## 一、概述

H.264，又称为MPEG-4第10部分，高级视频编码（AVC），也称为H.264或MPEG-4第10部分，高级视频编码（MPEG-4 AVC），是基于面向块，运动补偿的整数DCT编码的视频压缩标准。

## 视频为什么需要进行编码压缩：

一张为720x480的图像，用YUV420P的格式来表示，其大小为：7204801.5 约等于0.5MB如果是25帧，10分钟的数据量 0.5M1060\*25 = 7500MB -> 7GB多

视频编码压缩的目的是降低视频数据大小，方便存储和传输。

## 为什么压缩的原始数据采用YUV格式：

## 视频编码是对一张张图像来进行的。我们知道彩色图像的格式是 RGB 的，但RGB 三个颜色是有相关性的。

## 采用YUV格式，利用人对图像的感觉的生理特性，对于亮度信息比较敏感，而对于色度信息不太敏感，所以视频编码是将Y分量和UV分量分开来编码的，并且可以减少UV分量。

视频压缩原理

编码的目的是为了压缩，各种视频编码算法都是为了让视频体积变得更小，减少对存储空间和传输带宽的占用。

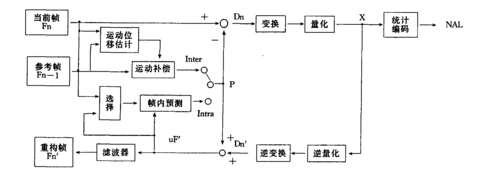
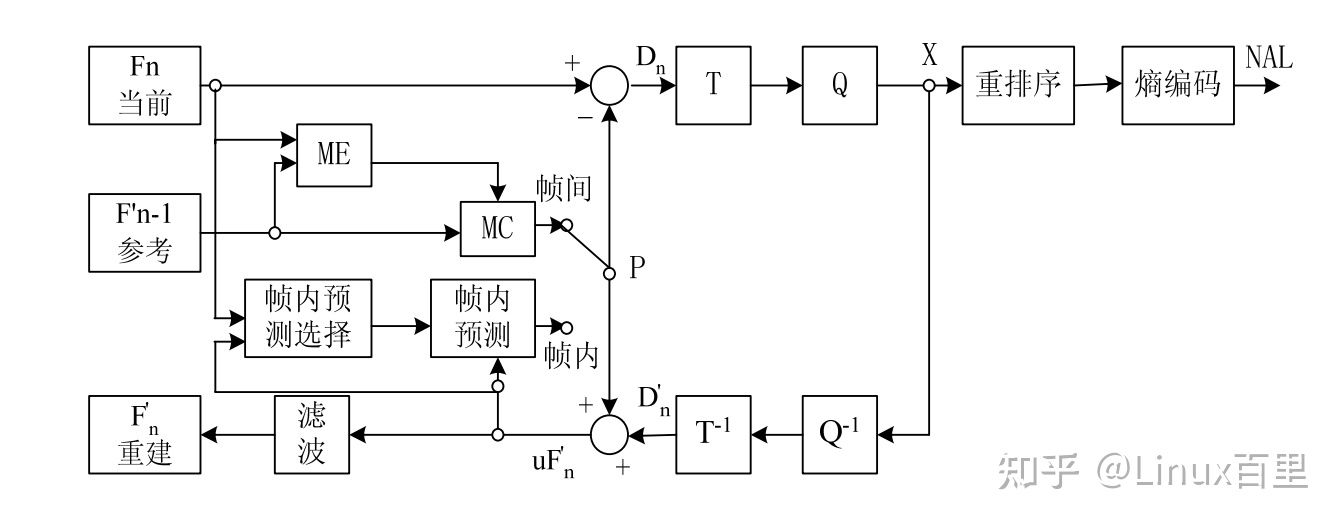
编码的核心是去除冗余信息，通过以下几种冗余来达到压缩视频的目的：

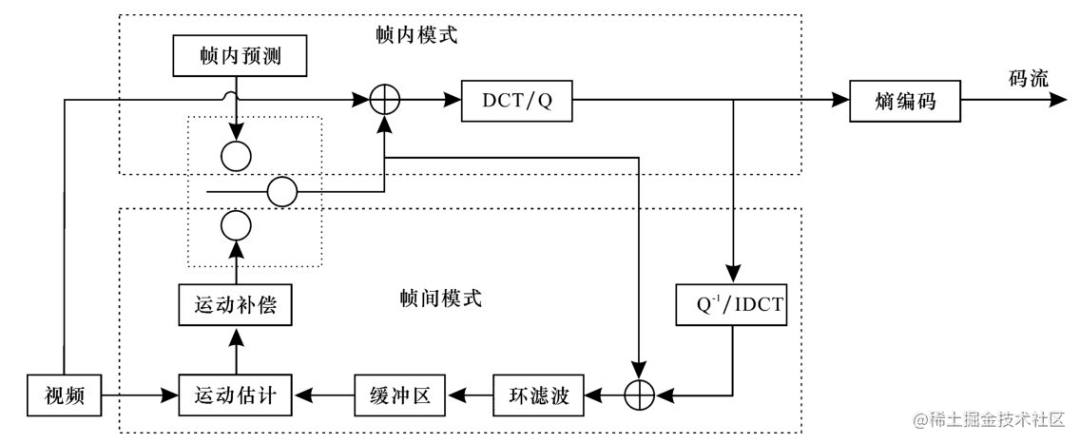
1. 空间冗余：图像相邻像素之间有较强的相关性，比如一帧图像划分成多个 16x16 的块之后，相邻的块很多时候都有比较明显的相似性。
2. 时间冗余：视频序列的相邻前后帧图像之间内容相似，比如帧率为 25fps 的视频中前后两帧图像相差只有 40ms，前后两张图像的变化较小，相似性很高。
3. 视觉冗余：我们的眼睛对某些细节不敏感，对图像中高频信息的敏感度小于低频信息的。可以去除图像中的一些高频信息，人眼看起来跟不去除高频信息差别不大（有损压缩）。
4. 编码冗余（信息熵冗余）：一幅图像中不同像素出现的概率是不同的。对出现次数比较多的像素，用少的位数来编码。对出现次数比较少的像素，用多的位数来编码，能够减少编码的大小。比如哈夫曼编码。

H264视频压缩算法现在无疑是所有视频压缩技术中使用最广泛，最流行的。

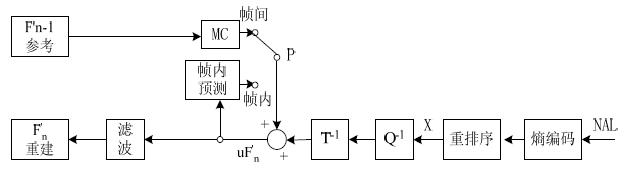
随着 x264/openh264以及[ffmpeg](https://so.csdn.net/so/search?q=ffmpeg&spm=1001.2101.3001.7020" \t "_blank)等开源库的推出，大多数使用者无需再对H264的细节做过多的研究，这大降低了人们使用H264的成本。







解码器：



H264 编码器采用的是变换和预测的混合编码法。

输入的帧或场 Fn 以宏块为单位被编码器处理。首先，按帧内或帧间预测编码的方法进行处理。

如果采用帧间预测编码，其预测值 PRED（图中用 P 表示）是由当前片中前面已编码的参考图像经运动补偿（ MC）后得出，其中参考图像用 F’n-1 表示。为了提高预测精度，从而提高压缩比，实际的参考图像可在过去或未来（指显示次序上）已编码解码重建和滤波的帧中进行选择。

预测值 PRED 和当前块相减后，产生一个残差块 Dn，经块变换、量化后产生一组量化后的变换系数 X，再经熵编码，与解码所需的一些边信息（如预测模式量化参数、运动矢量等）一起组成一个压缩后的码流，经 NAL（网络自适应层）供传输和存储用。

正如上述，为了提供进一步预测用的参考图像，编码器必须有重建图像的功能。

因此必须使残差图像经反量化、反变换后得到的Dn’与预测值P 相加，得到uFn’（未经滤波的帧）。为了去除编码解码环路中产生的噪声，为了提高参考帧的图像质量，从而提高压缩图像性能，设置了一个环路滤波器，滤波后的输出Fn’即重建图像可用作参考图像。

