Centro Universitário da FEI

NEC130/EL0130 - SISTEMA DE VIDEO E AUDIO DIGITAL

**PROCESSAMENTO DIGITAL DE IMAGEM**

Prof. Cleiton Fidelix Pereira

Gustavo Sanomia Ryuji 12.115.481-9

Jéssica Trajano Matheus Benedito 12.218.167-0

São Bernardo do Campo

2021

1. Objetivo

O objetivo desse projeto é programar no Octave um algoritmo simplificado da compactação MPEG, o projeto consiste em 3 etapas:

1. Subamostragem de crominância (4:2:0), DCT (Transformada Discreta de Cosseno), quantização e serialização zigzag.
2. Tornar o resultado da etapa 1 em binário.
3. Converter e resultado da etapa 2 em decimal, e realizar as técnicas inversas da etapa 1.
4. Código

**2.1 Código: Etapa 1**

O programa inicia fechando todas as janelas e limpando todas as variáveis da workspace e limpa o console. Também importa a pacote *signal* do Octave, para utilizarmos a função *dct2*, que é a função DCT para matrizes de 2 dimensões. A variável *str\_time* armazena o tempo de execução do Octave no começo de programa.

|  |
| --- |
| clear all  close all  clc  pkg load signal  str\_time **=** cputime**;** % Tempo de execução inicial |

O programa abre a imagem e redimensiona para um tamanho divisor de 8. Para isso foi decidido preencher os pixeis com uma cópia dos últimos pixels, pois assim manteria a qualidade da imagem nas bordas.

|  |  |
| --- | --- |
|  |  |
| Copiando a última linha de pixeis | Preenchendo com pixeis pretos |

|  |
| --- |
| X\_r **=** imread**(**'image.png'**);** % Carrega a imagem em X\_r  % Dados da imagem  l\_lin **=** size**(**X\_r**,** 1**);**  l\_col **=** size**(**X\_r**,** 2**);**  x\_l **=** mod**(**l\_lin**,** 8**);**  x\_c **=** mod**(**l\_col**,** 8**);**  % Redimensiona a imagem, adiciona pixeis  X **=** **[**X\_r**;** repmat**(**X\_r**(**end**,** **:,** **:),** 8 **-** x\_l**,** 1**)];**  l\_lin **=** size**(**X**,** 1**);**  X **=** **[**X repmat**(**X**(:,** end**,** **:),** 1**,** 8 **-** x\_c**)];**  l\_col **=** size**(**X**,** 2**);** |

Para a conversão RGB para YCrCb, foram utilizadas as seguintes equações:

Essa são as mesmas equações utilizadas pelo algoritmo MPEG.

No Octave foi utilizada a função *im2double* para que as matrizes resultantes aceitassem valores negativos.

A subamostragem e realizada obtendo a média de conjuntos de 4 pixeis, o canal de iluminância não é subamostrado.

Em seguida preenchemos as matrizes de subamostragem com valores 0, para que fique do mesmo tamanho do canal iluminância Y.

|  |
| --- |
| % Extrai o RGB  R **=** im2double**(**X**(:,:,**1**),** 'indexed'**)-**1**;** %separa o canal R na matriz R  G **=** im2double**(**X**(:,:,**2**),** 'indexed'**)-**1**;** %separa o canal G na matriz G  B **=** im2double**(**X**(:,:,**3**),** 'indexed'**)-**1**;** %separa o canal B na matriz B  % Converte de RGB para YCbCr  Y **=** 0.299**\***R **+** 0.587**\***G **+** 0.114**\***B**;** % Calcula o valor da luminância  Cb **=** 128 **-** 0.168736**\***R **-** 0.331264**\***G **+** 0.5**\***B**;**% Calcula o valor de Cb  Cr **=** 128 **+** 0.5**\***R **-** 0.418688**\***G **-** 0.081312**\***B**;**% Calcula o valor de Cr  % Subamostragem 4:2:0  Cr\_a **=** **(**  Cr**(**1**:**2**:**end**,** 1**:**2**:**end**)** **+**  Cr**(**2**:**2**:**end**,** 1**:**2**:**end**)** **+**  Cr**(**1**:**2**:**end**,** 2**:**2**:**end**)** **+**  Cr**(**2**:**2**:**end**,** 2**:**2**:**end**)**  **)/**4**;** % Amostra Cr  Cb\_a **=** **(**  Cb**(**1**:**2**:**end**,** 1**:**2**:**end**)** **+**  Cb**(**2**:**2**:**end**,** 1**:**2**:**end**)** **+**  Cb**(**1**:**2**:**end**,** 2**:**2**:**end**)** **+**  Cb**(**2**:**2**:**end**,** 2**:**2**:**end**)**  **)/**4**;** % Amostra Cb  % Redimencionamento de Cr e Cb  Cr\_c **=** zeros**(**l\_lin**,** l\_col**);** % Nova matriz Cr  Cb\_c **=** zeros**(**l\_lin**,** l\_col**);** % Nova matriz Cb  Cr\_c**(**1**:**end**/**2**,** 1**:**end**/**2**)** **=** Cr\_a**;**  Cb\_c**(**1**:**end**/**2**,** 1**:**end**/**2**)** **=** Cb\_a**;** |

