清 华 大 学

综 合 论 文 训 练

题目：基于早终止的 AV1 视频编码加速

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：杨皖宁

指导教师：温江涛教授

2018 年 6 月 7 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

对于新生的视频编码标准 AV1，其在编码速度方面不尽如人意，通过移植在已有视频编码标准上的加速算法来加速，是有广阔的空间与前景的。本文着眼于基于早终止思想的视频编码加速算法，以剪枝策略为核心，通过不同的剪枝条件移植了几种加速算法。移植的算法是基于 K Choi 和 SH Park 等人的工作[5]，笔者根据 AV1 标准的实际情况进行了一定程度的移植、修改与改进。

本文首先介绍了 VP9 标准下，视频编码的主要流程，然后通过对比介绍了 AV1 标准的改动点；后续有简入难，介绍了三种早终止加速算法，并在随后给出了定量化的实验结果；文末对于所做的工作做了总结，并对未来的工作作出展望。

关键词：AV1；早终止；视频编码；树剪枝

ABSTRACT

The new video coding standard AV1, which is unsatisfactory in the coding speed, has a broad space and prospect through the acceleration algorithm transplanted on the existing video coding standard. This paper focuses on the accelerating algorithm of video coding based on the idea of early termination. With the pruning strategy as the core, several accelerating algorithms are transplanted through different pruning conditions.The transplant algorithm is based on the work of K Choi and SH Park et al[5]. I have carried out a certain degree of transplant, modification and improvement according to the actual situation of the AV1 standard.

This paper first introduces the main process of video coding under the VP9 standard, and then introduces the modifications of AV1 standard through comparison. Three kinds of early termination acceleration algorithms are introduced, and experimental results are given afterwards. At the end of the paper, the work done is summarized, and the future work is prospected.

Keywords: AV1; early-termination; video coding; tree pruning

目 录

主要符号对照表

HEVC	高效视频编码 (High Efficiency Video Coding)
CU	控制单元（Control Unit）
PU	预测单元（Prediction Unit）
TU	变换单元（Transform Unit）
JCT-VC	（Joint Collaborative Team on Video Coding ）
ISO/IEC	国际标准化组织 (International Standardization Organization)
IEC	国际电工委员会（International Electrotechnical Commission)
MPEG	动态图像专家组（Moving Picture Experts Group）
ITU-T	国际电信联盟电信标准分局 (ITU Telecommunication Standardization Sector)
VCEG	视频编码专家组（Video Coding Experts Group）
AVC	高级视频编码（Advanced Video Coding）
ME	运动估计（Motion Estimation）
MV	运动向量（Motion Vector）
DCT	离散余弦变换（Discrete Cosine Transform）
ADST	非对称离散正弦变换（Asymmetric Discrete Sine Transform）
CAVLC	上下文自适应可变长编码（Context Adaptive Variable Length Coding）
QCIF	四分之一公共中间格式（Quarter Common Intermediate Format）
AOM	开源媒体同盟（Alliance for Open Media）
AV1	AOM 视频编码标准一版（AOM Video-1）
RGB	红绿蓝（Red Green Blue）
PSNR	峰值信噪比（Peak Signal to Noise Ratio）
SVM	支持向量机（Support Vector Machine）

第 1 章　引言

1.1　选题背景

随着时代的发展，人们对于数字娱乐媒体形式的需求愈发从文字、图片转向视频，这一点一方面得益于网络传输技术的发展所带来的更大的带宽，另一方面，视频编码技术的进步，使得同样质量的视频所需要的数据量变小，视频体积的减小方便了其传播与分享。

自 20 世纪 80 年代起，人们开始注重对数字视、音频的压缩与存储，著名的 MPEG 正是于这一时期成立，并先后为视频编码制定了 H.261、H.263 等一系列标准，并于 2003 年与 ITU-T 的视频编码专家组 VCEG 合作，正式发布了影响深远的 H.264 视频编码标准。然而，在人们对视频分辨率追求愈发提高的今天，1080p 甚至 4k 等高分辨的视频已大幅超出了 H.264 的能力，且对于 60fps 的高帧率以及更高压缩率等要求，也无法被纳入 H.264 的框架内，H.265 应运而生。H.265 作为新一代的视频编码标准，力求在更高的分辨率、帧率、压缩率等方面成为 H.264 的上位替代。

尽管 MPEG 组织所定制的一系列视频编码标准取得了良好的效果，但其高昂的使用许可费用使得无数开发研究者望而却步。为此，Google 于 2010 年启动了 WebM 项目，该项目旨在制定高效且不需要版权费的视频编码标准。作为项目的一部分，2010 年，视频编码标准 VP8 出炉，并成功应用于 Hangouts；而 2013 年，新一代的 VP9 定制完成，并在应用于 Youtube，相较于 H.264，其减少了 30%-40% 的带宽，且这些带宽上的收益都来自于软件解码。

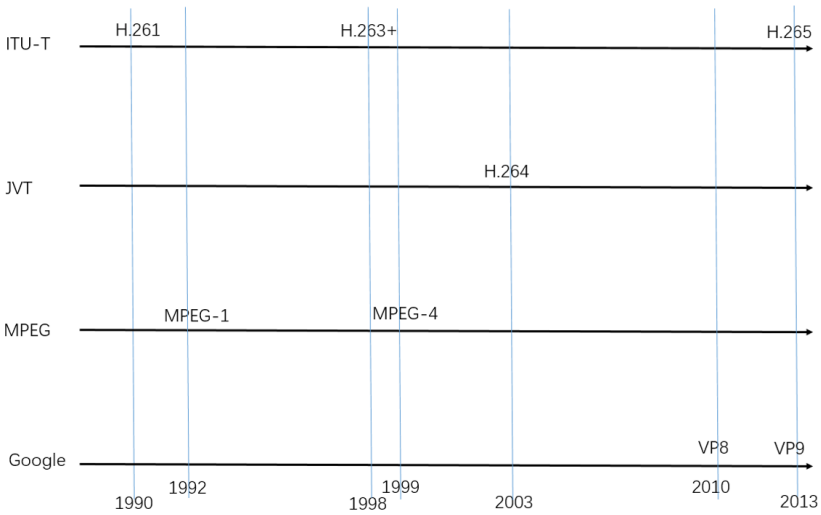


图 1.1　视频编码标准发展

本着开源、免版权税的原则，2015 年，开源媒体联盟 AOM 成立，并为定制 VP9 的接班者、下一代的开源免版权视频编码标准——AV1 而努力。近年来直播行业的兴起，使得直播系统成为互联网带宽的主要消耗者，因此 AV1 应在 VP9 的基础上进一步减少编码视频所用的比特数，且直播的特点决定了降低解码端复杂度应是新标准的目标之一。历经 3 年的开发时间，2017 年底 AV1 的代码框架初步完成，初版的编码器效率极低，耗时极长，对编码器的优化亟待进行。

AV1 虽为新生的视频编码标准，但其主体框架 H.264、H.265、VP9 等大同小异。如前所述，AV1 代码框架近期初步搭建完成，其优化空间是广阔的，又因为其在框架结构上与之前的标准大同小异，因此许多针对 H.264、H.265、VP9 等视频编码标准的优化算法，是极有可能移植到 AV1 上并展现出良好效果的。

1.2　研究内容

本文主要针对新的视频编码标准 AV1 做优化，在以视频质量几乎不变或是有轻微降低的前提下，尽量提升视频编码的速度。

考虑到 AV1 编码标准是一个新标准，其代码框架尚处于比较初步的阶段，几乎没有做过优化，因此，优化算法主要考虑将已有的、成功应用于过去的编码标准的优化算法移植过来。视频编码流程较为复杂，整个流程中有数点可以进行针对性的优化：例如在将视频帧划分为块时，既可以对划分算法本身进行优

化，也可以通过预测，尽可能早地终止划分过程，从而达到加速的目的；在运动估计方面，针对运动估计过程或是与运动矢量进行优化的算法也有许多；在预测方面，帧间预测和帧内预测两大预测模式，不同的预测算法也可能带来极大的加速。

在算法上，本文针对视频帧划分为块的这一过程进行优化。这一过程中，帧被不断划分为更小的矩形，并以树结构进行存储，对于每个树节点，如何划分为子树是一个复杂且耗时的过程。若能提前判断某一节点不需要再进行划分，则该节点的子树的计算均可被省略，由此大大提升计算效率，加速编码过程。这种通过更早地终止编码树分叉、从而实现视频编码的加速的思想，就是本文所研究的加速算法的核心。

1.3 研究现状

对于尚处于开发中的 AV1 标准，笔者没有找到较好的针对帧划分为块这一过程，以早终止为思路进行优化的相关工作。因此，笔者把目光放在 H.264 与 H.265 这两个被人研究较多的编码标准，通过考察这两种标准上的早终止优化算法现状，来进行 AV1 编码标准下的移植。

表 1.1　H.264、H.265、AV1 比较

	H.264	H.265	AV1
宏块大小	16 x 16	64 x 64	128 x 128
四叉编码树	否	是	是
划分种类	7	4	10

H.264 标准中编码树的概念比较薄弱，对于某一视频帧，首先会被划分为宏块，而每一宏块有且仅有七种划分模式，在这七种模式中选择最优的一种即是优化的关键。可能的算法包括：根据块的运动矢量 MV 做一些统计上的分析并与阈值比较，从而提前终止运动估计过程[1]；或者是 Libo Yang 和 Keman Yu 等人（2005）[2] 提出的根据宏块对应的运动矢量预测宏块的划分模式的方法等。由于 H.264 没有采用四叉编码树的结构，因此对其进行的优化工作在移植上有一定难度，但仍具有一定参考价值。

H.265 则与 AV1 更为相近，同样是采用四叉编码树进行组织，因此对 H.265

标准进行的优化工作在移植时相对容易。真对 H.265 的早终止优化，一种思路是 Shen Xiaolin 等人使用 SVM 进行分类预测[3]。该方法效果较好但缺点在于他们的工作中，对于 SVM 中特征空间的选取十分复杂，由此在移植时将带来极大的不确定性。也有一些相对清晰且易于移植的方法，例如：Jongho Kim 等人提出了基于先前跳过的 CU 的率失真代价进行预测，从而提前终止 CU 划分的方法[4]；类似的，K Choi 和 SH Park 等人也使用率失真代价作为衡量标准，提出了对编码树进行剪枝的算法[5]。可以看到，使用率失真代价进行比较，从而提前终止 CU 的划分，是一种有效且易于移植的方法。

1.4 论文结构

本文章节安排如下：

第 2 章 以 VP9 为例介绍视频编码流程，在此基础上介绍 AV1

第 3 章 介绍本项目最使用的早终止算法

第 4 章 展示实验流程及结果，并加以分析

第 5 章 总结本项目所做的工作，并对未来可能进行的改进加以展望

第 2 章　视频编码概述

2.1　VP9 视频编码标准

2.1.1　基本编码单元：块

视频流可以看成由许多图片按顺序播放组成，而这些“图片”正是视频流的基本单位：帧。尽管视频流以帧为基本单位，但在视频分辨率日益增长的今天，帧作为视频编码的基本单位过于庞大了。以 3840Œ2160 像素的典型 4K 视频为例，稍加计算我们不难发现，一帧中包含着多达 38402160 = 8294400 像素点，以 800 万以上的像素点作为基本单位，对于后续的分割、预测、变换都是相当不利的。此外，即使是较为古老的 1280Œ720 的 720p 视频，也包含着多达约 77 万的像素点。显然，作为视频流基本单位的帧，若是作为视频编码的基本单位，显得有些过于庞大了。

因此，对于视频流中的每一帧，我们将其划分为更小的成分，以作为编码的基本单位，这就是块。块的大小因编码标准的不同而不同，在针对于早年分辨率较低视频的编码标准 H.264 以及 VP8 中，块的大小都为 16x16；而在 VP9 中，为了适应更高分辨率的视频，块的大小被拓展到了 64x64。

2.1.2　块的四叉树划分

如 2.1.1 所述，在 VP9 标准中，视频流中的每一帧被划分为块，而对于每个块，还要进行一轮或是数轮的递归式划分，并以四叉树的形式进行组织。事实上，不同于每个节点会有 0 至 4 个子节点的标准四叉树，对 VP9 中用于编码的四叉树来说，其每个节点有且仅有四种划分模式：不划分、水平分割、垂直分割以及四分，如 2.1 所示。当某一节点被四分时，其四个子节点将递归地分别进行新一轮的划分。

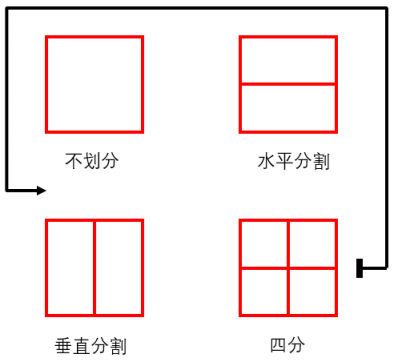


图 2.1　VP9 的四种划分模式

举例来说，假设我们对某一个 64x64 的块进行划分，过程如下：

1. 64x64 的块采用四分模式，它的四个 32x32 的子块按从左到右从上到下的顺序，继续递归划分。
2. 左上的 32x32 子块被认为不必继续划分，子块划分停止；而右上的子块则被认为应当采用水平分割模式分割为 1 和 2，然后划分停止。
3. 对左下角的块，它被四分，4 个 16x16 的子块开始新一轮的、更小尺寸的分割，他们有的被垂直分割（例如 3 和 4），有的停止分割（例如 5 和 6），也有的被四分后继续递归。这一过程持续到最小的块为 4x4 时，递归结束。
4. 右下角的块也被判定为不必再划分，整个 64x64 块的划分过程结束。

