# Resultados obtidos.

As implementações foram feitas para tirar qualquer dúvida de o OpenCv estar influenciando nos resultados obtidos no artigo do SAC em relação ao PSNR. Não sabemos se a conversão RGB -> YCbCr e sua inversa são feitas com alguma subamostragem que possa deixar os resultados de PSNR baixo. Pode ser também que haja outras funções do OpenCV que estejam prejudicando o PSNR.

Com isso, fizemos o código que lê manualmente uma imagem e faz a trinca de conversões do espaço de cor RGB -> YCbCr -> RGB. Somente fazendo essas conversões diretamente em memória, trabalhando tudo com float e armazenando o resultado em um bmp (fazendo um round no float e armazenando em unsigned char), o PSNR se mantém 100dB (código professor Rudinei).

Implementei usando essa abordagem de round ao código já feito em OpenCV, e o resultado (já dentro da técnica da reversão) foi só um pouquinho melhor do que o já obtido. O PSNR-Red pra hei02.bmp foi de 33,6017dB (anteriormente era de 33,5916 dB).

Como eu fiquei na dúvida se eu não estava deixando alguma coisa passar despercebido ou se o OpenCV realmente influencia, voltei ao código que o professor me passou e fiz mais alguns testes. Lembrando que a gente estava trabalhando neste código com o processo direto RGB -> YCbCr -> RGB sem nenhum auxiliar. Separei então os processos I - (RGB -> YCbCr -> gravar em arquivo dat) e II - (ler arquivo dat -> YCbCr -> RGB). Segue a minha análise (Anexo: código01.cpp).  
  
1 - Gravação no arquivo dat utilizando **double** e **sem** round em I (o round está sendo feito em II para criar o bmp)  
PSNR-Red (4.bmp): 100 dB (7MB de arquivo .dat)  
  
2 - Gravação no arquivo dat utilizando **double** e **com** round em I (round também é feito em II para criar o bmp)  
PSNR-Red (4.bmp): **51,5541 dB**  
  
3 - Gravação no arquivo dat utilizando **int** e **com** round em I (round também é feito em II para criar o bmp)  
PSNR-Red (4.bmp): 51,5541 dB  
  
4 - Gravação no arquivo dat utilizando **int** e **sem** round em I (round também é feito em II para criar o bmp)  
PSNR-Red (4.bmp): 46,247 dB  
  
5 - Gravação no arquivo dat utilizando **char** e **com** round em I (round também é feito em II para criar o bmp)  
PSNR-Red (4.bmp): 51,5541 dB  
  
6 - Gravação no arquivo dat utilizando **char** e **sem** round em I (round também é feito em II para criar o bmp)  
PSNR-Red (4.bmp): 46,247 dB  
  
Com isso, eu concluí que o PSNR está caindo agora devido ao round para criar o arquivo intermediário.

Para provar isso, peguei o código que o professor Rudinei passou e fiz somente uma alteração, armazenando os valores durante a conversão RGB->YCbCr com round (Anexo: codigo02.cpp)  
Y[i][j] = round(a);  
Cb[i][j] = round(b);  
Cr[i][j] = round(c);  
O PSNR foi de 51,5541 dB novamente.  
  
Ou seja, a não ser que armazenemos o arquivo auxiliar (a color index table no código final) em float e sem round, o PSNR vai continuar pela metade...

A última prova que eu fiz foi adaptar o código da técnica da reversão para criar os arquivos dat (anáglifo verde-magenta com subsampling e a tabela de índice de cores) em float e sem round. Na reversão, o PSNR-Red (hei02.bmp) obtido foi de 33,6456Db, 4dB melhor. O Arquivo final (sem a compressão lossless) foi de 5,93MB contra 1,48MB quando armazenado com uchar e round (PSNR 33,6017Db). Anexo:codigo03.cpp