由编码器的NAL输出一个压缩后的H.264压缩比特流。  
经熵解码得到量化后的一组变换系数X，再经反量化、反变换，得到残差Dn’。利用从该比特流中解码出的头信息，解码器就产生一个预测块PRED，它和编码器中的原始PRED是相同的。当该解码器产生的PRED与残差Dn’相加后，就产生uFu’，再经滤波后，最后就得到滤波后的Fn’，这个Fn’就是最后的解码输出图像。

H264 压缩技术主要采用了以下几种方法对视频数据进行压缩：

* 帧内预测压缩，解决的是空域数据冗余问题。
* 帧间预测压缩（运动估计与补偿），解决的是时域数据冗徐问题。
* 整数离散余弦变换（DCT），将空间上的相关性变为频域上无关的数据，然后进行量化。
* CABAC 熵编码， 对量化后的系数进一步的压缩。

## 二、划分宏块

通过摄像头采集到的视频帧（按每秒 30 帧算），被送到 H264 编码器的缓冲区中。

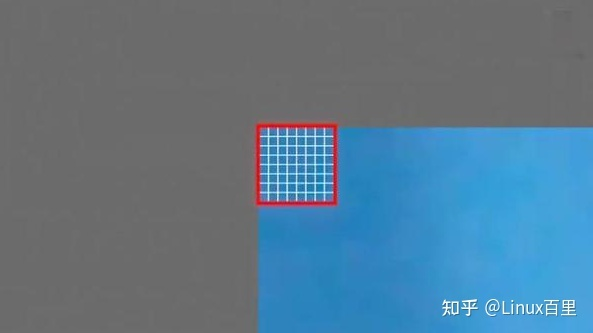
对于每一帧图像，是划分为一个个块进行编码，就是我们说的宏块。

宏块大小一般是： 16x16（H264、VP8），32x32（H265、VP9），64x64（H265、VP9、AV1），128x128（AV1）

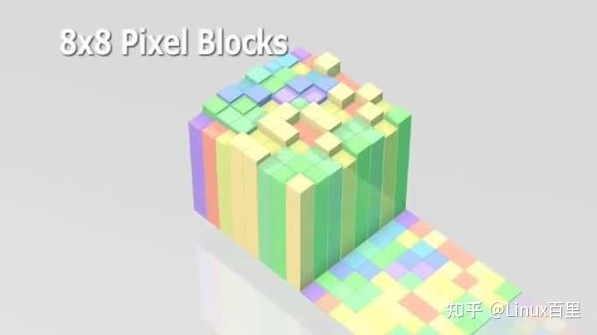
1. 编码器先要为每一幅图片划分宏块。

宏块是编码标准的基本处理单元，通常它的大小也为 16x16 像素。16X16 的宏块上可以划分出更小的子块。子块的大小可以是 8X16､ 16X8､ 8X8､ 4X8､ 8X4､ 4X4。

H264 默认是使用 16X16 大小的区域作为一个宏块，也可以划分成 8X8 大小的宏块。



1. 划分好宏块后，计算宏块的象素值。

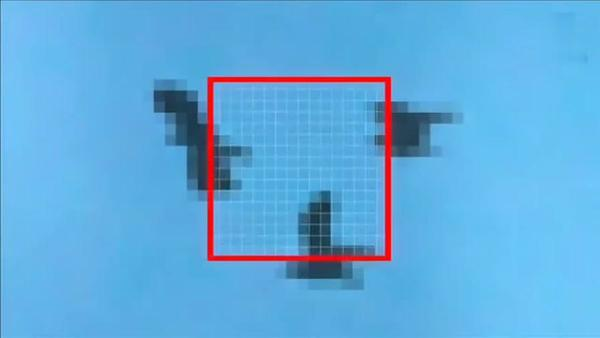


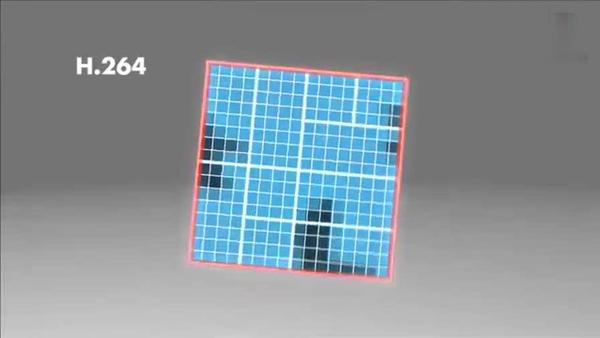


1. 划分子块

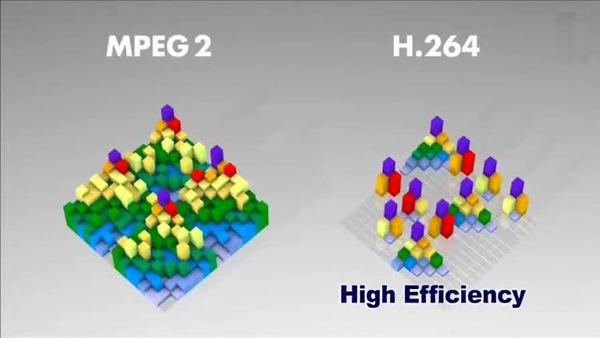
H264 对比较平坦的图像使用 16X16 大小的宏块。但为了更高的压缩率，还可以在 16X16 的宏块上更划分出更小的子块。

子块的大小可以是 8X16､ 16X8､ 8X8､ 4X8､ 8X4､ 4X4，非常的灵活。





这样再经过帧内压缩，可以得到更高效的数据。下图是分别使用mpeg-2和H264对上面宏块进行压缩后的结果。其中左半部分为MPEG-2子块划分后压缩的结果，右半部分为H264的子块划压缩后的结果，可以看出H264的划分方法更具优势。



1. 宏块扫描

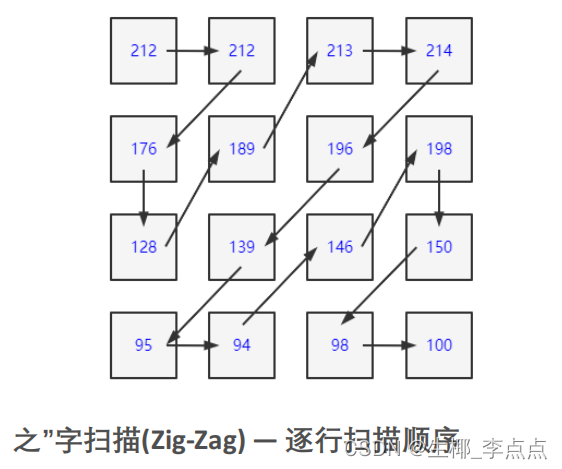
对于一个 YUV 图像，可以把划分成一个个 16x16 的宏块（以 H264 为例），Y、U、V 分量的大小分别是 16x16、8x8、8x8。

这里我们只对 Y 分量进行分析（U、V 分量同理）。

假设 Y 分量这16x16 个像素就是一个个数字，采用“之”字方式扫描每一个像素值，则可以得到一个“像素串”。

压缩的目的是使得编码器当前的字符出现连续相同的字符，比如1,1,1,1,1,1,1,1，我们可以描述为8个。

数字越小越容易用更少的bit做压缩，比如一连串数字很小（比如 0，1，2，1，0）的“像素串” ，因为 0 在二进制中只占 1 个位，2只占2个位即可。



## 三、帧间预测

帧间预测中所需要的两个主要技术就是运动估计和运动补偿。

H.264帧间预测是利用已编码视频帧/场和基于块的运动补偿的预测模式。与以往标准帧间预测的区别在于块尺寸范围更广（从16×16 到4×4）、图像素运动矢量的使用（亮度采用1/4 像素精度MV）及多参考帧的运用等等。

我们在前面已经编码完成的图像中，循环遍历每一个块，将它作为预测块，用当前的编码块与这个块做差值，得到残差块，取残差块中像素值的绝对值加起来最小的块为预测块，预测块所在的已经编码的图像称为参考帧。预测块在参考帧中的坐标值 (x0, y0) 与编码块在编码帧中的坐标值 (x1, y1) 的差值 (x0 - x1, y0 - y1) 称之为运动矢量。

