits and Systems for Video Technology, 2004, 14(3):357–369
- [37] Hamaguchi T, Fujii T, Honda T. Real-Time View-Interpolation System for Super Multi-View 3D Display. IEICE TRANSACTIONS on Information and Systems, 2003, 86(1):109–116
- [38] Carranza J, Theobalt C, Magnor M, et al. Free-viewpoint video of human actors. Proceedings of ACM SIGGRAPH 2003 Papers. ACM, 2003. 577
- [39] Würmlin, S and Lamoray, E and Staadt OG and Gross MH. 3D video recorder: A system for recording and playing three-viewpoint video. Proceedings of Computer Graphics Forum, volume 22. Citeseer, 2003. 181–193
- [40] Würmlin S, Lamoray E, Gross M. 3D video fragments: Dynamic point samples for real-time free-viewpoint video. Computers & Graphics, 2004, 28(1):3–14
- [41] Matusik W, Pfister H. 3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. ACM Transactions on Graphics (TOG), 2004, 23(3):814–824
- [42] Shum H, Kang S, Chan S. Survey of image-based representations and compression techniques. IEEE transactions on circuits and systems for video technology, 2003, 13(11):1020–1037

- [43] Haskell B, Puri A, Netravali A. Digital video: an introduction to MPEG-2. Kluwer Academic Publishers, 1997
- [44] Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video. ITU-T Rec. H.222.0|ISO/IEC 13818-1 (MPEG 2 Systems), ITU-T and ISO/IEC JTC 1, 1994.
- [45] Video Coding for Low Bit Rate Communication. ITU-T Rec. H.263, version 1, Nov. 1995; version 2 Jan. 1998; version 3, Nov. 2000.
- [46] Coding of audio-visual objects-Part 2: Visual. ISO/IEC 14496-2 (MPEG-4 visual version 1), Apr. 1999; ISO/IEC JTC 1, Amd. 1 (ver. 2), Feb. 2000; Amd. 2, 2001; Amd. 3, 2001; Amd. 4 (streaming video profile), 2001; Amd 1 to 2nd ed. (studio profile), 2001; Amd. 2 to 2nd ed. 2003.
- [47] Sun S, Lei S. Stereo-view video coding using H. 264 tools. Proceedings of Proceedings of SPIE, volume 5685, 2005. 177
- [48] Wiegand T, Sullivan G, Bjontegaard G, et al. Overview of the H. 264/AVC video coding standard. IEEE Transactions on circuits and systems for video technology, 2003, 13(7):560–576
- [49] Advanced Video Coding for Generic Audiovisual Services 2005. ITU T and ISO/IEC JTC 1, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, 2003.
- [50] Fehn C, Kauff P, De Beeck M, et al. An evolutionary and optimised approach on 3D-TV. Proceedings of Proc. of IBC. Citeseer, 2002. 357–365
- [51] Fehn C. 3D-TV using depth-image-based rendering (DIBR). Proceedings of Proceedings of Picture Coding Symposium (PCS), 2004. 307–312
- [52] Text of ISO/IEC FDIS 23002-3 Representation of Auxiliary Video and Supplemental Information. ISO/IEC JTC1/SC29/WG11, 2007, Doc. N8768, Marrakesh, Morocco
- [53] Text of ISO/IEC 13818-1:2003/FDAM2 Carriage of Auxiliary Data. ISO/IEC JTC1/SC29/WG11, 2007, Doc. N8799, Marrakech, Morocco
- [54] Shade J, Gortler S, He L, et al. Layered depth images. Proceedings of Proceedings of the 25th annual conference on Computer graphics and interactive techniques. ACM, 1998. 242
- [55] Zitnick C, Kang S, Uyttendaele M, et al. High-quality video view interpolation using a layered representation. Proceedings of ACM SIGGRAPH 2004 Papers. ACM, 2004. 600–608
- [56] Kauff P, Atzpadin N, Fehn C, et al. Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability. Signal Processing: Image Communication, 2007, 22(2):217–234
- [57] Magnor M, Girod B. Data compression for light-field rendering. IEEE Transactions on Circuits and Systems for Video Technology, 2000, 10(3):338–343
- [58] Oh K J, Ho Y S. Multi-view video coding based on the lattice-like pyramid GOP structure. Proceedings of Proc. PCS 2006, Picture Coding Symp., Beijing, China, 2007
- [59] Cheng X, Sun L, Yang S. A Multi-View Video Coding scheme using shared key frames for high interactive application. Proceedings of Proc. PCS 2006, Picture Coding Symp., Beijing, China, 2006

- [60] Yang Y, Jiang G, Yu M, et al. Hyper-space based multiview video coding scheme for free viewpoint television. Proceedings of Proc. PCS 2006, Picture Coding Symp., Beijing, China, 2006
- [61] Vetro A, Matusik W, Pfister H, et al. Coding approaches for end-to-end 3D TV systems. Proceedings of Picture Coding Symposium. Citeseer, 2004
- [62] Kalva H, Christodoulou L, Mayron L, et al. Design and evaluation of a 3D video system based on H. 264 view coding. Proceedings of Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video. ACM, 2006. 12
- [63] Socek D, Culibrk D, Kalva H, et al. Permutation-Based Low-Complexity Alternate Coding in Multi-View H. 264/AVC. Proceedings of 2006 IEEE International Conference on Multimedia and Expo, 2006. 2141–2144
- [64] Kalva H, Christodoulou L, Mayron L, et al. Challenges and opportunities in video coding for 3D TV. Proceedings of 2006 IEEE International Conference on Multimedia and Expo, 2006. 1689–1692
- [65] Bilen C, Aksay A, Akar G. A multi-view video codec based on H. 264. Proceedings of 2006 IEEE International Conference on Image Processing, 2006. 541–544
- [66] Schwarz H, Marpe D, Wiegand T. Analysis of hierarchical B pictures and MCTF. Proceedings of 2006 IEEE International Conference on Multimedia and Expo, 2006. 1929–1932
- [67] Merkle P, Smolic A, Müller K, et al. Coding efficiency and complexity analysis of mvc prediction structures. EURASIP 2007, 2007.
- [68] Martinian E, Behrens A, Xin J, et al. View synthesis for multiview video compression. Proceedings of Proc. Picture Coding Symp. Citeseer, 2006
- [69] Kitahara M, Kimata H, Shimizu S, et al. Multi-view video coding using view interpolation and reference picture selection. Proceedings of 2006 IEEE International Conference on Multimedia and Expo, 2006. 97–100
- [70] Martinian E, Behrens A, Xin J, et al. Extensions of H. 264/AVC for multiview video compression. Proceedings of 2006 IEEE International Conference on Image Processing, 2006. 2981–2984
- [71] Kim J, Lai P, Ortega A, et al. Results of CE2 on multi-view video coding. Joint Video Team, Doc. JVT-T117, Klagenfurt, Austria, 2006.
- [72] Lee Y, Hur J, Lee Y, et al. Results of CE2 on Multi-view Video Coding. Joint Video Team, Doc. JVT-T110, Klagenfurt, Austria, 2006.
- [73] Fecker U, Barkowsky M, Kaup A. Improving the prediction efficiency for multi-view video coding using histogram matching. Proceedings of Picture Coding Symposium (PCS 2006). Citeseer, 2006
- [74] Lu J, Cai H, Lou J, et al. An effective epipolar geometry assisted motion-estimation technique for multi-view image coding. Relation, 2009, 10(1.68):8331

