# Estratégia utilizando paleta de cores

Temos o par estéreo. Criamos o anaglífico, o qual contém o canal de cor vermelho da imagem direita e os canais verde e azul da imagem esquerda. Os canais restantes, aqui nomeados “paleta de cores”, devem ser armazenados em alguma estrutura e gravados como um arquivo auxiliar, o que servirá para o processo de reconstrução do par estéreo.

## 1) Abordagem utilizando uma imagem auxiliar

Comando de chamada do programa:

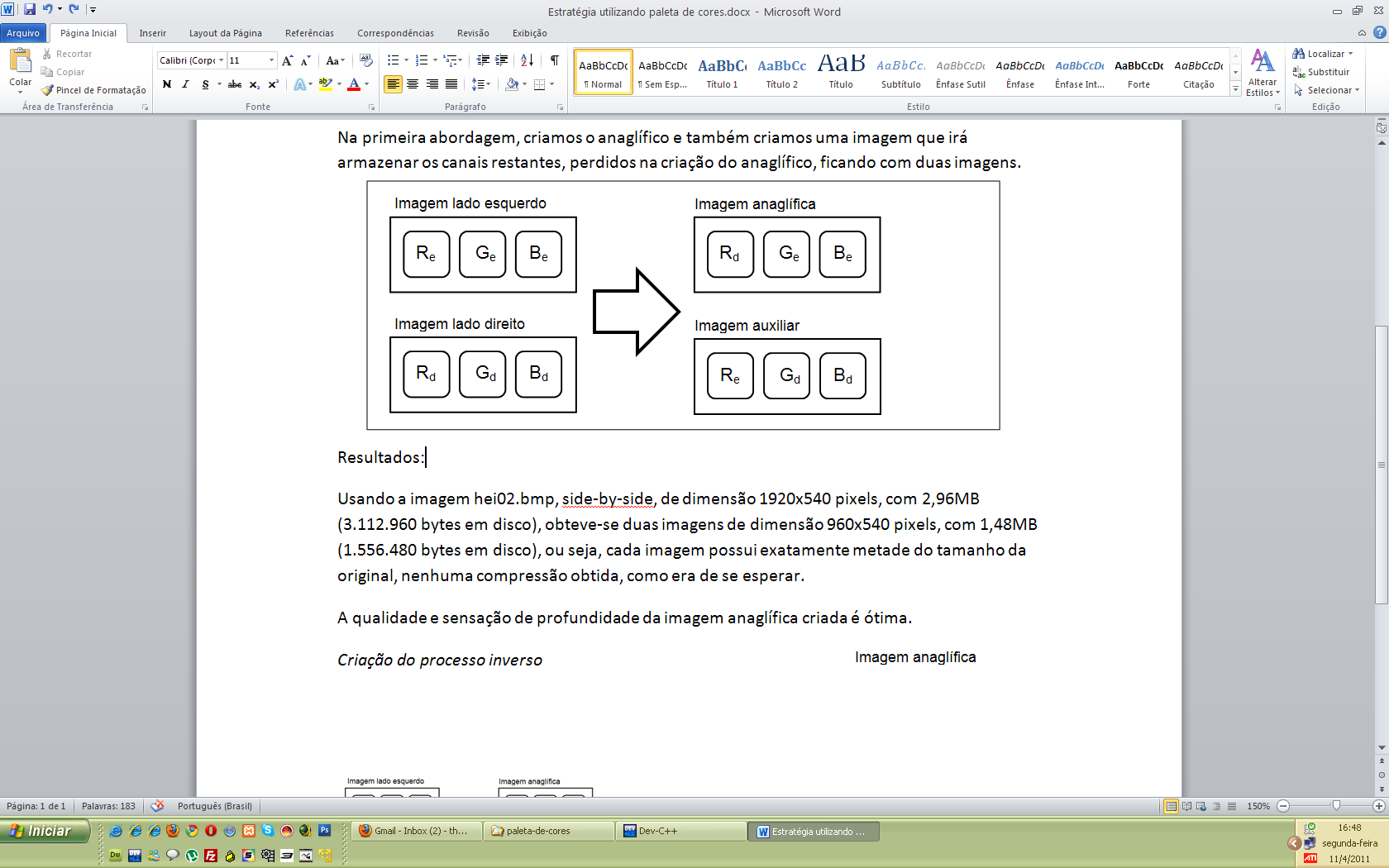
paleta-imagem <image.bmp> {O | 1} {-e | [-d <auxiliary\_image.bmp>]}

0: side-by-side image

1: above/below image

-e: encoding

-d: decoding



*Criação do processo de transformação anaglífica*

Na primeira abordagem, criamos o anaglífico e também criamos uma imagem que irá armazenar os canais restantes, perdidos na criação do anaglífico, ficando com duas imagens.