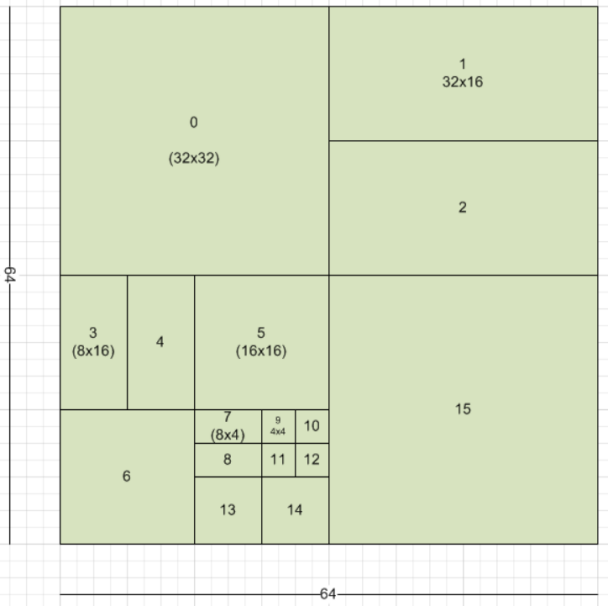
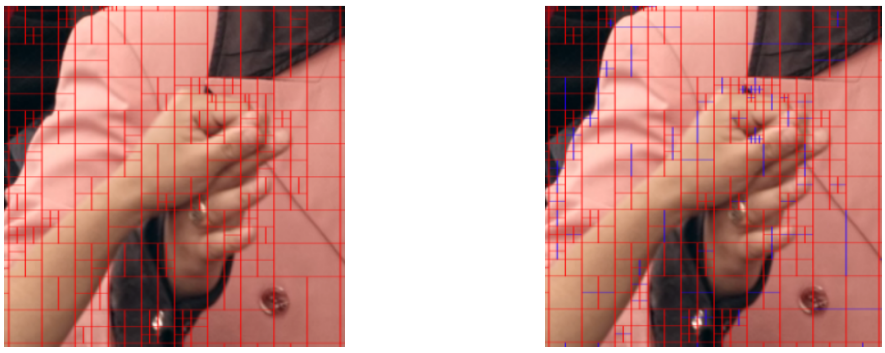


图 2.2　VP9 四叉树划分示例

在将块划分为四叉树结构的过程中，每一叶子节点将被标记，以便后续的处理。首先是段编号，该编号允许后续的变换过程中按块使用量化器或是环路滤波；此外，段编号还可用于编码固定的参考块，或是将块标记为跳过，这一点在处理视频流中的静态背景时是十分有效的。与此同时，有一些叶子节点将被打上跳过标记，拥有这种标记的块没有残差系数。另外，还需要决定节点的预测模式，分为帧内预测与帧间预测两种，两种预测模式将在 2.1.4 和 2.1.3 中详细介绍。需要明确的是，将块组织为四叉编码树，很大程度上是为了后续的变换过程。在 VP9 中，变换的基本单位是 4x4、8x8、16x16、32x32 的块，对于由水平分割或是垂直分割产生的非正方形子块，将被分为两个或是多个部分进行变换，即，一个块是可以包含多个变换块的。



(a) 块的四叉树划分　　(b) 变换单元的划分

图 2.3　块与变换单元组织示意

2.1.3　帧间预测

当一些块完成编码后，就可以作为后续块编码的参考使用。当一个块决定使用帧间预测模式时，它通常会去寻找与它相近的 1 到 2 个已编码的块来作为参考，该块与用来参考的块两者间的向量被称为运动矢量。之所以会去寻找参考块，是基于视频流的画面通常是连续的、不会发生体突变的这样一个事实，因此，在相邻的两帧中，我们可以期望在块级别上会有相当多的块是非常相似的。当使用 2 个参考块时，我们采用两个参考块预测值的平均值来作为当前块最终的预测值。并且在 VP9 中，参考块以 1/8 像素为步长进行移动，由此尽可能保证当前块能找到相似度尽可能高的参考块，在运动补偿中，将用插值的方式对不足 1 的亚像素点进行生成。

VP9 编码标准中，一个包含 8 个参考帧的缓存序列会被动态地维护，每当有

一帧需要被编码时，序列中的 3 个可参考帧会被选取，作为该帧的参考列表，这 3 个可参考帧中会分别进行运动矢量的搜索，再选出最优的。对于每个运动矢量的编码，其具有四种模式：NEARESTMV、NEARMV、ZEROMV 和 NEWMV。ZEROMV 表明该块没有移动，具有为零的运动矢量。除 ZEROMV 模式外的其余模式中，参考运动矢量列表根据附近块和先前帧中的该块生成，并做排序。当模式为 NEARESTMV 或 NEARMV 时，该块将使用来参考运动矢量列表的第一个或第二运动矢量。如果模式为 NEWMV，则该块将使用全新的运动矢量，将全新运动矢量与 NEARESTMV 模式中的差值送至运动矢量列表的第一个位置，与 NEARESTMV 模式中的运动矢量相加生成新运动矢量。

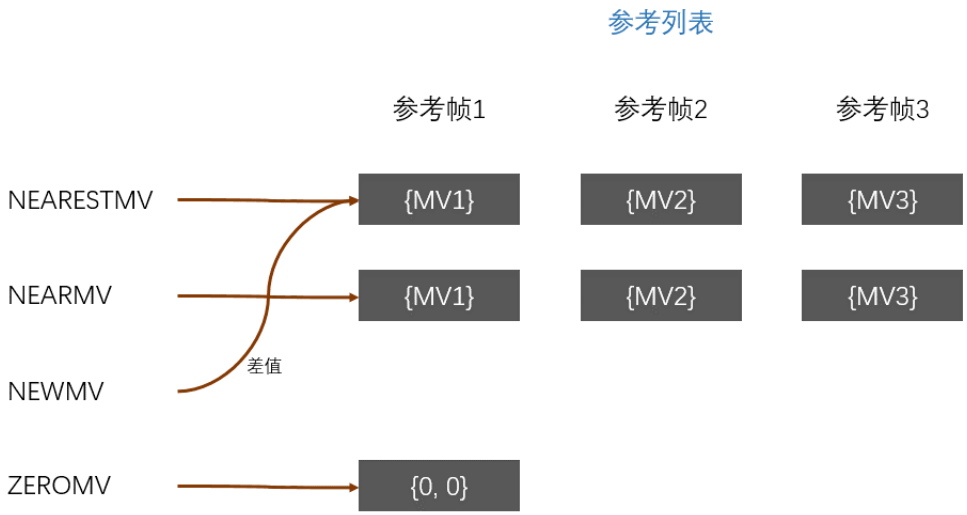
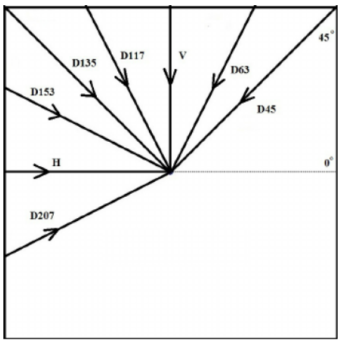


图 2.4　运动矢量参考列表

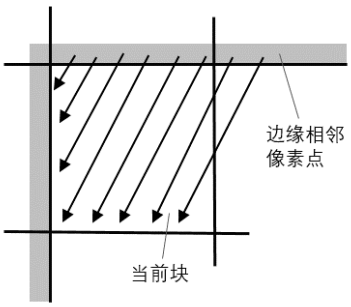
2.1.4　帧内预测

当帧间预测无法给出令人满意的参考块时，帧内预测也是一种可以考虑的预测模式。当前块的左、左上、上、右上等方向的相邻像素点可以做为参考，用于预测。VP9 制定了总共 10 种帧内预测方式，其中有 8 种是按方向预测，一种是 TM 模式，另一种是 DC 模式。

按方向预测是指 VP9 中规定了 8 种方向，根据当前块的具体情况，选择一个合适的方向，从该方向上的已编码的块中，寻找在边缘相邻的像素点，并以此预测当前块中对应像素点的值。



(a) VP9 帧内 8 种定向预测



(b) 以 63 度为方向进行预测

图 2.5　按方向预测模式与示例

TM 为 True-Motion 的缩写，该模式使用当前块的左上方已编码相邻块的 3 个像素点，对当前像素点的值使用 2-1 进行预测。

$$p(x,y) = L_x + T_y - TL \tag{2-1}$$

这里，$p(x,y)$ 为当前待预测像素点的预测值，L_x 为当前像素点左侧已编码块中的相邻像素点，T_y 为当前像素点上方已编码块中的相邻像素点，TL 则是左上角块中的相邻像素点。2.6 给出了简单示意图，蓝色部分示意了一个 4x4 的当前块，灰色部分标识了位于左上方向的 9 个已编码块中的像素点，这些像素点被用来预测当前块中的各个像素点。一般地，对于一个 NxN 的块，TM 模式使用左上方的 $2N+1$ 个相邻已编码像素点，根据 2-1 给出预测值。

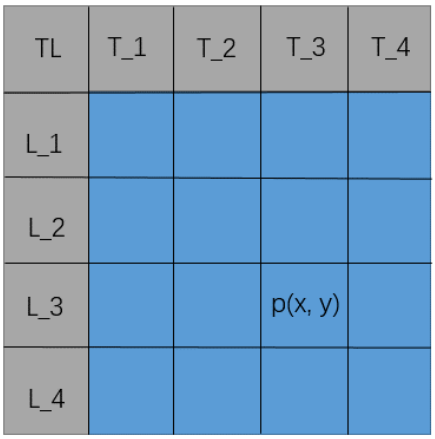


图 2.6　对 4x4 的块进行帧内预测

最后一种 DC 模式则是 Direct-Current，该模式使用均值对整个块中的像素点进行预测，如 2-2 所示。

$$p(x,y)=\frac{1}{2N}(\sum_{i=1}^{N}T_i+\sum_{i=1}^{N}L_i) \tag{2-2}$$

这里，不妨认为当前块是一个大小为 NxN 的块，DC 模式将左和上的 $2N$ 个相邻已编码像素点的值做平均，将该平均值作为块内每个点的预测值。

2.1.5 残差编码

首先给出 DCT 和 ADST 两种变换的数学公式（一维情形），如 2-3 和 2-4 所示。

$$X_n=\sum_{n=1}^{N}x_n cos(\frac{(2n-1)(k-1)\pi}{2N})\quad k=0,...,N-1 \tag{2-3}$$

$$X_n=\sum_{n=1}^{N}x_n sin(\frac{(2n-1)(2k-1)\pi}{4N})\quad k=0,...,N-1 \tag{2-4}$$

为便于理解，我们不妨假设我们有原始帧 A，经过划分为四叉树、帧间预测、帧内预测等一系列过程，我们得到了由预测像素值构成的帧 A'，则残差为 $A-A'$，对该残差矩阵进行变换、编码即可。所谓变换，是指对残差矩阵在水平、垂直两个维度上，分别使用 DCT 或是 ADST 变换，依据预测模式的不同，两个维度上具体使用哪种变换也会不同，具体规则如下：

- 对于仅依赖上方相邻像素点的帧内预测模式（帧内按方向预测中的垂直预测模式），水平方向上使用 DCT 而垂直方向上使用 ADST
- 对于仅依赖左侧相邻像素点的帧内预测模式（帧内按方向预测中的水平预测模式），水平方向上使用 ADST 而垂直方向上使用 DCT
- 对两侧相邻像素点都有依赖的帧内预测模式（TM 以及按向下或右的方向帧内预测模式），两个方向上都应当使用 ADST
- 所有帧间预测、DC 以及其他按方向预测，两个方向上都使用 DCT

变换后的残差矩阵会经过量化过程，使右下角的部分高频低能量分量被过滤掉，从而减少所需编码的数据量。

2.1.6 后续过程

视频编码的整个过程是较为漫长而复杂的，如 1.2 所述，对视频编码标准的优化可以针对其中的某一个步骤进行，考虑到本文主要针对将块进行四叉树划分这个过程，且可能需要结合或是影响到预测以及变换过程，因此对这些部分进行了较为详细的介绍。后续的编码过程在本文中的重要性相对较低，因此仅作简单的介绍。

在完成残差编码后，变换块和预测块间会出现极不平滑的边缘，为了平滑这些边缘，VP 标准使用了环路滤波器。总计 4 个环路滤波器在块的边缘独立工作，修改 16、8、4 或 2 个像素点以平滑过渡。较小的变换只允许小的滤波器，而最大的变换可以用任意滤波器处理，具体选取哪个取决于滤波器强度和边缘硬度。

