# 第二章 H.264 视频图像压缩基本原理

## 2.1 H.264 视频图像压缩理论

H.264 视频图像压缩算法采用分层设计，从概念上可分为两层：视频编码层（VCL，Video Coding Layer）和网络提取层（NAL，Network Abstraction Layer）。其中视频编码层层负责将视频图像高效率的编码表示，网络提取层负责将视频编码层中得到的数据进行打包和传送，使其符合网络传输协议的标准[10]。视频编码层是 H.264 视频图像压缩算法的核心部分，视频编码的效率主要取决于视频编码层，因此可以通过改进视频编码层优化算法。图 2.1 表示了 H.264 分层结构图。

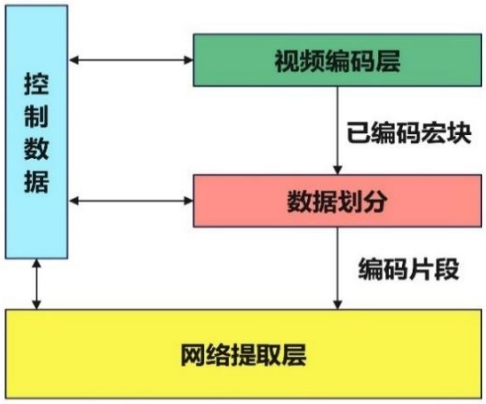


图 2.1 H.264 分层结构

视频图像中含有信息冗余，视频编码的基本思想是寻找信息的相关性，从而减少冗余信息，达到压缩的目的。目前很多的视频编码算法，都是基于块的混合编码方法，其主要模块有预测编码、变换编码、量化、运动估计与运动补偿等，对这些模块进行整合就可以得到完整的压缩算法。H.264 中的 VCL 层也是一种基于块的混合编码方法，其基本原理为通过帧内预测去除空间冗余，通过运动估计去除时间冗余，通过变换编码去除频率冗余，通过熵编码去除信息熵冗余。

在 H.264 编码中，通常将待编码的帧分为三类，分别为 I 帧（独立编码帧）、P 帧（前向预测帧）、B 帧（双向预测帧）。当需要进行编码的当前帧为 I 帧时，选择帧内预测模式，当前帧经过帧内预测再与原始图像相减得到残差值，将残差值进行变换、量化和熵编码，即可得到编码后的码流。当需要编码的帧为 P 帧或 B 帧时，选择帧间预测模式，将当前帧与参考帧进行运动估计得到的预测帧与当前

帧相减得到残差，将残差进行变换、量化和熵编码，就得到了编码后的码流[11]。

在解码时，若为I帧，则直接进行反量化和反变换重构图像；如果为P帧或B帧，所得到的为重构的残差图像，此时需要利用已经编码后的参考帧进行运动补偿，重新计算出预测帧，将解码得到的残差值与预测帧相加，即可重建当前帧。

## 2.2 色彩空间转换

原始图像的色彩空间一般都为RGB格式，即每个像素都有红色、绿色、蓝色三个分量，每个颜色占 8bit 数据。而 H.264 算法中需要将图像的色彩空间转化为YCbCr格式，其中 Y 分量代表图像的亮度，Cb 分量和 Cr 分量分别代表图像的蓝色色度和红色色度。式（2-1）为RGB色彩空间与YCbCr色彩空间的转换公式。

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = -0.169R - 0.331G + 0.5B + 128 \\ Cr = 0.500R - 0.419G - 0.081B + 128 \end{cases} \qquad 式(2-1)$$

有研究表明，人眼对亮度变化的敏感度比色度变化的敏感度高很多，因此我们可以认为，图像色彩空间中的 Y 分量相对于 Cb 分量和 Cr 分量更加重要，故在对待压缩的图像进行色彩空间变换时，可以降低 Cb 分量和 Cr 分量的采样率从而在对图像预处理阶段降低图像的数据大小。目前主要的采样格式有 YCbCr 4:4:4、YCbCr 4:2:2 和 YCbCr 4:2:0。其示意图分别如图 2.2 所示：

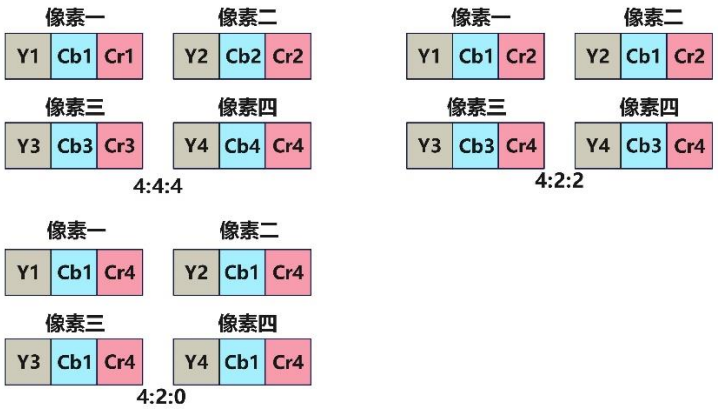


图 2.2 三种采样方式示意图

其中，4:4:4 表示色度分量无下采样，即一个 Y 分量对应一个 Cb 分量和一个

Cr 分量；4:2:2 表示在采样中，采用 2:1 的水平下采样，没有垂直下采样，即每两个同一行相邻的 Y 分量共用一个 Cb 分量和一个 Cr 分量；4:2:0 表示在采样中，采用 2:1 的水平下采样，2:1 的垂直下采样，即每相邻 2×2 的像素块共用一个 Cb 分量和 Cr 分量。如图 2.3 和图 2.4 分别是一张图像的 R、G、B 分量和 Y、Cb、Cr 分量。

