**Porto Alegre, 03 de Julho de 2012**



**INF01 046 – Fundamentos de Processamento de Imagens**

**Trabalho Prático 2**

**Prof. Jacob Scharcanski**

**André Figueiredo - 137757**

**Lucas Lazari da Silva - 149722**

**Bárbara Flores - 114305**

**TURMA : U**

Nós estudamos em aula a Transformada DCT 2D em blocos. Neste trabalho você vai compactar imagens usando técnicas baseadas na DCT, de forma similar ao algoritmo básico do JPEG**.** Por simplicidade, os dados das imagens comprimidas não precisam seguir a sintaxe proposta pelo padrão JPEG. Usaremos **imagens em tons de cinza.**

(1)  Subtraia 128 do valor de cada pixel da imagem, e aplique a Transformada DCT 2D em blocos 8 x 8. Use a **Tabela de Quantização dos coeficientes da DCT** mostrada em aula (vide transparências de aula). Mostre que existe uma relação entre a distorção introduzida pela quantização (e.g. usando **PSNR**) e o número de bits usados na quantização dos coeficientes da DCT. Para cada imagem, use um fator multiplicativo para a Tabela de Quantização “q” (q ≥ 1) para mudar os passos de quantização especificados na Tabela;

|  |
| --- |
| % onde I é a imagem e q é o fator multiplicativo q >= 1 function jpeg = jpeg\_compacta(I, q)  % imagem original   % jpeg = Img retorno  % 1. Subtraia 128 do valor de cada pixel da imagem  jpeg = double(I);  jpeg = jpeg - 128;   % 2. Aplique a transformada DCT 2D em blocos de 8x8  % jpg = dct2(jpg);  % figure, imshow(J, 'Após DCT2');   % 3  Tabela de Quantização dos coeficientes da DCT usando o multiplicador passado  Q = q .\* [...  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];   % cria blocos 8x8  G = separaBlocos(jpeg);  DC = [];  AC = [];   for x = 1 : size(G, 3)  % aplica DCT2 pra cada um dos blocos  G(:, :, x) = dct2(G(:, :, x));     % aplica quantização em cada um dos elementos dos blocos  % Bj,k = round(Gj,k / Qj,k)    G(:, :, x) = round(G(:, :, x)./Q);     DC = [DC; G(1, 1, x)];  AC = [AC; [runLength(zigzag(G(:, :, x)))].'];  end   % Faz a compressão DPCM dos coeficientes DC  % Isto codifica cada coeficiente DC como um delta do coeficiente anterior  DC = DPCM(DC);  % Coloca numa estrutura os dados comprimidos e a info relevante  jpeg = struct('field\_q', q,...  'field\_sizeI', size(I), ...              'field\_sizeG', size(G),...  'field\_DC', DC,...  'field\_AC', AC); |

(2)  Use **DPCM** para codificar os coeficientes DC (i.é, DCT(0,0) ). Use uma varredura em **zig-zag** e represente os coeficientes **AC** como um seqüência 1D de valores;

|  |
| --- |
| function vetor = DPCM(vetor)  % Representa cada elemento do vetor pela sua diferença para o elemento anterior  % Iniciamos pelo último elemento e iteramos até o segundo  for x = size(vetor, 1) : -1 : 2   % Para cada elemento, substitui-o por ele menos o anterior   vetor(x,1) = vetor(x, 1) - vetor(x-1, 1);  end  return |

|  |
| --- |
| function v = zigzag(m)  %Função que constrói o vetor v a partir da matriz m percorrida em zigzag.  %inicializa o vetor v  v = [];  %i indexa as linhas que serão acessadas em zigzag  i = [ 1 2 3 2 1 1 2 3 4 5 4 3 2 1 1 2 3 4 5 6 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 3 4 5 6 7 8 8 7 6 5 4 5 6 7 8 8 7 6 7 8 8];   %j indexa as colunas que serão acessadas em zigzag  j = [ 2 1 1 2 3 4 3 2 1 1 2 3 4 5 6 5 4 3 2 1 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1 2 3 4 5 6 7 8 8 7 6 5 4 3 4 5 6 7 8 8 7 6 5 6 7 8 8 7 8];    %acessa os elementos de m indexados por i e j e coloca-os no vetor v  for x = 1 : size(i, 2)  v = [v, m(i(x), j(x))];  end return |

(3)  Para cada imagem, calcule a distribuição estatística dos **coeficientes** **quantizados** (em todas as seqüências), e estime quantos bits seriam usados para codificar estes coeficientes quantizados usando a **entropia** da distribuição encontrada (que será aproximadamente o número médio de bits/coeficiente que seria obtido usando os códigos de **Huffman**) – o número total de bits é o produto do número de “símbolos” da seqüência pelo número de bits necessário para codificar cada um dos “símbolos” com o código ótimo (este número é dado pela entropia da distribuição encontrada);

