第一章　图形表示理论

由于读者的背景各不相同，本章先安排讨论图形表示的理论。它并不是对这门理论的全面论述，而只是介绍一些基本的论题，这些论题是读者完全掌握本书内容所必须熟悉的。

1.1　位图和矢量表示

表示图像主要有两种截然不同的方式，即位图（又称光栅或像素图）方式和矢量方式。图形文件可以采用任何一种方式，也可以两种同时采用。

1.1.1　定义

位图表示法是迄今为止更为常用的方法，因为它易于实现并且在一定范围内能为任何图像所用。位图表示意味着一幅图像被划分为一张栅格，格中每一部分（像素）的光度值（亮、暗或彩色）单独记录。图 1.1 显示的是用位图表示的一幅低分辨率图像的一行扫描线。典型情况下，位图域中一个数据点的位置决定了该数据点所代表的像素，换句话说，数据点（位）与图像相对应，“位图”的名称由此而来。

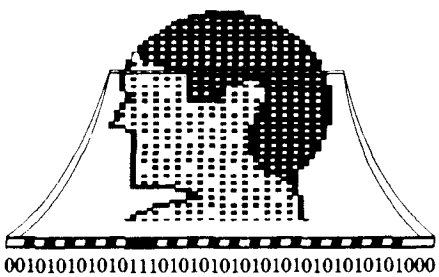

图 1.1　一幅图像中位图数据的一行

矢量表示是指用一系列的线段或其他造型描述一幅图像，在其中可能某些区域用阴影（均匀的或者带深浅的）或彩色填充（严格地说，矢量这个词仅表示线段，不过矢量文件的常见解释包括长方形、圆等的造型）。

如果检查一下矢量文件，我们会发现它们很像程序。它们可以包含用 ASCII 码表示的接近英语的命令和数据，从而使之能用字处理器进行编辑。例如，一个半径为 100mm，圆心坐标 x＝2.25cm，y＝5cm 的圆，就可以用命令 circle（100，2500，5000）来体现，该命令以 ASCII 码记录存储。

图 1.2 表示一种简单的画线方法，线用矢量格式是非常合适的。图 1.3 列出了矢量格式 DXF 中的数据。

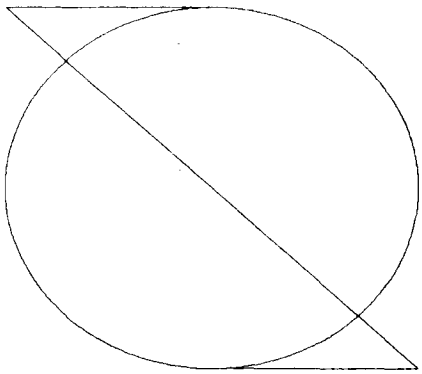

图 1.2 适合矢量表示一种简单的画线法。

文件格式通常只用矢量或者只用位图，不是两者同时使用。偶尔会有这两种方法在诸如 PostScript 这样的页描述语言中同时出现的情况，但即便如此，两者也是截然分开的。每一种方法都有它自己最适合的应用范围。

为方便起见，有些图形应用程序在一个图形文件中对同一幅图像采用了多种表示方法。例如，一个 PostScript 文件可能包含一个 TIFF 预显。TIFF 阅读程序由于比 PostScript 阅读程序更为简单，所以比后者更为常用，所需的内存也少，并且提供显示也快。当一个 PostScript 文件包含一个图像的一份质量降低且按 TIFF 编码的拷贝时，应用程序在非关键的功能方面就可使用该图像，诸如让用户辨别图像文件的异同点。

1.1.2 应用

位图格式最适于色彩、阴影或形状变化复杂的图像，如照片、绘画和数字化视频祯等。有些图像如计算机屏幕显示，本身就是用位图格式生成的，因此用相同的方式记录是最方便的。

矢量格式适合于线型图、像造型、阴影和色彩简单的 CAD 图形和图像。表、图和某些类型的徒手画，典型情况均以矢量文件记录。不过，有时如果用位图方式记录更容易，也可以用位图来记录这几类图像。

在排版应用时，根据图像是处于从字体设计到印刷的哪一阶段，文件中用到的字母字体和字母、符号的描述频繁地在矢量和位图格式之间来回转换。虽然这一做法显得不简洁，但它确实是在两种方法的优缺点之间作了一个很好的折衷。比如，在矢量方式下，字体容易缩放和修改(即斜体、黑体和其他变化)，但对打印机来说，由于是使用类似于电视的光栅扫描，故用位图方式效率更高。相反，绘图仪是为画线而设计的，故需要矢量数据。

越来越多的(但不是所有的)应用程序已经既能处理矢量图像，又能处理位图图像。甚至连 Microsoft Word 这样的字处理程序也能将矢量图像(如 CAD 程序送来的 HPGL 绘图仪程序)和位图图像(如绘画程序的 TIFF 文件)加以集成。不过，字处理器并不需要处理图像，而只需要把图像联到文件中去，后者的要求显然简单一些。

由于图形文件必须处理图像数据，它们总是偏爱与其目的最相吻合的一种方式。在典型

```
  0
SECTION
  2
HEADER
  9
$CECOLOR
 62
     0
  0
ENDSEC
  0
SECTION
  2
TABLES
  0
TABLE
  2
LAYER
  0
LAYER
  2
0
 70
     0
 62
     7
  6
CONTINUO
US
  0
LAYER
  2
obj1
 70
     0
 62
     7
  6
CONTINUO
US
  0
ENDTAB
  0
ENDSEC
  0
SECTION
  2
ENTITIES
  0
CIRCLE
  8
obj1
 62
     7
 10
9.000000
0000
 20
7.500000
0000
 30
0.000000
0000
 40
1.750000
0000
  0
POLYLINE
  8
obj1
 62
     7
 66
     1
 70
     8
  0
VERTEX
  8
obj1
 62
     7
 10
9.000000
0000
 20
9.250000
0000
 30
0.000000
0000
 70
    32
  0
VERTEX
  8
obj1
 62
     7
 10
7.250000
0000
 20
9.250000
0000
 30
0.000000
0000
 70
    32
  0
VERTEX
  8
obj1
 62
     7
 10
10.75000
00000
 20
5.750000
0000
 30
0.000000
0000
 70
    32
  0
VERTEX
  8
obj1
 62
     7
 10
9.000000
0000
 20
5.750000
0000
 30
0.000000
0000
 70
    32
  0
SEQEND
  0
ENDSEC
  0
EOF
```

