上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：新一代视频编码（HEVC）中的帧间预测编码及仿真

学生姓名：李 雪

学生学号：5080309215

专　　业：信息工程

指导教师：高志勇

学院(系)：电子信息与电气工程学院（电子工程系）

新一代视频编码(HEVC)中的帧间预测及仿真

摘要

在视频编码器中，采用运动估计、运动补偿、帧间模式判别等技术的帧间预测编码是视频压缩的主要技术，不同视频压缩的帧间预测标准和算法会严重影响编码器的性能。本文研究了新一代视频编码（HEVC）中的帧间预测标准和参考编码器（HM6.1）的帧间预测算法；结合 H.264 实时编码器 Medialib 中帧间预测算法修改得到 HEVC 实时编码帧间预测算法；并在 HM6.1 编码器中实现 4 种帧间预测改进算法，在超高清视频序列（4Kx2K——4096x2048p@30fps、3840x2160p@30fps）下验证帧间预测改进算法的效率和性能。在帧间预测中整像元运动估计和模式判别是影响帧间预测部分运算量的重要因素。在不大幅度降低性能的情况下，本文在整像元运动部分通过控制搜索点个数和采用降低精度的方法来降低一定的复杂度；在模式判别部分采用部分模式判断提前终止的方式来降低运算复杂度。然后通过仿真实现 4 种改进算法，验证了该算法可以在不明显降低视频质量的前提下大幅度降低帧间预测运算量。

总的来说，本文结合 HEVC 编码器算法和 H.264 实时编码器算法得到的 HEVC 实时编码器算法在 HM6.1 编码器平台上通过改进帧间预测算法验证了性能，取得了良好的编码性能、大幅度降低了搜索时间、在保证视频质量的前提下提高了编码效率，具有一定的可行性。

关键词:HEVC 视频编码器，超高清，帧间预测算法，帧间模式判别，整像素运动估计

INTER PREDICTION CODING AND SIMULATION OF THE NEW VIDEO ENCODER STANDARD (HEVC)

ABSTRACT

Inter prediction which uses motion estimation, motion compensation and inter mode decision is the key point of video compression technology. Different inter prediction standard and algorithm will make a difference on the encoder performance. The paper does research on the inter prediction standard and the inter prediction algorithm of the reference encoder software HM6.1 of HEVC. Further more, with the inter prediction algorithm of H.264 real-time encoder, the paper propose a new inter prediction algorithm used in real-time encoder of HEVC and four kinds of improved inter prediction algorithms used in HM6.1 encoder. The four kinds of algorithms are carried out in case of UHDV(4Kx2K——4096x2048p@30fps 、3840x2160p@30fps) to validate their performance.

Integer motion estimation and inter mode decision affect the performance of the encoder directly. Under the premise of not an obvious decrease in video performance, this paper utilizes controlling the number of search points and the decreasing a little accuracy in integer motion estimation part to reduce the complexity. Likely, the paper utilizes several mode early skip algorithms in mode decision part to reduce complexity. Then the paper verified these four improved inter prediction algorithms though simulation in HM6.1, and these experiments achieves a good performance by obviously reducing the complexity as well as not obviously affecting the video performance.

Overall, the real-time encoder algorithm and the improved algorithm which based on the HM6.1 encoder were proposed to achieve better performance between complexity and video quality, and proved a better performance.

Key words: HEVC video encoder, UHDV, inter prediction algorithm, inter mode decision, inter pixel motion estimation

目录

第一章 绪论

1.1 课题背景

从 2003 年 H.264、AVS[1]标准至今已经有 9 年历史，H.264 和 AVS 的应用都已经很成熟，从视频监控到流媒体以至到电子消费产业都已经采用了 H.264 视频压缩标准进行编解码。又随着网络技术和终端能力的不断提高和发展，人们对于目前广泛使用的编解码技术提出了新的要求，希望能够达到高清、3D、移动无线的视频传输目标。

新一代视频压缩标准（High Efficiency Video Coding——HEVC）是由 ITU-T 和 ISO/IEC 组成的联合视频组(JVT-VC)制定的。从 2010 年 4 月 JCT-VC 在德国德累斯顿召开的制定 HEVC 标准的第一次会议至今，已经召开了 9 次会议，制定出了 7 版标准草案（Working Draft），计划在 2012 年 7 月出台 CD 版草案，2013 年 1 月出台最终版草案。本毕设论文主要是以 2012 年 2 月第 8 次会议出台的第六版 HEVC 草案标准的第 18 次修订[2]（JCTVC-H1003_dH）作为参考标准进行研究，这版草案已经基本涵盖了 HEVC 编解码标准的大部分技术。

HEVC 是为实现超高清视频序列（4Kx2K——4096x2048p、3840x2160p）编码而设计的视频压缩标准。核心目标是在 H.264/AVC high profile 的基础上，压缩效率提高一倍,即在保证相同视频图像质量（PSNR）的前提下，视频流的码率减少 50%，使得高清视频对带宽的要求进一步下降。为了达成这一目标，HEVC 的复杂度必然也将大大提高，有可能是 H.264 的三倍以上。

高分辨率视频处理及 HEVC 多层处理机制使得数据处理实时吞吐量巨大，编码性能要以编码效率为代价，因此其架构设计需要权衡码率失真性能、实时编码速度、外部硬件数据吞吐能力等相互制约的因素。在编码器中帧间预测部分运算量约占总运算量的 50%-90%，其中对于消费级编码器，帧间预测部分运算量约占总运算量的 2/3；对于专业级编码器，帧间预测部分运算量占总运算量的 70%以上。可见，帧间预测部分不仅对于编码器性能有很大影响，而且可以大幅度得影响编码器效率，对于实现实时编码器是十分重要的一个环节。

1.2 视频压缩编码简介

1.2.1 视频压缩标准发展史

数字视频技术广泛应用于通信、计算机、广播电视等领域，带来了会议电视、可视电话及数字电视、媒体存储等一系列应用，促使了许多视频编码标准的产生。ITU-T 与 ISO/IEC 是制定视频编码标准的两大组织，ITU-T 的标准包括 H.261、H.263、H.264，主要应用于实时视频通信领域，如会议电视；MPEG 系列标准是由 ISO/IEC 制定的，主要应用于视频存储(DVD)、广播电视、因特网或无线网上的流媒体等，如图 1-1 所示。两个组织也共同制定了一些标准，H.262 标准等同于 MPEG-2 的视频编码标准，H.264 标准则被纳入 MPEG-4 的第 10 部分，最新的 HEVC 标准由 ITU-T 与 ISO/IEC 两大组织正在制定中。

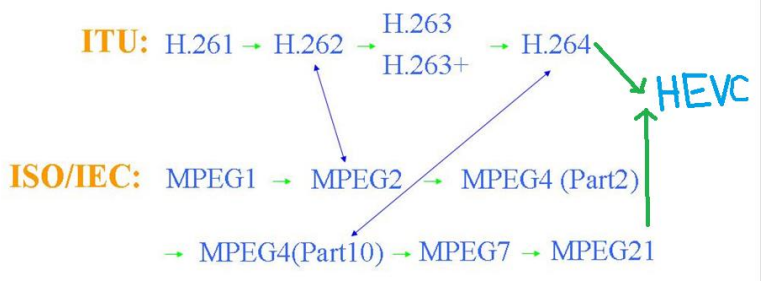


图 1-1 视频压缩标准发展史

1.2.2 HEVC 发展历史

HEVC，也被称作 H.265 或者 MPEG-H Part2，是一个视频压缩标准草案，是 H.264/AVS 视频压缩标准技术的发展。目前 ITU-T 与 ISO/IEC 两大组织建立了一个联合视频编码组织（JCT-VC）共同制定 HEVC 标准。

2010 年 4 月，JCT-VC 在德国德累斯顿召开了关于制定 HEVC 标准的第一次会议。会议上确定了新一代视频编码标准名称：HEVC(High Efficiency Video Coding)，成立了 AD HOC 小组，分领域搜集和审阅技术提案，初步定于 2012 年 7 月完成标准的终稿。

2010 年 7 月，JCTVC 在瑞士日内瓦城召开了关于制定 HEVC 视频压缩标准的第二次会议。会议上主要对于提交的核心技术进行实验性能验证。

2010 年 10 月，JCT-VC 在广州召开了关于制定 HEVC 视频压缩标准的第三次会议。会上出台了 HEVC 标准第一版草案和 HEVC 参考软件编解码器 HM1。

2011 年 1 月，JCT-VC 在韩国大邱广域市召开了关于制定 HEVC 视频压缩标准的第四次会议。会上出台了 HEVC 标准第二版草案和相对应的软件参考编解码器 HM2。

2011 年 3 月，JCT-VC 在瑞士日内瓦城召开了关于制定 HEVC 视频压缩标准的第五次会议。会议上出台了 HEVC 标准第三版草案和相对应的软件参考编解码器 HM3。

2011 年 7 月，JCT-VC 在意大利托里诺召开了关于制定 HEVC 视频压缩标准的第六次会议。会议上出台了 HEVC 标准第四版草案和相对应的软件参考编解码器 HM4。

2011 年 11 月，JCT-VC 在瑞士日内瓦城召开了关于制定 HEVC 视频压缩标准的第七次会议。会议上出台了 HEVC 标准第五版草案和相对应的软件参考编解码器 HM5.

2012 年 2 月，ICT-VC 在美国圣何塞召开了关于制定 HEVC 视频压缩标准的第八次会议，会上制定了 HEVC 标准第六版草案和相对应的软件参考编解码器 HM6。JCT-VC 小组计划于 2012 年 7 月出版 HEVC CD 版草案，2013 年 1 月 HEVC 标准正式定稿，第六版草案和最终版标准已经相差不大。本毕设是以 HEVC 第六版草案以及 HM6.1 参考编码器作为参考文献进行的编写。

2012 年 4 月，JCT-VC 在瑞士日内瓦城召开了关于制定 HEVC 视频压缩标准的第九次会议上于 2012 年 5 月 16 号出台了 HEVC 标准第七版草案。

1.3 HEVC 视频编码介绍

1.3.1 HEVC 视频基本编码工具

HEVC 视频中使用了一系列具有更高编码效率的编码工具[3]，其中包括：

（1）单元： H.264 编码以宏块（Block——16×16 的像素区域）为单位，把当前要编码的一帧图像分成若干宏块。宏块是一个矩形的二维数组，也就是说一个矩形框里的像素点需要以一个亮度宏块和两个色度宏块分别存储。而在 HEVC 中采用了一个新的概念——单元（unit），单元不单单表示亮度数组或者色度数组，而是指对应像素点中亮度数组和色度数组的集合。所以在 HEVC 中不再特意区分是在亮度宏块和色度宏块，所有的信息都在包

含在单元当中。单元这个概念并不单独使用，而是加上前缀表示特指的意思，譬如编码单元（Coding Unit）、预测单元(Prediction Unit)、变换单元(Transform Unit)。

（2）编码单元：编码单元是进行帧间预测以及帧内预测分割编码的基本单元分为 skip 和 non-skipped 两种。对于 skip 的编码单元，不需要进一步分割，整个编码单元是一个预测模式，不传输运动矢量残差和像素残差。对于 non-skip 的编码单元，可以采用帧间编码（INTER）或者帧内编码(INTRA)。HEVC 中编码单元的大小为可变的，可以取 8x8（Smallest Coding Unit），16x16，32x32，64x64（Largest Coding Unit——LCU），记做 2Nx2N(N=4,8,16,32)。每幅图片被分割为若干个 LCU 进行编码，最大编码单元（LCU）可以采用四叉树(Quad-tree)的方式划分为较小的编码单元（CU），每个编码单元又可以进一步划分为更小的编码单元，直到最小的编码单元（SCU），即 8x8 的亮度像素区域，如图 1-2 所示。多层次编码单元分割模式使得图像的编码灵活性大大增加，对于码率压缩有很大的提高。

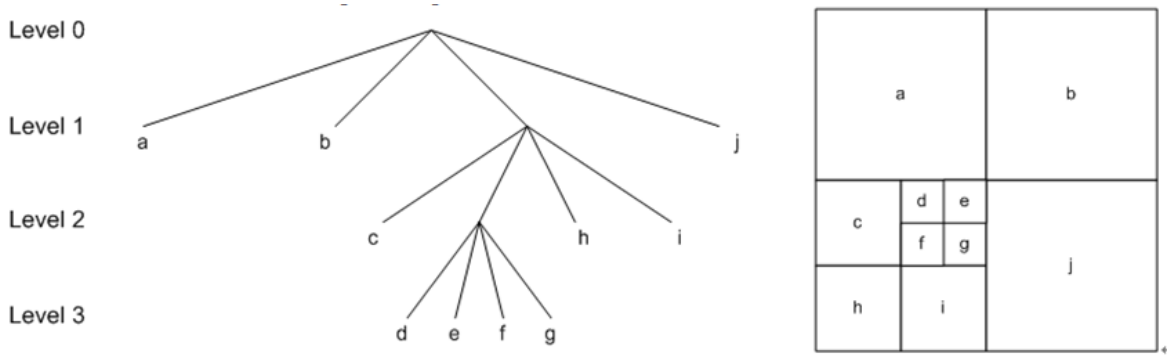


图 1-2 LCU 的分割及编码示意图

（3）预测单元：编码单元可以分割成预测单元用于帧内预测或者帧间预测。帧内预测的预测单元均为方形，分割为 2Nx2N 或者 NxN(N=4 时)；帧间预测的编码单元可以分割成 8 种预测单元，分别记做 2Nx2N，Nx2N，2NxN，NxN, 2NxnU, 2NxnD, nLx2N, and nRx2N。前四种为对称分割模式，如图 1-3 所示。后四种为非对称分割方式，分成的两个非对称块的大小比例为 1:3，如图 1-4 所示。

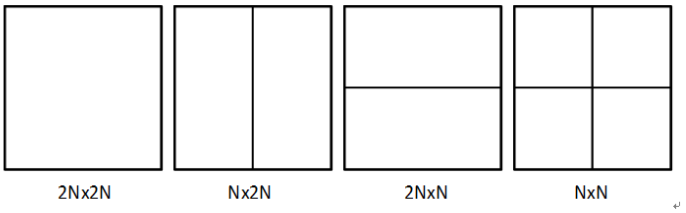


图 1-3 对称分割预测单元

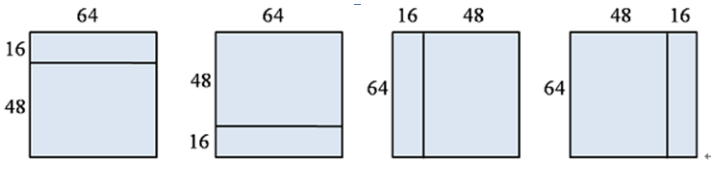


图 1-4 不对称分割预测单元

（4）变换单元：变换单元大小从 4x4 至 32x32。对于低复杂度编码来说，所有的变换单元均为方形的；对于高效率编码来说，帧内块的变换单元为方形的，帧间预测块的变换单元可为方形也可不为方形，与预测单元大小相对应。

（5）帧内预测：平面帧内预测，35 个方向的方向帧内预测

（6）帧间预测：Merge 模式替代了 SKIP 模式和 DIRECT 模式，亮度运动矢量的 8 阶 1/4 插值和色度运动矢量的 4 阶 1/8 插值

（7）熵编码：CABAC

（8）环路滤波：去块滤波、自适应样本偏移（SAO)、自适应环路滤波（ALF）

（9）样本精度：可选择样本精度

HEVC 视频编码标准主要面向超高清视频编码（4096x2048p、3840x2160p），具有以下特点：第一：性能高，压缩码率是 H.264 的 50%；第二：复杂度高，复杂度是 H.264 的三倍以上；第三：技术新，在 HEVC 标准提案中出现了很多新的帧间编码技术，对于 HEVC 性能的提高起到了决定性作用。

1.3.2 HEVC 视频编码流程

HEVC 标准采用了同 H.264 相同的混合编码架构，采用了帧内预测、帧间预测、变换、量化、反变化、反量化、去块滤波、熵编码等技术。帧内预测使用空域预测来消除空间域上的信息冗余，帧间预测使用空域-时域预测来消除时间域上的信息冗余，变换量化消除图像内的视觉冗余，去块滤波去除反量化反变换后的块效应，CABAC 熵编码对编码最后得到的比特流进行压缩。编码原理结构图如图 1-5 所示。

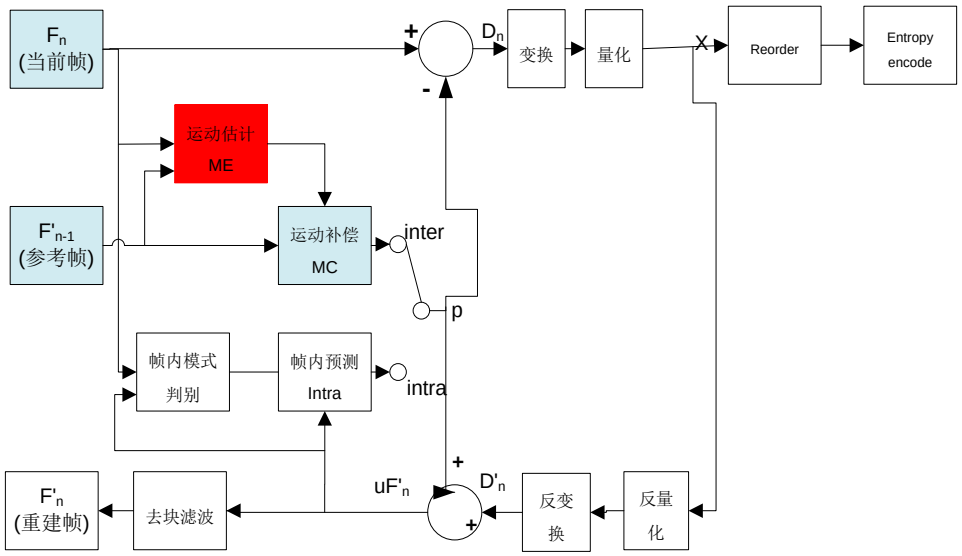


图 1-5 HEVC 视频编码结构框图[1]

图 1-5 中 Fn 表示当前要编码的帧，每一帧被分为若干个最大编码单元进行编码。每个编码单元进行帧内或者帧间编码得到最优的一种编码方式，然后通过已重建帧得到预测块，再用当前块减去预测块得到残差块，最后对残差块进行变化、量化、熵编码。其中重建帧是由变换量化后的残差块经反量化、反变化，再加上编码以前的重建帧，然后去块滤波得到的重建帧。解码流程同得到重建帧的流程是相同的。

1.4 主要研究内容

HEVC 是新一代高清视频编码技术，可以支持超高清视频序列 4096x2048p@30fps、3840x2160p@30fps 的编解码处理。HEVC 标准在相同的视频质量(PSNR)下码率比 H.264 压缩了一倍，带来了运算复杂度的大幅度提高，由于硬件实现对外存数据吞吐量限制较大，因此运算量成为实现 HEVC 实时编码器的瓶颈问题。帧间预测是 HEVC 视频编码系统中运算量最大(50%-90%)的模块，其中帧间预测的模式判别算法、整像素运动估计算法、代价缓存机制将直接影响算法性能，大幅度降低帧间预测算法的运算量。

本毕业设计包括 HEVC 帧间预测模块介绍、HEVC 帧间预测改进算法、改进算法的软件及性能分析评估 3 个方面内容。HEVC 视频标准的高性能是以算法实现复杂度为代价的，为了降低算法实现复杂度，又不能明显损害视频的性能，要对帧间预测算法进行改进：整像素运动搜索要采取合适的搜索策略、分层模式判别要增加提前终止的条件、代价缓存机制减少计算量。由于硬件吞吐量的限制，为了达到实时编码效果，需要将实时读取数据量降低，可以充分利用代价缓存机制、将已计算过的运动矢量代价保存在内存中，在下层需要到上层已经计算过的运动矢量代价时直接去上层进行提取。虽然硬件实现部分不在本文中出现，但是以上种种降低帧间预测实现复杂度的算法都是为了以后的硬件实现而设计的。

在本毕设中我学习了解了 HEVC 帧间预测模块的标准和功能、HEVC 参考编码器 HM6.1 中帧间预测算法原理，然后进行了帧间预测算法研究：参照 H.264 实时编码器帧间预测算法设计了 HEVC 实时编码器帧间预测算法，并基于 HM6.1 编码器平台设计出了帧间预测改进算法。帧间预测改进算法将 HEVC 实时编码器帧间预测算法部分实现于 HM6.1 编码器平台上，在不明显降低视频性能的前提下大大降低了帧间预测模块的运算量，提高了编码效率。

本课题主要包换以下研究内容：

1.整像素运动估计算法。整像素运动估计算法首先考虑 H.264 中 16x16 宏块的整像素运动估计算法，在运动估计过程中通过 hash 算法去除冗余信息，控制运动估计搜索点的个数，充分利用 SKIP 模式优先的原则进行提前终止；然后根据 HEVC 标准中改变了预测运动矢量的得到方式、新增了 Merge 模式等标准修改运动估计算法；最后结合算法 HEVC 参考编码器 HM6.1 代码中运动估计部分算法确定不明显降低性能的高效率整像素运动估计算法。

2. 分层帧间模式判别算法。首先根据 H.264 中模式判别算法，在 HEVC 算法中我采用了当前层 SKIP 提前终止、同一层 2Nx2N 优先级最高、PU 判断分割条件等算法；然后结合 HM6.1 模式判别代价修改 HEVC 算法。

3.结合已得到的整像素运动估计算法和模式判别算法对 HM 中帧间预测的算法进行修改，增加了模式判别提前终止条件，修改了整像素运动估计算法，通过实验数据对比进一步修改 HM 帧间预测算法，最后得到一种在小幅度损害视频性能的前提下大幅度降低帧间预测运算量的一个方法，并统计。

本毕设对 HEVC 帧间预测算法进行了研究并改进，寻求在搜索速度、编码效率之间达到最平衡的实现方案。

1.5 本文组织结构

第一章，阐述课题背景、国内外研究现状、研究的意义、HEVC 视频编码简介、主要研究内容以及本文的组织结构。

第二章，介绍帧间预测模块的标准：亚像素插值子模块、预测运动矢量（MVP）子模块、Merge 子模块，以及 HEVC 参考编码器 HM6.1 帧间预测算法：模式判别算法、运动估计算法。

第三章，根据 HM6.1 帧间预测算法与 H.264 实时编码器帧间预测算法设计 HEVC 实时编码器帧间预测算法，其中包括 P 帧的运动估计算法和帧间模式判别算法，算法中权衡了编码效率和编码性能两个方面，采用了多步提前终止和控制搜索点个数等来实现高效率的编码。为了验证实时编码器性能，本毕设基于 HM6.1 编码器平台设计了帧间预测改进算法，对整像素运动估计和模式判别进行了修改，部分实现了实时编码器帧间预测算法。

第四章，在 HM6.1 编码器平台上软件仿真验证帧间预测改进算法的性能，通过实验结果比较修正帧间预测算法，最后实现了帧间预测算法效率和性能的平衡。

第五章，结论，对本文工作的总结。

最后是谢辞与参考文献。

第二章 HEVC 帧间预测模块介绍

2.1 HEVC 帧间预测编码标准简介

HEVC 帧间预测模块包含整像素运动估计、亚像素运动估计、运动补偿、帧间模式判别等子模块，目标是为了最大限度得利用空域-时域预测来消除时间域上的信息冗余，以达到压缩码率的目的。通过运动估计得到运动矢量（MV），再通过空域-时域已经编码的编码单元运动矢量预测得到预测运动矢量（MVP），然后用运动矢量减去预测运动矢量得到运动矢量残差（MVD=MV-MVP）进行传输。可见预测运动矢量（MVP）越准确，所需要编码的编码单元残差就越小。

2.1.1 HEVC 帧间预测新增技术

HEVC 帧间预测原理与 H.264 相同，只是相较于 H.264 新增了一些新的帧间编码工具，修改了一些编码流程，整体帧间编码环节更加复杂。其中包括：

(1)GPB 概念：前向预测 P 帧和双向预测 B 帧通过使参考帧列表 list0 和 list1 相同这一方法统一为 GPB 帧，并且采用了联合参考帧(combined-list)的方法提高了编码效率。

(2)多参考帧预测：在 HEVC 中预测参考帧在每个列表下最多可以达到 33 帧，提高了运动搜索范围和准确度，编码效率也大大提高。

(3)自适应的搜索范围：运动估计搜索窗的大小随着参考帧与编码帧的距离远近成线性变化，提高了运动搜索的自适应能力，也提高了临近参考帧的搜索效率。

(4)亚像素插值：HEVC 中基于 8 阶 DCT 变换的 1/4 亮度插值和基于 4 阶 DCT 变换的 1/8 色度插值相较于 H.264 中的亚像素插值而言更为精确。

(5)预测运动矢量 MVP：修改了预测运动矢量的候选运动矢量位置与得到流程，通过传递预测运动矢量的在候选预测运动矢量列表中的索引到解码端得到预测运动矢量，大大减少了由于预测运动矢量判断失误引起的解码端误码率。

(6)Merge 模式：在 HEVC 中 Merge 模式是实现 SKIP 模式的一种方式，即不传运动矢量也不传残差。但 Merge 模式不只用于 SKIP 模式，还用于 INTER 模式下预测单元运动矢量的判断，若在 INTER 模式下经判断出当前为 Merge 模式，则不传运动矢量，仅传残差。Merge 模式是 H.264 中 SKIP 模式和 DIRECT 模式的升级，是 HEVC 帧间预测模块最大的变化。