而在参考帧中去寻找预测块的过程称之为运动搜索。事实上编码过程中真正的运动搜索不是一个个块去遍历寻找的，而是有快速的运动搜索算法的。

通过预测得到的残差块的像素值相比编码块的像素值，去除了大部分空间冗余信息和时间冗余信息，这样得到的像素值更小。如果把这个残差块做扫描得到的像素串送去做行程编码，是不是相比直接拿编码块的像素串去做编码更有可能得到更大的压缩率？

1. 帧分组

宏块划分好后，就可以对H264编码器缓存中的所有图片进行分组了。

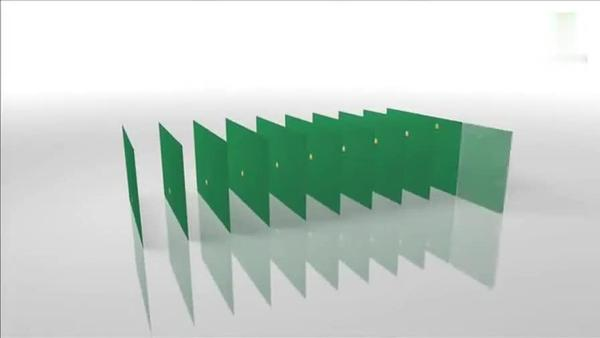
对于视频数据主要有两类数据冗余，一类是时间上的数据冗余，另一类是空间上的数据冗余。其中时间上的数据冗余是最大的。

为什么说时间上的冗余是最大的呢？假设摄像头每秒抓取30帧，这30帧的数据大部分情况下都是相关联的。也有可能不止30帧的的数据，可能几十帧，上百帧的数据都是关联特别密切的。

为了达到相关帧通过预测的方法来压缩数据，就需要将视频帧进行分组。

那么如何判定某些帧关系密切，可以划为一组呢？我们来看一下例子，下面是捕获的一组运动的台球的视频帧，台球从右上角滚到了左下角。





H264编码器会按顺序，每次取出两幅相邻的帧进行宏块比较，计算两帧的相似度。



通过宏块扫描与宏块搜索可以发现这两个帧的关联度是非常高的。进而发现这一组帧的关联度都是非常高的。因此，上面这几帧就可以划分为一组。

其算法是：在相邻几幅图像画面中，一般有差别的像素只有10%以内的点,亮度差值变化不超过2%，而色度差值的变化只有1%以内，我们认为这样的图可以分到一组。

在这样一组帧中，经过编码后，我们只保留第一帖的完整数据，其它帧都通过参考上一帧计算出来。我们称第一帧为IDR/I帧，其它帧我们称为P/B帧，这样编码后的数据帧组我们称为GOP。

1. 运动估计

基本思想是将图像序列的每一帧分成许多互不重叠的宏块，并认为宏块内所有象素的位移量都相同，然后对每个宏块到参考帧某一给定特定搜索范围内根据一定的匹配准则找出与当前块最相似的块，即匹配块，匹配块与当前块的相对位移即为运动矢量。 视频压缩的时候，只需保存运动矢量和残差数据就可以完全恢复出当前块。得到运动矢量的过程被称为运动估计。

运动矢量和经过运动匹配后得到的预测误差共同发送到解码端，在解码端按照运动矢量指明的位置，从已经解码的邻近参考帧图像中找到相应的块或宏块，和预测误差相加后就得到了块或宏块在当前帧中的位置。

实际应用时，只将运动矢量及最佳匹配块与当前块之间的差值块一起编码传输。在接收端，通过运动矢量在已经恢复的相邻帧中找到当前块的最佳匹配块，并与接收到的差值块相加恢复出当前块，这就是运动补偿基本过程。

运动估计算法是视频压缩编码的核心算法之一。高质量的运动估计算法是高效视频编码的前提和基础。其中块匹配法（BMA, Block Match Algorithm）由于算法简单和易于硬件实现，被广泛应用于各视频编码标准中。

块匹配法的基本思想是先将图像划分为许多子块，然后对当前帧中的每一块根据一定的匹配准则在相邻帧中找出当前块的匹配块，由此得到两者的相对位移，即当前块的运动矢量。

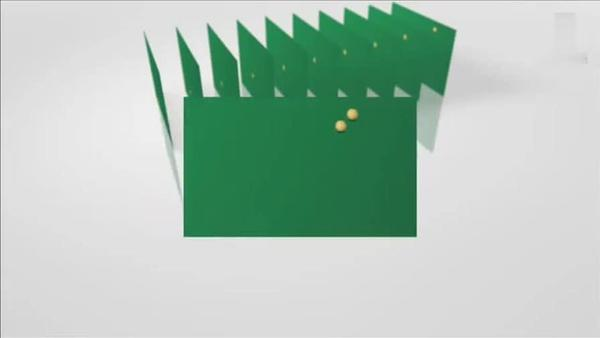
在H.264标准的搜索算法中，图像序列的当前帧被划分成互不重叠16×16大小的子块，而每个子块又可划分成更小的子块，当前子块按一定的块匹配准则在参考帧中对应位置的一定搜索范围内寻找最佳匹配块，由此得到运动矢量和匹配误差。运动估计的估计精度和运算复杂度取决于搜索策略和块匹配准则。

这里使用H.264推荐算法UMHexagonS（Unsymmetrical-cross Multi-Hexagon-grid Search）作为DSP实现的算法参考，与FS算法比较，它在保证可靠搜索精度的前提下大幅降低搜索复杂度。同时使用绝对差和（SAD, the Sum of Absolute Difference）标准作为匹配准则，它具有便于硬件实现的优点。

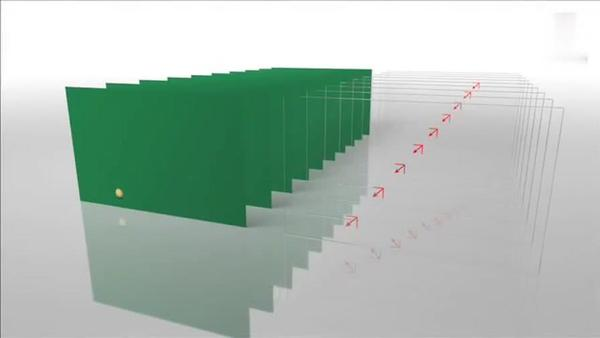
在H264编码器中将帧分组后，就要计算帧组内物体的运动矢量了。

H264编码器首先按顺序从缓冲区头部取出两帧视频数据，然后进行宏块扫描。当发现其中一幅图片中有物体时，就在另一幅图的邻近位置（搜索窗口中）进行搜索。如果此时在另一幅图中找到该物体，那么就可以计算出物体的运动矢量了。

下面这幅图就是搜索后的台球移动的位置。

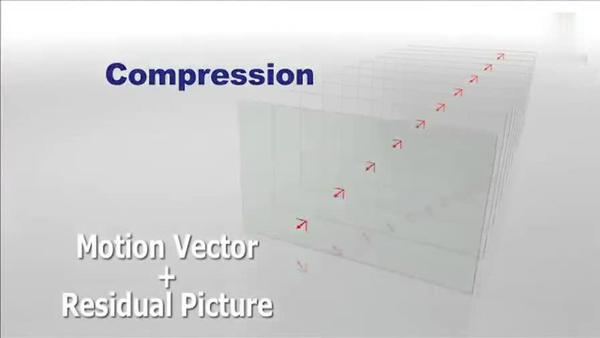


通过上图中台球位置相差，就可以计算出台图运行的方向和距离。H264依次把每一帧中球移动的距离和方向都记录下来就成了下面的样子。

运动矢量计算出来后，将相同部分（也就是绿色部分）减去，就得到了补偿数据。

我们最终只需要将补偿数据进行压缩保存，以后在解码时就可以恢复原图了。

压缩补偿后的数据只需要记录很少的一点数据。



1. 运动补偿

运动补偿是通过先前的局部图像来预测、补偿当前的局部图像，它是减少帧序列冗余信息的有效方法。

运动补偿的基本原理是，当编码器对图像序列中地第N帧进行处理时，利用运动补偿中地核心技术－运动估值ME（Motion Estimation），得到第N帧得预测帧N´。在实际编码传输时，并不总时传输第N帧，而是第N帧和其预测帧N´得差值△。

