Ajuste de brilho/contraste, limiarização, limiarização de Otsu, histograma.

Transformações de pixel

- Neste tipo de processamento de imagem, o valor de cada pixel de saída depende somente do valor do pixel de entrada correspondente (mais, potencialmente, alguma informação global ou parâmetros).
- Exemplos de tais operadores incluem ajuste de brilho e contraste e correção e transformações de cor.

Ajuste de brilho e contraste

- Duas transformações de pixel comuns são multiplicação e adição por constantes (http://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html):

$$g(i,j)=\alpha\cdot f(i,j)+\beta$$

- Os parâmetros $\alpha>0$ e β são chamados de *gain* e *bias*; esses parâmetros controlam *contraste* e *brilho* da imagem.
- Para separação de modificações de brilho e contraste, pode-se adotar (http://math.stackexchange.com/questions/906240/algorithms-to-increase-or-decrease-the-contrast-of-an-image):

$$g(i,j)=\alpha\cdot(f(i,j)-128)+128+\beta$$

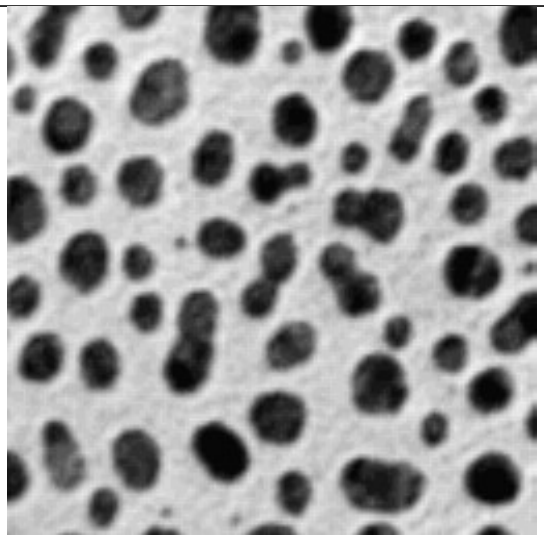
- Para especificar brilho β e contraste α no intervalo de -1 a +1:

$$x=f(i,j)/255$$

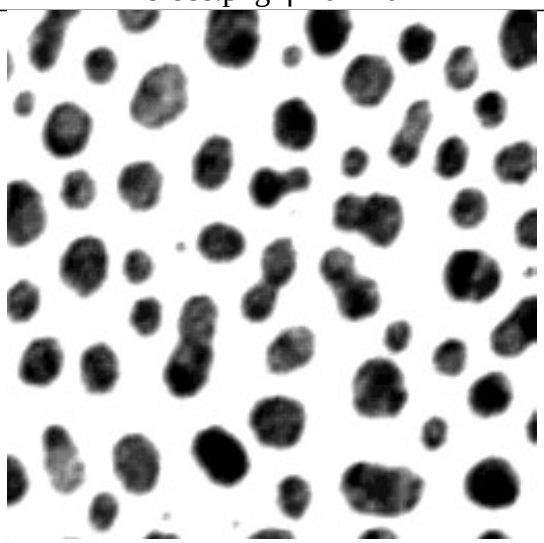
$$y=\begin{cases}(x+\beta-0.5)\cdot(1+\alpha)+0.5, & \text{se } -1\leq\beta<0\\ (x+\beta-0.5)\cdot(1/(1-\alpha))+0.5, & \text{se } 0\leq\beta<1\\ (x+\beta-0.5)/\varepsilon+0.5, & \text{se } \beta=1\end{cases}$$

