# 媒体信号处理基础-实验报告4

学号：3170105728 姓名：林昭炜

## 实验内容及要求

实验工具：MATLAB 2017b

实验内容：

基于图像压缩基本流程，实现对彩色图像的压缩与恢复。

## 关键代码及注释

% … means some codes are omitted.

## Encode

% padding the image, 2\*N(8)to ensure that after down-sampling

% image size still a multiple of 8

padding = 2 \* N - mod(original\_size, 2 \* N);

padding = padding(1:2);

im = padarray(img, padding,'replicate', 'post');

im = rgb2ycbcr(im);

conv\_k = ones(2, 2, 'uint8');

% downsampling the image

% 对图片颜色进行下采样

color1 = my\_conv2(im(:, :, 2),conv\_k, 2) ./ 4;

color2 = my\_conv2(im(:, :, 3),conv\_k, 2) ./ 4;

% …

% my\_dct\_quan\_zigzag put three functions(dct, quantify, zigzag) into one

% first column of the result matrix is dc signal

% and the rest column is ac signal in accordance with dc

% 讲三个功能（dct，量化，展开）放一起实现，最终返回的矩阵是第一列是dc信号,

% 每一行后63个元素是ac信号，和第一列的dc信号对应

im\_dct\_quan\_zigzag = my\_dct\_quan\_zigzag(double(im(:, :, 1)), basis, quan\_lumi);

color\_zigzag = cat(3, my\_dct\_quan\_zigzag(color1, basis, quan\_color), my\_dct\_quan\_zigzag(color2, basis, quan\_color));

% dc signal use dpcm(which is actually pcm, because they have little

% influence on the final file size)

% dc信号用dpcm编码（实际上原函数用的是pcm，因为pcm和dpcm对最后的文件大小影响不大）

dc = cat(2, dpcm\_encode(im\_dct\_quan\_zigzag(:,1)), dpcm\_encode(color\_zigzag(:,1, 1)), dpcm\_encode(color\_zigzag(:,1, 2)));

% ac signal use rle, which encodes the luminous matrix & color matrices

% altogether

% ac信号使用游程编码, 将亮度和色彩一起编码

ac = rle\_encode(im\_dct\_quan\_zigzag(:,2:end), color\_zigzag(:,2:end,1), color\_zigzag(:,2:end,2));

signal = cat(2, dc, ac);

% …

% use getprob to get symbol table & its probability by traversing all

% elements

% getprob 通过遍历的方法获得symbol表和对应的概率

[sym, prob] = getprob(signal);

% header: image size, padding size

% table: luminous quantifier, color quantifier

% detail: dictionary size , data size

% dictionary: [symbol, dict\_size, dict\_data]

% data: huffman encoded data

fp = fopen('lena508\_510.leon', 'w');

% …

add = 8 - mod(bit\_cnt+comp\_size(2), 8);

% note that we are using bits to write file, but the final file size is

% multiple of bytes, here is trying to align bits to bytes

% 因为之前用bit写入, 但最终文件必须是bytes 的整数倍, 所以这里

% 通过添加无用比特位来对齐字节

comp = [comp, ones(1,add)];

## Decode

% calculate picture size

% 计算图片大小

pic\_size = original\_size(1:2) + padding;

% …

% dc\_decode = [[luminous], [color1], [color2]], the same applies to

% ac\_decode

dc\_decode = signal\_decode(1:dc\_len);

ac\_decode = signal\_decode(dc\_len+1:end);

% …

% seperating ac & dc and obtain original signal by dpcm\_decode(which actually is pcm\_decode)

lumi\_dc\_decode = dpcm\_decode(dc\_decode(1:lumi\_len));

color\_dc\_decode = cat(3, (dpcm\_decode(dc\_decode(lumi\_len+1:lumi\_len+color\_len)))', (dpcm\_decode(dc\_decode(lumi\_len+color\_len+1:end)))');