图 1.3 Vecto-Image 图 1.2 的矢量数据(DXF 格式)

情况下,如果它们需要另一种方式,则是用文件转换来处理的。例如,Corel Draw (PC 机上的一个应用程序)使用的是矢量表示法,但它能通过跟踪阴影区域的轮廓线和将其记录为矢量来把一个位图 TIFF 图像矢量化。

矢量概念的一种扩展就是三维模型和获取软件的使用。像 DXF 这样的语言不但可以包括二维造型,而且可以包括被称为"模型"的三维造型。利用这些模型的获取软件,艺术家等就可以在软件中指定一幅景物,它包含真实世界景物的光度、表面特性、反射性、透明度和其他特性。该景物然后作为一系物模型的指令表示为一个获取程序,例如一个 Renderman 界面文件。界面程序能够计算出景物的一个二维图像外形,并产生一幅位图图像。

1.1.3 位图格式与矢量格式的优点比较

位图表示法可以记录任何可以见到的图像，因为在人眼看来每一幅图像均可以分解为一张栅格。所以，程序开发者常常借助位图来简化编程。

不过，位图图像有好几个问题，既有理论方面的，又有实用方面的。一个实用方面的问题是图像大小。一个高分辨率彩色图像文件可能需要几兆字节的存储空间，还需要数量可能更多的内存用以处理和显示。正因为如此以压缩形式存储图像的能力对于位图文件格式领域是很重要的一个课题。

处理大容量的数据还对计算机的处理器和内部数据总线提出了苛刻的要求。对于位图图像处理来说，使用 32 位内部总线的计算机和 16 位内部总线的计算机，其效率就大相径庭。就目前而言，处理大的高分辨率彩色图像，如欲达到任何可容忍的反应时间，一般就需要有图形工作站或最高性能的个人计算机。

位图图像存在的另一个问题可以认为是缺少灵活性（不过与上述处理问题比起来，这一问题并不纯属文件格式问题）。灵活性问题一方面是像素间没有内在的相互关系。例如，在一幅表现草地上的粮仓的图像中，就不存在粮仓这样的像素，甚至连表示多边形的专门像素都没有。因此，如果想对图像中粮仓的部分作一些处理（如扩大或折转），程序就要做复杂的工作，比如找出相邻像素中的所有阴影为接近红色的那些。但这样一来，粮仓的灰色的顶部可能就找不到了。

位图图像灵活性还有一个问题，它与图像的固定分辨率有关。当图像分解为像素后，X 像素、Y 像素处的分辨率就已固定。当我们试图放大图像时，像素也就变大，大到足以看清它们的长方形形状——这种效果称为变形或阶梯化。灵巧的图形程序可以通过在两个像素间插入一条线来补偿这个问题，但还是涉及到费时的处理过程。类似地，如果把一个图像加以缩小然后以缩小的形式存储，则分辨率常会丢失；如果过后将其恢复到原大小，则图像会变得模糊不清和变形，阴影区域的效果可能会变成如图 1.4 的情况。

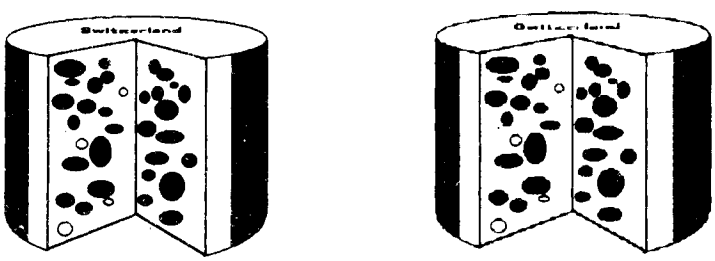

图 1.4 对一幅图像先缩小后放大的效果

就其所能表示对象的范围看，矢量表示法的限制比位图的多，但对许多应用场合前者有效率和灵活性更高的优点。例如，一条线段仅用其两个端点就能描述，一条曲线可以用一系列前后相连的线段去逼近。如果不同的造型使用的是不同的代码，矢量表示法就变得更为有效，比如，用圆、半径、两个端点的代码就能描述一段圆弧。

用这种方法，上述例子中的粮仓就可以用相关多边形的一组带名字的集合体来存储，每

个多边形填充以不同的阴影或色彩。由于这一集合体有统一的识别标志(例如多边形1号),对它加以移动或处理就很容易,而且任意缩放图像也不影响分辨率。

1.1.4 页描述语言、显示清单和元文件

图形描述语言,又称页描述语言(PDLs)、显示清单或元文件,属于文件格式的范畴,但与之又不完全相同。严格说来,在文件格式中,一个数据记录的位置带有某些信息;在图形描述语言中,一个数据记录的位置则无关紧要。这样,考虑到前面关于位图和矢量文件的定义,图形描述语言本质上更适于作矢量表示而不是位图表示(不过,在语言文件中可以为位图表示提供专用的服务功能)。

这些语言依赖于一个复杂的解释程序。例如,这些语言中的命令(像画某一造型物体的命令)经常被类似子例程的模块调用。这些子例程的执行方式取决于早先时候的命令,这些命令定义了该物体绘出的环境。

PostScript 是这些语言中最常用也是最复杂的一个。它具有大量(且仍能扩充)的命令词汇,是一种复杂的页描述语言。在其本来的形式中,它主要是为矢理表示法设计的,用于像打印机和计算机屏幕这样的输出设备。在 PostScript 中,几乎任何东西,甚至连字母打印体这样的奇怪造型也是矢量表示法描述的。不过,有些图像如照片用矢量表示就不妥,因此,PostScript 的设计也考虑到这一点,使其文件能包含位图。

像 PostScript 这样的页描述语言,要对其加以详尽讨论,其复杂程度已超出了本书的范围。不过,PostScript 和其他图形描述语言的某些方面我们将加以讨论,以使读者能理解图像格式转换中的常见问题。

1.2 单色和彩色图像

计算机存储的图像,如照片,可以是彩色的,也可以是黑白(又称单色 monochrome,因为单色可以代替黑色)的。矢量和位图格式均支持这两种类型的图像。矢量格式适用于点色彩,也就是所定义的区域范围使用一种统一的色彩,而位图格式通常既适合于点色彩,又适合于照片色彩。