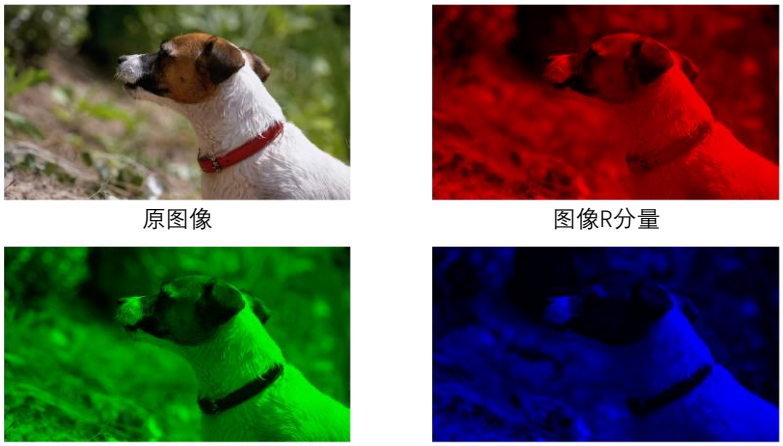


图 2.3 图像 RGB 分量

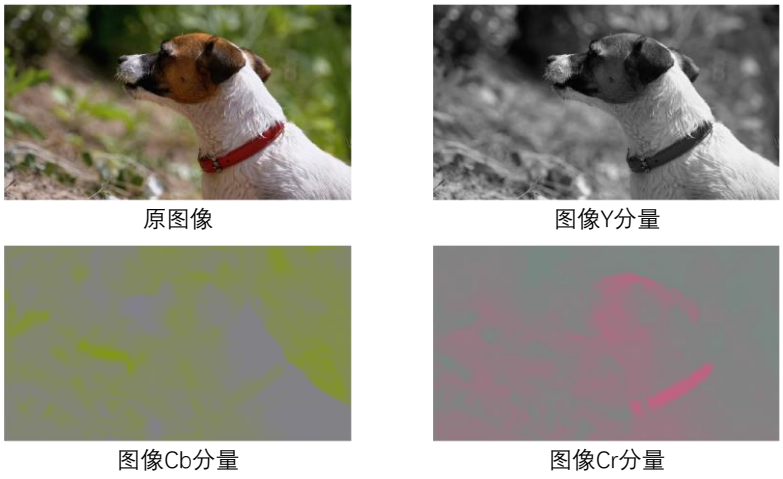


图 2.4 图像 YCbCr 分量

H.264 算法采用 YCbCr4:2:0 的采样方式，实际操作的过程中可以将图像分为 2×2 的单元，每个单元中 Y 值采样 4 次，而只采样左上角的 Cb 值和右下角的 Cr 值，作为整个单元的 Cb 和 Cr 值。这样采样的优点是虽然损失了一定精度，但也在人眼几乎不可见的前提下减小了数据存储量。

## 2.3 变换编码

### 2.3.1 变换编码概述

变换编码是一种应用及其广泛的编码方式，其主要思想是通过数学映射，将时域或空域的信号转化为变换域的信号进行描述，从而改变信号能量的分布。通常，图像能量在空域中的分布都是较为分散的，若通过变换能将图像能量集中，就更加有利于采取其他处理方式对图像进行压缩，例如 Zig-Zag 扫描，游程编码等，从而减少图像数据量。

有研究发现，人眼不易察觉图像的高频信息，因此可以将图像变换至频域，再通过量化的方法去除图像中的高频信息，即可在人眼不易察觉的情况下减少图像的信息量，从而达到压缩的目的。

早期在图像压缩时，采用了快速傅立叶变换的方法，后来的研究中，又陆续出现了多种正交变换的方法，如离散余弦变换（Discrete Cosine Transform，DCT）、K-L 变换等。其中，K-L 变换的效果最为理想，但由于变换矩阵随着图像像素值的变化而变化，算法时间复杂度较高，运行速度慢，因此实现较为困难。DCT 变换的性能与 K-L 变换相比差距不大，并且算法简单，易于在硬件上实现，所以广泛应用与图像的编码[12]。

除正交变换编码外，小波变换也可应用于图像处理。小波变换的基本思想是将图像信号变为一簇基函数的加权和，而这一簇基函数是通过基本函数平移和伸缩构成。小波变换继承了傅立叶变换的优点，在变换中不会产生能量损失，因此在图像编码领域具有很好的应用前景[13]。

### 2.3.2 变换编码原理及过程

目前，包括 H.264 标准的主流视频编码标准主要应用了 DCT 变换，通过 DCT 变换压缩图像可分为三个步骤，分别是变换、量化、Zig-Zag 扫描法。

（1）变换过程：

式（2-2）、式（2-3）和式（2-4）表示二维 DCT 变换。其中，$F(u，v)$为图像进行 DCT 变换后的序列，$f(x, y)$为 N×N 的数字序列。式（2-5）表示二维逆变换

IDCT 变换：

$$F(u,v)=\frac{2}{N}c(u)c(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)cos\left[\frac{\pi}{2N}(2x+1)u\right]\cos\left[\frac{\pi}{2N}(2y+1)v\right] \text{式}(2-2)$$

$$c(u)=\begin{cases}\frac{1}{\sqrt{2}}, & u=0\\ 1, & u\neq 0\end{cases}, \quad c(v)=\begin{cases}\frac{1}{\sqrt{2}}, & v=0\\ 1, & v\neq 0\end{cases} \text{式}(2-3)$$

$$u,v=0,1,\dots,N-1 \text{式}(2-4)$$

$$f(x,y)=\frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}c(u)c(v)F(u,v)\cos\left[\frac{\pi}{2N}(2x+1)u\right]\cos\left[\frac{\pi}{2N}(2y+1)v\right] \text{式}(2-5)$$

在视频图像压缩中，比较常见的做法是将图像划分为 8×8 像素大小的宏块，对每一个宏块分别做变换。对于 8×8 像素大小的变换，其过程可视为，将图像分解为 64 个基函数的加权和。图 2.5 表示了 64 个基函数。

