《JPEG解码实验》实验报告

实验人员：侯明 指导老师：张远 时间: 2012/5/12

1. 实验目的
2. 深入理解变换,量化,熵编码的原理.
3. 了解DCT变换,及其系数的量化.并分析DCT变换与DFT变换的在边界处的差异.
4. 熟悉jpeg文件格式标准.
5. 深入理解实验提供的代码.包括,理解程序设计的整体框架

理解三个结构体的设计目的:

struct huffman\_table

struct component

struct jdec\_private

理解在视音频编解码调试中TRACE的目的和含义,会打开和关闭TRACE,会根据自己的要求修改TRACE.

1. 实验原理
2. DCT变换
3. DC系数的差分编码

DC系数的特点:数的数值比较大;相邻8×8图像块的DC系数值变化不大(冗余). 根据这个特点，JPEG算法使用了差分脉冲调制编码(DPCM)技术，对相邻图像块之间量化DC系数的差值DIFF进行编码.

1. AC系数的编码
2. Zig-zag扫描

由于经DCT变换后，系数大多数集中在左上角，即低频分量区，因此采用Z字形按频率的高低顺序读出，可以出现很多连零的机会。可以使用游程编码。尤其在最后，如果都是零，给出EOB (End of Block)即可。

1. 游程编码

在JPEG和MPEG编码中规定为：(run, level)

表示连续run个0，后面跟值为level的系数

如：0，2，0，0，3，0，-4，0，0，0，-6，0，0，5，7

表示为(1, 2)， (2, 3) ，…

编码：

Run: 最多15个，用4位表示RRRR

Level：类似DC

分成16个类别，用4位表示SSSS表示类别号

类内索引

对(RRRR, SSSS)联合用Huffman编码

对类内索引用定长码编码

1. JPEG的文件组织

JPEG在文件中以Segment的形式组织，它具有以下特点：

均以0xFF开始，后跟1 byte的Marker和2 byte的Segment length（包含表示Length本身所占用的2 byte，不含“0xFF” + “Marker” 所占用的2 byte）；采用Motorola序（相对于Intel序），即保存时高位在前，低位在后；Data部分中，0xFF后若为0x00，则跳过此字节不予处理

