



## 交流图片




图像和视频编码课程

2014, Pages 291-316

## 第9章-基于块的混合视频编解码器

大卫·R·布尔

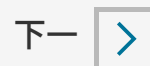
显示更多 ∨

 大纲 |  共享  引用<https://doi.org/10.1016/B978-0-12-405906-1.00009-X>

获取权利和内容

## 摘要

本章介绍如何将转换、[量化](#)、运动估计和熵编码集成到一个完整的编码架构中。讨论了基本混合架构的局限性，并描述了最近标准（如H.264/AVC和HEVC）中引入的实用编解码器增强功能。这些技术有助于显著提高基本视频编码结构的性能。本章描述了内部预测——一种通过对每个帧中块的空间预测来提高性能的方法。然后，它通过子像素估计和使用多个参考帧来扩展传统的运动预测方法。它还考察了可变块大小及其在编码过程中在变换和运动估计方面的影响。最后，描述了环内解块作为减少边缘[工件](#)方法的重要领域。通篇提供了示例。



## 关键词

视频压缩; 位移帧差 (DFD); 帧内; 子像素运动估计; 插值; 多个参考帧; 整数变换; 解块滤波器

在[第8章](#)中，我们看到了运动估计是如何成为高效视频编码系统的基础的。本章介绍如何将基于混合结构，结合基于块的运动预测和基于块的变换编码，将其集成到一个实用的视频压缩框架

中。介绍了通用视频编码器的架构，并详细介绍了编解码阶段的操作。

讨论了基本混合架构的局限性，并描述了最近引入的一系列实用编解码器增强功能。这些技术有助于显著提高基本视频编码结构的性能，并使基本混合架构在过去40年左右每8-10年提供一倍的编码增益。描述的技术不是现代编解码器中所有功能的详尽列表，而是一些最重要的功能。

我们首先关注内部预测——一种通过对每个帧中块的空间预测来提高性能的手段。然后，我们通过亚像素估计和使用多个参考帧来扩展第8章中介绍的运动预测方法。我们还从变换和运动估计的角度研究了可变块大小的影响及其在编码过程中的利用。最后，我们研究了环内解块作为减少边缘伪影的方法的重要领域。

## 9.1. 视频压缩的基于块的混合模型

### 9.1.1. 图片类型和预测模式

#### 预测模式

正如我们在第八章中所看到的，在视频压缩中使用了三种主要类型的预测：

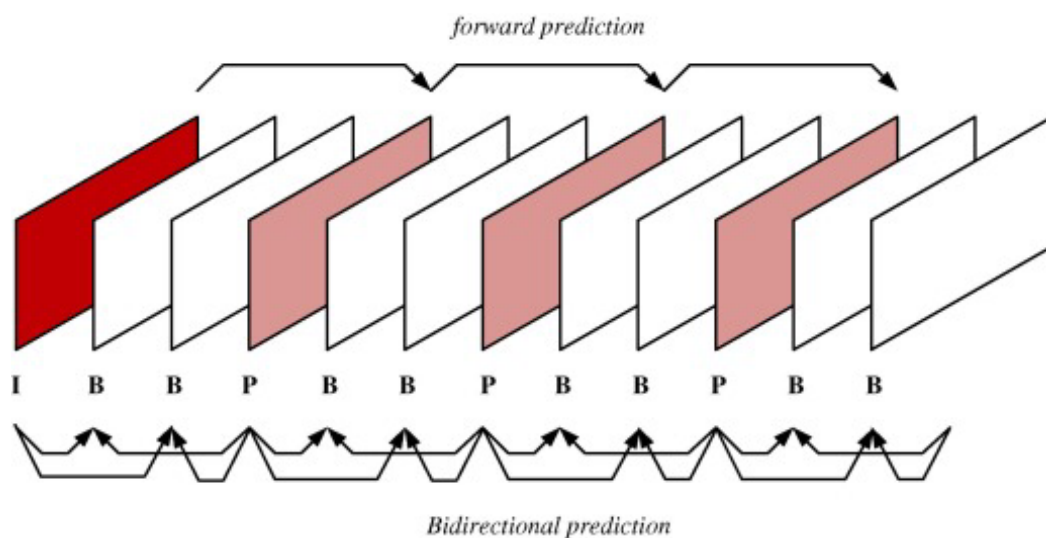
- **前瞻预测**：参考图片在当前图片之前临时出现。
- **反向预测**：参考图片在当前图片之后临时出现。
- **双向预测**：采用两张（或更多）参考图片（向前和向后），并以某种方式组合候选预测，形成最终预测。

#### 图片类型

大多数视频编解码器都使用了三种主要类型的图片（或帧）：

- **I-图片**：这些是内部编码的（编码不参考任何其他图片）。
- **P-图片**：这些与另一张 I 或 P 图片的前向（或后向）预测相互编码。
- **B-图片**：这些图片与来自多个 I-和/或 P-图片的双向预测相互编码。

编码图片按一组图片（GOP）的顺序排列。图9.1显示了由12帧组成的典型共和党结构。共和党将包含一张I-picture和0个或0个P-和B-picture。图9.1中的12帧GOP有时被称为IBBBBBBBBBBBB结构，由于因果关系，很明显，编码顺序与图中所示不同，因为P-图片必须在前面的B图片之前编码。



[下载: 下载全尺寸图像](#)

图9.1。典型的一组图片结构。

### 9.1.2。DFD信号的性质

DCT是卡尔胡宁-洛夫展开式的特殊情况，适用于自相关系数接近单位的静止和一阶马尔可夫过程。大多数静止图像具有良好的相关特性，通常与 $\rho \geq 0.9$ ；这证明使用DCT进行帧内编码是合理的，因为它将提供近乎最佳的装饰关系。然而，在帧间编码中，通过DPCM环路中的运动估计预测运动，DFD信号的自相关系数通常降至0.3-0.5。斯特罗巴赫[1]表明，在许多实际情况下，DFD信号的DCT编码增益与帧内情况相比很小，大部分增益与时间装饰有关。由于运动故障，DFD信号中引入的边缘也可能在量化后引起响铃伪影。

这种讨论意味着混合编解码器使用装饰变换（如DCT）来编码DFD残差，并不总是最优的。然而，在实践中，如果运动模型更准确，并且任何人工边缘被过滤掉，DFD可以变得更平滑（即显示更高的自相关值）。这些特性通过编解码器增强得到促进，如子像素运动估计、可变块大小、多个参考帧以及在运动预测循环中使用解块滤波器。这种增强，加上高效的速率-失真优化方法，在很大程度上决定了基于块的混合编解码器的性能和普遍接受。都在本章后面更详细地讨论它们。

### 9.1.3。视频编码循环的操作

根据第一章对视频编码架构的简要概述，我们在这里更正式、更详细地描述了编码循环的操作。图9.2给出了帧内编码的通用结构，图9.3给出了帧间模式的通用结构。编码器在模内操作是算法9.1模内编码器操作

1. Inter/intra switch is in the intra position;
2. Compute displaced frame difference (DFD) signal (equal to the video input frame in this case):  $E_k = S_k$ ;
3. Perform a forward decorrelating transform (typically a DCT or variant) on the input frame and quantize according to the prevailing rate–distortion criteria:  $C_k = Q(DCT(E_k))$ ;
4. Entropy code the transformed frame and transmit to the channel;
5. Inverse quantize  $C_k$  and perform an inverse DCT to produce the same decoded frame pixel values as at the decoder:  $E'_k = DCT^{-1}(Q^{-1}(C_k))$ ;
6. Update the reference memory with the reconstructed frame:  $S'_k = E'_k + 0$ .