(7)加权预测：HEVC 修改了加权预测子模块的实现，但是由于加权预测子模块不属于本文研究的低复杂度编码，故在本文中不予讲解。

(8)运动估计代价与模式识别代价：在 HEVC 中采用了基于绝对残差和(SAD)和经过哈德曼变换的（SATD）的预测单元运动矢量判断，如公式（2-1）和公式（2-2）所示；和基于亮度方差和色度方差的编码单元模式判断吗，如公式（2-3）所示。

$$J_{pred,SAD} = SAD + \lambda_{pred} * B_{pred} \quad (2\text{-}1)$$

$$J_{pred,SATD} = SATD + \lambda_{pred} * B_{pred}, \quad (2\text{-}2)$$

$$J_{mode} = (SSE_{luma} + w_{chroma} * SSE_{chroma}) + \lambda_{mode} * B_{mode}, \quad (2\text{-}3)$$

2.1.2 亚像素插值子模块[4]

亚像素插值是在做亚像素运动估计和亚像素运动补偿时用到的子模块。在 HEVC 中亚像素插值采用了基于 8 阶 DCT 变换的 1/4 亮度插值和 4 阶 DCT 变换的 1/8 色度插值，如表 2-1 和表 2-2 所示。

表 2-1 1/4 亮度插值的 8 阶滤波器系数

位置	滤波器系数
1/4	{ -1, 4, -10, 57, 19, -7, 3, -1 }
2/4	{ -1, 4, -11, 40, 40, -11, 4, -1 }
3/4	{ -1, 3, -7, 19, 57, -10, 4, -1 }

表 2-2 1/8 亮度插值的 4 阶滤波器系数

位置	滤波器系数
1/8	{ -3, 60, 8, -1 }
2/8	{ -4, 54, 16, -2 }
3/8	{ -5, 46, 27, -4 }
4/8	{ -4, 36, 36, -4 }
5/8	{ -4, 27, 46, -5 }
6/8	{ -2, 16, 54, -4 }
7/8	{ -1, 8, 60, -3 }

1/4 亮度插值的算法流程如图 2-1 所示，阴影所示的点为整像素点，其余为亚像素点，有部分整像素点未在图中画出。

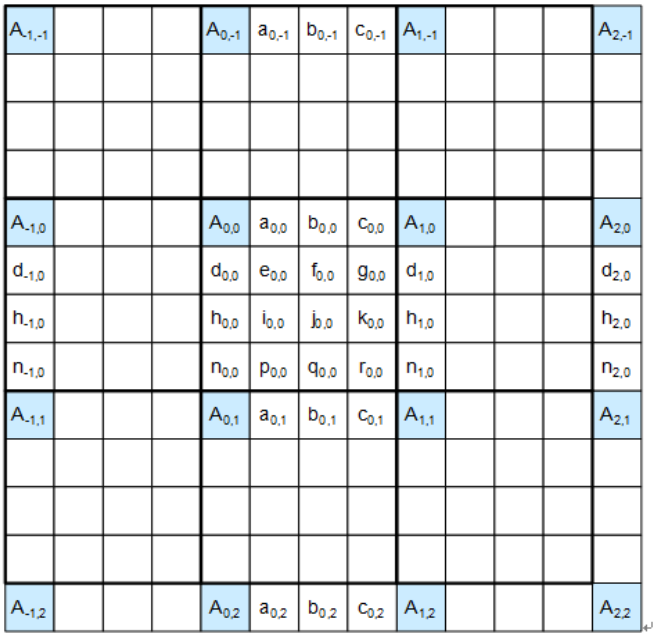


图 2-1 8 阶 1/4 亮度插值示意图

1/4 亮度插值流程如下所示：

(1) xAi, j = Clip3(0, PicWidthInSamplesL – 1, xIntL +i)

yAi, j = Clip3(0, PicHeightInSamplesL – 1, yIntL +j)

变量 shift1= BitDepthY – 8，shift2= 6，shift3=14- BitDepthY。图中 Ai, j 为整像素点亮度值，1/4 亚像素点的亮度值‘a0,0’到‘r0,0’按以下得到。

(2)亚像素点 a0,0, b0,0, c0,0, d0,0, h0,0, n0,0 的亮度值将由临近整像素点亮度值应用 8 阶滤波器计算得到，如下所示：

a0,0=(−A−3,0 + 4*A−2,0 − 10*A−1,0 + 57*A0,0 +19*A1,0 − 7*A2,0 + 3*A3,0 − A4,0) >> shift1 (2-4)

b0,0=(−A−3,0 + 4*A−2,0 − 11*A−1,0 + 40*A0,0 +40*A1,0 − 11*A2,0 + 4*A3,0 − A4,0) >> shift1 (2-5)

c0,0=(−A−3,0 + 3*A−2,0 − 7*A−1,0 + 19*A0,0 + 57*A1,0 − 10*A2,0 + 4*A3,0 − A4,0) >> shift1 (2-6)

d0,0=(−A0,−3 + 4*A0,−2 − 10*A0,−1 + 57*A0,0 +19*A0,1 − 7*A0,2 + 3*A0,3 − A0,4) >> shift1 (2-7)

h0,0=(−A0,−3 + 4*A0,−2 − 11*A0,−1 + 40*A0,0 +40*A0,1 − 11*A0,2 + 4*A0,3 − A0,4) >> shift1 (2-8)

n0,0=(−A0,−3 + 3*A0,−2 − 7*A0,−1 + 19*A0,0 + 57*A0,1 − 10*A0,2 + 4*A0,3 − A0,4) >> shift1 (2-9)

(3)亚像素点 e0,0, f0,0, g0,0, i0,0, j0,0, k0,0, p0,0, q0,0 的亮度值将由 a0,i, b0,i and c0,i（i=−3…4）应用 8 阶滤波器计算得到，如下所示：

e0,0=(−a0,−3 + 4*a0,−2 − 10*a0,−1 + 57*a0,0 +19*a0,1 − 7*a0,2 + 3*a0,3 − a0,4) >> shift2 (2-10)

f0,0 = (−a0,−3 + 4*a0,−2 − 11*a0,−1 + 40*a0,0 +40*a0,1 − 11*a0,2 + 4*a0,3 − a0,4) >> shift2 (2-11)

g0,0 = (−a0,−3 + 3*a0,−2 − 7*a0,−1 + 19*a0,0 +57*a0,1 − 10*a0,2 + 4*a0,3 − a0,4) >> shift2 (2-12)

i0,0 = (−b0,−3 + 4*b0,−2 − 10*b0,−1 + 57*b0,0 +19*b0,1 − 7*b0,2 + 3*b0,3 − b0,4) >> shift2 (2-13)

j0,0 = (−b0,−3 + 4*b0,−2 − 11*b0,−1 + 40*b0,0 +40*b0,1 − 11*b0,2 + 4*b0,3 − b0,4) >> shift2 (2-14)

k0,0 = (−b0,−3 + 3*b0,−2 − 7*b0,−1 + 19*b0,0 +57*b0,1 − 10*b0,2 + 4*b0,3 − b0,4) >> shift2 (2-15)

p0,0 = (−c0,−3 + 4*c0,−2 − 10*c0,−1 + 57*c0,0 +19*c0,1 − 7*c0,2 + 3*c0,3 − c0,4) >> shift2 (2-16)

q0,0 = (−c0,−3 + 4*c0,−2 − 11*c0,−1 + 40*c0,0 +40*c0,1 − 11*c0,2 + 4*c0,3 − c0,4) >> shift2 (2-17)

r0,0 = (−c0,−3 + 3*c0,−2 − 7*c0,−1 + 19*c0,0 +57*c0,1 − 10*c0,2 + 4*c0,3 − c0,4) >> shift2 (2-18)

1/8 色度插值的算法流程如图 2-2 所示，阴影所示的点为整像素点，其余为亚像素点，整像素点部分未画出。

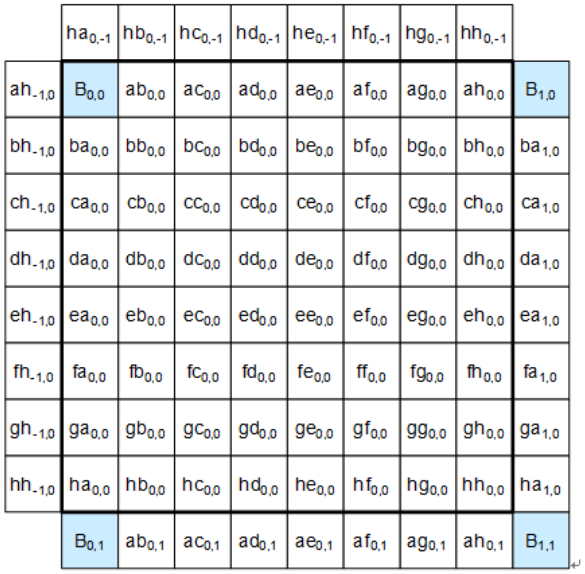


图 2-2 4 阶 1/8 色度插值示意图

1/8 色度插值流程如下所示：

(1)xBi, j = Clip3(0, PicWidthInSamplesC – 1, xIntC +i)

yBi, j = Clip3(0, PicHeightInSamplesC – 1, yIntC +j)

变量 shift1= BitDepthC – 8，shift2= BitDepthC – 2，shift3=14 – BitDepthC。Bi, j 为已知整像素点色度值，亚像素点‘ab0,0’ 到 ‘hh0,0’的色度值按以下步骤得到。

(2)第一行亚像素点 ab0,0, ac0,0, ad0,0, ae0,0, af0,0, ag0,0, ah0,0 的色度值将由临近整像素点的色度值应用 4 阶滤波器得到，如下所示。

ab0,0 = (−3*B−1,0 + 60*B0,0 + 8*B1,0 – B2,0) >> shift1 (2-19)

ac0,0 = (−4*B−1,0 + 54*B0,0 + 16*B1,0 – 2*B2,0) >> shift1 (2-20)

ad0,0 = (−5*B−1,0 + 46*B0,0 + 27*B1,0 – 4*B2,0) >> shift1 (2-21)

ae0,0 = (−4*B−1,0 + 36*B0,0 + 36*B1,0 – 4*B2,0) >> shift1 (2-22)

af0,0 = (−4*B−1,0 + 27*B0,0 + 46*B1,0 – 5*B2,0) >> shift1 (2-23)

$$ag_{0,0} = (-2*B_{-1,0} + 16*B_{0,0} + 54*B_{1,0} - 4*B_{2,0}) >> shift1 \quad (2\text{-}24)$$

$$ah_{0,0} = (-B_{-1,0} + 8*B_{0,0} + 60*B_{1,0} - 3*B_{2,0}) >> shift1 \quad (2\text{-}25)$$

(3)第一列的亚像素点的色度值 $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$, $ha_{0,0}$ 也将由临近整像素点的色度值应用 4 阶滤波器得到，如下所示。

$$ba_{0,0} = (-3*B_{0,-1} + 60*B_{0,0} + 8*B_{0,1} - B_{0,2}) >> shift1 \quad (2\text{-}26)$$

$$ca_{0,0} = (-4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2}) >> shift1 \quad (2\text{-}27)$$

$$da_{0,0} = (-5*B_{0,-1} + 46*B_{0,0} + 27*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (2\text{-}28)$$

$$ea_{0,0} = (-4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (2\text{-}29)$$

$$fa_{0,0} = (-4*B_{0,-1} + 27*B_{0,0} + 46*B_{0,1} - 5*B_{0,2}) >> shift1 \quad (2\text{-}30)$$

$$ga_{0,0} = (-2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2}) >> shift1 \quad (2\text{-}31)$$

$$ha_{0,0} = (-B_{0,-1} + 8*B_{0,0} + 60*B_{0,1} - 3*B_{0,2}) >> shift1 \quad (2\text{-}32)$$

(3)剩下的 7x7 的亚像素值一列一列得计算出来，每次计算 $bX_{0,0}$, $cX_{0,0}$, $dX_{0,0}$, $eX_{0,0}$, $fX_{0,0}$, $gX_{0,0}$,$hX_{0,0}$ 这 7 个值（X= b, c, d, e, f, g ,h）。

$$bX_{0,0} = (-3*aX_{0,-1} + 60*aX_{0,0} + 8*aX_{0,1} - aX_{0,2}) >> shift2 \quad (2\text{-}33)$$

$$cX_{0,0} = (-4*aX_{0,-1} + 54*aX_{0,0} + 16*aX_{0,1} - 2*aX_{0,2}) >> shift2 \quad (2\text{-}34)$$

$$dX_{0,0} = (-5*aX_{0,-1} + 46*aX_{0,0} + 27*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (2\text{-}35)$$

$$eX_{0,0} = (-4*aX_{0,-1} + 36*aX_{0,0} + 36*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (2\text{-}36)$$

$$fX_{0,0} = (-4*aX_{0,-1} + 27*aX_{0,0} + 46*aX_{0,1} - 5*aX_{0,2}) >> shift2 \quad (2\text{-}37)$$

$$gX_{0,0} = (-2*aX_{0,-1} + 16*aX_{0,0} + 54*aX_{0,1} - 4*aX_{0,2}) >> shift2 \quad (2\text{-}38)$$

$$hX_{0,0} = (-aX_{0,-1} + 8*aX_{0,0} + 60*aX_{0,1} - 3*aX_{0,2}) >> shift2 \quad (2\text{-}39)$$

亚像素插值子模块的逻辑流程图如图 2-3 所示。

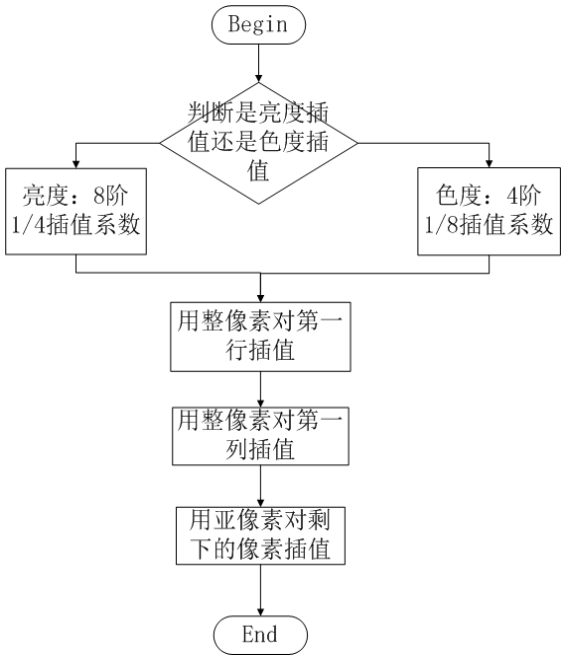


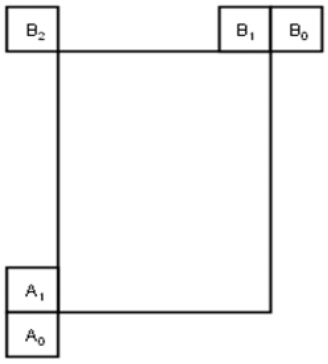
图 2-3 亚像素插值的逻辑流程图

2.1.3 预测运动矢量 MVP 子模块

每个分割运动矢量(MV)的编码需要相当数目的比特，特别是使用小的分割尺寸时。为减少传输比特数，利用临近分割块运动矢量(MV)较强的相关性，可由邻近已编码分割的(MV')预测得到当前块的预测运动矢量(PMV)。计算预测运送矢量 MVP 和运动矢量 MV 的差值（MVD=bestMV-MVP），然后对差值进行编码传输。预测运动矢量 MVP 子模块的功能是为预测单元预测最优的运动矢量，MVP 取决于运动补偿尺寸和邻近 MV 的有无，是实现

帧间预测的重要模块。

预测运动矢量是要根据当前正在编码的预测单元时域-空域上周边预测单元中运动矢量信息进行预测。预测单元 MVP 的空域候选运动矢量块如图 2-4 所示，为 A0、A1、B0、B1、B2 五个候选运动矢量。时域候选运动矢量块如图 2-5 所示，为 H、C3 两个候选运动矢量。



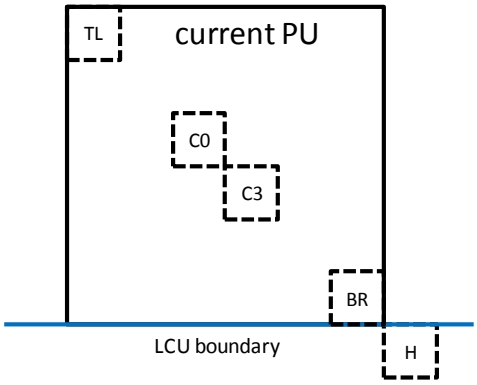


图 2-4 MVP 空域预测候选运动矢量位置[4]　图 2-5 MVP 时域预测候选运动矢量位置

在这 7 个时域空域候选运动矢量中，只有一个运动矢量会最终被确定为 MVP，其整体 MVP 得到流程如图 2-6 所示:

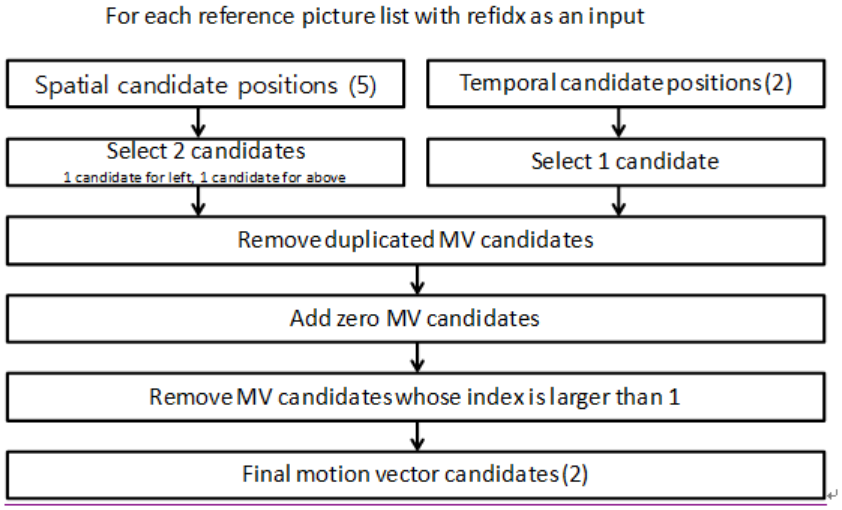


图 2-6 MVP 运动矢量流程图[4]

(1) 首先进行空域候选运动矢量判断。从 5 个空域预测运动矢量中左方选出一个候选运动矢量，上方选出一个候选运动矢量作为 MVP 候选运动矢量列表中成员。MVP 空域预测的 5 个候选运动矢量位置如图 2-4 所示，得到当前预测单元左边空域 MVP 的顺序如下：A0 → A1 → scaled A0 → scaled A1；得到当前预测单元右边空域 MVP 的顺序如下：B0 → B1 → B2 → scaled B0 → scaled B1 → scaled B2。注意：当运动矢量所在 PU 不是帧内编码并且可用时才可以作为 MVP 候选运动矢量。

在做 MVP 选择过程中，先考虑不缩放的情况再考虑缩放的情况。当周围预测单元运动矢量的参考帧 POC 和当前预测单元运动矢量参考帧的 POC 不一致时考虑使用缩放。如果所有的左边候选预测单元都不可用或者是帧内编码，可采用缩放上边候选运动矢量的方法来平衡左边和上边候选运动矢量。否则，不可以对上边运动矢量进行时域缩放。缩放如图 2-7 所示。

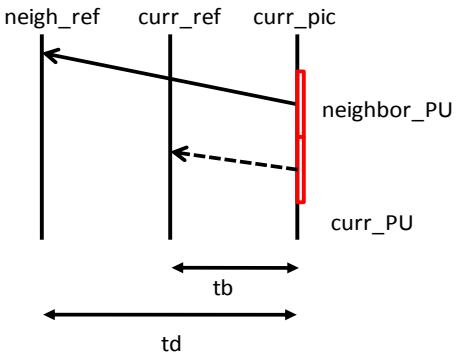


图 2-7 空域预测候选运动矢量时域缩放[4]

(2)如果由空域预测得到的 MVP 标志空域左边候选运动矢量与上边候选运动矢量均为可用且运动矢量不同（（availableFlagLXA==1 ） &&（availableFlagLXB==1 ）&&(mvLXA!=mvLXB)），则时域 colPu 标志为不可用（availableFlagCOL=0），不再进行时域候选运动矢量判断，跳出步骤（2）。否则，进行时域候选运动矢量判断。

时域候选运动矢量的判断如下。时域 co-located PU 有 2 个候选位置，C3 和 H，如图 2-5 所示。首先考虑 H 位置的候选运动矢量作为时域 MVP 候选运动矢量，如果 H 位置的预测单元不可用、帧内编码或者超出了当前 LCU，采用 C3 的位置的运动矢量作为时域 MVP 候选运动矢量。

在得到时域 MVP 候选运动矢量的过程中，缩放比列大小是根据当前图像参考帧列表中离当前图像最近的参考帧中对应 PU 所在的帧确定的。时域候选运动矢量缩放如图 2-8 所示，缩放参考帧对应 PU 的运动矢量用了 POC 距离，tb 和 td。Tb 为当前帧的参考帧和当前帧的 POC 差值，td 是对应帧的参考帧和对应帧的 POC 距离。对于 B 帧，两个运动矢量，一个是参考帧 0 中的运动矢量，一个是参考帧 1 中的运动矢量，得到后组成双向预测 MVP。

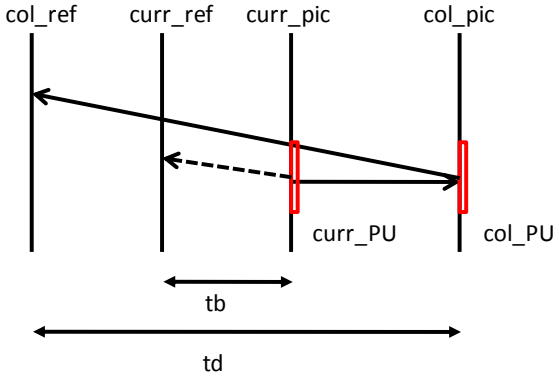


图 2-8 时域预测候选运动矢量时域缩放[4]

变量 colPu 和对应的位置(xPCol,yPCol)得到流程[2]如下：

(a)首先比较 colpic 中的 H 位置处的 PU 是否可用。如果 colpic 中的 H 位置所在的 PU 和当前 PU 在同一个对应的 LCU 中，则 colPu 为覆盖 ((xPRb>>4)<<4,(yPRb>>4)<<4)点的 PU，否则 colPu 标记为不可用。

(b)如果上面比较的 H 位置的 colPu 为帧内预测或者标记为不可用，则应用下面的步骤：1）计算当前 PU 的中心位置 xPCtr=(xP+(nPSW>>1) ,yPCtr=(yP+(nPSH>>1)2)colPu 为 colpic 中覆盖((xPCtr>>4)<<4,(yPCtr>>4)<<4)点的 PU。

(c)(xPCol, yPCol)为 colPu 左上角像素点相对于 colPic 左上角像素点的位置

(3)建立 MVP 候选运动矢量列表，如下所示：

(a) 如果 availableFlagLXA=1，在表中加入 mvLXA；

(b) 如果 availableFlagLXB=1，在表中加入 mvLXB；

(c) 如果 availableFlagLXCol=1，在表中加入 mvLXCol。

(4)如果运动矢量 A 和运动矢量 B 相同（mvLXA== mvLXB），在表中移除运动矢量 B，记 numMVPCandLX 为当前 MVP 表中元素个数，最大 MVP 表元素个数 maxNumMVPCand=2。

(5)修改 MVP 候选表。

(a) 如果当前表中元素个数小于 2，补充零运动矢量至表中，numMVPCandLX = numMVPCandLX + 1 （ if numMVPCandLX<2 ，mvpListLX[numMVPCandLX][0]=0 , mvpListLX[numMVPCandLX][1] = 0)。

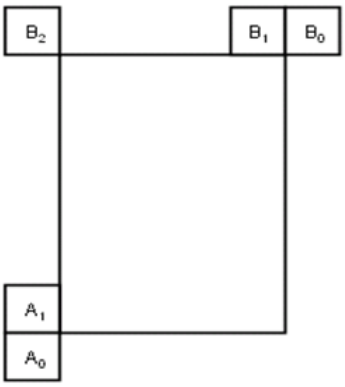
(b)否则（当前表中元素个数大于等于 2），移除表中下标大于 1 的元素。

(6) 最后比较候选列表中 MVP 的 Jpred,SAD，选择最小的一个对应的候选 MVP，被选中的运动矢量索引号放入 mvp_idx_lX 中。

2.1.4 Merge 子模块

在 HEVC 中新增了 Merge 子模块[5]，取代了 H.264 中 SKIP 子模块和 DIRECT 子模块的功能。如果当前编码单元是 SKIP 模式，则通过 Merge 子模块实现，编码端不传送运动矢量残差和像素残差，只传送被选中的 Merge 运动矢量候选列表中的索引号（Merge_idx）至解码端。解码端通过索引号查询到运动矢量并对当前编码单元进行预测。如果当前预测单元被选为 Merge 模式，则通过 Merge 子模块实现，编码端部传送运动矢量残差，只传送像素残差和 Merge_idx 至解码端。解码端通过解码得到 Merge_idx，然后查询到运动矢量，计算出运动矢量对应的 Merge 像素值，然后像素重构值=Merge 像素值+像素残差解码值。

当 PredMode= MODE_SKIP 或者当 PredMode= MODE_ INTER 并且 merge_flag[xP][yP] =1 时，可以采用 Motion Merge 模式。Merge 模式下候选运动矢量列表位置与 MVP 位置相同，如图 2-9 和图 2-10 所示。