# Nova abordagem

Conversando com o professor, excluímos esse round em I. Nossa preocupação agora ficou em relação ao PSNR, deixando a compressão de lado, por enquanto**. A nova abordagem então é fazer todo o processo, armazenando em float o arquivo intermediário, lendo dele e então fazer o round para armazenar em um arquivo bmp, como já feito em 1. Fazer então a subamostragem 4:4:0 para verificar o quanto o PSNR cai. Com isso, vamos saber se é a subamostragem ou o OpenCV que está prejudicando nossos cálculos.**

Imagem testada: 4.bmp

PSNR:

|  |  |
| --- | --- |
| Y | 53,7873 dB |
| Cb | 47,3075 dB |
| Cr | 49,7291 dB |
| **Média** | **50,2746 dB** |
| R | 42,9696 dB |
| G | 50,3826 dB |
| B | 43,3991 dB |
| **Média** | **45,5838 dB** |

Ou seja, em média, o PSNR cai para 50 dB com a subamostragem. Armazenando valores em float. O arquivo intermediário (data.dat) ficou com 2,34MB. O arquivo original tinha 900KB.

O Léo fez um código utilizando as funções que ele criou. Os resultados de PSNR para a mesma imagem ficaram:

|  |  |
| --- | --- |
| Y | 58,9132 dB |
| Cb | 48,7629 dB |
| Cr | 50,6890 dB |
| **Média** | **52,7884 dB** |
| R | 46,3850 dB |
| G | 51,4761 dB |
| B | 42,5827 dB |
| **Média** | **46,8146 dB** |

Para tentar a chegar em PSNR parecido, comecei a verificar a solução de arredondamento que ele usa. O PSNR no meu código abaixou, então descartei. O Léo mandou uma outra solução trocando o round pela função de arredondamento dele. O PSNR foi similar (codigo5.cpp).

Passei então a verificar a subamostragem, para ver se ele fazia de um jeito diferente. Primeiro, imprimi os valores de RGB e YCbCr antes e depois da subamostragem, tanto no codigo4.cpp quanto no código do Léo. Descobri que estou fazendo a leitura dos dados RGB errado, pois devo ler na sequência BGR!!

Novos PSNR

|  |  |
| --- | --- |
| Y | 63,0551 dB |
| Cb | 47,3199 dB |
| Cr | 49,661dB |
| **Média** | **53,3453 dB** |
| R | 45,5082 dB |
| G | 52,0349 dB |
| B | 41,1804 dB |
| **Média** | **46,2412 dB** |

Pode ser observado que agora meu PSNR se equipara com o do Léo. E que realmente a subamostragem reduz o PSNR pela metade.

# Teste com armazenamento em char

Agora irei testar o armazenamento dos valores de YCbCr em estruturas char (codigo06.cpp).

PSNR

|  |  |
| --- | --- |
| Y | 58,7946 dB |
| Cb | 46,9720 dB |
| Cr | 49,1948 dB |
| **Média** | **51,6538 dB** |
| R | 44,9539 dB |
| G | 50,5486 dB |
| B | 40,8107 dB |
| **Média** | **45,4377 dB** |

Resultados aceitáveis. Agora meu arquivo possui 600KB (original: 900KB). O que é correto, já que eu reduzi 1/3 do arquivo com a remoção de metade dos dados de crominância.

# Teste com a conversão anaglífica

Agora falta o teste real, utilizando a conversão anaglífica.

## Conversão anaglífica, com o anáglifo verde-magenta sem ser subamostrado ou convertido para YCbCr + tabela de índice de cores formada pelo Cb e Cr subamostrado (4:4:0) do complementar. Dados armazenados em char ao fim da conversão.

Código: codigo07.cpp

Imagem: hei02.bmp

Tamanho original: 2,96MB

Tamanho comprimido (sem compressão lossless): 1,97MB

|  |  |  |  |
| --- | --- | --- | --- |
|  | PSNR Esquerdo | PSNR Direito | Média |
| Y | 34,3986 dB | 38,9035 |  |
| Cb | 39,1694 dB | 40,4391 |  |
| Cr | 37,1303 dB | 38,1112 |  |
| **Média** | **36,8994 dB** | **39,1513 dB** | **38,0254 dB** |
| R | 100 dB | 29,6557 |  |
| G | 28,4472 dB | 100 |  |
| B | 100 dB | 30,0646 |  |
| **Média** | **76,1491 dB** | **53,2401 dB** | **64,6946 dB** |

Como eu estou mantendo o anáglifo verde-magenta, o PSNR quando calculado separado é muito alto. Quando calculado junto (redimensionando a figura pra 1244+350 para poder usar o VQMT), ele é bem menor (34,1317 + 32,5485 + 34,7060) /3= 34,7954 RGB e (37,2826+41,1358+38,8489)/3= 39,0891 YCbCr.