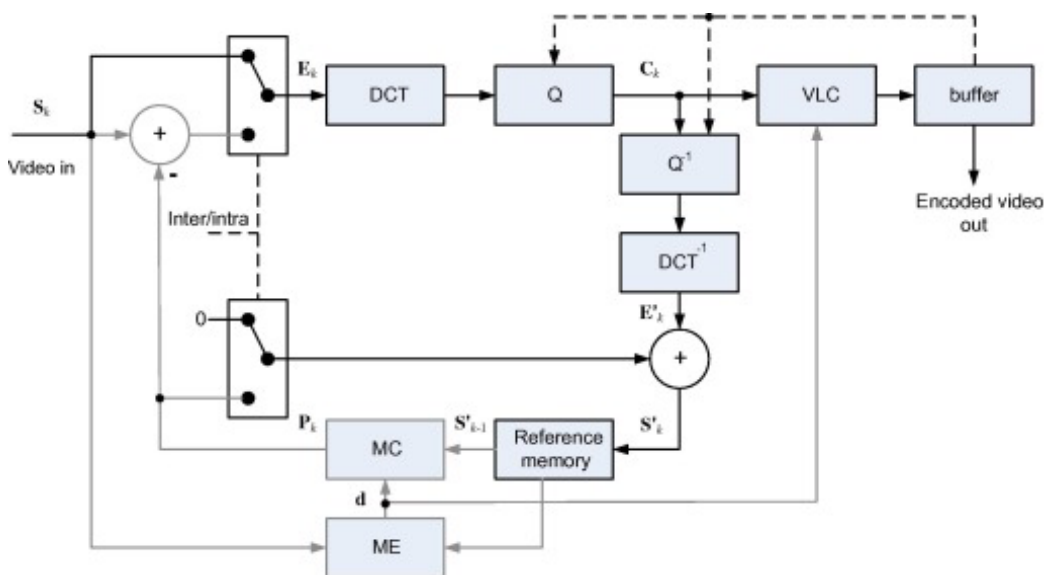
[下载：下载全尺寸图像](#)

## 算法9.2模间编码器操作

1. Inter/intra switch is in the inter position;
2. Estimate motion vector for current frame:  $d = ME(S_k, S_{k-1})$ ;
3. Form the motion compensated prediction frame,  $P_k$ :  $P_k = S'_{k-1}[p + d]$ ;
4. Compute the DFD signal:  $E_k = S_k - P_k$ ;
5. Perform a forward decorrelating transform on the DFD and quantize according to the prevailing rate–distortion criteria:  $C_k = Q(DCT(E_k))$ ;
6. Entropy code the transformed DFD, motion vectors and control parameters, and transmit to the channel;
7. Inverse quantize  $C_k$  and perform an inverse DCT to produce the same decoded frame pixel values as at the decoder:  $E'_k = DCT^{-1}(Q^{-1}(C_k))$ ;
8. Update the reference memory with the reconstructed frame:  $S'_k = E'_k + P_k$ .

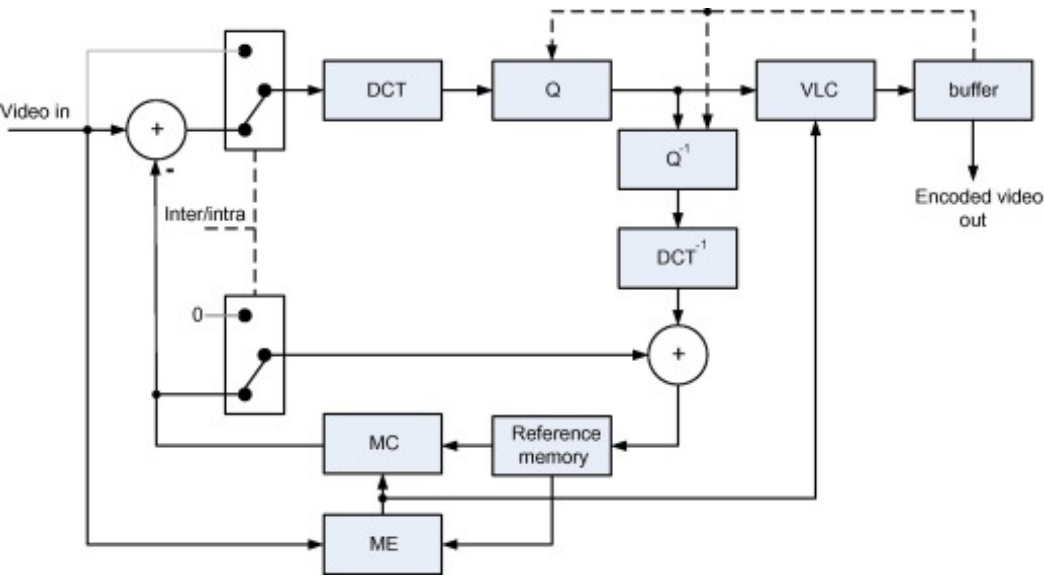
[下载：下载全尺寸图像](#)

在算法9.1中描述。同样，算法9.2描述了模式间操作。



[下载：下载全尺寸图像](#)

图9.2。视频编码器结构（帧内模式）。虚线描绘了与量化器步长等相关的控制信号。



下载： [下载全尺寸图像](#)

图9.3。视频编码器结构（帧间模式）。

9.1.4。视频解码器的操作

视频解码器的操作如图9.4所示，算法9.3、算法9.4中进行了正式描述。通过比较编码器和解码器架构，我们可以看出编码器在其预测反馈循环中包含解码器的完整副本。这确保了（在没有信道错误的情况下）编码器和解码器操作之间没有漂移。

示例9.1基本视频编码器的操作

基于算法9.1中的编码器图，图9.3以及k时间的以下数据，计算：

1. 当前整数像素运动矢量d，时间k。假设搜索窗口是整个参考框架，并且矢量仅限于指向框架内。

算法9.3 模内解码器操作

1. Inter/intra switch is in the intra position;
2. Perform entropy decoding of control parameters and quantized DFD coefficients;
3. Inverse quantize  $C_k$  and perform an inverse DCT to produce the decoded frame pixel values:  $E'_k = DCT^{-1}(Q^{-1}(C_k))$ ;
4. Update the reference memory with the reconstructed frame and output to file or display:  $S'_k = E'_k + 0$ .

下载： [下载全尺寸图像](#)

## 算法9.4 帧间解码器操作

1. Inter/intra switch is in the inter position;
2. Perform entropy decoding of control parameters, quantized DFD coefficients and motion vector;
3. Inverse quantize  $\mathbf{C}_k$  and perform an inverse DCT to produce the decoded DFD pixel values:  $\mathbf{E}'_k = \text{DCT}^{-1}(\mathbf{Q}^{-1}(\mathbf{C}_k))$ ;
4. Form the motion compensated prediction frame,  $\mathbf{P}_k$ :  $\mathbf{P}_k = \mathbf{S}'_{k-1}[\mathbf{p} + \mathbf{d}]$ ;
5. Update the reference memory with the reconstructed frame and output to file or display:  $\mathbf{S}'_k = \mathbf{E}'_k + \mathbf{P}_k$ .

[下载：下载全尺寸图像](#)

2. 运动补偿输出， $\mathbf{P}_k$ ，在 $k$ 时间。
3. 当前输入帧的DFD， $\mathbf{E}_k$ 。
4. 转化和量化的DFD输出， $\mathbf{C}_k$ 。

假设图像帧大小为 $12 \times 12$  pixels和**大块**大小 $4 \times 4$  pixels。编解码器以帧间模式运行，使用基于块的平移运动估计，块大小为 $4 \times 4$  pixels冒号：

变换矩阵：

$$\mathbf{A} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

量化矩阵：

$$\mathbf{Q} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 4 & 4 \\ 2 & 2 & 2 & 4 \end{bmatrix}$$

当前输入块：

$$\mathbf{S}_k = \begin{bmatrix} 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{bmatrix}; \text{假设框架的左上角}$$

参考内存：

$$\mathbf{S}'_{k-1} = \begin{bmatrix} 0 & 0 & 5 & 6 & 2 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 8 & 8 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 6 & 6 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 8 & 12 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 4 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 5 & 8 & 0 \\ 7 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 6 & 7 & 0 & 0 \\ 0 & 0 & 0 & 10 & 3 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## 解决方案

1. 当前运动矢量：

通过检查：

$$\mathbf{d}=[2,0]$$

2. 运动补偿输出：

$$\mathbf{P}_k = \begin{bmatrix} 5 & 6 & 2 & 6 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 12 & 8 \end{bmatrix}$$

3. 当前输入帧的DFD：

$$\mathbf{E}_k = \mathbf{S}_k - \mathbf{P}_k = \begin{bmatrix} 1 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 \end{bmatrix}$$