(4)  Observe a **taxa de compressão total** obtida com base no número médio de bits/coeficiente calculado pela **entropia** e o número total de coeficientes (veja ítem 3, acima). *Nota:* Você deve incluir na esti  
mativa também os valores **DPCM** dos coeficientes **DC** (veja item 2, acima). Teste valores diferentes de “q” (veja item 1, acima) e inclua os resultados no seu relatório, junto com uma curva “PSNR x número de bits usados na imagem comprimida”.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| q | Entropia | PSNR | Bits | Simbolos | Taxa de Compressão | Tamanho relativo ao original |
| 1 |  | 30.5251 |  |  |  |  |
| 4 |  | 26.4782 |  |  |  |  |
| 7 |  | 24.7734 |  |  |  |  |
| 10 |  | 23.5056 |  |  |  |  |

**“PSNR x número de bits usados na imagem comprimida”.**

Conclusão: Quanto maior o PSNR, que significa menos ruído, maior o tamanho da imagem e melhor qualidade. Quanto maior a compressão, menor a qualidade da imagem.

(5)  Escreva um programa para **reconstruir** e **exibir** a imagem invertendo os passos usados na versão simplificada de codificação JPEG, descrita nos passos 1-3.

|  |
| --- |
| function Img = jpeg\_descompacta(jpeg)  % Coloca numa estrutura os dados comprimidos e a info relevante % jpeg = struct('field\_q', q,... %   'field\_sizeI', size(I), ... %            'field\_sizeG', size(G),... %            'field\_DC', DC,... %            'field\_AC', AC);  Q = jpeg.field\_q .\* [...  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];   % faz caminho inverso: % 1. DPCM inverso -> DC % 2. RunLength inverso -> AC % 3. zig-zag inverso -> AC % 4. coloca as infos do DC, AC -> bloco % 5. desfaz a quantização -> bloco % 6. faz o DCT2 inverso -> bloco % 7. bloco -> Imagem Saída  % 1. DESFAZ CORRIDAS DE ZEROS AC = irunLength(jpeg.field\_AC.');  % 2. DESFAZER DPCM DC = iDPCM(jpeg.field\_DC);  % 3. DESFAZ DCT E QUANTIZAÇÃO G = zeros(jpeg.field\_sizeG(1), jpeg.field\_sizeG(2), jpeg.field\_sizeG(3));    % para cada um dos blocos, faz os passos 4 a 7 for i = 1 : jpeg.field\_sizeG(3)   G(:, :, i) = iZigZag( AC(1 + 63\*(i-1) : 63\*i ) );   G(1, 1, i) = DC(i, 1);   G(:, :, i) = G(:, :, i) .\* Q;   G(:, :, i) = idct2(G(:, :, i)); end    Img = joinBlocos(G); Img = Img + 128; Img = uint8(Img);  return |

Você deve entregar o código executável (de preferência em MATLAB), e fazer uma explicação da sua implementação no relatório final do projeto. Também, teste o codificador e o decodificador para a imagem “Lenna” com distorção apenas imperceptível (q=1) e com distorção maior (ex: q=3), indicando os tamanhos dos arquivos e as taxas de compressão obtidas para as imagens comprimidas com o seu algoritmo. Compare os resultados para outras imagens. A imagem compactada deve ser descompactada, e reconstruída ! As etapas de (1)-(5) valem até 5 pontos.

Codificação de Huffman (esta etapa será valorada em até 5 pontos se implementada corretamente com o restante do trabalho):

Supondo que você tenha implementado a quantização dos coeficientes das DCT e o zig-zag, nós teremos uma série de coeficientes AC em um dado bloco, como mostrado abaixo (sugestão: use as tabelas de códigos prontas do livro texto do Gonzalez et. al.) :

