JPEG 图像文件格式分析及显示

陶 胜

JPEG (Joint Photographic Experts Groups)是国际标准化组织(ISO)和国际电报电话咨询委员会(CCITT)联合制定的静态图像的压缩编码标准。和相同图像质量的其它常用文件格式相比，JPEG 是目前静态图像中压缩比是最高的，能将图像压缩到 10% 以下、且几乎不产生失真，因此 JPEG 图像文件是网络和 BBS 上最常用的图像文件之一。本文介绍 JPEG 图像文件格式，并实现对 JPEG 图像文件格式的显示。

一、JPEG 文件格式分析

JPEG 文件由两部分组成：标记码和压缩数据。标记码部分记录了 JPEG 图像的所有信息，在每个标记码之前可以有个数不限的填充字节 0XFF。常用的标记码的结构及其含义如下：

● SOI(start of image)

标记结构	字节数
0XFF	1
0XD8	1

● APP0(application)

标记结构	字节数	意义
0XFF	1	
0XE0	1	
Lp	2	APP0 标记码长度
Identifier	5	JFIF 识别码
Version	2	JFIF 版本号
Units	1	单位
Xdensity	2	水平分辨率
Ydensity	2	垂直分辨率
Xthumbnail	1	水平点数
Ythumbnail	1	垂直点数
RGB0	3	RGB 的值
RGB1	3	RGB 的值
…		
RGBn	3	RGB 的值

APP0 是 JPEG 保留给 application 使用的标记码，而 JFIF 将文件的相关信息定义在此标记中。

● DQT(define quantization table)

标记结构	字节数	意义
0XFF	1	
0XDB	1	
Lq	2	DQT 标记码长度
(Pq,Tq)	1	在基本系统中，Pq=0,Tq=0~1
Q0	1 或 2	量化表的值
Q1	1 或 2	量化表的值
…		
Qn	1 或 2	量化表的值

n 的值为 0～63，表示量化表中的 64 个值，Pq = 0, 为一个字节；Pq = 1, 为两个字节。

● DRI(define restart interval)

标记结构	字节数	意义
0XFF	1	
0XDD	1	
Lr	2	DRI 标记码长度
Ri	2	重入间隔的 MCU 个数

● DHT(define huffman table)

标记结构	字节数	意义
0XFF	1	
0XC4	1	
Lh	2	DHT 标记码长度
(Tc,Th)	1	
L1	1	
…		
L16	1	
V1	1	
…		
Vt	1	

● SOF(start of frame)

标记结构	字节数	意义
0XFF	1	
0XC0	1	
Lf	2	SOF 标记码长度
P	1	基本系统中为 8
Y	2	图像高度
X	2	图像宽度
Nf	1	为 1 代表灰度图 为 3 代表彩色图
C1	1	成分编号 1
(H1,V1)	1	第一个采样因子
Tq1	1	该量化表编号
…		
Cn	1	成分编号 n
(Hn,Vn)	1	第 n 个采样因子
Tqn	1	该量化表编号

在基本系统中，Tc 为 0 指 DC 所用的 Huffman 表、Tc 为 1 指 AC 所用的 Huffman 表；Th 的值为 0 或 1，2 Tc + Th 表示 Huffman 表(最多有 4 个)的编号。Ln 表示每个 n 位 Huffman 码字的个数，n = 1 ~ 16。Vt 表示每个 Huffman 码字所对应的值。t = L1 + L2 + … + L16

● SOS(start of scan)

标记结构	字节数	意义
0XFF	1	
0XDA	1	
Ls	2	SOS 标记码长度
Ns	1	
Cs1	1	
(Td1,Ta1)	1	
…		
CsNs		
(TdNs, TaNs)		
Ss		
Se		
(Ah,Al)		

Ns 为 scan 中成分的个数，CsNs 为在 scan 中的编号。TdNs 为高 4 位，TaNs 为低 4 位，分别表示 DC 和 AC 编码表的编号。基本系统中，Ns = Nf，Ss、Se、Ah、Al 均为 0。

● EOI(end of image)