4. 转换和量化输出：

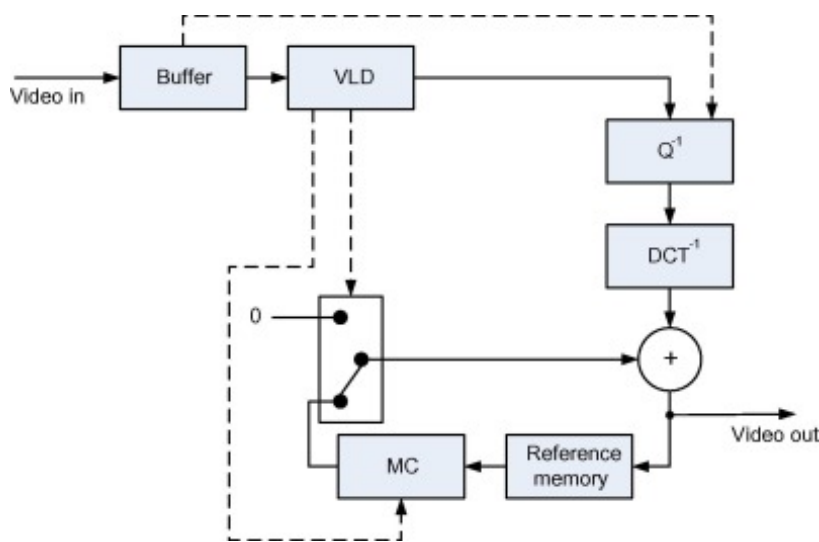
转换后的输出是：

$$\mathbf{C}_k = \mathbf{A}\mathbf{E}_k\mathbf{A}^T = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \\ 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \end{bmatrix}$$

量化的输出是：

$$\mathbf{C}_{k(Q)} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \\ 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \end{bmatrix} / \mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$





下载: 下载全尺寸图像

图9.4。视频解码器结构。

## 9.2。帧内预测

在第8章中，我们将时间预测作为转换前帧之间装饰的一种手段的益处。H.264/AVC和HEVC还支持空间域的内部预测，以及允许直接编码像素值的I-PCM模式。在具有某些定向纹理或定向边缘的区域，发现内部编码特别有益。H.264/AVC支持两种主要预测模式——**intra\_4 × 4**和**intra\_16 × 16**。这些描述如下。

### 9.2.1.小亮度块的内预测

当本地细节水平很高时，特别是边缘等定向功能时，这种模式表现良好。给定块的编码（4 × 4在H.264/AVC中，是指位于当前块左侧和上方的先前编码块的样本子集执行的。正如我们将在第11章中所看到的，当基于以前编码宏块预测的数据时，内部编码可能导致帧之间的错误传播。为了避免这种情况，可以使用一种约束的编码内模式，将内预测限制为基于编码内邻居。

例如，图9.5显示了H.264/AVC中使用的编码内部关系和方向。对每个4 × 4 block (pixels *a-p* here) is based on its **neighboring pixels** (A–M). In H.264/AVC, the predictor can select from nine prediction orientations plus a DC mode where the average of the neighboring pixels is used as the prediction. These nine modes are shown in Figure 9.6. Some examples of how the predictions are computed are given below:

模式0（垂直）

$$\{a, e, i, m\} = A \quad (9.1)$$

模式1（水平）

$$\{a, b, c, d\} = I \quad (9.2)$$



模式2 (DC)

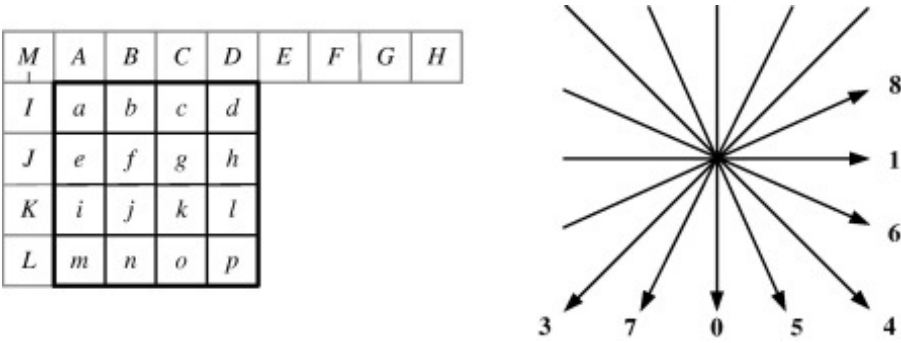
$$\{a,b,\cdots,p\}=\left\lfloor\left(\frac{A+B+C+D+I+J+K+L}{8}\right)+0.5\right\rfloor$$

(9.3)

模式4 (右下)

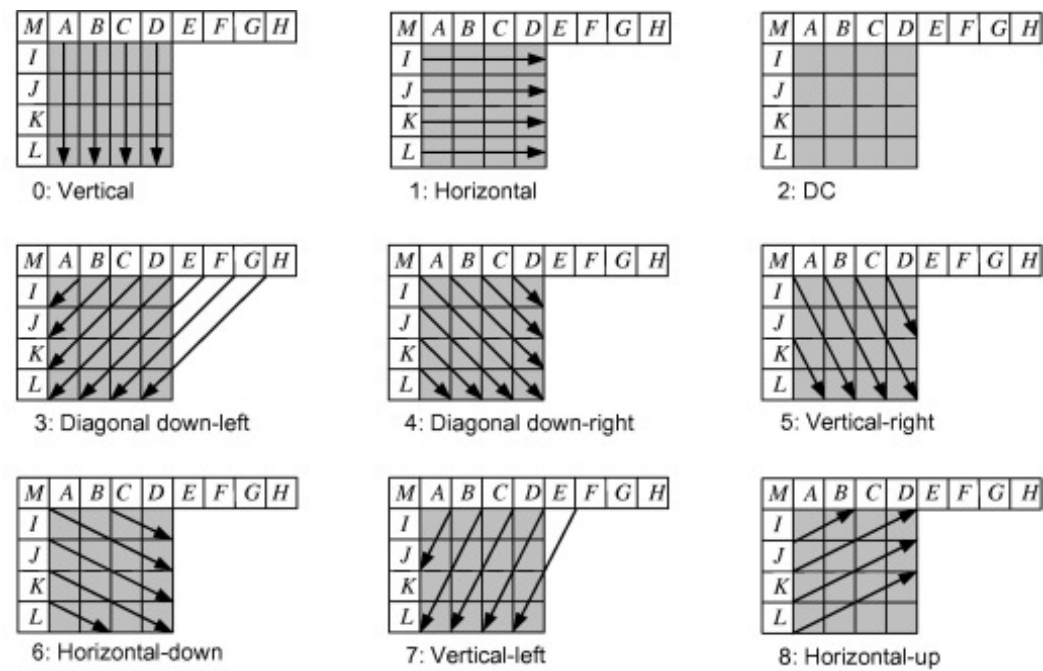
$$\{a,f,k,p\}=\left\lfloor\left(\frac{A+2M+I}{4}\right)+0.5\right\rfloor$$

(9.4)



[下载：下载全尺寸图像](#)

图9.5。H.264/AVC中的预测内模式。



[下载：下载全尺寸图像](#)

图9.6。4 × 4预测模式。

其他模式是由类似的外推方法形成的，完整的描述可以在参考文献中找到。[\[2\]](#)。

9.2.2。对较大块的内预测

对于更大的更平滑的图像区域，对较大的块进行内部预测可能是有益的。在H.264中 **intra\_16 × 16**模式，整个16 × 16同时预测亮度块。该模式仅支持四种预测类型：模式0（垂直）、模式1（水平）、模式2（DC）和模式3（平面）。前三种模式与4 × 4上面描述的模式，除了它们引用了当前块上方和左侧的16个像素，而不仅仅是四个像素。有关平面模式的详细信息，请参阅参考文献。[2]。