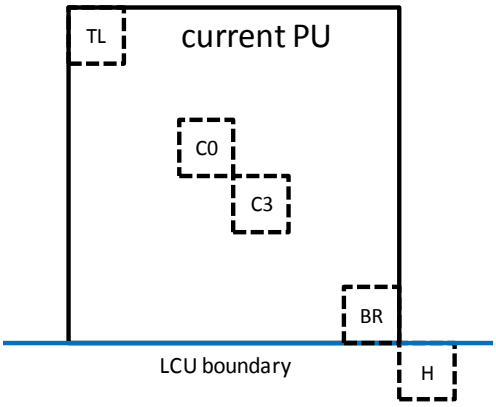


图 2-9 Merge 空域预测候选运动矢量位置 图 2-10 Merge 时域预测候选运动矢量位置

Merge 运动矢量得到流程如图 2-11 所示，共有空域运动矢量、时域运动矢量、组合双向预测运动矢量、零运动矢量 4 种运动矢量。

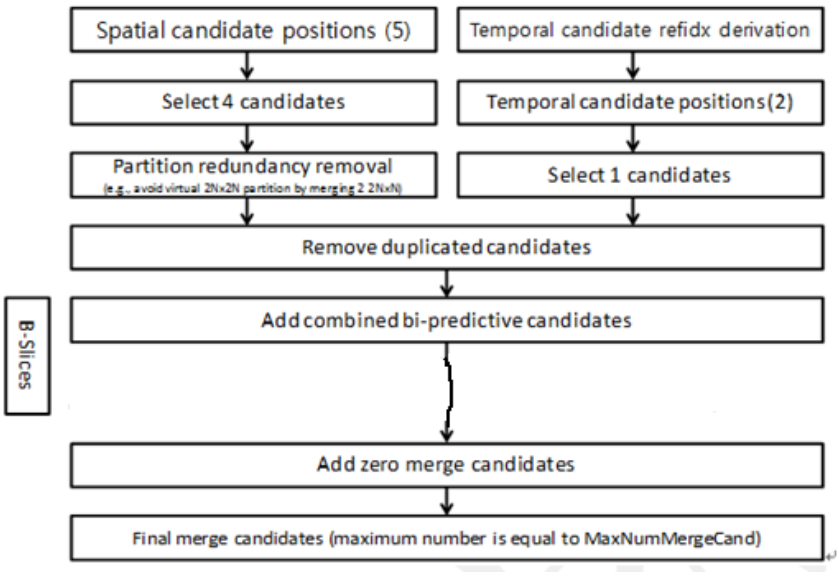
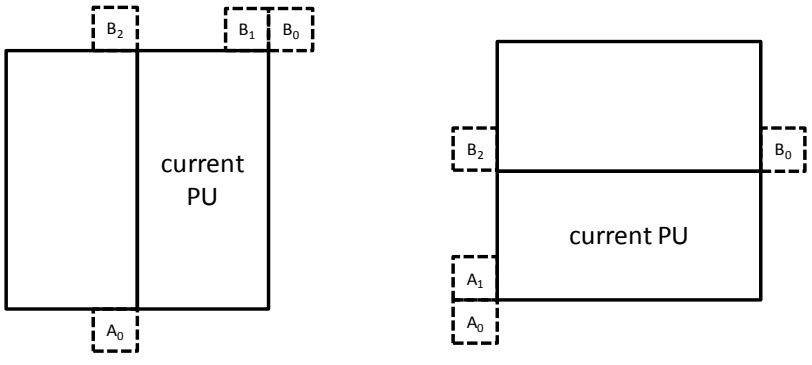


图 2-11 Merge 候选运动矢量得到流程图[4]

(1) 如果 log2_parallel_merge_level_minus2>0&&nCS=8,singleMCLFlag=1;否则 singleMCLFlag=0。

(2) 得到空域候选运动矢量。空域候选运动矢量位置如图 2-9 所示，空域预测要在这 5 个位置中挑选 4 个候选运动矢量，顺序为 A1 → B1 → B0 → A0 → (B2)。当 A1 , B1 , B0 , A0 这四个位置的候选运动矢量任一个无效或者是帧内编码时，才考虑采用 B2 位置的运动矢量。

为了防止伪 2Nx2N 分割的出现，对于 Nx2N, nLx2N and nRx2N 这几个分割当中的第二个预测单元，不使用 A1 位置的运动矢量做为 Merge 的候选运动矢量，所以顺序为 B1 → B0 → A0 → B2。同样的，对于 2NxN, 2NxnU and 2NxnD 分割中的第二个预测单元，不使用 B1 位置的运动矢量做为 Merge 的候选运动矢量，所以顺序为 A1 → B0 → A0 → B2。图 2-12 中指出了对于 Nx2N 和 2NxN 分割中第二个预测单元的 Merge 候选运动矢量的位置。



(a) second PU of Nx2N　　(b)second PU of 2NxN

图 2-12 Nx2N 和 2NxN 分割中第二个 PU 的候选运动矢量位置[7]

具体得到 Merge 空域候选运动矢量中 availableFlagN 的流程如下所示，N = A0, A1, B0, B1 or B 依次进行，如表 2-3 所示。

表 2-3 Merge 空域候选运动矢量得到流程[2]

空域运动矢量索引顺序 (A1 → B1 → B0 → A0 → B2)	位置	(xN , yN)
1	左 Left-down	(xP -1, yP + nPSH -1)
2	上 Above	(xP+nPSW-1, yP – 1)

续表 2-3

空域运动矢量索引顺序 (A1 → B1 → B0 → A0 → B2)	位置	(xN , yN)
3	右上 Above Right	(xP+nPSW , yP – 1)
4	左下 Left bottom	(xP – 1, yP + nPSH)
5	左上 Above left	(xP – 1, yP – 1)
* nPSW x nPSH is the current PU size		

下面的一个条件为真时，令(XN,YN)对应的 PU 的 availableFlagN=0，mvLXN=0.

(a)(XN,YN)与 (XP,YP)在同一个 log2_parallel_merge_level_minus2 语句对应的并行区域内；

(b)A0，A1，B0，B1 均为可用时 (availableFlagA0 + availableFlagA1 + availableFlagB0 + availableFlagB1)=4，B2 标记为不可用（availableFlagB2=0）；

(c)（XN,YN）处对应的 PU 根据 block z-scan process 标记为不可用或者 PU 为帧间编码；

(d)singleMCLFlag=0 并且分割模式为 2NXN,2NXnU,2NXnD 时对于第二个分割块（PartIdx=1）而言，A1 处不可用；

(e)singleMCLFlag=0 并且分割模式为 NX2N,nLx2N,Nrx2N 时对于第二个分割块（PartIdx=1）而言，B1 处不可用；

(f)N=B1，检查 B1 处 mv 和 refIdex 是否和 A1 处相同，相同则不可用；

(g)N=B0，检查 B0 处 mv 和 refIdex 是否和 B1 处相同，相同则不可用；

(h) N=A0，检查 A0 处 mv 和 refIdex 是否和 A1 处相同，相同则不可用；

(i) N=B2，检查 B2 处 mv 和 refIdex 是否和 B1 处相同，相同则不可用；

(j) N=B2，检查 B2 处 mv 和 refIdex 是否和 A1 处相同，相同则不可用；

若以上条件均不为真，则 availableFlagN=1。

(3)得到 Merge 的时域候选运动矢量的参考帧索引号。如果下面条件全部为真，则当前 PU 的参考帧索引为 A1（左下角）所对应的参考帧索引：(a)A1 所在的 PU 是可用的；(b)PartIdx=0，即是分割模式下第一个分割块；(c)预测模式不是帧内预测；(d)当前 PU 和 A1 所在的 PU 不在同一个 log2_parallel_merge_level_minus2 所对应的并行区域内。否则参考帧索引 refIdx=0。

(4)得到 Merge 时域候选运动矢量，同 2.1.3 章节中得到 MVP 时域候选运动矢量流程相同。

(5)建立 Merge 运动矢量列表。numMergeCand=numOrigMergeCand。

(6)如果是 B 帧，且 numMergeCand<MaxNumMergeCand(MaxNumMergeCand=5)，则在 Merge 列表中增加由 Merge 列表中已存在运动矢量组成的组合双向预测运动矢量[6]，如图 2-13 所示。

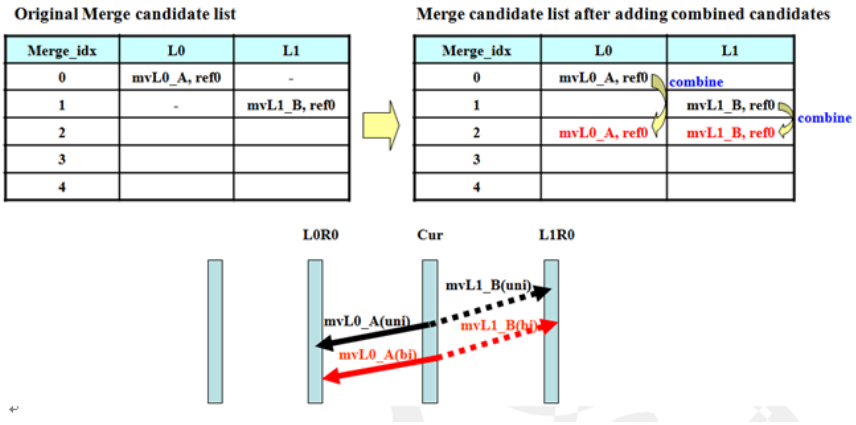


图 2-13 Merge 组合双向预测运动矢量[6]

Merge 的组合双向预测候选运动矢量得到流程如下：当 numOrigMergeCand=2,3,4 时，numInputMergeCand= numOrigMergeCand，combIdx=0,combCnt=0，combSTOP=FALSE，下面开始循环直到 combSTOP=TURE。

(a)由 combIdx 的值推倒出双向预测候选运动矢量的来源 l0CandIdx 和 l1CandIdx，对应表 2-4。也就是说添加的组合双向预测候选运动矢量按顺序来源于 list0 和 list1 中对应候选运动矢量组合 索引值 (0,1)→(1,0)→(0,2)→(2,0)→(1,2)→(2,1)→(0,3)→(3,0)→(1,3)→(3,1)→(2,3)→(3,2)。

表 2-4 Merge 组合双向候选运动矢量

Comb_Merge_Idx	L0_Cand_Idx	L1_Cand_Idx
0	0	1
1	1	0
2	0	2
3	2	0
4	1	2
5	2	1
6	0	3
7	3	0
8	1	3
9	3	1
10	2	3
11	3	2

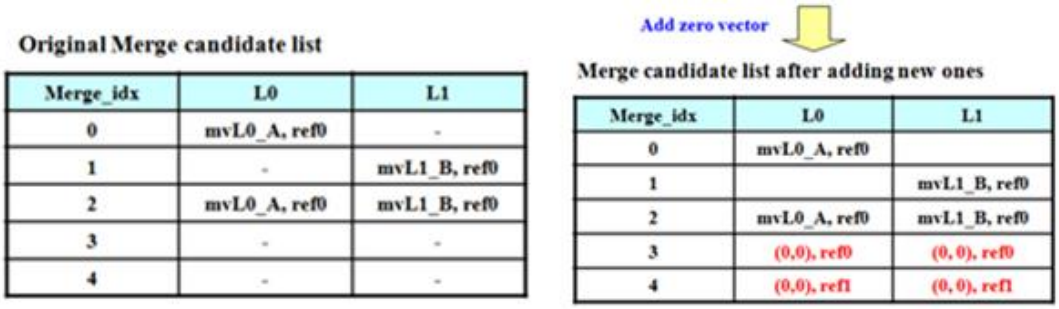
(b)根据索引值得到在 MergeCandList 中得到对应的运动矢量。l0Cand=MergeCandList[l0CandIdx]，l1Cnd=MergeCandList[l1CandIdx]。

(c)当参考帧 0 可用并且参考帧 1 可用且参考帧 0 与参考帧 1 不是同一副图片或者运动矢量不同时((predFlagL0l0Cand = = 1) &&(predFlagL1l1Cand = = 1)&& (PicOrderCnt(RefPicListL0(refIdxL0l0Cand)) != PicOrderCnt(RefPicListL1(refIdxL1l1Cand)) | | mvL0l0Cand != mvL1l1Cand))，将运动矢量 combCandk 添加到 mergeCandList 尾部，numMergeCand= numMergeCand+1。

(d)combIdx++。

(f)当可以添加的 Merge 超过了现在 Merge 列表中可以提供的最大组合双向预测运动矢量的时候（combIdx=(numOrigMergeCand*(numOrigMergeCand−1))）或者 numMergeCand 已经达到最大数目 MaxNumMergeCand 或者 combCnt=5 时， combSTOP=TRUE，此流程停止。

(7) 对于 P 帧和 B 帧，在候选列表结尾处都将添加零运动矢量，零运动矢量如图 2-14 所示。



Merge_idx	L0	L1
0	mvL0_A, ref0	-
1	-	mvL1_B, ref0
2	mvL0_A, ref0	mvL1_B, ref0
3	-	-
4	-	-

Merge_idx	L0	L1
0	mvL0_A, ref0	
1		mvL1_B, ref0
2	mvL0_A, ref0	mvL1_B, ref0
3	(0,0), ref0	(0, 0), ref0
4	(0,0), ref1	(0, 0), ref1

图 2-14 Merge 零候选运动矢量示意图[6]

Merge 的零候选运动矢量得到流程[2]如下：

当 Merge 候选列表中候选运动矢量的个数没有达到最大值时（umMergeCand < MaxNumMergeCand）,开始考虑 Merge 零候选运动矢量。令 numInputMergeCand= numMergeCand， zeroStop=FALSE。开始循环下面流程直至 zeroStop= TRUE ，此流程终止。

(a)首先要得到参考帧索引值 refIdx、预测列表使用标志 predFlag、零 merge 候选运动矢量值 mv。对于 P 帧：list1 对应的标志值记为-1 或者 0，list0 对应的标记值 refIdx、predFlag、mv 进行相应的标志。numMergeCand++。对于 B 帧：list0 和 list1 对应的标记值 refIdx、predFlag、mv 进行相应的标志。numMergeCand++。

(b)zeroIdx++

(c)当现在 Merge 候选列表中元素个数达到最大 Merge 候选个数时（numMergeCand =MaxNumMergeCand）或者当前获取的零运动矢量已经超过可以获取的最大零运动矢量时（zeroIdx>=numRefIdx），令 zeroSTOP=TRUE，结束零运动矢量获取流程。

零运动矢量获取是按照表 2-5 进行的。

表 2-5 Merge 候选零运动矢量

zero_Merge_Cand	L0_index	L1_index
M=0	0	0
M=1	1	1
M=2	2	2
M=3	3	3

(8)在任何一个阶段，如果候选列表中的候选运动矢量的数目到达 MaxNumMergeCand（MaxNumMergeCand=5），则终止。

(9)如果 PredMode =MODE_SKIP，比较候选列表中的运动矢量的对应的代价 Jmode，然后选择其中最小的一个，把索引号赋值给 merge_idx，skip_flg=1。

(10)如果 PredMode = MODE_INTER 并且 mergeflag=1，比较列表中所有候选 Merge 运动矢量的代价 Jpred,SATD，选择最小的一个，把索引号赋值给 merge_idx。

2.2 HEVC 参考编码器 HM6.1 的帧间预测函数架构和算法

在HEVC的HM6.1编码器中对于一个LCU(64x64块像素点)的分割模式的判断是在函数compressCU中实现的。图2-15中的函数compressCU为顶层函数compressSlice所调用，功能是对编码单元进行编码，输入为TComDataCU的数据类型。在函数compressCU中首先调用了initCU函数对当前LCU对应的TEncCU类型中数据m_ppcBestCU[0]和m_ppcTempCU[0]进行

初始化；然后调用了xCompressCU函数对当前LCU进行编码；最后调用函数xLcuCollectARLStates函数进行自适应QP判断。其中运动估计和模式判别的实现是在递归函数xCompressCU中完成的。

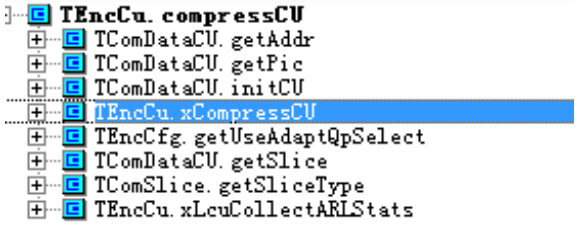


图2-15 函数compressCU结构图

2.2.1 帧间模式判别函数架构及算法

上层函数compressCU调用函数xCompressCU中实现分割模式、预测模式的具体判断。函数xCompressCU是一个递归函数。xCompressCU函数实现了一个LCU（64x64）的模式判别（skip、inter、intra），及每种模式下分割方式的判别。为了实现不同大小CU的编码（64x64、32x32、16x16），xCompressCU在每层CU的模式判别完后会递归调用4次xCompressCU函数对当前CU的4个子CU进行模式判别。

函数xCompressCU中调用的和模式判别相关的主要函数如图2-16所示，在xCompressCU中实现的模式判别的算法框图如图2-17所示。

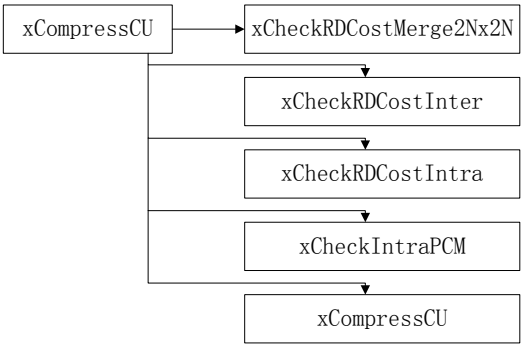


图2-16 函数xCompressCU中调用的和模式判别相关的主要函数

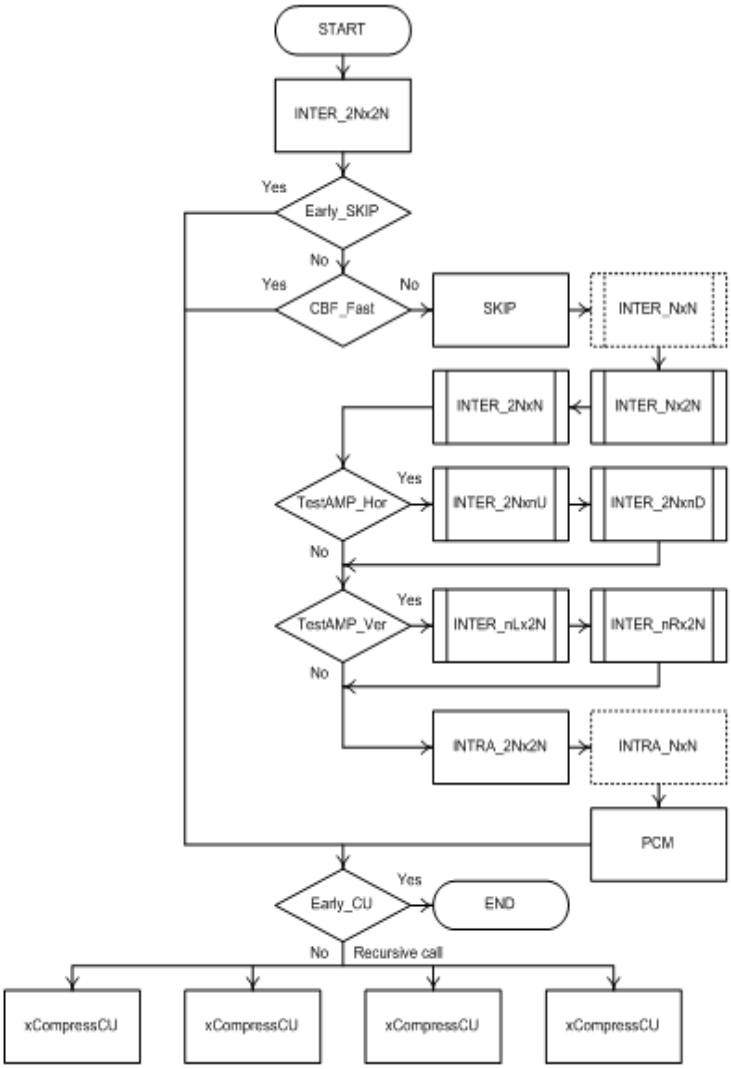


图2-17 函数xCompressCU算法流图[3]

模式判别实现算法流程如下所示：

（1）前期准备，对于模式判别及帧内帧间搜索所需的元素赋初值。

（2）检查SKIP模式，主要在函数xCheckRDCostMerge2Nx2N中实现。如果在当前LCU中与当前CU同样大小的CU采用skip的个数多于5个并且SKIP的cost小于阈值（aiNum [iIdx] > 5 && fRD_Skip < EARLY_SKIP_THRES*afCost[iIdx]/aiNum[iIdx]），则进行提前终止，不再向下一层进行分割搜索（bEarlySkip = true，bTrySplit = false）。

（3）检查INTER_2Nx2N分割模式，主要在函数xCheckRDCostInter中实现。判断是否进行PU分割标志位doNotBlockPu。

（4）依次判断INTER_NxN, INTER_Nx2N, INTER_2NxN分割模式，分别在函数xCheckRDCostInter中实现。

（5）检查AMP分割模式下，如下： SIZE_2NxnU, SIZE_2NxnD, SIZE_nLx2N, SIZE_nRx2N，分别在函数xCheckRDCostInter中实现。

（6）检查INTRA模式，主要在函数xCheckRDCostIntra中实现。

（7）检查PCM模式，主要在函数xCheckIntraPCM中实现。

（8）根据bSubBranch判断是否进行下一层CU的分割。如果进行下一层CU的分割，则递归4次调用函数xCompressCU对下一层4个子CU进行编码。

（9）对于当前CU模式判别得到的数据值及其子分割的数据值进行后期数据处理。

2.2.3 INTER 模式下的函数架构及运动估计算法说明

(1)总体函数架构

帧间预测功能由函数xCheckRDCostInter完成，xCheckRDCostInter的函数说明如表2-6所示，主要函数结构图如图2-18所示。

表2-6 xCheckRDCostInter函数说明

输入	输出	功能
TComDataCU*& rpcBestCU, TComDataCU*& rpcTempCU, PartSize ePartSize	Null	通过输入变量ePartSize指定分割块的分割类型，然后对当前编码单元进行帧间预测，计算残差和率失真代价，并比较保存最优结果至变量rpcBestCU中。

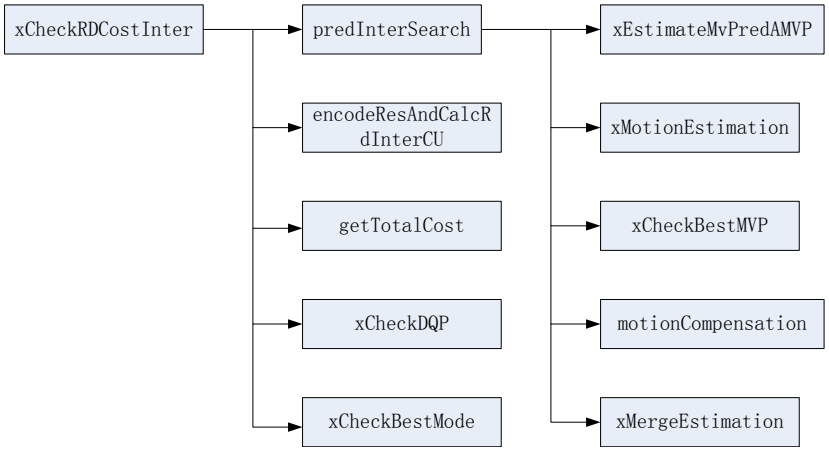


图2-18 函数xCheckRDCostInter调用的主要函数结构图

在函数xCheckRDCostInter中完成了分割块的预测、残差代价计算、比较更新最优结果等功能。其中帧间预测主要在predInterSearch函数中完成；在通过函数predInterSearch得到当前分割模式下的最优预测方式后计算残差和率失真代价在函数encodeResAndCalcRdInterCU中完成；最后检验是否当前尝试为最优结果，通过函数xCheckBestMode完成。

函数predInterSearch主要功能为完成帧间预测，包括检验Merge预测方式、运动估计、运动补偿等。其中函数xEstimateMvPredAMVP主要功能是得到预测运动矢量MVP，为计算残差（MVD=MV-MVP）做准备；函数xMotionEstimation完成了运动估计功能，通过运动搜索得到了较优的运动矢量；函数xMergeEstimation检查Merge模式下的5个运动矢量和当前运动矢量得到，更新保留其中最好的一个结果；motionCompensation函数完成运动补偿功能。

函数xEstimateMvPredAMVP的主要算法说明在2.1.3章预测运动矢量MVP子模块中。函数xMotionEstimation的说明如表2-7所示，函数架构如图2-19所示。

表2-7 xMotionEstimation函数说明

输入	输出	功能
TComDataCU* pcCU, TComYuv* pcYuvOrg, Int iPartIdx, RefPicList eRefPicList, TComMv* pcMvPred, Int iRefIdxPred, TComMv& rcMv, UInt& ruiBits, UInt& ruiCost, Bool bBi	Null	为当前分割块完成在一帧中的运动搜索的功能，eRefPicList和iRefIdxPred表示参考帧列表号和帧在列表中的索引号。pcMvPred为预测运动矢量的值，rcMv是搜索得到的最优运动矢量的值，iPartIdx是当前PU在CU中的索引号，ruiBits和 ruiCost分辨表示最优运动矢量的比特数和运动估计代价值。