标记结构	字节数
0XFF	1
0XD9	1

二、JPEG 基本系统解码器

JPEG 基本系统解码器流程图 1 所示

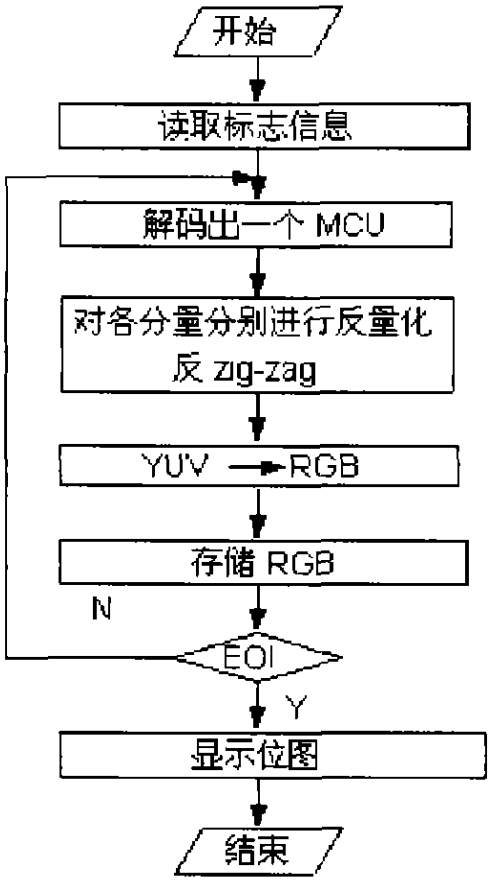

图 1 JPEG 基本系统解码器流程图

● 读取标志信息

包括读取量化表、水平取样因子、垂直取样因子以及 Huffman 表等。JPEG 使用的是 YUV 格式，Y 代表亮度信息，UV 代表色差信息。量化表是一个 8×8 的矩阵数据，有两个，一个是专门处理亮度的频率系数，另一个是针对色度的频率系数。Huffman 表有四个，其中两个对应亮度的 DC 值和 63 个 AC 值，另两个针对色度的 DC 值和 63 个 AC 值。

● 解码出一个 MCU

先介绍 MCU(最小编码单元)的概念。Y 分量的数据比较重要，而 U、V 分量的数据相对不重要，所以可以只取 U、V 的一部分，以增加压缩比。通常的两种取样方式是 YUV411 和 YUV422。如果 Y 取四个数据单元，即水平取样因子 Hy 乘以垂直取样因子 Vy 的值为 4，而 U 和 V 各取一个数据单元，即 Hu × Vu = 1，Hv × Vv = 1，那么这种取样就称做 YUV411。对于压缩数据，以 MCU 为单位，先进行 Huffman 解码，然后进