**色度样本**的预测方式与亮度块的预测方式相似。由于**色度信号**通常比亮度信号更平滑，因此倾向于使用更大的块大小。

示例9.2预测内部

使用**intra\_4 × 4**垂直、水平和直流模式，用于4 × 4如下所示像素块：

1	2	3	4	5	6	7	8	9
1	2	3	4	5				
1	2	2	4	6				
2	2	3	4	3				
2	2	3	4	5				

[下载：下载全尺寸图像](#)

解决方案

基于垂直、水平和直流模式的预测块如下图所示：

2	3	4	5
2	3	4	5
2	3	4	5
2	3	4	5

vertical

1	1	1	1
1	1	1	1
2	2	2	2
2	2	2	2

horizontal

3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3

DC

[下载：下载全尺寸图像](#)

相应的SAD值是：SAD（垂直）=4；SAD（水平）=30；SAD（DC）=16。因此，基于这个有限的子集，预测**剩余能量**的最佳匹配是模式0（垂直）。在这种情况下，编码前的剩余部分如下所示：

0	0	0	0
0	-1	0	1
0	0	0	-2
0	0	0	0

[下载](#): [下载全尺寸图像](#)

## 9.3。亚像素运动估计

### 9.3.1.子像素匹配

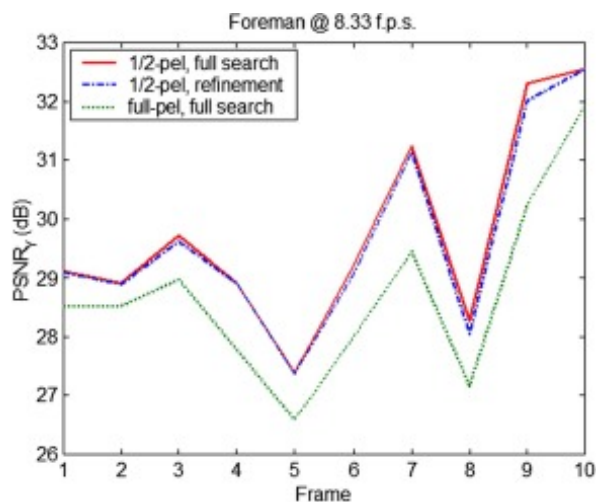
[第八章](#)介绍了运动估计的概念和实现，作为高效视频编码的基础。我们将搜索和匹配的分辨率限制在用于获取图像的采样网格的分辨率上。实际上，场景中的运动与采样网格无关，最准确的匹配可能会发生亚像素位移。爱立信[\[3\]](#)于1985年引入了子像素估计的概念，证明对于所使用的序列，当从全像素精度更改为1/8像素精度时，可以实现高达2分贝的编码增益。Girod[\[4\]](#)正式确定了这一点，他表明，通过将估计精度降低到1/8像素分辨率以上，可以获得有限的收益。

显然，子像素估计的好处将取决于许多因素，包括视频格式（空间和时间分辨率）和内容类型。然而，人们承认，亚像素估计提供了显著的性能增益，现代编码标准都包括以分数像素精度预测运动的选项。MPEG-1是第一个使用半像素估计的国际标准，紧随其后的是1994年的MPEG-2。H.263在1996年引入了更灵活的块大小和四分之一像素估计，其性能的提高在很大程度上归功于这一创新[\[5\]](#)。

值得花几分钟讨论支持子像素运动估计所需的插值。我们可以采取一些方法：

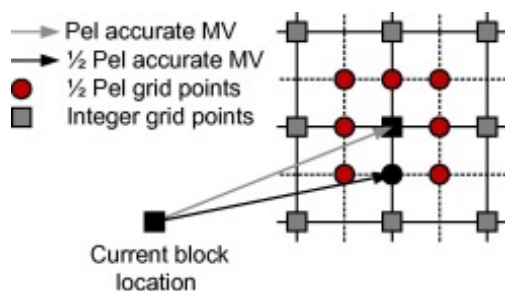
1. 插值当前所有帧块和所有搜索窗口，并以子像素分辨率执行整个运动估计过程。
2. 仅插入搜索窗口，然后按（1）条进行。
3. 首先执行整数像素搜索，然后使用插值搜索窗口和插值当前块在本地优化此搜索。
4. 首先执行整数搜索，然后在本地优化此搜索窗口。

在实践中，采用了方法（4），因为它提供了最低的复杂性，性能只有轻微的下降（见[图9.7](#)）。[图9.8](#)说明了这种方法，图中显示了黑色正方形的最佳全像素解决方案。然后以半像素分辨率对网格进行本地插值，并进行精细搜索，以获得最佳的半像素解决方案（黑圈）。细化搜索通常完全执行，与整数像素大小写相同的匹配标准。



下载: 下载全尺寸图像

图9.7。带有局部精细的子像素搜索。



下载: 下载全尺寸图像

图9.8。半像素精细化的运动估计搜索网格。

### 9.3.2。插值方法

图9.9说明了H.264/AVC中用于半像素和四分之一像素插值的插值方法。在半像素估计的情况下，搜索窗口围绕最佳整数像素结果进行本地插值。如果我们假设帧搜索窗口中的像素在本地上采样了2倍（有关上采样的更详细讨论，请参阅第6章），那么上采样结果为 $x$ 方向由以下方式给出：

$$s_u[x] = \begin{cases} s[x/2]; & x = 2m \\ 0; & \text{otherwise} \end{cases} \quad (9.5)$$

在哪里 $m$ 是一个整数。然后，我们可以过滤结果来插值半像素位置。通常使用的过滤器是对称和可分离的，以减少复杂性。在H.264/AVC[6]的情况下，用于创建插值的半像素值的过滤器， $g[x]$ ，如下：

$$g[x]=\sum_{i=-5}^{+5}h[i]s_u[x-i] \tag{9.6}$$

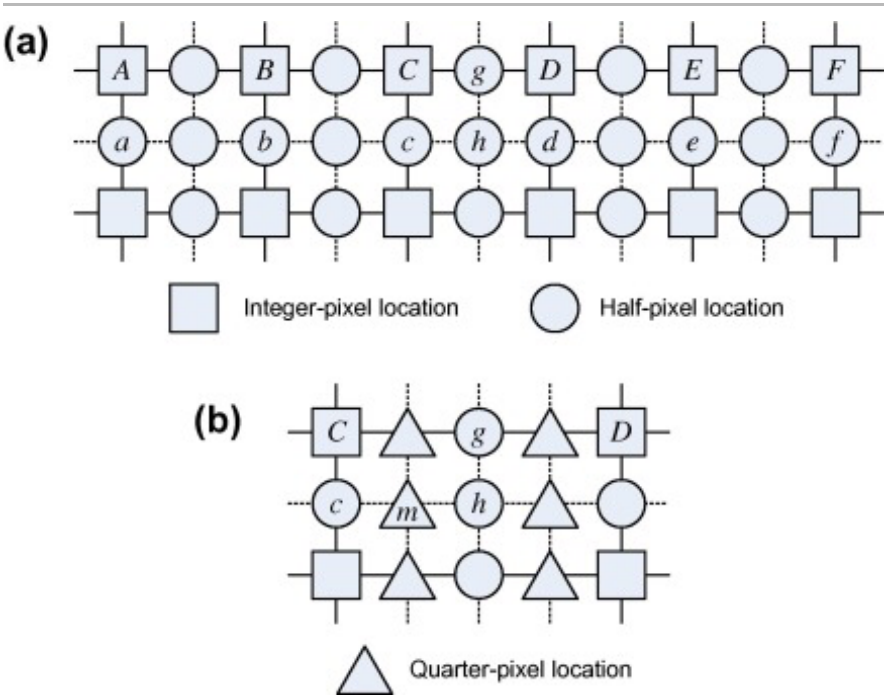
其中滤波系数为：

$$h[i]=\{1,0,-5,0,20,0,20,0,-5,0,1\}/32 \tag{9.7}$$

简单地说，这意味着，参照图9.9中的标签，：

$$g=\frac{(A-5B+20C+20D-5E+F)}{32} \tag{9.8}$$

类似的方法也用于垂直和中间像素，例如*h*，基于像素{*a,b,c,d,e,f*}。



[下载：下载全尺寸图像](#)