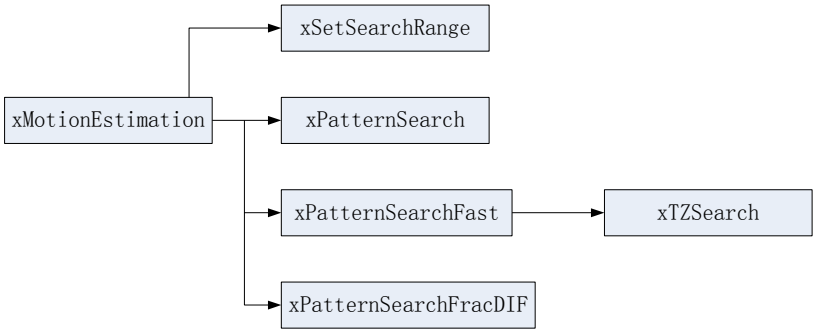


图2-19 函数xMotionEstimation结构图

运动估计分为整像素运动估计和亚像素运动估计。在函数xMotionEstimation中整像素运动估计有2种模式，通过变量m_iFastSearch控制，如果m_iFastSearch=0，进行全局搜索；如果m_iFastSearch=1，进行钻石搜索。具体算法流程如下：

（1）进行运动估计所需变量初始化，其中设置搜索范围是一个比较重要的工作，由函数xSetSearchRange完成。在HM6.1最高层cfg文件里设置SearchRange=64、BipredSearchRange=4：对于单向搜索来说，在实现过程中采用了自适应搜索范围设置，整像素点搜索范围iNewSearchRange=clip3(8,64,64x1x(currGOP-refGOP)/8)；对于双向搜索，整像素点搜索范围为4。

（2）如果m_iFastSearch=0或者是双向搜索（!m_iFastSearch || bBi），则进行全局整像素搜索，由函数xPatternSearch完成，跳过步骤3。

（3）进行快速整像素搜索（m_iFastSearch=1），由函数xPatternSearchFast完成。在函数xPatternSearchFast中，函数xTZSearch完成了快速整像素运动估计的主要功能。

（4）进行亚像素运动估计，由函数xPatternSearchFracDIF完成。

(2) 整像素运动估计算法说明

在HEVC参考编码器HM6.1中整像素运动搜索分为全局运动搜索和快速运动搜索两种，分别在函数xPatternSearch和函数xPatternSearchFast中完成运动搜索的功能。

A. 整像素全局运动搜索 xPatternSearch 函数架构及算法说明

在函数xPatternSearch中完成了全局搜索，函数说明如表2-8所示。

表2-8 xPatternSearch函数说明

输入	输出	功能
TComPattern* pcPatternKey, Pel* piRefY, Int iRefStride, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB, TComMv& rcMv, UInt& ruiSAD	Null	以rcMv为中心，左上运动矢量pcMvSrchRngLT和右下运动矢量pcMvSrchRngRB为框，对其中的运动矢量进行全局运动搜索，得到运动矢量更新至rcMV中，其SAD值保存在变量ruiSAD中。

在全局整像素运动搜索中采用了一种下采样快速算法来计算SAD值。当iSubShift=0，不使用下采样的方法计算SAD值；当iSubShift=1，采用1/2下采样计算SAD值；当iSubShift=2时，采用1/4下采样计算SAD值。

当PU块的行数大于8时（iRows>8），令变量iSubShift = 1，则采用每2行去一行数据的方式来快速计算SAD值，详细过程在setDistParam函数中指定的SAD计算函数中实现。

例如：如果对于16x32的PU进行整像素全局运动估计，则在计算过程中由于iSubShift=1，只取1,3,5,7,9,11,13,15这8行32列数据进行SAD计算，在得到这1/2的SAD值后再乘以2得到近似总SAD值。

B. 整像素快速运动搜索 xPatternSearchFast 函数架构及算法说明

函数xPatternSearchFast实现的功能是整像素快速运动估计，当m_iFastSearch=1时，调用函数xTZSearch完成整像素快速运动估计的功能。函数xTZSearch完成了整像素运动估计算法功能，函数功能如表2-9所示。其函数架构如图2-20所示，主要调用的函数有xTZSearchHelp、xTZ2PointSearch、xTZ8PointDiamondSearch、xTZ8PointSquareSearch。

表2-9 xTZSearch函数说明

输入	输出	功能
TComDataCU* pcCU, TComPattern* pcPatternKey, Pel* piRefY, Int iRefStride, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB, TComMv& rcMv, UInt& ruiSAD	Null	以rcMv为中心，左上运动矢量pcMvSrchRngLT和右下运动矢量pcMvSrchRngRB为框，对其中的运动矢量进行快速运动搜索（以中心点rcMV为其实搜索点），得到运动矢量更新至rcMV中，其SAD值保存在变量ruiSAD中。

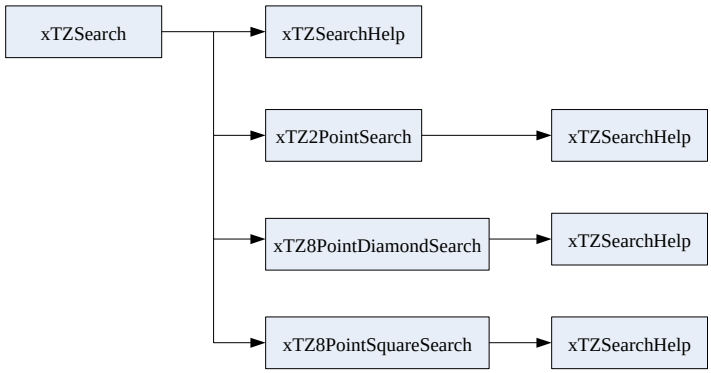


图2-20 函数xTZSearch调用的主要函数结构图

函数xTZSearch的算法流程如下所示：

（1）首先确定起始搜索点的位置。

分别按顺序比较中心点运动矢量（rcMV）、A的位置运动矢量、B位置运动矢量、C位置运动矢量、（0,0）运动矢量，从中选出SAD+mv_cost值最小的一个作为起始搜索点，调用函数函数xTZSearchHelp，令当前点为搜索中心点（uiBestDistance=0、ucPointNr=0）。A、B、C的位置如图2-21所示。

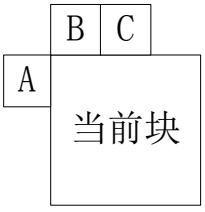


图2-21 候选块A、B、C与当前块的位置

（2）进行运动估计第一步粗搜索。

(a)令iDist=1,2,4,8……uiSearchRange，循环进行8点大钻石搜索或者8点大方形搜索，如图2-22所示，分别调用函数xTZ8PointDiamondSearch、函数xTZ8PointSquareSearch。提前终止条件是进行过3次及3次以上8点大钻石搜索或者8点大方形搜索（cStruct.uiBestRound >= uiFirstSearchRounds=3）。

注意：在函数中设置bFirstSearchDiamond=1，关闭了方形搜索，仅剩钻石搜索。

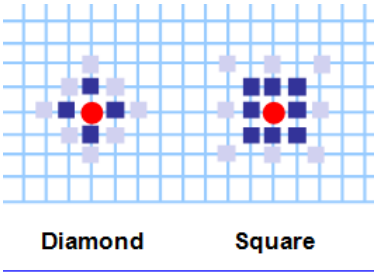


图2-22钻石搜索和方形搜索示意图

(b)如果当前最优运动矢量距离中心搜索点的距离为1（uiBestDistance=1），则以当前最优运动矢量作为中心搜索点（uiBestDistance=0），进行周围2个点的搜索，调用函数xTZ2PointSearch；否则跳过步骤(b)，进行步骤(d)。

(c)如果当前最优运动矢量距离中心搜索点的距离大于5（uiBestDistance>5），则对搜索框内以5为步伐进行光栅搜索，否则跳过步骤4。

图2-23为光栅扫描示意图，但在此步骤中采用的间隔为5。对于每个点调用函数xTZSearchHelp，从中选出其中SAD+mv_cost值最小的一个作为当前最优运动矢量，距离中心搜索点的距离记为5，不再进行周围2个点的搜索。（uiBestDistance=5，ucPointNr=0）。

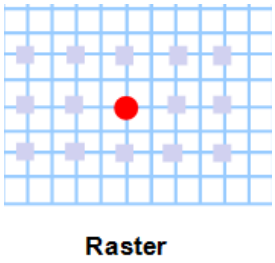


图2-23 光栅扫描示意图

（3）进行运动估计的第二步细搜索。

如果当前最优运动矢量不是中心搜索点（uiBestDistance>0），则进行细微修正；否则跳过步骤(3)。步骤(3)的结束要求最优运动矢量为中心搜索点（uiBestDistance=0），并且经过当前最优运动矢量周围2点搜索（函数xTZ2PointSearch）后，最优运动矢量仍为中心搜索点（ucPointNr=0）。

程序流程图如图2-24所示。详细步骤如下所示：

(a)令当前最优运动矢量作为中心搜索点（uiBestDistance=0、ucPointNr=0）。

(b)再循环进行8点大钻石搜索（iDist=1,2,4,8,…,uiSearchRange+1），调用函数xTZ8PointDiamondSearch。

(c)如果当前最优运动矢量距离中心搜索点的距离为1（uiBestDistance=1），令当前最优运动矢量为中心搜索点（uiBestDistance=0）。如果ucPointNr！=0，则进行周围2个点的搜索，调用函数xTZ2PointSearch。如果当前最优运动矢量是不是中心搜索点，（uiBestDistance>1）返回1）继续循环；否则，跳出步骤5。

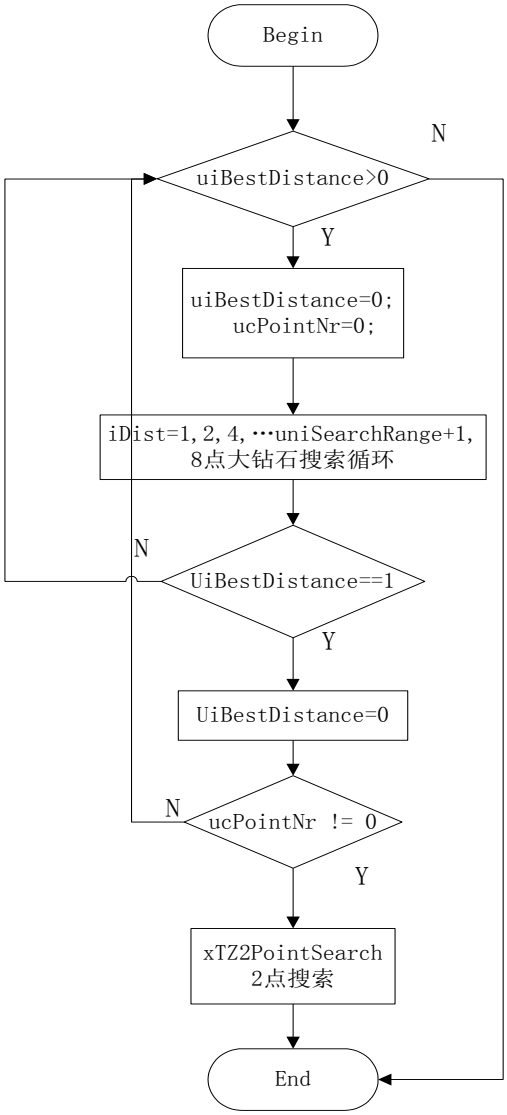


图2-24 函数xTZSearch中运动估计第二步细搜索程序流程图

C. 整像素快速运动搜索中估计一些函数说明

a. 函数 xTZSearchHelp

函数xTZSearchHelp完成了一个整像素运动矢量的比较，如表2-10所示。

表2-10 xTZSearchHelp函数说明

输入	输出	功能
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct, const Int iSearchX, const Int iSearchY, const UChar ucPointNr, const UInt uiDistance	Null	完成一个整像素运动矢量的比较：比较结构体rcStruct中运动矢量和运动矢量（iSearchX，iSearchY），从中选出SAD+mv_cost较小的一个记为当前最优运动矢量，并保存至rcStruct中，并相应更新结构体rcStruct中的SAD值，把参数ucPointNr和参数uiDistance更新至结构体rcStruct中。

在函数xTZSearchHelp中同xPatternSearch一样也采用了下采样的方法，当PU块的行数大于8时（iRows>8），令变量iSubShift = 1，则采用每2行去一行数据的方式来快速计算SAD值。

b. 函数 xTZ2PointSearch

函数xTZ2PointSearch完成了2个整像素运动矢量的比较，如表2-11所示。

表2-11 xTZ2PointSearch函数说明

输入	输出	功能
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB	Null	完成2个整像素运动矢量的比较：根据结构体rcStruct中变量ucPointNr的值比较2个不同位置的运动矢量和结构体rcStruct中运动矢量，取SAD+mv_cost小的一个，其中pcMvSrchRngLT和pcMvSrchRngRB分别表示搜索框左上角与右下角运动矢量，超出搜索框的运动矢量不进行搜索。

函数xTZ2PointSearch中2个整像素运动矢量搜索点位置的确定是由变量ucPointNr指定的。假设结构体rcStruct中指定的当前运动矢量为（x，y）：当ucPointNr=1时，2个待比较的整像素运动矢量为（x-1，y）、（x，y-1）；当ucPointNr=2时，2个待比较的整像素运动矢量为（x-1，y-1）、（x+1，y-1）；当ucPointNr=3时，2个待比较的整像素运动矢量为（x，y-1）、（x+1，y）；当ucPointNr=4时，2个待比较的整像素运动矢量为（x-1，y+1）、（x-1，y-1）；当ucPointNr=5时，2个待比较的整像素运动矢量为（x+1，y-1）、（x+1，y+1）；当ucPointNr=6时，2个待比较的整像素运动矢量为（x-1，y）、（x，y+1）；当ucPointNr=7时，2个待比较的整像素运动矢量为（x-1，y+1）、（x+1，y+1）；当ucPointNr=8时，2个待比较的整像素运动矢量为（x+1，y）、（x，y+1）。如图2-25所示。

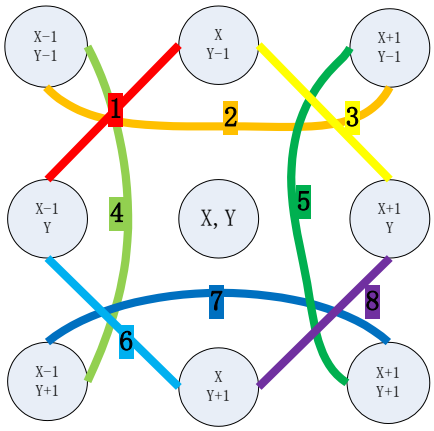


图2-25 函数xTZ2PointSearch示意图（图中数字为ucPointNr的值）

经过函数xTZ2PointSearch后，若最优运动矢量发生变化，为搜索过的这2个运动矢量之一，则令uiBestDistance=2，ucPointerNr=0，表示当前最优运动矢量距离中心搜索点为2，不需要再进行周围2点整像素运动矢量搜索。

c．函数 xTZ8PointDiamondSearch

函数xTZ8PointDiamondSearch完成了自适应8点大钻石搜索功能，如表2-12所示。

表2-12 xTZ8PointDiamondSearch函数说明

输入	输出	功能
TComPattern* pcPatternKey, IntTZSearchStruct& rcStruct, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB, const Int iStartX, const Int iStartY, const Int iDist	Null	完成自适应8点钻石搜索：结构体rcStruct保留当前最优运动矢量的信息，以运动矢量(iStartX, iStartY)为中心，以iDist为半径进行自适应8点大钻石搜索，取SAD+mv_cost小的一个留存信息至rcStruct中。其中pcMvSrchRngLT和pcMvSrchRngRB分别表示搜索框左上角与右下角运动矢量，超出搜索框的运动矢量不进行搜索。

函数算法流程如下：

（1）如果当前搜索半径iDist=1,则以运动矢量(iStartX, iStartY)为中心，以1为半径进行上下左右的小菱形搜索，搜索位置如图2-26所示。

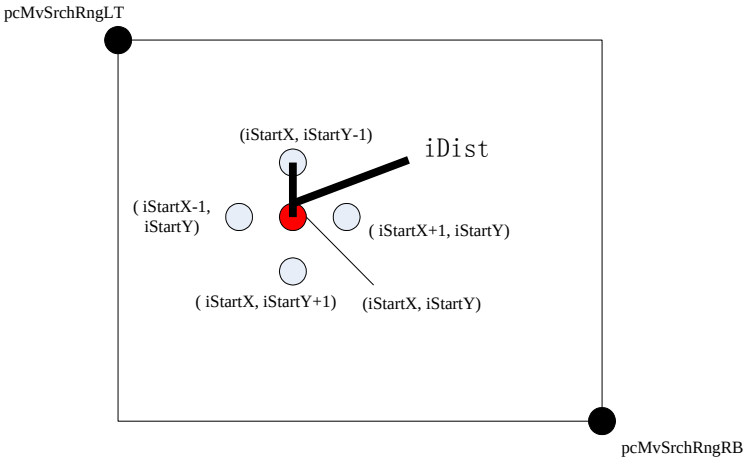


图2-26 iDist=1时，自适应钻石搜索——小菱形搜索

若搜索结果为(iStartX, iStartY-1) 、(iStartX-1, iStartY)、(iStartX+1, iStartY)、(iStartX, iStartY+1)四点之一，则令uiBestDistance=1，表示当前最优运动矢量和中心搜索点的距离为1，ucPointNr分别为2、4、5、7，与图2-25中ucPoint的位置相对应。

（2）当搜索半径iDist=2、4、8时，则以运动矢量(iStartX, iStartY)为中心，以iDist为最大半径进行大钻石搜索，搜索位置如图2-27所示。

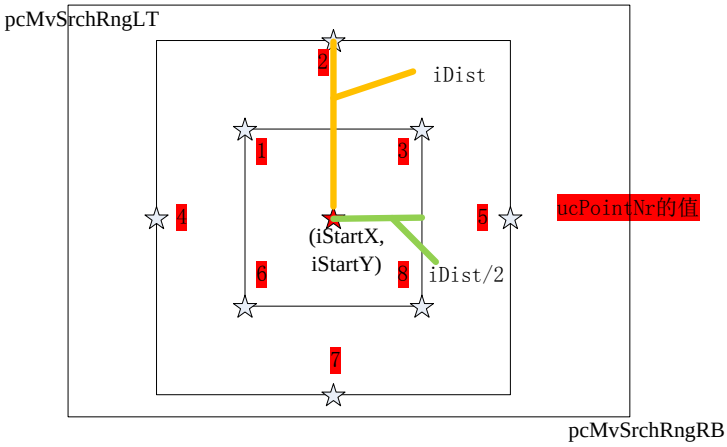


图2-27 iDist=2、4、8时，大钻石搜索

大钻石搜索分别以iDist为半径和iDist>>1为半径进行了8个点的搜索，如图3-14所示。由图3-14和图3-12可知，当进行细化搜索时，先进行8点大钻石搜索再进行2点周边搜索，则2点的位置与这钻石搜索得到的最优运动矢量的方向有关，如图2-28所示。

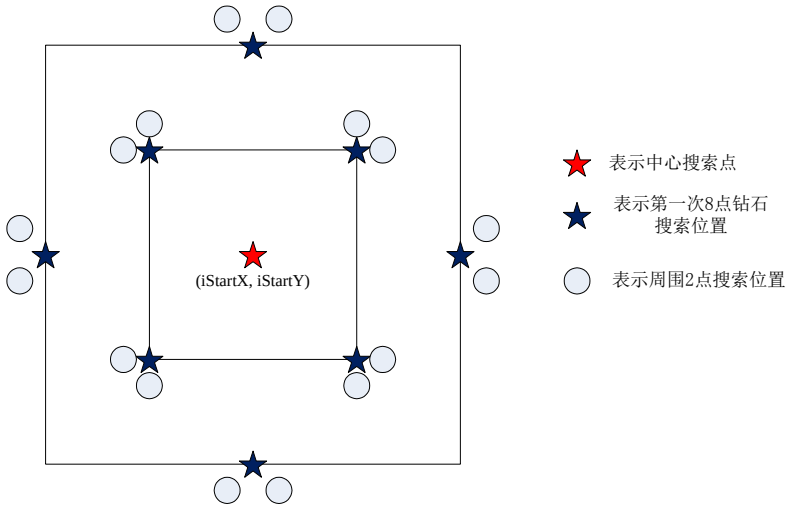


图2-28 根据钻石搜索方向（ucPointNr）得到的周围2点搜索搜索位置

（3）当搜索半径大于8时（iDist>8）,进行变半径大菱形搜索，搜索位置如如图2-29所示。

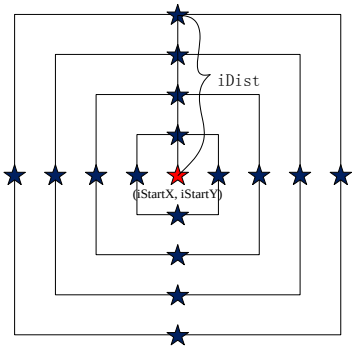


图2-29 大菱形搜索

2.3 本章小结

本章首先介绍了 HEVC 采用的新的帧间预测编码技术：基于 8 阶的 1/4 亮度插值和基于 4 阶的 1/8 色度插值、修改的 MVP 子模块、融合 SKIP 和 DIRECT 的 Merge 子模块等。这些新的帧间预测技术有效地降低了码率，尤其是 MVP 和 Merge 候选运动矢量位置的确定，采取了空域-时域预测、组合双向预测、零矢量预测等方法，对于去除时间域冗余信息具有良好的效果。

然后介绍了 HEVC 参考编码器 HM6.1 的帧间预测算法。在 HM6.1 编码器中帧间预测算法采取了很多有效减少计算量和提高精确度的算法，譬如在整像素运动搜索时采取了下采样的方法减少运算量、自适应半径钻石搜索提高了搜索精确度、光栅搜索预防过大运动矢量、粗搜索后细搜索进一步提高运动估计精确度；在模式判别中采用了 SKIP 模式部分提前终止、预测单元分割判断等大大提高了编码性能。

本毕设中研究的实时编码器帧间预测算法和基于 HM 框架的改进帧间预测算法就是基于这个帧间模块的架构进行研究的。

第三章 HEVC 帧间预测改进算法

在 HEVC 参考编码器 HM6.1 中提出了很多提高运动矢量精度、压缩码率的有效方法，但是同时过于复杂的帧间预测算法也使得编码复杂度大大增加，编码效率大大降低。

为了大幅度降低帧间预测模块运算量、实现 HEVC 高效实时编码器，本毕设基于 H.264 实时编码器的帧间预测算法[11]、HM6.1 帧间预测算法、HEVC 帧间编码标准[2]，设计出了用于实时编码器的帧间预测算法，大大降低了帧间预测环节的运算量。但是由于用于实时编码器的帧间预测模块较复杂，在现有的 HM 平台上不能完全验证，需要在新的编码框架下实现，我将和其他同学在硕士阶段继续完成这一部分工作。

为了验证用于实现实时编码器的帧间预测算法性能，本毕设设计出了在 HM6.1 编码器平台上的帧间预测改进算法，在 HM6.1 平台上部分实现了用于实时编码器的帧间预测算法，大大降低了帧间预测模块的运算量。

3.1 用于实时编码器的帧间预测算法

3.1.1 P 帧运动估计搜索算法

P 帧的运动估计搜索算法分为两种，一种是对于 64x64 的编码单元；另一种是对 64x64 以下的编码单元（32x32、16x16），以及每个编码单元下预测单元的划分。

注意：在修改的帧间预测算法中帧间预测最小分割单元为 8x8，不再进行 8x8 以下的帧间分割。

(1)64x64 编码单元的运动估计

64x64 编码单元的运动估计步骤如下所示：

（1）对于 Merge 的 5 个运动矢量（从 MergeCandList 中去除）进行代价计算，从中选出代价最小的一个作为 Merge 模式下的 MV，索引号记为 Merge_idx。详细流程见 2.1.4 章 Merge 子模块。

（2）候选运动矢量及起始搜索点的选择。起始搜索点位置如下所示：

(a)Skip MV：将步骤（1）中确定的 Merge 模式下的 MV——也是 SKIP 模式下的 MV 放入 MECand_List 中。

(b)MVP：从 MvpCandList（见 2.1.3 章预测运动矢量 MVP 子模块）中 2 个元素中选出代价较小的一个运动矢量 bestMvpCand，索引号记为 mvp_flag，记为 MVP。如果 MVP 与 Skip MV 不同，放入 MECand_List 当中，否则不放入 MECand_List 当中。

(c) Skip MV surrounding：Skip MV取整像素之后的运动矢量为中心，上、下、左、右各偏移一个整像素构成小菱形的四个整像素的运动矢量（0，-1），（0,1），（-1,0），（1,0）加入 MECand_List当中，如图3-1所示。

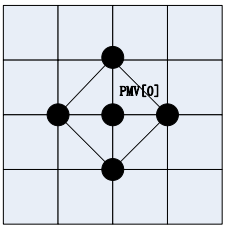


图3-1 SKIP MV周围四个整像素运动矢量

(d) MVP surrounding： MVP 取整像素之后的运动矢量为中心，上、下、左、右各偏移一个整像素构成小菱形的四个整像素的运动矢量（0，-1），（0,1），（-1,0），（1,0）加入 MECand_List 中，如图 3-20 所示。

(e)A、B、C 位置运动矢量：当前预测单元周边三个预测单元块，如图 3-2 所示，放入 MECand_List 中。

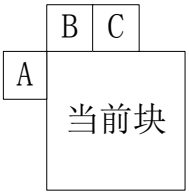


图3-2 候选块A、B、C与当前块的位置

(f)运动矢量（0,0）和时域相关性运动矢量。时域相关性运动矢量为 Colocated 编码单元与其上下左右四个编码单元，位置如图 3-3 所示。共 6 个运动矢量加入 MECand_List 中。

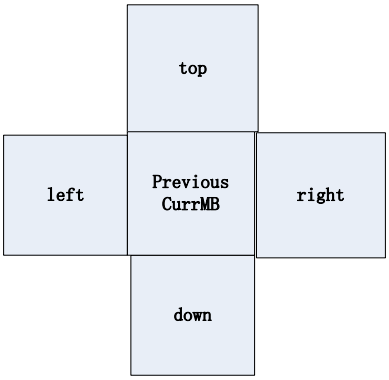


图 3-3 时域相关性运动矢量

（2）计算Skip_mv_cost,如果 Skip_mv_cost<阈值T1，PredMode=SKIP，不再进行运动估计搜索；否则PredMode=INTER，继续进行步骤（3）。

