JPEG 编码实验报告

目录

一. 实验题目

编写一段程序，实现 JPEG 算法中的 8*8 的二维 DCT 变换、量化、逆量化和逆二维 DCT 变换。具体要求：逐个读入下列 4 个 8*8 的十六进制整数串，量化采用标准亮度量化表，输出内容（ASCII 码）同 8.3.3 例（原始数据、变换后的数据、量化表、量化后的数据、逆量化的数据和反变换的数据）

二. 实验结果

本程序使用 Python 2.7 实现，为了提升程序的运行性能，使用了 numpy。所以，运行程序前，请确保安装了 numpy library。

进入 src 目录，使用命令 python jpegCodec.py 即可运行程序，生成结果位于 output.txt 中。

```
donald@donald-Aspire-E1-471G:~/Code/multimedia/JPEGEncoder$ python jpegEncoder.py
Successfully extracted data from the input file!
Dealing with the image block 0......
Dealing with the image block 1......
Dealing with the image block 2......
Dealing with the image block 3......
Successfully finished! Open file 'output.txt' to check out the result
```

限于篇幅，下面仅贴出第一个图像块的结果，更多结果请打开 output.txt 文件查看。

```
The original data:
98 9C 96 99 9C A1 A1 A6
94 95 95 96 98 A0 A1 A7
95 94 91 94 9D A3 A9 A6
8D 92 8F 94 8F 8F 8C 87
7F 7C 7B 74 72 73 72 6F
5A 61 6A 5D 58 54 4D 49
6A 72 74 73 74 74 6F 70
77 7F 85 89 87 9A A2 A6
```

Figure 1 原始数据（10 进制）

```
Transform to decimal:
 152  156  150  153  156  161  161  166
 148  149  149  150  152  160  161  167
 149  148  145  148  157  163  169  166
 141  146  143  148  143  143  140  135
 127  124  123  116  114  115  114  111
  90   97  106   93   88   84   77   73
 106  114  116  115  116  116  111  112
 119  127  133  137  135  154  162  166
```

Figure 2 原始数据（16 进制）

```
Forward DCT:
  46.75  -19.17    0.80   -2.76   -8.50   -1.61    1.29   -3.25
 129.12   -5.04    8.98    7.94    5.58   -2.10   -2.90   -0.68
  62.60  -38.81    7.64   -3.81   -0.13   -2.08    1.06   -3.75
-105.71   39.42   -7.51   -3.51   -0.46   -3.35    3.05    2.86
  47.00   -5.71    5.18    0.10    0.25    2.38   -1.87   -3.86
   1.95    8.29   -9.63    2.20    1.69   -3.01   -3.79   -1.11
 -10.23   -5.44    1.06   -0.21   -2.92    1.34   -0.14    2.20
  16.92   -7.64    8.71    6.12    1.72   -0.71   -1.03    2.06
```

Figure 3 正向离散余弦变换结果

```
Quantizer(HUE_QUANTISER):
  16   11   10   16   24   40   51   61
  12   12   14   19   26   58   60   55
  14   13   16   24   40   57   69   56
  14   17   22   29   51   87   80   62
  18   22   37   56   68  109  103   77
  24   35   55   64   81  104  113   92
  49   64   78   87  103  121  120  101
  72   92   95   98  112  100  103   99
```

Figure 4 色差量化表

Quantization:

3	-2	0	0	0	0	0	0
11	0	1	0	0	0	0	0
4	-3	0	0	0	0	0	0
-8	2	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

Figure 5 量化后数据

Inverse Quantization:

48	-22	0	0	0	0	0	0
132	0	14	0	0	0	0	0
56	-39	0	0	0	0	0	0
-112	34	0	0	0	0	0	0
54	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

Figure 6 反量化后数据

Inverse DCT:

25.77	24.79	23.77	24.01	26.29	30.24	34.48	37.21
19.89	19.70	20.03	21.80	25.42	30.34	35.13	38.09
21.31	21.53	22.38	24.39	27.70	31.82	35.65	37.97
20.18	19.75	19.12	18.60	18.45	18.67	19.09	19.40
-2.93	-4.03	-6.22	-9.40	-13.24	-17.15	-20.39	-22.23
-30.70	-30.86	-31.63	-33.53	-36.72	-40.72	-44.47	-46.74
-28.76	-26.30	-22.42	-18.70	-16.37	-15.80	-16.38	-17.07
-7.28	-2.44	5.69	14.75	22.54	27.88	30.76	31.88

Figure 7 反向离散余弦变换数据

Reconstruct data:

153	152	151	152	154	158	162	165
147	147	148	149	153	158	163	166
149	149	150	152	155	159	163	165
148	147	147	146	146	146	147	147
125	123	121	118	114	110	107	105
97	97	96	94	91	87	83	81
99	101	105	109	111	112	111	110
120	125	133	142	150	155	158	159