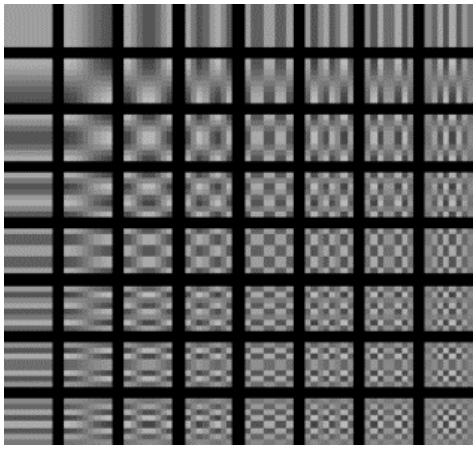

图 2.5 64 个基函数

从图 2.5 中可以看出，水平频率从左到右递增，垂直频率从上到下递增。因此，对图像进行 DCT 变换后，低频分量全部分布在左上角，高频分量全部分布在右下角，从而便于在随后的量化阶段和 Zig-Zag 扫描中去除人眼不易察觉的高频分量。

（2）量化过程

量化过程将图像经过 DCT 变换得到的系数矩阵与量化系数表中的对应值相除，并四舍五入。一般量化表中的左上方的数较小，右下方的数较大，故经过量化后，

高频系数一般会出现较多的零值，从而达到压缩的目的。式（2-6）、式（2-7）分别表示了量化原理和反量化原理：

$$q(x,y) = round\left(\frac{F(x,y)}{Q} + 0.5\right) \qquad 式(2-6)$$

$$F(x,y) = q(x,y) \times Q \qquad 式(2-7)$$

由于在量化过程中，需要取整，无法完全还原图像，所以 DCT 变换会产生失真，是一种有损压缩的方法。

（3）Zig-Zag 扫描法

在经过变换与量化后，图像信息的能量会得到重新分布，此时能量主要集中在左上角，而右下方区域将会出现许多零值。为了便于进行游程编码，不仅需要将图像信息重新排列为一维向量，还需要将能量集中的左上角与能量稀疏的右下角分别连续串联起来，因此引入了 Zig-Zag 扫描法。Zig-Zag 扫描法示意图如图 2.6 所示：

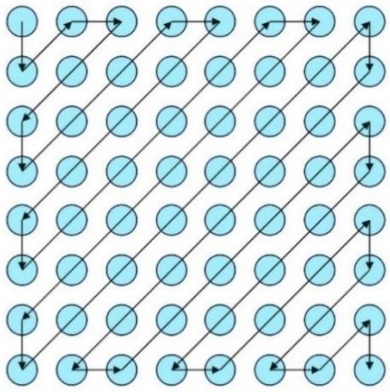

图 2.6 Zig-Zag 扫描法示意图

在经过 Zig-Zag 扫描法后，图像的每一个像素块都被重新排列为一个含有 64 个元素的一维数组。其中，数组的第一个元素称为直流系数（DC 系数），其余的 63 个元素称为交流系数（AC 系数）。将变换后的系数分为直流系数与交流系数的一个显著优点为可以根据其不同的特点应用合适的编码方法，从而达到压缩的目的。图像经过变换编码可以去除其高频分量，如图 2.7 所示，为一幅原始灰度图像与其经过变换编码复原后的图像对比。通过人眼观察，很难发现两幅图像有明

显变化，说明变换编码可以在人眼无法察觉的情况下有效的消除高频分量，从而达到压缩目的。

原始图像

复原图像

图 2.7 原始图像与变换编码后的复原图像

## 2.4 预测编码

预测编码的基本思路是通过已编码的帧或像素得到待编码的帧或像素的预测值，将待编码的帧或像素的真实值与其预测值相减得到残差，再对残差进行编码。由于残差数据量相较于真实值有很大缩减，相应的表示位数也会减少。在复原图像时，只需要将解码后的残差值与参考帧或参考像素相加，即可完全复原。

H.264 中常见的预测编码有差分脉冲编码调制和运动估计算法。一般差分脉冲调制编码应用在帧内编码中，对图像变换后的 DC 系数编码；运动估计算法一般应用在帧间编码中，对待编码的视频帧整体进行预测。

### 2.4.1 差分脉冲编码调制原理

差分脉冲编码调制（Differential Pulse Code Modulation，DPCM）是一种对信号幅度的差值进行编码的调制方式，其利用样本与样本之间存在信息冗余度来进行编码[14]。图像变换后的 DC 系数具有两个显著的特点：第一，由于图像能量主要分布在低频，导致 DC 系数的数值比较大，直接编码会占用大量数据；第二，相邻像素块的 DC 系数值变化不大。根据这两个特点，使用 DPCM 可以很大程度地减少数据量。式（2-8）、式（2-9）表示了 DPCM 原理，其中，$f(k)$为经过 DPCM 编码后的值，N 为一张图像的像素块的数量：

$$f(1) = DC(1) \qquad 式(2-8)$$

$$f(k) = DC(k) - DC(k-1), k = 2,3, \dots, N \qquad 式(2-9)$$

经过 DPCM 编码后，一张图像中的每个像素块的 DC 系数被编码为一个一维数组，数组的第一个元素为第一个像素块的 DC 系数，其余元素为对应位置像素块与上一个像素块 DC 系数的差值。在重建图像时，利用差值与上一个像素块的重建值求和，从而得到重构信号。

### 2.4.2 运动估计算法原理

运动估计是视频处理领域中的一项重要技术，用于分析视频序列中物体的运动信息。通过运动估计，可以确定视频图像序列中相邻帧内运动目标的位移，得到预测帧，从而实现图像的预测编码。

在视频图像序列中，相邻帧之间的像素值会随着时间发生变化，这种变化可以用来描述物体的运动。运动估计的基本原理是通过比较视频序列中的不同帧之间的像素差异，找到最佳的匹配，以确定物体的运动信息。常见的运动估计方法包括块匹配、光流法、特征点匹配等。在包括 H.264 标准的众多主流标准中，都选择使用块匹配法进行运动估计。

## 2.5 熵编码

熵编码是指按照信息熵原理进行的无损编码方式。根据信息论的理论，信源本身具有相关性，并且信源内事件概率分布不均匀导致了信源的冗余。熵编码就是利用信源的统计特性，去除统计冗余，因此，熵编码也被称为统计编码[15]。在视频图像压缩中，熵编码位于压缩系统的末尾，把一系列符号转变为用来存储的二进制码流。在H.264中，主要使用了熵编码中的霍夫曼编码（Huffman Coding）和游程编码（Run Length Coding，RLC）。