（3）利用hash方法从以上去除Skip MV和MVP的候选运动矢量中（最多有17个运动矢量）选择4个，计算6个运动矢量(包含Skip MV和MVP)的代价，并选择一个代价最小的运动矢量存入best_pos，将其对应的最优代价存入best_cost中。

（4）如果当前搜得的最优运动矢量为Skip MV，且best_cost<阈值T2，则不再进行搜索，结束；如果当前搜得的最优运动矢量为Skip MV，且best_cost>阈值T2，则进行亚像素运动搜索，跳至步骤（6）。如果当前搜得的最优运动矢量为MVP，则跳至步骤（6），直接进行亚像素运动搜索。

（5）对当前搜索得到的最优运动矢量取整，并作为小菱形搜索的搜索中心进行可变步长（Step=1,2）迭代搜索，得到最优的运动矢量与最优代价（终止条件：得到的最优结果仍然为小菱形中心点或者超过最大搜索点数，最大搜索点数为64）。整像素运动估计结束。

Jpred,SAD =SAD + λpred * Bpred，其中Bpred是CABAC编码MV所需要的比特数。

（6）进行小菱形搜索的分像素运动估计，以步骤（5）中得到的最优位置为中心，进行一个方向的小菱形的迭代搜索（终止条件：得到的最优结果仍然为小菱形中心点），得到最优运动矢量bset_pos0及最优代价best_cost0，最优模式为P_L0_2Nx2N。

（7）进行避免分割模式判决的简单提前终止，对以上运动估计过程中搜索过的所有点进行32x32块的代价分割，并取每个点对应的宏块级的代价的最小值（min_sad_64x64）与所有搜索点中最小的4个NxN块的代价之和（min_sad_32x32_0，min_sad_32x32_1，

min_sad_32x32_2，min_sad_32x32_3）比较，如果两者的代价与后者的代价相差不大，小于某个阈值T3则不再进行分割的模式判决，将partition_depth0标志置为0，否则partition_depth0标志置1，继续进行分割块的模式判决。

注意：

1．在进行运动估计时，在比较不同mv的cost值寻找最优mv的过程中，比较的不是64x64的代价，而是修正的64x64的代价。代价分为两部分：高位为64x64的代价，低位为其中最大的32x32的代价。这样在寻找LCU的最优mv时，首先比较64x64块的总代价，找最小的一个；在总代价相同的情况下再比较最大的32x32的代价，找最小的一个。目的在综合考虑总代价与分割代价的情况下，找到与LCU内纹理最相似的像素块。

(2)其他大小编码单元及各个预测单元的运动估计

在进行32x32、16x16编码单元和64x32、32x64、32x16、16x32、16x8、8x16、8x8预测单元运动估计的时候，只进行当前单元目前最小的best_cost和Merge_cost的比较。即对于这些编码单元和预测单元，只需要计算尚未计算的Merge候选运动矢量的代价即可。如图3-4所示，在进行编号为1的预测单元编码时只需要计算周围2个预测单元的运动矢量代价即可，其他的代价可只第0层编码单元运动矢量代价保留处获取；同理，在进行编码为2的预测单元编码时只需要计算周围3个预测单元的运动矢量代价即可，剩下2个预测单元的运动矢量代价也可至第0层编码单元运动矢量代价保留处获取。若当前编码单元或者预测单元被确定为Merge模式，则不再进行继续向下分割。

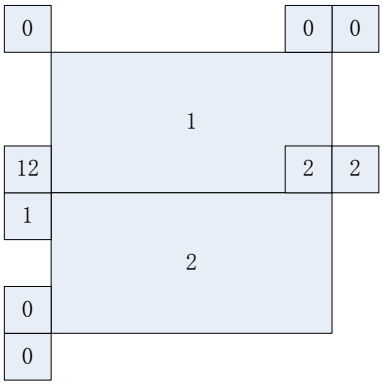


图3-4 预测单元Merge运动矢量获取

(3)帧间预测运算量

在 HEVC 实时编码器的帧间预测部分设计中采取了分割存储的代价缓存机制的设计。由于 HEVC 的多层编码结构，在分层计算代价时常常会出现运动矢量重复的现象，为了减少编码器的硬件数据读入，减少数据吞吐量对于编码的限制，在软件设计中采取了代价缓存的方案。即在 HEVC 中一个 64x64LCU 的代价以 8x8 块方格进行存储，并且把历次每个方格搜索过的运动矢量的代价保留至各个方格中，以便在下层分割时减少搜索点的个数。

经计算知，在一个 LCU 中运动估计的最大搜索点个数为 64+24+26=114（以 64x64 方框的 SAD 为单位）。计算过程如下所示：

(1)64x64 运动估计最大搜索点个数：64，折合成 64x64 块的 SAD 值的个数为 64。

(2)Merge 模式的运动矢量代价计算：

64x64 分割：5，折合成 64x64 块的 SAD 值的个数为 5。

32x64、64x32 分割：（2+3）x2=10，折合成 64x64 块的 SAD 值的个数为 5。

32x32 分割：0+2+2+1=5，折合成 64x64 块的 SAD 值的个数为 1.25。

32x16、16x32 分割：(2+3)x2x4=40，折合成 64x64 块的 SAD 值的个数为 5。

16x16 分割：（0+2+2+1）x4=20，折合成 64x64 块的 SAD 值的个数为 1.25。

16x8、8x16 分割：(2+3)x2x4x4=160，折合成 64x64 块的 SAD 值的个数为 5。

8x8 分割：（0+2+2+1）x16=80，折合成 64x64 块的 SAD 值的个数为 1.25。

最后总计 Merge 模式的运动矢量运算量为 23.75，近似为 24。

(3)MVP 运动矢量代价计算：

64x64 分割：2，折合成 64x64 块的 SAD 值的个数为 2。

32x64、64x32 分割：2x2x2=8，折合成 64x64 块的 SAD 值的个数为 4。

32x32 分割：2x4=8，折合成 64x64 块的 SAD 值的个数为 2。

32x16、16x32 分割：2x2x2x4=32，折合成 64x64 块的 SAD 值的个数为 4。

16x16 分割：2x16=32，折合成 64x64 块的 SAD 值的个数为 2。

16x8、8x16 分割：2x2x2x16=128，折合成 64x64 块的 SAD 值的个数为 4。

8x8 分割：2x64=128，折合成 64x64 块的 SAD 值的个数为 2。

最后总计计算 MVP 的运动矢量代价运算量为 26。

相比于 H.264 实时编码器运动估计部分的运算量为 64，HEVC 实时编码器的运动估计运算量不足 H.264 的 3 倍，在硬件实现的允许范围之内。

3.1.2 P 帧帧间预测模式判别算法

(1)64x64 层模式判别

第一层LCU(64x64)：depth=0

（1）调用hevc_motionestimate_P_LCU函数，进行64x64级别的运动估计，得到best_mv和best_cost,best_cost以8x8宏块为单位进行存储，共有64个8x8代价进行存储。

（2）若PredMode为SKIP模式或者Inter模式下的merge，模式识别结束，否则继续步骤（3）。

（3）判断patition_depth0是否为1。如果patition_depth0=0，不再继续进行分割，模式判别结束。如果patition_depth0=1，继续进行64x32、32x64代价计算比较，记录最优的分割模式和结果更新至Part_type，best_mv，best_cost中。

（4）若比较得最优分割结果为64x64，则不再进行分割比较，模式判别结束；若比较得最优分割结果为32x64或者64x32，则继续步骤（5）。

（5）继续比较当前最优分割模式与32x32分割模式代价，若32x32分割模式代价较小，则继续对32x32中每个CU进行下一层的模式判别，否则结束模式判别。

注意：

（1）在进行运动估计时是针对64x64最大编码单元进行的，对搜索到得最后运动矢量进行代价计算时是以8x8宏块进行存储的，以后进行分割时运动矢量代价更新也是以8x8宏块为单位进行的。

（2）计算64x32、32x64、32x32分割快的某一块代价时，每个分割块计算代价时除了比较MECand_List列表中运动矢量对应的最小代价外，还要比较每个分割快的MergeCand_List列表中运动矢量对应的最小代价，并把结果最小的分割块代价记为当前分割块的代价。

（3）进行分割模式代价比较时，要把64x32的2个分割块的分别最小代价之和、32x64分别最小代价之和、32x32的分别最小代价之和进行比较，并把最小的代价更新至LCU的每个8x8宏块里。

（4）若比较结果为64x32、32x64模式之一，并且其中某一个分割块最小代价为PU对应的MergeCand_List中最优运动矢量代价，则记该PU的merge_flag=1,并记录merge_index。

（5）若比较结果为32x32模式。若其中某一个32x32分割块最小代价为对应的MergeCand_List中最优运动矢量代价时，记录merge_idx。如果cost_merge_32x32<阈值T1/4，则此CU的PredMode=SKIP；否则此CU的PredMode=Inter，merge_flag=1。

（6）在判断是否进行分割时，比较的是4个分割块的值。例如，在比较是否进行64x64

继续分割的时候，计算diffence=mincost_64x64-sum(mincost_32x32_0, mincost_32x32_1, mincost_32x32_2, mincost_32x32_3),然后与阈值T3进行比较。

（7）在比较32x64或者64x32中每个PU的最优mv时候，比较的是相应的块的sad值。

（8）在进行模式判别比较cost的时候，比较的是经过修正的值。64x64分割：高位——64x64总代价，低位——最大32x32分割块的代价；64x32分割：高位——2个64x32块总代价，低位——2个64x32总代价；32x64分割：高位——2个32x64总代价，低位——2个32x64总代价；32x32分割：高位——4个32x32总代价，低位——4个32x32总代价。

(2) 32x32 层模式判别

第二层CU（32x32）：depth=1

（1）若CU的PredMode为SKIP模式或者PredMode=Inter，merge_flag=1，结束，不再进行模式判决，否则继续步骤（2）。

（2）计算32x32最小代价与4个16x16最小代价的差值，若差值＜阈值T3/4,patition_depth1=0，结束，不再进行分割块的模式判决；否则patition_depth1=1，继续分割块的模式判决，继续步骤（3）。

（3）把当前32x32CU的最优代价与32x16、16x32两种分割方式下CU的总代价进行比较，记录最优的分割模式和结果更新至当前CU的Part_type，best_mv，best_cost中。

（4）如果比较得最优的分割结果为32x32，则不再进行分割比较，模式判别结束；若比较得最优分割结果为16x32或者32x16，则继续步骤（5）。

（5）比较当前最优分割模式与16x16分割模式代价，若16x16分割模式代价较小，则继续对16x16中每个CU进行下一层的模式判别，否则结束模式判别。

注意：

（1）若比较结果为16x32、32x16模式之一，并且其中某一个分割块最小代价为PU对应的MergeCand_List中最优运动矢量代价，则记该PU的merge_flag=1,并记录merge_index。

（2）若比较结果为16x16模式。若其中某一个16x16分割块最小代价为对应的MergeCand_List中最优运动矢量代价时，记录merge_idx。如果cost_merge_16x16<阈值T1/16，则此CU的PredMode=SKIP；否则此CU的PredMode=Inter，merge_flag=1。

（3）在判断是否进行分割时，比较的是4个分割块的值。例如，在比较是否进行32x32继续分割的时候，计算diffence=mincost_32x32-sum(mincost_16x16_0, mincost_16x16_1, mincost_16x16_2, mincost_16x16_3),然后与阈值T3/4进行比较。

（4）在比较32x16或者16x32中每个PU的最优mv时候，比较的是相应的块的sad值。

（5）在进行模式判别比较cost的时候，比较的是经过修正的值。32x32分割：高位——32x32总代价，低位——最大16x16分割块的代价；16x32分割：高位——2个16x32块总代价，低位——2个16x32总代价；32x16分割：高位——2个32x16总代价，低位——2个32x16总代价；16x16分割：高位——4个16x16总代价，低位——4个16x16总代价。

(3) 16x16 层模式判别

第三层CU（16x16）：depth=2

（1）若CU的PredMode为SKIP模式或者PredMode=Inter，merge_flag=1，结束，不再进行模式判决，否则继续步骤（2）。

（2）计算16x16最小代价与4个8x8最小代价的差值，若差值＜阈值T3/16,patition_depth2=0，结束，不再进行分割块的模式判决；否则patition_depth2=1，继续分割块的模式判决，继续步骤（3）。

（3）把当前16x16CU的最优代价与8x16、16x8两种分割方式下CU的总代价进行比较，记录最优的分割模式和结果更新至当前CU的Part_type，best_mv，best_cost中。

（4）如果比较得最优的分割结果为16x16，则不再进行分割比较，模式判别结束；若比

较得最优分割结果为16x8或者8x16，则继续步骤（5）。

（5）比较当前最优分割模式与8x8分割模式代价，记录最优的分割模式下的模式和代价。

注意：

（1）若比较结果为16x8、8x16模式之一，并且其中某一个分割块最小代价为PU对应的MergeCand_List中最优运动矢量代价，则记该PU的merge_flag=1,并记录merge_index。

（2）若比较结果为8x8模式。若其中某一个8x8分割块最小代价为对应的MergeCand_List中最优运动矢量代价时，记录merge_idx。如果cost_merge_8x8<阈值T1/64，则此CU的PredMode=SKIP；否则此CU的PredMode=Inter，merge_flag=1。

（3）在判断是否进行分割时，比较的是4个分割块的值。例如，在比较是否进行16x16继续分割的时候，计算diffence=mincost_16x16-sum(mincost_8x8_0, mincost_8x8_1, mincost_8x8_2, mincost_8x8_3),然后与阈值T3/16进行比较。

（4）在比较8x16或者16x8中每个PU的最优mv时候，比较的是相应的块的sad值。

（5）在进行模式判别比较cost的时候，比较的是经过修正的值。16x16分割：高位——16x16总代价，低位——最大8x8分割块的代价；16x8分割：高位——2个16x8块总代价，低位——2个16x8总代价；8x16分割：高位——2个8x16总代价，低位——2个8x16总代价；8x8分割：高位——4个8x8总代价，低位——4个8x8总代价。

3.2 帧间预测改进算法

帧间预测改进算法是基于 HM6.1 编码器平台结合 HEVC 实时编码器算法设计的用于 HM6.1 编码器的帧间预测算法，目的是为了不明显降低视频质量的前提下，大大降低帧间预测算法的运算量。帧间预测改进算法主要是针对 HM6.1 帧间预测算法中的整像素运动估计算法和模式判别算法进行了修改。

3.2.1 帧间预测改进算法 1——修改整像素运动估计算法

在帧间预测算法中整像素运动估计对于运动矢量的精确度、运动矢量残差、像素残差、运动估计搜索点个数有重要的影响，因此帧间预测改进算法 1 主要是是针对整像素运动估计算法进行了修改。修改的整像素运动估计算法如下所示：

（1）确定中心搜索点：在中心搜索点、A、B、C、(0,0)5 个运动矢量中挑选中心搜索点，选取其中代价最小的一个作为中心搜索点。A、B、C 的位置如图 3-5 所示。

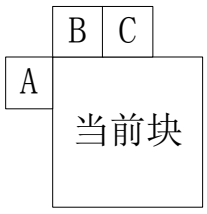


图3-5 候选块A、B、C与当前块的位置

（2）判断中心搜索点运动矢量的代价，如果代价<阈值T4，则结束整像素运动搜索。阈值T是根据搜索块大小进行自适应的。

（3）进行运动矢量粗搜索。

在中心搜索点附近进行变半径自适应钻石搜索，分别令半径为1、2、4、8。半径为1时搜索点位置如图3-6所示，半径为2、4、8时搜索点位置如图3-7所示。

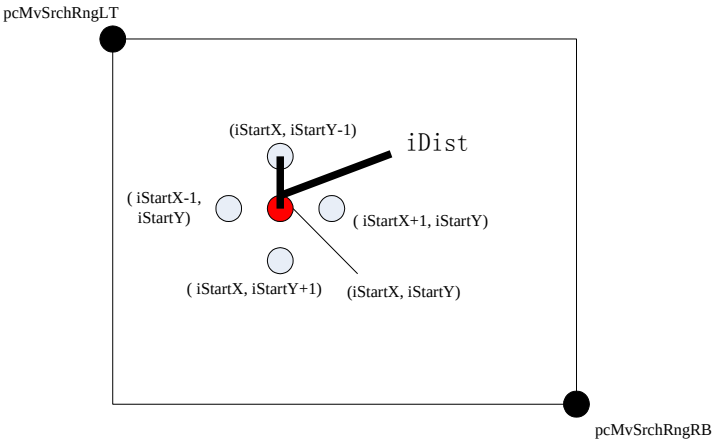


图3-6 iDist=1时，自适应钻石搜索——小菱形搜索

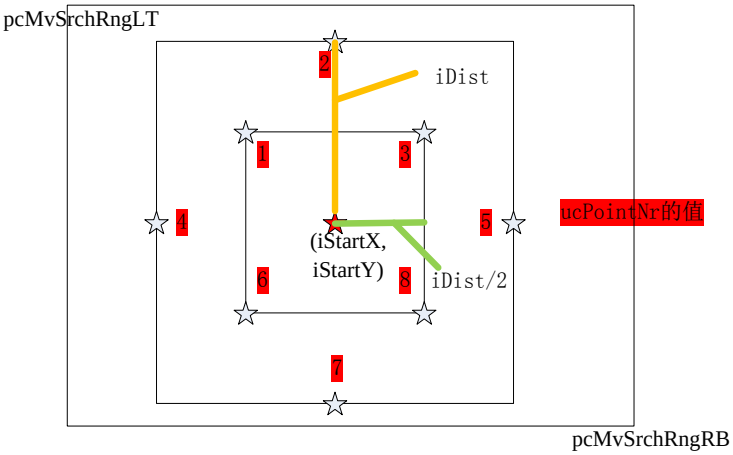


图3-7 iDist=2、4、8时，大钻石搜索

若最优运动矢量和中心搜索点的距离为1，则按照大钻石搜索后最优运动矢量位置方向取周围2点位置进行搜索，如图3-8所示。否则跳过这一步。

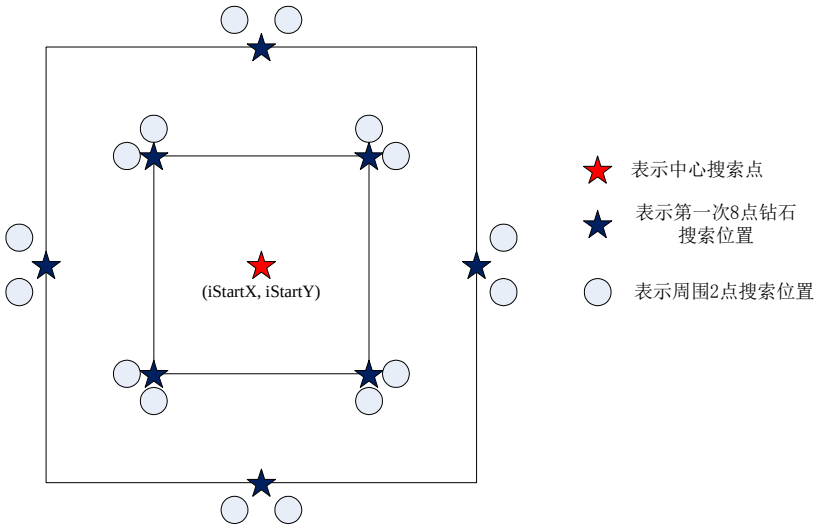


图3-8 根据钻石搜索方向（ucPointNr）得到的周围2点搜索搜索位置

最后比较最优运动矢量和中心搜索点距离。若距离大于5，则在搜索窗内进行全局步骤为5的光栅扫描，否则跳过光栅扫描。

（3）进行运动矢量细搜索。

经过运动矢量粗搜索，最优运动矢量和中心搜索点的距离已经不大于 5。若最优运动矢

量距离中心搜索点的距离小于等于 2，则不再进行运动矢量细搜索，整像素运动估计结束。否则，令搜索半径=4，进行周围 8 点大钻石搜索，如图 3-7 所示，搜索的 8 个点距离中心搜索点的距离为 2 或者 4。若搜索得最优运动矢量距离中心搜索点距离小于等于 2，则对于最优运动矢量按照方向取周围 2 点进行搜索，然后结束运动矢量细搜索；否则继续令搜索半径=4，进行周围 8 点大钻石搜索，直至最优运动矢量和中心搜索点的位置小于等于 2 为止。

3.2.2 帧间预测改进算法 2——修改模式判别+整像素运动估计算法

在帧间预测算法中模式识别对于实现多层编码搜索是十分重要的，模式判别算法的不同会导致视频质量和帧间预测算法运算量的显著差异。在 HEVC 编码器帧间预测算法中，采取了 SKIP 模式在多次出现的情况下提前终止的算法（bEarlySkip）、判断编码单元是否进行预测单元分割的算法（doNotBlockPu）、本层编码单元编码完毕提前终止算法（bTrySplitDQP）。

在程序运行测试过程中， SKIP 模式提前终止、高层模式提前终止、PU 分割提前终止算法被触发的概率很低。在大多数情况下，每一个 LCU 会从最高层编码单元 1 个 64x64 像素块，循环到每个最底层编码单元 64 个 8x8 像素块；并且每个编码单元在绝大多数情况下都会进行预测单元 2NxN、Nx2N 的分割判断。但是根据模式识别判断的最终结果可知：在很多情况下，较高层编码单元（64x64、32x32）运动估计得到的像素残差块和最终得到的像素残差块相差不大，在这种情况下仍然进行低层次的编码单元分割是对于运算空间的浪费，大大影响了帧间预测模块的运算量，影响了编码性能。

考虑到模式判别提前终止算法在帧间预测算法中发挥的重要作用，帧间预测改进算法 2 在帧间预测改进算法 1 的基础上进行了修改，主要是针对帧间预测模式判别提前终止算法进行了修改，帧间模式判别算法如下：

（1）bEarlySkip = false，bTrySplit = true。

（2）检查SKIP模式。如果在当前LCU中与当前CU同样大小的CU采用skip的个数多于5个并且SKIP的cost小于阈值（aiNum [iIdx] > 5 && fRD_Skip < EARLY_SKIP_THRES*afCost[iIdx]/aiNum[iIdx]），则进行提前终止，不再向下一层进行分割搜索（bEarlySkip = true，bTrySplit = false）。

（3）检查INTER_2Nx2N模式。如果经过SKIP模式和INTER_2Nx2N模式比较之后，模式代价小于阈值T5，则进行提前终止，不再进行预测单元分割判断，不再进行下一层编码单元分割判断（bEarlySkip = true，bTrySplit = false）。

（4）若bEarlySkip = false&&doNotBlockPu=flase，则继续进行分割块的模式代价判断，依次判断INTER_Nx2N, INTER_2NxN分割模式。

（5）若经过比较，在2Nx2N、2NxN、Nx2N三种分割情况模式下2Nx2N代价为最优，则令 bEarlySkip = true，bTrySplit = false。

（6）若bTrySplit = false，则不再进行下一层编码单元（depth=depth+1）的分割判断，模式判别终止；否则循环继续下一层编码单元的模式判别

3.2.3 帧间预测改进算法 3——修改模式判别算法

经过帧间预测改进算法 1 和帧间预测改进算法 2 实验结果的性能分析和对比，笔者发现整像素运动搜索算法大约会降低帧间预测运算量 10%-15%，不会使视频质量明显下降；而改进算法 2 中较大程度降低了帧间预测运算量（60%-70%）的同时也带来了视频质量的明显下降（0.4dB-0.6dB）。可见模式判别算法对于降低帧间预测算法的运算量效果显著，但是过于粗糙的阈值赋值和帧间预测算法的大幅度提前终止也给视频性能带来了较大的损害。

基于上述实验结果，帧间预测改进算法 3 对于模式判别算法重新做了修改。在帧间预测改进算法 3 中重新合理地定义提前终止阈值，并考虑增加提前终止的算法限制，目的是为了减少提前终止算法对于视频性能的损害。

首先对提前终止阈值进行统计更改。在 QPI=32 时，根据 20 帧 Basketball_832x480 视频序列和 20 帧 PartySence_832x480 视频序列编码结果统计显示：