如果运动估计十分有效，△中得概率基本上分布在零附近，从而导致△比原始图像第N帧得能量小得多，编码传输△所需得比特数也就少得多。

运动补偿的步骤如下：

1.图像分割为静止得和运动的两部分，估计物体得位移向量（位移值）。

2.按照估计得到的位移向量取得前一帧的图像数据。

3.通过使用预测滤波器，得到前一帧图像数据的预测像素。

最后得到残差还需要DCT一下。

包括全局运动补偿和分块运动补偿两类。

运动补偿是一种描述相邻帧(相邻在这里表示在编码关系上相邻，在播放顺序上两帧未必相邻) 差别的方法，具体来说是描述前面一帧(相邻在这里表示在编码关系上的前面，在播放顺序上未必在当前帧前面)的每个小块怎样移动到当前帧中的某个位置去。这种方法经常被视频压缩/视频编解码器用来减少视频序列中的空域冗余。它也可以用来进行去交织(deinterlacing)的操作。

最早的运动补偿的设计只是简单的从当前帧中减去参考帧，从而得到通常含有较少能量(或者成为信息)的"残差"，从而可以用较低的码率进行编码。

稍微复杂一点的设计是估计一下整帧场景的移动和场景中物体的移动，并将这些运动通过一定的参数编码到码流中去。这样预测帧上的像素值就是由参考帧上具有一定位移的相应像素值而生成的。这样的方法比简单的相减可以获得能量更小的残差，从而获得更好的压缩比。

I帧、P帧、B帧的运动补偿：

通常，图像帧是一组一组进行处理的。每组的第一帧(通常是第一帧)在编码的时候不使用运动估计的办法，这种帧称为帧内编码帧(Intra frame)或者I帧。该组中的其它帧使用帧间编码帧(Inter frame)，通常是P帧。这种编码方式通常被称为IPPPP，表示编码的时候第一帧是I帧，其它帧是P帧。

在进行预测的时候，不仅仅可以从过去的帧来预测当前帧，还可以使用未来的帧来预测当前帧。当然在编码的时候，未来的帧必须比当前帧更早的编码，也就是说，编码的顺序和播放的顺序是不同的。通常这样的当前帧是使用过去和未来的I帧或者P帧同时进行预测，被称为双向预测帧，即B帧。这种编码方式的编码顺序的一个例子为 IBBPBBPBBPBB。

全局运动补偿：

在全局运动补偿中，运动模型基本上就是反映摄像机的各种运动，包括平移，旋转，变焦等等。这种模型特别适合对没有运动物体的静止场景的编码。全局运动补偿有下面的一些优点：

a.该模型仅仅使用少数的参数对全局的运行进行描述，参数所占用的码率基本上可以忽略不计。

b.该方法不对帧进行分区编码，这避免了分区造成的块效应。

c.在时间方向的一条直线的点如果在空间方向具有相等的间隔，就对应了在实际空间中连续移动的点。其它的运动估计算法通常会在时间方向引入非连续性。

但缺点是，如果场景中有运动物体的话，全局运动补偿就不足以表示了。这时候应该选用其它的方法。

分块运动补偿：

在分块运动补偿中，每帧被分为若干像素块 (在大多数视频编码标准，如MPEG中，是分为16x16的像素块)。从参考帧的某个位置的等大小的块对当前块进行预测，预测的过程中只有平移，平移的大小被称为运动矢量。

对分块运动补偿来说，运动矢量是模型的必要参数，必须一起编码加入码流中。由于运动矢量之间并不是独立的(例如属于同一个运动物体的相邻两块通常运动的相关性很大)，通常使用差分编码来降低码率。这意味着在相邻的运动矢量编码之前对它们作差，只对差分的部分进行编码。使用熵编码对运动矢量的成分进行编码可以进一步消除运动矢量的统计冗余(通常运动矢量的差分集中于0矢量附近)。

运动矢量的值可以是非整数的，此时的运动补偿被称为亚像素精度的运动补偿。这是通过对参考帧像素值进行亚像素级插值，而后进行运动补偿做到的。最简单的亚像素精度运动补偿使用半像素精度，也有使用1/4像素和1/8像素精度的运动补偿算法。更高的亚像素精度可以提高运动补偿的精确度，但是大量的插值操作大大增加了计算复杂度。

分块运动补偿的一个大缺点在于在块之间引入的非连续性，通常称为块效应。当块效应严重时，解码图像看起来会有像马赛克一样的效果，严重影响视觉质量。另外一个缺点是，当高频分量较大时，会引起振铃效应。

运动补偿的原理与步骤：

运动补偿的基本原理是，当编码器对图像序列中地第N帧进行处理时，利用运动补偿中地核心技术－运动估值ME（Motion Estimation），得到第N帧得预测帧N´。在实际编码传输时，并不总时传输第N帧，而是第N帧和其预测帧N´得差值△。如果运动估计十分有效，△中得概率基本上分布在零附近，从而导致△比原始图像第N帧得能量小得多，编码传输△所需得比特数也就少得多。

运动补偿的步骤如下：

a.图像分割为静止得和运动的两部分，估计物体得位移向量（位移值）。

b.按照估计得到的位移向量取得前一帧的图像数据。

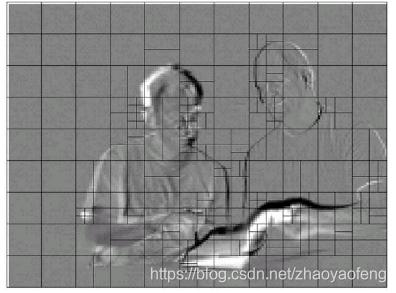
c.通过使用预测滤波器，得到前一帧图像数据的预测像素。

树状结构运动补偿：

每个宏块（16×16 像素）可以4种方式分割： 一个16×16，两个16×8，两个8×16，四个8×8。其运动补偿也相应有四种。  
而8×8 模式的每个子宏块还可以四种方式分割：一个8×8，两个4×8 或 两个8×4 及4 个4×4。

这些分割和子宏块大大提高了各宏块之间的关联性。这种分割下的运动补偿则称为树状结构运动补偿。

每个分割或子宏块都有一个独立的运动补偿。每个MV必须被编码、传输，分割的选择也需编码到压缩比特流中。对大的分割尺寸而言，MV选择和分割类型只需少量的比特，但运动补偿残差在多细节区域能量将非常高。小尺寸分割运动补偿残差能量低，但需要较多的比特表征MV和分割选择。分割尺寸的选择影响了压缩性能。整体而言，大的分割尺寸适合平坦区域，而小尺寸适合多细节区域。  
H.264 编码器为帧的每个部分选择了最佳分割尺寸，使传输信息量最小，并将选择的分割加到残差帧上。  
在帧变化小的区域（残差显示灰色），选择16×16分割；多运动区域（残差显示黑色或白色），选择更有效的小的尺寸。



运动矢量：

帧间编码宏块的每个分割或者子宏块都是从参考图像某一相同尺寸区域预测而得。两者之间的差异（MV）对亮度成分采用1/4 像素精度，色度1/8 像素精度。图像素位置的亮度和色度像素并不存在于参考图像中，需利用邻近已编码点进行内插而得。

MV预测：

每个分割MV的编码需要相当数目的比特，特别是使用小分割尺寸时。  
为减少传输比特数，可利用邻近分割的MV较强的相关性，MV可由邻近已编码分割的MV预测而得。  
预测矢量MVp基于已计算MV和MVD（预测与当前的差异）并被编码和传送。MVp则取决于运动补偿尺寸和邻近MV的有无。

## 四、帧内预测

在帧内预测模式中，预测块P是基于已编码重建块和当前块形成的。  
对亮度像素而言，P块用于4×4子块或者16×16宏块的相关操作。  
4×4亮度子块有9种可选预测模式，独立预测每一个4×4亮度子块，适用于带有大量细节的图像编码；16×16亮度块有4种预测模式，预测整个16×16亮度块，适用于平坦区域图像编码；色度块也有4种预测模式，类似于16×16亮度块预测模式。