O algoritmo percorre os canais em blocos de 8x8, onde aplicamos a DCT e dividimos pela matriz de quantização Q.

Aplicamos a função *zigzag* para formar um vetor de uma dimensão para transmissão.

Nota: Como aplicamos a DCT e dividimos pela quantização, a função *zigzag* faz com que a maior quantidade de zeros fique concentrada no final do bloco de bits.

|  |
| --- |
| % Variavel de coeficientes  l\_coef **=** zeros**(**1**,** l\_lin**/**8**\***l\_col**/**8**\***3**\***64**);**  % Varre a imagem e separa em blocos de 8  id **=** 1**;**  **for** y **=** 1**:**8**:**l\_lin  **for** x **=** 1**:**8**:**l\_col  % Aplica a função DCT, divide pela quantizção, então aplica o zigzag  l\_coef**(**id**:**id**+**63**)** **=** zigzag**(**round**(**dct2**(**Y**(**y**:**y**+**7**,** x**:**x**+**7**))./**Q**));**  id **=** id **+** 64**;**  l\_coef**(**id**:**id**+**63**)** **=** zigzag**(**round**(**dct2**(**Cr\_c**(**y**:**y**+**7**,** x**:**x**+**7**))./**Q**));**  id **=** id **+** 64**;**  l\_coef**(**id**:**id**+**63**)** **=** zigzag**(**round**(**dct2**(**Cb\_c**(**y**:**y**+**7**,** x**:**x**+**7**))./**Q**));**  id **=** id **+** 64**;**  **endfor**  %disp(y/l\_lin);  **endfor**  l\_coef **=** round**(**l\_coef**);** % Arredonda os valores dos coeficientes |

|  |
| --- |
| **zigzag.m** |
| **function** result **=** zigzag**(**matriz**)**  a **=** **[**  1  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  **];**  b **=** **[**  1  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  **];**  result **=** zeros**(**1**,** 64**);**  **for** i **=** 1**:**64  result**(**i**)** **=** matriz**(**a**(**i**),** b**(**i**));**  **endfor**  **endfunction** |

**2.2 Código: Etapa 2**

Essa etapa simula um arquivo salvo, os coeficientes decimais são convertidos para binário, com a estrutura:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **End-of\_block** | **Bit de sinal** | **Valor do coeficiente** | | | | | | |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Cada coeficiente será representado por 9 bits.

Foi escolhida a técnica de complemento para 2, isso permite utilizar a função de conversão em toda a matriz agilizando o código e simplificando o programa.

|  |
| --- |
| bin\_c **=** **[];**  **for** i **=** 1**:**64**:**length**(**l\_coef**)**  bin\_c **=** **[**bin\_c**;** codifica**(**l\_coef**(**i**:**i**+**63**))];**  **endfor**  temp **=** dec2bin**(**bin\_c**(:,** 2**)** **+** 256**,** 9**);**  bin **=** zeros**(**length**(**temp**),** 9**);**  bin**(:,** 1**)** **=** bin\_c**(:,** 1**);**  **for** id**=**2**:**9  bin**(:,** id**)** **=** str2num**(**temp**(:,** id**));**  **endfor** |

|  |
| --- |
| **codifica.m** |
| **function** b **=** codifica**(**bloco**)**  b **=** **[];**  b **=** **[**b**;** **[**0 bloco**(**1**)]];**  **for** c**=**2**:**64  **if** sum**(**abs**(**bloco**(**c**:**end**)))** **==** 0  % Adiciona byte end of block  b **=** **[**b**;** **[**1 0**]];**  **break**  **else**  b **=** **[**b**;** **[**0 bloco**(**c**)]];**  **endif**  **endfor**  **endfunction** |

