(1) $D(p,q)\geqslant 0$,(当且仅当 $p=q$ 时,$D(p,q)=0$)

(2) $D(p,q)=D(q,p)$

(3) $D(p,z)\leqslant D(p,q)+D(q,z)$

则 D 是距离函数或度量。

p 和 q 间的欧氏距离定义如下

$$D_e(p,q)=[(x-s)^2+(y-t)^2]^{\frac{1}{2}} \tag{3.2.9}$$

对于距离度量,距点(x,y)的距离小于或等于某一值 r 的像素是中心在(x,y),半径为 r 的圆平面。

p 和 q 间的距离 D_d(也叫城市街区距离)如下式定义

$$D_d(p,q)=|x-s|+|y-t| \tag{3.2.10}$$

在这种情况下,距点(x,y)的 D_d 距离小于或等于某一值 r 的像素形成一个中心在(x,y)的菱形。例如,距点(x,y)的 D_4 距离小于或等于 2 的像素形成固定距离的下列轮廓。

		2		
	2	1	2	
2	1	0	1	2
	2	1	2	
		2		

具有 $D_4=1$ 的像素是(x,y)的 4 邻域。

p 和 q 间的 D_g 距离(也叫棋盘距离)定义为下式

$$D_g(p,q)=\max(|x-s|+|y-t|) \tag{3.2.11}$$

在这种情况下,距点(x,r)的 D_g 距离小于或等于某一值 r 的像素形成中心在(x,y)的方形。例如,距点(x,y)(中心点)的 D_8 距离小于或等于 2 的像素形成下列固定距离的轮廓。

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

具有 $D_8=1$ 的像素是关于(x,y)的 8 邻域。

注意:p 和 q 之间的 D_d 和 D_g 距离与任何通路无关,通路可能存在于各点之间,因为这些距离仅与点的坐标有关。然而,如果选择考虑 m 邻接,则两点间的 D_m 距离用点间最短的通路定义。在这种情况下,两像素间的距离将依赖于沿通路的像素值以及它们的邻点值。例如,考虑下列安排的像素并假设 p,p_2 和 p_4 的值为 1,p_1 和 p_3 的值为 0 或 1。

$$\begin{matrix} p_3 & p_4 \\ p_1 & p_2 \\ p & \end{matrix}$$

假设考虑值为1的像素邻接(即$V=\{1\}$)。如果p_1和p_3是0,p和p_4间最短m通路的长度(D_m距离)是2。如果p_1是1,则p_2和p将不再是m邻接(见m邻接的定义),并且m通路的长度变为3(通路通过点$pp_1p_2p_4$)。类似地,如果p_3是1(并且p_2为0),则最短的通路距离也是3。最后,p_1和p_3都为1,则p和p_4间的最短m通路长度为4,在这种情况,通路通过点$pp_1p_2p_3p_4$。

4. 基于像素的图像操作

在后续各章,大量的文献是关于像素间运算的。例如,用一幅图像去除另一幅图像。在式(3.2.9)中,图像以矩阵的形式表示,而矩阵除法无定义。因此,当提到类似用一幅图像除另一幅图像的运算时,意思是在两幅图像相应的像素间执行除法运算。例如,如果f和g是两幅图像,用g除f形成的图像的第一个像素值是用g中的第一个像素去除f中的第一个像素的结果。当然,是假设在g中没有一个像素值为0。其他的算术和逻辑操作也类似地定义为图像中相对应像素间的操作。

3.3 数字图像的文件格式与数据结构

要利用计算机对数字化图像进行处理,首先要对图像的文件格式有清楚的认识。当获取的图像经过模数转换后,得到的图像一般为裸图形式,如果想要生成目标图像文件,必须根据文件的格式做相应的处理。无论是什么设备,它总是按一定的图像文件格式来提供信息,比较常用的有BMP格式、JPEG格式和GIF格式等,所以在进行图像处理以前,首先要对图像的格式有清晰的认识,只有在此基础上才可以进行进一步的开发处理。