### 2.5.1 熵编码基本理论

熵编码的的过程是将信源中的不同字符用不同的二进制码字表示，使每个字符对应一个二进制码字。假设字符$x_i$取自信源符号集合$X_m = \{x_1, x_2, \dots, x_m\}$，对于

字符$x_i$，其出现概率为$p_i$，编码长度为$L_i$，则$L_i$可由式（2-10）表示：

$$L_i = -lgq_i(0 \le q_i \le 1, i = 1,2,\dots,m, \sum_{i=1}^{m} q_i = 1) \qquad 式(2-10)$$

则信源$X_m$的平均编码长度可由式（2-11）表示：

$$l = \sum_{i=1}^{m} p_i L_i = -\sum_{i=1}^{m} p_i lgq_i \qquad 式(2-11)$$

由于信息熵具有极值性，易得当且仅当$\{q_i\} = \{p_i\}$时，式（2-12）成立：

$$H(X) = -\sum_{i=1}^{m} p_i lgp_i \le -\sum_{i=1}^{m} p_i lgq_i \qquad 式(2-12)$$

若式（2-12）中的$q_i = 1/m$，就可得到如式（2-13）所示的最大离散熵定理。

$$H_m(p_1, p_2, \dots, p_m) \le lgm \qquad 式(2-13)$$

最大离散熵的文字表述为，所有概率分布$p_i$所构成的熵，以等概率时最大。

由式（2-12）和式（2-13），可以看出，信源的冗余度是信源符号非等概率分布引起的。只要信源符号不符合等概率分布，就可以使用熵编码进行压缩。具体的思路为对出现概率大的信源符号使用较短的二进制码字，对出现概率较小的信源符号使用较长的二进制码字。

### 2.5.2 霍夫曼编码原理

霍夫曼编码是由美国计算机科学家大卫·霍夫曼在 1952 年发明的一种无损压缩编码算法。霍夫曼编码主要利用信源中不同字符出现的频率不同，使用不同长短的码字对字符编码，因此是一种可变长编码。

霍夫曼编码的关键就是要构建霍夫曼树，霍夫曼树也称为最优二叉树，其特点为带权路径长度（Weighted Path Length，WPL）最短[16]。带权路径长度的计算方法是将一棵树中的所有叶子结点的权重与其到根结点的路径长度的乘积之和。

考虑一颗有 N 个叶子结点的二叉树，每个叶子结点的权重为$W_i\ (i = 1,2, \dots, N)$，它到根结点的路径长度为$L_i$，那么这棵树的带权路径长度可由式（2-14）表示：

$$WPL = \sum_{i=0}^{N} W_i \times L_i \qquad 式(2-14)$$

在进行编码时，构造霍夫曼树需要将所有信源中出现的符号都视为叶子结点，每个符号出现的频率视为该叶子结点的权值。假设信源中共有N个符号，第$i$个符号的权值为$W_i$，则具体的构造步骤为：

（1）将所有叶子结点看成是有 N 棵树的森林，每棵树仅有一个结点；

（2）在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，新树的根结点权值为其左、右子树根结点权值之和；

（3）从森林中删除（2）中选取的两棵树，并将新树加入森林；

（4）重复（2）和（3），直到森林中只剩最后一颗树为止，该树即为所求的霍夫曼树。

求得霍夫曼树后，对于除根结点外的每一个结点，若其为其父亲结点的左孩子，则在其到其父亲结点的路径上记录“0”，若其为其父亲结点的右孩子，则在其到其父亲结点的路径上记录“1”。对于每个信源符号，分别记录其到根结点经过的路径，按顺序记录路径上的“1”和“0”，就得到了信源符号的霍夫曼编码。

现举例说明如何对信源进行霍夫曼编码，各信源符号及相应出现概率如表 2.1 所示。根据上述方法，可得到如图 2.8 所示的霍夫曼树和如表 2.2 所示的各信源符号对应的霍夫曼编码码字。

表 2.1 各信源符号及相应出现概率

| 信源符号$\boldsymbol{X}$ | $\boldsymbol{X_1}$ | $\boldsymbol{X_2}$ | $\boldsymbol{X_3}$ | $\boldsymbol{X_4}$ | $\boldsymbol{X_5}$ |
|---|---|---|---|---|---|
| 出现概率$\boldsymbol{P}$ | 0.40 | 0.20 | 0.20 | 0.15 | 0.05 |

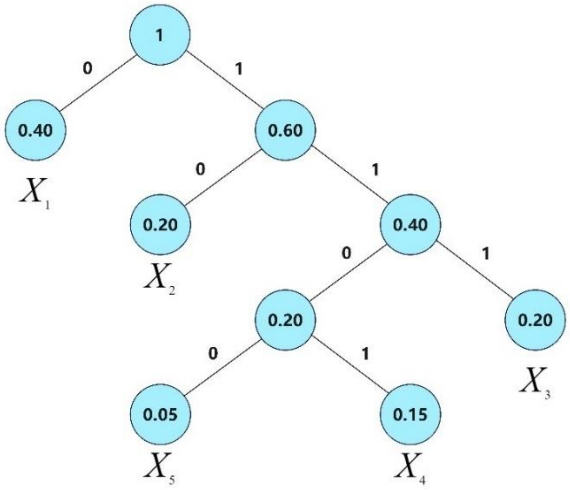


图 2.8 根据表 2.1 所得的霍夫曼树

表 2.2 各信源符号及相应码字

| 信源符号$\boldsymbol{X}$ | $\boldsymbol{X_1}$ | $\boldsymbol{X_2}$ | $\boldsymbol{X_3}$ | $\boldsymbol{X_4}$ | $\boldsymbol{X_5}$ |
|---|---|---|---|---|---|
| 码字 | 0 | 10 | 111 | 1101 | 1100 |