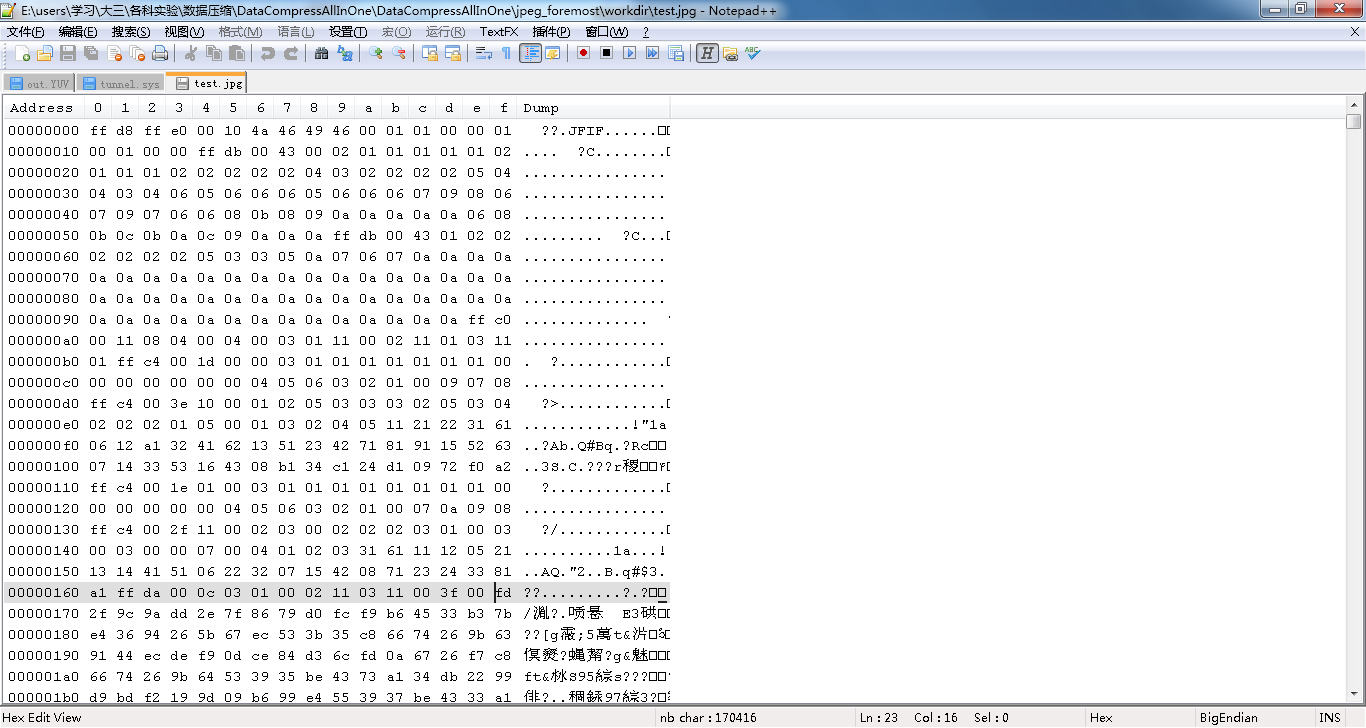
1. 代码分析 未完成

代码核心是解码器,即结构体struct jdec\_private.面向该结构体的解码器jdec,程序的过程大致为:

1. 调用函数tinyjpeg\_init(…),先于内存上分配一个jdec空间.
2. 调用函数tinyjpeg\_parse\_header(…),初始化jdec.

完成内存分配后,解码器将对特定的jpeg文件进行解码.由于jpeg文件格式之间也存在差异.例如,……  
于是再要对jdec进行内存初始化.该初始化过程,与jpeg头部信息有关.下面对jpeg头不的部分块做出解释.

截取部分jpeg头部,以二进制表示如下:



jpeg文件,采用块形式组织.简单解释,就是以0xff为块头开始的标志,接着的字节,描述该块的类型,再接着的是payload.payload开始处两字节描述payload的长度.针对该形式的块,可用while循环来遍历文件.不断读出相应的块做需要的处理.

块类型描述符如附表1所示.

下图为tinyjpeg\_parse\_header的流程

tinyjpeg\_parse\_header

Identify the file

parse\_JFIF

验证头部SOI

以块为单位,遍历缓冲区

头部marker:

DQT = 0xDB, /\* Define Quantization Table \*/

SOF = 0xC0, /\* Start of Frame (size information) \*/

DHT = 0xC4, /\* Huffman Table \*/

SOI = 0xD8, /\* Start of Image \*/

SOS = 0xDA, /\* Start of Scan \*/

RST = 0xD0, /\* Reset Marker d0 -> .. \*/

RST7 = 0xD7, /\* Reset Marker .. -> d7 \*/

EOI = 0xD9, /\* End of Image \*/

DRI = 0xDD, /\* Define Restart Interval \*/

APP0 = 0xE0,

1. 函数build\_quantization\_table(…),建立量化表函数.

建立过程,zigzag顺序,从量化系数表buffer中读出...没能理解,量化表的矩阵运算的原理.写不出来.

量化系数表:

DC\_Q\_TABLE:

2.000000 1.387040 1.306563 2.351751 2.000000 3.142780 2.705981 1.655396

1.387040 1.923880 1.812255 3.261973 4.161119 6.538741 4.503963 2.296101

1.306563 1.812255 3.414214 3.072711 5.226252 6.159359 4.949748 2.162879

1.175876 3.261973 3.072711 4.148050 5.879378 8.314916 5.091034 1.946540

2.000000 2.774080 5.226252 7.055254 7.000000 8.642645 5.411961 2.207195

1.571390 4.359161 6.159359 5.543277 6.285560 6.173166 4.677365 1.950955

2.705981 4.503963 5.656854 5.727414 5.411961 5.102581 3.514719 1.493157

1.931296 3.444151 3.604799 3.244233 3.034893 2.167727 1.493157 0.761205

AC\_Q\_TABLE:

2.000000 2.774080 2.613126 5.879378 10.000000 7.856950 5.411961 2.758994

2.774080 3.847759 5.436765 11.416904 13.870399 10.897902 7.506606 3.826834

2.613126 5.436765 10.242640 15.363555 13.065630 10.265599 7.071068 3.604799

5.879378 11.416904 15.363555 13.826835 11.758756 9.238795 6.363793 3.244233

10.000000 13.870399 13.065630 11.758756 10.000000 7.856950 5.411961 2.758994

7.856950 10.897902 10.265599 9.238795 7.856950 6.173166 4.252151 2.167727

5.411961 7.506606 7.071068 6.363793 5.411961 4.252151 2.928932 1.493157

2.758994 3.826834 3.604799 3.244233 2.758994 2.167727 1.493157 0.761205

1. 函数parse\_SOF(…)

Start of frame .该块保存了相应的 宽 高 分量数(测试图像为3分量)以及各种帧信息.