行逆离散余弦变换。本文采用快速逆离散余弦变换整数算法，详见程序。

● 对各分量分别进行反量化、反 zig－zag

对解码出的各分量分别乘以相应的亮化矩阵对应的系数，即为反炼化。由于数据数据是按照 8×8 矩阵的“之”字行排列，所以要进行反 zig－zag。

● YUV 转换为 RGB

JPEG 使用的是 YUV 格式，所以要将 YUV 转换为 RGB 才可以显示。对应关系如下：

R = Y + 1.402 · V

G = Y − 0.34414 · U − 0.71414 · V

B = Y + 1.1772 · U

三、JPEG 图像文件格式显示

本程序用 Turbo C2.0 在标准模式下开发而成。程序将文件 Flower.jpg 进行解压，生成位图文件 Flower.bmp，对于位图文件，很容易实现 DOS 和 WINDOWS 下的快速显示。限于篇幅，本文略出快速显示位图文件的部分，读者可以用 WIN95 的画图查看 Flower.bmp。本程序在 Turbo C2.0 Small 模式下编译通过。程序代码如下：

```
#include <stdio.h>
#include <alloc.h>
#include <math.h>
#define PI 3.1415927
#define widthbytes(il ((i+31)/32*4)
int sampleYH, sampleYV, sampleUH, sampleUV, sampleVH, sampleVV,
int HYtoU, VYtoU, HYtoV, VYtoV, YinMCU, UinMCU, VinMCU,
int compressnum=0, Qt[3][64], * YQt, * UQt, * VQt,
codepos[4][16], codelen[4][16],
unsigned char compressindex[3], YDCindex, YACindex, UVDCindex, UVACindex
unsigned char HufTabindex, And[9]={0, 1, 3, 7, 0xf, 0x1f, 0x3f, 0x7f, 0xff};
unsigned int codevalue[4][256], hufmax[4][16], hufmin[4][16],
int bitpos=0, curbyte=0, run=0, value=0, MCUbuffer[10*64], blockbuffer[64],
int ycoef=0, ucoef=0, vcoef=0, intervalflag=0, interval=0, restart=0,
long Y[4*64], U[4*64], V[4*64], QtZMCUbuffer[10*64],
unsigned long imgwidth=0 imgheight=0, width=0, height=0 linebytes,
int Z[8][8]={{0, 1, 5, 6, 14, 15, 27, 28}, {2, 4, 7, 13, 16, 26, 29, 42}, {3, 8, 12, 17, 25, 30, 41, 43}, {9, 11, 18, 24, 31, 40, 44, 53}, {10, 19, 23, 32, 39, 45, 52, 54}, {20, 22, 33, 38, 46, 51, 55, 60}, {21, 34, 37, 47, 50, 56, 59, 61}, {35, 36, 48, 49, 57, 58, 62, 63}},
struct{
 unsigned char type[2],
 long size,
 long reserved,
 long offset,
 }head,
struct{
 long size,
 long width,
 long height,
 int plane,
 int bitcount,
 long compression,
 long imagesize
 long xpels,
 long ypels,
 long coloruse,
 long colorimportant;
 }bmp,
void error(char *s)
 {
 printf("%s\n",s),
 exit(1),
 }
void makebmpheader(FILE *fp)
 {
 int i,j,
 unsigned long colorbits, imagebytes,
 colorbits=24,
 linebytes=widthbytes(colorbits*imgwidth),
 imagebytes=(unsigned long) imgheight*linebytes,
 head type[0]='B', head type[1]='M',
 head size=imagebytes+0x36,
 head reserved=0,
 head offset=0x36;
 fwrite(&head, sizeof(head), 1, fp),
 bmp size=0x28,
 bmp width=(long)imgwidth,
 bmp height=(long)imgheight,
 bmp plane=1L,
 bmp bitcount=colorbits,
 bmp compression=0,
 bmp imagesize=imagebytes,
 bmp xpels=0xece,
 bmp ypels=0xec4,
 bmp coloruse=0,
 bmp colorimportant=0
 fwrite(&bmp, sizeof(bmp), 1, fp),
 for(j=0, j<imgheight, j++)
  for(i=0, i<linebytes, i++)
   fputc(0, fp),
 }
```

```
void initialize(FILE *fp)
{
 unsigned char *p, *q, hfindex, qtindex, number,
 int i, j, k, finish=0, huftab1, huftab2, huftabindex, count,
 unsigned int length, flag,
 fread(&flag, sizeof(unsigned int), 1, fp),
 if(flag!=0xd8ff)
  error("Error Jpg File format!"),
 while(!finish) {
  fread(&flag, sizeof(unsigned int), 1, fp),
  fread(&length, sizeof(int), 1, fp),
  length=((length<<8)|(length>>8))-2,
  switch(flag) {
   case 0xe0ff
      fseek(fp, length, 1), break,
   case 0xdbff
      p=malloc(length),
      fread(p, length, 1, fp),
