Gabriel CV

Luís Gabriel P. Condados

Sumário

Prefácio

Programas/atividades desenvolvidas para a disciplina DIM0141 - VISÃO COMPUTACIONAL, da **Universidade Federal do Rio grande do Norte** UFRN

Prefácio

Todos os programas, neste documento, foram desenvolvidas em `C++` utilizando-se da biblioteca ***OpenCV*** e em ambiente *Linux*. Para compilar qualquer programa presente neste documento, pode-se fazer uso deste Makefile, coloca o Makefile na mesma pasta do código fonte, extensão .cpp, e execute via terminal o comando `make <nome_do_programa>`. Todos os códigos encontram-se no Repositório do github.

Chapter 1. Projetos e Tarefas da Primeira Unidade

1.1. Tarefa 1

1. *Descreva a distribuição de células fotorreceptoras na retina e seu impacto na percepção visual*

   Possuimos dois tipos de células fotorreceptoras, são elas os *cones* e os *bastonetes*. Resumidamente os *cones* são responsaveis pela percepção das cores e os *bastonetes* pela intensidade luminosa, os cones predominantemente se localizam em um ponto na retina, chamado de fovea, já os bastonetes ficam distribuidos na periferia dessa região, essa distribuição faz com que os seres humanos possuam um ponto focal no centro do seu campo de visão, essa região central é a que possui maior nitidez, e devido a distribuição dos bastonetes, e sua sensibilidade a luminosidade, eles são responsaveis pela visão noturna e nossa visão periférica.

2. *Qual a diferença entre os cones S, M e L? Quais deles são mais ativados quando uma luz amarela é projetada na retina?*

   A imagem a seguir resume bem o papel de cada tipo de cone na nossa perceção de cor, o tipo S é responsavel pela perceção da faixa mais proxima ao azul, M reage principalmente as frequencias proximas do verde e o L ao vermelho. A luz amarela possui comprimento de onda entre 565 e 590 nm, portando podemos observar pelo gráfico que é uma combinação, entre a reação do cone M com o cone L, provoca a reação que nos faz perceber a cor amarela.

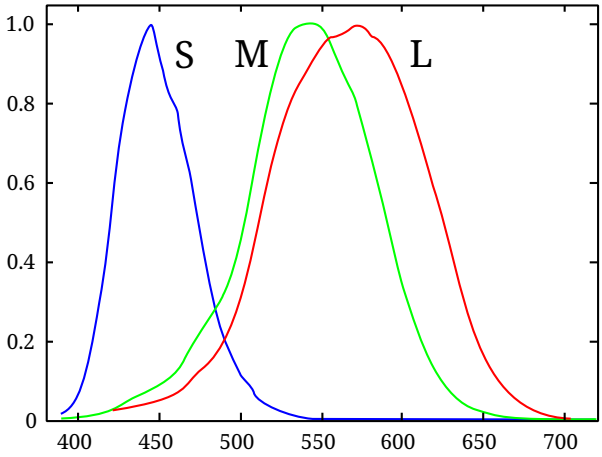

Figura 1. Simplified human cone response curves

- Escreva um programa para
  - abrir uma imagem e exibir na tela os 3 canais separadamente