最后，VP9 标准使用二进制算术范围编码器，每个符号具有与其相关联的概率表，这些概率既可以是默认值，也可以基于先前帧的解码数据的熵，在下一帧的编码开始前动态更新，这是一种能使概率有效适应数据熵的方法。

2.2 AV1 视频编码标准

2.2.1 概述

如 1.1 所述，AV1 在编码框架上与 VP9 等现行的编码标准是一致的，且标准的制定上很大程度上以 VP9 作为基础。事实上，这一标准原定应为 VP10，但 2015 年底成立的 AOM 负责其制定后，最终定为 AV1。因此，在 2.1 已经对 VP9 的关键环节做了较为详细的介绍的基础上，对于 AV1 以对比的形式进行介绍，是十分合理且易于理解的。

接下来几部分依次介绍了 AV1 与 VP9 在四叉编码树的划分上、帧间预测、帧内预测等方面的不同。最后，对于其他一些本文不重点讨论的步骤上两者的差异，也进行了一些简要的介绍。

2.2.2 块的划分

相较于 VP9 将帧划分为 64x64 的块，AV1 为了适应更高分辨率的视频，采用 128x128 作为帧的划分尺寸，避免了处理更高分辨率视频时块数量的爆炸。在块的划分上，AV1 仍采用四叉编码树的组织形式。

相较于 VP9 的 4 中块划分模式，AV1 将其拓展到了如 2.7 所示的多达 10 种。VP9 中的 4 种传统划分方式得到了保留，在此基础上，水平划分 HORZ 和垂直划分 VERT 又拓展出了新型的“T”型分割：在 HORZ 的基础上，将上方的子块再次分割所得到的变种分割方式被称为 HORZ_A；而将下方子块再次分割的变种则称为 HORZ_B；同理 VERT 也有两种被称为 VERT_A 和 VERT_B 的变种。此外，水平四等分 HORZ_4 和垂直四等分 VERT_4 也被加入到了 AV1 的划分模式中，由此构成了总计多达 10 种的划分模式。需要注意的是，仅 SPLIT 模式生成的子块参与递归式的下一轮划分，HORZ_A、HORZ_B、VERT_A、VERT_B 中的正方形子块并不会再次参与划分。

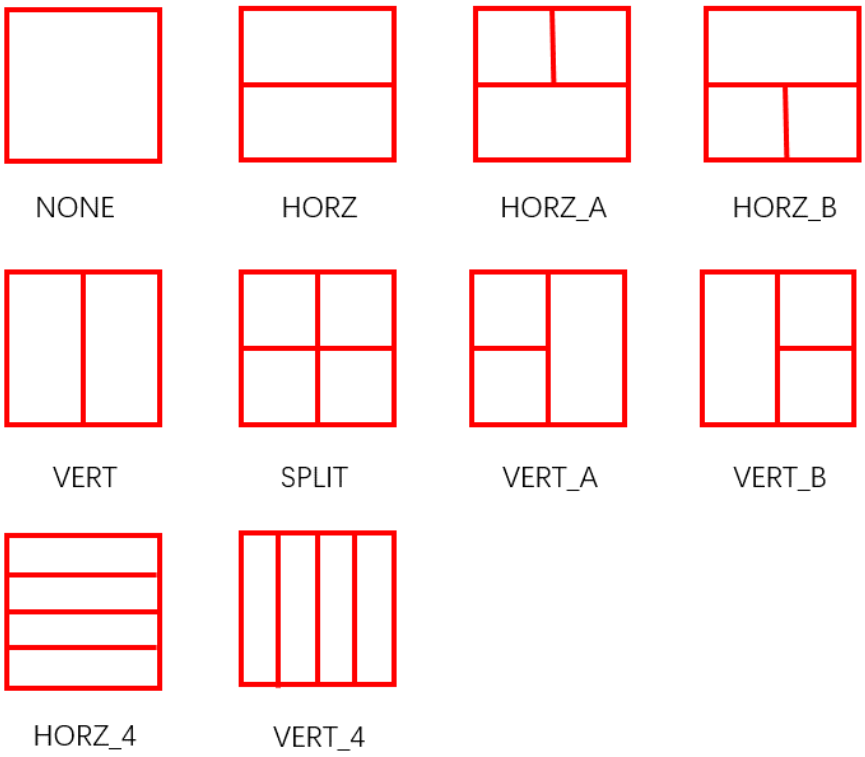


图 2.7　AV1 中 10 种块划分方式

除了划分模式的拓展，在对尺寸较小的子块处理的灵活性上，AV1 也进行了改进。VP9 对于 4x4 的子块有着较多的限制，包括需要将参考块与差值滤波器结合使用、帧内与帧间混合的预测模式在块的尺寸小于 8x8 时不能使用等。而 AV1 则放开了部分限制，在色度层上，允许以 2x2 的尺寸进行帧间预测，但最小变换单元尺寸仍保持 4x4；在亮度层，允许较小尺寸的块使用帧内、帧间混合预测模式，但在色度层不允许；为了保持硬件吞吐率与 VP9 一致，AV1 仍禁止小

于 8x8 的块使用复合模式（指帧间预测中使用多于一个运动矢量预测的模式）。

2.2.3 帧间预测

在帧间预测上，AV1 与 VP9 一样动态地维护一个长度为 8 的帧缓存序列。不同的是，VP9 标准规定从其中选取 3 帧用于参考，而 AV1 将这一数字扩大到了 7，即会从缓存序列中选择 7 帧作为参考。在运动估计的过程中，VP9 会在周围块进行扫描，相邻块的运动向量纳入候选序列中，三个参考列表每个会维护 2 个运动向量。而 AV1 标准进行了拓展，一方面，相邻块的搜索范围有所扩大，使得更多的运动向量有机会进入参考运动向量列表；另一方面，在每个参考列表中，AV1 会维护四个参考运动向量，且不同于 VP9 中参考列表中向量的排名与搜索顺序有较强的相关性，AV1 使用概率模型对参考运动向量做排序，使得靠前的结果有着更高的可信度。

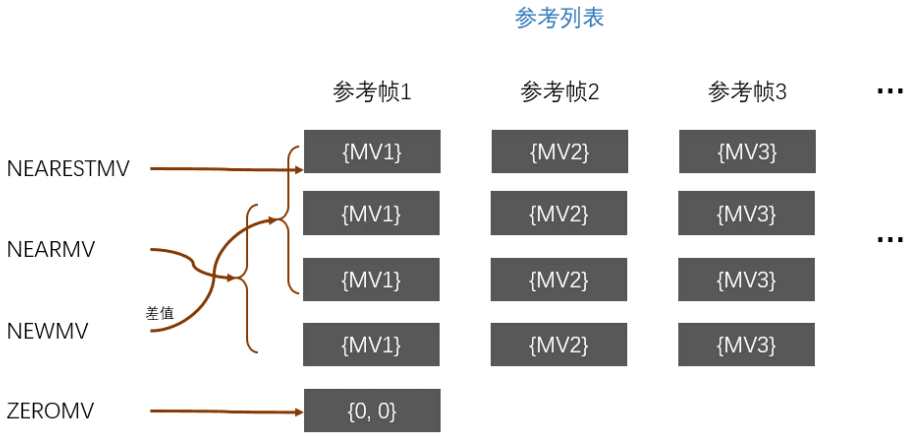


图 2.8　AV1 运动矢量参考列表

在复合模式上，VP9 仅支持两运动向量以 1/2、1/2 为比例进行复合，且这两个运动向量不能是同侧的。AV1 中放开了对同侧的限制，即其允许两同侧的运动向量复合使用，进行预测；此外，AV1 中的复合更为广义，不再局限于两种帧间预测器复合，帧间-帧内的复合也是可行的。

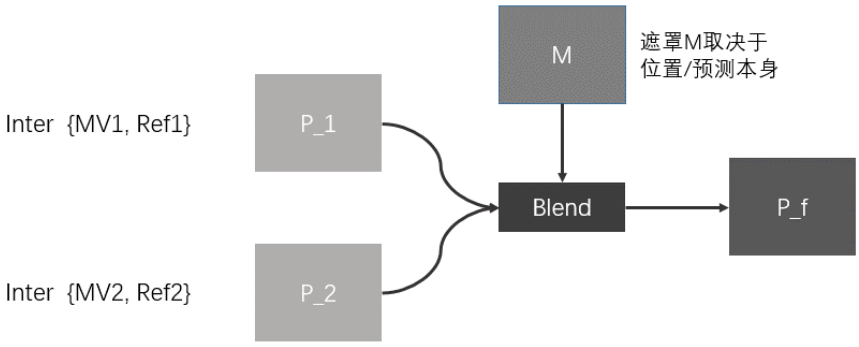


图 2.9　帧间-帧间复合模式

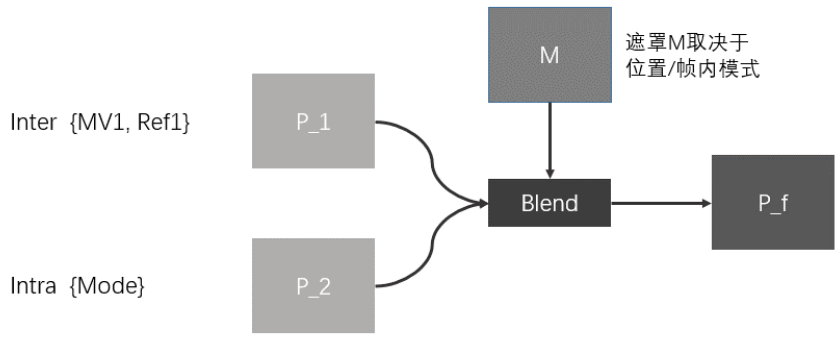


图 2.10　帧间-帧内复合模式

2.2.4　帧内预测

在帧内预测上，VP9 中的 8 种按方向预测的模式得到了保留，并且根据块的尺寸的不同，所能使用的预测方向被拓展到了 56 个。根据不同的块的尺寸选择更加细致的预测方向，对于预测结果的准确性是有着积极意义的。而非按方向预测中的 TM 模式被 Paeth 所取代，仍以 2.6 为例，Paeth 模式将根据公式 2-5 给出预测值。

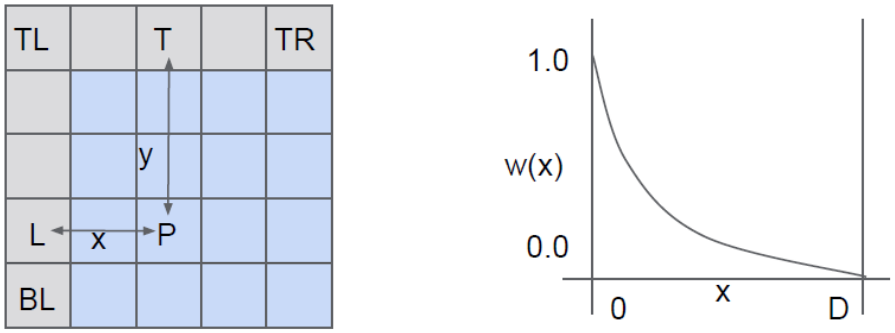
$$P_{Paeth}(x,y) = argmin|x - P_{TM}(x,y)|,\ x \in \{L_x, T_y, TL\} \tag{2-5}$$

这里，$P_{Paeth}(x,y)$ 为 Paeth 模式给出的预测值，他由 $P_{TM}(x,y)$（TM 模式给出的预测值）与 T_y、L_x、TL 三个候选点差的绝对值中的最小者决定。这一方法

会选择梯度最小的方向作为预测的方向，以计算量有所增加为代价，能给出比TM 模式更好的预测结果。

此外，非固定方向的预测方法中还加入了平滑模式，该模式分为水平、垂直、双边三种子模式。该模式加入平滑因子函数 $w(x)$，根据像素点在块中位置的不同使用不同的平滑因子，将候选点的值做线性加权来给出预测值。需要说明的是，2.11(b) 给出的仅为趋势示意图，实际实现时，$w(x)$ 是离散的，根据块的大小预先设定了一系列的离散值供预测函数使用。在水平平滑模式中，选取左侧和右上角的像素点，并选用像素点在水平方向的距离计算平滑因子；垂直平滑模式则选取上方和左下角的像素点，并选用垂直方向的距离生成平滑因子；双边模式则结合上述两种模式给出的预测值做平均，作为最终的结果。2-6 给出了公式化的描述，P_{SMOOTH_H}、P_{SMOOTH_V}、P_{SMOOTH_B} 依次为水平、垂直、双边平滑模式。

$$\begin{aligned} P_{SMOOTH\_H} &= w(x)L + (1 - w(x))TR \\ P_{SMOOTH\_V} &= w(y)T + (1 - w(y))BL \\ P_{SMOOTH\_B} &= \frac{1}{2}(P_{SMOOTH\_H} + P_{SMOOTH\_V}) \end{aligned} \tag{2-6}$$