图9.9。H.264子像素插值。（a）半像素插值。（b）四分之一像素插值。

在四分之一像素细化的情况下，使用更简单的插值；例如，获取位置的插值四分之一像素值*m*在图9.9中，我们使用以下方程：

$$m=\frac{(c+h)}{2} \tag{9.9}$$

所有插值结果四舍五入到范围0...255，在执行运动搜索阶段之前。

应当指出，[插值滤波器](#)的选择很重要，因为它可能会引入偏差（例如，请参阅贝勒斯和德哈恩的工作[7]）。

### 示例9.3子像素运动估计

在6 × 6搜索窗口*S<sub>k-1</sub>*，使用当前框架模板*S<sub>k</sub>*，如下所示。使用简单的两点插值过滤器将整数像素

结果细化为半像素精度：

$$\mathbf{S} = \begin{bmatrix} 1 & 5 & 4 & 9 & 6 & 1 \\ 6 & 1 & 3 & 8 & 5 & 1 \\ 5 & 7 & 1 & 3 & 4 & 1 \\ 2 & 4 & 1 & 7 & 6 & 1 \\ 2 & 4 & 1 & 7 & 8 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}; \mathbf{M} = \begin{bmatrix} 3 & 9 \\ 1 & 4 \end{bmatrix}$$

解决方案

所选整数像素的SAD值 $[d_x, d_y]$ 候选人是：

<i>i</i>	<i>j</i>	SAD( <i>i,j</i> )	<i>i</i>	<i>j</i>	SAD( <i>i,j</i> )	<i>i</i>	<i>j</i>	SAD( <i>i,j</i> )
-1	-1	17	0	1	7	-1	-2	8
-1	0	18	1	-1	11	-2	-2	14
-1	1	15	1	0	13	-2	-1	18
0	-1	2	1	1	17	-2	0	5
0	0	11	0	-2	7	2	-2	18

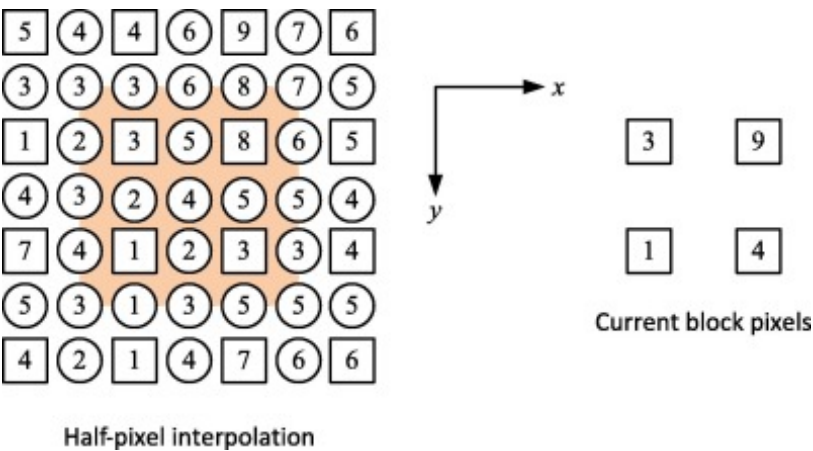
[下载：下载全尺寸图像](#)

显然，通过检查，所有其他候选向量的SAD值都大于2。因此，此块的整数像素运动矢量是  $\mathbf{d} = [0, -1]$ 。

接下来计算半像素精确运动矢量。使用简单的两点过滤器仅插值参考帧的值。例如，在水平方向：

$$s[x + 0.5, y] = \frac{s[x, y] + s[x + 1, y]}{2}$$

一旦所有值都插值，在计算子像素SAD之前，通常要四舍五入结果。我们可以使用： $[s[x + 0.5] - 0.5]$ 四舍五入以0.5结尾的值。使用这种方法，整数像素结果附近的插值半像素值如下：



[下载：下载全尺寸图像](#)

下表给出了子像素SAD值：

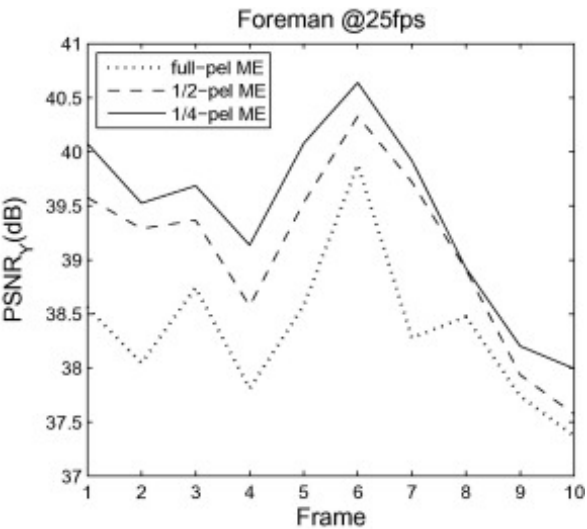
$i$	$j$	$SAD(i,j)$	$i$	$j$	$SAD(i,j)$
0	-1	2	0.5	-1.5	>2
-0.5	-1	10	0	-0.5	>2
0.5	-1	7	-0.5	-0.5	>2
0	-1.5	>2	0.5	-0.5	>2
-0.5	-1.5	>2	-	-	-

[下载：下载全尺寸图像](#)

因此，该块的半像素运动矢量与以前相同，即 $\mathbf{d} = [0, -1]$ 。

### 9.3.3。表演

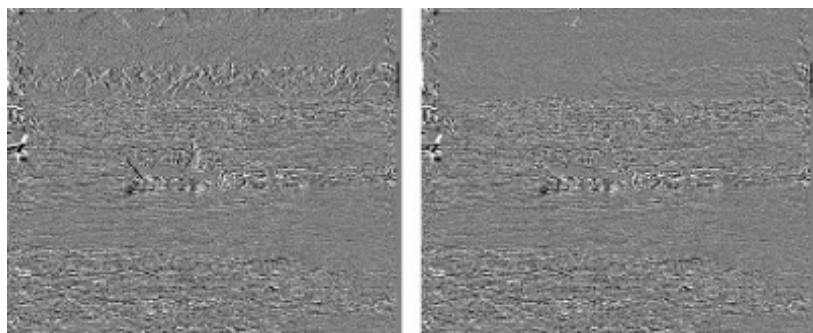
分数像素搜索的好处见图9.10，图9.11。图9.10显示了工头序列的10帧（CIF为25 fps）的半像素和四分之一像素估计，从而提高了PSNR。这是基于在每个情况下使用原始框架作为参考的。可以看出，在这种情况下，可以获得高达1.5分贝的增益。还可以观察到，虽然四分之一像素搜索确实提供了额外的好处，但增益减少了。在大多数情况下，以八分之一像素分辨率进行搜索将提供一些较小的额外增益，但超出此范围，回报会递减。



[下载：下载全尺寸图像](#)

图9.10。全像素、半像素和四分之一像素运动估计的比较。





[下载：下载全尺寸图像](#)

图9.11。亚像素运动估计对剩余误差的影响。左：整数像素ME。右：半像素的我。

图9.11显示了海岸警卫队的DFD信号示例，比较了整数和半像素的搜索方法。残余能量的减少是显而易见的，特别是在背景中船只和河岸区域。然而，虽然河流区域有所改善，但水的动态纹理给平移运动模型造成了严重问题。第13章将进一步讨论使用动态模型建模此类纹理的技术。研究发现，与整数像素情况相比，在H.264/AVC中使用四分之一像素精确运动预测有助于节省10%至20%的比特率。

#### 9.3.4。无插值方法

我们已经看到，四分之一像素运动估计可以为视频编解码器的速率失真性能做出重要贡献。通过使用子像素估计，基于插值的搜索值，每个预测宏块的剩余能量可以显著减少，但只能以计算复杂性增加为代价。后一点对H.264/AVC和HEVC等编解码器特别相关，其中需要估算一组详尽的宏块分割的成本，以实现最佳模式选择。

为了降低插值成本，提出了完全基于全像素SAD分布的亚像素运动估计新方案。Hill等人[8]的方法使半像素和四分之一像素搜索都以SAD表面的智能无模型估计为指导。他们使用一种基于估计模糊度样本之间二维参数化抛物面插值的方法，在纯平移存在的情况下近似实际行为。它们还包括一个后备策略，将速率失真性能提高到接近完全插值的水平。这种方法比完全插值系统快12%至20%。