      qtindex=(*p)&0x0f,
      q=p+1,
      if(length-2<80)
       for(i=0, i<64, i++)
        Qt[qtindex][i]=(int)*(q++),
      else {
       for(i=0, i<64, i++)
        Qt[qtindex][i]=(int)*(q++),
       qtindex=*(q++)&0x0f,
       for(i=0, i<64, i++)
        Qt[qtindex][i]=(int)*(q++),
      }
      free(p), break,
   case 0xc0ff
      p=malloc(length),
      fread(p, length, 1, fp),
      mgheight=((*(p+1))<<8)+*(p+2),
      mgwidth=(*(p+3))<<8)+*(p+4),
      compressnum=*(p+5),
      if(!(compressnum!=1)&&(compressnum!=3))
       error("Error Jpg File format!"),
      if(compressnum==3) {
       compressindex[0]=*(p+6),
       sampleYH=(*(p+7))>>4,
       sampleYV=(*(p+7))&0x0f,
       YQt=(int *)Qt[*(p+8)],
       compressindex[1]=*(p+9),
       sampleUH=(*(p+10))>>4,
       sampleUV=(*(p+10))&0x0f,
       UQt=(int *)Qt[*(p+11)],
       compressindex[2]=*(p+12),
       sampleVH=(*(p+13))>>4,
       sampleVV=(*(p+13))&0x0f,
       VQt=(int *)Qt[*(p+14)],
      }
      else {
       compressindex[0]=*(p+6),
       sampleYH=(*(p+7))>>4,
       sampleYV=(*(p+7))&0x0f,
       YQt=(int *)Qt[*(p+8)],
       compressindex[1]=*(p-6),
       sampleUH=1,
       sampleUV=1,
       UQt=(int *)Qt[*(p+8)],
       compressindex[2]=*(p+6),
       sampleVH=1,
       sampleVV=1,
       VQt=(int *)Qt[*(p+8)],
      }
      free(p) break,
   case 0xc4ff
    p=malloc(length+1),
    fread(p, length, 1, fp),
    p[length]=0xff,
    if(length+2<0xd0) {
     huftab1=(int)(*p)>>4,
     huftab2=(int)(*p)&0x0f,
     huftabindex=huftab1*2+huftab2,
     q=p+1
     for(i=0, i<16, i++)
      codelen[huftabindex][i]=(int)(*(q++)),
     j=0,
     for(i=0, i<16, i++)
      if(codelen[huftabindex][i]!=0) {
       k=0,
       while(k<codelen[huftabindex][i]) {
        codevalue[huftabindex][k++]=(int)*(q++),
        k++,
       }
       j+=k,
      }
     i=0,
     while(codelen[huftabindex][i]==0) i++,
     for(k=0, k<i, j++) {
      huffmin[huftabindex][j]=0,
      huffmax[huftabindex][j]=0,
     }
     huffmin[huftabindex][i]=0,
     huffmax[huftabindex][i]=codelen[huftabindex][i]-1,
     for(j=i+1, j<16, j++) {
      huffmin[huftabindex][j]=(huffmax[huftabindex][j-1]
+1)<<1,
      huffmax[huftabindex][j]=huffmin[huftabindex][j]+
codelen[huftabindex][j]-1,
     }
     codepos[huftabindex][0]=0,
```

```
        for(j=1,j<16,j++)
         codepos[huftabindex][j]=codelen[huftabindex][j-
1]+codepos[huftabindex][j-1];
       }
       else {
        hfindex=*p,
        while(hfindex!=0xff) {
         huftab1=(int)hfindex>>4,
         huftab2=(int)hfindex&0x0f,
         huftabindex=huftab1*2+huftab2,
         q=p-1,count=0,
         for(i=0,i<16,i++) {
          codelen[huftabindex][i]=(int)(*(q++)),
          count+=codelen[huftabindex][i],
         }
         count+=17,j=0,
         for(i=0,i<16,i++)
          if(codelen[huftabindex][i]!=0) {
           k=0,
           while(k<codelen[huftabindex][i]) {
            codevalue[huftabindex][k+j]=(int)(*(q++));
            k++
           }
           j+=k,
          }
          i=0,