- [75] Guo X, Lu Y, Wu F, et al. Inter-view direct mode for multiview video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 2006, 16(12):1527–1532
- [76] Garbas J, Fecker U, Troger T, et al. 4D scalable multi-view video coding using disparity compensated view filtering and motion compensated temporal filtering. *Proceedings of 2006 IEEE 8th Workshop on Multimedia Signal Processing*, 2006. 54–58
- [77] Drose M, Clemens C, Sikora T. Extending single-view scalable video coding to multi-view based on H. 264/AVC. *Proceedings of 2006 IEEE International Conference on Image Processing*, 2006. 2977–2980
- [78] Yang W, Lu Y, Wu F, et al. 4-D wavelet-based multiview video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 2006, 16(11):1385–1396
- [79] Yang W, Wu F, Lu Y, et al. Scalable multiview video coding using wavelet. *Proceedings of IEEE International Symposium on Circuits and Systems*, 2005. ISCAS 2005, 2005. 6078–6081
- [80] Ozbek N, Tekalp A. Scalable multi-view video coding for interactive 3DTV. *Proceedings of 2006 IEEE International Conference on Multimedia and Expo*, 2006. 213–216
- [81] Guo X, Lu Y, Wu F, et al. Distributed multi-view video coding. *Proceedings of Proceedings of SPIE*, volume 6077. Citeseer, 2006. 290–297
- [82] Kurutepe E, Civanlar M, Tekalp A. Interactive Multi-view Video Delivery with View-Point Tracking and Fast Stream Switching. *Multimedia Content Representation, Classification and Security*, 2006. 586–593
- [83] Kurutepe E, Civanlar M, Tekalp A. Interactive transport of multi-view videos for 3DTV applications. *Journal of Zhejiang University-Science A*, 2006, 7(5):830–836
- [84] Vetro A, Su Y, Kimata H, et al. Joint draft 1.0 on multiview video coding. Joint Video Team, 2006, Doc. JVT-U209
- [85] Vetro A, Su Y, Kimata H, et al. Joint multiview video model. ITU-T and ISO/IEC Joint Video Team, Document JVT-U207, Nov, 2006.
- [86] Merkle P, Smolic A, Muller K, et al. Efficient compression of multi-view depth data based on MVC. *Proceedings of 3DTV Conference*, 2007, 2007. 1–4
- [87] Merkle P, Smolic A, Müller K, et al. Multi-view video plus depth representation and coding. *Proceedings of Proceedings of ICIP*, 2007. 201–204
- [88] Saishu T, Numazaki S, Taira K, et al. Flatbed-type autostereoscopic display system and its image format for encoding. *Proceedings of Proceedings of SPIE*, volume 6055, 2006. 60550S

致 谢

衷心感谢导师孙立峰副教授和杨士强教授在毕业设计过程中对本人的精心指导。他们在研究方向和方法上给了我很多建议和意见。

在整个实验过程中，胡伟栋师兄在我遇到困难的时候一直给予帮助，对我完成实验有巨大帮助，在此对他表示感谢！

论文写作过程中用到了 \LaTeX 和很多提供扩展功能的宏包，在此对这些开源软件的作者表示感谢。感谢清华大学高性能所薛瑞尼编写的 `ThuThesis`，让我的论文写作避免了不少繁琐的格式调整，同时让我的论文格式规整了许多。论文中的图由 `gnuplot` 绘制，比起很多商业软件更加方便，图形也更美观，在此感谢 `gnuplot` 的开发人员。

声 明

本人郑重声明：所提交的学位论文，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

A.1 摘要

最近, 3D 电视技术相关的研究工作在全球范围内都有增加。这些研究覆盖了整个媒体处理流程, 从采集到显示无所不包。不同的 3D 电视系统依靠各种类型数据构成的不同的 3D 场景表示方式。对这些数据的高效编码对于 3D 电视的成功至关重要。对双目视频、多视点视频、深度图或视差图中像素类型数据的压缩拓展了已有的经典视频编码的原则。针对多视点视频和视频加深度数据的强大算法和开放的国际标准已经出现并一直有着进展, 这为在不久的将来引进各种 3D 电视系统奠定了基础。对 3D 网格模型的压缩也达到了高度成熟的程度。对静态的几何体, 有多种强大的算法可以有效地压缩顶点和连接数据量。对动态 3D 几何体的压缩是现在更受关注的研究方向。基于时间的预测是一个重要的对动画 3D 网格序列去冗余信息的机制。容错性对于在易出错的信道上传输数据十分重要, 而多描述编码 (MDC) 是一种可行的保护数据的方法。对静态图和 2D 视频的 MDC 已经得到了广泛的研究, 但是对多视点视频和 3D 网格的 MDC 最近才刚刚提出。用水印进行 3D 数据的知识产权保护也是一个前沿的研究方向。3D 水印方法方面的文献根据表示场景的主要部分在应用算法前后的维数可以归为三类。总而言之, 3D 电视编码技术正在走向成熟。相关系统和服务可能在不久的将来进入市场。然而, 这个方向的研究相对其他媒体数据的研究而言又很少, 所以仍有很大的空间可以改进, 有很多新的算法可能被提出。

A.2 简介

将我们的视觉延伸到三维空间在近几十年中一直都在研究, 但是大规模的消费市场仍然没有发展起来。3D 视频还在小众市场发展, 包括专业应用 (比如科学数据可视化、医药学等) 和娱乐方面 (比如 IMAX 影院、3D 游戏等)。近年来, 更多的研究精力投向了 this 领域, 以期能够跨越阻碍 3D 视频应用得到更广泛成功应用的障碍^[28-32]。从获得信号、处理信号到 3D 场景表示、编码、传输, 再到渲染和 3D 显示, 整个处理流水线的各个部分都取得了显著的进步。很可能在不久的将来, 各种 3D 视频系统、组件、应用和服务就会进入市场。