由于现存的图像文件格式众多,且一般存在压缩,因此只对常见的格式简要介绍,其中主要介绍BMP图像文件格式,并且文件里的图像数据是未压缩的。因为图像的数字化处理主要是对图像中的各个像素进行相应的处理,而未压缩的BMP图像中的像素数值正好与实际要处理的数字图像相对应,这种格式的文件最适合我们对之进行数字化处理。压缩过的图像是无法直接进行数字化处理的,如JPEG、GIF等格式的文件,此时首先要对图像文件解压缩,这就要涉及到一些比较复杂的压缩和解压缩算法。经过转换,可以利用得到的未压缩的BMP文件格式进行后续处理。对于JPEG、GIF等格式,由于涉及到压缩算法,这要求读者掌握一定的信息论方面的知识。限于篇幅原因,这里只作一般性的讲解,深入的了解可以参考相关书籍资料。

3.3.1 BMP图像格式

位图(bitmap,BMP)是Windows操作系统中的标准图像文件格式,能够被多种Windows应用程序所支持。随着Windows操作系统的流行与丰富的Windows应用程序的开发,BMP位图格式理所当然地被广泛应用。这种格式的特点是包含的图像信息较丰富,几乎不进行压缩,但由此导致了它占用磁盘空间过大的缺点。所以,目前BMP在单机上比较流行,而在图像通信中往往不直接使用BMP位图格式。

1. BMP 文件组成

BMP 文件由文件头、位图信息头、颜色信息和图形数据 4 部分组成，如表 3.1 所示。文件头主要包含文件的大小、文件类型、图像数据偏离文件头的长度等信息；位图信息头包含图像的尺寸信息、图像用几位数值来表示一个像素、图像是否压缩、图像所用的颜色数等信息。颜色信息包含图像所用到的颜色表，显示图像时需要用到这个颜色表来生成调色板。但如果图像为真彩色，即图像的每个像素用 24 位来表示，文件中就没有这一块信息，也就不需要操作调色板。文件中的数据块表示图像的相应像素值，需要注意的是，图像的像素值在文件中的存放顺序为从左到右，从下到上。也就是说，在 BMP 文件中首先存放的是图像的最后一行像素，最后才存储图像的第一行像素。但对于同一行像素，则是按照先左边后右边的顺序存储的，这与 3.2.5 节介绍的图像表示方法是不一样的。另外一个需要关注的细节是，文件存储图像的每一行像素值时，如果存储该行像素值所占的字节数为 4 的倍数，则正常存储；否则，需要在后端补 0，凑足 4 的倍数。

表 3.1　BMP 图像文件结构

文件部分	属　性	说　明
BITMAPFILEHEADER（位图文件头）	bfType	文件类型，必须是 0×424D，即字符串 BM
	bfSize	指定文件大小，包括 14 个字节
	bfReserved1	保留字，不用考虑
	bfReserved2	保留字，不用考虑
	bfOffBits	从文件头到实际位图数据的偏移字节数
BITMAPINFOHEADER（位图信息头）	bfSize	该结构的长度，为 40
	biWidth	图像的宽度，单位为像素
	biHeight	图像的高度，单位为像素
	biplanes	位平面数，必须是 1，不用考虑
	biBitCount	指定颜色倍数，1 为二值，4 为 16 色，8 为 256 色，16，24，32 为真彩色
	biCompression	指定是否压缩，有效值为 BI_RGB，BI_RLE8，BI_RLE4，BI_BITFIELDS
	biSizeImage	实际的位图数据占用的字节数
	biXpelsPerMeter	目标设备水平分辨率，单位是每米的像素数
	biYpelsPerMeter	目标设备垂直分辨率，单位是每米的像素数
	biClrUsed	实际使用的颜色数，若该值为 0，则使用颜色数为 2 的 biBitCount 次方种
	biClrImportant	图像中重要的颜色数，若该值为 0，则所有的颜色都是重要的
Palette（调色板）	rgbBlue	该颜色的蓝色分量
	regGreen	该颜色的绿色分量
	regRed	该颜色的红色分量
	rebReserved	保留值
ImageData（位图数据）	像素按行优先顺序排序，按照从下到上，从左到右进行	每一行的字节数必须是 4 的整数倍