Figure 8 重构图像数据

三. 实验分析

1. 正向离散余弦变换

公式：$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}\right]$$

由公式可看出，如果直接实现算法，需要四重循环，时间复杂度为 O(n^4)，这对于图像处理来说，是非常不理想的。所以，需要尽可能使用并行化的计算，更多的考虑矩阵运算实现，另外，对于大量重复计算的中间过程，可以考虑先计算并存储中间过程。

观察公式，可以考虑按照如下步骤实现

- 编写函数，根据所给的 u，生成 i 从 0 到 7 的 $\cos\frac{(2i+1)u\pi}{16}$ 的 list
- 对于给定的 u，v，利用上面生成的函数，生成两个 list。将第一个 list 转化为列向量，这样对两个 list 向量做向量乘法，就可以构造一个 cos 部分的 8x8 的矩阵
- 对于中括号中的部分，可以直接使用 f 矩阵和 cos 矩阵的 element-wise 的乘法，最后对得到的剧中求 sum 就可以了。
- 最后，用两层循环遍历 u，v，就可以求得结果了。

注意到，u 和 v 都只是在 0 到 7 之间取值，不断的重复计算可以避免。所以可以将它们先提前计算好，存储到一个数组中，如下面代码所示。

```python
def forward_dct(image):
    """
    return the numpy array of the forward DCT of the given image block
    image_block: numpy array with float data type
    """
    def generate_cos(idx):
        return [math.cos((2 * i + 1) * idx * math.pi / 16) for i in xrange(8)]

    image_block = image.astype(float) - 128.0
    fdct = np.zeros((8, 8))

    uv_cos = []
    for idx in xrange(8):
        uv_cos.append(generate_cos(idx))
    uv_cos = np.array(uv_cos, dtype=float)

    for u in xrange(8):
        for v in xrange(8):
            # construct a 8x8 matrix of the cos part
            cos_part = uv_cos[u, :].reshape((8, 1)) * uv_cos[v, :]
            # get the sum part
            sum_part = np.sum(np.multiply(image_block, cos_part))
            fdct[u, v] = 1.0 / 4 * c_func(u) * c_func(v) * sum_part

    return fdct
```

Figure 9 正向离散余弦变换函数

2. 反向离散余弦变换

公式：$f(i,j) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16}\right]$

观察上面的公式可以得到，对于任意的 i，j，中括号中的前三项都是一样的，所以可以先算出$C(u)C(v)F(u,v)$矩阵，然后类似正向离散余弦变化，构造出 cos 部分的矩阵，这样，遍历 i，j 就可以求得结果。如下代码所示。

```python
def inverse_dct(iqnt):
    def generate_cos(idx):
        return [math.cos((2 * idx + 1) * u * math.pi / 16) for u in xrange(8)]

    iqnt_f = iqnt.astype(float)

    c_list = [c_func(idx) for idx in xrange(8)]
    cc_part = np.array(c_list, dtype=float).reshape((8, 1)) * np.array(c_list, dtype=float)
    f_part = np.multiply(cc_part, iqnt_f)

    ij_cos = []
    for idx in xrange(8):
        ij_cos.append(generate_cos(idx))
    ij_cos = np.array(ij_cos, dtype=float)

    idct = np.zeros((8, 8))
    for i in xrange(8):
        for j in xrange(8):
            cos_part = ij_cos[i, :].reshape((8, 1)) * ij_cos[j, :]
            idct[i, j] = 1.0 / 4 * np.sum( np.multiply(f_part, cos_part) )

    return idct
```

Figure 10 反向离散和弦变换函数

3. 量化/反量化

按照课件公式实现即可。相对简单。

对于 round 部分实际上有不同的实现标准，不同标准会导致部分结果有所不同。本程序在实现中，使用了 numpy 提供的 round 函数，它根据的是 Banker's Rounding 策略，也就是将一半取为与其最接近的偶数(round half to even)，更多 round 规则，可参照 wikipedia（https://en.wikipedia.org/wiki/Rounding ）。

另外，本程序提供了色差量化表(HUE_QUANTISER)和亮度量化表(LUMINANCE_QUANTISER)。默认使用色差量化表，可以根据需求更改量化表，也可以传递自定义的二维 8x8 Python List 作为量化表参数。

四. 实验心得

对于图像的处理算法，由于需要进行大量的浮点数运算，如果使用本身运行性能较差的语言（如 Python, Matlab 等）实现，则需要着重考虑在实现方面进行优化。

对于 Python，可以考虑使用 numpy 库，然后尽最大可能使用矩阵运算，这与直接使用 for 循环实现，速度可以提升几十倍到上百倍，甚至更高。原因在于矩阵运算本身就是一个可以高度并行化的过程，而 numpy 库在这方面做了优化。

2016-10-20