精确描绘单色和彩色图像全部内容的技术,其种类繁多又极为复杂,详尽地进行讨论也许需要好几本书的篇幅。但是,其中有几条基本概念对于理解图像文件格式是至为关键的,下文即予阐述。

1.2.1 单色

最简单的单色图像由纯的色彩区(通常是黑色)和纯的白色区所组成。这种形式的典型图像是线段图和图表。这些图像的文件通常根本不引用色彩内容,它们假定输出设备(如黑白激光打印机)会使用其所配置的任何一种色彩。对于给定的图像大小,这些文件相对说是紧凑的,比如在位图单色文件中,一位就可以代表一个像素:1 代表黑色,0 代表白色,或者反过来。

一些单色图像如照片和带阴影的图需要用到灰色的阴影(或者另一种色彩),这类图像常称为灰度图像。

由于人眼会把一种足够细致的黑白像素方式解释为灰色，故单色文件格式可用于灰度图像，其数据可以是黑白的（例如，图 1.1 中的图像就用这种方法达到阴影的效果）。用黑点或某种色彩的点获得该色阴影的过程称为半色处理或高频抖动处理过程。所有图像在印刷时几乎都用半色，因为常规的印刷机一次只能使用一种色彩的墨水（这并不是说最终要付印的所有图像文件都必须是半色形式的单色文件，图像可以在整个过程的较晚时候转换为半色形式，最好是在印刷机上）。

用单色文件中的半色图像数据达到灰度的效果，这一做法也有缺点。首先，它增加了图像编辑的难度，因为大多数图形程序不能区分阴影所使用的点和线所使用的点。其次，使用半色数据在显示图像时会引入一些波纹。再次，如果输出设备上的像素数量相同，则使用半色会降低图像的有效分辨率。一个点的灰色值就需要占用几个像素，其中既有黑的又有白的。例如，为造出四级阴影就至少需要三个像素（长方形像素的轮廓要四个像素）。如果输出设备只理会两种值，如某激光打印机只理会黑和白，则最后的输出文件无一例外地必须包括半色数据。不过在付印之前的阶段，可以有更好的选择，即真正的灰度表示。

1.2.2 真正的灰度单色

灰度图像的记录也可以不用半色或重抖动格式的数据，而用真正的灰色阴影。大多数电子图像应用程序都倾向用真正的灰度表示，因为可以简化编辑过程（编辑一幅半色形式的图像常引起问题，编辑过程打乱了黑白像素的精巧安排，引起可察觉的失真，像边缘不光洁或者图 1.4 所示的那种图像）。

使用了真正灰色阴影（即亮度）的图像称为具有一定量的位深度。位图单色图像用三个坐标来描述，即像素的列数（宽度）、行数（高度）和用来表达像素亮度的位数—图像深度，又称灰度分辨率。例如，一个 500 行、500 列、16 种灰色阴影的位图图像可表示为 500×500×4。这几个坐标是有用的数字，因为它们给出了图像文件大小（未经压缩时）的大致概念。上例中的文件大小是 500×500×(1/2 字节)，即 125,000 字节。在一幅图像中，人眼可以轻易分辨出大约 64 种灰色阴影，所以对于实际图像，6 位的深度是必需的（$2^6=64$）。

把灰度和高频抖动结合起来以获得比实际颜色更深的显示也是一种常见的情况。例如，NeXT 机器的双位深色显示器上的 Nextstep 就能相当有效地完成这一工作。彩色显示也有类似的情形。例如，Silicon Graphics 的新型 Indigo 系统（使用 24 位彩色数据）使用的就是 8 位高频抖动彩色视频。当然，这样做要付出分辨率的代价。

1.2.3 灰度精确度的保证

当图像在电子媒体和物理媒体之间转换时，灰色阴影的精确表示可能会成为一个问题。例如，扫描一幅照片时就会出现这个问题，扫描仪对光度变化的反应通常是一种线性的方式：两倍的亮度得到的灰度值也是两倍大。但人眼的反应却是光度的立方根，一个点若其亮度为另一点的两倍，则其实际光强或反射度为另一点的八倍。

几乎所有的输入输出媒体都有自己所能达到的亮暗度极限，因而都有本质的非线性。比如在印刷品中，这些极限是墨水、纸和所用半色的质量所决定的；在扫描仪中，传感器的灵敏度是有限的，而且不可能完全线性；在 CRT 中，屏幕的荧光输出也有一定的限度和非线性特征。

处理这些问题主要就是要了解各种极限和媒体的特点，并对数据作相应的调整。因此，有些文件格式包含了允许对灰度作调整的信息(这一信息通常以图像数据的源而不是目的为特征)。但只有极少数应用程序目前能处理这些调整，允许这些调整的文件格式也很少。

考虑用非线性扫描仪扫描一个图像的情况。输入输出设备中的非线性通常集中反应到一个称为 gamma 的量中，其方程为 OUTPUT=(INPUT)** GAMMA。扫描仪的输出文件就能记录该扫描仪的一个 gamma 值，这里的 INPUT 是某像素的初始光强，OUTPUT 是扫描仪产生的相应该像素的灰度值。

接收到扫描仪的 gamma 值之后，应用程序就能精确地重建初始灰度信息。如果没有向其提供 gamma 值，那么程序就认为数据已作了调整或者要使用缺省的 gamma 值。

某些设备的非线性特性并不符合用 gamma 表达的一个简单的对数曲线。对这种情况，图形文件可以换用表格的形式来记录响应曲线。

文件中可能出现的另一种灰度数据包括初始图像及其亮暗度的对比信息。当图像数据已被放大或值有变化以适应某些表示方案中的值的范围时，这些数据通常是要用到的。比如说若缺少这一信息，则一个有意放低对比度的图像可能被显示成不合适的高对比度图像。

1.3 色　彩

直到不久前，电子存储图像中色彩的使用仍局限于某些专业应用，这很大程度上是因为彩色输入、输出和处理硬件的费用和效率问题。目前，色彩方面的进步方兴未艾，彩色输入、输出设备价格下降，印刷技术蓬勃发展，像桌面排版系统这样大众化的 PC 应用软件利用当前 PC 机的强大处理功能以支持扩大彩色的应用范围。

由于彩色图形文件的结构反映了彩色的理论和应用情况，我们开始先讨论这些基本因素。

1.3.1 点色彩

彩色表示的最简单形式是我们将称为有限点色彩的形式，其中的色彩通常是从输出设备的几种色彩中选出、并应用于图像中的其个图形元素。比如说，配备有多种彩色笔的绘图仪，其文件就应指定某个特定的圆用哪支笔画。由于点色彩通常应用于线和造型，因此在以矢量格式或图形描述语言存储数据的应用程序中最常使用点色彩。