在解码时，需要利用到霍夫曼码表。由于霍夫曼编码互不为前缀的特点，可以在解码端设置一个寄存器，编码开始寄存器为空，每收到一个比特，将其放入寄存器，并于霍夫曼码表比较，若码表中有相同的码字，则输出对应的信源符号；否则，继续读取码流中下一个比特，直到找到码表中对应符号。

### 2.5.3 游程编码原理

游程编码的原理为，用一个符号值或串长代替具有相同值的连续符号，仅在数据发生变化时，记录变化后的值和相同数据重复的次数。例如：有一个字符串“11115550000222”，经过游程编码后可以用“41354032”表示。

游程编码适合在信源中出现连续相同符号的情况，在这种情况下，游程编码具有极高的压缩效率。但如果信源中每两个相邻元素都不同，则这种算法不仅不能对数据进行压缩，反而使数据量增加一倍。在图像经过变换编码后，其 AC 系数的一个显著特点是，系数中含有很多连续的值为 0 的系数，因此，对于变换编码得到的 AC 系数，使用游程编码进行压缩是十分合适的。

在实际程序中，可以使用数据对的形式存储游程编码后的数据，假设编码后

得到了一个（M,N）的数据对，则 M 代表两个非零系数之间连续的零的个数，N 代表下一个非零系数的值。最后，再将数据对进行霍夫曼编码得到二进制码流。

## 2.6 本章小结

在当今主流的视频压缩编码算法中，基于块的混合编码占据了很大一部分，H.264 也包含在内。本章主要以 H.264 视频压缩算法为例，介绍了其编码方法以及所用技术的基础原理。本章对其中色彩空间转换、变换编码、预测编码和熵编码做了详细的阐述和分析，详细的介绍了每种编码方式的具体流程。

# 第三章 改进的视频图像压缩算法

## 3.1 结合帧间差分法与块匹配法的运动估计算法

在视频图像压缩的帧间预测模块，目前主流的算法是基于块匹配法的运动估计算法。虽然随着技术的发展，陆续出现了许多快速块匹配算法；但是块匹配法的时间复杂度依然较大。本文设计了一种结合帧间差分法与块匹配法的运动估计算法，该算法可以在保证压缩效率的基础上，加快运动估计算法的运行速度，从而更有效的对视频图像进行压缩。

### 3.1.1 帧间差分法原理

帧间差分法是一种利用视频中的相邻两帧做差分运算获得运动目标轮廓的方法[17]，通过帧间差分法得到的残差图像也可用于视频图像压缩。由于视频中连续的两帧在时间上跨度极小，一般运动目标的运动位移量也较小，而相邻两帧的背景部分除灰度跳动外，几乎没有变动。因此，当两帧图像做差分运算时，得到的残差图像只有运动目标边缘处灰度值较大，其余部分的灰度值较小，具有较少的信息量。对残差图像进行编码的数据量就会远小于直接对原图像编码。式（3-1）和式（3-2）表示了帧间差分法的原理：

$$D_n(x,y) = f_n(x,y) - f_{n-1}(x,y) \qquad 式(3-1)$$

$$f_{n_{rec}}(x,y) = f_{n-1}(x,y) + D_n(x,y) \qquad 式(3-2)$$

其中，$f_n(x,y)$和$f_{n-1}(x,y)$分别表示视频中的第 n 帧和第 n-1 帧，$D_n(x,y)$表示第 n 帧残差图像。在重建图像时，只需要将解码后的残差图像与上一帧解码端的重建图像相加，即可得到当前帧的重建图像。

帧间差分法具有一个显著优点：由于算法中只涉及两帧图像对应像素值相减，不涉及其他复杂运算，因此，程序设计复杂度低，并且有较快的运行速度。然而，使用帧间差分法进行视频图像压缩也具有许多缺点：（1）当镜头移动导致背景与

前景同时变化时，残差图像的信息量并不会显著减少；（2）帧间差分法对光照等容易造成灰度跳动的噪声比较敏感，导致压缩效率下降。因此，帧间差分法通常需要和其他方法结合使用，以提高其压缩效率和鲁棒性。

### 3.1.2 块匹配法原理

在视频图像序列中，相邻帧之间的像素值会随着时间发生变化，这种变化可以用来描述物体的运动。运动估计的基本原理是通过比较视频序列中的不同帧之间的像素差异，找到最佳的匹配，以确定物体的运动信息。常见的运动估计方法包括块匹配、光流法、特征点匹配等。在 H.264 视频压缩算法中，使用块匹配法进行运动估计。

块匹配法基于一个基本假设，将当前帧图像分为多个大小相同且互相不重叠的宏块，每个宏块中的像素具有相同的运动，且只做平移运动。但实际上块内各个像素的运动不一定相同[18]，所以仅当宏块较小时，上述假设成立。

匹配算法的原理如图 3.1 所示，将视频帧划分为 N×N 大小的宏块，在参考帧的搜索窗口中选择当前编码帧的最佳匹配块，将当前帧的所有最佳匹配块拼接在一起，就得到了当前帧的预测帧，当前帧中的原始宏块与参考帧中的最佳匹配块之间的位移称为该宏块的运动矢量（Motion Vector，MV）。

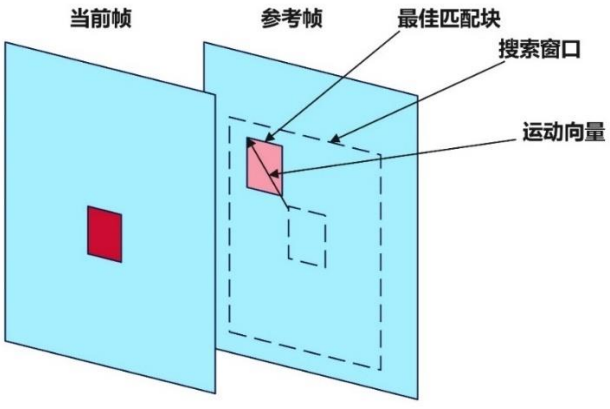


图 3.1 块匹配法原理示意图

使用块匹配法进行运动估计时，在参考帧的搜索窗口内搜索最佳匹配块是十分关键的一个步骤，宏块以及搜索窗口的大小都会影响到搜索效果以及运算速度。宏块大小的选择受以下矛盾影响：（1）宏块较大时，块内所有像素做相同平移运