2. BMP 文件头

BMP 文件头数据结构含有 BMP 文件的类型、文件大小和位图起始位置等信息。其结构定义如下：

```
typedef struct tagBITMAPFILEHEADER
{
WORD bfType;          //位图文件的类型,必须为 BM
DWORD bfSize;         //位图文件的大小,以字节为单位
WORD bfReserved1;     //位图文件保留字 1,必须为 0
WORD bfReserved2;     //位图文件保留字 2,必须为 0
DWORD bfOffBits;      //位图数据的起始位置,用相对于位图文件头的偏移量表示,以字节为单位
} BITMAPFILEHEADER;   //该结构占据 14 个字节
```

3. 位图信息头

BMP 位图信息头数据用于说明位图的尺寸等信息。其结构定义如下：

```
typedef struct tagBITMAPINFOHEADER{
DWORD biSize;          //本结构所占用字节数
LONG biWidth;          //位图的宽度,以像素为单位
LONG biHeight;         //位图的高度,以像素为单位
WORD biPlanes;         //目标设备的平面数,必须为 1
WORD biBitCount        //每个像素所需的位数,必须是 1(双色), 4(16 色),8(256 色)或 24(真
                         彩色)之一
DWORD biCompression;   //位图压缩类型,必须是 0(不压缩),1(BI_RLE8 压缩类型)或 2(BI_RLE4
                         压缩类型)之一
DWORD biSizeImage;     //位图的大小,以字节为单位
LONG biXPelsPerMeter;  //位图水平分辨率,每米像素数
LONG biYPelsPerMeter;  //位图垂直分辨率,每米像素数
DWORD biClrUsed;       //位图实际使用的颜色表中的颜色数
DWORD biClrImportant;  //位图显示过程中重要的颜色数
} BITMAPINFOHEADER;    //该结构占据 40 个字节
```

注意：对于 BMP 文件格式，在处理单色图像和真彩色图像时，无论图像数据多么庞大，都不对图像数据进行任何压缩处理。一般情况下，如果位图采用压缩格式，那么 16 色图像采用 RLE4 压缩算法，256 色图像采用 RLE8 压缩算法。

4. 颜色表

颜色表用于说明位图中的颜色，它有若干个表项，每一个表项是一个 RGBQUAD 类型的结构，定义一种颜色。RGBQUAD 结构的定义如下：

```
typedef struct tagRGBQUAD {
BYTErgbBlue;           //蓝色的亮度(值范围为 0～255)
BYTErgbGreen;          //绿色的亮度(值范围为 0～255)
BYTErgbRed;            //红色的亮度(值范围为 0～255)
BYTErgbReserved;       //保留,必须为 0
} RGBQUAD;
```

颜色表中 RGBQUAD 结构数据的个数由 BITMAPINFOHEADER 中的 biBitCount 项来确定，当 biBitCount＝1，4，8 时，分别有 2，16，256 个颜色表项；当 biBitCount＝24 时，图像为真彩色，图像中每个像素的颜色用三个字节表示，分别对应 R、G、B 值，图像文件没有颜色表项。位图信息头和颜色表组成位图信息，BITMAPINFO 结构定义如下。

```
typedef struct tagBITMAPINFO {
BITMAPINFOHEADER bmiHeader;                    //位图信息头
RGBQUAD bmiColors[1];                          //颜色表
} BITMAPINFO;
```

注意：RGBQUAD 数据结构中，增加了一个保留字段 rgbReserved，它不代表任何颜色，必须取固定的值为 0。同时，RGBQUAD 结构中定义的颜色值中，红色、绿色和蓝色的排列顺序与一般真彩色图像文件的颜色数据排列顺序恰好相反，即若某个位图中的一个像素点的颜色的描述为 00，00，ff，00，则表示该点为红色，而不是蓝色。