随着能打印大范围色彩的彩色打印机逐渐普及，点色彩也可以引用把多种原有色彩加以混和得到的色彩，参见下节。

1.3.2 色彩的混和

当需要更大范围的色彩时，可以通过把原有的有限数量的色彩加以混和来得到。根据输出媒介的不同，有两种基本的色彩混和方案可供选择，即 RGB 和 CMYK。

RGB 色彩代表红、绿和蓝等三种加型底色，这些底色通常在诸如计算机监控器这样的发光设备中使用。彩色屏幕由荧光点的光栅点组成，每点发出的光是红、蓝、绿的一种，每个栅点均形成一个像素(当从远处看时)。通过调整这三种光的比例，就可以产生大范围的色彩。由于所获的的色彩是通过加以不同的纯色彩(即红、绿、蓝)的光度来得到的，故它又称为

加型色彩。

CMYK 色彩代表青、粉和黄等三种减型底色，再加上黑色作为对比色，它一般用在印品上。每种底色的墨水均从打印纸的白光中吸收某些色彩。由于 CMYK 是通过从白色中消除某些特定色彩获得的，它又称为减型色彩。

对于计算机和视频屏幕，混和色彩的正宗方法是 RGB(因输出光由红、绿和蓝点给成)，对于打印，正宗方法是 CMYK，尽管如此，输出设备或其软件驱动器通常还通过翻译数据来处理任何一种类型的文件。不过，并不是所有色彩都翻译得好，这属于文件转换或解释的一个问题。

另有一种越来越普及的色彩说明方法即 HSI，它可以独立于混和方案而单独使用。HSI 代表色调、饱和度和光强。色调就是我们所认为的色彩(如橙色)；饱和度描述白色光的含量比例，例如烤巧克力的棕色其饱和度就很高，而奶油巧克力虽然与之色调相同，但饱和度相对要低些；光强代表亮度，举例说，就是使阳光下的和阴影中的桔子相区别的因素(经常用光度 Luminosity 代表光强，这时前面的 HSI 就变成 HSL)。

HSI 是用于指定色彩的更好方法，在目前的大多数图形应用软件中都有。甚至有一些图形适配器可以接受 HSI 数据，但 HSI 数据最终都必须转换为 RGB 或 CMYK 数据。例如，计算机监控器仍要求红、绿、蓝输入信号，因此一台 HSI 兼容的适配器必须做从 HSI 到 RGB 的翻译工作。

在视频应用程序中，另一种常见的色彩方案是 YCbCr。其中，Y 代表光度，Cb 代表蓝底色的色度，Cr 代表红底色的色度，绿底色从这三者值可以得出。粗略地说，光度就是一个色素的明亮程度，而色度就是色彩。把亮度与色度分开就能使图像压缩技术利用人眼的一个特点，即对色彩的分辨率要求比对亮度的分辨率要求要低。随着应用程序趋于支持桌面视频，这种色彩方案被越来越多地采用。甚至像 TIFF 这样用于静态图像的一些格式，也有允许实现 YCbCr 的选项。

这些色彩方案也被描述为色彩空间：即三维或四维的空间，其中的每个点定义一种色彩。色彩空间之间的相互转换在理论上是简单的，而事实上却较复杂。四维色彩空间的 CMYK 比 RGB 效率要低，因为黑色部分要加倍 C、M、Y 三个部分的暗效果。因此，质量(色彩和空间分辨率)相当的文件，若按 CMYK 表示，要比按 RGB 法表示来得大。转换也不总是可逆的，一幅从 RGB 转换为 CMYK 的图像如果转换回 RGB，其样子与原图像不会完全相同。这里部分原因是，两种坐标系描述的是真实色彩空间的不同造型，而在 RGB 空间中的某些色彩在 CMYK 空间中根本就不存在。使 RGB 值映射为 CMYK 值的算法必须把一些超界的值强制映射到 CMYK 空间，从而引进了失真。我们可以考虑一下用一根不到一米的棍子表示一米长度的办法。以这种极端值记录度量结果时，可能要用到从棍长度表示的近似值，但折算为以米表示时就得不到原始的值。当然，厘米与“棍的刻度”的映射关系是可以建立起来，只是要损失一些精度。反映射回厘米时的舍入误差，即引入不精确结果。

1.3.3 彩色图像平面

彩色位图图像中的每个像素(或者彩色矢量图像中的每个图形元素)都需要有三个或四个值：每一种底色(RGB、CMYK)或加上独立度量(HSI)各一个值，具体取哪一类值取决于所采用的色彩表示方案。这些值在图形文件中的组织结构方式一般有两种。组织位图图像

数据的最常见方式是用色彩来组织，即组织成“色彩平面”。这种方式很容易被看作三个或四个单色图像，每种底色各有一个单色图像（与印刷工业中的“色彩分离”相似）。例如，在RGB方案中，对图像中的每个像素，所有代表红色的数据都被归并到一起，然后归并绿色数据，然后再归并蓝色数据。

组织图像数据的第二种方式是将其组织为单一的“图像平面”，其中，每个像素的红、绿、蓝值都归并到一起。这也是大多数矢量图像所使用的方式，其中的RGB、CMYK或者HSI值与矢量归并到一起。

一台输入或输出设备所能支持的色彩数量受位数的限制。例如，24位色彩（通常认为这对于苛刻的应用场合也已足够）是指这样的一种设备，它有三种底色，每种底色的分辨率为8位。这样24位色彩就能提供2^{24}或超过一千六百万种的色彩。使用如此高分辨率彩色图像的不足，是这些图像的文件大小。一个1024×1024×24位位图的图像需要3兆不压缩字节。

有些计算机监控器的图形适配器通过采用“调色板”实现色彩的办法绕过文件过大的问题（因而也降低了费用）。这一办法可以在图形文件格式中反映出来。在这一方法中，一个图像只能有这么多不同的色彩，即使这些色彩的分辨率很高也一样。例如，一幅图像由16位调色板的色彩即65536种色彩组成，而这些色彩可以是24位色彩。一幅1024×1024分辨率的图像若使用16位调色板，则只需2兆字节。在实际使用中，24位“色彩空间”中的一个8位调色板通常已足以产生使人满意、质量与电视画面相当的色彩了。调色板图像中的像素数据由n位像素编码构成，每一个像素码都指向调色板表中代表RGB值的三个数。因此，数据文件不仅必须包括像素数据，而且必须包括调色板数据表。