## Conversão anaglífica, com o anáglifo verde-magenta sem ser subamostrado ou convertido para YCbCr + tabela de índice de cores formada pelo Cb e Cr subamostrado (4:4:0) do complementar. Dados armazenados em float ao fim da conversão.

Código: codigo08.cpp

Imagem: hei02.bmp

Tamanho original: 2,96MB

Tamanho “comprimido” (sem compressão lossless): 3,45MB

|  |  |  |  |
| --- | --- | --- | --- |
|  | PSNR Esquerdo | PSNR Direito | Média |
| Y | 34,4026 | 38,9109 |  |
| Cb | 39,1733 | 40,4819 |  |
| Cr | 37,1342 | 38,1317 |  |
| **Média** | **36,9034 dB** | **39,1748 dB** | **38,0391 dB** |
| R | 100 | 29,6694 |  |
| G | 28,4512 | 100 |  |
| B | 100 | 30,0807 |  |
| **Média** | **76,1504 dB** | **53,2500 dB** | **64,7002 dB** |

Valores similares ao encontrado armazenando a tabela de índice de cores em char.

## Conversão anaglífica, com ambos anáglifos sendo subamostrados. Dados armazenados em char ao fim da conversão.

Código: codigo09.cpp

Imagem: hei02.bmp

Tamanho original: 2,96MB

Tamanho comprimido (sem compressão lossless): 1,48MB

|  |  |  |  |
| --- | --- | --- | --- |
|  | PSNR Esquerdo | PSNR Direito | Média |
| Y | 32,4280 | 35,9761 |  |
| Cb | 40,4328 | 41,1808 |  |
| Cr | 38,5659 | 38,8756 |  |
| **Média** | **37,1422 dB** | **38,6775 dB** | **37,9099 dB** |
| R | 33,6015 | 29,6638 |  |
| G | 28,4437 | 36,7186 |  |
| B | 33,3915 | 30,0641 |  |
| **Média** | **31,8122 dB** | **32,1488 dB** | **31,9805 dB** |

Observando os dados de YCbCr, pode-se notar valores médios similares ao experimento anterior, porém neste com a vantagem de mais redução de dados por estar amostrando ambos os anáglifo. Os PSNR obtidos em RGB são similares aos já obtidos usando a implementação em OpenCV.

## [TESTE FINAL] Conversão anaglífica, com ambos anáglifos sendo subamostrados. Dados armazenados em float ao fim da conversão.

Código: codigo10.cpp

Imagem: hei02.bmp

Tamanho original: 2,96MB

Tamanho “comprimido” (sem compressão lossless): 5,93MB

|  |  |  |  |
| --- | --- | --- | --- |
|  | PSNR Esquerdo | PSNR Direito | Média |
| Y | 32,4409 | 35,9939 |  |
| Cb | 40,4767 | 41,2405 |  |
| Cr | 38,5910 | 38,8922 |  |
| **Média** | **37,1695 dB** | **38,6775 dB** | **38,7089 dB** |
| R | 33,6456 | 29,6694 |  |
| G | 28,4512 | 36,7714 |  |
| B | 33,4547 | 30,0807 |  |
| **Média** | **31,8505 dB** | **32,1488 dB** | **32,1738 dB** |

Valores médios pouca coisa superior ao experimento anterior. Armazena-se o dobro de informações do que o original. Os PSNR obtidos em RGB são similares (pouca coisa inferior) aos já obtidos utilizando a implementação em OpenCV.

## Conclusão

A implementação em OpenCV estava correta. Os valores dão isso mesmo. Ou eu errei novamente na implementação.