$$f(i,j)=255\cdot y$$

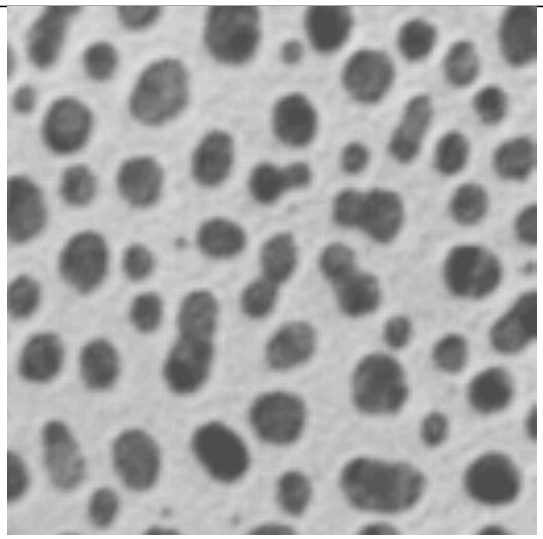
```
//brilcong.cpp - 2017
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=5) {
    printf("BrilConG: Ajusta brilho/contraste de Mat_<GRY>\n");
    printf("BrilConG ent.pgm sai.pgm brilho contraste\n");
    printf("  brilho e contraste no intervalo -1.0 a +1.0\n");
    erro("Erro: Numero de argumentos invalido");
  }

  Mat_<FLT> a; le(a,argv[1]);

  double brilho;
  if (sscanf(argv[3],"%lf",&brilho)!=1) xerro1("Erro: Leitura brilho");
  if (brilho<-1.0 || 1.0<brilho) xerro1("Erro: Brilho fora do intervalo");

  double contraste;
  if (sscanf(argv[4],"%lf",&contraste)!=1) xerro1("Erro: Leitura contraste");
  if (contraste<-1.0 || 1.0<contraste) xerro1("Erro: contraste fora do intervalo");

  for (int i=0; i<a.total(); i++)
    if (contraste<0.0) // contraste negativo
      a(i)=(a(i)+brilho-0.5)*(1.0+contraste)+0.5;
    else if (contraste<1.0) // contraste positivo ou zero
      a(i)=(a(i)+brilho-0.5)*(1.0/(1.0-contraste))+0.5;
    else // contraste = 1
      a(i)=(a(i)+brilho-0.5)/epsilon+0.5;
  imp(a,argv[2]);
}

//brilcong.cpp - pos2018
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=5) {
    printf("BrilConG: Ajusta brilho/contraste de Mat_<GRY>\n");
    printf("BrilConG ent.pgm sai.pgm brilho contraste\n");
    printf("  brilho e contraste no intervalo -1.0 a +1.0\n");
    erro("Erro: Numero de argumentos invalido");
  }

  Mat_<FLT> a; le(a,argv[1]);

  double brilho;
  convArg(brilho,argv[3]);
  if (brilho<-1.0 || 1.0<brilho) xerro1("Erro: Brilho fora do intervalo");

  double contraste;
  convArg(contraste,argv[4]);
  if (contraste<-1.0 || 1.0<contraste) xerro1("Erro: contraste fora do intervalo");

  for (unsigned i=0; i<a.total(); i++)
    if (contraste<0.0) // contraste negativo
      a(i)=(a(i)+brilho-0.5)*(1.0+contraste)+0.5;
    else if (contraste<1.0) // contraste positivo ou zero
      a(i)=(a(i)+brilho-0.5)*(1.0/(1.0-contraste))+0.5;
    else // contraste = 1
      a(i)=(a(i)+brilho-0.5)/epsilon+0.5;

  imp(a,argv[2]);
}



REM roda.bat
brilcong blobs.png bc+15+40.png  0.15  0.40
brilcong blobs.png bc+15-40.png  0.15 -0.40
brilcong blobs.png bc-15+40.png -0.15  0.40
brilcong blobs.png bc-15-40.png -0.15 -0.40

threshg blobs.png th+0+0.png 127
threshg bc+15+40.png th+15+40.png 127
threshg bc+15-40.png th+15-40.png 127
threshg bc-15+40.png th-15+40.png 127
threshg bc-15-40.png th-15-40.png 127
```