1.3.4 彩色精确度的保证

就像单色（灰度）图像那样，彩色图像在不同的媒体上传送时也会有精确度的问题。这两者中大多数问题的解决方式是类似的，通常的做法是使用色彩平面结构，把每一个平面当作一个单独的单色图像来处理，这样就可用前面讨论过的灰度精确度校正法了，从而可以将每种底色内部的错误加以补偿。

但是，色彩平面之间的错误也必须加以修正，否则的话，图像的整体色调会向某种底色漂移。修正的方法一般是针对某一标准指定图像的“白点”（或者其他色彩的点，像纯的红、绿和蓝等）。

一幅以电子形式存储的图像在各底色值相等的地方出现“白色”。但当这些值转换到诸如监控器这样的输出设备上时，所出现的结果就未必是我们想象中的白色。另外，对于纸面图像上的一个白点，即使它与监控器上的白点不同，彩色扫描仪对它的响应也可能是一个“白色”像素即它的R、G和B值相同。为完全解决这些问题，输入或输出设备上的“白点”的概念必须按某一个标准赋以一定的特征。

人们使用的色彩度量标准有好些种。一些机构创立了自己的标准，这些机构包括CIE，NTSC（全国电视系统委员会），SMPTE（电影和电视工程师学会），ISO（国际标准组织，它在美国的代表机构为ANSI，即美国全国标准学会）。每一种标准都能用一组数定义一种可以客观测量的色彩。因此，输入或输出设备所记录的“白色”这种色彩就可以用它的矢量来测量，用一组数来定义。这组数称为相应设备的“白点”。

所以，为了精确记录一幅彩色图像，其图形文件就必须按照某个标准记录下表明初始设

备白点的那些值，最初由计算机监控器产生的图像使用的是该监控器的白点，扫描仪产生的图像则使用扫描仪的白点。

与单色精确度测量的情况类似，目前只有相当少的设备和应用程序可用于维护彩色精确度信息，而这一信息几乎从来不包含在图形文件格式中。所以，在彩色精确度显示重要的场合，我们建议读者应了解在各种文件格式中有无这一信息以及所选的应用程序是否用到这一信息。

1.4 通用的编码和压缩方法

如果不考虑图形文件格式的变化，则数据编码的办法有些是通用的。此外，在压缩图形数据、节省内存空间方面也有一些传统的方法，不过，压缩的具体细节是各有区别的，一般在文件格式说明中加以描述。

编码和压缩的方法是非常之多的，也许比本书的页数还要多，但其中很多方法只是以下基本概念的组合和变化形式。

1.4.1 二进制和符号编码

图形数据的编码可以是二进制的形式，可以是符号的形式，也可以是两者的混和形式。例如，PostScript 文件中的命令和数据通常就以符号形式记录，故用文本处理器就可以读懂它们；TIFF 文件是以二进制形式记录的，为了读起来容易，需要有调试器或者其他的二进制解释器。

符号编码（几乎总是用 ASCII 码）的效率相对要低些。不过，许多矢量文件和图形描述语言都是用这种形式记录的，这里，ASCII 码的低效从矢量表示法的高效中得到补偿。

二进制编码通常用于位图数据，但一个值得指出的例外情况是 PostScript，它的位图几乎总是使用 ASCII 码。因此，包含位图数据的 PostScript 文件与其他任何位图文件相比，在数据量相同时，前者的文件要大得多（但是请注意，Level 2 和显示器 PostScript 对于某些应用场合允许使用二进制编码）。

1.4.2 二进制数据中的位和字节顺序

符号编码的数字通常以其自然顺序出现，与在印刷时出现的一样。二进制编码的数字则可以使用不同的字节顺序，在位图图像中，还可以有不同的位顺序。

举例来说，在一个简单的只有一位的位图图像中，图像可以按从左到右，从上到下或者其他任何方向组合的顺序进行扫描。此外，所扫描的 n 个像素的第一个可以与 n 位长的数据字的高位对应，也可以与低位对应。所以，我们必须弄清扫描顺序、字长度和位映射顺序，这样才能很好地完成对图像的解码。

位图图像如有一个像素带好几位的情况，则变化的可能性就更多了。每个像素的多位值都可以一次性记录下来，比如一个 16 位值可以记录为两个连续的字节。这一顺序可以认为是建立一个单一的 16 位深的图像平面。换一种处理方式，这一图像还可以分解为 16 个一位深的位平面，其中，所有像素的最高位记录到一起形成一个平面，所有像素的次最高位也记录到一起形成一个平面，其余类推。

字节顺序也会有变化，它既能影响位图图像，又能影响矢量图像。字节顺序通常取决于计算机所使用的处理器。字节顺序的一种变化就是对于超过一个字节的数，其最高字节和最低字节的顺序，因为这些数的编码可以最高字节放在前面，也可以放在后面。例如，一个16位的整数表示为两个字节既可以是0000001 00000000（即16进制0100），也可以是00000000 0000001(16进制0001)。

浮点数的各个元素（符号、分数、指数）也可以用不同顺序存储，对于分数和指数，字节顺序可以是最低位(LSB)在先，也可以是最高位(MSB)在先。在这里，这些元素的顺序仍然决于计算机的CPU。

IBM PC机和其他采用Intel CPU的计算机的内存中，LSB在前，而Motorola CPU的机器如苹果Macintosh机，则是MSB在前。按照<<格列佛游记>>中的说法，人们常把Intel顺序称为"little－endian"，把Motorola的顺序称为"big－endian"。Sun，IBM和HP等公司的UNIX工作站存储数据时通常MSB在前，Intel和DEC公司的UNIX工作站则LSB在前。这些系统上的文件中所存储的数据几乎总是反映机器本来的顺序，也就是数据在内存中存储的数据，从低位地址到高位地址依次读取。

由于存在这些差别，一种图形文件格式如果要在不同厂家不同计算机上使用，就必须记录有关位和字节顺序的某些信息。一种专为某类计算机开发的文件格式通常没有字节顺序信息，但如果已知该类计算机的CPU类型，则正确顺序可以从中推测出。位顺序和字长度能由图像源如扫描仪单独控制，另外也能用文件格式说明加以控制。