(a) 平滑模式预测示意图　　(b) 平滑因子函数 $w(x)$

图 2.11　帧内预测平滑模式

此外，VP9 中的 DC 模式得到了保留，额外新加入的还有调色板模式（Palette Mode）等。

2.2.5 其他

在对残差矩阵的变换、编码上，VP9 依据预测方式的不同，从 $\{DCT, ADST\}^2$ 选择二维变换对残差矩阵进行处理。在 AV1 中，仍是分水平、垂直两个方向分别作一维变换，不同的是，除 DCT 和 ADST 外，还加入了 ADST 的逆变换（Flip ADST）以及恒等变换（IDTX），由此将变换的可能性拓展到了 $|\{DCT, ADST, FlipADST, IDTX\}^2|$ 总计 16 种。此外，不同于 VP9 仅支持正方形的块作为变换的基本单位，AV1 对一些尺寸的矩形块作为变换的基本单位有了支持。

第 3 章　早终止加速算法

3.1　H.265 树剪枝早终止算法

3.1.1　原算法原理

在 1.3 部分曾提到，K Choi 和 SH Park 等人在 HEVC 上针对四叉树划分过程提出了优化算法，该算法使用率失真代价作为衡量标准，在满足一定条件时对四叉树进行剪枝，从而节省了子节点的计算量，以达到加速的目的。该算法是清晰而简洁的，作为跨视频编码标准来移植的对象是十分合适的，因此，笔者选定该算法作为移植的最初对象。

首先对原算法做介绍。原算法针对 H.265 标准提出，与 AV1 有所不同，在四叉编码树的组织上，H.265 中使用 CU、PU、TU 来实现对编码单元我的组织，PU 和 TU 是在 CU 划分的基础上进一步划分而来的，因此，我们不妨仅关注对 CU 的划分优化。原文中提出的一种较为直接的思路是，当当前 CU 节点的代价低于该节点子节点的代价时，我们就不必再对子树进行进一步的处理，公式化的表述如 3-1 所示。

$$RDCost(CU_t) < \sum_{i=0}^{3} RDCost(CU_{t-1}^{i}) \tag{3-1}$$

这里，CU_t 是当前节点，CU_{t-1}^{i} 为子节点，$RDCost$ 为率失真代价。这一方法问题在于子树的代价必须是已知的，这一要求使得计算复杂度的减少程度是有限的。因此，为了更大限度地减少计算复杂度，原文提出了新的剪枝公式 3-2。

$$(m' = \underset{m \in Mode}{argmin}\, RDCost(CU_t|PU = m)) \leq Threshold \tag{3-2}$$

这里，Mode 表示 H.265 中预测模式的集合，而 m' 则是为当前 CU 深度选择的预测模式，Threshold 的值由 Mode 决定。即新的剪枝公式通过为不同的预测模式定制不同的阈值，并将节点代价与该阈值进行比较进行剪枝，从而避免了子树代价必须已知的条件。根据上述分析，绘制原算法的流程图如 3.1 所示。需

要说明的是，该流程图只是一个将主要过程及分支点抽出后的示意图，实际情况更为复杂，例如叶子节点的处理以及非四分模式最优时递归也会停止等情况。

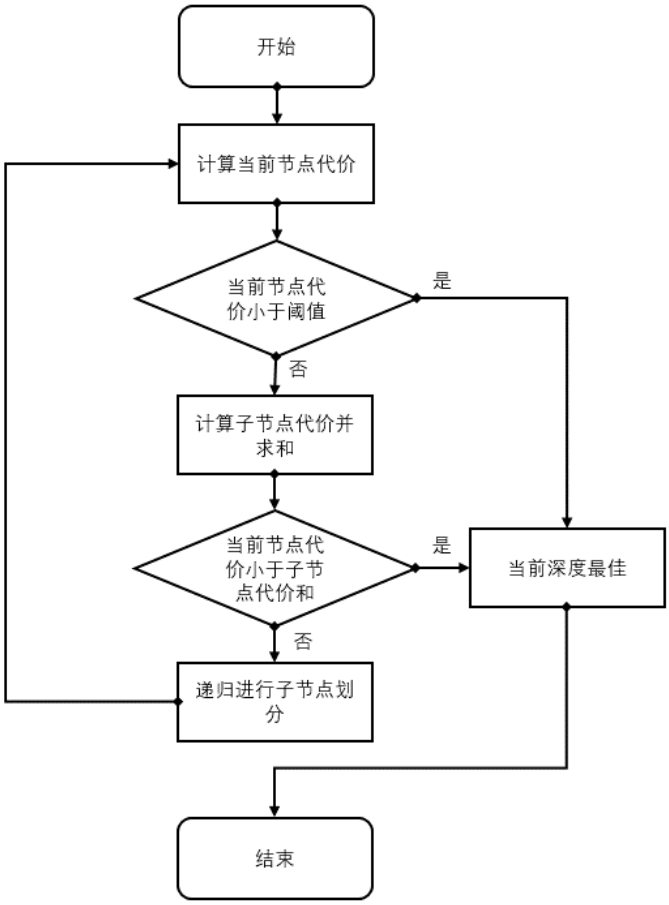


图 3.1　H.265 树剪枝早终止算法流程图

3.1.2　原算法成果

原论文中，编码的表现以 $\Delta Bitrate[(B_{PRO} - B_{REF})/B_{REF} * 100]$ 和 $\Delta PSNR(P_{PRO}-P_{REF})$ 来衡量，并且时间的减少以 $\Delta Time[(T_{PRO}-T_{REF})/T_{REF}*100]$ 来衡量。所选用的测试视频为 H.265 上所使用的五类典型测试用视频。

在以可忽略不计的画面质量降低以及比特率损失为代价的前提下，相较于 HM3.0 编码器，该方法可以提供最差 28%、最好 58 % 的时间缩减，平均来看，该方法能有效节省编码时间约 40%。这是十分出色的成果，在移植算法时，该方法是十分清晰的，因此有着很高的可移植性，尽管 AV1 标准与 H.265 有诸多不同，但如此优秀的成果在移植后，我们可以期望其有着不错的表现。

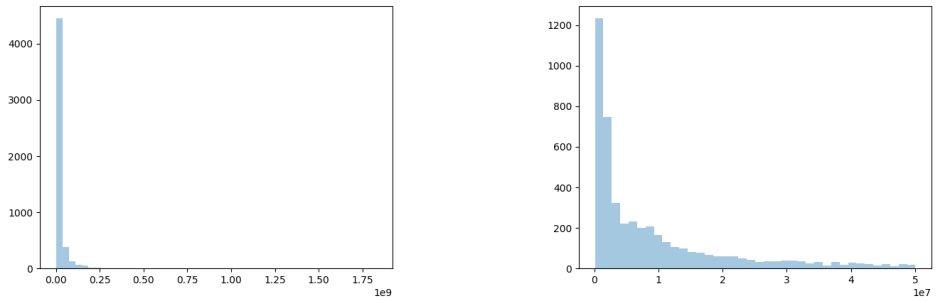
3.1.3 原算法缺陷

尽管原算法有着简明的形式和优异的表现，但观察剪枝的核心条件 3-2 不难发现，Threshold 的值对于剪枝至关重要，该阈值完全决定了当前节点的子节点代价是否需要计算。然而，原论文中仅指出 Threshold 的值应根据预测模式进行选取，而该值具体是多少、应当如何确定，原文并没有给出详细的策略。因此，在移植时，针对单一模式确定 Threshold 的值是十分困难的。甚至 H.265 中，所需要应对的模式仅有 $\{SKIP, Inter2Nx2N, Inter2NxN, InterNx2N, InterNxN\}$ 五种，而在 AV1 中，除去常规的上述几种外，还引入了如 2.7 所示的 T 型划分和水平/垂直四分，这使得 Threshold 的数量增多，测定难度进一步加大。

3.2 AV1 树剪枝早终止算法

3.2.1 阈值测定的尝试

较大的阈值测定难度并不妨碍笔者对 Threshold 值的推断作出一些初步的努力，笔者以 park_joy_90p_8_420 这一 AV1 上的典型测试用视频进行探究，设定编码后视频的比特率为 200，对各节点 RDCost 的值做一些统计。被统计节点共 5166 个，这些节点中，RDCost 最小值为 68,393、最大值 1,829,405,236、平均值为 23,431,775、中位数为 5,683,608。可以看到，RDCost 的极差是非常大的。更直观的分布示意图如 3.2 所示，不难看出，RDCost 主要分布在值较小的区域内。

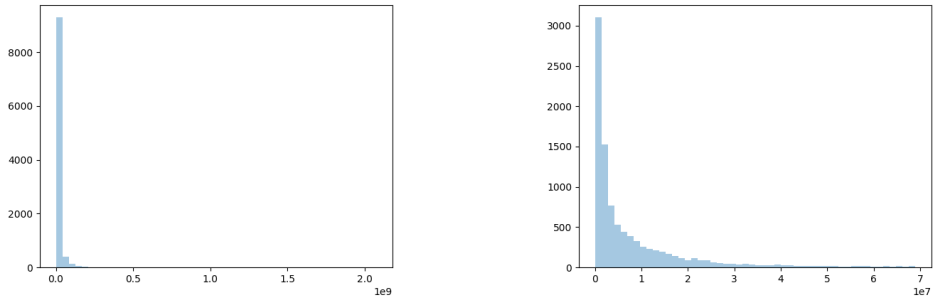


(a) park_joy 视频 RDCost 分布直方图　　(b) 去除 500 个最大值后的直方图

图 3.2　park_joy 视频 RDCost 分布

同样是对 park_joy_90p_8_420 视频限定比特率为 200 时，在测定 RDCost 的同时，笔者还记录了当时节点所存储的各个模式中的最佳 RDCost 的值，并做统

计分析。在最佳 RDCost 的统计中，最小值 581、最大值 2,077,618,636、平均值 24,020,189、中位数 3,490,404。直方图如 3.3 所示，可以看到最佳 RDCost 的分布趋势与 RDCost 一致，都集中在数值较小的范围内，且平均来看数值更小。

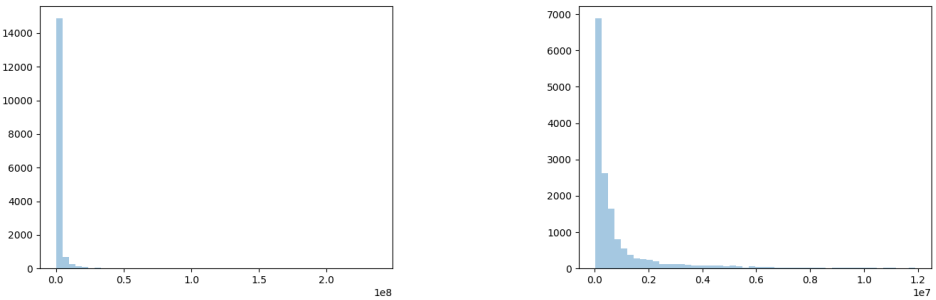


(a) park_joy 视频最佳 RDCost 分布　　(b) 去除 500 个最大值后的分布

图 3.3　park_joy 视频最佳 RDCost 分布

由上述分析可以看出，确实可以通过选择一个值作为 Threshold 来过滤掉 RDCost 大于该值的节点划分模式，从而实现加速。例如，可以得知 80% 的最佳 RDCost 小于 16,037,601，因此我们可以较为保守地选取该值作为 Threshold，当当前节点所对应模式的 RDCost 大于该值时，该模式不大可能是该节点的最优划分方式。

当然，上述的结论仅是针对 park_joy_90p_8_420 这一个视频成立的。笔者又对另一个 AV1 测试用视频 miss_am_qcif 进行了测试，当限定比特率为 200 时，得到的分布结果如 3.4 图所示。



(a) miss_am_qcif 视频 RDCost 分布　　(b) 去除 500 个最大值后的分布

图 3.4　miss_am_qcif 视频 RDCost 分布

对测试用视频 miss_am_qcif 编码过程中各节点的 RDCost 分布进行统计，得到其最小值 15754、最大值 236,766,106、平均值 1,905,168、中位数 349,230。显然，这与 park_joy 视频所得到的数值差了一个数量级，若仍以之前针对 park_joy 提出的值作为 Threshold，则该阈值将几乎无法做出有效的筛选。

由上述两个个例的测试结果，不难发现在 AV1 编码标准下，针对不同的视频，RDCost 的分布差距可能是巨大的，因此，针对每种划分模式提出一个合理的、适用于绝大多数视频的 Threshold 是十分困难的。因此，在时间精力有限的前提下，笔者放弃了通过对大量视频测试筛选 Threshold 的思路，退而求其次，使用效果不那么好、但确定性强、基于与子节点代价和比较的剪枝策略。

3.2.2 AV1 中的 RDCost

尽管在整个早终止树剪枝优化中都不会对 AV1 的 RDCost 计算方式做改变，但仍需要对这种代价计算方式做简单的介绍：RDCost 为率失真代价，是一种在率失真优化过程中用于比较的代价。R 指的是码率，D 则是指失真程度，在高码率的条件下，R-D 有着如 3-3 所示的关系。

$$R(D) = \alpha ln(\frac{\delta^2}{D}) \tag{3-3}$$

这里 $R(D)$ 为码率，D 为失真程度，δ^2 为方差，α 为某种系数。不难看出，图像失真的减小意味着码率的增大，而要减小码率则要以图像失真为代价，因此，R-D 间的平衡就是率失真优化过程的意义所在，而这一寻找平衡过程中用来比较的代价正是 RDCost。