Resultados:

paleta-imagem hei02.bmp 0 -e

Usando a imagem hei02.bmp, side-by-side, de dimensão 1920x540 pixels, com 2,96MB (3.112.960 bytes em disco), obteve-se duas imagens de dimensão 960x540 pixels (anaglyph.bmp e auxiliary.bmp), com 1,48MB (1.556.480 bytes em disco), ou seja, cada imagem possui exatamente metade do tamanho da original, nenhuma compressão obtida, como era de se esperar.

A qualidade e sensação de profundidade da imagem anaglífica criada é ótima.

*Criação do processo inverso*

Para o processo inverso, devemos fazer um rearranjo das cores contidas tanto na imagem anaglífica quanto na auxiliar. Criamos um novo container de imagem no OpenCV que, dependendo do parâmetro passado pelo usuário, terá o dobro de largura da imagem anaglífica (para o caso de a imagem original ser do tipo side-by-side) ou o dobro de altura da imagem anaglífica (para o caso de a imagem original ser do tipo above/below).

Considerando o caso de a imagem original ser do tipo side-by-side, temos a seguinte estratégia: A primeira metade da largura do container criado, considerando o sentido da esquerda para a direita, deve receber os valores contidos no canal vermelho da imagem auxiliar e os valores contidos nos canais verde e azul da imagem anaglífica. A segunda metade do container criado receberá o inverso, os valores contidos nos canais verde e azul da imagem auxiliar e os valores do canal vermelho da imagem anaglífica. A estratégia é similar para imagens do tipo above/below, sendo tratado a metade de cima da altura do container criado e depois a metade de baixo.

Resultados:

paleta-imagem anaglyph.bmp 0 -d auxiliary.bmp

Obteve-se uma imagem (Par\_Estereo.bmp), do tipo side-by-side, de dimensão 1920x540px e tamanho 2,96MB (3.112.960 bytes em disco). Nenhum defeito encontrado. Um teste melhor é verificar o PSNR. Com isso, comprovamos que é possível fazer o processo reverso de uma imagem anaglífica. Resta agora estudarmos métodos para que a transformação anaglífica utilizando um arquivo auxiliar seja feita com compressão.

## 2) Abordagem utilizando vetores

A criação de uma imagem auxiliar, abordagem utilizada em 1, carrega dados que não nos interessam, como o cabeçalho da imagem. O realmente necessário é o que está disponível no campo imageData da estrutura IplImage utilizada no OpenCV. Desta forma, uma segunda abordagem é utilizar um vetor que armazene apenas os valores dos pixels de cada um dos canais removidos no processo anaglífico. Tal vetor posteriormente deve passar por algum algoritmo de compressão *lossless*. Novamente, devemos verificar se o processo reverso é possível e reverte a uma imagem sem perda de qualidade.

Comando de chamada ao programa

paleta-vetor <image.bmp> {O | 1} {-e | -d}

0: side-by-side image

1: above/below image

-e: encoding

-d: decoding

Resultados

paleta-vetor hei02.bmp 0 –c

No processo de transformação anaglífica, é criado um arquivo chamado pixelData.dat que contém os valores dos pixels removidos na criação do anaglífico. Podemos dividir este arquivo em três partes. A primeira terça parte contém os dados do canal vermelho da imagem à esquerda, a segunda terça parte contém os dados do canal verde da imagem à direita e por fim, a terceira terça parte contém os dados do canal azul da imagem à direita. O arquivo pode ser lido e armazenado em um vetor, e para acessar cada parte basta acrescentar um offset do tamanho da resolução da imagem (largura\*altura da imagem).