（1）在 20 帧 Basketball_832x480 视频序列中共有 64x64 级编码单元：

19x104=1976 个

其中共有 565 个 64x64 编码单元最终模式代价和 64x64 分割模式代价相同，可以提前终止，可以提前终止的比率为：

565/1976=28.58%

图 3-9 所示为提前终止的 64x64 块模式代价的正态分布图，期望代价为 92157。

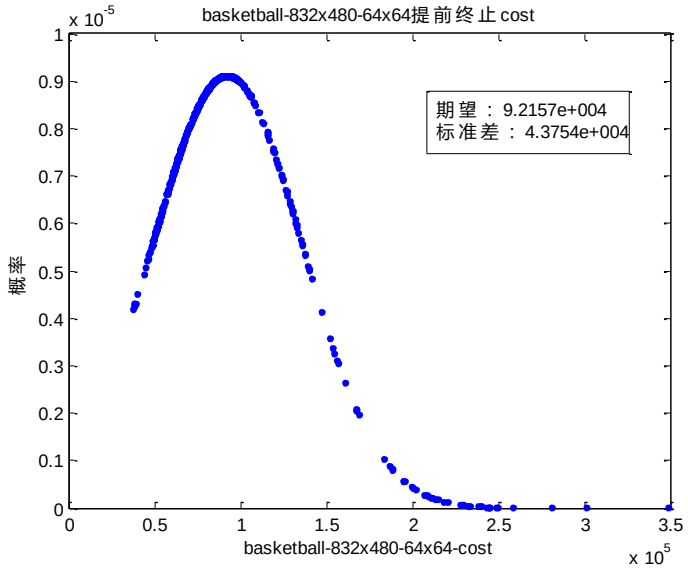


图 3-9 Baketball 提前终止的 64x64 块模式代价的正态分布图

（2）在 20 帧 PartySence_832x480 视频序列中共有 64x64 级编码单元：

19x104=1976 个

其中共有 395 个 64x64 编码单元最终模式代价和 64x64 分割模式代价相同，可以提前终止，可以提前终止的比率为：

395/1976=19.99%

图 3-10 所示为提前终止的 64x64 块模式代价的正态分布图，期望代价为 213050。

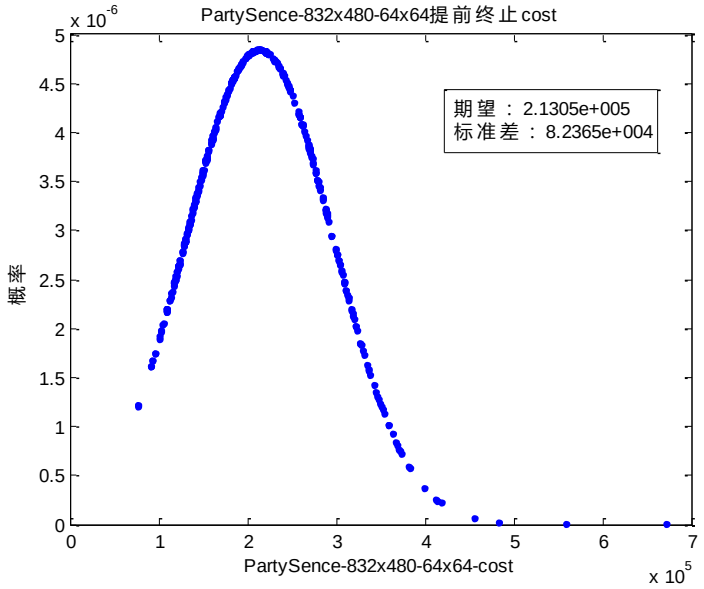


图 3-10 PartySence 提前终止的 64x64 块模式代价的正态分布图

（3）在 20 帧 Basketball_832x480 视频序列中共有 32x32 级编码单元：

19x390=7410 个

其中共有 3900 个 32x32 编码单元最终模式代价和 32x32 分割模式代价相同，可以提前终止，可以提前终止的比率为：

3900/7410=52.63%

图 3-11 所示为提前终止的 32x32 块模式代价的正态分布图，期望为 24728。

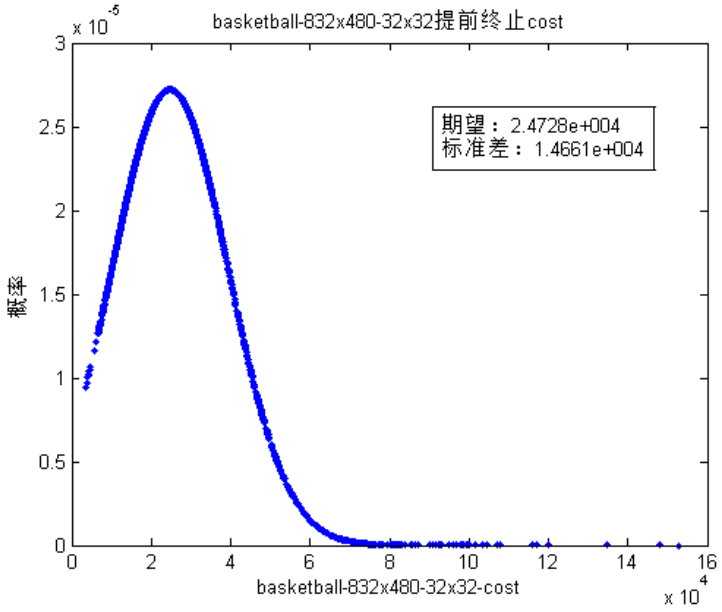


图 3-11 Baketball 提前终止的 32x32 块模式代价的正态分布图

（4）在 20 帧 PartySence_832x480 视频序列中共有 32x32 级编码单元：

19x390=7410 个

其中共有 2551 个 32x32 编码单元最终模式代价和 32x32 分割模式代价相同，可以提前终止，可以提前终止的比率为：

2551/7410=34.42%

图 3-12 所示为提前终止的 32x32 块模式代价的正态分布图，期望代价为 55527。

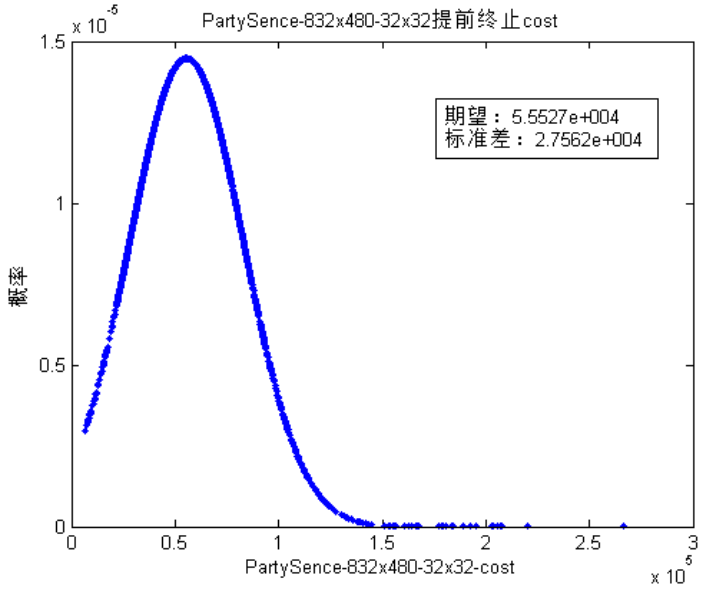


图 3-12 PartySence 提前终止的 32x32 块模式代价的正态分布图

从上面Baketball视频序列和PartySence视频序列统计数据中可以看出：编码单元在64x64分割和32x32分割下提前终止块数目已经可以站到视频序列块数目的一半以上，可见模式提前终止算法对于提高编码效率、减少帧间预测运算量十分重要。Baketball视频序列64x64块提前终止的代价期望为92157，PartySence视频序列64x64块提前终止的代价期望为213050；Baketball视频序列32x32块提前终止的代价期望为24728，PartySence视频序列32x3块提前终止的代价期望为55527。考虑到不同视频序列的差异，在算法3中选取64x64提前终止门限值为60000，32x32提前终止门限值为15000进行实验。

在增加提前终止算法限制上主要是针对下一层编码单元分割进行的，增加了一些提前终止的算法限制。

3.2.4 帧间预测改进算法 4——修改模式判别+整像素运动估计算法

由算法 1 结果可以看出：修改了整像素运动估计算法可以在不明显降低视频质量的前提下降低帧间预测运算量；同样由算法 3 结果中可以看出：修改过的模式判别算法可以在不明显降低视频质量的前提下较大幅度降低帧间预测运算量。算法 4 中结合了算法 1 和算法 3 的优点，对于模式判别和整像素运动估计算法都做了修改，目的是在不明显降低视频质量的前提下，较大幅度得降低帧间预测运算量。

3.3 本章小结

本章主要介绍了用于实现实时编码器的帧间预测算法和在 HM6.1 上实现的帧间预测改进算法。基于 H.264 实时编码器中高效帧间预测算法和 HEVC 编码器 HM6.1 帧间预测算法的特点，在本文中设计了 HEVC 实时编码器帧间预测算法。在 HEVC 实时编码器帧间预测算法中，采用了 SKIP 模式提前终止、Merge 模式提前终止、2Nx2N 在本层分割最优时不再进行下一层分割、hash 算法控制运动估计搜索点个数、分割存储代价缓存等机制，在不影响 HEVC 帧间编码性能的前提下大大减少了 HEVC 帧间预测模块的运算量，提高了 HEVC 编码效率。然后本文又结合了 HEVC 实时编码器帧间预测算法，设计出了基于 HM6.1 编码器的帧间预测改进算法。在四种帧间预测改进算法中，分别针对整像素运动估计算法和模式判别算法进行了不同方面的研究，譬如增加中心运动矢量提前终止算法、2NX2N 模式判别提前终止算法、修改阈值门限等。改进帧间算法的目的是在不明显降低视频质量的前提下，较大幅度得降低帧间预测模块运算量，提高编码效率。本章提出的实时编码器帧间预测算法和 HM 编码器帧间预测改进算法是根据 HM6.1 编码器算法、实时编码器要求的性能、HEVC 帧间预测模块功能和标准而设计，同时也是第四章软件仿真改进帧间预测算法的主要依据。

第四章 帧间预测改进算法的软件仿真及性能分析评估

4.1 算法仿真环境

4.1.1 基础仿真环境

本毕设帧间预测改进算法是基于 HEVC 参考编码器 HM6.1 的编码架构进行仿真的。帧间预测改进算法也是针对 HM6.1 的帧间预测算法进行修改的。在帧间预测改进算法中主要修改了整像素运动搜索算法和模式判别算法，分别在函数 xTZSearch 和函数 xCompressCU 中实现，详细实现过程见 2.2 章。

函数 xTZSearch 的函数架构如图 4-1 所示。函数 xCompressCU 的函数架构和算法流图如图 4-2 和 4-3 所示。

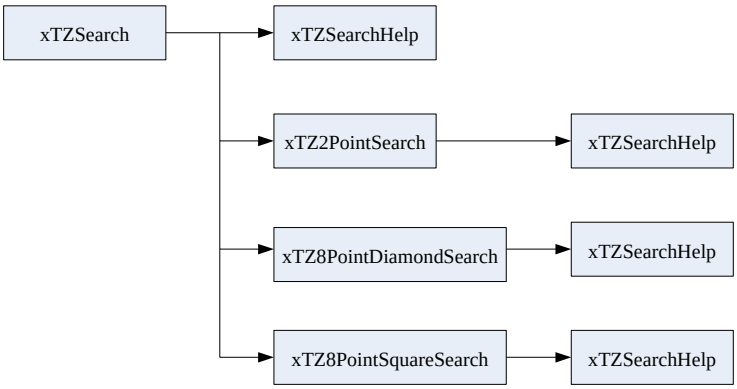


图4-1 函数xTZSearch调用的主要函数结构图结构图

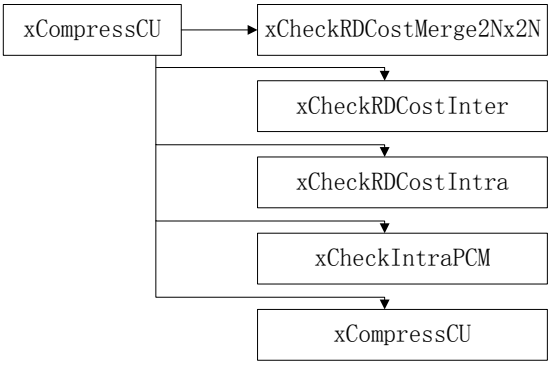


图4-2 函数xCompressCU中调用的和模式判别相关的主要函数

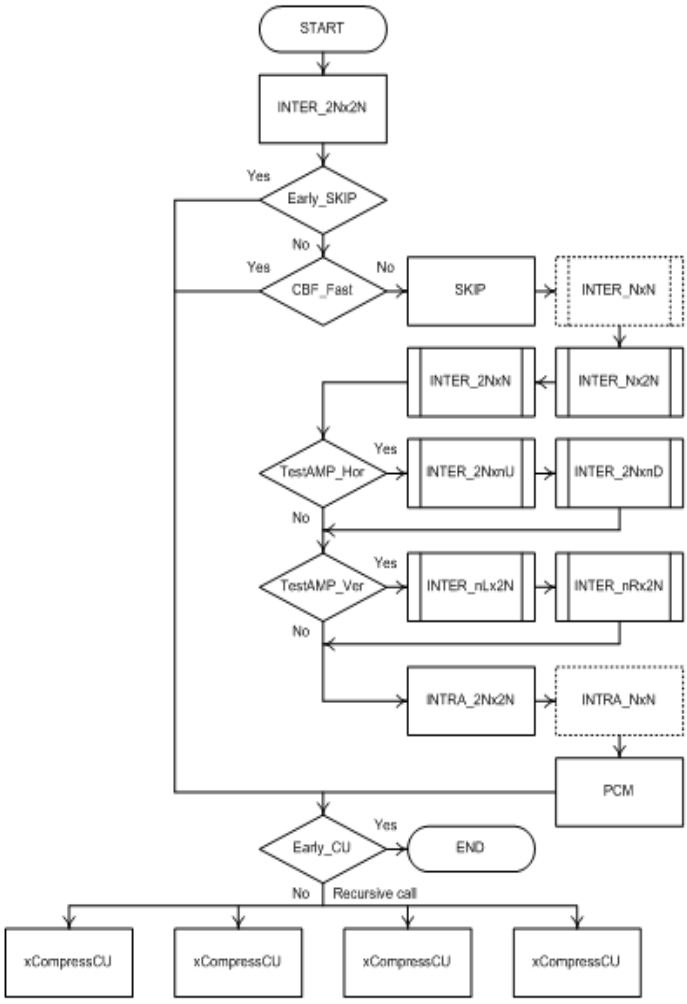


图4-3 函数xCompressCU算法流图[3]

4.1.2 仿真编码结构配置

本毕设目的是为了模拟仿真低复杂度编码设置下的帧间预测算法，因此在编码结构配置中也采用了低复杂度编码结构，如采用了随机存取预测结构、移除了不对称单元分割等。编码工程文件设置如表 4-1 所示。毕设实验中对比的视频序列如表 4-2 所示。

表 4-1 工程文件路径

环境	工程文件位置
MS Visual Studio 2008	build\HM_vc9.sln

表 4-2 视频序列设置

视频序列	视频序列设置
BasketballDrill_832x480_50.yuv	-h -i E:\hevc\software\BasketballDrill_832x480_50.yuv -c E:\hevc\software\trunk\cfg\encoder_randomaccess_main.cfg -c E:\hevc\software\trunk\cfg\per-sequence\BasketballDrill.cfg
PartyScene_832x480_50.yuv	-h -i E:\hevc\software\PartyScene_832x480_50.yuv -c E:\hevc\software\trunk\cfg\encoder_randomaccess_main.cfg -c E:\hevc\software\trunk\cfg\per-sequence\PartyScene.cfg
Traffic_2560x1600_30_crop.yuv	-h -i E:\hevc\software\Traffic_2560x1600_30_crop.yuv -c E:\hevc\software\trunk\cfg\encoder_randomaccess_main.cfg -c

4.2 算法改进及性能分析评估

在实时编码器开始实现之前，需要完成功能规格书、执行规格书等相关文档，并确定算法的可靠性、高效性、均衡性，因此需要在实时编码器进入编码前进行算法的仿真模拟以验证算法性能。

由于实时编码器的框架尚未搭建，所以此次算法性能分析评估是在 HEVC 参考编码器 HM6.1 中测试了部分算法性能并根据性能分析结构进行修改。

4.2.1 改进算法 1 性能评估——修改整像素运动估计算法

帧间预测算法 1 主要修改了整像素运动估计函数，设置了中心搜索矢量提前终止算法、对于细化运动矢量搜索过程进行了范围扩大，目的是在不大程度损害视频质量性能的前提下减少帧间预测的运算量，提高编码效率。

首先用 20 帧视频序列 Basketball_832x480 测试算法 1 性能。HM 原始算法和修改算法 1 的视频质量和帧间运算量比较如图 4-4 和 4-5 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-3 所示。

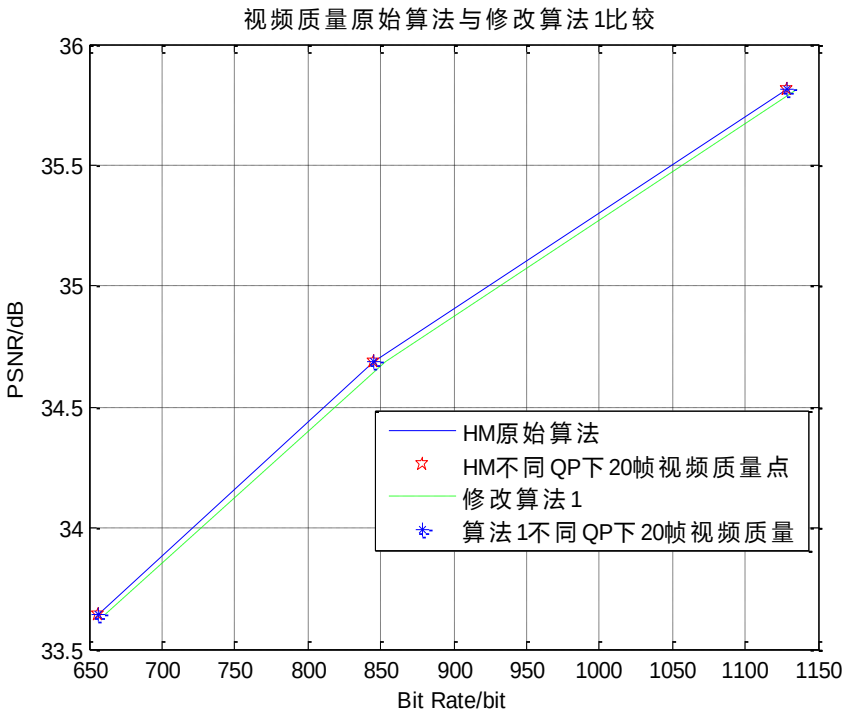


图4-4 Baketball视频序列算法1视频质量比较

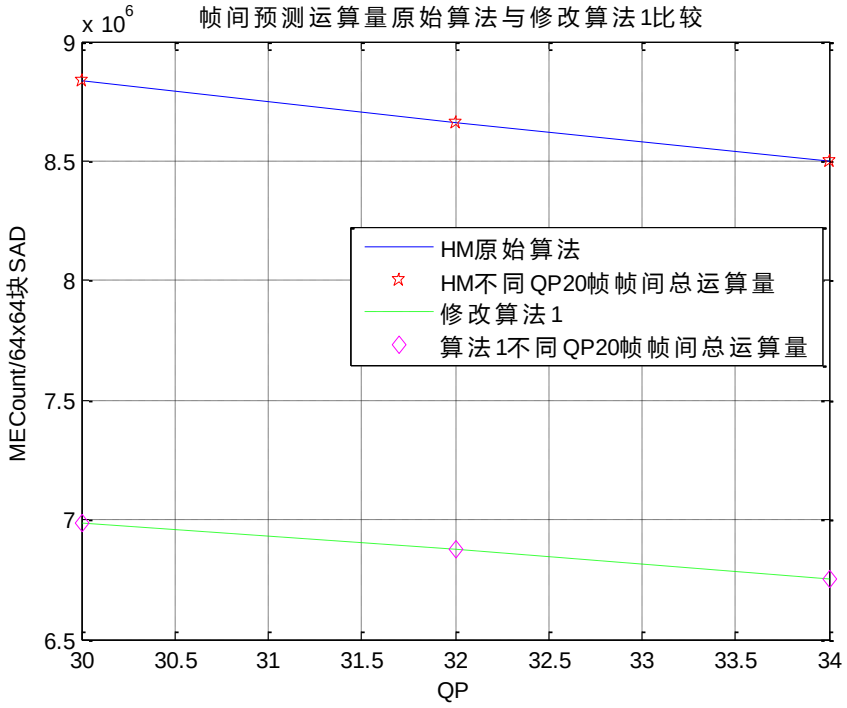


图4-5 Baketball视频序列算法1帧间预测运算量比较

表4-3 Baketball视频序列HM原始算法和修改算法1数据比较

	QP	30	32	34
HM 原来算法	PSNR	35.80997	34.689688	33.645065
	BitRate	1128.92	845.04	656
	MECount	8834487.72	8662020.97	8499995.37
修改算法 1	PSNR	35.80997	34.689688	33.645065
	BitRate	1136.08	853.02	661.46
	MECount	6989191.13	6874620.1	6749528.88

由表 4-3 可以看出，在原始算法中视频序列每个 LCU 的运算量为：

（8834487.72+8662020.97+8499995.37）/（3x19x104）=4385.38（64x64SAD）

在修改算法 1 中视频序列每个 LCU 的运算量为：

（6989191.13+6874620.1+6749528.88）/（3x19x104）=3477.28（64x64SAD）

运算量下降约为：

（4385.38-3477.28）/4385.38=20.7%

视频质量 PSNR 下降很小，不到 0.05dB。

然后用 20 帧视频序列 PartySence_832x480 测试算法 1 性能。HM 原始算法和修改算法 1 的视频质量和帧间运算量比较如图 4-6 和 4-7 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-4 所示。

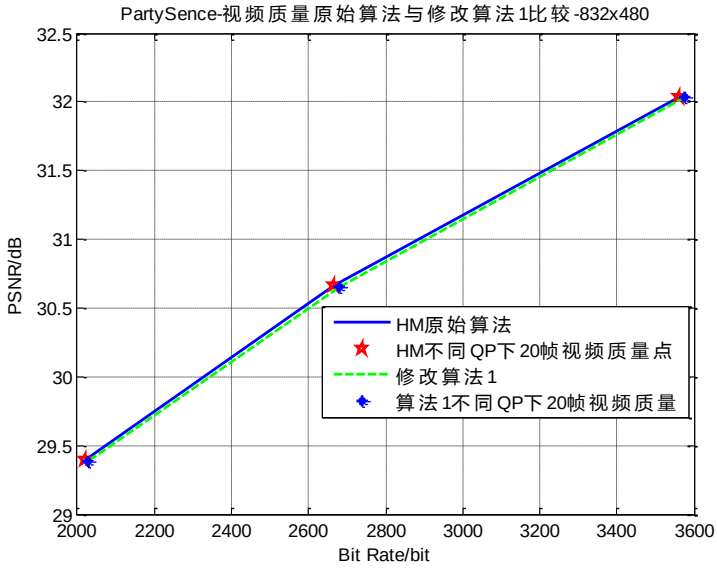


图4-6 PartySence视频序列算法1视频质量比较

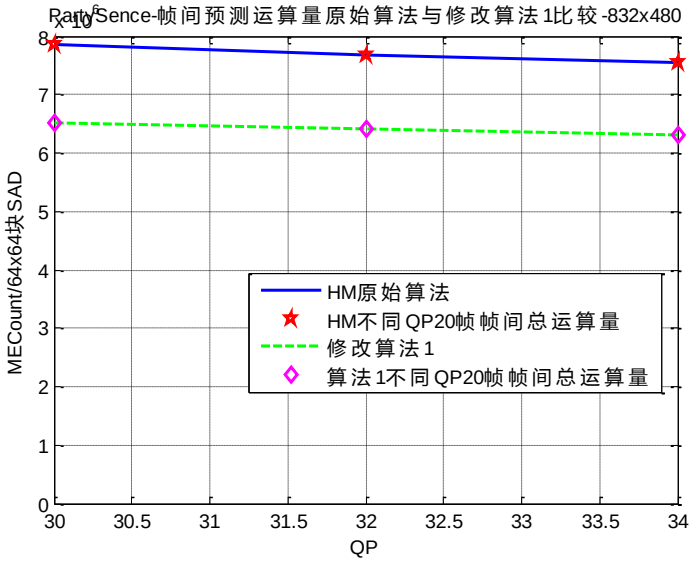


图4-7 PartySence视频序列算法1帧间预测运算量比较

表4-4 PartySence视频序列HM原始算法和修改算法1数据比较

	QP	30	32	34
HM 原来算法	PSNR	32.032452	30.666974	29.398978
	BitRate	3560.12	2666.46	2022
	MECount	7847747.56	7678533.46	7540765.41
修改算法 1	PSNR	32.034091	30.654444	29.381095
	BitRate	3577.52	2681.1	2026.72
	MECount	6526297.8	6417587.86	6316352.720

由表 4-4 中可以看出，在原始算法中视频序列每个 LCU 的运算量为：

（7847747.56+7678533.46+7540765.41）/（3x19x104）= 3891.20（64x64SAD）

在修改算法 1 中视频序列每个 LCU 的运算量为：

（6526297.8+6417587.86+6316352.72）/（3x19x104）=3249.03（64x64SAD）

运算量下降约为：

（3891.20.38-3249.03）/3891.20=16.5%

视频质量 PSNR 下降很小，不到 0.05dB。

综上所述，帧间预测改进算法 1 在不明显降低视频质量的前提下，可以使帧间预测的运算量下降 15%-20%，对于提高编码效率非常有利。

4.2.2 改进算法 2 性能评估——修改模式判别+整像素运动估计算法

在帧间预测算法中模式识别对于实现多层编码搜索是十分重要的，模式判别算法的不同会导致视频质量和帧间预测算法运算量的显著差异。，帧间预测算法 2 在算法 1 修改了整像素运动估计算法的基础上进行了修改，主要是针对帧间预测模式判别提前终止算法进行了修改：

首先用 20 帧视频序列 Basketball_832x480 测试算法 2 性能。HM 原始算法和修改算法 2 的视频质量和帧间运算量比较如图 4-8 和 4-9 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-5 所示。