这一期中给出了各种尖端 3D 视频技术的综述，分别针对一个 3D 视频处理流水线中的一个特定组件。这篇文章介绍编码相关的部分。[33] 中介绍了用在不同的 3D 视频系统和应用中的不同 3D 场景表示格式在。这些格式整合了各种类型的数据，比如多视点视频、深度或 3D 网格的几何数据。总的来说，3D 视频就意味着有大量的数据需要传输、存储和水印。于是高效的压缩就是 3D 视频成功与否的一个关键条件。同时对发明一个鲁棒的 3D 水印技术来保护所有权也有很强的需求。此外，开放的国际标准对市场和相关商业的发展也是一个推动力。ISO/IEC JTC1/SC29/WG11 (Moving Picture Experts Group—MPEG) 就是一个在数字媒体标准化中起到重要作用的国际标准化组织。所以，这篇文章关注现有的和正在成形的各种数据的 MPEG 标准。

接下来的几节分别给出多视点视频、深度和关联数据的编码综述，集中关注已有的和正在成形的 MPEG 标准。第 3 节介绍用于 3D 视频表示以及 3D 计算机图形学的静态和动态 3D 网格数据的压缩。容错以及 3D 视频的多描述编码 (MDC) 在第 4 节中介绍。然后在第 5 节中介绍用水印保护 3D 内容的方法。最后，第 6 节概况和总结这篇文章的观点。

A.3 多视点视频、深度和关联数据编码

很多 3D 电视系统都基于场景，这些 3D 场景由 N 个相机获取（见 [34–41]）。最简单的情形就是经典的两路视频构成的立体视频，更复杂一些的系统用 8 路、16 路或更多路相机获取的视频。有些系统还使用可以当作视频信号的深度信息。这一节综述这类数据的压缩算法和标准。更早对这个研究方向的综述可以参见 [42]。通过共享内容、共享一些相机，多视点视频比起单视点视频的编码可以一定提升。在多视点视频编码中，除了挖掘单个相机的数据在时间上的关联之外，临近的相机之间的关系也被挖掘出来。于是多视点视频在单视点视频的基础上增加了一个新的压缩维度：视点间的方向。

A.3.1 传统立体视频编码

立体视频是多视点视频中最重要的一个特例，是有 $N=2$ 个视点的情形。压缩传统的立体视频已经经过了长时间的研究，相关的标准也已经问世。一个传统的立体对由从两个略微不同的视点（对应人的瞳距）观察同一场景的两幅静态图组成。这两幅图片总的来说极其相似，使得它们很适合压缩，比如从一幅

图片预测另一幅。例如，其中一幅可以不参考另一幅而单独压缩，而第二幅图片可以从已经编码的一幅图片来预测，就像视频压缩中的基于时间预测的动态补偿那样。

对两幅图片的样本基于场景和相机属性（包括位置和相机内部参数比如焦距）的 3D 几何各自对应起来。每个样本的位移或视差就等价于一个视频序列中两幅连续图像的 **dense motion field**。于是就可以用运动估计和运动补偿一样的方法去做视差分析和视差补偿，以此实现图像预测，然后就只需要编码预测误差或者进一步的残差即可。

然而，一些特定的动态补偿和视差补偿间的区别需要考虑进来。视差向量场的统计与运动向量场的统计不同。视差是有偏的，而且相对较大。零视差以为着一个点在 3D 场景中的深度极大，而一个距离相近很近的 3D 点会有很大的视差值。这需要对视差向量的熵编码做调整。总的来说，在正常的帧率下，视频序列中在时间上邻近的图像倾向于比立体视频的两个视点的图像更加相似。图像损失（内容在一张图中可见但是在另一张图中不存在，也就无法被预测）平均来说在立体视频中出现得比时间上邻近的两幅图更多。此外，立体图片对中的差别可能来自不正确的白平衡或色彩平衡，也可能来自场景的光照和表面反射效果的差别。

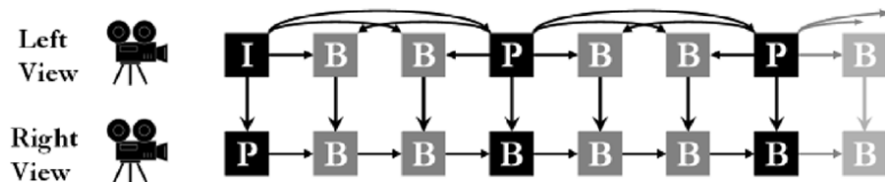


图 A.1 H.262/MPEG-2 视频多视点类 (multiview profile) 的预测示意

视角间和时间上的预测组合是有效压缩传统立体视频的基本方法。对应的标准已经在 ITU-T Rec.H.262/ISO/IEC 13818-2 MPEG-2Video, the Multiview Profile^[43,44] 中定义了，如图A.1所示。左眼视频不参考右眼视频单独编码，使用标准的 MPEG-2。这保证了对 H.262/MPEG-2 的主类 (Main Profile) 的前向兼容性，因为这样可以解码左眼视频流来显示 2D 视频。对右眼视频，时间轴预测之外还可以允许做视点间预测。

然而，对比分别压缩两个视频流而言，这样的压缩效率提升很有限。这主要是由于基于时间的预测已经达到了很高的性能。特别是当基于时间的预测对

于一张特定图像效率不错的时候（比如图A.1中右眼视频的B帧），那么增加视角间的预测就无法显著提升编码性能了。时间上邻近的图像平均而言一般都比空间上邻近的图像更相似。

对于编码为I帧的图像，也就是说不参考其他视频序列中时间上邻近的帧来编码的帧，加入视角间的预测有明显成效。一般每半秒到一秒时间就会在视频序列中插入这样一个I帧来保证随机访问和容错性。在图A.1中，左眼序列的最左侧一帧被编码为I帧，当分别编码两个视频序列时，对应的右眼序列中的那一帧也可以编码为I帧。但是在H.262/MPEG-2视频多视点编码中，可以采用视角间预测来达到压缩效率上比编为I帧方式的显著提升。

对于传统立体视频压缩的研究还在各个方向上继续进行着，包括对各个频道的最优码率分配以及放弃前向兼容去设计一个更加有效的视角间预测的结构。相关算法都是基于更新的视频编码比如H.263^[45]，MPEG-4 Visual^[46]或H.264/AVC^[47-49]。但是由于立体视频的应用还没有发展到一个足够大的市场，所有这些进展包括原有的多视点类（Multiview profile）都尚未能商业化。