编码器通常选择使P块和编码块之间差异最小的预测模式。

此外，还有一种帧内编码模式称为I\_PCM 编码模式。  
该模式下，编码器直接传输图像像素值，而不经过预测和变换。  
在一些特殊的情况下，特别是图像内容不规则或者量化参数非常低时该模式比起“常规操作”（帧内预测—变换—量化—熵编码）效率更高。

I\_PCM 模式用于以下目的：  
允许编码器精确的表示像素值。  
提供表示不规则图像内容的准确值，而不引起重大的数据量增加。  
严格限制宏块解码比特数，但不损害编码效率。

在以往H.263+、MPEG-4 等视频压缩编码标准中，帧内编码被引入变换域。H.264帧内编码则参考预测块左方或者上方的已编码块的邻近像素点，被引入空间域。但是，如果参考预测块是帧间编码宏块，该预测会因参考块的运动补偿引起误码扩散。所以，参考块通常选取帧内编码的邻近块。

包括帧内预测的所有预测都不能跨片边界预测，每片必须独立编解码。

帧内预测以绝对误差和（SAE）为标准选取最佳预测模式，使预测帧更加接近原始帧，减少了相互间的差异，去除时间上的数据冗余，提高了编码的压缩效率。帧内预测中，块或宏块利用已编码并重建的块作为参考，进行预测。具体编程时，编码器通过计算并比较各种模式下的SAE，选取SAE 值最小的模式作为最佳预测模式，并将该模式信息化，同时传送至解码端，以供正确解码。

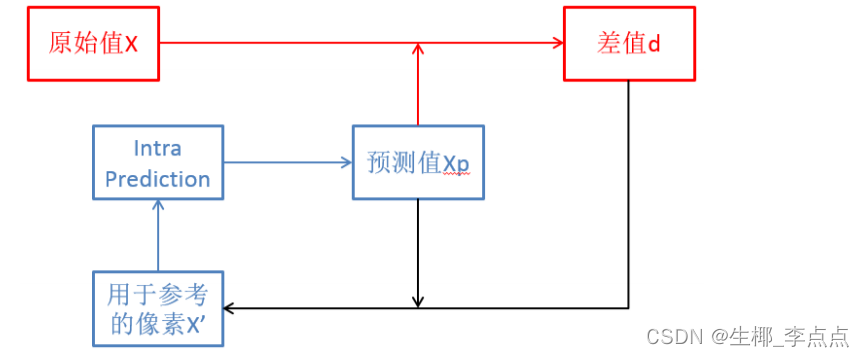
我们把运动矢量与补偿称为帧间压缩技术，它解决的是视频帧在时间上的数据冗余。

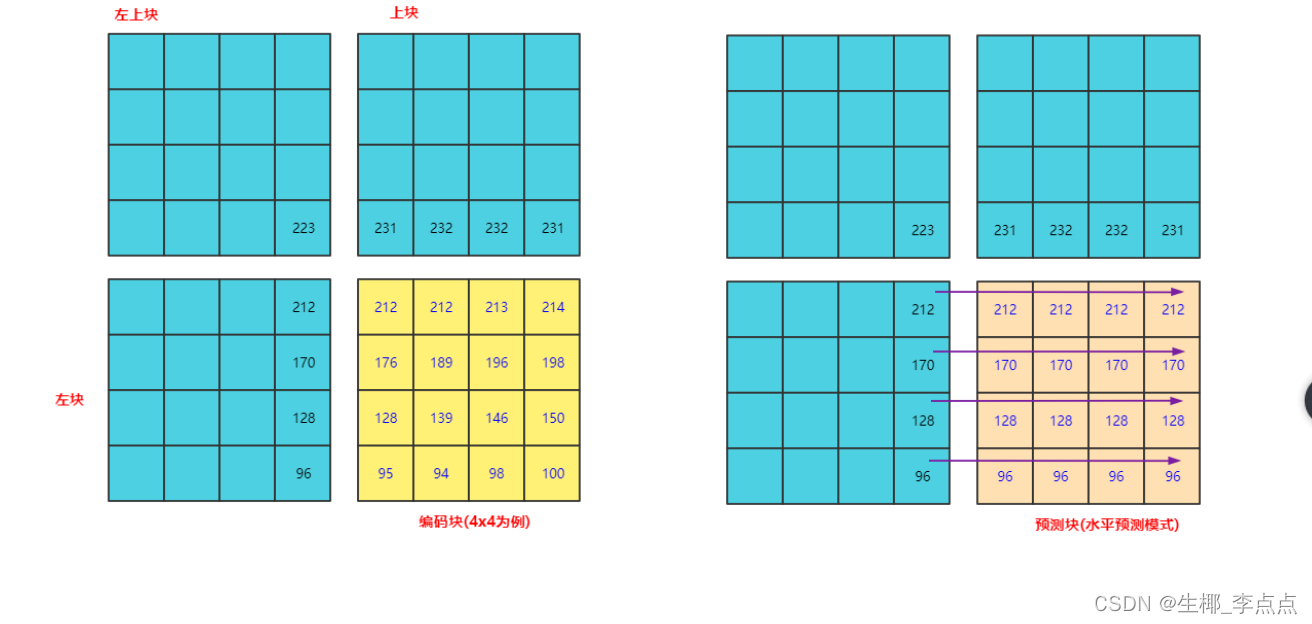
除了帧间压缩，帧内也要进行数据压缩，帧内数据压缩解决的是空间上的数据冗余。

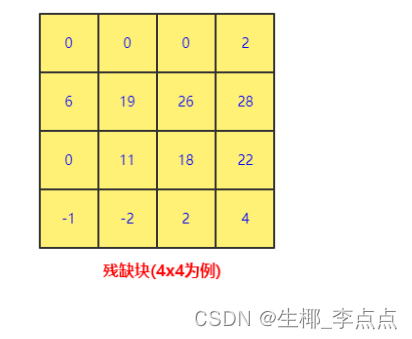
人眼对图象都有一个识别度，对低频的亮度很敏感，对高频的亮度不太敏感。所以基于一些研究，可以将一幅图像中人眼不敏感的数据去除掉。这样就提出了帧内预测技术。

步骤：

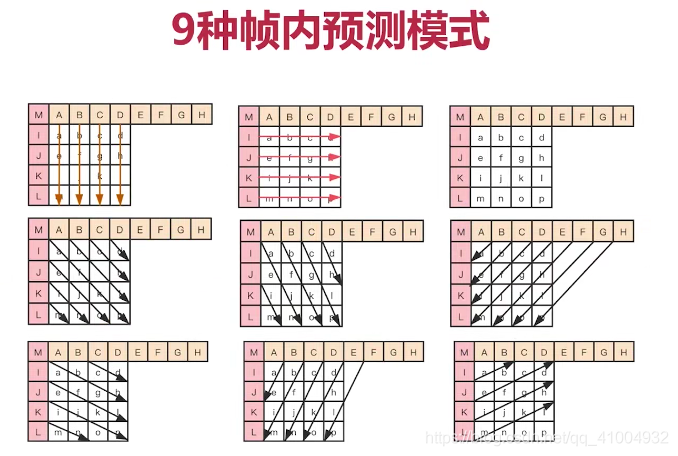
1. 帧内预测就是在当前编码图像内部已经编码完成的块中找到与将要编码的块相邻的块。一般就是即将编码块的左边块、上边块、左上角块和右上角块，通过将这些块与编码块相邻的像素经过多种不同的算法得到多个不同的预测块。
2. 然后我们再用编码块减去每一个预测块得到一个个残差块。最后，我们取这些算法得到的残差块中像素的绝对值加起来最小的块为预测块。而得到这个预测块的算法为帧内预测模式。