O tamanho do arquivo anaglífico foi de 1,48 MB (1.555.254 bytes) e o do arquivo pixelData.dat foi de 1,48MB (1.555.200 bytes). O arquivo original possui um tamanho de 2.96MB (3.110.456 bytes), ou seja, obteve-se a redução de apenas 2bytes comparado ao arquivo original. Tal redução deve ser maior ao aplicarmos uma compressão *lossless* em pixelData.dat. Fizemos um teste zipando o arquivo pixelData.dat utilizando o programa WinRAR. O tamanho do arquivo ficou em 855KB (875.542 bytes), totalizando 2,31 MB (2.430.796 bytes) com a imagem anaglífica, resultando em 22% de compressão. Isso comprova que é possível obter uma taxa razoável de compressão aplicando algoritmos do tipo *lossless*.

paleta-vetor anaglyph.bmp 0 -d

O resultado da reversão do método gerou novamente o par estéreo com sucesso, nenhum defeito encontrado. Um teste melhor é verificar o PSNR.

Para assegurar o resultado positivo da reversão, o par estéreo obtido foi novamente submetido ao programa para a formação da imagem anaglífica. O resultado foi satisfatório, gerando uma imagem anaglífica com sensação de profundidade.

## 3) Cálculo do PSNR para a abordagem utilizando vetores

O que é o PSNR?

Peak-to-peak signal-to-noise ratio, é uma métrica medida em decibéis que relaciona a quantidade de ruído introduzida em uma imagem ou um vídeo (sequência de imagens) após um processo de compressão com perdas, ou seja, mede quão semelhante ficou a imagem após a compressão. Quanto maior o PSNR, maior a semelhança (ou menor a diferença), sendo 100 o valor para imagens iguais. O cálculo do PSNR é bom tanto para saber o quanto de ruído estamos inserindo na imagem com a nossa técnica quanto para compararmos esta com outras técnicas disponíveis.

Iremos utilizar o software desenvolvido pela MSU Graphics & Media Lab (Video Group), chamado MSU VQMT (Video Quality Measurement Tool), que oferece diversas métricas envolvendo imagens e vídeos. Em sua versão free, possibilita o cálculo do PSNR conforme desejamos.

Na instalação, ele pode pedir para fazer downloads adicionais de plug-ins e codecs de vídeo. Eu instalei todos.

Um problema é que o software na versão free não permite análise de imagens e vídeos com resoluções maiores do que 1280x720px. Dessa forma, a imagem original foi redimensionada para 1244x340px e rodamos novamente nosso programa.

Os testes foram feitos em três etapas, para se analisar o PSNR por canal. Ou seja, foi calculado o PSNR das imagens para o canal vermelho, depois para o canal verde e por fim, para o canal azul.

Resultados:

O PSNR para todos os canais obteve a média de 100dB, ou seja, as imagens são semelhantes. O mesmo se deu na comparação do anaglífico criado com a imagem original e o anaglífico criado com a imagem originada na reversão do método.

O software gera dois arquivos por teste, um .CSV que contém informações sobre a métrica utilizada, as imagens analisadas e a média do PSNR. O outro arquivo é uma imagem contendo as diferenças entre as duas imagens utilizadas.

Os resultados podem ser vistos em:

psnr\_rrgb\_hei02-1244x350\_Par\_Estereo.csv

psnr\_grgb\_hei02-1244x350\_Par\_Estereo.csv

psnr\_brgb\_hei02-1244x350\_Par\_Estereo.csv

psnr\_brgb\_anaglyph-original\_anaglyph-Par-Estereo.csv

psnr\_rrgb\_hei02-1244x350\_Par\_Estereo.bmp

psnr\_grgb\_hei02-1244x350\_Par\_Estereo.bmp

psnr\_brgb\_hei02-1244x350\_Par\_Estereo.bmp

psnr\_brgb\_anaglyph-original\_anaglyph-Par-Estereo.bmp

## 4 - Transformação anaglífica verde/magenta utilizando modelo de cores YUV com amostragem de cor 4:2:2

Ao final do processo de conversão anaglífica, podemos trabalhar um pouco mais com a imagem para obter um pouco mais de compressão. Para isso, iremos converter a imagem anaglífica do espaço de cores RGB para YUV. Sendo assim, poderemos aplicar uma amostragem de cor (*subsampling*) 4:2:2 de modo a remover um pouco da informação de cor, promovendo compressão (lossy). A amostragem 4:2:2 faz com que, para cada 4 pixels de luminância (Y), tenhamos apenas 2 de U e 2 de V, reduzindo pela metade o intervalo de cores. Iremos novamente aplicar o PSNR para verificar se o nível de ruído inserido é aceitável e se a percepção de profundidade não se perde. Iremos utilizar a transformação anaglífica verde/magenta, pois esta nos artigos do Léo se mostraram melhores na visualização.