第二章　图像的压缩

随着用户在其计算机应用程序中要求更大的图形容量和更高的图像质量，所需要的数据量也迅速增大——其速度已不是更大的存储媒体和更强的I/O能力就能弥补的。这一个增长了的需求已把重点放在数据压缩技术上，也就是用更少的位数存储相同容量的信息。多媒体应用程序中，计算机显示视频或动画，因此最需要采取数据压缩。然而，即使是在"静态"图像的程序中，用户都希望有更高的分辨率和更好的色彩质量，或者能存储更多的图像。所有这些提高都需要在同样的物理系统上能得到更多的数据，因此对压缩的需求更为迫切了。

"压缩"就是更高效地表示信息，"压出"数据中的"空气"，因而得名。它利用了图形数据的三条共同特性，即数据常常是冗余的、可预测的和不必要的。像运行长度编码这样的方法利用的是数据的冗余性，它会这样说："这里有三个相同的红像素"，而不是说："这里有一个红像素，还有一个红像素，还有一个红像素"。Huffman和算术编码所依赖的是一个统计模型，它利用了数据的可预测性，对于更常出现的像素值采用更短的编码。不必要的数据的存在引出了有损（或译舍弃，见2.5节）压缩法。例如，人眼随意观看一幅图像时，对色彩信息的分辨率要求就没有像对光强信息的分辨率那样高，表示高的分辨率的数据就可以舍弃。数十年前，艺术家们就利用了这一事实并发明了"着色"的照片，即把一个刷子（低分辨率）应用到一幅黑白照片（高分辨率的光强位图）上。今天的彩色视频压缩法通常采用相同的原理。

然而，压缩是一种折衷。压缩算法的效率越高（因而越复杂），图像解压缩所需的计算量或时间就越多。时间对于静态图像的关系并不大，但对于视频或动画却是至关重要的。目前使用的PC机大多数都没有足够的CPU能力用以对高质量的压缩视频解压缩。它们通常也不具备读或写高质量的未压缩视频数据的I/O能力，除非机器功能已专门增强过。因此，程序常常是自动地在质量和数量上求折衷。帧速率、分辨率、色彩深度和帧大小常常都作极小化处理，以允许处理视频。

目前，压缩通常用于存储大容量的图像数据，或者是单独的图像，或者是电影。毫不奇怪，用于压缩和解压缩的软硬件正越来越多地成为计算机平台的组成部分。例如，Macintosh计算机的System 7操作系统现在就包括一个Compression engine（压缩引擎），提供了称为codecs的几种不同的压缩和解压缩方法，用于编码器/解码器。标准的编码/解码方法，像用于静态图像的JPEG和用于动态图像的MPEG，正越来越多地受到平台和文件格式两者的支持，不过有专利权的方法也同样常见。

本章我们将回顾一系列压缩方法，从简单的运行长度编码到预测方法，到复杂的有损（即舍弃）压缩法。其中的有些方法，其处理细节是如此的复杂，以至于超出了本书所能覆盖的范围。事实上，有些方法太复杂了（在某些情况下还受法律保护），我们建议开发人员购买其代码而不要亲自去编写代码。更复杂的情况也会出现，因为像JPEG这样的某些方法的名称实际上是指一组算法，其中的每一种算法又可以按很多方式实现，造成文件格式也相当不同。

2.1　运行长度压缩法

压缩文件最简单的方法之一就是运行长度编码。在这种方法中，一系列重复的值(像素值)要换成一个值加一个计数。例如，用字母来代表值，则像 abbbbbbb ccdddeedddd 这样的一串值可以替换为 1a7b2c4d2e3d。对于有重复值的长字符串，这一方法易于实现，而且很有效。这种压缩方法的很好应用例子是那些有大块恒定阴影或色调的图像，像那些用"图画"程序产生的图像。

运行长度压缩法的一种著名实现是苹果 Macintosh 机上位图数据所使用的"PackBits"。PackBits 假定对象为 8 位数据，用两个字节对重复值进行编码。第一个字节包括一个数 n，n 在－127 到－1 之间(含两端点值)，这就提供了重复计数－n＋1；第两个字节包括要重复的值。非重复的值，条 abcde，用一字节的编码 m 代表其起始处，m 在 0 到 127 之间(含两端点值)，它给出了字符串长度 m＋1(本例中 m 为 4)。重复运行不能长于 128 个字节，而且必须分解为多个运行来处理。在通常情况下，压缩不能从一个扫描行跨到下一个扫描行。

本书中所述的许多位图文件格式，像 Macpaint、TIFF、GEM 和 PCX 都用到运行长度编码。

2.2　Huffman 编码法

Huffman 编码(Huffman，1952)是一种通用的压缩方法，它把数据替换成更为有效的代码。这一方法是 1952 年为文本文件而提出的，其后已经历了多次发展变化。它的基本做法是赋给每一种值以一个二进制码，码的长度各不相同，短的码用于出现频度高的值。这些赋值情况存于一张转换表中，发出代码之前，这个表要发送给解码软件。

例如，abbbccccddeeeeeeeeef 中有 6 种值，其出现频率如下：

a:1 b:3 c:3 d:2 e:9 f:1

为了给每种值建立一个最小编码，我们使用如图 2.1 所示的二进制树。基本的算法就是把出现频率最低的元素配成对，然后把这一对元素当作一个元素，它们的频率也进行合并，对其他元素也按其频率高低作同样处理，直到所有元素合并完毕。

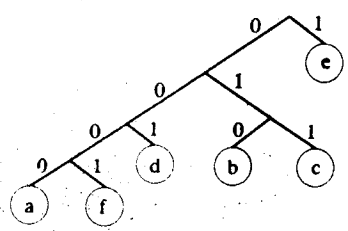

图 2.1　用于 Huffman 编码的一棵二进制树

本例中出现频率最低的值是 a 和 f，所以它们组成第一对，a 被赋给 0 分支，f 赋给 1 分支等。这就是说，0 和 1 分别是 a 和 f 编码的最低有效位。随着二进制树的逐步建立，我们得到了各个高有效位。

然后把前面两个的频率进行合并，总共是2，于是2就变成新的最低频率，这一对就与d（其频率也是2）合并。原来的a:f对赋给0分支，d赋给1分支等。这样，a的编码以00结束，f的以01结束，而d以1结束，且比a和f的编码都少一位。