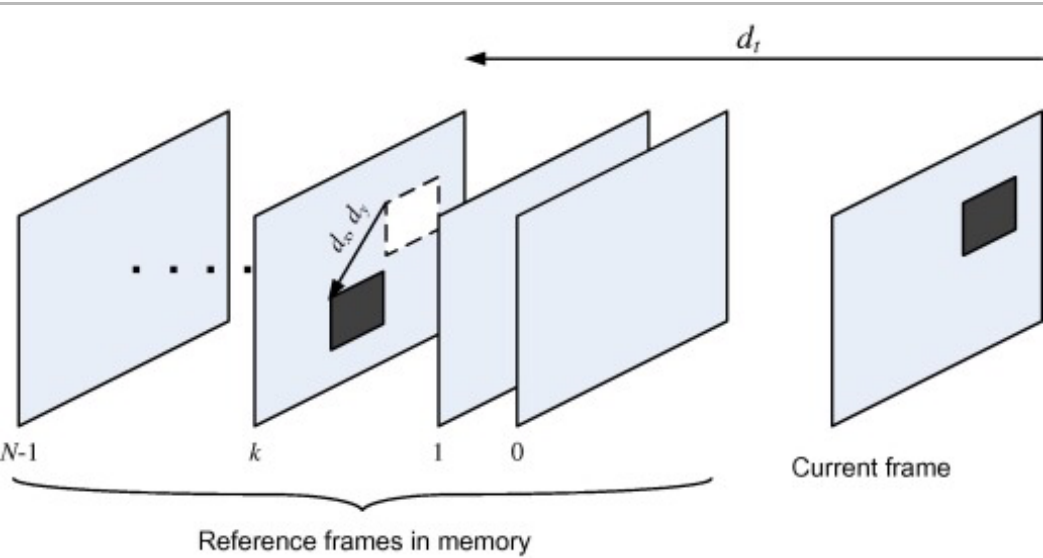
希尔和布尔[9]使用二维内核方法扩展了参考文献[8]的方法。与EPZS四分之一像素方法相比，这减少了30%的四分之一像素搜索位置，总体速度约为10%。

### 9.4。多参考框架运动估计

#### 9.4.1。理由

在常规运动估计中，只使用一个参考框架。然而，Wiegand等人[10]观察到，此类方法不利用视频序列中的长期统计依赖性，多参考帧运动估计（MRF-ME）可以提供许多好处。多个参考帧首先在H.263中标准化，从那时起一直是大多数编解码器的功能[6]。通常，用作参考的帧只是经过解码的帧；然而，它们也可能包括扭曲的版本，以解释各种相机运动。

参考帧在编码器和解码器的参考图片缓冲区中组装，内容的同步必须在两端保持同步。图9.12说明了这一点。当使用滑动窗口更新时，这很简单，但当使用其他更智能的更新时，需要信号传递。



下载：下载全尺寸图像

图9.12。多参考帧运动估计。

### 9.4.2。MRF-ME的性能、复杂性和性能

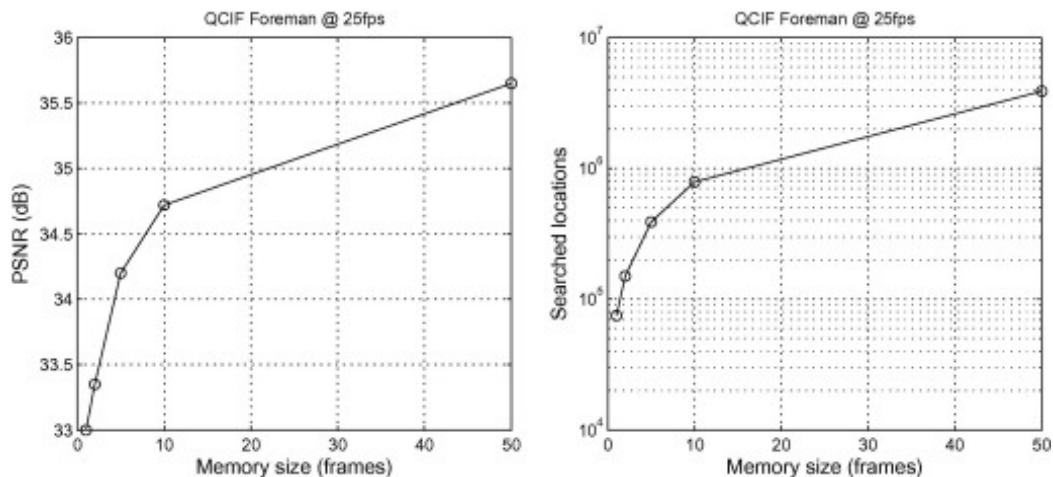
#### 财产

Al-Mualla等人对长期内存的特性进行了很好的描述。[12] 详细介绍了读者。总结一下（假设滑动窗口更新方法）：

1. 长期内存空间位移的分布是中心偏置的。这意味着 $\mathbf{d} = [0,0,d_z]$ 具有最高的相对发生频率。
2. 长期记忆时间位移的分布为零偏。这意味着最新的参考框架是最常用的候选框架。 $\mathbf{d} = [0,0,0]$ 是最常见的矢量。
3. 长期记忆运动场平滑且变化缓慢。因此，适用于单帧搜索的搜索方法也可以为多帧情况提供解决方案。

#### 性能和复杂性

MRF-ME利用图像序列中的长期统计依赖性来提供编码增益。然而，为了做到这一点，运动估计算法不仅必须跨搜索网格中的空间位置进行搜索，还必须在多个候选帧中进行时间搜索。随着处理器功率的增加以及集成的增加和内存成本的降低，这种方法现在是可行的。尽管如此，它仍然可以提供主要的计算开销，降低复杂性的方法是可取的。图9.13（左）显示了Foreman（QCIF @25 fps）可能带来的性能提升。该图显示了长达50个参考帧的长期内存的好处，表明PSNR从一个帧的33分贝增加到50帧的35.5分贝。



下载：下载全尺寸图像

图9.13。MRF-ME性能。左：PSNR。对：复杂性。

然而，编码性能的这种改进被搜索最佳运动矢量所需的额外复杂性所抵消。图9.13（右）显示了同一序列中每帧搜索的位置数量（用于完整搜索）从一个参考帧的80,000个增加到50个参考帧的4,000,000多个。即使有了今天的技术，复杂性的增加也可能使MRF-ME变得棘手。

### 9.4.3. 降低复杂性 MRF-ME

提出了几种在不影响其相关编码增益的情况下提高MRF-ME效率的方法。Su和Sun[11]利用块运动场内的平滑度和相关性，提供与H.264/AVC JM相似的编码性能，但复杂度约为20%。Al-Mualla等人[13]表明，他们的单纯形方法可以为长期记忆运动估计提供好处。基于单纯形的方法具有许多有用的属性，在这种情况下具有吸引力。例如，它并非固有基于单模态错误表面，而是在其候选搜索位置中提供多样性。表9.1列出了该算法的编码性能和复杂性结果；这是基于300帧工头（QCIF @25 fps），使用块大小为 $16 \times 16$  pixels最大运动位移为15像素。搜索在50个参考帧内以全像素精度进行。