A.3.2 视频加深度数据的压缩

除了前文描述的经典立体视频外，还有一种传输视频信号的方式——传输一路视频和每个样本的深度图。从视频和深度信息，可以在解码端渲染出一对立体图^[50,51]。这提供了另一项功能：当用户的头部移动被捕捉到时，能够提供相对应的观察角度的图像。此外，这种格式从压缩效率的角度来看很有价值。深度信息可以被当作单色、仅有明度的视频信号。深度被限制在两个表示对应3D点到摄像机的最小和最大距离的极限距离 Z_{near} 和 Z_{far} 之间。深度范围线性地量子化到8位，也就是最近的点赋值为255，最远点赋值为0。这样一来，一张深度图就确定下来了，成为一张灰度图。这些灰阶图可以被插入视频信号的光照信道中，其色彩可以被设定为一个常数。这样获得的标准视频信号就可以用任何先进的视频编码处理了。

欧洲ATTEST的一个项目^[50]结果表明深度信息可以用这种方式非常高效地压缩。一些前沿的视频编码参与了测试（MPEG-2, MPEG-4, H.264/AVC）。一项估计显示，只要编码彩色视频信号10%到20%的比特率就足以编码高质量的深度信息了。

基于这些观察，这个ATTEST项目提出了一种新的（对比经典的DVB）向

前兼容的实现 3DTV 的方式。它采用一种分层的比特流语法。基层是一个传统的 MPEG-2 编码的彩色 2D 视频。基层可以被任何现有的 MPEG-2 解码器处理，保证前向兼容性。除此之外，比特流还包含更高一层传输编码后的深度信息。更高级的系统可以访问这一层来解码深度信息，然后通过插值生成一对立体图像来显示。

这种概念由于保证了前向兼容性、压缩率高并且相比传统立体视频功能扩展更多，所以非常有价值。不仅如此，它并不引入一种特定的编码算法。只需要制定一种高层的语法，让解码器能够把输入的两路视频正确地分辨成一路视频和一路深度即可。附加的深度信息 (Z_{near} 和 Z_{far}) 需要另外传输。所以 MPEG 为视频加深度信息制定了一种对应的容器格式“ISO/IEC 23002-3 Representation of Auxiliary Video and Supplemental Information”，或者称为 MPEG-3 Part 3^[52,53]。此外，H.264/AVC 包含一个选项通过辅助的图片语法传输深度图。此处对彩色视频信号和对应的深度视频信号的视频编码都用的是 H.264/AVC。这种方法对任何已部署的 H.264/AVC 设备都能做到前向兼容。

对视频加深度的格式而言，有一个普遍存在的问题就是深度信息的生成。能够在捕获视频的同时自动获取每个像素深度的摄像机已经问世而且在进一步增强，但是得到的深度场的精度还是很有限。计算机视觉相关的文献中对深度估计的算法有了很深入的研究，强大的解决方案也已经出现。但是这总是一个建立在一定残差误差概率上的估计值。估计的误差会影响渲染出的图像的质量。全自动而且精确的深度获取系统仍然有待开发。对于特定的应用，用户辅助的内容生成可以作为一种选择。即使有了完美的深度图，遮挡也会使渲染出的图像有瑕疵。这些效应在距离原始摄像机位更远的虚拟点上会更加严重。增加一些层（深度视频层作为深度图像层的扩展^[54]）或者扩展到多视点视频加深度^[55,56]能够以增加数据量和复杂度为代价，帮助减小这些问题。

A.3.3 多视点视频编码

3DTV 系统的一个共同部分是传输给用户的同一场景不同视角视频的使用。最直接的解决方法就是把所有的视频信号分别用最新的编码方式比如 H.264/AVC 编码。但是多视点视频在视角间存在统计意义上很大的相关性，因为每个摄像机捕获的都是不同视角的同一场景的图像。这些相关性可以用来进行时间和视角间预测，如图A.2所示。图像不仅仅根据时间上临近的图来预测，

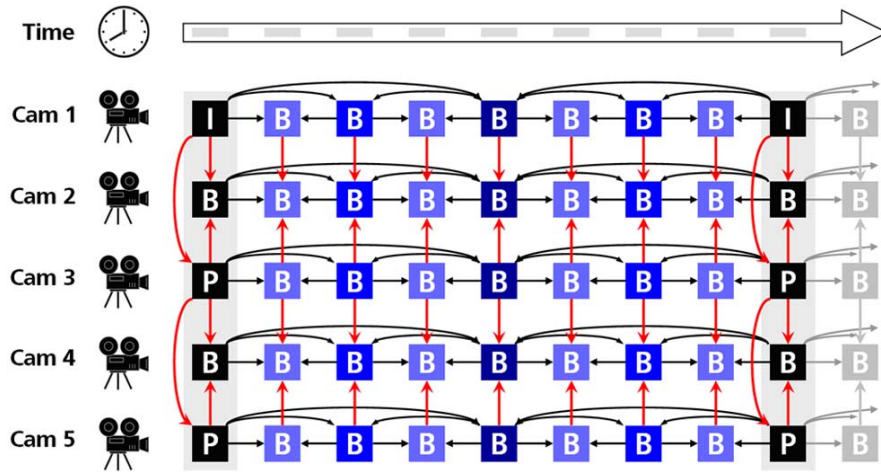


图 A.2 MVC 的时间和视角间预测结构

还根据相邻视角的对应图像预测。统计表明这样的组合式预测能够极大地提高编码效率^[3,4]。多视点图像编码的前沿研究在 [5,57] 中有报告。

几个研究小组提出了多视点视频 (MVC)，并且发展了专门的视角间和时间预测的架构来有效地挖掘利用多视点视频数据集集中的统计相关性 (见 [58–65])。其中，H.264/AVC 时间和视角间预测的语法支持的基于 B 帧的算法^[66] 在 MPEG 标准的穷尽测试下被证明是性能最好的^[6–9]。在这些实验中，客观和主观的测量都显示 MVC 极大地超越了独立压缩各个视频。然而，能够获得的提升极其依赖内容和相机间距、帧率、内容复杂度 (运动、纹理) 等属性。对于一些数据集，信噪比峰值 (PSNR) 增益在 0.5dB 以下。最大的增益达到过 3dB。