Compilando e Executando.

```
$ make q3a
$ ./q3a <caminho_para_a_imagem>
```

Exemplo de funcionamento

Figura 2. Entrada do programa

Figura 3. Apenas o Canal R

Figura 4. Apenas o Canal G

Figura 5. Apenas o Canal B

Download do código completo: q3a.cpp

<details><summary> <mark>Clique aqui pra ver o código completo</mark> </summary>

```
//Programa que separa os três canais (RGB) e exibi uma imagem de cada canal
#include <opencv2/opencv.hpp>
using namespace cv;

int main(int argc, char**argv)
{
  Mat img;
  Mat ch[3];

  img = imread(argv[1],CV_LOAD_IMAGE_COLOR);

  if(!img.data){
    std::cout << "imagem nao carregou corretamente\n";
    return(-1);
  }
  imshow("Original", img);

  ch[0] = Mat(img.size(), CV_8UC3);
  ch[1] = Mat(img.size(), CV_8UC3);
  ch[2] = Mat(img.size(), CV_8UC3);

  for(int y = 0; y < img.rows; y++)
  for(int x = 0; x < img.cols; x++)
  {
    ch[0].at<Vec3b>(y,x)[0] = img.at<Vec3b>(y,x)[0];//B
    ch[1].at<Vec3b>(y,x)[1] = img.at<Vec3b>(y,x)[1];//G
    ch[2].at<Vec3b>(y,x)[2] = img.at<Vec3b>(y,x)[2];//R
  }

  imshow("Canal B", ch[0]);
  imshow("Canal G", ch[1]);
  imshow("Canal R", ch[2]);

  imwrite("canal_B.png", ch[0]);
  imwrite("canal_G.png", ch[1]);
  imwrite("canal_R.png", ch[2]);

  imwrite("entrada_q3a.png", img);

  waitKey(0);
  return 0;
}
```

</details>

- abrir uma imagem e exibir na tela a imagem invertida horizontalmente

Compilando e Executando.

```
$ make q3b
$ ./q3b <caminho_para_a_imagem>
```

Exemplo de funcionamento

Figura 6. Entrada do programa

Figura 7. Imagem invertida horizontalmente

Download do código completo: q3b.cpp

<details><summary> <mark>Clique aqui pra ver o código completo</mark> </summary>

```
//Este programa recebe uma imagem como entrada e inverte a mesma, espelha na vertical,
//exibi e salva o resultado
#include <opencv2/opencv.hpp>
using namespace cv;

int main(int argc, char**argv)
{
  Mat img;
  Mat imgFlip;

  img = imread(argv[1],CV_LOAD_IMAGE_COLOR);

  if(!img.data){
    std::cout << "imagem nao carregou corretamente\n";
    return(-1);
  }
  imshow("Original", img);

  imgFlip = img.clone();

  for(int y = 0; y < img.rows; y++)
  for(int x = 0; x < img.cols; x++)
  {
    imgFlip.at<Vec3b>(y,img.cols - 1 - x) = img.at<Vec3b>(y,x);
  }

  imshow("Imagem Invertida Horizontalmente", imgFlip);
  imwrite("entrada_q3b.png", img);
  imwrite("flipHorizontal.png", imgFlip);

  waitKey(0);
  return 0;
}
```

</details>

- abrir duas imagens (a e b) de mesmo tamanho e exibir na tela uma nova imagem (c) com o blending entre ambas, usando uma combinação linear entre elas

Compilando e Executando.

```
$ make q3c
$ ./q3c <caminho_para_a_imagem A> <caminho_para_a_imagem B>
```

Exemplo de funcionamento

▶ https://www.youtube.com/watch?v=KDlCsc6b9kg *(YouTube video)*

Demonstração Blending

Download do código completo: q3c.cpp

<details><summary> <mark>Clique aqui pra ver o código completo</mark> </summary>

```
#include <opencv2/opencv.hpp>
using namespace cv;

const char* mainWindow = "Blending";

Mat imgA, imgB;
Mat imgC;

void blending(int, void*);

int main(int argc, char**argv)
{

  if(argc != 3)
  {
    std::cout << "Voce deve informar o caminho de duas imagens de mesmo tamanho\n";
    return(-1);
  }

  imgA = imread(argv[1],CV_LOAD_IMAGE_COLOR);
  imgB = imread(argv[2],CV_LOAD_IMAGE_COLOR);

  //Teste se as imagens foram carregadas corretamente
  if(!imgA.data){
    std::cout << "imagem "<< argv[1] <<"nao carregou corretamente\n";
    return(-1);
  }
  if(!imgB.data){
    std::cout << "imagem "<< argv[2] <<"nao carregou corretamente\n";
    return(-1);
  }
  //Testa se as imagens possuem o mesmo tamanho
  if(imgA.size() != imgB.size())
  {
    std::cout << "As imagens devem possuir o mesmo tamanho\n";
    return(-1);
  }

  imgC = imgB.clone();

  //Cria o slider, para ajustar a variavel alpha do calculo do blending
  namedWindow(mainWindow, WINDOW_AUTOSIZE);
  createTrackbar( "Alpha [0-100%]",  mainWindow,
                                     NULL,
                                     100,
```

```
                                    blending);
  int key;
  while(true)
  {
    imshow(mainWindow, imgC);
    key = waitKey(10);
    if(key == 27)//esq
      break;
  }
  return 0;
}

void blending(int pos, void*)
{
  float alpha = pos/100.0;
  for(int y = 0; y < imgA.rows; y++)
  for(int x = 0; x < imgA.cols; x++)
  {
      imgC.at<Vec3b>(y,x) = imgA.at<Vec3b>(y,x)*alpha + imgB.at<Vec3b>(y,x)*(1.0 -
alpha);
  }
}
```