表9.1。基于单纯形（SMS）多参考帧运动估计的性能。结果来自参考文献。[13]。

算法：	FS	MR-FS	MR-SMS
PSNR（分贝）：	32.2	33.97	33.87
搜索位置/帧	77,439	3,554,700	106,830

可以看出，使用这种方法可以显著提高性能；编码增益仅减少0.1分贝，同时加速约33倍。

## 9.5。用于运动估计的可变块大小

9.5.1。块大小的影响

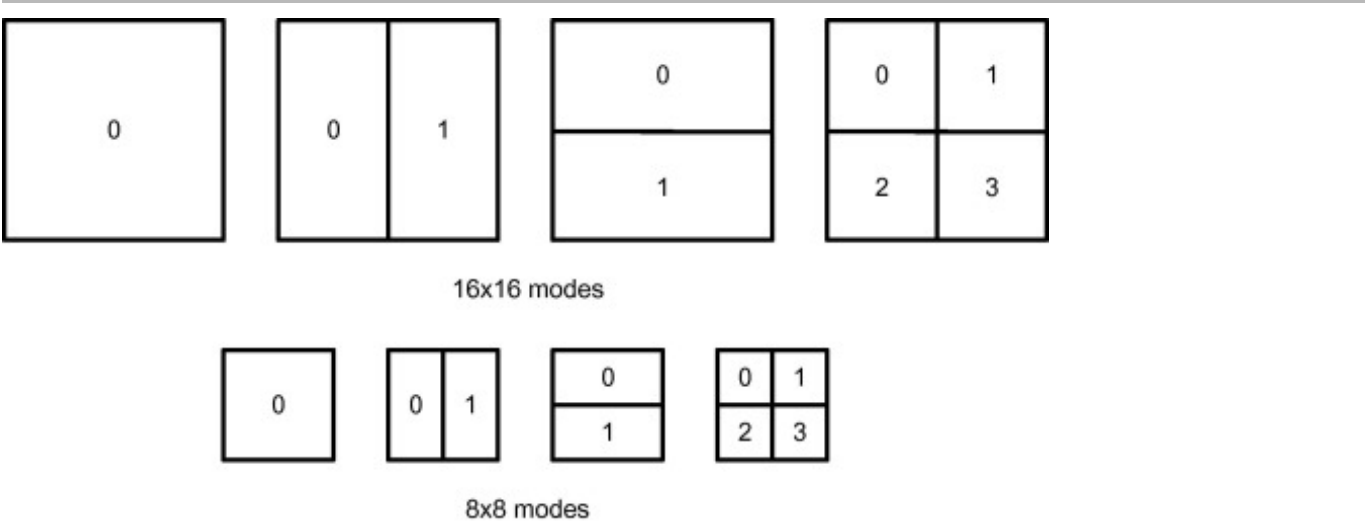
最近标准性能提高的主要原因之一是在变换和运动估计中引入了可变块大小。这使块的形状和大小能够在速率失真意义上进行优化，同时考虑到内容类型、空间和时间特征以及比特率约束。我们在第八章中看到，块大小会影响DFD精度和运动比特率开销。因此，块大小不应该固定，而应该允许在速率-失真框架内适应内容类型，这似乎是完全明智的。

9.5.2。实际操作中的可变块大小

H.264/AVC支持的块大小是：

- **16 × 16 模式：** 16 × 16,16 × 8,8 × 16,8 × 8。
- **8 × 8 模式：** 8 × 8,8 × 4,4 × 8,4 × 4。

这些如图9.14所示，发现使用固定块尺寸可节省约15%的费用。使用Elecard Streameye分析仪[14]制作的Foreman序列，实际编码帧中的块大小分布如图9.15所示。



[下载：下载全尺寸图像](#)

图9.14。H.264/AVC支持的可变块大小。顶部：16 × 16模式1-24。底部：8 × 8模式1-24。



[下载：下载全尺寸图像](#)

图9.15。块大小分布（使用H.264/AVC编码的工头序列）。使用参考文献制作。[\[14\]](#)。

## 9.6. 可变大小变换

### 9.6.1. 整数变换

最近的标准，如H.264/AVC[\[6\]](#)和HEVC[\[15\]](#)，使用块变换来编码帧内或预测残差，但使用可变大小，而不是使用单一的变换类型。例如，在H.264/AVC中，最小的块大小是 $4 \times 4$ 与 $4 \times 4$ 使用DCT。这个整数变换由Malvar等人引入[\[16\]](#)，并提供了好处：

- 变换块大小与基本运动估计块匹配，因此在复杂局部运动和具有随机纹理的区域时，转换块大小表现更好。
- 由于HEVC和H.264/AVC的块大小变化而得到改善，运动补偿后的[剩余信号](#)降低了[空间相关性](#)。如前所述，这意味着转换效率较低，因此 $4 \times 4$ [就装饰关系而言](#)，整数变换通常与较大的变换一样好。
- 它提供了减少边缘噪音的好处。
- [逆变换与正向变换匹配](#)，具有精确的整数运算，因此消除了编码器-解码器不匹配。
- 整数变换在计算上是有效的，需要更小的处理字长，并且只需加法和移位操作即可实现。

[第五章](#)我们看到 $1-D4 \times 4$ DCT由以下方式给出：

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}} \cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}} \cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}} \cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}} \cos(\frac{\pi}{8}) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}} \cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}} \cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}} \cos(\frac{\pi}{8}) & -\sqrt{\frac{1}{2}} \cos(\frac{3\pi}{8}) \end{bmatrix} \\ &= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad \text{where:} \quad \begin{aligned} a &= \frac{1}{2} \\ b &= \sqrt{\frac{1}{2}} \cos(\frac{\pi}{8}) \\ c &= \sqrt{\frac{1}{2}} \cos(\frac{3\pi}{8}) \end{aligned} \end{aligned} \quad (9.10)$$

放开  $d = \frac{c}{b}$ , 变换的整数版本可以写成如下:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \otimes \begin{bmatrix} a & a & a & a \\ b & b & b & b \\ a & a & a & a \\ b & b & b & b \end{bmatrix}$$

在哪里  $\otimes$  表示标量乘法。自  $d \approx 0.414$ , 下一步是将  $d$  近似为 0.5, 给出:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a & a & a & a \\ \frac{b}{2} & \frac{b}{2} & \frac{b}{2} & \frac{b}{2} \\ a & a & a & a \\ \frac{b}{2} & \frac{b}{2} & \frac{b}{2} & \frac{b}{2} \end{bmatrix} \quad (9.11)$$

因为对于所有行向量,  $\mathbf{a}_i \mathbf{a}_j^T = 0$ , 基向量保持正交。要使矩阵正交, 我们必须确保  $\|\mathbf{a}_i\| = 1$ 。因此:

$$a = \frac{1}{2}; \quad b = \sqrt{\frac{2}{5}}; \quad d = \frac{1}{2}$$

将这个可分离变换扩展来处理二维信号得到:

$$\begin{aligned} \mathbf{C} &= \mathbf{A} \mathbf{S} \mathbf{A}^T \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [\mathbf{S}] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \otimes \mathbf{E} \end{aligned} \quad (9.12)$$

其中:

$$\mathbf{E} = \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

因此, 我们可以执行正向和逆变换操作:  $\mathbf{S} = \mathbf{A} \mathbf{C} \mathbf{A}^T$  with the scalar multiplication by  $\mathbf{E}$  absorbed into the [quantization](#) process.



9.6.2。直流系数变换

我们早些时候看到，对亮度和色度信号的内部预测可以为编码非纹理区域提供显著的好处。如果一个平滑变化的区域延伸到整个编码块，那么通过对直流系数进行进一步转换，可以获得额外的好处。 $4 \times 4$ 转化。在H.264中， $a2 \times 2$ 变换也应用于四项的直流系数 $4 \times 4$ 每个色度成分的块。这提供了好处，因为对于平滑变化的区域，自相关系数将接近统一，重建精度与变换大小成反比。对于非纹理区域， $8 \times 8$ 因此，变换是 $4 \times 4$ 转化。

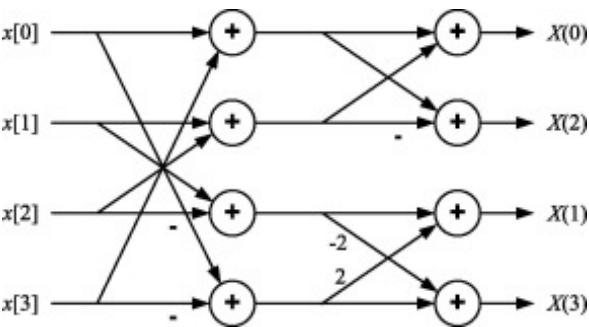
例9.4 简化复杂度整数变换



下载

解决方案

快速整数变换的架构如下：



下载：下载全尺寸图像

可以看到，该变换只需要8个加/减法和2个1位移位。

9.7。环内解封操作

由于DFD信号的运动估计和转换编码，混合视频编解码器中可能会出现阻塞伪影。因此，自H.261以来，标准化视频编解码器中以各种形式使用解块过滤器。事实证明，它们可以改善客观和主观的表现，并随着时间的推移变得越来越复杂。Wiegand等人[6]和List等人[17]对H.264/AVC中使用的解块过滤器进行了出色的概述；摘要如下。