5. 位图数据

位图数据记录了位图的每一个像素值或该对应像素的颜色表的索引值，图像记录顺序在扫描行内是从左到右，扫描行之间是从下到上。这种格式又称为 Bottom_Up 位图，当然与之相对的还有 Up_Down 形式的位图，它的记录顺序是从上到下的，对于这种形式的位图，也不存在压缩形式。位图的一个像素值所占的字节数：当 biBitCount＝1 时，8 个像素占 1 个字节；当 biBitCount＝4 时，2 个像素占 1 个字节；当 biBitCount＝8 时，1 个像素占 1 个字节；当 biBitCount＝24 时，1 个像素占 3 个字节，此时图像为真彩色图像。当图像不是真彩色时，图像文件中包含颜色表，位图的数据表示对应像素点在颜色表中相应的索引值；当为真彩色时，每一个像素用三个字节表示图像相应像素点彩色值，每个字节分别对应 R、G、B 分量的值，这时图像文件中没有颜色表。上面已经提及，Windows 规定图像文件中一个扫描行所占的字节数必须是 4 的倍数（即以字为单位），不足的以 0 填充。图像文件中一个扫描行所占的字节数计算方法：

```
DataSizePerLine = (biWidth * biBitCount + 31)/8;          //一个扫描行所占的字节数
```

位图数据的大小按下式计算（不压缩情况下）：

```
DataSize = DataSizePerLine * biHeight。
```

上述是 BMP 文件格式的说明，弄清楚了以上的结构，就可以正确地操作图像文件，对它进行读或写操作了。

3.3.2　GIF 图像文件格式

20 世纪 80 年代，美国一家著名的在线信息服务机构 CompuServe 针对当时网络传输带宽的限制，开发出了 GIF 图像格式。GIF 图像格式的全称为 Graphics Interchange Format，从这个名字可以看出，这种图像格式主要是为了通过网络传输图像而设计的。GIF 文件不支持 24 位真彩色图像，最多只能存储 256 色的图像或灰度图像；GIF 格式文件也无法存储

式(9.6.9)中，σ 为图像信号的均方根。取量化分层总数为 $K=2^n$，可以进一步简化为

$$\left(\frac{S}{N}\right)_q \approx -6.5+6n+10\lg\left(\frac{\sigma^2}{\sigma_e^2}\right) \quad \text{dB} \tag{9.6.10}$$

式(9.6.10)中的信噪比指的是图像信号的均方根与量化噪声的均方根之比。对于电视信号，常用的信噪比(S/N)表示式指的是复合信号的峰-峰值与噪声的均方根之比。实验测量表明，电视信号中图像信号的均方根大约是复合信号峰-峰值的 1/10。加入这一修正后，式(9.6.10)变为

$$\left(\frac{S}{N}\right)_q \approx -13.6+6n+10\lg\left(\frac{\sigma^2}{\sigma_e^2}\right) \quad \text{dB} \tag{9.6.11}$$

这就是 DPCM 应用于电视信号时的量化信噪比公式。从该式中可以看出，数字视频信号的精度，直接决定了信噪比的大小，每增加 1 位精度，信噪比就大约增加 6dB。

可以把式(9.6.10)的结果与均匀量化的结果作一个比较。由前面内容可知，对线性 PCM 系统，图像信号的峰-峰值与量化噪声的均方根之比为

$$\left(\frac{S}{N}\right)_{PCM} = 20\lg\sqrt{12}+20\lg(2^n) \approx 10.8+6n \quad \text{dB} \tag{9.6.12}$$