二进制树就按这种方法合并，这样出现频率最低的值就用较短的编码描述，而频率最高的编码只需一位就够。任一编码都不会浪费，而且是最简洁的。

Huffman方法所使用的具体算法依文件格式而定，但压缩率很少有超过8:1的。所以，对于非重复值很多的文件用Huffman方法并不合适，改用运行长度或其他编码方法可以更好地完成压缩。

Huffman方法还需要对原始文件中每个值所出现的频率作精确的统计。没有精确的统计，最后的文件就可能比原始文件短不了多少，甚至还有可能更长。所以，为使Huffman方法正常工作，应用时经常分为两步，第一步是建立统计模型，第二步是对数据进行编码。因为变长度的代码需要很多处理措施加以解码，Huffman压缩和恢复方法的速度相对来说就较慢（有几种Huffman编码器使用固定的编码表，最有名的就是用于传真机图像压缩的那些编码器）。

Huffman法的最后一个问题（其他变长度压缩法中也存在）是它对删去的和加上的位的敏感性。由于所有的位，不论字节边界，都被塞在一起，因此，解码器判断一个代码器是否结束的唯一办法是到达一个分支的顶点。如果有某位被删去或加上，则解码器就从代码的中开始工作，其余数据就失去意义了。

2.3 LZW压缩法

LZW是较晚时候出现的压缩算法，由Welch提出，Lempel和Ziv将其发展为实用方法（Welch 1984）。这一算法起初是为硬件实现而设计的，它也有几种常用的变化形式。对于这一算法，Unisys公司的Terry Welch申请到了专利#4,558,302号，IBM公司的Miller和Wegman申请到了专利#4,814,746号。有趣的是，从专利的观点看，像UNIX的压缩设施这样的一些软件实现已可公开使用。

LZW法不需要在编码前先建好一个代码表，这点与Huffman法不同。前者从一张简单的表开始，建立一张更为有效的代码表，它有一种自适应性。另一点与Huffman法不同的是，LZW实现对于出现频率高的值一般也不使用更短的代码。

LZW法从这样的一张表开始，其中为数据的每一种可能值分配一个代码项，例如8位数据就有256项。之后，每找到值的一种组合模式它就往表中加项。这样就有必要选定表的最大长度使得代码的长度可以固定下来（LZW的有些实现，如Compuserve的GIF格式中所用的，还允许长度为可变）。

我们考虑一个例子，在序列ababaaac aaaad中，每个字符都代表2位数据值。LZW的编码器和解码器都从同一张表开始，随着表的扩充而扩充。这一字符序列的初始代码表示可以是这样的：

a:00

b:01

c:10

d:11

LZW 寻找它所能识别(或从表中发现)的最长的组合模式。它找到第一个值 a,能识别出该值,然后找出 ab,但识别不出来。这样它就发送出它能识别的值的代码(000),并对于它不识别的值产生一个新的表项。编码器表就变成:

a:000
b:001
c:010
d:011
ab:100

编码器然后结合新表项的最后一个值 b,继续寻找给合模式。这一次是 ba,编码器还不能识别,所以它发出 b 的代码(001)。解码器因此收到这些数据,然后做基本相同的事情,把 ab 的编码加入到自己的表中。

下面就比较顺利了,编码器和解码器都能识别出下一个的 ab 序列,所以一个 3 位代码取代了两个 2 位代码,其好处对于本例虽然不明显,但在实用例子的长序列中会显得很突出。甚至就在本例中,之后的 a 值多次重复也将进入表格并翻译成一个 3 位的代码。

LZW 编码所提供的压缩比通常在 1:1 到 3:1 之间,组合模式较理想的图像压缩比有时可达 10:1。数据值有随机变化的噪声图像难于用 LZW 压缩。随机性是组合模式的对立面,LZW 法只有能找出组合模式时才能有效(为解决噪声问题,低位可以舍弃,或者对值在局部作平均处理)。

本书中的 GIF 和 TIFF 格式使用到 LZW 法。

2.4　算术压缩法

与 Huffman 编码类似的是,算术压缩法也用短的代码表示出频率高的东西,用长的代码表示低频的东西。不过,后者的效率更高,它能压缩的不但是值,还有值序列,这点与 LZW 法相像。此外,它的编码方法对于由大量重复的相同序列组成的文件最为适用。算术压缩法能够接近压缩比的理论极限。算术压缩理论的参考文献可参见以下作者的文章:Abrahamson(1980,Langdon(1984),Rissanen(1979)和 Witten(1987)。

算术压缩法有几种变化形式,每一种都很复杂,不可能在这里详细讨论。简单地说,它把每一个不同的序列映射到 0 到 1 之间的虚数行上的一个区域。这个区域用可变精度(位数)的一个二进制分数来表示,出现少和数据需要精度更高的数(更多的位)。

我们仍以前面讨论 Huffman 编码时以字母表示的一组像素值为例。假设在一幅 1900 像素的图像上每个像可能取六个值中的一个,这些值出现的频率如下:

a:100 b:30 c:300 d:200 e:900 f:100

也可以换一种说法,即像素值出现的概率分别为 P(a)=0.0526,P(b)=0.1579,P(c)=0.1579,P(d)=0.1052,P(e)=0.4737,P(f)=0.0526。像素序列 eb 的概率则是 P(e)×P(b)。我们可以通过概率表示像素或像素序列。困难在于,这些概率不是唯一的,所以我们无法区分 b 和 c(均为 0.1579),eb 和 ec(均为 0.4737×0.1579),也无法区分 eb 和 be,等等。