当解块过滤器集成到运动预测循环中时，它们表现最好，因为这消除了漂移，并使所有帧类型受益。图9.16说明了解块操作的想法。边缘样本根据一套与当前量化条件相关的标准进行过滤，并估计边缘是实数还是通过编码诱导的估计。首先计算块边缘附近像素的绝对差异，然后根据算法9.5解封操作

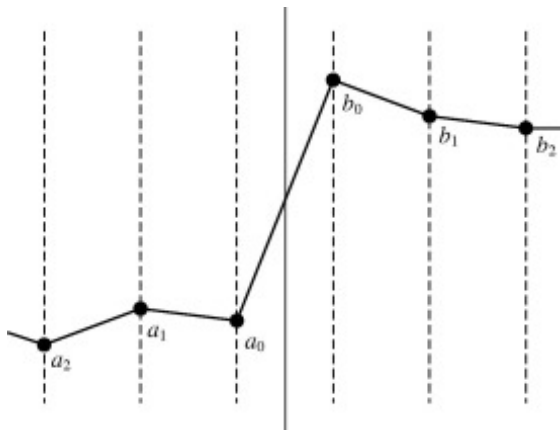
1. Set thresholds  $\alpha(QP)$  and  $\beta(QP)$ ;
2. Filter  $a_0$  and  $b_0$  iff:  $|a_0 - b_0| < \alpha(QP)$  AND  $|a_1 - a_0| < \beta(QP)$  AND  $|b_1 - b_0| < \beta(QP)$ ;
3. Filter  $a_1$  and  $b_1$  iff:  $|a_2 - a_0| < \beta(QP)$  OR  $|b_2 - b_0| < \beta(QP)$ .

下载：下载全尺寸图像



任何量化诱导的工件的可能大小。如果边缘比编码可能诱导的任何边缘都大，那么它最有可能是实际的边缘，不应该被过滤。据报道[6]，加入去阻塞过滤器可节省高达10%的比特率，以获得同等质量。

FEEDBACK 



[下载：下载全尺寸图像](#)

图9.16。一维边缘示例。

被调用的过滤条件取决于两个阈值 $\alpha(QP)$ 和 $\beta(QP)$ 其中第二个阈值比第一个阈值小得多。解决过程在[算法9.5](#)中描述。

工头的示例框架，包括和不应用解决过滤器，如[图9.17](#)所示。过滤操作的主观好处可以看得很清楚。无法直接看到的是平滑和关联更好的DFD信号的好处，这将提高转换编码效率，从而降低比特率。



[下载：下载全尺寸图像](#)

图9.17。解封过滤器的效果说明左：没有解封。右：应用解决过滤器。

## 9.8。摘要

本章整合了第4章、第5章、第7章和第8, [章第4](#), [章第5](#), [章第7](#) and [章第8章](#)的概念，形成了所谓的

基于块的**混合编码器**。这种架构已通过**标准化流程得到**普遍采用，是我们所有电视、存储和流媒体应用程序的主力。通过推进国际标准，这一基本架构得益于几项增强。本章介绍了一些关键增强功能，正是这些增强功能负责提供H.264/AVC和HEVC等最新标准提供的卓越性能。

对于视频编解码器操作的交互式演示，读者请参阅与本文相关的**Matlab代码**，该代码可以从[www.bristol.ac.uk/vi-lab/demos](http://www.bristol.ac.uk/vi-lab/demos)下载。

Recommended articles

Citing articles (0)

## 参考文献

- [1] P.斯特罗巴赫  
**树形结构场景自适应编码器**  
IEEE通信交易, 38 (4) (1990年), 第477-486页  
[在Scopus中查看记录](#) [谷歌学术](#)
- [2] 建议ITU-T H.264 国际标准ISO/IEC 14496-10, ITU-T, 2013年。  
[谷歌学术](#)
- [3] 美国。爱立信  
**用于混合预测/转换编码的固定和自适应预测器**  
IEEE Communications Transactions, 33 (12) (1985), 第1291-1302页  
[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)
- [4] B.吉罗德  
**分数佩尔精度的运动补偿预测**  
IEEE Communications Transactions, 41 (4) (1993年), 第604-612页  
[在Scopus中查看记录](#) [谷歌学术](#)
- [5] B.吉罗德, E. 斯坦巴赫, N.法伯  
**H.263压缩标准的性能**  
VLSI信号处理期刊, 17 (1997年), 第17页。101-111  
[在Scopus中查看记录](#) [谷歌学术](#)
- [6] T. 维冈德, G. 沙利文, G. Bjøntegaard, A. 卢特拉  
**H.264/AVC视频编码标准概述**  
IEEE 视频技术电路和系统交易, 13 (7) (2003), p.560-576  
[在Scopus中查看记录](#) [谷歌学术](#)
- [7] E. 贝勒斯, G. de Haan  
**亚像素运动估计分析**  
SPIE视觉通信和图像处理会议记录 (1999年), p.1452-1463  
[在Scopus中查看记录](#) [谷歌学术](#)
- [8] P.希尔, T. 邱, D. 公牛, C. 卡纳加拉贾

## 无插值子像素精度运动估计

IEEE 视频技术电路和系统交易, 16 (12) (2006), p.1519-1526

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[9] P.希尔, D.公牛

### 使用内核方法进行子像素运动估计

信号处理: 图像通信, 25 (4) (2010年), p.268-275

[文章](#)  [下载PDF](#) [在Scopus中查看记录](#) [谷歌学术](#)

[10] T. Wiegand, X. 张, B. 吉罗德

### 长期记忆运动补偿预测

IEEE 视频技术电路和系统交易, 9 (1)(1999), pp.70-84

[在Scopus中查看记录](#) [谷歌学术](#)

[11] Y. 苏, M.-T.周日

### H.264/AVC快速多参考帧运动估计

IEEE Transactions on Circuits and Systems for Video Technology, 16 (3) (2006), pp.447-452

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[12] M.Al-Mualla, C.卡纳加拉贾, D.公牛

### 移动通信视频编码

学术出版社 (2002)

[谷歌学术](#)

[13] M.Al-Mualla, C.卡纳加拉贾, D.公牛

### 用于单引用和多引用运动估计的单纯形最小化

IEEE 视频技术电路和系统交易, 11 (12) (2001), pp.1209-1220

[在Scopus中查看记录](#) [谷歌学术](#)

[14] <<http://www.elecard.com/en/products/professional/analysis/streameye.html>>。

[谷歌学术](#)

[15] G.沙利文, J.-R.哦, W.J.韩, T.维冈德

### 高效视频编码 (HEVC) 标准概述

IEEE 视频技术电路和系统交易, 22 (12) (2012), p.1649-1668

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[16] H. 马尔瓦尔, A. 哈拉普罗, M.卡尔切维奇, L. 凯罗夫斯基

### H.264/AVC中的低复杂度变换和量化

IEEE 视频技术电路和系统交易, 13 (2003), 第598-603

[在Scopus中查看记录](#) [谷歌学术](#)

[17] P.A.列表, J. 莱内马, G. Bjøntegaard, M.卡尔切维奇

### 自适应解块过滤器

IEEE 视频技术电路和系统交易, 13 (2003), 第614-619

☆ 有关图9.17的颜色和更高质量的版本，请参阅电子版本或网站。

版权所有?2014爱思唯尔有限公司。保留一切权利。



[关于ScienceDirect](#)

[远程访问](#)

[购物车](#)

[广告](#)

[联系和支持](#)

[条款和条件](#)

[隐私政策](#)



我们使用cookie来帮助提供和增强我们的服务，并定制内容和广告。继续即表示您同意**使用cookie**。

版权所有?2021爱思唯尔B.V.或其许可方或贡献者。ScienceDirect ®是爱思唯尔B.V.的注册商标。

ScienceDirect ®是爱思唯尔B.V.的注册商标。