**2.3 Código: Etapa 3**

Primeiro reconstituímos o vetor com os coeficientes, o programa identifica quando o bit EoB é igual a 1, e preenche o resto com zeros até formar o bloco de 64 coeficientes.

|  |
| --- |
| % Reconstitui o vetor intero para realizar as operações inversas  rbin **=** zeros**(**l\_lin**/**8**\***l\_col**/**8**\***3**\***64**,** 8**);**  ibin **=** 1**;**  **for** id**=**1**:**64**:**length**(**rbin**)**  **for** id2 **=** 0**:**63  cbin **=** bin**(**ibin**,** **:);**  rbin**(**id**+**id2**,** **:)** **=** cbin**(**2**:**9**);**  ibin **+=** 1**;**  **if** cbin**(**1**)** **==** 1  **break;**  **endif**  **endfor**  **endfor**  l\_dcoef **=** bin2dec**(**num2str**(**rbin**(:,**2**:**8**))).-**128**\***rbin**(:,**1**);** |

O vetor de 1 dimensão *l\_dcoef* é reconstituído em uma matriz do tamanho da imagem original redimensionada. No mesmo loop, a matriz é multiplicada pela quantização Q então é aplicada a função DCT inversa *idct2*.

|  |
| --- |
| % Cria matrizes YCrCb  Y\_d **=** zeros**(**l\_lin**,** l\_col**);**  Cr\_d **=** zeros**(**l\_lin**,** l\_col**);**  Cb\_d **=** zeros**(**l\_lin**,** l\_col**);**  id **=** 1**;**  **for** y **=** 1**:**8**:**l\_lin  **for** x **=** 1**:**8**:**l\_col  % Aplica a função inversa: zigzag, quantização e DCT  Y\_d**(**y**:**y**+**7**,** x**:**x**+**7**)** **=** idct2**(**izigzag**(**l\_dcoef**(**id**:**id**+**63**)).\***Q**);**  id **=** id **+** 64**;**  Cr\_d**(**y**:**y**+**7**,** x**:**x**+**7**)** **=** idct2**(**izigzag**(**l\_dcoef**(**id**:**id**+**63**)).\***Q**);**  id **=** id **+** 64**;**  Cb\_d**(**y**:**y**+**7**,** x**:**x**+**7**)** **=** idct2**(**izigzag**(**l\_dcoef**(**id**:**id**+**63**)).\***Q**);**  id **=** id **+** 64**;**  **endfor**  **endfor** |

A subamostragem da etapa 1 gerou matrizes Cb e Cr com metade do tamanho, para gerar os blocos de dados, geramos a matriz com o mesmo tamanho do canal de luminância Y.

|  |
| --- |
| % Subamostragem inversa 4:2:0  Cr\_f **=** zeros**(**l\_lin**,** l\_col**);**  Cb\_f **=** zeros**(**l\_lin**,** l\_col**);**  **for** id **=** 0**:**3  % Permutação binaria  b **=** dec2bin**(**id**,** 2**);**  b **=** str2num**([**b**(**1**);** b**(**2**)])** **+** 1**;**  % Percorre quadrados de 4x4 e adiciona pixel  Cr\_f**(**b**(**1**):**2**:**end**,** b**(**2**):**2**:**end**)** **=** Cr\_d**(**1**:**end**/**2**,** 1**:**end**/**2**);**  Cb\_f**(**b**(**1**):**2**:**end**,** b**(**2**):**2**:**end**)** **=** Cb\_d**(**1**:**end**/**2**,** 1**:**end**/**2**);**  **endfor** |

Finalmente as matrizes de YCbCr são convertidas novamente para RGB e gera o vetor de imagem novamente. As margens adicionadas são removidas, então, a imagem fica novamente com o tamanho original.