此时我们把原始编码块的像素值减去预测块的像素值得到残缺块，可以看到对应的数值更为接近0。  
原始值-预测值=残缺块（预测算法 ：水平预测 垂直预测）



H264的帧内压缩与JPEG很相似。一幅图像被划分好宏块后，对每个宏块可以进行 9 种模式的预测。找出与原图最接近的一种预测模式。



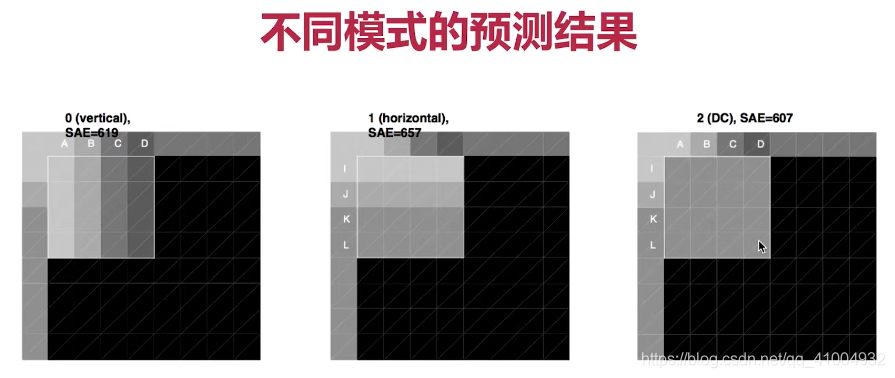
第一种模式是纵列的即垂直模式，左边以及上边都是已经推测出来的宏块，这个时候就可以将下面4x4的宏块预测出来，对于垂直模式来说，上边A对应下面的一整列都是A，第二个B，对应的一列下面也都是B，以此类推。

第二种模式是横向的即水平模式，还是左边以及·上边都是预测出来的数据，预测结果第一行是I，第二行是J，以此类推。

第三种模式是求平均值，在目标的4x4的宏块内都是ABCD加上IJKL的求平均值，例如宏块内的小a，他的值就是ABCD加IJKL求平均值得出，也就是下面每一个4x4内的像素值都是一样的。

需要注意的是：像素a对应的值是(ABCD+IJKL)/2,而不是(A+I)/2.

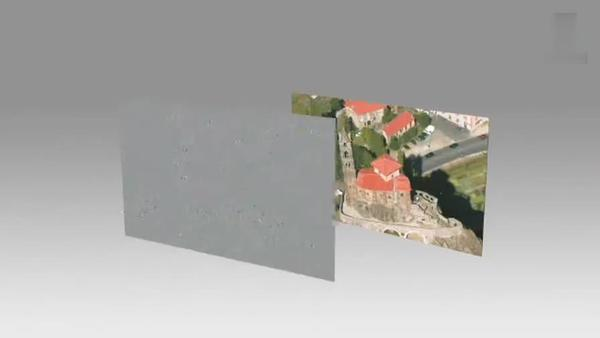
前三种的预测结果如下图所示：

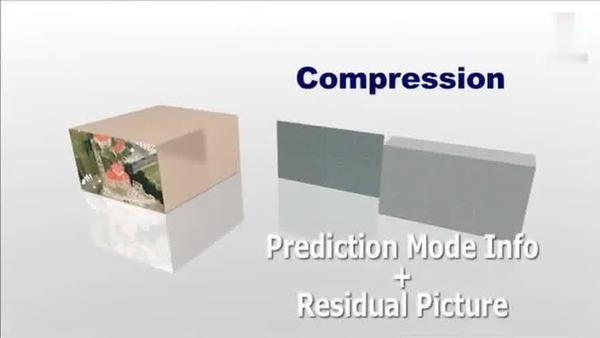






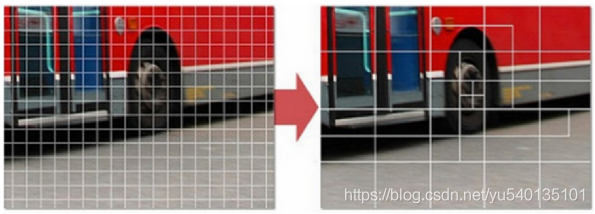
将原始图像与帧内预测后的图像相减得残差值：

再将我们之前得到的预测模式信息一起保存起来，这样我们就可以在解码时恢复原图了。效果如下：



经过帧内与帧间的压缩后，虽然数据有大幅减少，但还有优化的空间。

H.264和H.265的宏块对比。在H.264中的宏块大小是固定的16x16，在H.265中宏块的大小是可变的，最小8x8最大64x64。这种方案有几个优点：如图所示，对于颜色变化不明显的区域，如车体的红色区域和地面的灰色区域，则会使用大宏块进行划分，可以更大的压缩画面；对于颜色变化比较多的地方，例如汽车的轮胎附近，则可以更精细的划分成更小的宏块，更小的宏块虽然占用了更多的空间，但是会使画面精细。显然H.265的方案更加灵活和智能，有针对性、有重点的进行帧内编码，降低整体的存储空间。



## 五、DCT （整数离散余弦变换）变换和量化（傅里叶变换）

变换编码将图像时域信号变换成频域信号，在频域中图像信号能量大部分集中在低频区域，相对时域信号，码率有较大的下降。

H.264 对图像或预测残差采用了4×4 整数离散余弦变换技术，避免了以往标准中使用的通用8×8 离散余弦变换逆变换经常出现的失配问题。量化过程根据图像的动态范围大小确定量化参数，既保留图像必要的细节，又减少码流。

在图像编码中，变换编码和量化从原理上讲是两个独立的过程。但在H.264 中，将两个过程中的乘法合二为一，并进一步采用整数运算，减少编解码的运算量，提高图像压缩的实时性，这些措施对峰值信噪比（PSNR）的影响很小，一般低于0.02dB，可不计。

量化过程在不降低视觉效果的前提下减少图像编码长度，减少视觉恢复中不必要的信息。H.264采用标量量化技术，它将每个图像样点编码映射成较小的数值。

在量化和反量化过程中，量化步长QP决定量化器的编码压缩率及图像精度。如果QP比较大，则量化值FQ动态范围较小，其相应的编码长度较小，但反量化时损失较多的图像细节信息；如果QP比较小，则FQ动态范围较大，相应的编码长度也较大，但图像细节信息损失较少。编码器根据图像值实际动态范围自动改变QP 值，在编码长度和图像精度之间折衷，达到整体最佳效果。

在H.264 中，量化步长共有52 个值。当QP取最小值0 时代表最精细的量化，当QP取最大值51时代表最粗糙的量化。对于色度编码，一般使用与亮度编码同样的量化步长。为了避免在较高量化步长时的出现颜色量化人工效应，色度的QP最大值大约限制在亮度QP 最大值的80％范围内，亮度QP的最大值是51，而色度QP的最大值是39。

* 我们的目标不只是将像素值变小，而是希望能出现连续的 0 像素
* 这就需要利用我们人眼的视觉敏感性的特点了。我们刚才说了人眼对高频信息不太敏感。因为人眼看到的效果可能差别不大，所以我们可以去除一些高频信息。