1. 函数parse\_DHT(…)

内存结构:

0x0402A839 ff c4 00 1d 00 00 03 01 01 01 01 01 01 01 00 00

0x0402A849 00 00 00 00 00 04 05 06 03 02 01 00 09 07 08 ff

0x0402A859 c4 00 3e 10 00 01 02 05 03 03 03 02 05 03 04 02

0x0402A869 02 02 01 05 00 01 03 02 04 05 11 21 22 31 61 06

0x0402A879 12 a1 32 41 62 13 51 23 42 71 81 91 15 52 63 07

0x0402A889 14 33 53 16 43 08 b1 34 c1 24 d1 09 72 f0 a2

读取payload部分前17个字节,既val的值.

/\* We need to calculate the number of bytes 'vals' will takes \*/

huff\_bits[0] = 0;

count = 0;

for (i=1; i<17; i++)

{

huff\_bits[i] = \*stream++;

count += huff\_bits[i];

}



count表示下一个部分的val个数.该实例中为: 04 05 06 03 02 01 00 09 07 08 共十个.

Index表示此时的码表类型,共分为,直流码表和交流码表.通过该值,调用相应的build\_huffman\_table函数,建立huffman码表.

1. 函数build\_huffman\_table(…).

累了,略.

附表1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol  (符号) | Code Assignment  (0xFF + Marker)  (标记代码) | | Description  (说明) | |
| Start Of Frame markers, non-hierarchical Huffman coding | | | | |
| SOF0 | 0xFFC0 | | Baseline DCT | |
| SOF1 | 0xFFC1 | | Extended sequential DCT | |
| SOF2 | 0xFFC2 | | Progressive DCT | |
| SOF3 | | 0xFFC3 | | Spatial (sequential) lossless | |
| Start Of Frame markers, hierarchical Huffman coding | | | | | |
| SOF5 | | 0xFFC5 | | Differential sequential DCT | |
| SOF6 | | 0xFFC6 | | Differential progressive DCT | |
| SOF7 | | 0xFFC7 | | Differential spatial lossless | |
| Start Of Frame markers, non-hierarchical arithmetic coding | | | | | |
| JPG | | 0xFFC8 | | Reserved for JPEG extensions | |
| SOF9 | | 0xFFC9 | | Extended sequential DCT | |
| SOF10 | | 0xFFCA | | Progressive DCT | |
| SOF11 | | 0xFFCB | | Spatial (sequential) Lossless | |
| Start Of Frame markers, hierarchical arithmetic coding | | | | | |
| SOF13 | | 0xFFCD | | Differential sequential DCT | |
| SOF14 | | 0xFFCE | | Differential progressive DCT | |
| SOF15 | | 0xFFCF | | Differential spatial Lossless | |
| Huffman table specification | | | | | |
| DHT | | 0xFFC4 | | Define Huffman table(s) | |
| arithmetic coding conditioning specification | | | | | |
| DAC | | 0xFFCC | | Define arithmetic conditioning table | |
| Restart interval termination | | | | | |
| RSTm | | 0xFFD0～0xFFD7 | | Restart with modulo 8 counter m | |
| Other marker | | | | | |
| SOI | | 0xFFD8 | | Start of image | |
| EOI | | 0xFFD9 | | End of image | |
| SOS | | 0xFFDA | | Start of scan | |
| DQT | | 0xFFDB | | Define quantization table(s) | |
| DNL | | 0xFFDC | | Define number of lines | |
| DRI | | 0xFFDD | | Define restart interval | |
| DHP | | 0xFFDE | | Define hierarchical progression | |
| EXP | | 0xFFDF | | Expand reference image(s) | |
| APPn | | 0xFFE0～0xFFEF | | Reserved for application use | |
| JPGn | | 0xFFF0～0xFFFD | | Reserved for JPEG extension | |
| COM | | 0xFFFE | | Comment | |
| Reserved markers | | | | | |
| TEM | | 0xFF01 | | For temporary use in arithmetic codi | |

1. 实验结果 未完成
2. HUFFMAN码表

在标记段DHT内，包含了一个或者多个的哈夫曼表。对于单一个哈夫曼表，应该包括了三部分：

* 哈夫曼表ID和表类型,这个字节的值为一般只有四个0x00、0x01、0x10、0x11:0x00,表示DC直流0号表；0x01,表示DC直流1号表；0x10,表示AC交流0号表；0x11,表示AC交流1号表。
* 不同位数的码字数量
* 编码内容