动的假设通常不合理；（2）宏块较小时，运算量增加，算法运算速度减慢，且易受灰度跳动等噪音影响，估计不够可靠。搜索窗口大小的选择受以下矛盾影响：（1）搜索窗口较大时，算法运算量增加，减慢了算法运行速度；（2）搜索窗口较小时，由于边界约束条件的影响，匹配块收敛不到全局最佳值的可能性增加，使估计的可靠性减少。所以，确定适当的宏块大小和搜索窗口是十分重要的。

为了搜索最佳匹配块，需要确定图像块匹配的判别标准，从而度量两个图像块的相似程度。常见的判别标准有均方误差（Mean Square Error，MSE）、平均绝对误（Mean Absolute Difference，MAD）、绝对误差之和（Sum of Absolute Difference，SAD），当各判别标准最小时，其对应的$(i,j)$为该图像块的运动向量。

设$(i,j)$为参考帧相对当前帧的位移矢量，$f(x,y)$和$f_{ref}(x,y)$分别为当前帧和参考帧的像素值，M×N 为宏块大小，则 MSE、MAD、SAD 的计算方法可分别由式（3-3）、式（3-4）、式（3-5）表示。

$$MSE(i,j)=\frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left|f(x,y)-f_{ref}(x+i,y+j)\right|^2}{MN} \qquad 式(3-3)$$

$$MAD(i,j)=\frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left|f(x,y)-f_{ref}(x+i,y+j)\right|}{MN} \qquad 式(3-4)$$

$$SAD(i,j)=\sum_{x=1}^{M}\sum_{y=1}^{N}\left|f(x,y)-f_{ref}(x+i,y+j)\right| \qquad 式(3-5)$$

其中，使用 MSE 作为判别标准的精度最高，但运算量最大，MAD 和 SAD 的精度相同，但 SAD 只涉及加法和减法，运算量更小，适合在硬件上实现，故一般选择 SAD 作为判别标准。

搜索最优匹配块最准确的方法是全搜索算法，该算法的实现思路是，对搜索窗口中的所有位置计算块匹配误差，从中找到最小值。全搜索算法实现简单，具有很强的鲁棒性，必然能找到搜索窗口中的最佳匹配块，但其运算量过大，无法适应需要实时性的应用场合，因此，各种快速搜索算法陆续出现并快速发展。快速搜索算法与全搜索算法相比，在搜索准确度可以接受的基础上，大大提高了搜索速率，成为了块匹配法中广泛应用的方法。

快速搜索算法是基于误差场单调分布的假设设计的算法，即当搜索窗口内有且仅有一个极小值时，快速搜索算法才是完全精确的。常见的快速搜索算法有三步搜索算法（Three Step Search，3SS）、新三步搜索算法（New Three Step Search，N3SS）、菱形搜索算法（Diamond Search，DS）、六边形搜索算法（Hexagon Based Search，HEXBS）等[19]，在本文中着重三步搜索算法和菱形搜索算法。

1.三步搜索算法

三步搜索算法是一种经典的块匹配快速搜索算法，其算法设计思路简单，实现方便，因而被广泛使用。其具体步骤为：

从搜索窗口的中心点作为起点，以最大搜索范围的一半为步长，计算中心点与周围方形的 8 个点的块匹配误差，找到其中最优匹配块。在下一步中，以该店为中心点，步长减半，再次搜索 9 个点，以此类推，直到步长减少到 1 个像素。在一般的情况下，搜索窗口可以设为±7，则三步搜索算法先后以 4、2、1 为步长，共搜索 25 个点，经历三步完成搜索，因此被称为三步搜索算法，如图 3.2 所示。

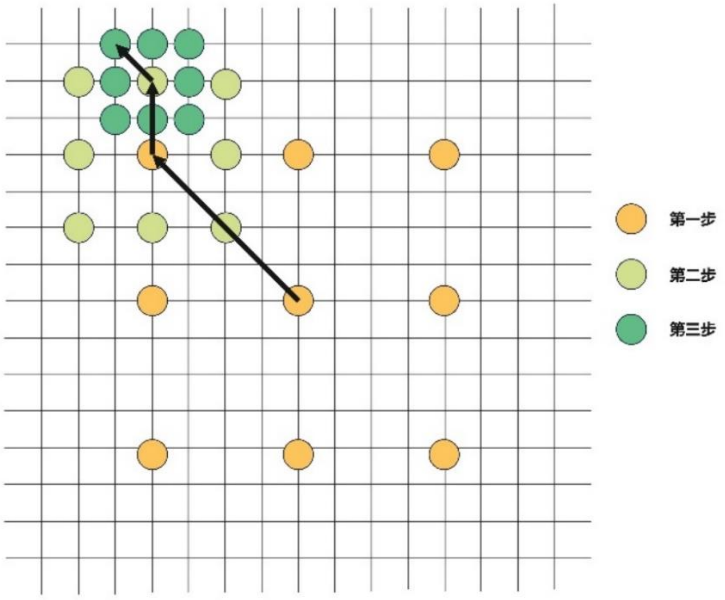


图 3.2 三步搜索算法示意图

三步搜索算法第一步选择的步长较大，对于运动位移量较大的视频序列，可以快速找到全局最小值，但另一方面，有些视频序列中，运动目标在相邻两帧间的位移量较小，在这种情况下，使用三步搜索算法可能会出现无法收敛到全局极小值的情况。相比于三步搜索算法，菱形搜索算法可以适应更多的情况，有更好