A transformação de RGB para YUV envolve cálculos com float. Se armazenarmos o valor em float, teremos um arquivo 4 vezes maior, já que o float consome 4 bytes.

A fórmula de transformação e reversão passada pelo Léo é esta:

//conversao RGB para YUV

for(j = 0; j < file\_in.larg; j++)

for(i = 0; i < file\_in.alt; i++)

{

file\_in.Y[j][i] = 0.299f \* file\_in.R[j][i] + 0.587f \* file\_in.G[j][i] + 0.114f \* file\_in.B[j][i];

file\_in.U[j][i] = 0.565f \* file\_in.B[j][i] - file\_in.Y[j][i];

file\_in.V[j][i] = 0.713f \* file\_in.R[j][i] - file\_in.Y[j][i];

}//for(i = 0; i < file\_in.alt; i++)

E a reversão:

//conversao YUV para RGB

for(j = 0; j < file\_in.larg; j++)

for(i = 0; i < file\_in.alt; i++)

{

file\_out.R[j][i] = file\_out.Y[j][i] + 1.403f \* file\_out.V[j][i];

file\_out.G[j][i] = file\_out.Y[j][i] - 0.344f \* file\_out.U[j][i] - 0.714f \* file\_out.V[j][i];

file\_out.B[j][i] = file\_out.Y[j][i] + 1.77f \* file\_out.U[j][i];

}//for(i = 0; i < file\_in.alt; i++)

Uma estratégia é arredondar o valor do float para caber em um signed char. O arredondamento foi manual, já que a linguagem C sempre arredonda um número para baixo usando trunc() ou para cima usando ceil(), então, foi criada uma função que verifica a primeira casa decimal do número, arredondando para baixo valores <= 5 e para cima valores > 5. O problema é que com isso conseguimos armazenar números no intervalo de -128 a 128 (28), ou seja, números fora desse intervalo serão aproximados para um dos extremos.

Resultados utilizando **signed char** fez com que a imagem na reversão de YUV para RGB apresentasse muito ruído (**PSNR de 22,06dB** para a imagem anaglyphYUV2RGB-signedchar.bmp – com **arredondamento manual**. O **PSNR foi de 28,72** para a mesma imagem, só que com arredondamento usando **cast para char**).

Outra estratégia foi utilizar **unsigned char**, cujo intervalo vai de 0 a 255. Novamente, há problema com os números fora desse intervalo, com o adicional de que não estamos armazenando nenhum negativo. O **PSNR neste caso foi de 6,15dB**.

Temos também a opção de utilizar unsigned char e fazer o shift do resultado para obtermos o intervalo -128 a 128, porém, intuitivamente não iremos obter resultados melhores do que utilizando signed char.

O OpenCV possui sua própria função para converter RGB para YCrCb (cvCvtColor(img, dst, CV\_BGR2YCrCb)), com a fórmula abaixo:

RGB<=>YCrCb JPEG (a.k.a. YCC) (CV\_BGR2YCrCb, CV\_RGB2YCrCb, CV\_YCrCb2BGR, CV\_YCrCb2RGB)

Y <- 0.299\*R + 0.587\*G + 0.114\*B

Cr <- (R-Y)\*0.713 + delta

Cb <- (B-Y)\*0.564 + delta

R <- Y + 1.403\*(Cr - delta)

G <- Y - 0.344\*(Cr - delta) - 0.714\*(Cb - delta)

B <- Y + 1.773\*(Cb - delta),

{ 128 for 8-bit images,

where delta = { 32768 for 16-bit images

{ 0.5 for floating-point images

Y, Cr and Cb cover the whole value range.

Utilizando a **função do OpenCV**, obtivemos **PSNR de 51,71** dB na imagem anaglyphYUV2RGB-openCV.bmp

Ao tentar **reproduzir a fórmula do OpenCV**, obtivemos **PSNR de 28,28 dB** com **arredondamento manual** (anaglyphYUV2RGB-cvFormulamanual.bmp) e **42,20dB com cast** (anaglyphYUV2RGB-cvFormula-cast.bmp).

Com isso parece que utilizar a fórmula do OpenCV se mostra a melhor opção.