同样，式(9.6.12)中如果考虑对复合信号进行修正，只需将常数项修改为 $20\lg\sqrt{12}/0.7=13.6$。于是，线性 PCM 系统中，电视复合信号峰-峰值与量化噪声均方根之比为

$$\left(\frac{S}{N}\right)_{PCM} \approx 13.6+6n \quad \text{dB} \tag{9.6.13}$$

由此可见，线性 PCM 的式(9.6.13)，只不过是 $\sigma_e=\sigma$ 的一个简化情况。

9.6.2 矢量量化

1. 矢量量化的基本原理

在前面介绍的标量量化里，每个样值的量化只和它本身的大小及分层的粗细有关，而和其他的样值无关。实际上，图像的样值之间是存在着或强或弱的相关性的，将若干个相关相邻像素当作一个整体来对待，就可以更加充分地利用这些相关性，达到更好的量化效果。这就是矢量量化的基本思路。如果将一个像素当作一组，则此时的矢量量化，就是标量量化。所以也可以说，标量量化是矢量量化的特殊情况。

矢量量化就是把图像的样值每 n 个作为一组，这 n 个样值可以看成一个 n 维空间。任何一组的 n 个样值都可以看成 n 维空间的一个点，或者说是 n 维空间的矢量。由于 n 维空间的每一维都是模拟量(或连续量)，所以 n 维空间也是一个连续空间，尽管每一维的最大值是有限制的(图像亮度或色度的最大值)，但它所包含的矢量数目是无穷多的。矢量量化要做的工作就是将此 n 维连续空间划分为有限个区间(这一过程相当于标量量化中的“分层”)，在每个区间找一个代表矢量(这一过程相当于标量量化中的“量化值”)，凡是落在本区间的所有的矢量都用该代表矢量来表示，这就是矢量量化的基本方法。

矢量量化的过程如图 9.14 所示，可以分为量化和反量化两部分。在标量量化中，可以根据均方误差最小原则分别求出决定分层范围的判决电平和量化电平。与此类似，在矢量量化中，也可以根据某种失真最小原则，来分别决定如何对 n 维矢量空间进行划分，以得到

合适的 C 个分块，以及如何从每个分块选出它们各自合适的代表 $\boldsymbol{X}_i'$。

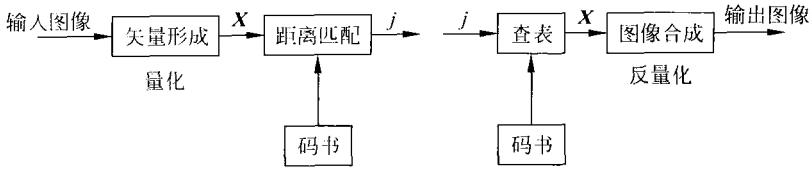

图 9.14　矢量量化过程示意图

量化过程：将一幅 $M\times N$ 的图像依次分为若干组，每组 n 个像素构成一个 n 维矢量 $\boldsymbol{X}$。将得到的每个矢量 $\boldsymbol{X}$ 和码书中预先按一定顺序存储的码矢量集合 $\{\boldsymbol{X}_i' \mid i=1,2,\cdots,C\}$ 相比较，得到最为接近的码矢量 $\boldsymbol{X}_j'$，并将其序号 j 发送到信道上。

反量化过程：解码器按照收到的序号 j 进行查表，从与编码器完全相同的码书中找到码矢量 $\boldsymbol{X}_j'$，并用该矢量代替原始的编码矢量 $\boldsymbol{X}$。

所谓矢量 $\boldsymbol{X}$ 和 $\boldsymbol{X}'$ 的“接近”程度可以有多种衡量方法，最常用的误差测度是均方误差，相当于两者之间的欧几里德距离，即

$$d(\boldsymbol{X},\boldsymbol{X}') = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_i')^2 \tag{9.6.14}$$

该误差虽不能总和视觉结果相一致，但由于它计算简单而得到广泛应用。

2. 码书的设计

由上面内容可知，矢量量化中的一个关键问题就是码书的设计。码书的设计越适合待编码的图像类型，矢量量化器的性能就越好，因为设计中不可能为每一幅待编码的图像单独设计一个码书，所以通常是以一些代表性的图像构成的训练集为基础，为一类图像设计出一个码书。