Huffman table DC[0] length=10

val=04 code=00000000 codesize=02

val=05 code=00000001 codesize=02

val=06 code=00000002 codesize=02

val=03 code=00000006 codesize=03

val=02 code=0000000e codesize=04

val=01 code=0000001e codesize=05

val=00 code=0000003e codesize=06

val=09 code=0000007e codesize=07

val=07 code=000000fe codesize=08

val=08 code=000001fe codesize=09

Huffman table DC[1] length=11

val=04 code=00000000 codesize=02

val=05 code=00000001 codesize=02

val=06 code=00000002 codesize=02

val=03 code=00000006 codesize=03

val=02 code=0000000e codesize=04

val=01 code=0000001e codesize=05

val=00 code=0000003e codesize=06

val=07 code=0000007e codesize=07

val=0a code=000000fe codesize=08

val=09 code=000001fe codesize=09

val=08 code=000003fe codesize=10

Huffman table AC[0] length=43

val=00 code=00000000 codesize=02

val=01 code=00000002 codesize=03

val=03 code=00000003 codesize=03

val=02 code=00000008 codesize=04

val=04 code=00000009 codesize=04

val=05 code=0000000a codesize=04

val=11 code=0000000b codesize=04

val=21 code=0000000c codesize=04

val=22 code=0000001a codesize=05

val=31 code=0000001b codesize=05

val=61 code=0000001c codesize=05

val=06 code=0000003a codesize=06

val=12 code=0000003b codesize=06

val=a1 code=0000003c codesize=06

val=32 code=0000007a codesize=07

val=41 code=0000007b codesize=07

val=62 code=0000007c codesize=07

val=13 code=000000fa codesize=08

val=51 code=000000fb codesize=08

val=23 code=000001f8 codesize=09

val=42 code=000001f9 codesize=09

val=71 code=000001fa codesize=09

val=81 code=000001fb codesize=09

val=91 code=000001fc codesize=09

val=15 code=000003fa codesize=10

val=52 code=000003fb codesize=10

val=63 code=000003fc codesize=10

val=07 code=000007fa codesize=11

val=14 code=000007fb codesize=11

val=33 code=000007fc codesize=11

val=53 code=000007fd codesize=11

val=16 code=00000ffc codesize=12

val=43 code=00000ffd codesize=12

val=08 code=00001ffc codesize=13

val=b1 code=00001ffd codesize=13

val=34 code=00003ffc codesize=14

val=c1 code=00003ffd codesize=14

val=24 code=00007ffc codesize=15

val=d1 code=0000fffa codesize=16

val=09 code=0000fffb codesize=16

val=72 code=0000fffc codesize=16

val=f0 code=0000fffd codesize=16

val=a2 code=0000fffe codesize=16

Huffman table AC[1] length=28

val=00 code=00000000 codesize=02

val=04 code=00000001 codesize=02

val=01 code=00000004 codesize=03

val=02 code=00000005 codesize=03

val=03 code=00000006 codesize=03

val=31 code=0000001c codesize=05

val=61 code=0000001d codesize=05

val=11 code=0000003c codesize=06

val=12 code=0000003d codesize=06

val=05 code=0000007c codesize=07

val=21 code=0000007d codesize=07

val=13 code=000000fc codesize=08

val=14 code=000000fd codesize=08

val=41 code=000000fe codesize=08

val=51 code=000001fe codesize=09

val=06 code=000007fc codesize=11

val=22 code=000007fd codesize=11

val=32 code=000007fe codesize=11

val=07 code=00001ffc codesize=13

val=15 code=00001ffd codesize=13

val=42 code=00001ffe codesize=13

val=08 code=0000fff8 codesize=16

val=71 code=0000fff9 codesize=16

val=23 code=0000fffa codesize=16

val=24 code=0000fffb codesize=16

val=33 code=0000fffc codesize=16

val=81 code=0000fffd codesize=16

val=a1 code=0000fffe codesize=16

1. 第3行，第4个8\*8块的量化后DCT, 反量化DCT和解码后的像素块值

解码后的像素值.

第1行,第1个8\*8块:

解码后的像素值,

量化DCT:

Y

303 12 0 1 0 0 0 0 10 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

U

-460 -4 0 0 0 0 0 0 -3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

V

-65 11 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

第3行，第4个8\*8块:

反量化DCT量化DCT 及真实值

真实值:

1. 实验心得