图A.2中描述的时间和视角间组合预测的一个不足就是其复杂度。包括计算复杂度、内存需求和延迟。在 [9,67] 中提到，复杂度在不牺牲多少编码效率的情形下可以极大地降低。视角间预测此时被限制在作为 I 帧独立编码的关键帧上 (例如图A.2中的 t_0 和 t_8)。MVC 中大部分编码增益来自于这些为了随机访问而不进行视角间预测的帧。忽略有时间预测的帧的视角间预测并不消耗多少编码效率却显著降低复杂度。

在结合时间和视角间预测之外，专门的 MVC 算法被提了出来。基于深度/视差的视角间插值预测^[68–70] 的基本思想是在编码 (这需要另外传输深度/视差) 或解码端估计深度或视差，然后进行视角间插值或者 3D 扭曲来预测。但是目前得到的增益不高。仅仅对于摄像机参数极其接近的很小一部分数据集，

这样的视角间插值预测在同样画质下能节省 5% 的比特率。想要优化性能还要做进一步的研究。

不仅如此，光照和色彩的不一致影响了对视角间统计相关性的挖掘。通常这样的效应能被正确的条件设定减小，但是 MVC 算法也要能够应对这样的情形，因为输入的白平衡和色彩平衡并不能保证。同样，一定的光照条件下，光照（聚光灯、阴影等）在视角间会有很大的不同。这些问题或许能通过 [71] 和 [72] 中提出的正确的光照和色彩补偿来解决。基本思想是在宏块级别修改运动补偿。在对待编码帧块和参考块的像素值相减之前，二者的平均值根据对应的像素值做补偿。其中假设了本地的光照和色彩变化是常值，这是个模型的准确率和复杂度之间的权衡。对于一些测试数据，光照补偿获得了 0.7dB 的增益。但是这个增益极大地依靠测试数据，在一些情况下，增益可以忽略甚至根本不存在。对所有的测试数据集平均下来，相同视觉质量下，能够节省 5% 的比特率。

另一种宏块级别的光照补偿可以整合到编码过程中，一个合适的预处理也可以在编码前进行。对图像和视频的光照更正算法广为人知。然后更正过的数据可以被传到一个标准的编码器。这种方法的一大优势是编码器、解码器和码流语法不需要做任何改动。一项这个方向上的前期调研发表在 [73] 中，但是结果并不完全，性能与整合光照补偿的比较也不清楚。

另一个研究方向是改进视差估计、补偿和编码^[74]。多数的视差都被当作运动来对待，但是我们知道视差向量的统计属性和运动向量可以完全不同。视差估计在计算机视觉文献中已经被广泛研究。通常，级别的几何属性和约束会考虑进来。比如搜索可以通过沿着核极线（epipolar lines）完成。这可能带来更好的估计和降低复杂度。此外，专门的视差编码或许能提高视角间预测的编码效率。专门的 MVC 编码模式比如视角间直接模式^[75]正在研究中。

MVC 的扩展性结合正在被研究，比如 [76–80]。目前可扩展的功能仅能依靠降低压缩效率。分布式 MVC 在 [81] 中被研究。加入用户交互考虑的有效传输在 [82,83] 中初次被讨论。最后，能够随机访问的有效的 MVC 编码机制、并行处理和内存管理也正在研究中。

由于结果明显表明 MVC 优于单独编码各路视频信号，业界又显然需要一个对应的标准，ISO/MPEG 和 ITU/VCEG 决定发展一种专门的 MVC 标准^[84,85]。这会是一个 H.264/AVC 的扩展（第 4 修正案），这预计在 2008 年内完工。MVC

是这一期特刊的主要关注点。我们因此向读者介绍包含最前沿 MVC 技术的文章。

A.3.4 结论和未来研究方向

对立体视频、多视点视频和相关深度、视差数据的研究已经有很高的成熟度。相关的国际标准也已经问世，使 3DTV 系统和应用成为可能。然而，相对于其他类型和媒体数据的编码，这个领域还很年轻，算法还有很大的进步空间。这包括对 MVC 实力的优化和新的专门的 MVC 算法的开发。深度和视差编码可能在将来的专门的算法中进一步优化。此外，有更多复杂的 3DTV 数据结构比如分层的深度视频和多视点视频加深度来提供更多的功能。对这些数据的有效编码算法还在研究之中。初步的结果在 [86] 和 [87] 中有报告。其他多视点描述的类型也在研究中，这往往和特定的一类 3D 显示器相关。[88] 中有一个例子。

原文索引

Smolic A, Müller K, Stefanoski N, et al. Coding algorithms for 3DTV—a survey. IEEE transactions on circuits and systems for video technology, 2007, 17(11):1606 – 1620

附录 B 实验原始数据

表 B.1 VS2008 性能分析报告 (前 50 项记录)

Function Name	Exclusive Time	Exclusive Time %	Number of Calls
macroblockPredGetDataUV	341.01	14.41	2,632,478
macroblockPredGetDataY	321.37	13.58	1,316,239
idct4x4_c	316.86	13.39	5,028,960
macroblockGetPred_axb	181.29	7.66	1,316,239
macroblockGetPred_axb_Bi	133.42	5.64	571,944
macroblockGetHalfPel	132.47	5.6	148,552
Filter	101.92	4.31	6,215,308
addMacroblockdata	85.36	3.61	184,171
iquant4x4_c	84.48	3.57	5,028,960
macroblockPBPrediction	70.95	3	184,171
macroblockInterDecode_uv	67.05	2.83	184,171
save_data_pb	55.32	2.34	200,880
macroblockInterDecode16x16_y	54.34	2.3	184,171
PSliceDecode	46.38	1.96	124
macroblock_read_cavlc_B_slice	37.29	1.58	155,520
GetHorFilterStrength	33.17	1.4	3,325,760
CheckMvDataB	31.38	1.33	4,916,240
getRefXYDirectB	29.65	1.25	582,670
macroblockIntraDecode_y	26.87	1.14	26,429
FilterMB	25.12	1.06	210,600
calc_B_skip_refs	19.16	0.81	145,645
LumaHorFilter	14.34	0.61	210,600
macroblockIntraDecode_uv	14.24	0.6	26,429
setRefXY	13.78	0.58	2,330,348
LumaVerFilter	13.15	0.56	210,600
unscan_zig_4x4	12.03	0.51	2,942,655
GetVerFilterStrength	9.82	0.42	3,325,760
macroblock_read_cavlc_P_slice	8.92	0.38	45,360
ChromaHorFilter	8.86	0.37	210,600
ChromaVerFilter	8.7	0.37	210,600