% …

% since dc&ac signal is encoded together, their rle may overlap each other,

% so the solution is keep track of how many are already decoded,

% here u is an index of the next rle code to be decodes

% and leftover is used to update how many 0s left,

% since last step may only process part of code at index u

[lumi\_ac\_decode, u, leftover] = rle\_decode(ac\_decode,[lumi\_len,63]);

ac\_decode = ac\_decode(u:end);

ac\_decode(1) = leftover;

[color1\_ac\_decode, u, leftover] = rle\_decode(ac\_decode,[color\_len,63]);

% …

% my\_dct\_quan\_zigzag\_decode is also a function that does 3 jobs fist

% unzigzag, then unquantify, and finally invert dct

lumi\_decode = uint8(my\_dct\_quan\_zigzag\_decode(lumi\_zigzag\_decode, basis, quan\_lumi, pic\_size));

% …

% do the invert desampling

color\_decode = my\_conv2\_invert(color\_conv\_decode, kernel, 2, pic\_size);

## functions

function arr = zigzag(A)

% …

% if it touches border, it will prepare to turn

if (row == 1 || row == A\_size(1)) && turn

col = col + 1;

flag = -flag;

turn = 0;

elseif (col == 1 || col == A\_size(2)) && turn

row = row + 1;

flag = -flag;

turn = 0;

% otherwise it will walk from left-bottom to top-right if flag = 1

% if flag = -1, it walks other direction

else

col = col + flag;

row = row - flag;

turn = 1;

% …

end

function res\_zigzag = my\_dct\_quan\_zigzag(A,basis, Q)

% basis & Q must be the same size

% …

% x1,y1 indicates the row position of the moving window

% x2,y2 specifies the column position

res(x1:y1,x2:y2) = basis \* A(x1:y1, x2:y2) \* basis'; % dct

res(x1:y1,x2:y2) = round(res(x1:y1, x2:y2) ./ Q); % quantify

res\_zigzag(cnt,:) = zigzag(res(x1:y1,x2:y2)); %zigzag

cnt = cnt + 1;

x2 = y2 + 1;

