**Stephen Igwue - SCC 416 Course work**

Table of Contents

[Table of Figures 1](#_Toc413782643)

[JPEG encoder 2](#_Toc413782644)

[Color Conversion (RGB to YCbCr) 2](#_Toc413782645)

[DCT transform 2](#_Toc413782646)

[Quantization 3](#_Toc413782647)

[Zig-zag reordering 3](#_Toc413782648)

[Difference DC encoding 5](#_Toc413782649)

[JPEG Decoder 5](#_Toc413782650)

[Colour Conversion (YCbCr to RGB) 5](#_Toc413782651)

[IDCT Transform 6](#_Toc413782652)

[Dequantization, Inverse Zig-zag 6](#_Toc413782653)

[Difference DC Decoding 7](#_Toc413782654)

[Tests 7](#_Toc413782655)

[Colour conversion 7](#_Toc413782656)

[DCT/IDCT Transform 7](#_Toc413782657)

[Differential DC encoding/decoding 8](#_Toc413782658)

[Plots and Visual Comparisons 9](#_Toc413782659)

# **Table of Figures**

Figure 1. Original image. 9

Figure 2. Image to quality factor of 5. 10

Figure 3. Image to quality factor of 10. 11

Figure 4. Image to quality factor of 20. 12

Figure 5. Image to quality factor of 30 13

Figure 6. Image to quality factor of 40. 14

# **JPEG encoder**

## Color Conversion (RGB to YCbCr)

function output = rgb\_to\_ycc( input )

%rgb\_to\_ycc rgb to y cb cr conversion

%multiplier matrices

mux\_y = [0.299 0.587 0.114];

mux\_cb = [-0.168736 -0.331264 0.500002];

mux\_cr = [0.5 -0.418688 -0.081312];

%values to be added

added = [0 128 128];

%initialise space for the output (ycc)

%a 512\*512 by 3 with Y, Cb and Cr saved in the 3 layers

output = zeros(size(input,1), size(input,2), size(input,3));

%perform calulations

output(:,:,1) = mux\_y(1) .\* input(:,:,1) + ...

mux\_y(2) .\* input(:,:,2) + ...

mux\_y(3) .\* input(:,:,3) + added(1);

output(:,:,2) = mux\_cb(1) .\* input(:,:,1) + ...

mux\_cb(2) .\* input(:,:,2) + ...

mux\_cb(3) .\* input(:,:,3) + added(2);

output(:,:,3) = mux\_cr(1) .\* input(:,:,1) + ...

mux\_cr(2) .\* input(:,:,2) + ...

mux\_cr(3) .\* input(:,:,3) + added(3);

end

## DCT transform

function multiplier = dct\_c( index\_k )

%dct\_c Returns the multipliers C(u) and C(v) for any C(k)

if (index\_k == 0)

multiplier = 1 / sqrt(2);

else

multiplier = 1;

end

end

function dct\_array = forward\_dct( input\_array )

%forward\_dct Performs DCT on an 8\*8 matrix, returns an 8\*8 too

dct\_array = zeros(8, 8);

%loop to scan row

for u = 0:7

%loop to scan columns

for v = 0:7

%loop for x

sum = 0;

for x = 0:7

%loop for y

for y = 0:7

sum = sum + ...

input\_array(x+1, y+1) \* ...

cos(pi\*(x + 0.5)\*u/8) \* ...

cos(pi\*(y + 0.5)\*v/8);

end

end

dct\_array(u+1, v+1) = (1/4) \* dct\_c(u) \* dct\_c(v) \* sum;

end

end

end

## Quantization

function [y\_quan\_matrix, c\_quan\_matrix] = scale\_quantization(qf)

%scale\_quantization Scale Luminance and chrominance quantization matrices

%by qf and return them for use in further operations

lum\_matrix = [...

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

];

chrom\_matrix = [...

17 18 24 47 99 99 99 99;

18 21 26 66 99 99 99 99;

24 26 56 99 99 99 99 99;

47 66 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

];

multiplier = 1;

if (qf < 50)

multiplier = ceil(50 / qf);

elseif (qf > 50)

multiplier = ceil((100 - qf) / 50);

else %qf == 50

multiplier = 1;

end

y\_quan\_matrix = multiplier \* lum\_matrix;

c\_quan\_matrix = multiplier \* chrom\_matrix;

end

## Zig-zag reordering

function output\_vector = zigzag( input\_matrix )

%zigzag Zig Zag re-ordering of matrix

%test if input is 8\*8

r = size(input\_matrix, 1);

c = size(input\_matrix, 2);

if (r == 8 & c == 8)

output\_vector = zeros(1, 64);

else

output\_vector = [];

return;

end

%zig zag is composed of the following steps

%0. start at origin (1,1)

%1. move horizontally right by one (or down by one if on last column)

%2. move diagonally left until you hit the boundary

%3. move vertically down by one (or right by one if on last row)

%4. move diagonally right until you hit the boundary

%5. goto 1

vi = 1; %the vector index

ri = 1; ci = 1;