          while(codelen[huftabindex][i]==0) i++,
          for(j=0,j<i,j++) {
           hufmin[huftabindex][j]=0,
           hufmax[huftabindex][j]=0;
          }
          hufmin[huftabindex][i]=0,
        hufmax[huftabindex][i]=codelen[huftabindex][i]-1,
          for(j=i+1,j<16,j++) {
       hufmin[huftabindex][j]=(hufmax[huftabindex][j-1]
+1)<<1,
       hufmax[huftabindex][j]=hufmin[huftabindex][j]+
codelen[huftabindex][j]-1,
          }
          codepos[huftabindex][0]=0,
          for(j=1,j<16,j++)
       codepos[huftabindex][j]=codelen[huftabindex][j-1]
+codepos[huftabindex][j-1],
          p+=count,
          hfindex=*p,
         }
         p+=length,
        }
        free(p),break,
    case 0xddff
        p=malloc(length),
        fread(p,length,1,fp),
        restart=((*p)<<8)|*(p+1)),
        free(p),break,
    case 0xdaff
    p=malloc(length*sizeof(unsigned char)),
    fread(p,length,1,fp),
    number=*p,
    if(number!=compressnum)
     error("Error Jpg File format!"),
    q=p+1,
    for(i=0,i<compressnum,i++) {
     if(*q==compressindex[0]) {
      YDCindex=*(q+1))>>4,
      YACindex=((*(q+1))&0x0f)+2,
     }
     else {
      UVDCindex=*(q+1))>>4,
      UVACindex=((*(q+1))&0x0f)+2,
     }
     q+=2
    }
    finish=1,free(p),break,
    case 0xd9ff
    error("Error Jpg File format!"),
        break,
    default
        if((flag&0xf000)!=0xd000)
     fseek(fp,length,1),
        break,
   }
  }
 }
void savebmp(FILE *fp)
 {
  int i,j,
  unsigned char r,g,b,
  long y,u,v,rr,gg,bb,
  for(i=0,i<sampleYV*8,i++) {
   if((height+i)<imgheight) {
    fseek(fp,(unsigned long)(imgheight-height-i-1)*
linebytes+3*width+54,0),
    for(j=0,j<sampleYH*8,j++) {
     if((width+j)<imgwidth) {
      y=Y[i*8*sampleYH+j],
      u=U[(i/VYtoU)*8*sampleYH+j/HYtoU],
      v=V[(i/VYtoV)*8*sampleYH+j/HYtoV],
      rr=((y<<8)+359*v)>>8,
      gg=((y<<8)-88*u-183*v)>>8,
      bb=((y<<8)+301*u)>>8,
      r=(unsigned char)rr,
      g=(unsigned char)gg,
      b=(unsigned char)bb,
      if(rr&0xffffff00)
```

```
    if (rr>255) r=255; else if (rr<0) r=0;
    if(gg&0xffffff00)
     if (gg>255) g=255; else if (gg<0) g=0;
    if(bb&0xffffff00)
     if (bb>255) b=255; else if (bb<0) b=0;
    fputc(b,fp); fputc(g,fp); fputc(r,fp);
   }
   else break;
  }
 }
 else break;
 }
}
unsigned char readbyte(FILE *fp)
{
 unsigned char c;
 c=fgetc(fp);
 while(c==0xff)
  fgetc(fp);
 bitpos=8; curbyte=c;
 return c;
}
int DecodeElement(FILE *fp)
{
 int codelength;
 long thiscode,tempcode;
 unsigned int temp,new;
 unsigned char hufbyte,runsize,tempsize,sign;
 unsigned char newbyte,lastbyte;
 if(bitpos>=1) {
  bitpos--;
  thiscode=(unsigned char)curbyte>>bitpos;
  curbyte=curbyte&And[bitpos];
 }
 else {
  lastbyte=readbyte(fp);
  bitpos--;
  newbyte=curbyte&And[bitpos];
  thiscode=lastbyte>>7;
  curbyte=newbyte;
 }
 codelength=1;
 while((thiscode<hufmin[HufTabindex][codelength-1])||
(codelen[HufTabindex][codelength-1]==0)||(thiscode>
hufmax[HufTabindex][codelength-1])) {
  if(bitpos>=1) {
   bitpos--;
   tempcode=(unsigned char)curbyte>>bitpos;
   curbyte=curbyte&And[bitpos];