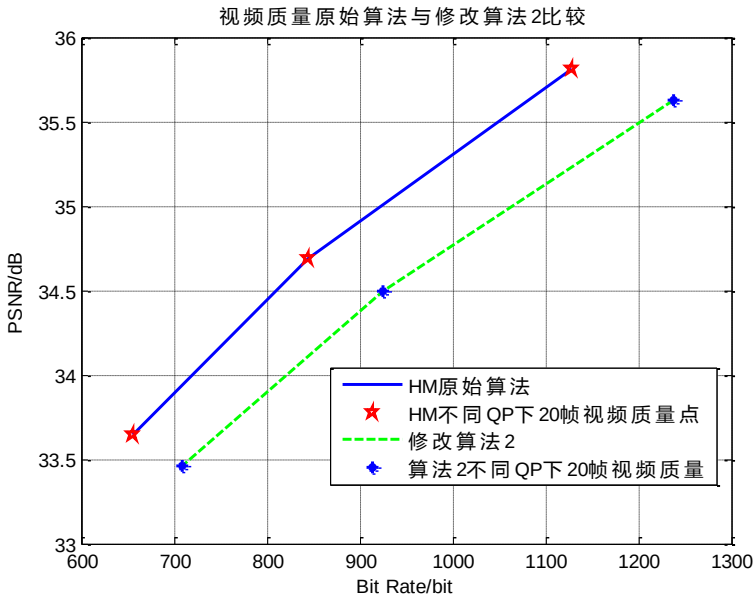


图4-8 Baketball视频序列算法2视频质量比较

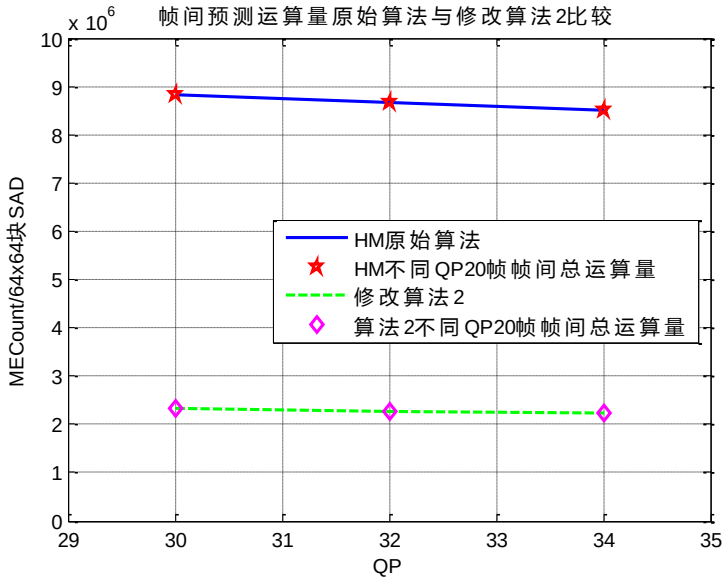


图4-9 Baketball视频序列算法2帧间预测运算量比较

表4-5 Baketball视频序列HM原始算法和修改算法2数据比较

	QP	30	32	34
HM 原来算法	PSNR	35.80997	34.689688	33.645065
	BitRate	1128.92	845.04	656
	MECount	8834487.72	8662020.97	8499995.37
修改算法 2	PSNR	35.629227	34.496619	33.458397
	BitRate	1237.58	925.08	709.42
	MECount	2336922.88	2258355.33	2237285.18

由表 4-5 可以看出，在原始算法中视频序列每个 LCU 的运算量为：

（8834487.72+8662020.97+8499995.37）/（3x19x104）=4385.38（64x64SAD）

在修改算法 2 中视频序列每个 LCU 的运算量为：

（2336922.88+2258355.33+2237285.18）/（3x19x104）=1152.59（64x64SAD）

运算量下降约为：

（4385.38-1152.59）/4385.38=73.7%

视频质量 PSNR 下降较大，约为 0.5dB。

然后用 20 帧视频序列 PartySence_832x480 测试算法 2 性能。HM 原始算法和修改算法 2 的视频质量和帧间运算量比较如图 4-10 和 4-11 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-6 所示。

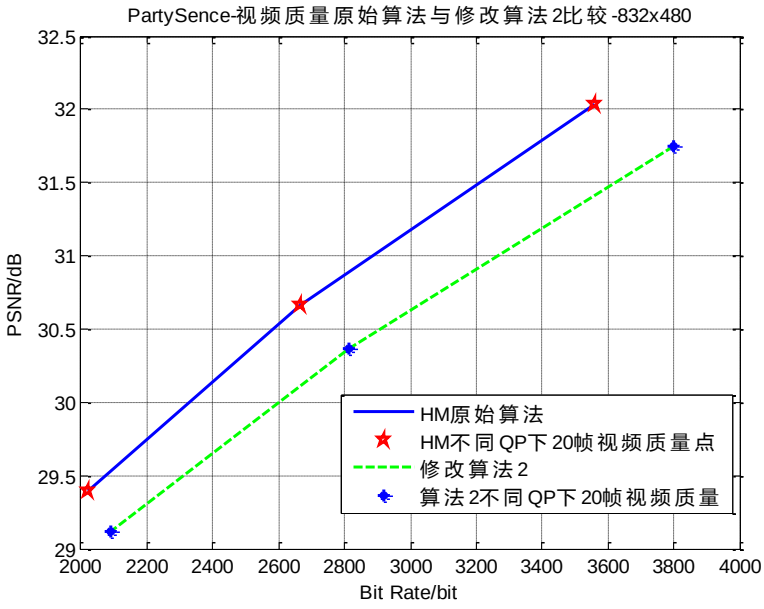


图4-10 PartySence视频序列算法2视频质量比较

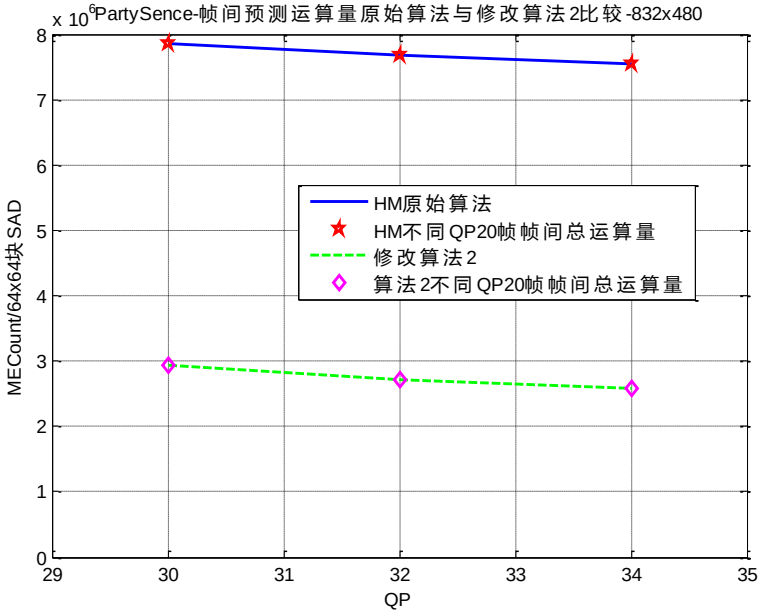


图4-11 PartySence视频序列算法2帧间预测运算量比较

表4-6 PartySence视频序列HM原始算法和修改算法2数据比较

	QP	30	32	34
HM 原来算法	PSNR	32.032452	30.666974	29.398978
	BitRate	3560.12	2666.46	2022
	MECount	7847747.56	7678533.46	7540765.41
修改算法 2	PSNR	31.752641	30.369019	29.1172
	BitRate	3801.68	2811.3	2090.54
	MECount	2929310.17	2716904.52	2576247.76

由表 4-6 可以看出，在原始算法中视频序列每个 LCU 的运算量为：

（7847747.56+7678533.46+7540765.41）/（3x19x104）= 3891.20（64x64SAD）

在修改算法 2 中视频序列每个 LCU 的运算量为：

（2929310.17+2716904.52+2576247.76）/（3x19x104）=1387.06（64x64SAD）

运算量下降约为：

（3891.20.38-1387.06）/3891.20=64.3%

视频质量 PSNR 下降较大，约为 0.4-0.6dB。

综上所述，帧间预测改进算法 2 虽然使得帧间预测算法运算量下降了 60%-70%，但是也给视频质量带来了较大损害，PSNR 下降了 0.4dB-0.6dB。可见，虽然帧间预测改进算法 2 有可取之处，可以大幅度降低帧间预测运算量，但是对于视频质量的损害较大，仍然需要继续改进。

4.2.3 改进算法 3 性能评估——修改模式判别

基于算法1和算法2的实验结果的分析对比，笔者发现模式判别算法对于降低帧间预测算法的运算量效果显著，但是过于粗糙的阈值赋值和帧间预测算法的大幅度提前终止也给视频性能带来了较大的损害。根据Baketball视频序列和PartySence视频序列的提前终止代价的统计数据，在改进模式判别的算法3中设定64x64提前终止门限值为60000，32x32提前终止门限值为15000进行实验。

首先用 20 帧视频序列 Basketball_832x480 测试算法 3 性能。HM 原始算法和修改算法 3 的视频质量和帧间运算量比较如图 4-12 和 4-13 所示，视频序列的 PSNR、BitRate、MEcount

如表 4-7 所示。

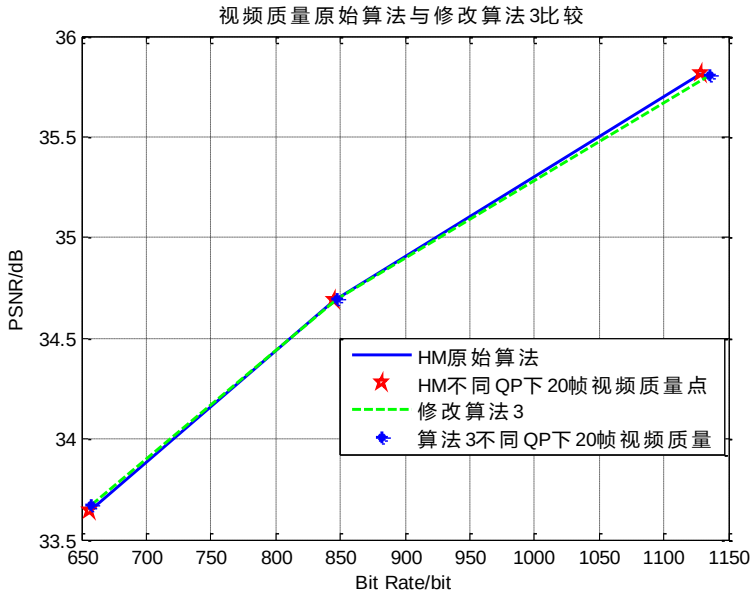


图4-12 Baketball视频序列算法3视频质量比较

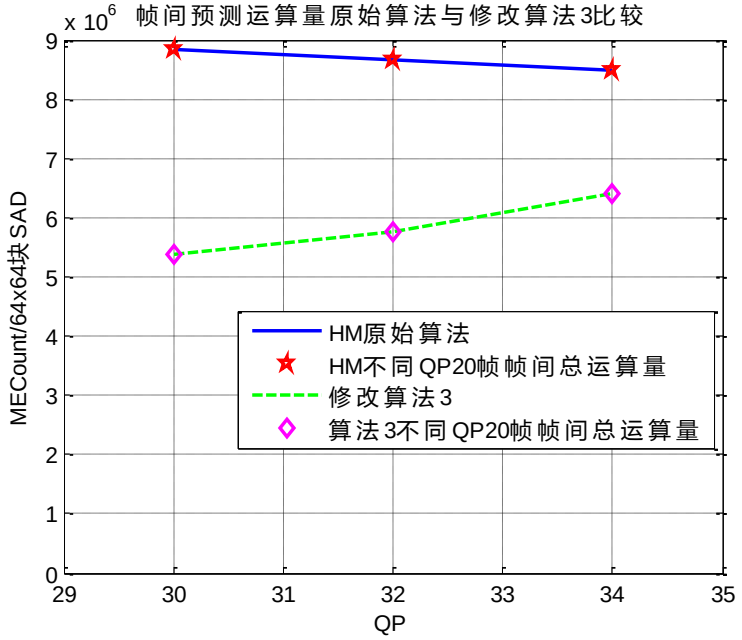


图4-13 Baketball视频序列算法3帧间预测运算量比较

表4-7 Baketball视频序列HM原始算法和修改算法3数据比较

	QP	30	32	34
HM 原来算法	PSNR	35.80997	34.689688	33.645065
	BitRate	1128.92	845.04	656
	MECount	8834487.72	8662020.97	8499995.37
修改算法 3	PSNR	35.805856	34.69383	33.66916
	BitRate	1135.22	847.1	657.54
	MECount	5365774.46	5768417.55	6419685.74

如表 4-7 所示，在原始算法中视频序列每个 LCU 的运算量为：

（8834487.72+8662020.97+8499995.37）/（3x19x104）=4385.38（64x64SAD）

在修改算法 3 中视频序列每个 LCU 的运算量为：

（5365774.46+5768417.55+6419685.74）/（3x19x104）=2961.18（64x64SAD）

运算量下降约为：

（4385.38-2961.18）/4385.38=32.47%

视频质量 PSNR 下降很小，不到 0.05dB。

然后用 20 帧视频序列 PartySence_832x480 测试算法 3 性能。HM 原始算法和修改算法 3 的视频质量和帧间运算量比较如图 4-14 和 4-15 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-8 所示。

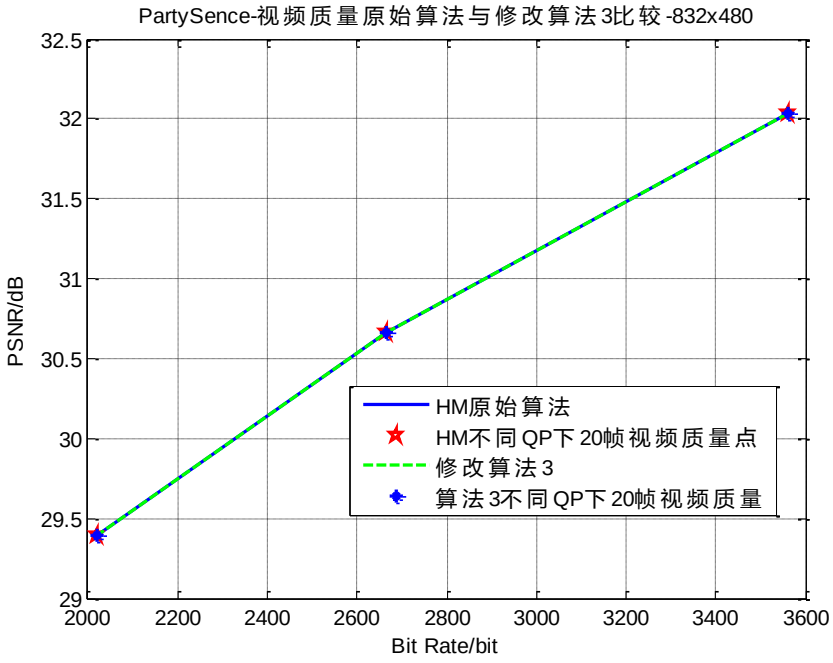


图4-14 PartySence视频序列算法3视频质量比较

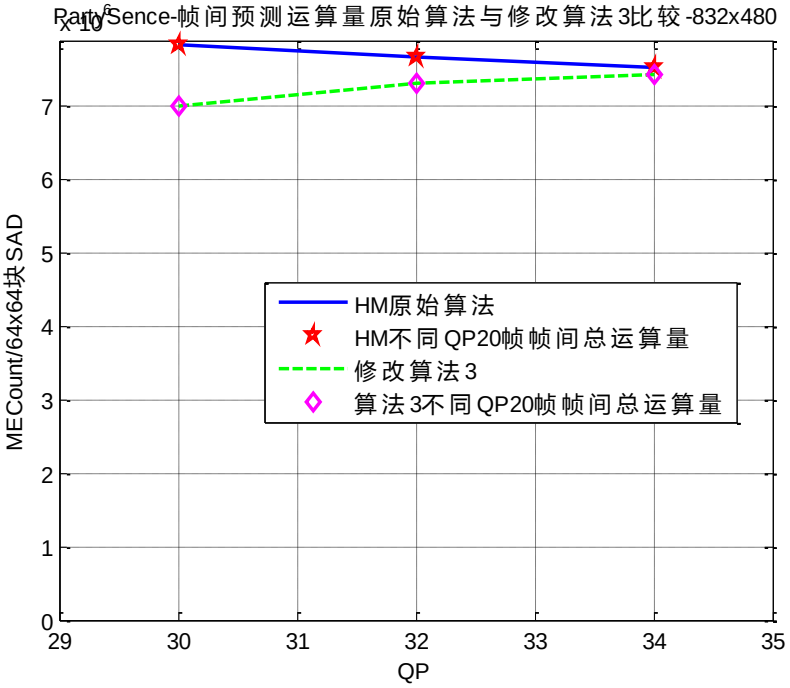


图4-15 PartySence视频序列算法3帧间预测运算量比较

表4-8 PartySence视频序列HM原始算法和修改算法3数据比较

	QP	30	32	34
HM 原来算法	PSNR	32.032452	30.666974	29.398978
	BitRate	3560.12	2666.46	2022
	MECount	7847747.56	7678533.46	7540765.41
修改算法 3	PSNR	32.032485	30.661625	29.391448
	BitRate	3562.6	2667.08	2020.3
	MECount	6526297.8	7012037.79	7321112.93

如表 4-8 所示，在原始算法中视频序列每个 LCU 的运算量为：

（7847747.56+7678533.46+7540765.41）/（3x19x104）= 3891.20（64x64SAD）

在修改算法 3 中视频序列每个 LCU 的运算量为：

（7012037.79+7321112.93+7438574.09）/（3x19x104）=3672.69（64x64SAD）

运算量下降约为：

（3891.20.38-3672.69）/3891.20=5.6%

视频质量 PSNR 下降很小，不到 0.02dB。

最后用 20 帧视频序列 Traffic_2560x1600（4096x2048p 截取）测试算法 3 性能。HM 原始算法和修改算法 3 的视频质量和帧间运算量比较如图 4-16 和 4-17 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-9 所示。

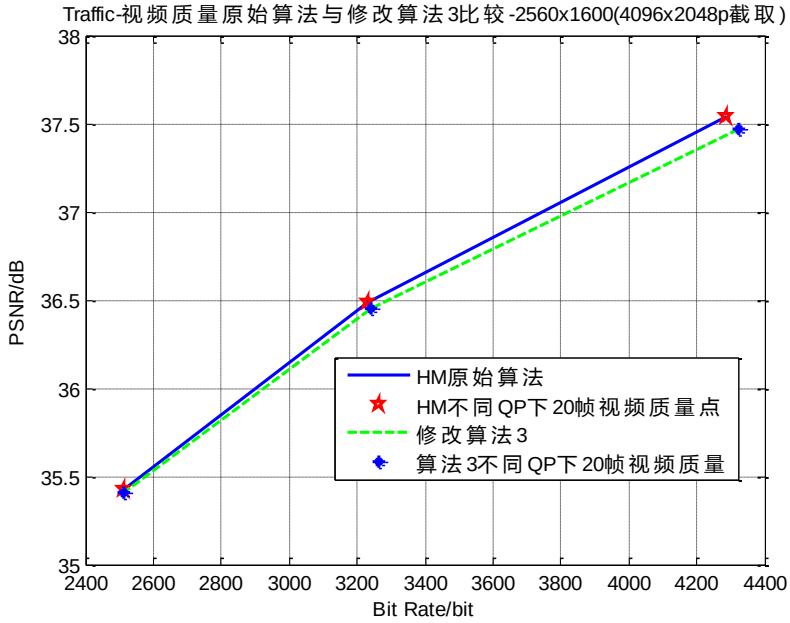


图4-16 Traffic视频序列算法3视频质量比较

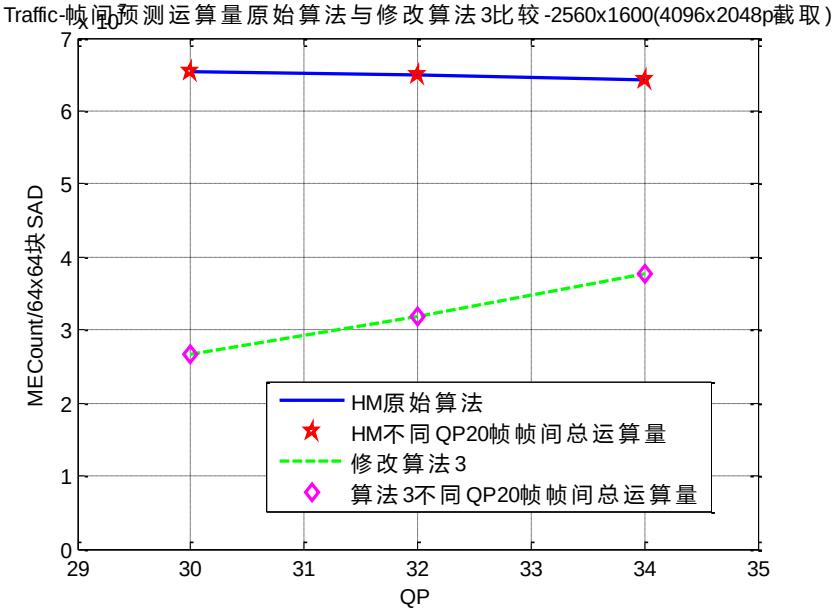


图4-17 Traffic视频序列算法3帧间预测运算量比较

表4-9 Traffic视频序列HM原始算法和修改算法3数据比较

	QP	30	32	34
HM 原来算法	PSNR	37.537207	36.484765	35.428112
	BitRate	4287.54	3231.564	2512.764
	MECount	65337049.04	64848651.16	64368444.97
修改算法 3	PSNR	37.470708	36.450798	35.40933
	BitRate	4326.072	3240.348	2514.396
	MECount	26640141.18	31892859.48	37694334.33

如图所示，在原始算法中视频序列每个 LCU 的运算量为：

（65337049.04+64848651.16+64368444.97）/（3x19x1000）=3413.23（64x64SAD）

在修改算法 3 中视频序列每个 LCU 的运算量为：

（26640141.18+31892859.48+37694334.33）/（3x19x1000）=1688.20（64x64SAD）

运算量下降约为：

（3413.23-1688.20）/3413.23=50.54%

视频质量 PSNR 下降很小，约为 0.05-0.1dB。

综上所述，帧间预测改进算法 3 对于不同视频序列的影响不同，可以降低帧间预测模块运算量 5%-50%，同时不会给视频质量带来明显降低，是可以有效提高编码效率的算法。

4.2.4 改进算法 4 性能评估——修改模式判别+整像素运动估计算法

由算法 1 结果可以看出：修改了整像素运动估计算法可以在不明显降低视频质量的前提下降低帧间预测运算量（15%-20%）；同样由算法 3 结果中可以看出：修改过的模式判别算法可以在不明显降低视频质量的前提下较大幅度降低帧间预测运算量（5%-50%）。算法 4 中结合了算法 1 和算法 3 的优点，对于模式判别和整像素运动估计算法都做了修改。

首先用 20 帧视频序列 Basketball_832x480 测试算法 4 性能。HM 原始算法和修改算法 4 的视频质量和帧间运算量比较如图 4-18 和 4-19 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-10 所示。

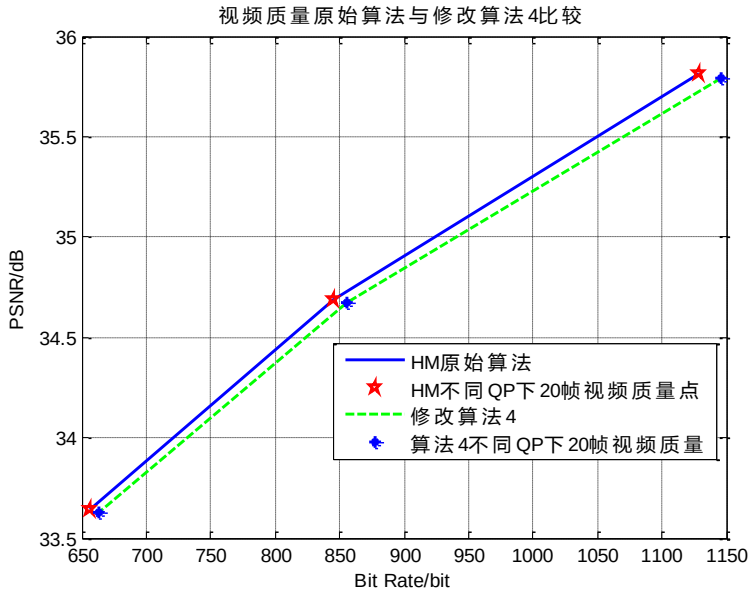


图4-18 Baketball视频序列算法4视频质量比较

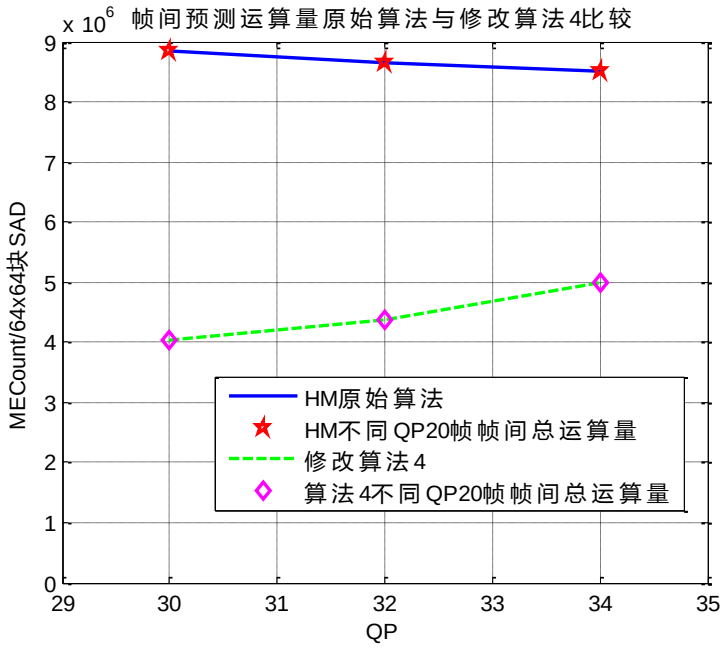


图4-19 Baketball视频序列算法4帧间预测运算量比较

表4-10 Baketball视频序列HM原始算法和修改算法4数据比较

	QP	30	32	34
HM 原来算法	PSNR	35.80997	34.689688	33.645065
	BitRate	1128.92	845.04	656
	MECount	8834487.72	8662020.97	8499995.37
修改算法 4	PSNR	35.789485	34.676495	33.630084
	BitRate	1145.7	856.12	662.76
	MECount	4027453.54	4376584.57	4985698.74