%step 0, start point

output\_vector(vi) = input\_matrix(ri, ci);

state = 'side-right'; %states: side-right, down-left, down-down, up-right

ci = ci + 1;

while (vi < 64)

vi = vi + 1;

output\_vector(1, vi) = input\_matrix(ri, ci);

if strcmp(state, 'side-right')

if (ci < 8)

if (ri == 1)

state = 'down-left';

ri = ri + 1;

ci = ci - 1;

continue;

else %bottom boundaries

state = 'up-right';

ri = ri - 1;

ci = ci + 1;

continue;

end

else %right hand boundary hit

if (ri == 1)

state = 'down-left';

ri = ri + 1;

ci = ci - 1;

continue;

elseif (ri == 8 & ci ~= 8)

state = 'up-right';

ri = ri - 1;

ci = ci + 1;

continue;

else

%terminates here i think

%disp('the end');

break;

end

end

elseif strcmp(state, 'down-left')

if (ci == 1 & ri ~= 8) %left boundary hit

state = 'down-down';

ri = ri + 1;

continue;

elseif (ci == 1 & ri == 8) %bottom corner boundary hit

state = 'side-right';

ci = ci + 1;

continue;

elseif (ri == 8) %bottom boundary hit

state = 'side-right';

ci = ci + 1;

continue;

else %continue the down-left diagonal

%state is unchanged

ci = ci - 1;

ri = ri + 1;

continue;

end

elseif strcmp(state, 'down-down')

if (ci < 8)

%state is unchanged

state = 'up-right';

ri = ri - 1;

ci = ci + 1;

continue;

else %right boundary hit

state = 'down-left';

ri = ri + 1;

ci = ci - 1;

continue;

end

elseif strcmp(state, 'up-right')

if (ri == 1 & ci ~= 8) %hit top boundary

state = 'side-right';

ci = ci + 1;

continue;

elseif (ci == 8) %hit right boundary

state = 'down-down';

ri = ri + 1;

continue

else %continue the up-right diagonal

%state is unchanged

ci = ci + 1;

ri = ri - 1;

continue;

end

end

end

end

## Difference DC encoding

function [ output\_vector ] = diff\_code( input\_vector )

%diff\_code Difference coding of DC coefficients

input\_size = size(input\_vector, 2);

output\_vector = zeros(1, input\_size);

for i = 1:input\_size

if (i == 1)

output\_vector(i) = input\_vector(i) - 0;

else

output\_vector(i) = input\_vector(i) - input\_vector(i-1);

end

end

end

# **JPEG Decoder**

## Colour Conversion (YCbCr to RGB)

function output = ycc\_to\_rgb( input )

%ycc\_to\_rgb rgb to r g b conversion

%multiplier matrices

mux\_r = [1.0 0.0 1.40210];

mux\_g = [1.0 -0.34414 -0.71414];

mux\_b = [1.0 1.77180 0.0];

%initialise space for the output (rgb)

%a 512\*512 by 3 with R, G and B saved in the 3 layers

output = zeros(size(input,1), size(input,2), size(input,3));

%perform calulations

output(:,:,1) = round(mux\_r(1) .\* input(:,:,1) + ...

mux\_r(2) .\* (input(:,:,2) - 128) + ...

mux\_r(3) .\* (input(:,:,3) - 128));

output(:,:,2) = round(mux\_g(1) .\* input(:,:,1) + ...

mux\_g(2) .\* (input(:,:,2) - 128) + ...

mux\_g(3) .\* (input(:,:,3) - 128));

output(:,:,3) = round(mux\_b(1) .\* input(:,:,1) + ...

mux\_b(2) .\* (input(:,:,2) - 128) + ...

mux\_b(3) .\* (input(:,:,3) - 128));

end

## IDCT Transform

function inv\_dct\_array = inverse\_dct( input\_array )

%inverse\_dct Performs inverse DCT on an 8\*8 matrix, returns an 8\*8 too

inv\_dct\_array = zeros(8, 8);

%loop to scan row

for x = 0:7

%loop to scan columns

for y = 0:7

%loop for x

sum = 0;

for u = 0:7

%loop for y

for v = 0:7

sum = sum + ...

dct\_c(u) \* dct\_c(v) \* ...

input\_array(u+1, v+1) \* ...

cos(pi\*(x + 0.5)\*u/8) \* ...

cos(pi\*(y + 0.5)\*v/8);

end

end

inv\_dct\_array(x+1, y+1) = round((1/4) \* sum);

end

end

end

## Dequantization, Inverse Zig-zag

%y component

lena\_y\_dbl\_dct\_zagged = zeros(lena\_row\_size, lena\_col\_size);

lena\_y\_dbl\_idct = zeros(lena\_row\_size, lena\_col\_size);

qi = 1; %block index

for u = 1:8:lena\_row\_size

for v = 1:8:lena\_col\_size

%build back dc & ac components

tmp\_zigged = zeros(1, 64);

%dc

tmp\_zigged(1) = lena\_y\_dbl\_dct\_zigged.dc(qi);

%ac

tmp\_zigged(2:64) = lena\_y\_dbl\_dct\_zigged.ac(64\*(qi-1)+2 : 64\*qi);