AV1 中的 RDCost 函数实现如 3-4 所示，整个优化过程中，RDCost 的计算方法是一致的，而该式背后所蕴含的数学原理较为复杂，所使用的常数其依据也相对繁琐，并且不是本文所关注的重点，因此在此不做过多分析，仅将其理解为一种合理的衡量节点计算复杂度的代价值即可。

$$\begin{cases} RDCOST(RM, R, D) = (R2(R * RM, 9) + D * 128) \\ R2(value, n) = \lfloor \frac{value + 2^{n-1}}{2^n} \rfloor \end{cases} \tag{3-4}$$

3.2.3 子节点和剪枝早终止算法

为了解决无法通过对大量视频进行测试选定 Threshold 的困境，笔者选用了效果不够好但确定性强的剪枝算法，即基于与子节点代价和比较的剪枝策略，其剪枝条件的公式化描述如 3-5 所示。

$$RDCost(T_t) < \sum_{i=0}^{m} RDCost(T_{t-1}^{i}) \tag{3-5}$$

这里，$RDCost(T_t)$ 是当前节点的 RDCost，m 为子节点个数，完全由划分模式决定，$RDCost(T_{t-1}^{i})$ 为某一子节点的代价值。即该算法中，当假定当前节点按某一模式划分后，若该节点的代价值小于其全部子节点的代价之和，可以认为该节点按该模式划分会使计算复杂度增大，由此排除该划分模式。算法的流程图如 3.5 所示。

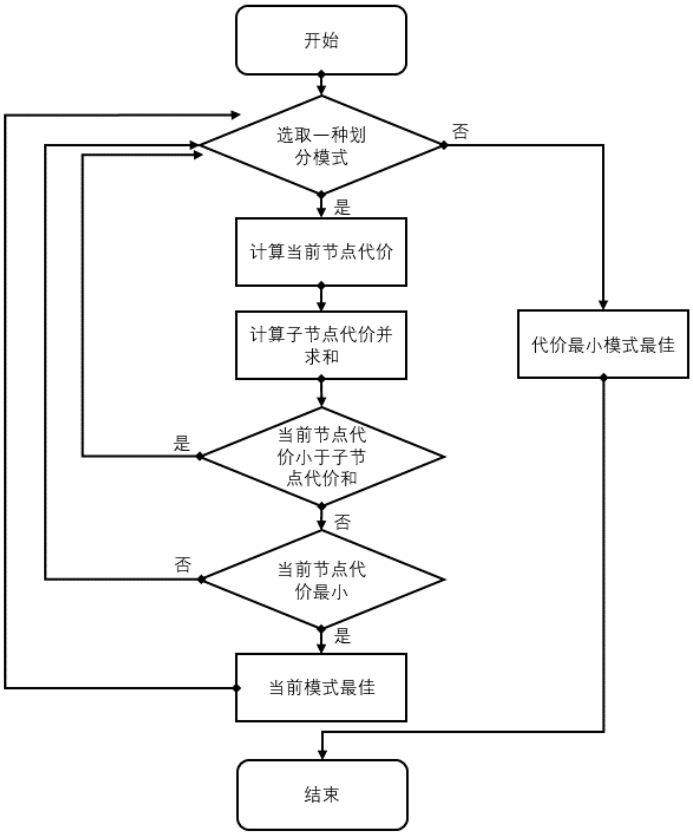


图 3.5　AV1 子节点和剪枝早终止算法流程图

3.2.4 子节点加权和剪枝早终止算法

基于子节点和的早终止剪枝算法能在一定程度上降低计算复杂度，起到加速视频编码的作用。再不对大量视频做测试选取阈值的前提下，笔者对基于子节点和的早终止剪枝算法稍作改进，以期更好的加速效果。考察 AV1 视频编码标准中的子节点划分模式，不难注意到如 3.6 所示的四种划分方法是“不规则”的。

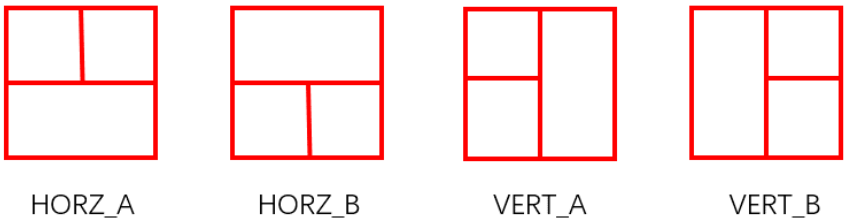


图 3.6　AV1 中四种“不规则”划分方式

这里的“不规则”是指相较于其他五种划分方式，这四种 T 型划分所分出的子节点的大小是不一致的，即一大两小。为方便讨论，不妨认为当前节点的块大小为 $2Nx2N$，则我们考虑大小为 $Nx2N$ 的子块。首先从所包含像素点个数的角度来看，$Nx2N$ 的子块包含更多的像素点；其次从后续的变换环节来看，尽管相较于 VP9，AV1 对非正方形子块直接参与变换有了一定的支持，但其仍有着更大的计算量。因此，在计算子节点和时，笔者希望 $Nx2N$ 的子块会更显著地影响子节点的代价和，将子节点代价做加权和的想法应运而生，其公式化的表述如 3-6 所示。

$$RDCost(T_t) < \lambda * RDCost(T_{t-1}^0) + RDCost(T_{t-1}^1) + RDCost(T_{t-1}^2) \tag{3-6}$$

上式中，T_{t-1}^0 为 $Nx2N$ 的子块，而 T_{t-1}^1、T_{t-1}^2 则是 NxN 的子块，λ 为权重。一个自然的想法是将权重值取为 2，实验证明，取权重 $\lambda = 2$ 时确实可以进一步地加速视频的编码，关于加速程度及比特率、画质损失的比较将在后续 4 部分给出定量的结果。

3.2.5 阈值剪枝早终止算法

如 3.2.1 所分析，提出一个适用于绝大多数视频的 Threshold 是十分困难的，但是针对测试用的视频集合，设计一个合理的 Threshold 来加速确是可能的。因此，基于与阈值比较进行剪枝的早终止算法也被移植到了 AV1 平台上，其剪枝条件如 3-7 所示。

$$(m' = \underset{m \in Mode}{argmin}\, RDCost(T_t)) \leq Threshold \tag{3-7}$$

Mode 指 AV1 标准中的九种子节点划分方式，针对不同 Threshold 的值所取得的不同成果将在 4 部分详细展示。

第 4 章　实验结果

4.1　子节点和剪枝算法实验结果

首先给出在 park_joy_90p_8_420 这个视频上的实验结果。在使用基于子节点和剪枝的早终止算法前后，考察视频编码在比特率、画质、编码时间三方面的变化，以下标 PRO 指代笔者所提出的算法、而 REF 指代不做改进时的情况，比特率的变化以 $\Delta Bitrate[(B_{PRO}-B_{REF})/B_{REF}*100]$ 来衡量；画质的变化则用 PSNR 度量，比较时使用 $\Delta PSNR(P_{PRO}-P_{REF})$ 作比较；而编码时间的比较也使用相对减少的百分比 $\Delta Time[(T_{PRO}-T_{REF})/T_{REF}*100]$ 衡量。具体结果如 4.1、4.2、4.3 所示，需要说明的是，对单一视频测试时，需要预先设定期望的比特率，表格中以 $Bitrate_{cond}$ 标识。

表 4.1　子节点和剪枝算法 park_joy 实验（比特率）

$Bitrate_{cond}$(kb/s)	$Bitrate_{REF}$(b/s)	$Bitrate_{PRO}$(b/s)	$\Delta Bitrate$(%)
200	297160	294880	-0.77
250	321160	319600	-0.49
300	344760	342120	-0.77
350	368800	365560	-0.88
400	392200	391320	-0.22
450	416800	413520	-0.79
500	437720	436760	-0.22

表 4.2　子节点和剪枝算法 park_joy 实验（PSNR）

$Bitrate_{cond}$(kb/s)	$PSNR_{REF}$(dB)	$PSNR_{PRO}$(dB)	$\Delta PSNR$(dB)
200	34.576	34.809	0.23
250	35.573	35.780	0.21
300	36.420	36.591	0.17
350	37.135	37.250	0.12
400	37.755	37.936	0.18
450	38.305	38.464	0.16
500	38.861	39.073	0.21

表 4.3　子节点和剪枝算法 park_joy 实验（编码时间）

$Bitrate_{cond}$(kb/s)	$Time_{REF}$(ms)	$Time_{PRO}$(ms)	$\Delta Time$(%)
200	492815	445677	-9.56
250	545196	423004	-22.41
300	541272	450985	-16.68
350	560691	481394	-14.14
400	573799	479245	-16.48
450	540554	490609	-9.24
500	597913	509196	-14.84

可以看到，针对 park_joy_90p_8_420 这个视频，基于子节点和剪枝的早终止加速算法有着不错的效果：在 PSNR 与比特率几乎没有变化的情况下，所需要的编码时间缩减幅度最小有 9.24%，最大有 22.41%，平均来看，这一算法能够使编码时间缩短 14.76%。此外，笔者还在 miss_am_qcif 等视频上做了相同的实验，结果中，比特率和 PSNR 几乎不变，而编码时间的变化如 4.4 所示。

表 4.4　子节点和剪枝算法 miss_am_qcif 实验（编码时间）

$Bitrate_{cond}$(kb/s)	$Time_{REF}$(ms)	$Time_{PRO}$(ms)	$\Delta Time$(%)
200	35468689	31944684	-9.94
250	37811001	33324426	-11.87
300	39371983	36279519	-7.85
350	44189312	42550631	-3.71
400	49631606	44715847	-9.90

可以看到在体积更大的视频 miss_am_qcif 上测试时，基于子节点和剪枝的早终止算法产生了最差 3.71% 最好 11.87% 的编码时间缩短，平均来看该算法减少 8.65% 的编码时间。编码时间的缩短幅度较 park_joy 上的更小，可能是因为随着视频的增大，编码树的结构更为复杂，较深的节点上的剪枝带来的收益变小所致。

4.2　加权子节点和剪枝算法实验结果

与基于子节点和剪枝的早终止算法实验相同，首先给出在 park_joy_90p_8_420 这个视频上的实验结果，在比特率、画质、编码时间三方面的对比如 4.5、4.6、4.7 所示。

表 4.5　子节点加权和剪枝算法 park_joy 实验（比特率）

$Bitrate_{cond}$(kb/s)	$Bitrate_{REF}$(b/s)	$Bitrate_{PRO}$(b/s)	$\Delta Bitrate$(%)
200	297160	295760	-0.47
250	321160	319480	-0.52
300	344760	343000	-0.51
350	368800	365680	-0.85
400	392200	390760	-0.37
450	416800	412600	-1.00
500	437720	435640	-0.48

表 4.6　子节点加权和剪枝算法 park_joy 实验（PSNR）

$Bitrate_{cond}$(kb/s)	$PSNR_{REF}$(dB)	$PSNR_{PRO}$(dB)	$\Delta PSNR$(dB)
200	34.576	34.857	0.28
250	35.573	35.685	0.11
300	36.420	36.541	0.12
350	37.135	37.238	0.10
400	37.755	37.922	0.17
450	38.305	38.494	0.19
500	38.861	38.996	0.14

表 4.7　子节点加权和剪枝算法 park_joy 实验（编码时间）

$Bitrate_{cond}$(kb/s)	$Time_{REF}$(ms)	$Time_{PRO}$(ms)	$\Delta Time$(%)
200	492815	414570	-15.87
250	545196	392822	-27.94
300	541272	425908	-21.31
350	560691	431319	-23.07
400	573799	456575	-20.43
450	540554	462260	-14.48
500	597913	474520	-20.64

同样可以看到 PSNR 与比特率变化相差无几，而编码时间进一步减少。在 miss_am_qcif 视频上的实验结果如 4.8 所示，仅给出编码时间上的对比。

表 4.8　子节点加权和剪枝算法 miss_am_qcif 实验（编码时间）

$Bitrate_{cond}$(kb/s)	$Time_{REF}$(ms)	$Time_{PRO}$(ms)	$\Delta Time$(%)
200	35468689	27476545	-22.53
250	37811001	28443458	-24.77
300	39371983	32556881	-17.31
350	44189312	38573681	-12.71
400	49631606	38613336	-22.20

可以看到相较于基于子节点和剪枝的算法，使用加权和能进一步降低视频

编码所用的时间。一点有趣的现象是，在体积更大的视频 miss_am_qcif 上，基于子节点加权和剪枝的早终止算法仍表现出了不错的效果，这可能是由于 AV1 标准对于四种新的 T 型划分在后续编码流程中的处理较为薄弱，对这四种 T 型划分的严格筛选可以有效加速编码过程。

4.3 阈值剪枝算法实验结果

如 3.2.1 所分析，笔者首先选取了 80% 最佳 RDCost 的临界点 16,037,601 作为 Threshold 进行实验，实验结果上，PSNR 与比特率几乎不变，这里不再列出；而所用编码时间上，时间缩短约 10% 至 20% 不等。这一实验结果与基于子节点和的剪枝算法所得到编码时间缩减程度近似。