DC07 00001 0 0 19 0 0 0 0 0 00000000 00000000 00000000 00000000 00000000 00000000

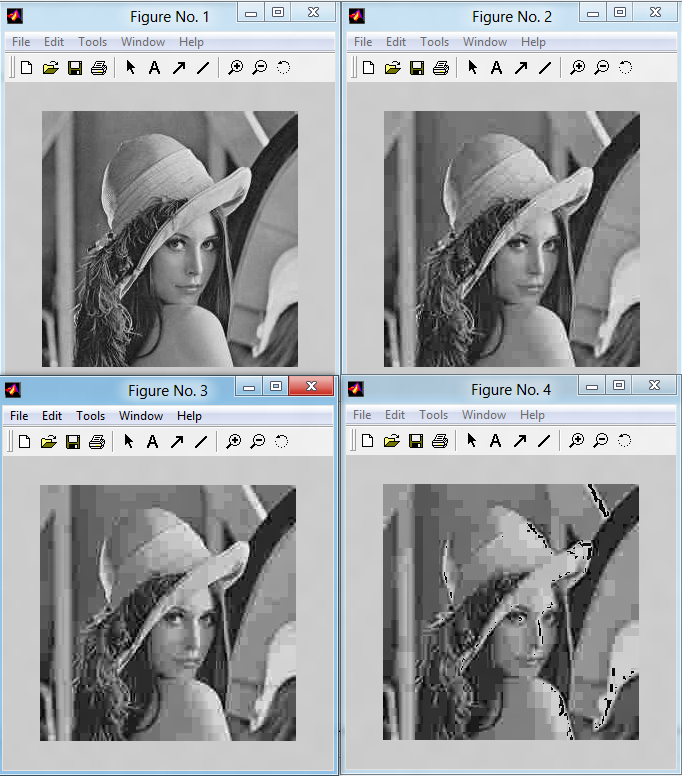
Podemos então codificar os coefs. AC como: (1)7, (4)1, (2)19, EOB.que

(1)  O símbolo (n) denota quantos zeros precedem um AC não-nulo;  
   
(2)  O número (ex: 7, 1 ou 19) é a palavra-símbolo que representa a corrida de zeros, e a distribuição de coeficientes quantizados será agora representada pela distribuição de palavras-símbolo – ou seja, você terá um dicionário de palavras-símbolo e uma distribuição das suas ocorrências nas corridas de todos os blocos da imagem;  
   
(3)  O símbolo **EOB** (end of block) representa o final do bloco.

Você deve entregar o código executável (de preferência em MATLAB), e fazer uma explicação da sua implementação no relatório final do projeto. Também, teste o codificador e o decodificador para a imagem “**Lenna**” com distorção apenas imperceptível (q=1) e com distorção maior (ex: q=3), indicando os tamanhos dos arquivos e as taxas de compressão obtidas para as imagens comprimidas com o seu algoritmo. Compare os resultados para outras imagens. A imagem compactada deve ser descompactada, e reconstruída !

|  |
| --- |
| % Elimina 0's no vetor v % RunLength / Cod. Huffman  function ret = runLength(AC) EOB = 255; % Símbolo End-Of-Block iZeros = 0; % contador dos zeros (n) pos = 1; % Indíce atual no vetor de restorno   for i = 1 : size(AC, 2);      % vai incrementando os zeros   if AC(i) == 0  iZeros = iZeros + 1;   else      % quando o elemento for diferente de zero      % atribui os zeros à posiçaõ e o elemento à seguinte      % (1)  O si´mbolo (n) denota quantos zeros precedem um AC na~o-nulo;            ret(pos) = iZeros;            % (2) O número (ex: 7, 1 ou 19) é a palavra-símbolo que representa a corrida de       % zeros, e a distribuição de coeficientes quantizados será agora representada       % pela distribuição de palavras-símbolo – ou seja, você terá um dicionário de       % palavras-símbolo e uma distribuição das suas ocorrências nas corridas de       % todos os blocos da imagem;            ret(pos + 1) = AC(i);            pos = pos + 2;  iZeros = 0;  end end  if (iZeros > 0)   ret(pos) = iZeros;  pos = pos + 1 end  % (3) O símbolo EOB (end of block) representa o final do bloco ret(pos) = EOB;  return |

# Resultados:



Conforme a tabela acima da seção de PSNR, esses são os resultados.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure No. 1 | 1 | 30.5251 |
| Figure No. 1 | 4 | 26.4782 |
| Figure No. 1 | 7 | 24.7734 |
| Figure No. 1 | 10 | 23.5056 |

## Referencias

* The MathWorks, Inc.; Product Documentation. <http://www.mathworks.com/help/techdoc>. Visitado em junho/julho de 2012.
* Wikimedia Foundation, Inc.; JPEG. <http://en.wikipedia.org/wiki/JPEG>. Visitado em junho/julho de 2012.
* Wikimedia Foundation, Inc.; Entropy (information theory). <http://en.wikipedia.org/wiki/Entropy_(information_theory)>. Visitado em junho/julho de 2012.
* Wikimedia Foundation, Inc.; Run-length encoding. <http://en.wikipedia.org/wiki/Run-length_encoding>. Visitado em junho/julho de 2012.
* Wikimedia Foundation, Inc.; Huffman coding. <http://en.wikipedia.org/wiki/Huffman_coding>. Visitado em junho/julho de 2012.