不过,如果我们把每个值的概率解释为一条线段的长度,并像图 2.2 左边那样以某种任

意的顺序把这些线段堆起来，就可以得到表示像素序列的一种更好方法。

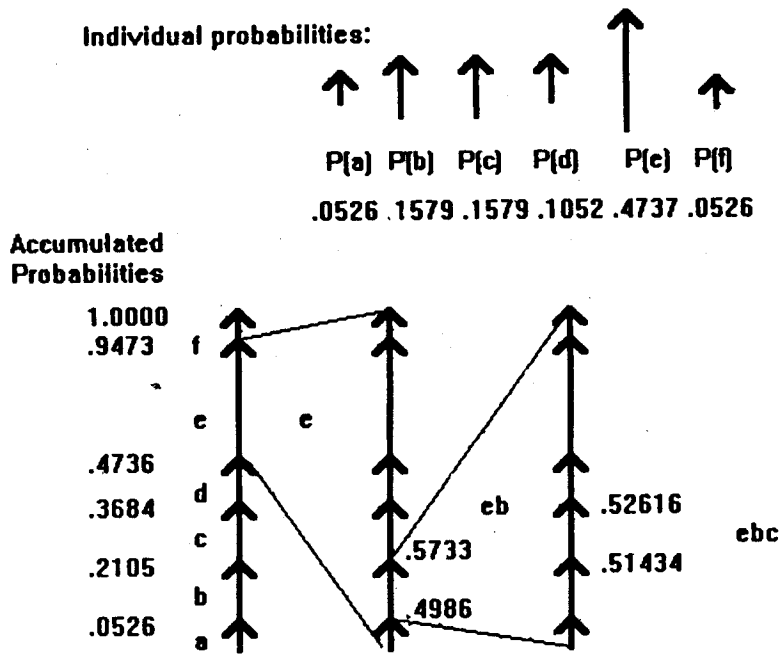

图 2.2 把像素序列映射为 0 到 1 之间的一组值

例如，现在我们用 0.0526 到 0.2105 之间的线段表示 b 值，该线段的长度即其概率 P(b)。c 值的概率与之相同，因此线段长也一样，但它的范围是唯一的：从 0.2105 到 0.3684。现在，我们就能通过代表线段内任何一点的一个数来唯一地表示任何像素值。例如，0.25 表示 c，因为它落在代表 c 的线段范围内。注意到对于表示 c，两位小数点就够了。

为了表示两个字母的序列，可以对这些线段逐条再分。例如，以 b 值开始的一个序列意味着对 b 的范围(0.0526 到 0.2105)进行再分。其中，第两个值为 a(概率 0.0526)的序列将以该范围的前 5.26%来表示，为 b(概率 0.1579)的序列将以紧接着的 15.79%表示，依此类推。这样，序列 ba 将可用 0.0526 到 0.0554 之间的任何数(如 0.53)表示；序列 ee(概率大表示线段长)将可用 0.4737 到 0.6981 之间的任何数(如 0.5)表示。

现在我们就有了一种唯一表示任何值序列的编码方案：即指向 0 到 1 线段上某个位置的一个指针。当然，随着序列变长，所需要的数字将增加，因为指示的准确度变高了。很清楚，我们达到软件的准确度极限时需要终止一个序列，开始新的一个序列。

有趣的是，对于低概率值如 a 和 b 的序列，我们也需要较大的精度(更多的值)。这样的序列用一小段线段的一小部分来表示。反之，对于高概率值如 e 的序列，需要的位数较少。前面曾提到，表示序列 ee 用小数后一位数就够了，但对于 ba，却需要小数点后三位数。虽然实际上用的是二进制而不是十进制，结果也是类似。线段越长，我们就更有可能找到像 1/2、1/4 或 1/8 这种低精度的二进制分数来表示线段。

正如 Huffman 压缩法那样，算术编码法能以更少的位数有效地表示更常出现的像素

值。这种方法的独特结果是，允许传送的不是每个值的整数位。

算术压缩法可以大大缩小文件长度，具体比例取决于源和所采用的统计模型的精度。100:1 的压缩法也是可能达到的。

2.5　舍弃压缩法

舍弃压缩法就是将原始数据中那些与最后应用无关的部分加以舍弃的技术。这样的技术在很挑剔的应用场合如医学图像中可能不宜使用，但对商业电视是很适用的。

舍弃压缩法中可以舍弃的东西包括人眼观察时所不需要的东西，诸如过尖的边角和色彩的过高的空间分辨率(例如，典型的美国彩电的色彩变化少于 50 种，人眼对此从来注意不到)。

舍弃压缩法在计算机视频应用程序中的应用逐渐见多。多媒体应用程序中的当前兴趣一般在于视频或动画，使得计算机和操作系统设计人员开始把舍弃压缩法的功能包括到基本系统中。例如，苹果计算机现在就以其压缩的视频压缩器的形式把舍弃压缩法包括到其 System 7 操作系统中。

最常引用的舍弃压缩方法是 JPEG 基算法，后者在本书中将有专门一章作介绍，这里给出一个定性的描述。JPEG 是世界标准化组织的联合摄影师专家组的英文首字母缩写，它定义了几种算法。最常被称作 JPEG 的算法是基本舍弃算法。JPEG 还定义了一种无舍弃算法，这是基于微分脉冲代码模块。这两种算法实际上都采用了多种压缩方法的组合，包括 Huffman 和算术压缩法。

JPEG 算法开始先把色彩信息与亮度信息分离，以充分利用人眼对低的色彩分辨率的容忍能力。接着，图像被分成几个更小的部分。舍弃压缩法然后就对每个更少的部分使用所谓离散余弦变化(DCT)的算法。经变化后，单个的像素就不再存在，它们以一系列造型表示，后者描述像素值的变化速度。由于单个像素在 DCT 中已不存在，所以 JPEG 的这种形式称为舍弃形式(或译有损形式)。

DCT 有点像在高速公路上通过注明车速而不是计算车数量来测度交通拥挤的情况。给出有关车速和交通公路的足够信息后，就可重构出沿高速的交通密度的一种较准确的情景，不过我们永远无法得出单独某辆车的确切图像。

我们可用另外一种非数学方法考虑舍弃 JPEG 法，就是想像把图像小块分成一系列模糊度逐渐降低的图像，把它们想像成为摄影师调好焦距后所拍摄的一系列图像。这与舍弃 JPEG 大致相仿，只是在 JPEG 中以前述谷仓为例，大块的红色区在较模糊的图像中显得突出，而在后来的图像中逐渐消失，主要是为了照顾像谷仓边缘这些细节。每个图像小块都是通过叠加这些图像而重建的。由于色度和亮度是分别处理的，所以色度方面的更详细“照片”可以舍弃掉。事实上，根据应用程序和平台能力的需要，亮度方面的细节也作同样处理。

JPEG 受大量格式的软件的支持，不过，每一种大体上都有自己的一些独特特点。例如，TIFF 6.0 支持 JPEG 编码，但程序开发人员宜获取详尽的 TIFF 规范，以确保正确处理每一个细节。类似地，苹果机的 System 7 操作系统的 QuickTime 中的照片压缩器支持基本的 JPEG 位流(9R9 版本)。

MPEG 是与 JPEG 有关的一种刚出现的标准，用于存储动画图像。它由动画图像专家组