码书是所有的输出码矢量的集合。这里以二维矢量量化为例来说明码书的生成方法。

此时的输入矢量为 $\boldsymbol{X}=\{x_1,x_2\}$，是一个二维矢量。图 9.15 所示为该二维矢量空间示意图，通过适当的方法将此二维平面空间划分为多个小区间，每个小区间找出一个代表矢量 $\boldsymbol{X}_i'$，也就是图中的黑点表示的码矢量。所有这些代表码矢量的集合 $\{\boldsymbol{X}_i' \mid i=1,2,\cdots,C\}$ 就是码书。因此，设计码书就是在给定训练矢量集的基础上对矢量空间进行划分，并确定所有的码矢量，以使量化误差为最小。

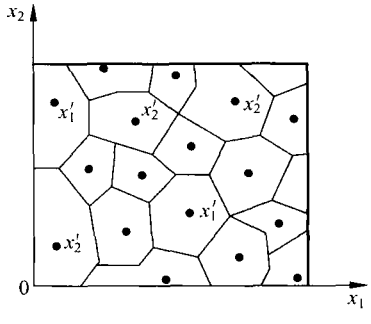

图 9.15　二维矢量空间的划分

码书设计常用的算法是 LBG(Linde-Buzo-Gray)算法，对于给定的训练矢量集 $\{\boldsymbol{X}_i \mid i=1,2,\cdots,N\}$，$N$ 是训练矢量的个数，所设计的矢量量化的码书共有 C 个输出码字。具体过程如下。

(1) 置迭代次数处置为 $m=1$，并任选初始码书为 $\{\boldsymbol{X}_i'^{(m)} \mid i=1,2,\cdots,C\}$，取相对误差变化量阈值为 ε(在 0～1 之间，一般不大于 0.01，用于判断算法是否收敛)，将所有训练矢量的

平均量化误差 $D^{(0)}$ 初始化为一个较大的值。

(2) 确定每个码矢量 $x_i'^{(m)}$ 所在的区间 R_i。即按照最小距离原则，将训练集中的每个矢量划归相应的码矢量。

(3) 计算用码矢量代替所在判决区间中所有训练矢量时的平均量化误差 $D^{(m)}$，若相对误差变化量 $(D^{(m-1)}-D^{(m)})/D^{(m-1)} \leqslant \varepsilon$，则算法已收敛，结束迭代；否则，继续下一步。$D^{(m)}$ 可计算如下，先计算各个判决中的判决量化误差，即

$$D_j^{(m)} = \frac{1}{N_j} \sum_{x_i \in R_j}^{n} \| x_i - x_j' \|^2 \tag{9.6.15}$$

式中，N_j 为判决区间 R_j 中含有训练矢量的数目，然后计算总的平均量化误差

$$D_j^{(m)} = \frac{1}{N_j} \sum_{j=1}^{c} N_j D_j^{(m)} \tag{9.6.16}$$

(4) 重新确定各判决区间中的码矢量，使它等于各区间中所有旋律矢量的平均矢量，即

$$\boldsymbol{X}'^{(m+1)}_j = \frac{1}{N_j} \sum_{X_j \in R_j} \boldsymbol{X}_j \tag{9.6.17}$$

并转到第(2)步。

上述 LBG 算法只能抱着所设计出的码书是局部最优的，但通常存在多个局部最优点，有一些局部最优点的性能并不好。因此，初始码书的选择非常重要，好的初始码书通常能导致高性能的矢量量化器。选取初始码书的方法很多，大多是以给定的训练矢量集为基础产生。一种最为简单的办法是随机码书法，即随机地从训练矢量集中选取 C 个矢量作为初始码书，一般取前 C 个矢量。