的效果。

2.菱形搜索算法

菱形搜索算法使用两种不同大小的菱形进行搜索，分别为大菱形搜索模式（LDSP）和小菱形搜索模式（SDSP）。图 3.3 和 3.4 分别表示了两种搜索模式。

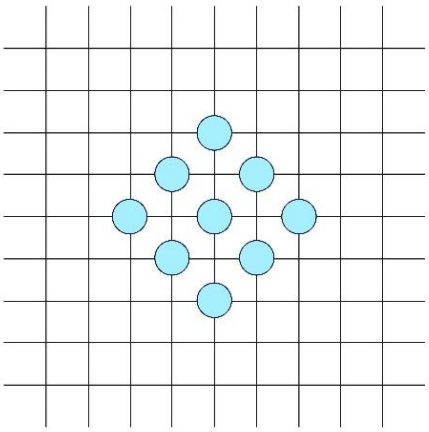

图 3.3 LDSP 搜索模式

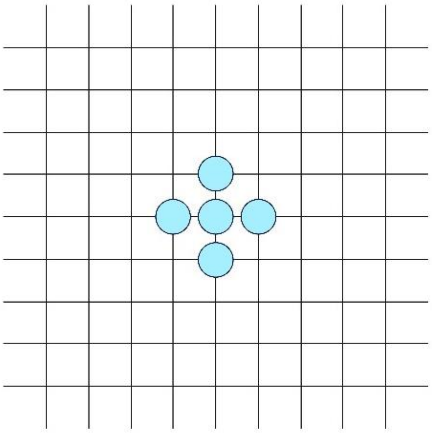

图 3.4 SDSP 搜索模式

菱形搜索算法原理如图 3.5 所示，具体步骤如下：

（1）以搜索原点作为 LDSP 的中心点，比较 LDSP 上 9 个位置的参考宏块，若中心点处的参考宏块为最优匹配块，则转向步骤（3），否则转向步骤（2）；

（2）以上一步中最佳匹配块的位置作为新的中心点重新比较 LDSP 上 9 个位置的参考宏块。若中心处的参考宏块为最优匹配块，则转向步骤（3），否则重复步骤（2）；

（3）以上一步中最佳匹配块的位置作为中心点比较 SDSP 上 5 个位置的参考宏块，得到的最优匹配块则是最终的最佳匹配块。

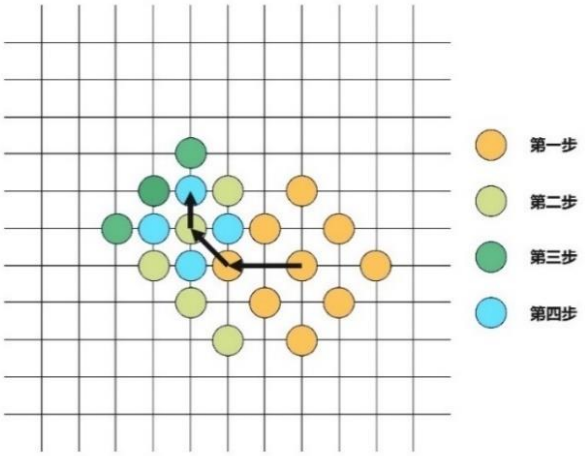


图 3.5 菱形搜索算法示意图

通过块匹配法，可以找到当前帧图像中每个宏块的最佳匹配块，将最佳匹配块拼接至当前帧中相应的位置，就可以得到当前帧的预测图像，将当前帧的真实图像与预测图像做差分运算得到残差图像，再对残差图像进行编码，就可以大大减少视频序列间的时间冗余，从而对视频进行压缩。

在解码时，首先，需要先在解码端重建参考帧以及当前帧的残差图像；下一步，可以根据参考帧的重建图像与运动向量重建当前帧的预测图像，将当前帧的预测图像与其残差图像相加，即可重建当前帧。如图 3.6 所示为当前帧图像、参考帧图像、使用块匹配法得到的预测图像和预测帧与当前帧做差得到的残差图像。从图 3.6 中的残差图像可以看出，预测帧与当前帧的残差所含信息量极少，对残差编码可以有效对图像进行压缩。


图 3.6 块匹配法示例图像

### 3.1.3 改进的运动估计算法原理

运动估计算法是视频图像压缩算法中的重要环节，通过运动估计算法，可以很大程度地减少视频编码的数据量。但是，现有的运动估计算法存在着运算量大，时间复杂度高等问题，本小节将帧间差分法与块匹配法结合，提出了一种结合两种算法优点的新型运动估计算法，以减少运动估计算法的时间复杂度。

基于块匹配法的传统运动估计算法需要对当前帧图像中的每一个宏块都做运动估计，但是在镜头不移动的情况下，视频序列中的背景部分基本上可以视为是静止不动的，所以对于背景部分，可以选择不使用运动估计，只使用帧间差分法得到残差图像；而对于视频序列的运动部分，依然使用运动估计。通过将需要做运动估计的宏块与不需要做运动估计的宏块划分开，就可以减少程序中运动估计算法的调用次数，从而加快算法运行速度。如图 3.7 表示了改进算法的原理。

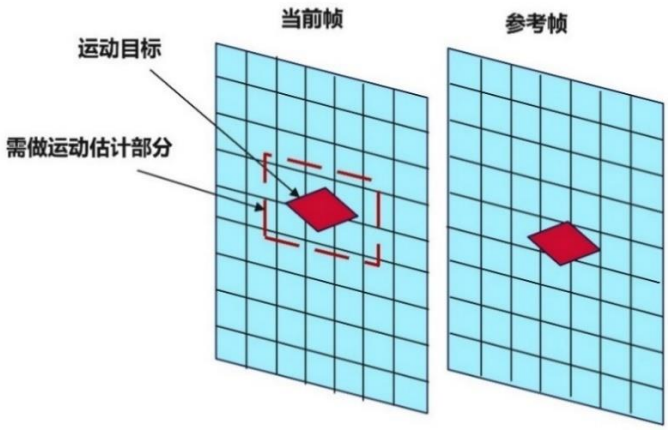


图 3.7 改进算法原理示意图

改进的运动估计算法实现的具体步骤是：

（1）将当前帧图像与参考帧图像做差分运算，得到残差图像；

（2）将残差图像划分为多个大小相同、互不重叠的宏块，对每个宏块中所有的像素值求和，并设置一个划分阈值，若该宏块的像素值之和大于阈值，则说明该残差宏块中含有较大时间冗余，需要做运动估计进一步压缩；否则，则说明该宏块在视频序列中没有明显位移，包含的时间冗余较小，可以直接使用参考帧中的对应位置宏块作为预测图像；