表 4.9　阈值剪枝算法 park_joy 实验（Threshold = 16037601）

$Bitrate_{cond}$(kb/s)	$Time_{REF}$(ms)	$Time_{PRO}$(ms)	$\Delta Time$(%)
200	492815	437619	-11.20
250	545196	432917	-20.59
300	541272	441926	-18.35
350	560691	475368	-15.21
400	573799	461355	-19.60
450	540554	487155	-9.88
500	597913	512709	-14.25

接下来笔者尝试了更多的阈值，分别选取了位于最佳 RDCost 分布中 50%、55%、60%、70%、80% 的值作为阈值进行实验，结果如 4.10 所示。可以看到选择过小的阈值会导致许多节点产生不好的剪枝，其结果是使得编码所用的时间不降反升；而当阈值的选取较为保守时，则可以产生节省 10% 左右编码时间的优化效果。

表 4.10　不同阈值在 park_joy 上的实验结果

Threshold	Bitrate(b/s)	$\Delta Bitrate$	PSNR(dB)	$\Delta PSNR$	Time(ms)	$\Delta Time$
349230	296800	-0.12%	34.695	0.12	613047	24.40%
421811	298200	0.35%	34.663	0.09	643915	30.67%
518589	298360	0.40%	34.668	0.09	448292	-9.03%
787617	298360	0.40%	34.668	0.09	457853	-7.09%
16037601	296920	-0.08%	34.709	0.13	437619	-11.20%

将这些阈值应用在 miss_am_qcif 视频上进行测试，结果如 4.11 所示。

表 4.11　不同阈值在 miss_am 上的实验结果

Threshold	Bitrate(b/s)	$\Delta Bitrate$	PSNR(dB)	$\Delta PSNR$	Time(ms)	$\Delta Time$
349230	123163	-0.54%	45.746	-0.01	37543458	5.84%
421811	123495	-0.27%	45.753	-0.003	43106798	21.53%
518589	123709	-0.10%	45.752	-0.004	43311828	22.11%
787617	123391	-0.36%	45.758	0.002	42662776	20.28%

该结果表明笔者所选取的基于 park_joy 上的简单统计所得到的阈值均不适用与 miss_am_qcif 视频。根据趋势分析，适合该视频的阈值可能会更小。由这两个视频上的测试可以看出，选取一个合适的阈值用于剪枝是非常困难的，一种可行的思路应当是根据一些易于获取的视频特征，如分辨率、码率等，来确定阈值。

第 5 章　结论

5.1　工作总结

随着人们对数字媒体娱乐要求的提高，传统视频编码标准将难以处理高分辨率的视频。而正处在定制中的 AV1 视频编码标准一方面有望适应更高分辨率的 4K 等高清视频，实现编码时间、压缩率等方面的提升；另一方面，作为 VP9 的下一代编码标准，其作为新一代免版权税开源视频编码标准的代表，将有力地推动视频编码相关的研究及应用。

作为最新一代的视频编码标准，AV1 距离实际使用仍有一段距离，现有实现框架在性能上难以令人满意。本文正是以对 AV1 标准的实现进行优化、缩短编码时间为目的，着眼于已有视频编码标准上的早终止算法进行移植，通过移植、简化、改进等方法提出并实现了三种剪枝早终止策略来加速。基于子节点和进行剪枝的早终止加速算法简明而有效，在保持比特率和视频质量几乎不变的前提下，将编码时间缩短了 15% 左右。在此基础上进行改进，针对 AV1 中四种“不规则”的划分方法提出了基于子节点加权和进行剪枝的早终止算法，将编码时间进一步缩短，达到 20% 左右。基于阈值剪枝的早终止加速算法受限于实验时间有限，无法给出一个普遍使用的阈值，但在所进行的有限的实验的基础上仍得出了一些结论，为未来的工作做出了铺垫。

5.2　进一步的工作

本文所提出的算法虽然取得了一些成果，但受限于时间因素，一方面实现的算法仍有较大的改进余地，另一方面很多从其他角度出的早终止算法也有移植的可能。笔者虽然无法实现这些算法，但仍可以做一些展望：

- 完善基于阈值剪枝的早终止加速算法。对更多的视频进行分析，提出一个更为合理、普遍适用的阈值，或是总结出一套计算阈值的方法，都是可行的改进思路。
- 使用运动向量的早终止加速算法。可以参考的工作为 Libo Yang 等人提出的方法[2]。根据运动矢量，尤其是零运动矢量进行早终止优化，是一种可行性很高的方案。

- 使用 SVM 对节点的划分方式进行预测的早终止加速算法。已有的工作有 Shen Xiaolin 等人的成果可以参考[3]。由于 SVM 本身属于较为经典且成熟的算法，因此该方法的可移植性也相对不错。而难点在于一方面特征空间的选取上可能性较多，由此带来很多的不确定性；而另一方面则是 AV1 中的划分方式有九种，分类的增多意味着移植算法的效果可能会下降。
- 针对 T 型划分做优化。AV1 中最具体创新性的就是子节点的四种 T 型划分方式，针对这四种划分方式做优化，可能是使 AV1 标准胜过其他已有视频编码标准的一个突破点。

本文提出的算法不尽完美，希望对于后续的相关工作起到一定的铺垫作用，也希望针对 AV1 标准的优化早日达到一个令人满意的程度，进而投入到实际应用中。

插图索引

表格索引

公式索引

参考文献

[1] Xu J, Chen Z, He Y. Efficient fast me predictions and early-termination strategy based on h. 264 statistical characters[C]//Information, Communications and Signal Processing, 2003 and Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on: volume 1. [S.l.]: IEEE, 2003: 218-222.

[2] Yang L, Yu K, Li J, et al. An effective variable block-size early termination algorithm for h. 264 video coding[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2005, 15 (6): 784-788.

[3] Shen X, Yu L. Cu splitting early termination based on weighted svm[J]. EURASIP Journal on Image and Video Processing, 2013, 2013(1): 4.

[4] Kim J, Jeong S, Cho S, et al. Adaptive coding unit early termination algorithm for hevc[C]// Consumer Electronics (ICCE), 2012 IEEE International Conference on. [S.l.]: IEEE, 2012: 261-262.

[5] Choi K, Park S H, Jang E S. Coding tree pruning based cu early termination[J]. JCTVC-F092, Torino, Italy, 2011: 14-22.

[6] 薛瑞尼. ThuThesis: 清华大学学位论文模板[EB/OL]. 2017. https://github.com/xueruini/thuthesis.

致 谢

衷心感谢温江涛老师在毕业设计及综合论文训练过程中的指导与帮助。在思路的拓展以及参考文献的查找过程中给予了我很多帮助。感谢实验室的谷嘉文学长和陈馨瑶学姐的帮助，在选题工作上，学长和学姐帮助对视频编码完全没有了解的我入了门，并选择了合适的题目，在现有工程代码的阅读上也给予了我极大的帮助。

感谢 LaTeX 和 ThuThesis[6]，在论文排版上帮助我减少了很多麻烦。感谢五班吐槽团提供的硬件支持，也感谢室友及班里的其他同学在论文写作期间提供的帮助。

声 明

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

签　名：_____ 日　期：_____

附录 A　书面翻译

A.1　基于编码树剪枝的 HEVC 快速编码单元推断方法

基于编码树剪枝的 HEVC 快速编码单元推断方法

摘要：本文为 HEVC 提出了一种快速决定编码单元划分的方法，这是一种通过基于树剪枝的策略，来更早推断出编码单元的尺寸的方法。在 HEVC 中，最为有效的是可变的编码单元尺寸，这是一个全新的概念。在推断最佳的编码单元尺寸的过程中，作为参考的 HEVC 编码器会逐个测试每种可能的编码单元尺寸，以便估测每种情况下，由编码单元尺寸所决定的编码器的表现。在整个编码过程中，这是主要的计算复杂度来源，因此，为了实现一个高效快速的编码器，这一过程应是攻坚的重点。一种简单的树剪枝算法是，若当前节点的编码模式已经足够好，那么其子节点的计算均可以被跳过（例如 SKIP 模式）。实验结果显示，本文所提出的方法，相较于测试模型 3.0 下的 HEVC 编码器，最多有可能达到 40 % 的时间缩减，而其代价仅仅是一点可以忽略不计的编码效果上的损失。在第六次 JCT-VC 会议上，本文所提出的方法被 HEVC 编码器测试模型 4.0 所采用。

关键词： HEVC；编码单元早推断；快速高效视频编码；视频编码。

A.1.1　引言

2010 年，ISO/IEC 旗下的组织 MEPG 和 ITU-T 旗下的 VCEG 拟定了一份新的视频编码标准，该标准被称为 HEVC。这份新标准被期望能够满足人们对于视频日益增长的需求，包括压缩效率、视频分辨率、帧率、计算复杂度等诸多方面。尽管 HEVC 标准尚处于发展阶段，迄今为止，其压缩效率已经达到了前一标准（MPEG-4 AVC/H.264）的二倍之高。

虽然 HEVC 遵循传统的编码结构，包括基于块的运动补偿和变换编码，但实质性改进来自于基于四叉树结构的新的分层编码概念。在 HEVC 中，视频编码和解码过程由以下三个单元组成：编码单元（CU），用于作为变换四叉树的根，以及用于帧间/跳过/帧内预测；预测单元（PU），用于决定模式，包括运动估计和率失真优化；变换单元（TU），用于变换编码和熵编码。在三种编码单元中，

CU 对于提升压缩效率是至关重要、最为关键的，因为它决定了初始编码块的大小，而初始编码块的大小对其他处理单元（例如，PU 和 TU）的性能影响很大。

当使用 CU、PU 和 TU 三种编码单元时，提升压缩效率是可能的，但同时，这也会显着地增加计算的复杂度。通常，在编码过程中会生成一个四叉树，其中根节点表示最大的 CU 大小，并且具有对应于下一个 CU 大小的四个子节点。这棵树被称为编码树，且其最大深度为 4。编码单元与编码树（例如，当根节点的 CU 大小为 64 x 64 且树的深度设定为 4 的时候，CU0 大小为 64 x 64，CU1 为 32 x 32，...，CU3 为 8 x 8。）被包含在 CU 尺寸优化的计算与选择中，这一点导致了 CU 数量的指数增长（例如，1 x CU0 + 4 x CU1 + 16 x CU2 + 64 x CU3），这一点被直观地展示在 A-1 中。由于每一个 CU 都对应着一个相应的 PU 和 TU 的计算，因此，日益增长的计算复杂对给实时编码器的设计带来了了严峻的挑战。HEVC 参考编码器（即，3.0 版）大约比 MPEG-4 AVC/H.264 参考编码器（即，JM 17.0 高配置版本）慢三倍。为了克服 HEVC 编码器在计算上的复杂性，我们提出了基于 CU 早终止的树剪枝策略，并在 JCT-VC 第六会议上介绍了该方法。

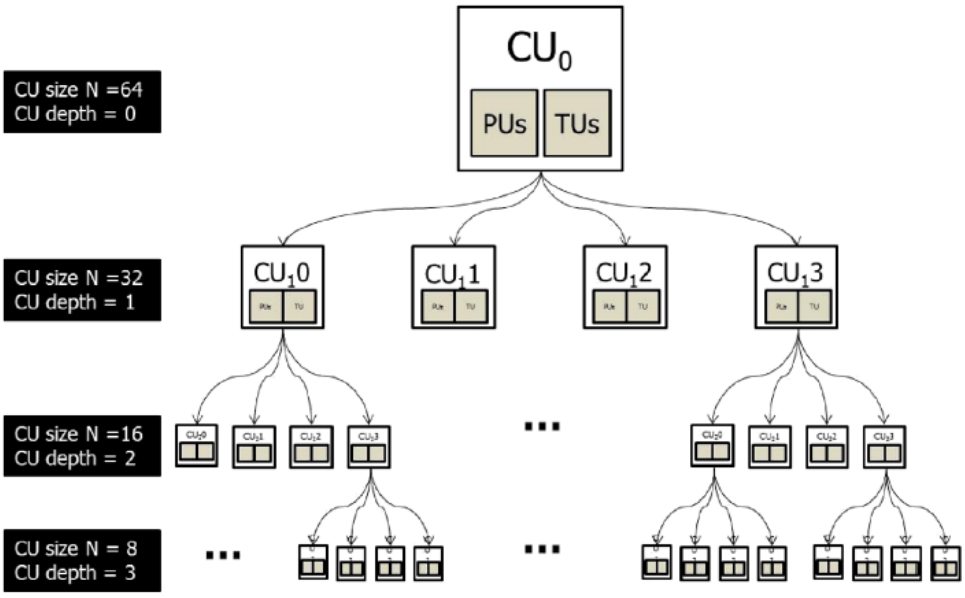


图 A-1　根节点 CU 为 64 x 64、深度为 4 的编码树结构示意图

A.1.2　基于树剪枝的快速编码单元推断

基于可变 CU 大小的编码树推断过程计算复杂度可以描述如下：

$$\begin{cases} f(n) = f(n-1) + 4^n * M_n, f(0) = M_0 \ \ and \\ M_i = (\frac{1}{4})^i * M_{i-1} \end{cases} \tag{A-1}$$