%dezigzag

lena\_y\_dbl\_dct\_zagged(u:u+7, v:v+7) = dezigzag(tmp\_zigged);

%dequantize

lena\_y\_dbl\_dct\_zagged(u:u+7, v:v+7) = lena\_y\_dbl\_dct\_zagged(u:u+7, v:v+7) .\* y\_quan\_matrix;

%inverse dct

lena\_y\_dbl\_idct(u:u+7, v:v+7) = inverse\_dct(lena\_y\_dbl\_dct\_zagged(u:u+7, v:v+7));

qi = qi + 1;

end

end

## Difference DC Decoding

function [ output\_vector ] = diff\_decode( input\_vector )

%diff\_code Difference decoding of DC coefficients

input\_size = size(input\_vector, 2);

output\_vector = zeros(1, input\_size);

for i = 1:input\_size

if (i == 1)

output\_vector(i) = input\_vector(i) + 0;

else

output\_vector(i) = input\_vector(i) + output\_vector(i-1);

end

end

end

# **Tests**

## Colour conversion

%% convert rgb to ycc

lena\_ycc\_dbl = rgb\_to\_ycc(lena\_rgb\_dbl);

%% test ycc conversion

if (tests == 1)

%reverse previous conversion

lena\_rgb\_test = ycc\_to\_rgb(lena\_ycc\_dbl);

%test for equality. this might fail if they are not integers

if (isequal(lena\_rgb\_dbl, lena\_rgb\_test))

disp('success: ycc conversion');

else

%randomly pick a cell to display to see they are equal, we'll be

%able to tell if the sameness test failed because of roundup errors

disp('test: ycc conversion');

[indices] = randi([1 size(lena\_rgb\_test, 1)], 1, 2);

disp(lena\_rgb\_dbl(indices(1), indices(2), 2));

disp(lena\_rgb\_test(indices(1), indices(2), 2));

end

end

## DCT/IDCT Transform

%dct on y

lena\_y\_dbl\_dct = zeros(lena\_row\_size, lena\_col\_size);

for u = 1:8:lena\_row\_size

for v = 1:8:lena\_col\_size

lena\_y\_dbl\_dct(u:u+7, v:v+7) = forward\_dct(lena\_y\_dbl(u:u+7, v:v+7));

%save some time by doing quantization and zigzag here

end

end

if (tests == 1)

% test using inverse\_dct

lena\_y\_dbl\_test = zeros(lena\_row\_size, lena\_col\_size);

for u = 1:8:lena\_row\_size

for v = 1:8:lena\_col\_size

lena\_y\_dbl\_test(u:u+7, v:v+7) = inverse\_dct(lena\_y\_dbl\_dct(u:u+7, v:v+7));

end

end

%test equality. might fail due to round offs

if (isequal(lena\_y\_dbl, lena\_y\_dbl\_test))

disp('success: dct conversion');

else

%randomly display a cell to see equality

disp('test: dct');

[indices] = randi([1 size(lena\_y\_dbl\_test, 1)], 1, 2);

disp(lena\_y\_dbl(indices(1), indices(2)));

disp(lena\_y\_dbl\_test(indices(1), indices(2)));

end

end

## Differential DC encoding/decoding

%% difference encode

if (tests == 1)

disp('test: diff en/de-coding');

%choose random position to test

[indices] = randi([1 size(lena\_y\_dbl\_dct\_zigged.dc, 2)], 1, 1);

disp(lena\_y\_dbl\_dct\_zigged.dc(indices));

end

lena\_y\_dbl\_dct\_zigged.dc = diff\_code(lena\_y\_dbl\_dct\_zigged.dc);

lena\_cb\_dbl\_dct\_zigged.dc = diff\_code(lena\_cb\_dbl\_dct\_zigged.dc);

lena\_cr\_dbl\_dct\_zigged.dc = diff\_code(lena\_cr\_dbl\_dct\_zigged.dc);

%% jpeg decoding

%difference decode

lena\_y\_dbl\_dct\_zigged.dc = diff\_decode(lena\_y\_dbl\_dct\_zigged.dc);

%test dc en/de-coding

if (tests == 1)

%display content of random position to compare

disp(lena\_y\_dbl\_dct\_zigged.dc(indices));

end

# **Plots and Visual Comparisons**



Figure 1. Original image.

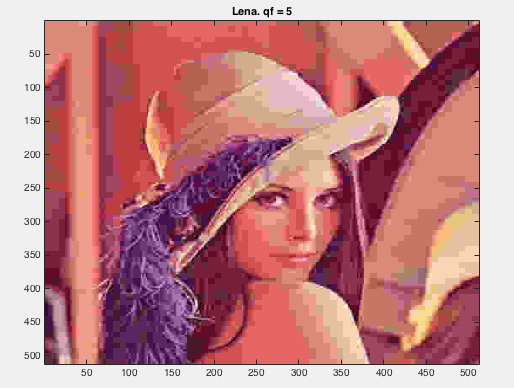


Figure 2. Image to quality factor of 5.



Figure 3. Image to quality factor of 10.



Figure 4. Image to quality factor of 20.



Figure 5. Image to quality factor of 30



Figure 6. Image to quality factor of 40.