% …

end

附录：

im\_dct\_quan\_zigzag & color\_zigzag 图解

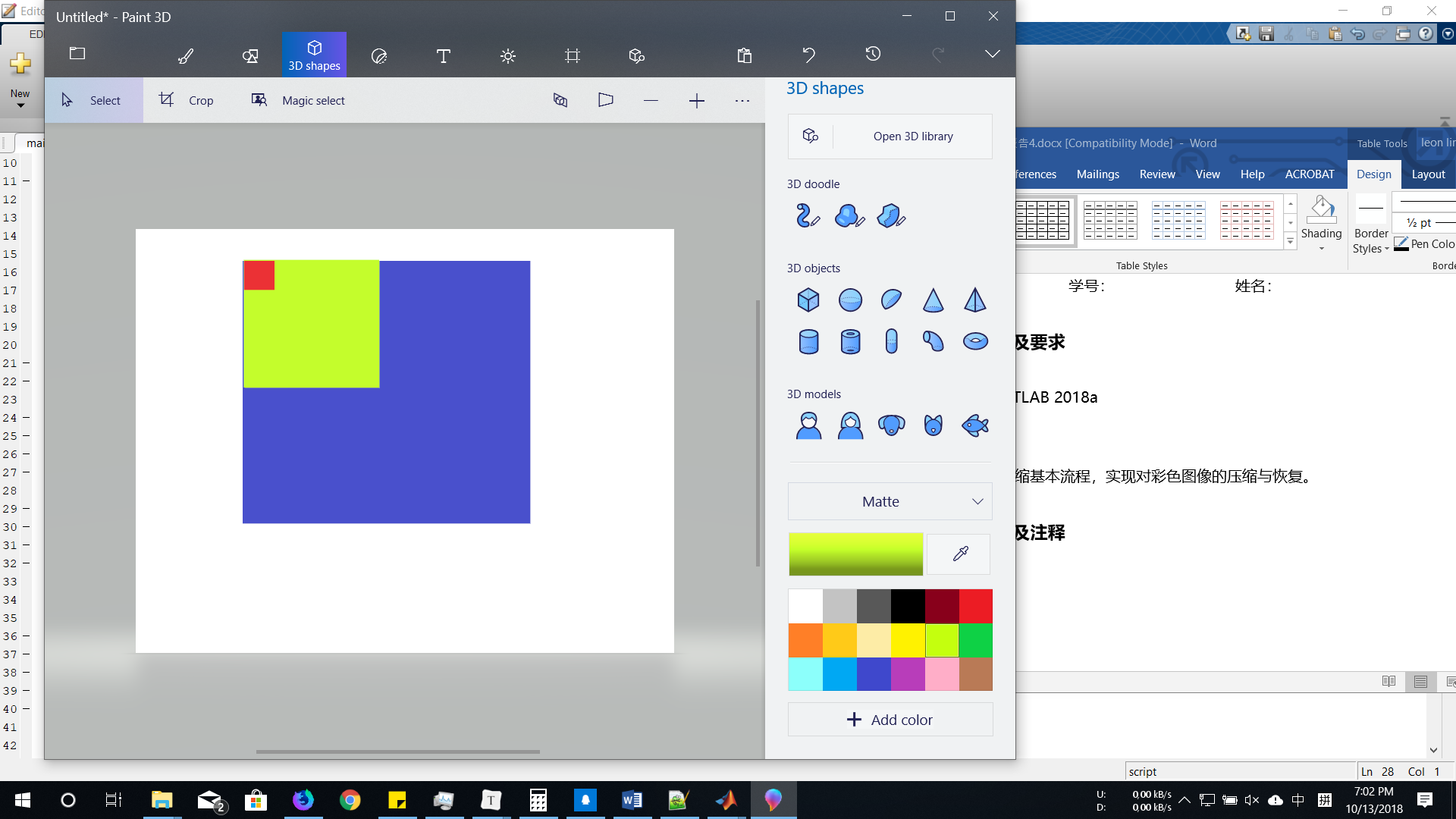
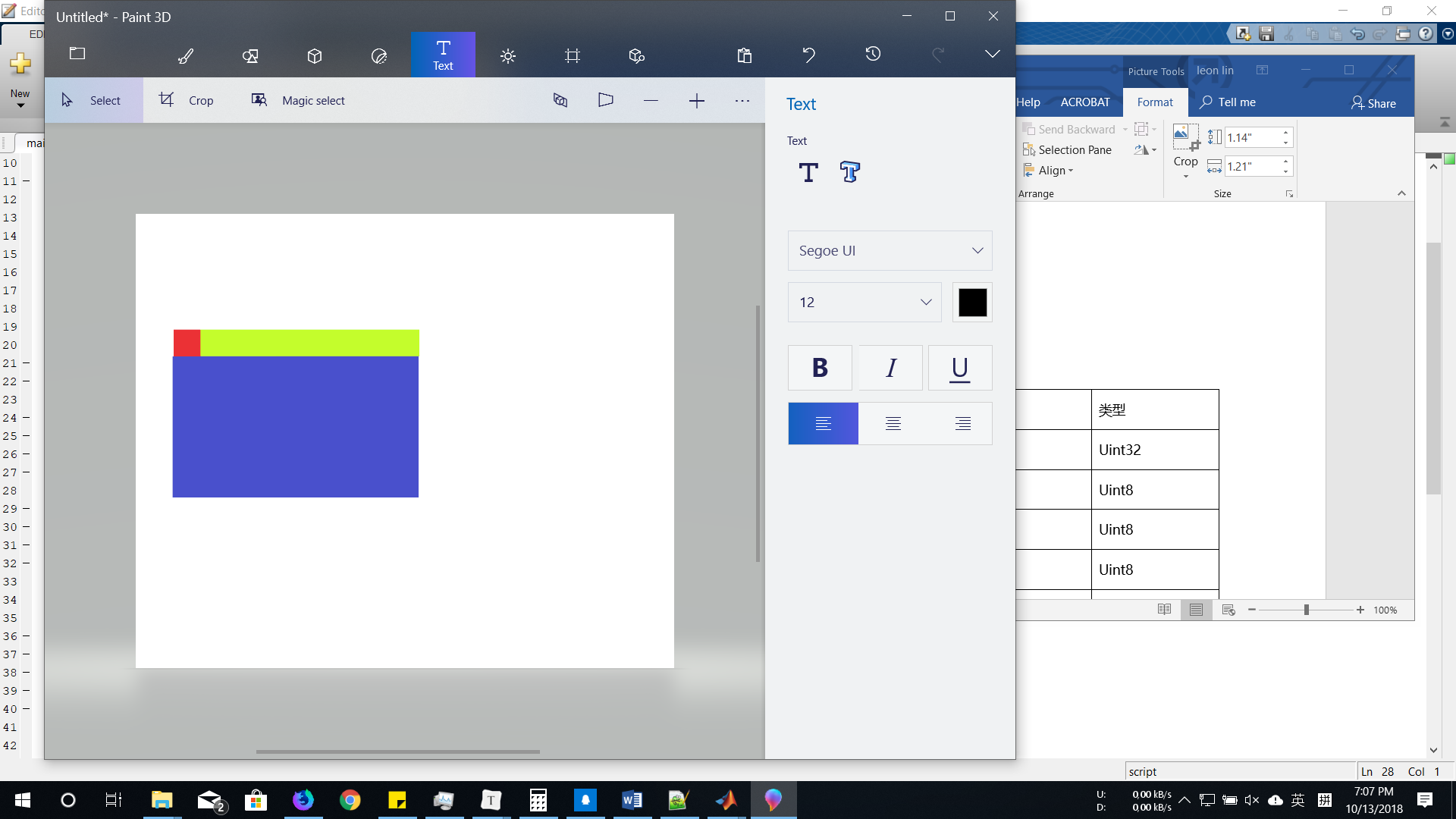
 