这里，$f(n)$ 是当编码树的最大深度被设定为 n 时所需操作的总数，而 M_i 则是对于第 i 层给定的 CU 大小所需的操作总数。在 A-1 中，我们假定 M_i 是 M_{i-1} 的四分之一是因为 CU 的大小比起前一层会减少四分之一。总的操作数可以被整合为如下的公式：

$$f(n) = \sum_{i=0}^{n} 4^i * M_i \cong O(M_0 * n) \tag{A-2}$$

如 A-2 所展示，计算的复杂度随着 CU 深度的增加单调增加。当考虑到 M_i 代表着包括对每个 CU 大小的运动估计在内的整个编码过程这一事实，最大 CU 深度就自然成为了决定编码时间的主要因素。

本文研究了一种通过更早地确定 CU 的深度来加速 CU 深度推断的策略。我们利用了这样一个事实,当当前 CU 节点的代价低于该节点子节点的代价时,我们就不必再对子树进行进一步的处理（例如,$RDCost(CU_t) < \sum_{i=0}^{3} RDCost(CU_{t-1}^i)$）。这一方法的唯一问题是，子树的代价必须是已知的，这一要求妨碍了子树计算复杂度的减少程度。我们可以避免子树代价的计算，如果当前节点的代价已经是最小的了（例如，SKIP 模式），而这正是本方法的核心概念。为了进一步减少计算复杂度，我们可以定义剪枝的条件如下：

Pruning codition :

$$(m' = \underset{m \in Mode}{argmin}\, RDCost(CU_t | PU = m)) \leq Threshold \tag{A-3}$$

$$Mode = \{SKIP, Inter2Nx2N, Inter2NxN, InterNx2N, InterNxN\}$$

这里，Mode 表示按根据复杂度确定的预测模式的有序集合，而 m' 则是为当前 CU 深度选择的预测模式，且 Threshold 将根据 Mode 选择。在 A-3 中，通过从 SKIP 到 InterNxN，不断改变 Threshold 的值，剪枝过程的可能性将被推断。

若 Threshold 被设置为 SKIP 时，剪枝过程将不会对压缩效率产生影响；否则，以降低压缩效率为代价，更多的子树将被剪去。根据上述分析，我们提出了 A-2 所示的剪枝过程。

```
Algorithm proposed CU decision
Recursive_CU_Processing(depth,index){
  parent_cost = CU_processing(depth,index)
  if (selected prediction_mode ≤ Threshold
      Best_CU = CU(depth)
    pruning remaining processes
  else
    for from index = 0 to to index = 3 do
      children_cost+ = Recursive_CU_Processsing(depth + 1,
index)
    end
    if(parent_Cost ≤ children_cost)
      Best_CU = CU(depth)
    else
      Best_CU = CU(depth+1)
    if(leaf node)
      return
}
```

图 A-2　基于树剪枝的 CU 推断方法的步骤

A.1.3　实验结果

为了进行性能评测，我们测试了了我们的方法与 HM 3.0 两种方法的总执行时间，以确认计算复杂度的降低程度。编码的表现以 $\Delta Bitrate[(B_{PRO} - B_{REF})/B_{REF}X100]$ 和 $\Delta PSNR(P_{PRO} - P_{REF})$ 来衡量，并且时间的减少以 $\Delta Time[(T_{PRO} - T_{REF})/T_{REF}X100]$ 来衡量。在实验中，我们吧 Threshold 值设置为 SKIP 模式。有关编码环境的更多细节在 A-3 被展示。

Test Sequences	• Class A (2560 × 1600): Traffic and People On Street • Class B (1920 × 1080): Kimono, ParkScene, Cactus, and BQTerrace • Class C (832 × 480): BasketballDrill, BQMall, PartyScene, and RaceHorsesC • Class E (416 × 240): BasketballPass, BQSquare, BlowingBubbles, RaceHorses • Class E (1280 × 720): Vidyo1, Vidyo3, and Vidyo4
Total Frames to be Coded	• Class A: 5 seconds of video duration • Other Classes: 10 seconds of video duration
Software	• HM 3.0
Quantization Parameter	• 22, 27, 32 and 37
Others	• High efficiency setting

图 A-3　测试环境与软件参考配置

(a)	HM 3.0			Proposed Method			Comparison		
Class	**Bitrate (kb/s)**	**PSNR (dB)**	**Time (ms)**	**Bitrate (kb/s)**	**PSNR (dB)**	**Time (ms)**	**Bitrate (%)**	**PSNR (dB)**	**Time (%)**
Class B Average	10,587	36.74	697.29	10,553	36.72	450.76	−0.34	−0.02	−38
Class C Average	3,803	35.04	141.45	3,794	35.02	104.81	−0.33	−0.02	−29
Class D Average	1,031	34.55	34.58	1,027	34.52	25.86	−0.44	−0.03	−28
Class E Average	1,862	40.19	246.60	1,847	40.16	104.67	−0.78	−0.03	−58
		Average					−0.44	−0.03	−37

(b)	HM 3.0			Proposed Method			Comparison		
Class A Average	14,423	37.11	1,116.5	14,363	37.05	690.30	−0.61	−0.06	−41
Class B Average	9,620	36.74	548.42	9,564	36.70	300.70	−0.55	−0.04	−47
Class C Average	3,533	35.09	112.66	3,523	35.04	74.15	−0.51	−0.05	−37
Class D Average	942	34.78	27.46	940	34.74	18.51	−0.44	−0.04	−36
		Average					−0.52	−0.05	−41

图 A-4　文中所提出的算法与 HM3.0 在编码时间上的对比：(a) 低延迟方案；(b) 随机访问方案

A-4 展示了在低延迟方案与随机访问方案两种环境下，我们评测了我们的方法与 HM 3.0 两种方法的总执行时间。尽管我们测试了每一种 A-3 中所列举的不同的量化参数下的结果，但 A-4 中，我们只展示每一类的平均结果。A-4 显示了我们的方法相较于 HM 3.0,，在低延迟方案的环境下减少了 63.34% 的时间，而

在随机访问方案的环境下，则减少了 59.43 % 的时间。以图像质量的轻微退化为代价所获得的编码增益，是所提出的方法的自然结果，因为编码树修剪过程会使图像质量上的比特减少。通过调整 Threshold 的值，我们可以节约更多的编码时间，但其代价是图片质量的进一步下降。目前，我们所提出的方法，在模式设置为 SKIP 时，能得到编码时间与图片质量相平衡的最好结果。

A.1.4 结论

在本文中，我们为 HEVC 提出了一种快速的 CU 决策方法，这种方法基于编码树修剪从而更早确定 CU 的大小，与 HM 3.0 编码器相比，仅以可以忽略不计的 BD-比特率损失为代价，就可以使编码时间减少约 40 %。实验结果表明，当你想设计一个快速的 HEVC 编码器时，我们所提出的编码树剪枝方法应当被考虑。在 JCT-VC 第六次会议上，HEVC 测试模型 4.0 编码器采用了我们所提出的方法。

References

[1] Choi, Kiho, Sang-Hyo Park, and Euee S. Jang. "Coding tree pruning based CU early termination." JCTVC-F092, Torino, Italy (2011): 14-22.

A.2 一种有效的 H.264 上的可变块大小早终止算法

一种有效的 H.264 上的可变块大小早终止算法

摘要： H.264 视频编码标准比以前的标准提供了更高的编码效率，但同时，其复杂度也显著提高。在 H.264 编码器中，最耗时的组成部分是可变块大小的运动估计。为了降低运动估计的复杂度，本文提出了一种早终止算法。它通过检查仅仅一个搜索点来预测最佳的运动矢量。使用该方法，一些运动搜索可以被提前停止，由此，大量的搜索点可以被跳过。本文所提出的方法适用于任何快速运动估计算法。实验采用的是 H.264 所采用的一种快速运动估计算法。结果表明，本算法显著降低了复杂度，而代价仅仅是可以忽略不计的视频质量的降低。

关键词：早终止；H.264；运动估计；可变块大小。

A.2.1 引言

近年来，视频编码技术高速发展。理所当然地，压缩性能的提升伴随着计算代价的增长。作为 ITU-T 的 VCEG 和 ISO/IEC 的 MPEG 所最新制定标准的

H.264，相较于之前的 H.261、H.263、MPEG-1/2/4 等标准，在编码效率上有着巨幅的提升。然而，其过高的计算复杂度使其无法被广泛应用到实时业务中。

通常来说，一个视频编码器最为耗时的部分是运动估计。减少其计算代价的方法有两种。第一种是加速算法本身，对 ME 过程来说，数种快速算法被提出，例如六边形搜索（HBS）、增强型预测分区搜索（EPZS）和混合非对称交叉多六边形网格搜索（UMHexagonS）等。另一种方法就是更早地终止 ME 的计算。通过预测那些 DCT 系数会被量化为 0 的块，一些方法有效地减少了 ME 的计算量。另一方面，一些块的重要部分在 ME 之后具有零运动矢量（MV）。据此，有算法提出的零运动检测算法（ZMD），通过将它们的差值的绝对值求和，并与预先定义的阈值进行比较来检测这样的块，然后跳过其余的搜索点。

然而，上述早终止方法都是针对以前的编码方案（如 H.263）而开发的。它们不能再应用于 H.264。这是因为相比于仅有两中块大小（16x16 和 8x8）的 H.263，H.264 拥有从 16x16 到 4x4 变化的七种块大小。

通过扩展 ZMD 的概念，我们提出了一种用于 H.264 视频编码的可变块大小最佳运动检测（VBBMD）算法。该方法在三个方面与 ZMD 不同：

1. 基于不同块大小的检测精度，分别获得 VBBMD 中使用的阈值，同时考虑了复杂度的降低，使得阈值的决策更加合理；
2. 在 VBBMD 中，除零运动点外，预测的移动到的点也会被检查，从而更多的搜索点可以被跳过；
3. 对于 16x16 的块，VBBMD 使用双阈值。使用较低的阈值跳过较小的块的运动搜索。

我们的方法可以适用于任何运动搜索算法，并且不需要额外的计算。注意，整个讨论集中于帧间帧（P 帧）和整数像素运动搜索。文章的剩余部分如下进行组织。A.2.2 介绍了早期终止算法及其改进。A.2.3 中给出了仿真结果。最后，在 A.2.4 ，我们总结了本文，并给出了未来的方向。

A.2.2 早终止算法

A.2.2.1 可变块大小零运动推断

通常来说，对大多数序列，在 ME 后相当数量的块会有这零运动矢量，正如 A-5 展示的一样。

ZMB RATES (%) OF SEVEN MODES (QP = 32)

Sequence	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
Akiyo	98.03	97.51	97.18	97.28	97.14	96.47	97.04
Salesman	96.80	95.92	95.69	95.15	94.78	94.13	94.33
News	91.90	91.72	91.98	92.35	91.93	91.31	91.78
Silent	85.01	84.65	84.15	85.01	84.75	84.23	85.06
Coastguard	32.44	33.51	32.80	30.91	31.40	30.71	31.06
Foreman	43.97	44.15	43.57	44.25	44.81	43.73	45.42

图 A-5　当量化参数为 32 时，七种模式下零运动块的比例

表中，模式 1-7 代表从 16x16 到 4x4 的七种帧间预测模式，如 A-6 所展示。

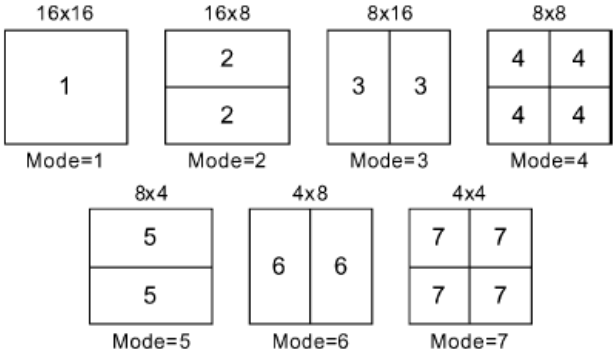


图 A-6　H.264 中的七种预测模式

可以看出，不论是运动较为活跃亦或是不太活跃的视频序列，有 30.71 % 到 98.03% 的块存在零运动。如果我们可以预测零运动块（ZMB），我们可以提前停止 ME 的计算，从而消除计算的一部分。通过扩展 ZMD 的概念，我们开发了一个可变块大小的 ZMD（VBZMD）算法。

在以前的 ZMD 方法中，当块的 SAD 小于给定阈值时，块可以被看作是 ZMB。在 H.264 中选择最佳匹配块时，使用代价函数 J 而不是 SAD 作为预测误差的度量。

$$J(m,\lambda) = SAD(s,c(m)) + \lambda R(m-p) \tag{A-4}$$

这里，$m = (m_x, m_y)^T$ 是当前的 MV，$p = (p_x, p_y)^T$ 是预测的 MV，而 λ 则是拉格朗日乘数，$R(m-p)$ 是编码 MV 所用的比特数。