  }
  else {
   lastbyte=readbyte(fp);
   bitpos--;
   newbyte=curbyte&And[bitpos];
   tempcode=(unsigned char)lastbyte>>7;
   curbyte=newbyte;
  }
  thiscode=(thiscode<<1)+tempcode;
  codelength++;
  if(codelength>16)
   error("Error Jpg File format!");
 }
 temp=thiscode-hufmin[HufTabindex][codelength-1]+
codepos[HufTabindex][codelength-1];
 hufbyte=(unsigned char)codevalue[HufTabindex][temp];
 run=(int)(hufbyte>>4);
 runsize=hufbyte&0x0f;
 if(runsize==0) {
  value=0;
  return 1;
 }
 tempsize=runsize;
 if(bitpos>=runsize) {
  bitpos-=runsize;
  new=(unsigned char)curbyte>>bitpos;
  curbyte=curbyte&And[bitpos];
 }
 else {
  new=curbyte;
  tempsize-=bitpos;
  while(tempsize>8) {
   lastbyte=readbyte(fp);
   new=(new<<8)+(unsigned char)lastbyte;
   tempsize-=8;
  }
  lastbyte=readbyte(fp);
  bitpos-=tempsize;
  new=(new<<tempsize)+(lastbyte>>bitpos);
  curbyte=lastbyte&And[bitpos];
 }
 sign=new>>(runsize-1);
 if(sign)
  value=new;
 else {
  new=new^0xffff;
  temp=0xffff<<runsize;
  value=-(int)(new^temp);
 }
 return 1;
}
int HufBlock(FILE *fp,unsigned char achufindex,unsigned
char achufindex)
{
 int i,count=0;
```

```
HufTabindex = dchufindex,
if(DecodeElement(fp)! = 1)
 return 0,
blockbuffer[count + + ] = value,
HufTabindex = achufindex,
while (count < 64)  {
 if(DecodeElement(fp)! = 1)
  return 0,
 if((run = = 0)&&(value = = 0))  {
  for(i = count, i < 64, i + + )
   blockbuffer[i] = 0,
  count = 64,
 }
 else {
  for(i = 0, i < run, i + + )
   blockbuffer[count + + ] = 0,
  blockbuffer[count + + ] = value
 }
}
return 1,
}
int DecodeMCUBlock(FILE *fp)
{
 int i, j,  pMCUBuffer,
 if(intervalflag)  {
  fseek(fp, 2, 1),
  ycoef = ucoef = vcoef = 0,
  bitpos = 0, curbyte = 0,
 }
 switch(compressnum)  {
  case 3:
   pMCUBuffer = MCUbuffer,
   for(i = 0, i < sampleYH * sampleYV, i + + )  {
    if(HufBlock(fp, YDCindex, YACindex)! = 1)
     return 0,
    blockbuffer[0] = blockbuffer[0] + ycoef,
    ycoef = blockbuffer[0],
    for(j = 0, j < 64, j + + )
     *pMCUBuffer + + = blockbuffer[j],
    }
    for(i = 0, i < sampleUH  sampleUV, i + + )  {
    if(HufBlock(fp, UVDCindex, UVACindex)! = 1)
     return 0,
    blockbuffer[0] = blockbuffer[0] + ucoef,
    ucoef = blockbuffer[0],
    for(j = 0, j < 64, j + + )
     *pMCUBuffer + + = blockbuffer[j],
    }
    for(i = 0, i < samoleVH * sampleVV, i + + )  {
   if(HufBlock(fp, UVDCindex, UVACindex)! = 1)
        return 0,
    blockbuffer[0] = blockbuffer[0] + vcoef,
    vcoef = blockbuffer[0],
    for(j = 0, j < 64, j + + )
     pMCUBuffer + + = blockbuffer[j],
    }
    break,
  case 1:
   pMCUBuffer = MCUbuffer,
   if(HufBlock(fp, YDCindex, YACindex)! = 1)
    return 0,
   blockbuffer[0] = blockbuffer[0] + ycoef,
   ycoef = blockbuffer[0],
   for(j = 0, j < 64, j + + )
    *pMCUBuffer + + = blockbuffer[j],
   for(i = 0, i < 128, i + + )
    *pMCUBuffer + + = 0
   break
  default:
   error("Error Jpg File format"),
 }
 return(1),
}
void idct(long  p, int k)
{
 long x, x0, x1, x2, x3, x4, x5, x6, x7,
 x1 = p[k + 4] < < 11,  x2 = p[k + 6],  x3 = p[k + 2],