如表 4-10 所示，在原始算法中**视频**序列每个 LCU 的运算量为：

（8834487.72+8662020.97+8499995.37）/（3x19x104）=4385.38（64x64SAD）

在修改算法 4 中视频序列每个 LCU 的运算量为：

（4027453.54+4376584.57+4985698.74）/（3x19x104）=2258.73（64x64SAD）

运算量下降约为：

（4385.38-2258.73）/4385.38=48.79%

视频质量 PSNR 下降很小，不到 0.05dB。

然后用 20 帧视频序列 PartySence_832x480 测试算法 4 性能。HM 原始算法和修改算法 4 的视频质量和帧间运算量比较如图 4-20 和 4-21 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-11 所示。

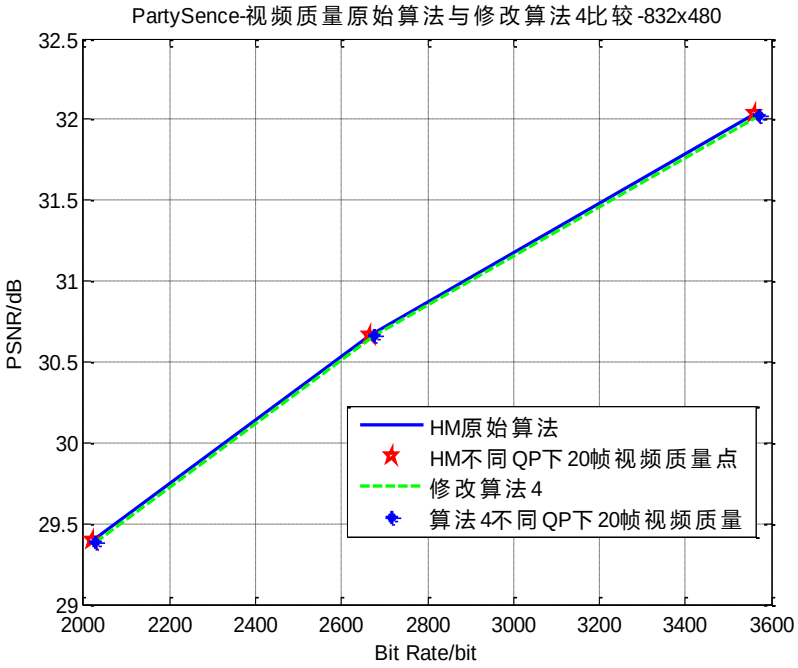


图4-20 PartySence视频序列算法4视频质量比较

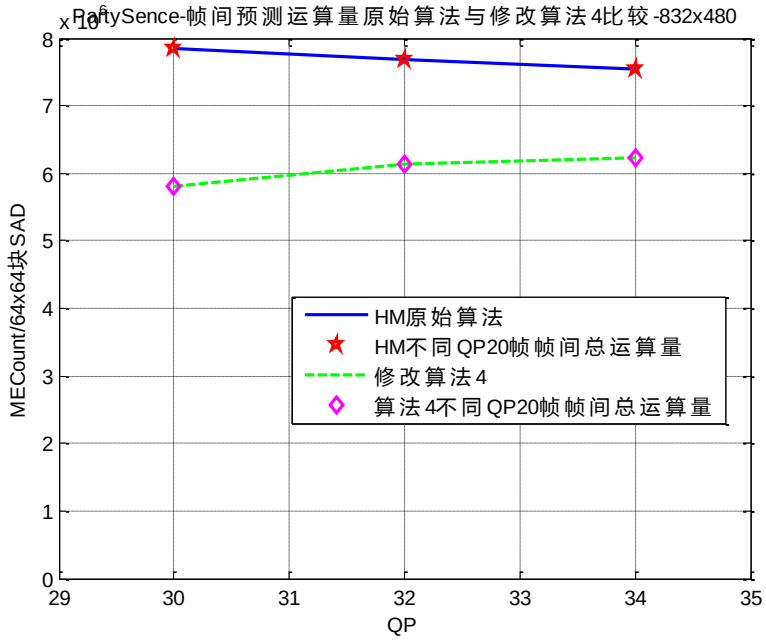


图4-21 PartySence视频序列算法4帧间预测运算量比较

表4-11 PartySence视频序列HM原始算法和修改算法4数据比较

	QP	30	32	34
HM 原来算法	PSNR	32.032452	30.666974	29.398978
	BitRate	3560.12	2666.46	2022
	MECount	7847747.56	7678533.46	7540765.41
修改算法 4	PSNR	32.022884	30.659771	29.382412
	BitRate	3571.94	2675.08	2026.42
	MECount	5816423.16	6131242.28	6222231.21

如表 4-11 所示，在原始算法中视频序列每个 LCU 的运算量为：

（7847747.56+7678533.46+7540765.41）/（3x19x104）= 3891.20（64x64SAD）

在修改算法 4 中视频序列每个 LCU 的运算量为：

（5816423.16+6131242.28+6222231.21）/（3x19x104）=3065.10（64x64SAD）

运算量下降约为：

（3891.20-3065.10）/3891.20=21.2%

视频质量 PSNR 下降很小，不到 0.02dB。

最后用 20 帧视频序列 Traffic_2560x1600（4096x2048p 截取）测试算法 4 性能。HM 原始算法和修改算法 4 的视频质量和帧间运算量比较如图 4-22 和 4-23 所示，视频序列的 PSNR、BitRate、MEcount 如表 4-12 所示。

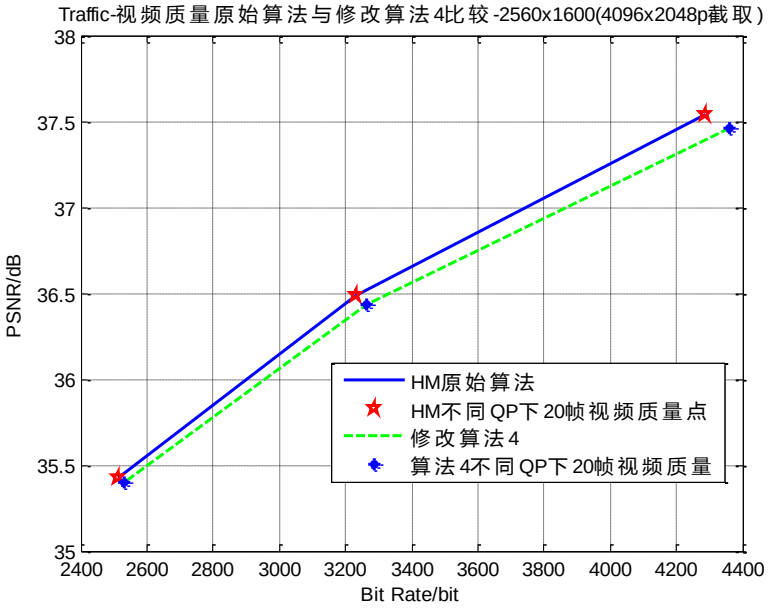


图4-22 Traffic视频序列算法4视频质量比较

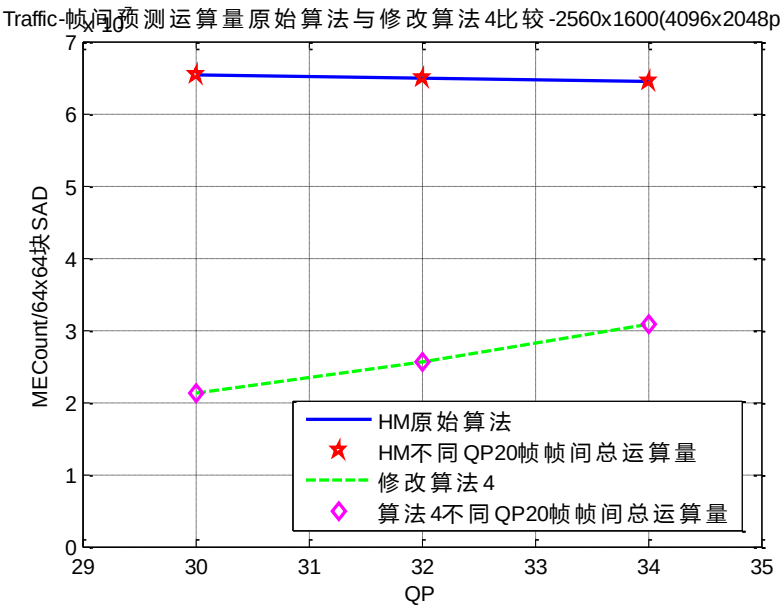


图4-23 Traffic视频序列算法4帧间预测运算量比较

表4-12 Traffic视频序列HM原始算法和修改算法4数据比较

	QP	30	32	34
HM 原来算法	PSNR	37.537207	36.484765	35.428112
	BitRate	4287.54	3231.564	2512.764
	MECount	65337049.04	64848651.16	64368444.97
修改算法 4	PSNR	37.458082	36.438164	35.394981
	BitRate	4362.576	3265.44	2532.48
	MECount	21160201.53	25645594.29	30762790.28

如表 4-12 所示，在原始算法中视频序列每个 LCU 的运算量为：

（65337049.04+64848651.16+64368444.97）/（3x19x1000）=3413.23（64x64SAD）

在修改算法 4 中视频序列每个 LCU 的运算量为：

（21160201.53+25645594.29+30762790.28）/（3x19x1000）=1360.85（64x64SAD）

运算量下降约为：

（3413.23-1360.85）/3413.23=60.13%

视频质量 PSNR 下降较小，约为 0.05-0.15dB。

综上所述，帧间预测算法 4 有效地降低了帧间预测运算量（20%-60%），且没有给视频质量带来明显降低，是有效的帧间预测算法。

4.3 本章小结

本章是对于第三章算法的具体实现，是帧间预测改进算法的软件实现和性能分析评估。帧间预测改进算法是基于 HM6.1 编码器帧间预测算法平台修改实现的。主要是对于整像素运动估计算法和模式判别算法进行了修改，目的是在不影响视频质量的同时尽量降低帧间预测算法的运算量。

改进算法 1 是针对整像素运动估计算法进行了修改，在不明显降低视频质量的前提下，可以使帧间预测的运算量下降 15%-20%，对于提高编码效率非常有利。 改进算法 2 是基于改进算法 1 的基础上对于模式判别算法进行了修改，虽然使得帧间预测算法运算量下降了

60%-70%，但是也给视频质量带来了较大损害，PSNR 下降了 0.4dB-0.6dB。改进算法 3 是针对模式判别算法进行了阈值数据统计并修改提前终止门限，实验结果结果显示该算法可以使帧间预测运算量下降 5%-50%，并没有明显视频质量损害。改进算法 4 是结合改进算法 1 和改进算法 3 的优点，有效得降低了帧间预测运算量 20%-60%，且没有给视频质量带来明显降低，是有效的帧间预测改进算法。

基于改进算法 1 至改进算法 4 的统计结果，可见改进算法 4 在不影响视频质量的前提下，有效得降低了帧间算法的运算量，基本完成了本毕设的任务。

第五章 结论

在新一代视频编码（HEVC）帧间预测标准中出现了许多新的视频压缩技术，大大提高了编码性能，但是同时会增加计算复杂度和帧间预测模块的运算量。为了保证实时 HEVC 编码器在超高清视频下的性能，本文基于 H.264 实时编码器帧间预测算法、HEVC 编码标准和 HEVC 参考编码器 HM6.1 的帧间预测算法，设计出了 HEVC 实时编码器帧间算法，并将该算法结合 HM6.1 平台设计出 4 种帧间预测改进算法，最后在 HM6.1 软件平台上进行验证。

HEVC 四叉树编码结构和对称非对称的预测单元分割使得帧间预测具有多层次、多分割的特点，又加上在 HEVC 中采用了新的 Merge 模式帧间编码标准，使得 HEVC 帧间预测模式判别算法变得异常复杂。在 HM6.1 帧间预测算法中，每个 LCU 的计算量是 3000-4000 个以 64x64 块为单位的 SAD 计算，运算量严重超过实时编码系统可以负载的帧间预测运算量。在本文中提出的实时编码器帧间预测算法和基于 HM6.1 改进的帧间预测主要分为两个部分，一个是整像素运动估计算法，一个是模式判别算法。整像素运动估计算法的搜索点个数、搜索算法、匹配准则直接影响算法性能，同时还需要考虑算法的运算量和效率，需要在两者之间达到一个平衡。模式判别算法也是对于帧间预测算法影响很大的一个算法，其中提前终止门限、阈值、分割代价存储缓存等机制也对于运算量和算法性能有很大的影响。

在 HEVC 实时编码器帧间预测算法中，采用了 SKIP 模式提前终止、Merge 模式提前终止、2Nx2N 在本层分割最优时不再进行下一层分割、hash 算法控制运动估计搜索点个数、分割存储代价缓存等机制。用于在 HM6.1 编码器平台上进行验证的 4 种帧间预测改进算法主要修改了整像素运动估计算法和模式判别算法，增加了中心运动矢量提前终止算法、2NX2N 模式判别提前终止算法、修改阈值门限等。经实验结果对比统计，这 4 种改进帧间算法达到了在不降低视频质量的前提下大幅度降低帧间预测算法运算量的目的。

本文有以下创新点：

（1）实时编码器设计中采用了分割代价缓存机制及下层 PU 只比较 Merge 模式等算法。经计算，一个 LCU 的运算量最多为 114 个（以 64x64 方框的 SAD 为单位），相比于 HM6.1 中每个 LCU3000-4000 的运算量下降颇大，是可实现的实时帧间预测算法。

（2）在帧间预测改进算法中对于采用了多步提前终止算法。在多处设立阈值进行提前终止判断，还进行预测单元模式判别分层：在 INTER_2Nx2N、INTER_2NXN、INTER_NX2N 模式比较后，若最优模式为 INTER_2Nx2N，则不再进行下层编码单元分割比较，即不再进行 INTER_NxN 模式比较。

（3）在帧间预测改进算法中对于模式判别阈值门限进行了实验统计。实验结果表明：编码单元在 64x64 分割和 32x32 分割下提前终止块数目已经可以占到视频序列块数目的一半以上，因此模式判别阈值门限设定的合理性将大大影响模式判别的效果。

本毕业设计通过对最新的视频压缩标准 HEVC 及其参考编码器 HM6.1 进行研究，并参考 H.264 实时编码器，对 HM6.1 帧间预测算法进行了修改和软件仿真，在视频性能和编码效率之间达到了一个很好的平衡。

参考文献

[1] 毕厚杰，王健. 新一代视频压缩编码标准——H.264/AVC[M]. 北京：人民邮电出版社.2009[51-95]

[2] Benjamin Bross. “High Efficiency Video Coding (HEVC) text specification draft 6”, JCTVC-H1003, 7th Meeting: Geneva, CH, 21–30 November, 2011

[3] Ken McCann (Samsung/ZetaCast). ” HM6: High Efficiency Video Coding (HEVC) Test Model 6 Encoder Description”. JCTVC-H1002 8th Meeting: San José, CA, USA, 1–10 February, 2012[20-25] [42-49]

[4] Ken McCann (Samsung/ZetaCast). “HM5: High Efficiency Video Coding (HEVC) Test Model 5 Encoder Description”, JCT-VC G1102, 7th Meeting: Geneva, CH, 21-30 November, 2011

[5] Martin Winken, Sebastian Bo ße, Benjamin Bross, Philipp Helle,etc. “Description of video coding technology proposal by Fraunhofer HHI”. JCTVC-A116 1st Meeting: Dresden, DE, 15-23 April, 2010

[6] Toshiyasu Sugio. Parsing Robustness for Merge/AMVP(D). JCTVC-F470. 6th Meeting: Torino: JCT-VC IT, 14-22 July, 2011 [5-10]

[7] T. K. Tan. “BoG report of CE9: Motion Vector Coding”. JCTVC-D441. 4th Meeting: Daegu, KR, 20-28 January, 2011

[8] Benjamin Bross. “WD5: Working Draft 5 of High-Efficiency Video Coding”, JCTVC-G1003, 7th Meeting: Geneva, CH, 21–30 November, 2011

[9] Benjamin Bross. “WD4: Working Draft 4 of High-Efficiency Video Coding”. JCTVC-F803, 6th Meeting: Torino, IT, 14-22 July, 2011

[10] Ken McCann (Samsung/ZetaCast)， Benjamin Bross (HHI)， Shun-ichi Sekiguchi (Mitsubishi)，Woo-Jin Han (Kyungwon University). “HM4: High Efficiency Video Coding (HEVC) Test Model 4 Encoder Description”. 6th Meeting: Torino, IT, 14-22 July, 2011

[11] 宋洪涛. Tilera H.264High Profile 编码器 B 帧运动估计与模式判决算法报告 上海：上海交通大学图像所 2013.3

[12] Philippe Bordes1. “Weighted Prediction”. Document:JCTVC-F265，6th Meeting: Torino, IT, 14-22 July, 2011

谢辞

首先，要感谢我的毕设指导老师高志勇老师。高老师在毕设期间一直对我悉心指导，给我指明了研究的方向，在我有问题时为我解答疑惑，教会我应该以怎样的态度和方法进行研究，对我以后的影响重大。其次，要感谢另外一位在毕设期间给与我指导和帮助的张小云老师，她在我对于工作无从下手的时候给我提出解决意见，同时也对我毕设工作的完成给出了很多建议，对于我的毕设论文的撰写也提出了宝贵意见。然后，要感谢在毕设期间细心解答我问题的邓刚学长和李向阳学长，我在遇到一些细节问题时他们给我指明了寻求结果的方向。最后，要感谢我身边的同学、朋友和家人，他们在我迷茫困惑的时候给了我很多支持，陪伴鼓励我克服各种困难，

在整个毕设过程中我学到了很多知识，也开阔了自己的眼界，进入到了一个全新的领域，收获很大。同时毕设期间我从两位老师、周围同学身上学习到的专心学术的钻研精神、一丝不苟的科研态度、踏踏实实的求实精神也使得我获益匪浅，终身受益。

INTER PREDICTION CODING AND SIMULATION OF THE NEW VIDEO ENCODER STANDARD (HEVC)

In the video encoder, one key video compression technology is inter prediction. It uses motion estimation, motion compensation and inter mode decision. As a result, it significantly reduces the code rate. This paper works on the inter prediction algorithm and proposes several new algorithms to achieve better performance between computation complexity and video quality. One of the new algorithms is for real-time encoder, the other four have improved the inter prediction algorithms of the HEVC reference encoder software HM6.1, and have been tested to have better capabilities. By changing integer pixel motion estimation and inter mode decision, these algorithms achieve better coding performance while reducing the complexity significantly without affecting the video quality obviously. The experiments in this paper are taken on the HM6.1 software platform in case of UHDV(4Kx2K——4096x2048p@30fps, 3840x2160p@ 30fps).

HEVC standard is designed to encode the UHDV video sequence. Its core objective is to achieve the same video quality with H.264/AVC while its compression efficiency is twice as H.264/AVS. To achieve this goal, the complexity of HEVC will increase largely, and maybe three times than that of H.264. In HEVC video compression standard, several new technologies have been proposed such as:

Quad-tree structured coding unit with variable seizes

Symmetrical and asymmetrical prediction unit used for inter or intra prediction

35 directions intra prediction

CABAC entropy coding

De-blocking filtering including Adaptive Loop Filter(ALF) and Sample Adaptive Offset(SAO)

extension precision option

inter prediction which uses new Merge mode

These technologies improved compression bit rate and guaranteed video quality, yet they brought about higher complexity. In the video encoder, inter prediction occupies about 50% -90% of the total computation operation: for the encoder of consumer, inter prediction occupies about 2/3 computation operation; for professional encoder, inter prediction occupies more than 70% computation operation. In conclusion, inter prediction act as an important role in video encoded process for encoder performance.

In the HEVC standard of inter prediction part, a series of new technologies have been used such as:

using Merge mode instead of SKIP mode and DIRECT mode

using spatial-temporal Motion Vector Estimation instead of only spatial Motion Vector Estimation

1/4 luma interpolation based on 8-tap separable DCT-based interpolation filter

1/8 chroma interpolation based on 4-tap separable DCT-based interpolation filter

These new inter prediction technologies reduce the bit rate effectively, especially the location

of the MVP and Merge motion vector candidates. These candidates use three methods to remove the redundancy information of the time domain such as spatial-temporal prediction, the combination of bi-directional prediction, the zero motion vector prediction. The Merge mode is the largest change of inter prediction. On the one hand, coding unit can utilize Merge mode to fulfill SKIP mode which transmit neither Motion Vection difference nor pixel difference. On the other hand, prediction unit can act as Merge mode which doesn't transmit any Motion Vector difference. A wide range of applications for Merge mode make HEVC compression result better, which proves Merge mode to be a very efficient coding method.

Moreover, this paper explained the inter prediction algorithm of HEVC reference encoder HM6.1. In HM6.1 encoder, inter prediction algorithm adopts several methods to reduce computation and improve the accuracy of the algorithm. For example, In integer pixel motion estimation, the encoder takes the integer pixel down-sampling method to reduce the computation, adaptive radius eight points diamond search to improve the search accuracy, raster search to prevent large motion vectors, the fine search after coarse search to further improve the motion estimation accuracy. In mode decision, the algorithm includes early termination for SKIP mode, prediction unit segmentation judgment, etc. These algorithms make the coding performance better.

However, the computation in the encoder is very large, every Largest Coding Unit needs about 3000-4000 64x64 SAD computation, which makes the efficiency of the encoder become so slow that real-time encoder can hardly been carried out by this inter prediction algorithm. Considering the need to reduce inter complexity for real-time encoder, this paper proposes an inter prediction algorithm for real-time encoder, which decreases the computation largely. This inter prediction algorithm adopts several new method to control complexity as follow:

the early termination of SKIP mode

the early termination of the Merge mode

2Nx2N segmentation priority principles

the hash algorithm which used to control the number of search points

cost storage mechanism

Under these mechanisms, every LCU only need at most 114 SAD computation cost based on 64x64 block, which can greatly reduce the inter prediction complexity under the premise of not affecting HEVC coding performance.

In addition, based on the inter prediction algorithm of HEVC real-time encoder and the HM6.1 encoder software, the author put forward four improved algorithms to achieve better performance. In these four algorithms, the author made a research on different parts on for integer pixel motion estimation algorithms and inter mode decision algorithms. At beginning, the author modified the integer pixel motion estimation algorithm for the improved inter prediction algorithm 1. During algorithm 1, the author increased zero motion vector early termination algorithm, modified the cumbersome refinement algorithm refined. Next, algorithm 2 was proposed based on algorithm 1 with improved inter mode decision. In this one, the author added two new early termination algorithms. One is for Merge or INTER_2Nx2N mode early termination, the other is for next coding unit early termination. By these early termination method, lots of inter prediction works could be skipped so that inter computation cost could be decrease greatly. The third improved algorithm is designed to consummate the mode decision algorithm, by changing the threshold of the early termination according to the video sequence coding statistics of the early skip threshold. The fourth improved algorithm combines the first algorithm and the third

algorithm, modified integer pixel motion estimation and inter mode decision, and achieved a better performance than HM6.1 inter prediction algorithm. This part is the main basis of the fourth chapter of the software simulation to improve the HM6.1 inter prediction algorithm.

Finally, based on HM6.1encoder inter prediction algorithm platform, the author did software simulation and analysed video performance. Through statistical data from the experiment, the performance of the algorithms are presented as follow:

The first one modified the integer pixel motion estimation, and made the inter prediction operation decreased by 15% -20%. It also made little effect on video quality. Thus the first algorithm is very beneficial for improving the coding efficiency.

The second one modified the inter mode decision based on the integer motion estimation. Although the inter prediction arithmetic operation decreased by 60% -70%, it brought greater damage to the video quality---- PSNR drops 0.4dB-0.6dB. So the second one cannot be considered as a good inter prediction algorithm. However, taking the sharp decline of the computation operations into account, the second algorithm does a good job on decreasing complexity.

In the third algorithm, the author redesign the threshold for early termination and parts of the mode decision based on the statistical data. As a result, the third algorithm decreased the inter prediction computation by 5% -50%, and there is no obvious video quality damage. Obviously, the third algorithm is also a better one than that of HM6.1.

The fourth algorithm combines the first algorithm and the second algorithm, reducing the inter prediction operations by 20% -60%. It did not reduce the video quality significantly. The fourth one is the best one among these four. Based on experiment results of these improved algorithms, we can infer that algorithm 4 does not affect the video quality, effectively reduces the computational complexity of the inter algorithm, and completes the task of this graduate design.

During the experiment, the author mainly take these video sequence 832x480 and 2560x1600 as tested video sequence. The sequence used in experiment are as follows: BasketballDrill_832x480_50.yuv, PartyScene_832x480_50.yuv, Traffic_2560x1600_30_crop.yuv. They are downloaded in HEVC website as encoder software tested sequence.

Overall, this graduation project researched on the latest video compression standard HEVC and its reference encoder software HM6.1. Based on the research above, the author proposed a new inter prediction algorithm used in the real-time encoder and four other improved algorithms applied in HM6.1 encoder. At the end of the paper, the author conducted the software simulation and proved that the algorithm outperform HM6.1 between coding efficiency and video quality.