#### DCT变换（余弦变换）：

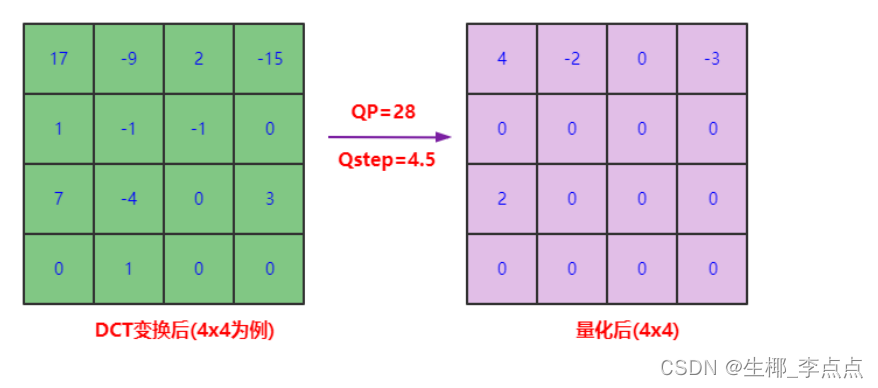
可以将残差数据做整数离散余弦变换，去掉数据的相关性，进一步压缩数据。

高频（大部分像素值）的值会变大，低频（占小部分）的会变小。

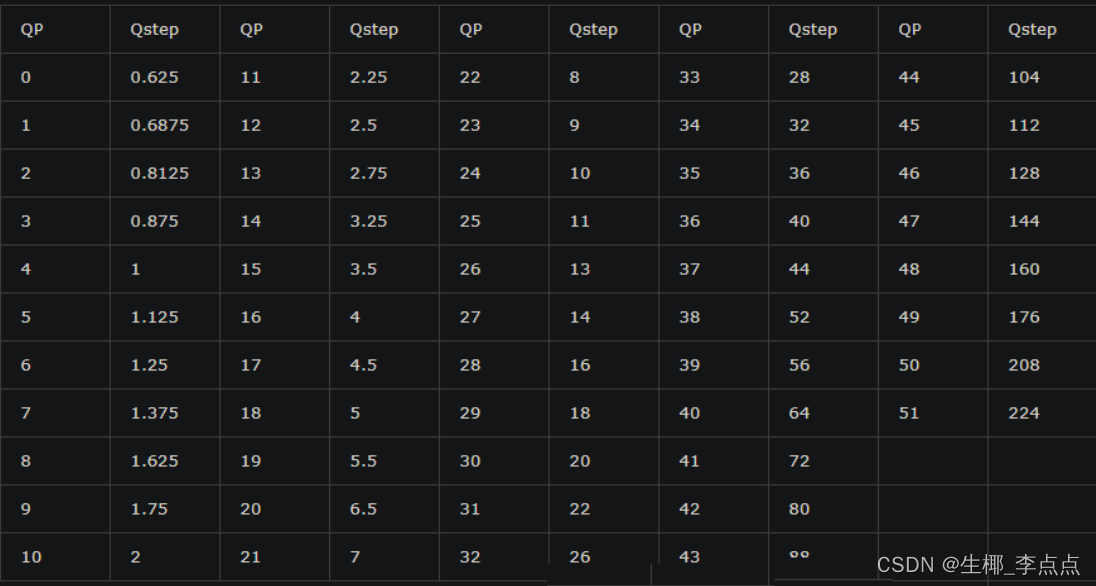
DCT变换取整数时又一定的损失：

#### 量化步长：

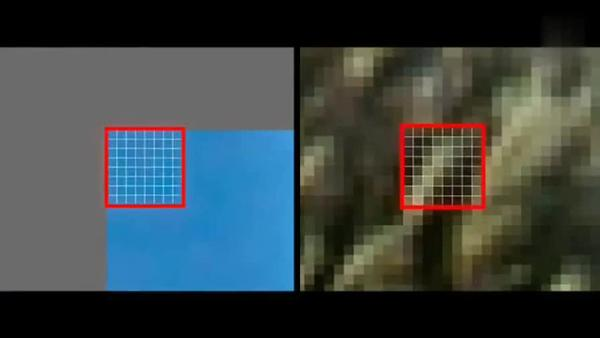
* 由于人眼对高频信息不太敏感，如果我们通过一种手段去除掉大部分高频信息，也就是将大部分高频信息置为 0，但又不太影响人的观感，是不是就可以达到我们最初的目标，即可以得到有一连串 0 的像素串？这就涉及到量化操作了。
* 我们让变换块的系数都同时除以一个值，这个值我们称之为量化步长，也就是 QStep（QStep 是编码器内部的概念，用户一般使用量化参数 QP 这个值，QP 和 QStep 得到的结果就是量化后的系数。QStep 越大，得到量化后的系数就会越小。同时，相同的 QStep 值，高频系数值相比低频系数值更小，量化后就更容易变成 0。这样一来，将大部分高频系数变成 0。如下图所示：



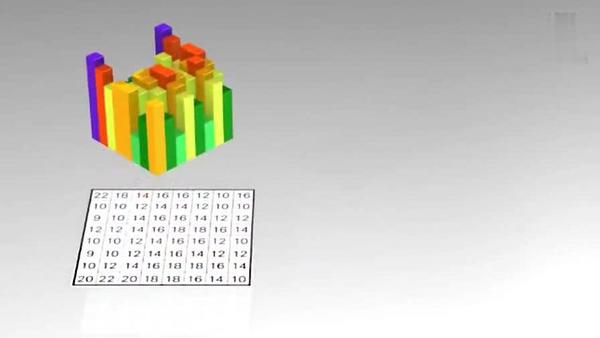
* 解码的时候，需要将 QStep 乘以量化后的系数得到变换系数，很明显这个变换系数和原始没有量化的变换系数是不一样的，这个就是常说的有损编码。
* 而到底损失多少呢？其由 QStep 来控制，QStep 越大，损失就越大。QStep 跟 QP 一一对应。从编码器应用角度来看，QP 值越大，损失就越大，从而画面的清晰度就会越低。同时，QP 值越大系数被量化成 0 的概率就越大，这样编码之后码流大小就会越小，压缩就会越高。



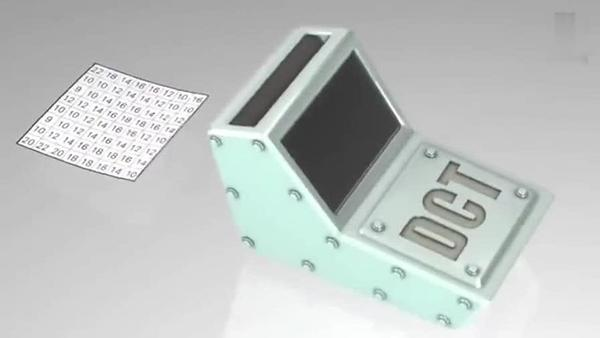
如下图所示，左侧为原数据的宏块，右侧为计算出的残差数据的宏块。



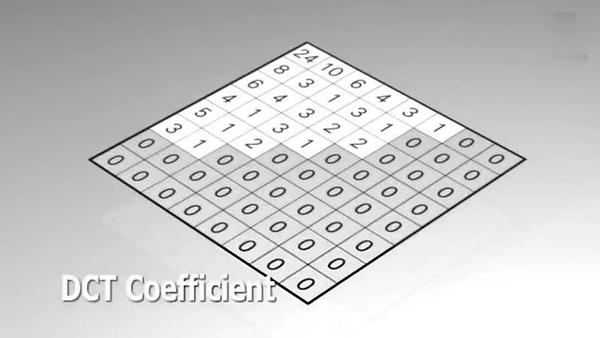
将残差数据宏块数字化后如下图所示：



将残差数据宏块进行 DCT 转换。



去掉相关联的数据后，我们可以看出数据被进一步压缩了。



### 环路滤波

产生原因：

各个块变换量化过程相对独立，因此引入的量化误差大小及其分布特性相互独立，导致相邻块边界的不连续；运动补偿中，相邻块的预测值可能来自不同图像的不同位置，导致预测残差信号在块边界产生数值不连续。

使用原因：

环路滤波处理后的重建像素更有利于参考，进一步减小后续编码像素的预测残差，有效地提高了视频的主客观质量。

#### 去方块滤波：

H.264/MPEG-4 AVC 视频编码标准中，在编解码器反变换量化后图像会出现方块效应。其产生的原因有两个。最重要的一个原因是基于块的帧内和帧间预测残差的DCT变换。变换系数的量化过程相对粗糙，因而反量化过程恢复的变换系数带有误差，会造成在图像块边界上的视觉不连续。第二个原因来自于运动补偿预测。运动补偿块可能是从不是同一帧的不同位置上的内插样点数据复制而来。因为运动补偿块的匹配不可能是绝对准确的，所以就会在复制块的边界上产生数据不连续。当然，参考帧中存在的边界不连续也被复制到需要补偿的图像块内。尽管H.264/MPEG-4 AVC采用较小的4×4变换尺寸可以降低这种不连续现象，但仍需要一个去方块滤波器以最大程度提高编码性能。

在视频编解码器中加入去方块滤波器的方法有两种：后置滤波器和环路滤波器。后置滤波器只处理编码环路外的显示缓冲器中的数据，所以它不是标准化过程中的规范内容，在标准中只是可选项。相反，环路滤波器处理编码环路中的数据。在编码器中，被滤波的图像帧作为后续编码帧运动补偿的参考帧；在解码器中，滤波后的图像输出显示。这要求所有与本标准一致的解码器采用同一个滤波器以与编码器同步。当然如果有必要，解码器也还可以在使用环路滤波器的同时使用后置滤波器。