如果一个块是零运动的，那么 $MV(0,0)$ 处极有可能有一个很小的代价值。因此，我们为七种模式分别定义阈值 $THZ_i(i = 1, ..., 7)$。在整个运动估计过程中，$MV(0,0)$ 首先被检查。如果 $MV(0,0)$ 处的代价满足 A-5 ，我们就认为这个块是一个 ZMB，由此剩余的搜索过程既可以被停止。

$$J_i < THZ_i,\ for\ i = 1, ..., 7. \tag{A-5}$$

H.264 中 ZMD 的关键是如何去决定不同块大小下的阈值。很明显，阈值越大，检测到的 ZMB 越多，更多的搜索点就可以被跳过。然而同时，更多的块被错误地选择，这会导致更大的图像质量损失。因此，在性能和复杂性之间存在一个平衡。在实践中，防止视频质量的损失可能比复杂度的轻微增加更重要。

因此，我们将检测精度作为确定阈值的指导。这里，精度表示当代价小于阈值时，块为 ZMB 的概率。我们的方法是，在实验中基于不同的准确率获得多个候选阈值集，并选择一个最优集合，在实践中，我们需要在质量和复杂度之间取一个较好的折衷。许多序列的实验表明，在相同的阈值下，低运动场景中的准确率和检测率都高于有较多运动的场景中的准确率和检测率。由于 Foreman 序列可以代表具有较大运动的场景，所以我们可以选择基于 Fordman 的阈值，并将它们应用到其他序列。A-7 显示了我们获得的候选阈值。

CANDIDATE THRESHOLDS FOR DIFFERENT ACCURACY RATES OF FOREMAN (QP = 32)

Accuracy	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7
80%	600	550	570	400	315	300	250
75%	960	650	660	480	360	350	270
70%	1650	920	890	600	410	400	300
65%	2270	1200	1160	700	470	460	350
60%	2800	1500	1430	880	530	530	400

图 A-7　对于 Foreman 序列，不同准确率的备选阈值（QP=32）

在我们的方法中，如果块的零运动代价小于相应的阈值，则我们将其视为块的最佳 MV 并跳到同一宏块中的下一个块。如果当前块是宏块中的最后一个块，则跳转到下一个模式。在所有模式被检查之后，拥有最小成本的模式被选择，并应用于后续的任一为 H.264 设计的 ME 算法。在模式 1 中，由于一个宏块只包含一个块，如果零运动成本足够小，那么不仅可能是最好的 MV，而且模式 1 也很可能是宏块的最佳模式。因此，我们为模式 1 定义了额外的低阈值 THS。如果零运动代价小于 THS，则整个宏块的运动搜索停止。VBZMD 的整个过程概述如下：

```
Assume the current macroblock is MB(x, y)
For each mode i (i = 1, ..., 7) of the MB(x, y)
  For each block in the macroblock
    Calculate the cost at (0, 0)
    If cost < THZ_i
      Set MV = (0, 0)
      If mode = 1 and cost < THS
        Set best mode = 1
        Exit two loops
      End if
      Exit one loop
    Else
      Do the normal motion search process
    End if
  End for
End for
If best mode is not set
  Set best mode to be the mode with the minimal cost
```

图 A-8　VBZMD 整个过程

我们所使用的 H.264 编码器是 6.1e 版本的 JVT 软件。我们将所提出的方法与 H.264 标准所采用的快速 ME 算法 UMHexagonS 相结合。与完全搜索相比，快速 ME 方法可以将整数像素 ME 的复杂度降低 90%，而通过使用所提出的方法可以进一步减少计算。因为 800 的阈值对应于模式 1 中相对较高的 78% 的准确率，并且它在实验中似乎工作得很好，所以我们选择它作为 THS。正如 A-9 所展示的，对于代表低运动场景的 Akiyo 序列，每个宏块（PPMB）的搜索点中，有高达 93.47% 的部分被减去，而其代价仅为平均峰值信号（PSNR）降级不超过 0.05 分贝，且比特率的增加是微不足道的。即使对于具有较大的面部运动和摄像机平移的 Foreman 序列，计算量的节省也是相当明显的，且 PSNR 丢失依旧轻微。这里，搜索点的数目是在宏块级别上计算的。例如，模式 4 中，大小为 8Œ8 的块中的一个搜索点，等同于 1/4 PPMB。这里，对应于 65% 的准确率的阈值集在性能和复杂度之间提供了良好的折衷。Akiyo 和 Foreman 序列的 PSNR 退化分别为 0.04 和 0.13，而计算量的节省分别为 91.9% 和 59.64%。

PERFORMANCE OF VBZMD WITH CANDIDATE THRESHOLDS (QP = 32)

Sequence	Method	Accuracy	PSNR (dB)	PPMB	Bit-rate (kbps)
Akiyo	Fast ME		35.22	63.86	14.52
	Fast ME +VBZMD	80%	35.18	29.02	14.65
		75%	35.20	21.41	14.67
		70%	35.19	7.96	14.70
		65%	35.18	5.17	14.59
		60%	35.17	4.17	14.67
Foreman	Fast ME		32.90	160.48	79.34
	Fast ME +VBZMD	80%	32.87	124.97	79.00
		75%	32.86	112.85	79.55
		70%	32.82	88.75	80.49
		65%	32.77	64.77	82.19
		60%	32.75	49.39	84.73

图 A-9　参考阈值下 VBZMD 的表现

直观地，高精度相对应的 PSNR 应该大于与低精度相对应的 PSNR。然而，在这个表中可以看出，Akiyo 序列中，精度为 75% 对应的的 PSNR 值和 70% 对应的 PSNR 值，比精度 80% 对应的 PSNR 高。这种情况是可能的，因为较大的阈值不仅减少了更多的计算，而且减少了编码 MV 所需的比特，然后节省的比特提高了总体 PSNR。

A.2.2.2　可变块大小最佳运动推断

在减少计算量上，尽管 VBZMD 已经取得了相当好的表现，但我们仍可以将其改进以得到更好的表现。在 H.264 的中, 当前块的左、上和右上（或上左）相邻块的 MV 将用作预测的当前块的 MV 等，如 A-10 所说明。由于相邻块的 MV 通常是相关的，预测向量很可能正是在 ME 之后最好的 MV。为方便描述，我们定义最佳 MV 为预测向量的块为预测向量块（PVB）。正如 A-11 所示，不同序

列的 PVB 率序列是大于相应的、如 A-5 所示的 ZMB 率得。特别是，对于运动较为激烈的场景，如 Coastguard 和 Foreman 序列，PVB 率较 ZMB 率高 23% 到 122%。另一方面，另一个实验显示，对大多数序列来说，PVB 率较 ZMB 率高 97%。我们所观察到的现象启示我们，应当使用预测向量来预测最佳 MV，而不是零 MV。此外，在 H.264 中，进行编码和传输是当前 MV 和预测矢量之间的差异，而不是当前的 MV 本身。因此，使用预测向量来预测最佳 MV 可以节省用于编码 MV 的比特。

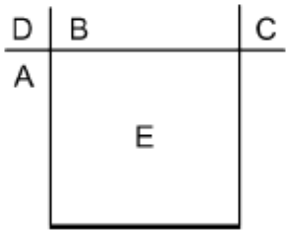


图 A-10　当前块与其相邻块

Sequence	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
Akiyo	98.05	97.51	97.23	97.37	97.27	96.61	97.24
Salesman	96.84	95.71	95.46	95.40	95.11	94.54	95.03
News	91.80	91.70	91.66	92.83	92.71	92.22	93.16
Silent	85.09	85.04	84.57	86.32	86.80	86.36	88.28
Coastguard	58.68	62.19	62.41	64.32	66.17	67.09	69.04
Foreman	54.19	56.26	55.46	57.79	60.00	59.15	63.15

图 A-11　预测向量块比例

基于前面的 VBZMD，我们提出了一种 VBBMD 算法。关键思想是将预测向量的成本与给定阈值进行比较。如果成本小于阈值，则将预测向量视为最佳 MV，然后可以跳过剩余的搜索点。

同样，如何选择不同块大小情况下的阈值仍然是问题的关键。使用与

VBZMD 中相同的方法，我们可以在不同的准确率下获得多个候选阈值集。VBBMD 的步骤与 VBZMD 几乎相同，唯一的区别在于，我们检查预测矢量而不是零 MV。我们用候选阈值集编码不同的视频序列，并选择最佳集合。结果表明，阈值的集合（2500, 1500, 1400，920, 625, 570，500）在准确度为 66% 时提供了一个很好的折衷。此外，模式 1 的低阈值 THS 也是 800，因为它工作得很好。A-12 展示出了具有最佳阈值集的 VBBMD 结果。与 VBZMD 在同一表中的结果相比，我们可以看到，更多的搜索点被进一步减少，而 PSNR 和比特率几乎一样。这验证了 VBBMD 比 VBZMD 更有效。

Sequence	Method	PSNR (dB)	PPMB	Bit-rate (kbps)
Akiyo	Fast ME	35.22	63.86	14.52
	Fast ME+VBZMD	35.18	5.17	14.59
	Fast ME+VBBMD	35.18	4.17	14.59
Foreman	Fast ME	32.90	160.48	79.34
	Fast ME+VBZMD	32.77	64.77	82.19
	Fast ME+VBBMD	32.77	30.12	82.34

图 A-12　最优阈值下，VBBMD 的表现（QP=32）

如前所述，阈值是在 $QP = 32$ 的条件下确定的。然而实验表明，不同的 QP 值将得到不同的阈值。这是因为代价值 J 与 QP 有关。在 A-4 中，J 的值与 λ 直接相关，而 λ 的值是从一个以 QP 为索引的表中查询得到的。当一个块被视为 PVB 时，当前 MV m 与预测 MV p 相差最多为 3/4 像素，这可以用 10 比特来编码。因此，我们近似地设置 $R(m - p)$ 为 10。我们定义 $QP = 32$ 条件下的阈值为标准阈值，对于其他的 QP 取值，相应的阈值应设置为标准阈值加上 $(\lambda_{QP} - \lambda_{32})x10$。

A.2.3　模拟结果

我们将 VBBMD 实现在了 H.264 参考编码器上，并使用 UMHexagonS 快速 ME 算法。我们设置运动搜多范围为 32，并设置参考帧数为 1。率失真优化被禁

止并且 CAVLC 熵编码开启。

为了验证所提出的方法在不同实验条件下的有效性，我们选择了六个序列，其运动活动从小到大变化。他们是 Akiyo, Salesman, News, Silent, Coastguard 和 Foreman 序列。所有序列均为 QCIF 格式，并以 30 帧每秒编码。除第一帧以外的所有帧都被编码为 P 帧。此外，为了检查在不同比特率下的性能，在我们的实验中使用了三个 QP 值：28, 32, 36。如 A.2.2.2 所述，在 VBBMD 中使用的阈值根据 QP 进行调整。

为便于比较，我们在 A-13 中显示了平均 PSNR 增益（$\Delta PSNR$）、PPMB 平均降低率（$\Delta PPMB$）和比特率的增长率（$\Delta Bit-rate$）。从结果可以看出，所提出的方法是稳定的，并且在不同的条件下工作得很好。与仅使用快速 ME 算法相比，我们的方法进一步减少了 76.10% 到 94.12% 的整数像素搜索点，而 PSNR 衰减平均仅为 0.06 分贝，并且比特率只是略微增加。

QP	Performance	Akiyo	Sales-man	News	Silent	Coast-guard	Fore-man
28	ΔPSNR (dB)	-0.05	-0.02	-0.06	-0.04	-0.05	-0.12
	ΔPPMB (%)	94.12	90.22	83.30	81.95	87.67	81.76
	ΔBit-rate (%)	0.04	1.60	1.33	2.42	3.72	4.60
32	ΔPSNR (dB)	-0.03	-0.02	-0.06	-0.04	-0.06	-0.13
	ΔPPMB (%)	93.39	91.07	83.48	83.46	86.37	80.61
	ΔBit-rate (%)	1.03	0.80	1.26	2.95	4.63	3.43
36	ΔPSNR (dB)	-0.05	-0.06	-0.08	-0.04	-0.06	-0.13
	ΔPPMB (%)	92.63	90.43	82.17	83.72	77.46	76.10
	ΔBit-rate (%)	0.66	0.37	1.41	-0.07	3.00	3.45

图 A-13　在不同条件下 VBBMD 的表现

A.2.4　结论

本文提出了一种 H.264 视频编码的早终止方法 VBBMD，该方法通过进一步减少计算总量来帮助现有的快速运动搜索算法。该方法通过检查一个搜索点来预测最佳的 MV。通过这种方法，一些运动搜索可以被提前停止，因此与这些

搜索相关联的计算可以被减少。在该方法中，考虑了复杂度的降低的同时，阈值通过基于不同块大小的检测精度来确定，这使得阈值决策更加合理。实验结果表明，我们提出的方法实现了显著的复杂度降低，而视频质量的退化是可以忽略不计的。未来的方向可能包括使阈值根据量化级别和运动级别自适应调整，并将早终止方法应用于子像素的运动估计。

References

[1] Yang, Libo, et al. "An effective variable block-size early termination algorithm for H. 264 video coding." IEEE Transactions on Circuits and Systems for Video Technology 15.6 (2005): 784-788.