续下页

续表 B.1 VS2008 性能分析报告 (前 50 项记录)

Function Name	Exclusive Time	Exclusive Time %	Number of Calls
require_data_decode	7.72	0.33	184,171
macroblock_processor_PB	6.02	0.25	4,464
iquant2x2dc_c	4.67	0.2	421,200
idct8x8	4.66	0.2	21,920
T264dec_mb_read_intra_cavlc	3.61	0.15	26,429
CheckMvDataP	2.55	0.11	868,249
block_residual_read_cavlc	2.52	0.11	106,045
ISliceDecode	2.47	0.1	6
save_data	2.46	0.1	9,720
iquant8x8_c	2.31	0.1	21,920
process_data_decode_I	2.31	0.1	26,429
macroblock_read_cavlc_PB_y	2.26	0.1	15,493
do_16x16_luma_prediction	2.24	0.09	21,359
idct4x4dc_c	2.01	0.08	21,359
_printf	1.89	0.08	911
do_8x8_luma_prediction	1.85	0.08	15,560
require_data_decode_I	1.6	0.07	26,429
macroblock_read_cavlc_uv	1.31	0.06	41,922
do_8x8_chroma_prediction	1.29	0.05	52,858
BitstreamGetBits	1.21	0.05	580,391

表 B.2 VTune 性能分析报告 (前 50 项记录)

Function	Calls	Self Time	% in fuction
macroblockPredGetDataUV	2632478	774863	0.61
macroblockPredGetDataY	1316239	702575	0.58
idct4x4_c	5028960	398076	1
macroblockInterDecode16x16_y	184171	328834	0.46
macroblockGetPred_axb	1316239	311730	0.11
FilterMB	210600	203506	0.21
Filter	6215308	191937	1
iquant4x4_c	5028960	157459	1
macroblockInterDecode_uv	184171	154697	0.48
macroblockPBPrediction	184171	150609	0.05

续下页

续表 B.2 VTune 性能分析报告 (前 50 项记录)

Function	Calls	Self Time	% in fuction
macroblockGetHalfPel	148552	134793	1
macroblockGetPred_axb_Bi	571944	132951	0.05
GetVerFilterStrength	3325760	125372	0.64
GetHorFilterStrength	3325760	117207	0.65
CheckMvDataB	4916240	115925	1
calc_B_skip_refs	145645	83987	0.47
addMacroblockdata	184171	82547	1
save_data_pb	200880	66356	0.76
setRefXY	2330348	59615	1
LumaHorFilter	210600	56700	0.49
LumaVerFilter	210600	56175	0.46
unscan_zig_4x4	2942655	55896	1
macroblock_read_cavlc_B_slice	155520	55255	0.23
macroblockIntraDecode_y	26429	47460	0.42
ChromaVerFilter	210600	37978	0.51
ChromaHorFilter	210600	37163	0.55
getRefXYDirectB	582670	36556	1
macroblock_processor_PB	4464	29068	0.01
macroblockIntraDecode_uv	26429	28597	0.5
PSliceDecode	124	19439	0
CheckMvDataP	868249	18065	1
macroblock_read_cavlc_P_slice	45360	13979	0.36
_security_check_cookie	942387	13589	1
block_residual_read_cavlc	106045	12540	0.64
iquant2x2dc_c	421200	9292	1
T264dec_mb_read_intra_cavlc	26429	9065	0.31
require_data_decode	184171	8725	1
BitstreamGetBits	580391	8431	1
idct8x8	21920	4918	1
macroblock_read_cavlc_PB_y	15493	3735	0.42
process_data_decode_I	26429	3681	0.02
save_data	9720	3592	1
ProcessData	1	2619	0
idct4x4dc_c	21359	2584	1

续下页

续表 B.2 VTune 性能分析报告 (前 50 项记录)

Function	Calls	Self Time	% in fuction
iquant8x8_c	21920	2566	1
macroblock_read_cavlc_uv	41922	2561	0.55
require_data_decode_I	26429	2389	1
do_16x16_luma_prediction	21359	2251	0.96
do_8x8_luma_prediction	15560	2192	0.93
T264dec_mb_read_coff_token_t0	83162	2112	1

表 B.3 不同视频解码速度一览 (基准性能)

Clip Name	Threads(s)	Time(ms)	Decoding FPS	Viewer FPS
dual144	1	249	522.34	261.17
dual288	1	877	148.25	74.12
dual576	1	3855	33.72	16.86
dual720	1	8754	14.85	7.43
golf25	1	827	241.80	30.23
golf25dual	1	199	251.13	125.56
golf623	1	26344	189.19	23.65
race81	1	3773	171.73	21.47
single144	1	115	563.08	563.08
single288	1	363	178.94	178.94
single576	1	1489	43.65	43.65
single720	1	3343	19.44	19.44

表 B.4 不同视频解码速度一览 (MS 编译器)

Clip Name	Threads(s)	Time(ms)	Decoding FPS	Viewer FPS
dual144	1	219	593.61	296.80
dual288	1	813	159.90	79.95
dual576	1	3422	37.99	18.99
dual720	1	7765	16.74	8.37
golf25	1	765	261.44	32.68
golf25dual	1	187	267.38	133.69
golf623	1	23328	213.65	26.71

续下页

续表 B.4 不同视频解码速度一览 (MS 编译器)

Clip Name	Decoding Threads(s)	Decoding Time(ms)	Decoding FPS	Viewer FPS
race81	1	3328	194.71	24.34
single144	1	93	698.92	698.92
single288	1	328	198.17	198.17
single576	1	1344	48.36	48.36
single720	1	3047	21.33	21.33
dual144	2	125	1040.00	520.00
dual288	2	407	319.41	159.71
dual576	2	2063	63.02	31.51
dual720	2	4843	26.84	13.42
golf25	2	391	511.51	63.94
golf25dual	2	94	531.91	265.96
golf623	2	12969	384.30	48.04
race81	2	1687	384.11	48.01
single144	2	62	1048.39	1048.39
single288	2	187	347.59	347.59
single576	2	719	90.40	90.40
single720	2	2125	30.59	30.59
dual144	3	94	1382.98	691.49
dual288	3	297	437.71	218.86
dual576	3	1625	80.00	40.00
dual720	3	4094	31.75	15.88
golf25	3	266	751.88	93.98
golf25dual	3	78	641.03	320.51
golf623	3	10297	484.02	60.50
race81	3	1218	532.02	66.50
single144	3	63	1031.75	1031.75
single288	3	187	347.59	347.59
single576	3	766	84.86	84.86
single720	3	2203	29.51	29.51
dual144	4	78	1666.67	833.33
dual288	4	250	520.00	260.00
dual576	4	1891	68.75	34.37
dual720	4	4656	27.92	13.96
golf25	4	219	913.24	114.16