</details>

- salvar uma nova imagem com o seguinte gradiente vertical

Compilando e Executando.

```
$ make q3d
$ ./q3d  <caminho_para_a_imagem>
```

Exemplo de funcionamento

Figura 8. Entrada do programa

Figura 9. Gradiente utilizado na imagem

Figura 10. Imagem de entrada x Gradiente

Download do código completo: q3d.cpp

<details><summary> <mark>Clique aqui pra ver o código completo</mark> </summary>

```cpp
#include <opencv2/opencv.hpp>
using namespace cv;

int main(int argc, char**argv)
{
  Mat img;
  Mat result;
  Mat gradient;

  img = imread(argv[1],CV_LOAD_IMAGE_GRAYSCALE);

  if(!img.data){
    std::cout << "imagem nao carregou corretamente\n";
    return(-1);
  }
  imshow("Original", img);

  result   = Mat(img.size(), CV_32FC1);
  gradient = img.clone();

  for(int y = 0; y < img.rows; y++)
  for(int x = 0; x < img.cols; x++)
  {
    result.at<float>(y,x) = (img.at<uint8_t>(y,x)/255.0)*y*(1.0/img.rows);
    gradient.at<uint8_t>(y,x) = y*(255.0/img.rows);
  }
  //converte para um formato que eh possivel salvar, neste caso equivalente ao
grayscale

//https://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#void%20Mat::conve
rtTo(OutputArray%20m,%20int%20rtype,%20double%20alpha,%20double%20beta)%20const
  result.convertTo(result, CV_8UC1, 255);

  imshow("Resultado", result);
  imshow("Gradiente", gradient);

  imwrite("gradiente.png", gradient);
  imwrite("imagem_com_gradiente.png", result);
  imwrite("entrada_q3d.png", img);

  waitKey(0);
  return 0;
}
```

</details>

---

1. *Considere o formato de imagem NetPBM*

Qual a diferença entre os números mágicos P1, P2, P3, P4, P5 e P6?

P1: BitMap, extenção .pbm, valores: 0-1 (Preto e Branco), ASCII

```
P2: GrayMap, extenção .pgm, valores:0-255, ASCII
```

```
P3: PixMap, extenção .ppm, valores: 3 canais(RGB) 0-255 cada canal, ASCII
```

```
P4: BitMap, extenção .pbm, valores: 0-1 (Preto e Branco),  Binário
```

```
P5: GrayMap, extenção .pgm, valores:0-255, Binário
```

```
P6: PixMap, extenção .ppm, valores: 3 canais(RGB) 0-255 cada canal, Binário
```

Converta uma imagem jpg para PBM (ASCII) utilizando convert e display para exibir

```
$ convert -compress none cafe.jpg cafe_ascii.pbm
$ display  cafe_ascii.pbm
```

Figura 11. Imagem Original

[cafe ascii] | *unidade1/cafe_ascii.pbm*

Figura 12. Imagem convertida para pbm

Imagem convertida: cafe.pbm

Converta a mesma imagem para PBM (binário) e para PPM (binário). Compare o tamanho dos 4 arquivos de imagem

Arquivo	Tamanho
cafe.jpg	98.9Kb
cafe_ascii.pbm	600.5Kb

cafe_binary.pbm	37.5Kb
cafe_ascii.ppm	3.2Mb
cafe_binary.ppm	900Kb

Por que o formato binário ocupa menos espaço que o formato ASCII?