 x4 = p[k + 1],  x5 = p[k + 7],  x6 = p[k + 5],
 x7 = p[k + 3],  x0 = (p[0] < < 11) + 1024,
 x = 565 * (x4 + x5), x4 = x + 2276 * x4, x5 = x - 3406 * x5,
 x = 2408 * (x6 + x7), x6 = x - 799 * x6, x7 = x - 4017 * x7,
 x = 1108 * (x3 + x2), x2 = x - 3784 * x2, x3 = x + 1568 * x3,
 x = x6, x6 = x5 - x7, x5 = x + x5, x7 = x0 + x1, x0 = x1,
 x1 = x + x4, x4 = x, x = x5, x5 = x7 - x3, x7 = x + x3,
 x3 = x0 + x2, x0 = x2,  x2 = (181 * (x4 + x) + 128)>>8,
 x4 = (181 * (x4 - x) + 128)>>8,  p[0] = (x7 + x1)>>11,
 p[k + 1] = (x3 + x2)>>11,
 p[k + 2] = (x0 + x4)>>11, p[k + 3] = (x5 + x6)>>11,
 p[k + 4] = (x5 - x6)>>11, p[k + 5] = (x0 - x4)>>11,
 p[k + 6] = (x3 - x2)>>11, p[k + 7] = (x7 - x1)>>11,
}
void IDCTint(long  metrix)
{
 int i,
 for(i = 0, i < 8, i + + )
  idct(metrix + 8 * i, 1),
 for(i = 0, i < 8, i + + )
  idct(metrix + i, 8),
}
void IQtZBlock(int *s, long  d, int *pQt, int correct)
{
 int i, j, tag;
 long *pbuffer, buffer[8][8],
 for(i = 0,  < 8, i + + )
```

```
  for( =0., <8, + +) {
   tag =Z[i][j],
   buffer[i][j] =(long)s[tag] * (long)pQt[tag],
  }
 pbuffer = (long *)buffer,
 IDCTint(pbuffer)
 for( =0, i<8, i + +)
  for(j=0, <8, j + +)
   d[i*8+j] = (buffer[i][j]>>3) + correct,
}
void IQtZMCU(int xx, int yy, int offset, int *pQt, int correct)
{
 int i, j, *pMCUBuffer,
 long *pQtZMCUBuffer,
 pMCUBuffer = MCUbuffer + offset,
 pQtZMCUBuffer = QtZMCUbuffer + offset;
 for(i=0, i<yy, i + +)
  for(j=0, j<xx, j + +)
    IQtZBlock(pMCUBuffer + (i * xx +j) * 64, pQtZM-
CUBuffer + (i * xx +j) * 64, pQt, correct),
}
void getYUV(int xx, int yy, long *buf, int offset)
{
 int i, j, k, n,
 long *pQtZMCU,
 pQtZMCU = QtZMCUbuffer + offset,
 for( =0, i<yy, i + +)
  for(j=0, j<xx, j + +)
   for(k=0, k<8, k + +)
    for(n=0, n<8, n + +)
buf[(i*8+k)*sampleYH*8+j*8+n] = *pQtZMCU + +,
}
void decode(FILE *fp1, FILE *fp2)
{
 int Yinbuf, Uinbuf, Vinbuf,
 YinMCU = sampleYH * sampleYV,
 UinMCU = sampleUH * sampleUV,
 VinMCU =sampleVH * sampleVV,
 HYtoU = sampleYH/sampleUH,
 VYtoU =sampleYV/sampleUV
 HYtoV =sampleYH/sampleVH,
 VYtoV = sampleYV/sampleVV,
 Yinbuf =0, Uinbuf =YinMCU * 64,
 Vinbuf = (YinMCU + UinMCU) * 64,
 while(DecodeMCUBlock(fp1)) {
  interval + +
  if((restart)&& (interval%restart = =0))
   intervalflag = 1,
  else
   intervalflag =0,
IQtZMCU(sampleYH, sampleYV, Yinbuf, YQt, 128),
IQtZMCU(sampleUH, sampleUV, Uinbuf, UQt, 0),
IQtZMCU(sampleVH, sampleVV, Vinbuf, VQt, 0),
  getYUV(sampleYH, sampleYV, Y, Yinbuf);
  getYUV(sampleUH, sampleUV, U, Uinbuf);
  getYUV(sampleVH sampleVV, V, Vinbuf),
  savebmp(fp2),
  width + =sampleYH * 8,
  if(width>=imgwidth) {
   width =0,
   height + =sampleYV * 8,
  }
  if((width = =0)&& (height>=imghe ght)}
   break,
 }
}
main( )
{
 FILE *fp1, *fp2,
 if((fp1 =fopen("flower.jpg", "rb")) = =NULL)
  error("Can not open Jpg File! "),
 if((fp2 =fopen("flower bmp", "wb")) = =NULL)
  error("Can not create Bmp File! "),
 initialize(fp1),
 makebmpheader(fp2), decode(fp1, fp2),
 fclose(fp1), fclose(fp2),
}
```

参考文献

1. 吕凤军编著．数字图象处理编程入门．清华大学出版社

2. 李振辉、李仁和编著．探索图象文件的奥秘．清华大学出版社

3. 马小虎等编著．多媒体数据压缩标准及实现．清华大学出版社

（收稿日期：2001 年 2 月 8 日）