续下页

续表 B.4 不同视频解码速度一览 (MS 编译器)

Clip Name	Decoding Threads(s)	Decoding Time(ms)	Decoding FPS	Viewer FPS
golf25dual	4	62	806.45	403.23
golf623	4	10093	493.81	61.73
race81	4	1079	600.56	75.07
single144	4	62	1048.39	1048.39
single288	4	171	380.12	380.12
single576	4	609	106.73	106.73
single720	4	1719	37.81	37.81

表 B.5 不同视频解码速度一览 (ICC 编译器)

Clip Name	Threads(s)	Time(ms)	Decoding FPS	Viewer FPS
dual144	1	219	593.61	296.80
dual288	1	813	159.90	79.95
dual576	1	3422	37.99	18.99
dual720	1	7765	16.74	8.37
golf25	1	765	261.44	32.68
golf25dual	1	187	267.38	133.69
golf623	1	23328	213.65	26.71
race81	1	3328	194.71	24.34
single144	1	93	698.92	698.92
single288	1	328	198.17	198.17
single576	1	1344	48.36	48.36
single720	1	3047	21.33	21.33
dual144	2	125	1040.00	520.00
dual288	2	407	319.41	159.71
dual576	2	2063	63.02	31.51
dual720	2	4843	26.84	13.42
golf25	2	391	511.51	63.94
golf25dual	2	94	531.91	265.96
golf623	2	12969	384.30	48.04
race81	2	1687	384.11	48.01
single144	2	62	1048.39	1048.39
single288	2	187	347.59	347.59
single576	2	719	90.40	90.40

续下页

续表 B.5 不同视频解码速度一览 (ICC 编译器)

Clip Name	Decoding Threads(s)	Decoding Time(ms)	Decoding FPS	Viewer FPS
single720	2	2125	30.59	30.59
dual144	3	94	1382.98	691.49
dual288	3	297	437.71	218.86
dual576	3	1625	80.00	40.00
dual720	3	4094	31.75	15.88
golf25	3	266	751.88	93.98
golf25dual	3	78	641.03	320.51
golf623	3	10297	484.02	60.50
race81	3	1218	532.02	66.50
single144	3	63	1031.75	1031.75
single288	3	187	347.59	347.59
single576	3	766	84.86	84.86
single720	3	2203	29.51	29.51
dual144	4	78	1666.67	833.33
dual288	4	250	520.00	260.00
dual576	4	1891	68.75	34.37
dual720	4	4656	27.92	13.96
golf25	4	219	913.24	114.16
golf25dual	4	62	806.45	403.23
golf623	4	10093	493.81	61.73
race81	4	1079	600.56	75.07
single144	4	62	1048.39	1048.39
single288	4	171	380.12	380.12
single576	4	609	106.73	106.73
single720	4	1719	37.81	37.81

附录 C 实验中使用的部分代码和配置文件

代码 C.1 JMVC 编码八路视频配置文件样例

```

1 # MVC.cfg
2 # JMVC Main Configuration File
3 #===== GENERAL =====
4 InputFile          golfl # input file
5 OutputFile         golfl # bitstream file
6 ReconFile          g      # reconstructed file
7 SourceWidth        320    # input frame width
8 SourceHeight       240    # input frame height
9 FrameRate          30.0   # frame rate [Hz]
10 FramesToBeEncoded  623    # number of frames
11
12 #===== CODING =====
13 SymbolMode         0      # 0=CAVLC, 1=CABAC
14 FRExt              1      # 8x8 transform (0:off, 1:on)
15 BasisQP            31     # Quantization parameters
16 #===== INTERLACED =====
17 MbAff              0      # 0=frameMb, 1=MbAff
18 PAff               0      # 0=frame, 1=field, 2=frame/field
19 #===== STRUCTURE =====
20 GOPSize            4      # GOP Size (at maximum frame rate)
21 IntraPeriod        12     # Anchor Period
22 NumberReferenceFrames 3    # Number of reference pictures
23 InterPredPicsFirst 1      # 1 Inter Pics; 0 Inter-view Pics
24 Log2MaxFrameNum    11     # specifies max. value for frame_num (4..16)
25 Log2MaxPocLsb      7      # specifies coding of 'POCs (4..15)
26 DeltaLayer0Quant   0      # differential QP for layer 0
27 DeltaLayer1Quant   3      # differential QP for layer 1
28 DeltaLayer2Quant   4      # differential QP for layer 2
29 DeltaLayer3Quant   5      # differential QP for layer 3
30 DeltaLayer4Quant   6      # differential QP for layer 4
31 DeltaLayer5Quant   7      # differential QP for layer 5
32 MaxRefIdxActiveBL0  2      # active entries in ref list 0 for B slices
33 MaxRefIdxActiveBL1  2      # active entries in ref list 1 for B slices
34 MaxRefIdxActiveP    1      # active entries in ref list for P slices
35 #===== MOTION SEARCH =====
36 SearchMode         4      # Search mode (0:BlockSearch, 4:FastSearch)
37 SearchFuncFullPel  3      # Search function full pel
38                      # (0:SAD, 1:SSE, 2:HADAMARD, 3:SAD-YUV)