|  |
| --- |
| % Converte de YCrCb para RGB  Rf **=** Y\_d **+** 1.402**\*(**Cr\_f **-** 128**);**  Gf **=** Y\_d **-** 0.344136**\*(**Cb\_f **-** 128**)** **-** 0.714136**\*(**Cr\_f **-** 128**);**  Bf **=** Y\_d **+** 1.772**\*(**Cb\_f **-** 128**);**  % Une RGB em uma variavel  Xf**(:,:,**1**)** **=** uint8**(**Rf**);**  Xf**(:,:,**2**)** **=** uint8**(**Gf**);**  Xf**(:,:,**3**)** **=** uint8**(**Bf**);**  % Redimencionamento  Xf\_r **=** Xf**(**1**:**l\_lin **-** 8 **+** x\_l**,** 1**:** l\_col **-** 8 **+** x\_c**,** **:);** |

As últimas linhas do código são para gerar o arquivo de imagem, visualização e relatório de execução.

|  |
| --- |
| % Salva imagem reconstruida  imwrite**(**Xf**,** 'result.jpg'**);**  % Plotagem  figure**(**'name'**,** 'Comparação das imagens (Com redimencionamento).'**);**  a **=** subplot**(**1**,**2**,**1**);**  imagesc**((**1**:**l\_col**)-**0.5**,** **(**1**:**l\_lin**)-**0.5**,** X**);**  title**(**'Original'**);**  axis off**;**  axis**(**'equal'**);**  b **=** subplot**(**1**,**2**,**2**);**  imagesc**((**1**:**l\_col**)-**0.5**,** **(**1**:**l\_lin**)-**0.5**,** Xf**);**  title**(**'Reconstruido'**);**  axis off**;**  axis**(**'equal'**);**  linkaxes**([**a**,** b**]);** % Alinhamento de zoom  figure**(**'name'**,** 'Comparação das imagens.'**);**  a **=** subplot**(**1**,**2**,**1**);**  imagesc**(**X\_r**);**  title**(**'Original'**);**  axis off**;**  axis**(**'equal'**);**  b **=** subplot**(**1**,**2**,**2**);**  imagesc**(**Xf\_r**);**  title**(**'Reconstruido'**);**  axis off**;**  axis**(**'equal'**);**  linkaxes**([**a**,** b**]);** % Alinhamento de zoom  end\_time **=** cputime**;** % Tempo de excução final  %% Relatório de execução  printf**(**'Relatório de execução:\n'**);**  printf**(**'Tempo de execução: %.2f segundos.\n'**,** end\_time **-** str\_time**);**  printf**(**'Tamanho depois da etapa 1: %d bits.\n'**,** length**(**l\_coef**)\***9**);**  printf**(**'Tamanho depois da etapa 2: %d bits.\n'**,** length**(**bin**)\***9**);**  printf**(**'Taxa de compressão: %.2f%%.\n'**,** **(**1**-**length**(**bin**)/**length**(**l\_coef**))\***100**);** |

1. Conclusões

Foi realizado testes com imagens fullhd (1920x1080), foi utilizado o formato png pois é um formato de compressão de imagem sem perdas, e as diferenças ficam mais visíveis.

Para o primeiro teste, foi utilizado uma imagem com poucas cores.



É visível como esse método de compressão diminui a qualidade da imagem, principalmente quando há alteração de cor, isso faz com que textos e áreas da imagem onde há um grande contraste (preto e branco no exemplo), fiquem borradas ou destorcidas.





Pode-se observar que a distorção ocorre principalmente onde há o contraste de duas cores.

Relatório de execução:

Tempo de execução: 724.00 segundos.

Tamanho depois da etapa 1: 56636928 bits.

Tamanho depois da etapa 2: 2790477 bits.

Taxa de compressão: 95.07%.

Para o segundo teste, foi utilizada uma imagem com mais detalhes e cores.



Nessa imagem é possível observar os blocos 8x8 que o algoritmo separou para calcular a DCT.



Relatório de execução:

Tempo de execução: 716.19 segundos.

Tamanho depois da etapa 1: 56636928 bits.

Tamanho depois da etapa 2: 4276782 bits.

Taxa de compressão: 92.45%.

No último teste foi utilizada uma imagem menor (700x493)



Como essa imagem é menor e com dimensões não divisíveis por 8, a imagem passou por um processo de redimensionamento.



O tempo de execução também é bem menor.

Relatório de execução:

Tempo de execução: 109.61 segundos.

Tamanho depois da etapa 1: 9427968 bits.

Tamanho depois da etapa 2: 1036332 bits.

Taxa de compressão: 89.01%.

O MPEG é um algoritmo amplamente utilizado por ter uma boa taxa de compressão apesar da qualidade da imagem diminuir. Por isso se tornou amplamente utilizada principalmente na internet.