自适应去方块滤波器利用简单的算法可靠地提高图像的主客观质量。其较好的性能是因为可靠地区分了真实的和人为的图像边界，并有效地滤除后者。在相同的PSNR 下可以节省码流超过9%，并同时明显地提高图像视觉质量。

## 六、熵编码

熵编码是无损压缩编码方法，它生成的码流可以经解码无失真地恢复出原数据。

熵编码是建立在随机过程的统计特性基础上的。

熵的大小与信源的概率模型有着密切的关系，各个符号出现的概率不同，信源的熵也不同。当信源中各事件是等概率分布时，熵具有极大值。信源的熵与其可能达到的最大值之间的差值反映了该信源所含有的冗余度。信源的冗余度越小，即每个符号所独立携带的信息量越大，那么传送相同的信息量所需要的序列长度越短，符号位越少。因此，数据压缩的一个基本的途径是去除信源的符号之间的相关性，尽可能地使序列成为无记忆的，即前一符号的出现不影响以后任何一个符号出现的概率。

当帧间压缩去掉了时间的冗余、视觉冗余（I-PCM模式），帧内压缩去掉了空间的冗余、视觉上冗余之后，得到的残差数据，这时候4x4的像素块经过变换和量化之后，低频信号集中在左上角，大量高频信号集中在右下角。

左边的低频信号相对数值较大，而右下角的大量高频信号都被量化成0、1和-1；变换量化后的残差信息有一定的统计特性和规律。

熵编码压缩是一种无损压缩，其实现原理是使用新的编码来表示输入的数据，从而达到压缩的效果。常用的熵编码有游程编码，哈夫曼编码和CAVLC编码等。

CAVLC(基于上下文自适应的可变长编码)：

在H.264的CAVLC(基于上下文自适应的可变长编码)中，通过根据已编码句法元素的情况动态调整编码中使用的码表，取得了极高的压缩比。

CAVLC用于亮度和色度残差数据的编码。残差经过变换量化后的数据表现出如下特性：4x4 块数据经过预测、变换、量化后，非零系数主要集中在低频部分，而高频系数大部分是零；量化后的数据经过zig-zag 扫描，DC系数附近的非零系数值较大，而高频位置上的非零系数值大部分是+1 和-1；相邻的4x4 块的非零系数的数目是相关的。CAVLC 充分利用残差经过整数变换、量化后数据的特性进行压缩，进一步减少数据中的冗余信息，为H.264 卓越的编码效率奠定了基础。

利用相邻已编码符号所提供的相关性，为所要编码的符号选择合适的上下文模型。利用合适的上下文模型，就可以大大降低符号间的冗余度。在CAVLC 中上下文模型的选择主要体现在两个方面, 非零系数编码所需表格的选择以及拖尾系数后缀长度的更新。

### 编码过程包括：

1.非零系数数目(TotalCoeffs)和拖尾系数数目(TrailingOnes)的编码

2.每一个拖尾系数的符号正负性编码（按照Z扫面结果的逆序编码）

3.除拖尾系数外的每一个非零系数幅值(Level，包含正负号信息)编码（按照Z扫描结果的逆序编码）

#### CABAC(基于上下文的自适应二进制算术熵编码)：

算术编码的思想是用0到1的区间上的一个数来表示一个字符输入流，它的本质是为整个输入流分配一个码字，而不是给输入流中的每个字符分别指定码字。算术编码是用区间递进的方法来为输入流寻找这个码字的，它从于第一个符号确定的初始区间（0 到1）开始，逐个字符地读入输入流，在每一个新的字符出现后递归地划分当前区间，划分的根据是各个字符的概率，将当前区间按照各个字符的概率划分成若干子区间，将当前字符对应的子2 区间取出，作为处理下一个字符时的当前区间。到处理完最后一个字符后，得到了最终区间，在最终区间中任意挑选一个数作为输出。解码器按照和编码相同的方法和步骤工作，不同的是作为逆过程，解码器每划分一个子区间就得到输入流中的一个字符。

CABAC 中内建了由大量实验统计而得到的概率模型。在编码过程中，CABAC 根据当前所要编码的内容以及先前已编码好内容，动态地选择概率模型来进行编码，并实时更新相对应的概率模型。并且，CABAC 在计算量和编码速度上进行了优化，用了量化查表、移位、逻辑运算等方法代替复杂的概率估计和乘法运算。在实际应用中，CABAC与其它主流的熵编码方式相比有更高的编码效率,用一组质量在28~40dB 的视频图像做测试，应用CABAC可使比特率进一步提高9%~14%。

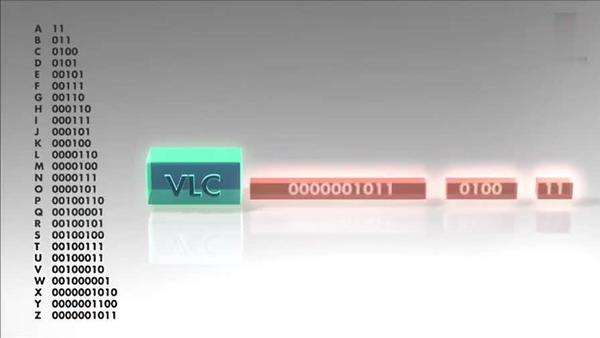
为了能够在最后熵编码的时候压缩率更高，对于送到熵编码（以行程编码为例）的“像素串”，包含的0越多，越能提高压缩率。为了达到这个目标：

* 先通过帧内预测或者帧间预测去除空间冗余和时间冗余，从而得到一个像素值相比编码块小很多的残差块。
* 然后再通过 DCT 变换将低频和高频信息分离开来得到变换块，然后再对变换块的系数做量化。
* 由于高频系数通常比较小，很容易量化为 0，同时人眼对高频信息不太敏感，这样就得到了一串含有很多个 0，大多数情况下是一串含有连续 0 的“像素串”，并且人的观感还不会太明显。这样，最后熵编码就能把图像压缩成比较小的数据，以此达到视频压缩的目的。

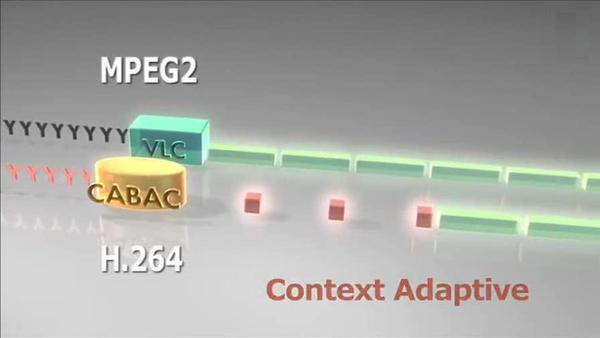
做完 DCT 后，还不够，还要进行 CABAC 进行无损压缩。

上面的帧内压缩是属于有损压缩技术。也就是说图像被压缩后，无法完全复原。而CABAC属于无损压缩技术。

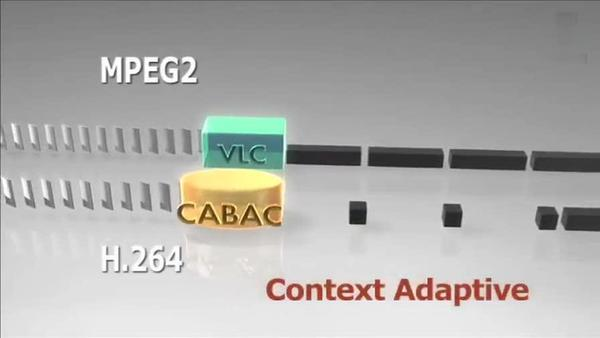
无损压缩技术大家最熟悉的可能就是哈夫曼编码了，给高频的词一个短码，给低频词一个长码从而达到数据压缩的目的。MPEG-2中使用的VLC就是这种算法，我们以 A-Z 作为例子，A属于高频数据，Z属于低频数据。看看它是如何做的。



CABAC也是给高频数据短码，给低频数据长码。同时还会根据上下文相关性进行压缩，这种方式又比VLC高效很多。其效果如下：



现在将 A-Z 换成视频帧，它就成了下面的样子。

从上面这张图中明显可以看出采用 CACBA 的无损压缩方案要比 VLC 高效的多。