（3）对于所有需要做运动估计的宏块使用块匹配法进行运动估计，得到运

动向量，对于不需要进行运动估计的宏块，视为其运动向量为(0,0)，并根据运动向量与参考帧图像得到当前帧的预测图像；

（4）将当前帧原始图像与预测图像做差分运算，得到残差图像，对残差图像进行下一步编码。

在解码时，首先，需要用到运动补偿算法重建预测帧图像，由于在编码中，已经将不需要做运动估计的宏块的运动向量记录为(0,0)，因此改进算法的运动补偿算法与传统块匹配法的运动补偿算法一致，不需要额外增加计算量。通过运动补偿算法得到预测帧图像后，再将预测帧图像与解码得到的残差图像求和，就可以在解码端复现当前帧图像。

## 3.2 改进的视频图像压缩算法

将改进后的运动估计算法与其他压缩编码模块整合，就得到了完整的视频图像压缩算法。该算法仍采用基于块的混合编码方法，其主要结构如图 3.8 所示。

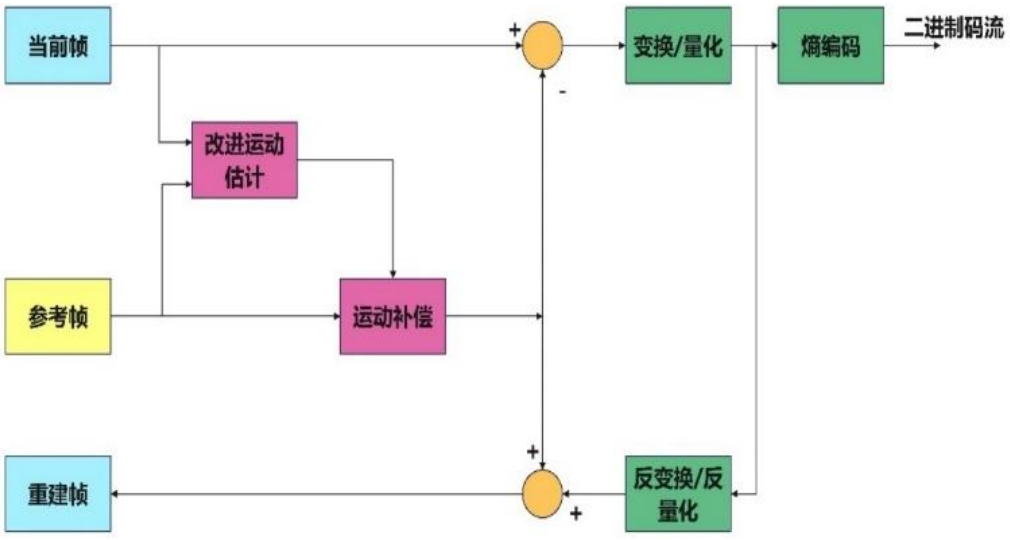


图 3.8 改进视频图像压缩算法结构图

在对视频进行压缩编码前，首先需要将视频序列分为若干画面组（Group of Pictures，GOP），在每一个 GOP 中，还需要将视频序列分为独立编码帧和前向预测帧。每一个 GOP 中的第一帧为独立编码帧，只依靠自身信息进行编码；其余帧为前向预测帧，需要利用前一帧作为参考帧进行帧间预测编码。在编码过程中，需要设置指针识别当前帧是否为独立编码帧。其示意图如图 3.9 所示。

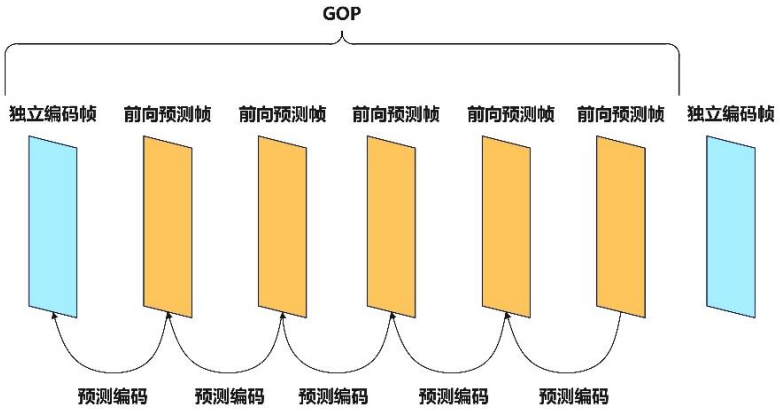


图 3.9 GOP 示意图

如图 3.9 所示，视频压缩算法流程中有两个数据流向，分别为前向的图像编码路径和反向的图像重构路径。在编码过程中，若当前帧为独立编码帧时，在前向编码路径中对原始图像进行变换、量化和熵编码得到输出的二进制码流。同时在反向重构路径中将编码得到的码流经过反量化和反变换后重构图像，存入帧存储器中，作为下一帧编码时的参考帧。若当前帧为前向参考帧时，首先结合参考帧图像采用改进后的运动估计算法得到预测帧和运动向量，再用当前帧减去预测帧得到残差图像，将残差图像进行变换、量化和熵编码，得到输出的二进制码流。在反向重构路径中也使用同样的方法重构图像，作为下一帧编码时的参考帧。

## 3.3 本章小结

本章第一节详细介绍了帧间差分法和块匹配法，阐述了其原理以及具体实现方法；并结合这两种方法，介绍了一种新型运动估计算法，该算法通过减少块匹配法的调用次数，从而在压缩效率没有太大改变的基础上，大大减少了算法时间复杂度，从而增加了算法的运行速度。本章第二节将这种算法与视频图像压缩算法的其他模块进行融合，形成了一种基于块的混合编码方法，本节主要介绍了这种方法的框架与具体实现流程。