Fig1: 红色为dc信号, 黄色ac信号, 经过zigzag展开后的矩阵

数据保存方式

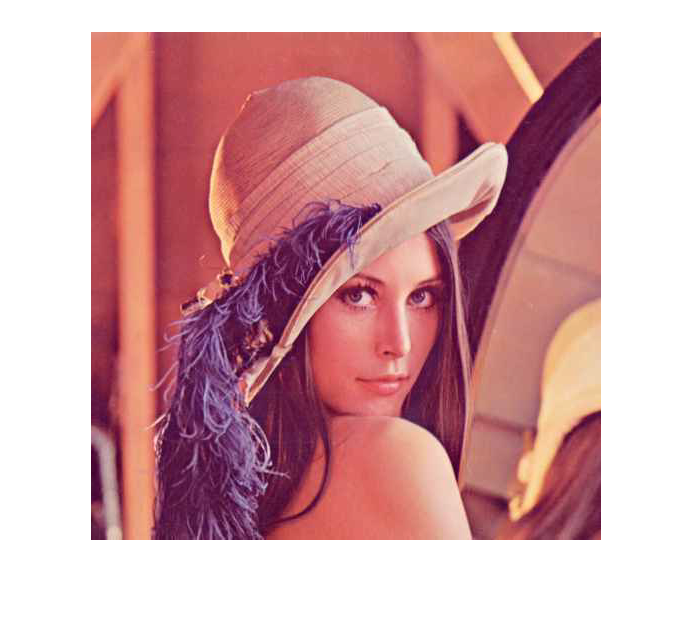
|  |  |
| --- | --- |
| 数据 | 类型 |
| Image size [1,2] (2048 PB bit map image) | Uint32 |
| Padding[1,2] | Uint8 |
| Luminous quantifier matrix[8,8] | Uint8 |
| Color quantifier matrix[8,8] | Uint8 |
| Dictionary size[1,1] | Uint64 |
| Huffman code size[1,1] | Uint64 |
| [symbol[1,1], code length[1,1], Huffman code[1, code length]] | bit32, ubit8, ubit1 |
| Huffman code[1, Huffman code size] | bit1 |

## 实验结果及分析

原图: 761kB



颜色降采样+量化, 亮度量化: 29kB



压缩后图片是原来的3.8%, 细节稍稍有些损失, 颜色略微暗下来一点,但是基本和原图差别不大。