Cada caracter ASCII ocupa 8 bits, já no binário cada bit representa um valor de pixel, no PBM já no PPM, cada conjunto de 3 caracteres asciis represantam um pixel(RGB).

Por que o formato PPM binário ocupa mais espaço que o formato PBM binário?

O formato PPM é um PixMap já o PBM é um BitMap, ou seja, cada unidade do PPM é é formado por um conjunto de 3 bytes, para representar as componentes RGB do pixel, já no padrão PBM, cada unidade representa um unico valor, 0 ou 1, ou seja 1 bit por unidade (pixel de 1 bit).

Quais desses formatos são vetoriais e quais são bitmaps? BMP, SVG, JPG, EPS, PNG

Formatos bitmaps

BPM, JPEG, PNG.

Formatos vetorais

SVG, EPS.

Imagens de algumas aplicações possuem um nível de ruído considerável, principalmente aquelas que captam em níveis baixos de iluminação, como na captura de imagens astronômicas. Uma das formas de atenuar esse tipo de ruído é através da média de inúmeras imagens. Utilizando as 9 imagens disponibilizadas, crie um programa que gere uma nova imagem com o ruído atenuado.

Compilando e Executando.

```
$ make q6
$ ./q6
```

Exemplo de funcionamento

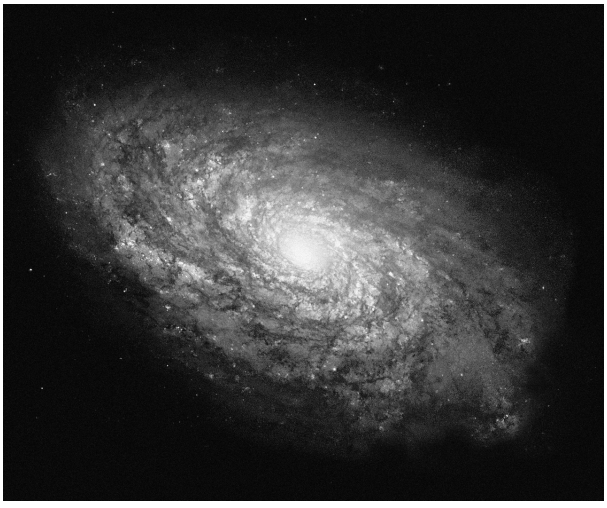

Figura 13. Imagem Media, Imagem filtrada

Download do código completo: q6.cpp

Entrada utilizada: imagensComRuido

<details><summary> <mark>Clique aqui pra ver o código completo</mark> </summary>

```cpp
//Programa que tira a media aritmetica de N imagens
//Este programa carrega todas as imagens de um arquivo, cujo caminho (path)
//esta indicado pela variavel path, e calcula a media de todas essas imagens
#include <opencv2/opencv.hpp>
using namespace cv;

int main(int argc, char**argv)
{
  String path("./imagensComRuido/*.jpg");
  std::vector<String> fn;
  std::vector<Mat> imagens;
  Mat media;

  glob(path, fn, true);
  //carrega as N imagens
  for(size_t i = 0; i < fn.size(); i++)
  {
    Mat im = imread(fn[i], CV_LOAD_IMAGE_GRAYSCALE);
    if(!im.data)continue;//caso falhe na leitura, vai para o proximo arquivo
    imagens.push_back(im);
  }

  if(imagens.empty())
  {
    std::cerr << "Erro ao carrega as imagens de " << path << "\n";
    return -1;
  }

  media = Mat::zeros(imagens[0].size(), CV_32FC1);
  //calcula da media das N imagens
  for(size_t i = 0; i < imagens.size(); i++)
  {
    for(int y = 0; y < imagens[0].rows; y++)
    for(int x = 0; x < imagens[0].cols; x++)
    {
      media.at<float>(y,x) += imagens[i].at<uint8_t>(y,x)/(float)imagens.size();
    }
  }
  media.convertTo(media, CV_8UC1, 1.0, 0);

  imshow("Media", media);
  imwrite("Media.png",media);

  waitKey(0);
  return 0;
}
```

</details>

Chapter 2. Bibliografia

- Rafael Gonzalez. 'Processamento Digital de Imagens'. Addison-Wesley. 1990. 2 ed.