如果 ε 取得太小，可能出现算法不收敛的情况。因此，通常规定迭代次数达到某个预定的最大值以后，算法强制结束。

3. 量化性能

矢量量化输出比特率的计算。设码书共有 C 个输出码字，矢量量化器输出为 $1,2,\cdots,C$ 序号，只需 $\log_2 C$ 位。输入矢量为 n 个(n 维)，那么，每个抽样的输入所需的位数为

$$B = (1/n)\log_2 C \tag{9.6.18}$$

在满足一定失真条件下选定了量化级 C 后，由式(9.6.18)可以看出，矢量量化可以减少所需的位数。

9.6.3 量化压缩机理

以上较详细地介绍了两类不同的量化器的设计。当使用量化器来进行数据压缩时，不管是使用标量量化器还是矢量量化器，都可以把整个量化分别看成量化和反量化两个过程，由下式表示

$$Q = \alpha \cdot \beta \tag{9.6.19}$$

其中，量化过程 α 相当于由输入值找到它所在的量化区间号，反量化过程 β 相当于由量化区间号得到对应的量化电平值。通常，直接用 Q 表示 α，而用 IQ 或 Q^{-1} 表示 β。量化实现数据压缩的根本原因在于传输中是以区间标号来取代原来的量值，而量化区间总数远远少

于原量值的总数，由此就能用较少的位将它们传输出去。很明显，经过反量化并不能保证得到原来的值，因此量化过程是一个不可逆过程，用量化的方法来进行压缩编码是一种非信息保持型编码。通常这两个过程均可用查表方法实现，量化过程在编码端完成，而反量化过程则在解码端完成。

既然量化后所传输的是量化区间的标号，那么就有一个如何对这些标号进行编码表示的问题。在大部分应用场合，为了简化硬件或软件实现的复杂性，对量化区间标号（量化值）的编码仅采用简单的等长编码方法。例如，当量化分层总数为 K 时，经过量化压缩后的二进制数码率为$\log_2 K$ 位/量值。在一些要求较高的场合，也可采用可变字长编码来进一步改善编码效率，例如采用霍夫曼编码或算术编码等方法。

9.7　静止图像与运动图像编码

9.7.1　静止图像编码

静止图像是指从显示屏上观察到的内容不变的图像。从被摄对象来看，静止图像包括内容本身是静止的图像以及活动场景某一时刻的“凝固”图像。如果从编码的角度来看，静止图像编码是指对单幅图像的编码。对于静止图像编码的要求主要有以下几点。

（1）清晰度。静止图像应有比运动图像更高的清晰度。

（2）逐渐浮现的显示方式。主要用于窄带传输时，不使观察者长时间等待一幅图像传输完成后才进行显示。

（3）抗干扰。防止由于传输时间过长带来的噪声影响。本节对静止图像编码采用的一些方法进行简单介绍。

1. 方块编码

方块截断编码简称为方块编码（block truncation coding，BTC），是把一幅图像分为大小为 $N\times N$ 的子像块（简称子块），由于小块内各相邻像素间具有亮度互相近似的相关性，于是只选用两个适当的亮度来近似代表小块内各像素原来的亮度，然后指明子块内的各像素分别属于哪个亮度。

1）基本编码方法

设图像中的一个子块的大小为 $m=N\times N$ 个像素，子块中第 i 个像素为 P_i，其亮度值为 X_i，方块的两个代表性亮度为 a_0、a_1，称之为亮度级分量，用一个二元码 ϕ_i 指明像素 P_i 编码后属于 a_0 或 a_1，ϕ_i 称为分辨力分量。设方块内的亮度阈值为 X_T，像素 P_i 编码后的电平值为 Y_i，则基本编码方法可用下式表示

$$Y_i = \bar{\phi}_i \cdot a_0 + \phi_i \cdot a_1 \tag{9.7.1}$$

$$\phi_i = \begin{cases} 1 & X_i \geqslant X_T \\ 0 & X < X_T \end{cases} \tag{9.7.2}$$

由此可知，编码后方块的像素亮度电平值可用 a_0、a_1 与 ϕ_i 的组合来表示。将这种组合作为传输内容，接收端在得到这种组合后，可以恢复出编码的图像。