```



```

39 SearchFuncSubPel      2      # Search function sub pel
40                        #      (0:SAD, 1:SSE, 2:HADAMARD)
41 SearchRange           32      # Search range (Full Pel)
42 BiPredIter            4      # Max iterations for bi-pred search
43 IterSearchRange       8      # Search range for iterations (0: normal)
44 #===== LOOP FILTER =====
45 LoopFilterDisable     0      # Loop filter idc (0: on, 1: off, 2:
46                        #      on except for slice boundaries)
47 LoopFilterAlphaC0Offset 0      # AlphaOffset(-6..+6): valid range
48 LoopFilterBetaOffset  0      # BetaOffset (-6..+6): valid range
49 #===== WEIGHTED PREDICTION =====
50 WeightedPrediction    0      # Weighting IP Slice (0:disable, 1:enable)
51 WeightedBiprediction  0      # Weighting B Slice (0:disable, 1:explicit,
52                        #      2:implicit)
53 #===== PARALLEL DECODING INFORMATION SEI Message =====
54 PDISEIMessage         0      # PDI SEI message enable (0:disable, 1:enable)
55 PDIInitialDelayAnc    2      # PDI initial delay for anchor pictures
56 PDIInitialDelayNonAnc 2      # PDI initial delay for non-anchor pictures
57 #===== NESTING SEI MESSAGE =====
58 NestingSEI           0      # (0: NestingSEI off, 1: NestingSEI on)
59 SnapShot             0      # (0: SnapShot off, 1: SnapShot on)
60 #===== ACTIVE VIEW INFO SEI MESSAGE =====
61 ActiveViewSEI         0      # (0: ActiveViewSEI off, 1: ActiveViewSEI on)
62 #===== VIEW SCALABILITY INFORMATION SEI MESSAGE =====
63 ViewScalInfoSEI      0      # (0: ViewScalSEI off, 1: ViewScalSEI on)
64 #===== Level conformance checking of the DPB size =====
65 DPBConformanceCheck  1      # (0: disable, 1: enable, 1:default)
66 #===== MULTIVIEW CODING PARAMETERS =====
67 NumViewsMinusOne     7      # (Number of view to be coded minus 1)
68 ViewOrder 0-2-1-4-3-6-5-7 # (Order in which view_ids are coded)
69
70 View_ID              0      # view_id (0..1024): valid range
71 Fwd_NumAnchorRefs    0      # (number of list 0 references for anchor)
72 Bwd_NumAnchorRefs    0      # (number of list 1 references for anchor)
73 Fwd_NumNonAnchorRefs 0      # (number of list 0 references for non-anchor)
74 Bwd_NumNonAnchorRefs 0      # (number of list 1 references for non-anchor)
75 View_ID              1      # view_id (0..1024): valid range
76 Fwd_NumAnchorRefs    1      # (number of list 0 references for anchor)
77 Bwd_NumAnchorRefs    1      # (number of list 1 references for anchor)
78 Fwd_NumNonAnchorRefs 1      # (number of list 0 references for non-anchor)
79 Bwd_NumNonAnchorRefs 1      # (number of list 1 references for non-anchor)
80 Fwd_AnchorRefs       0 0    # ref_idx view_id combination
81 Bwd_AnchorRefs       0 2    # ref_idx view_id combination
82 Fwd_NonAnchorRefs    0 0    # ref_idx view_id combination
83 Bwd_NonAnchorRefs    0 2    # ref_idx view_id combination
84

```

85	View_ID	2
86	Fwd_NumAnchorRefs	1
87	Bwd_NumAnchorRefs	0
88	Fwd_NumNonAnchorRefs	0
89	Bwd_NumNonAnchorRefs	0
90	Fwd_AnchorRefs	0 0
91	View_ID	3
92	Fwd_NumAnchorRefs	1
93	Bwd_NumAnchorRefs	1
94	Fwd_NumNonAnchorRefs	1
95	Bwd_NumNonAnchorRefs	1
96	Fwd_AnchorRefs	0 2
97	Bwd_AnchorRefs	0 4
98	Fwd_NonAnchorRefs	0 2
99	Bwd_NonAnchorRefs	0 4
100		
101	View_ID	4
102	Fwd_NumAnchorRefs	1
103	Bwd_NumAnchorRefs	0
104	Fwd_NumNonAnchorRefs	0
105	Bwd_NumNonAnchorRefs	0
106	Fwd_AnchorRefs	0 2
107	View_ID	5
108	Fwd_NumAnchorRefs	1
109	Bwd_NumAnchorRefs	1
110	Fwd_NumNonAnchorRefs	1
111	Bwd_NumNonAnchorRefs	1
112	Fwd_AnchorRefs	0 4
113	Bwd_AnchorRefs	0 6
114	Fwd_NonAnchorRefs	0 4
115	Bwd_NonAnchorRefs	0 6
116		
117	View_ID	6
118	Fwd_NumAnchorRefs	1
119	Bwd_NumAnchorRefs	0
120	Fwd_NumNonAnchorRefs	0
121	Bwd_NumNonAnchorRefs	0
122	Fwd_AnchorRefs	0 4
123	View_ID	7
124	Fwd_NumAnchorRefs	1
125	Bwd_NumAnchorRefs	0
126	Fwd_NumNonAnchorRefs	0
127	Bwd_NumNonAnchorRefs	0
128	Fwd_AnchorRefs	0 6

代码 C.2 JMVC 打包八路视频配置文件样例

```
1 # assembler.cfg
2 #===== Assembler: View Encode order =====
3 OutputFile          golf623.mvc
4 NumberOfViews      8
5 InputFile0          golf1_0.264
6 InputFile1          golf1_2.264
7 InputFile2          golf1_1.264
8 InputFile3          golf1_4.264
9 InputFile4          golf1_3.264
10 InputFile5          golf1_6.264
11 InputFile6          golf1_5.264
12 InputFile7          golf1_7.264
```

代码 C.3 JMVC 打包八路视频脚本样例

```
1 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 0
2 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 2
3 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 1
4 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 4
5 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 3
6 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 6
7 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 5
8 H264AVCEncoderLibTestStatic.exe -vf MVC.cfg 7
9 MVCBitStreamAssemblerStatic.exe -vf assembler.cfg
```

综合论文训练记录表

学生姓名	卿培	学号	2006011291	班级	计 62
论文题目	多视点（Multi-view Coding）视频的并行实时解码				
主要内容以及进度安排	<div>主要内容：<ul style="list-style-type: none">在已有的 MVC 解码器代码基础上，<ul style="list-style-type: none">修正 bug，使解码结果与参考软件完全一致针对 Intel Core 系列 CPU 进行单核心性能优化针对 Intel Core 系列 CPU 进行并行解码优化设计并实现基于 NVIDIA 3D Vision 平台的立体视频播放器</div> <div>进度安排：<ul style="list-style-type: none">第三周至第四周：熟悉现有的 MVC Decoder 代码第五周至第七周：设计并实现 3D 播放器第八周至第救周：进行解码器单核心性能优化第十周至第十一周：进行解码器并行优化第十二周至第十四周：进行解码测试，收集数据并撰写毕业论文</div> <div>指导教师签字：_____</div> <div>考核组组长签字：_____</div> <div>年 月 日</div>				
	<div>中期考核意见</div> <div>考核组组长签字：_____</div> <div>年 月 日</div>				

指导教师评语	<div>指导教师签字：_____</div> <div>年 月 日</div>
评阅教师评语	<div>评阅教师签字：_____</div> <div>年 月 日</div>
答辩小组评语	<div>答辩小组组长签字：_____</div> <div>年 月 日</div>

总成绩：_____

教学负责人签字：_____

年 月 日