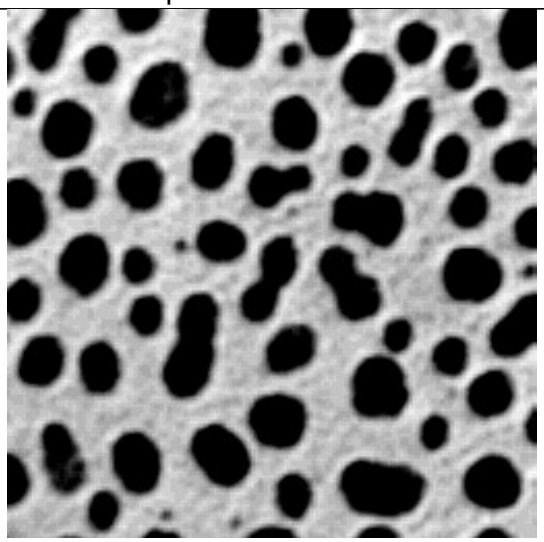
blobs.png β=0 α=0



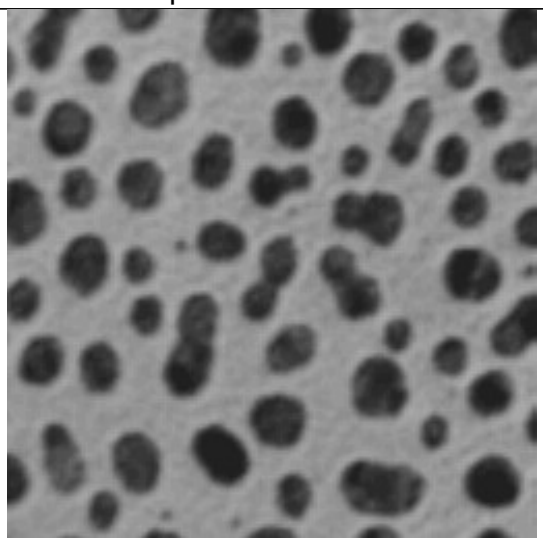
β=+0.15 α=+0.4



β=+0.15 α=-0.4



β=-0.15 α=+0.4



β=-0.15 α=-0.4

Imagens obtidas fazendo ajuste de brilho β e contraste α.

	
Imagem original: img3.pgm	Imagem com brilho β=+0.3 e contraste α=+0.5.

Ajuste de brilho/contraste de imagem colorida

Pode ser feita:

1) Ajustando brilho/contraste de cada banda RGB.

2) Convertendo a imagem para CieLAB (ou outro sistema de cores em que coloração e intensidade estejam separadas), ajustando L, e reconvertendo para RGB.

```
//brilrgb.cpp - Ajusta brilho/contraste de cada plano RGB
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=5) {
    printf("BrilRGB: Ajusta brilho/contraste de Mat_<COR> em cada plano RGB\n");
    printf("BrilRGB ent.ppm sai.ppm brilho contraste\n");
    printf("  brilho e contraste no intervalo -1.0 a +1.0\n");
    erro("Erro: Numero de argumentos invalido");
  }

  double brilho;
  convArg(brilho,argv[3]);
  if (brilho<-1.0 || 1.0<brilho) xerro1("Erro: Brilho fora do intervalo");

  double contraste;
  convArg(contraste,argv[4]);
  if (contraste<-1.0 || 1.0<contraste) xerro1("Erro: contraste fora do intervalo");

  Mat_<COR> ent; le(ent,argv[1]);
  Mat_<CORF> entf; converte(ent,entf);

  for (unsigned i=0; i<entf.total(); i++)
    for (unsigned p=0; p<3; p++)
      if (contraste<0.0) // contraste negativo
        entf(i)[p]=(entf(i)[p]+brilho-0.5)*(1.0+contraste)+0.5;
      else if (contraste<1.0) // contraste positivo ou zero
        entf(i)[p]=(entf(i)[p]+brilho-0.5)*(1.0/(1.0-contraste))+0.5;
      else // contraste = 1
        entf(i)[p]=(entf(i)[p]+brilho-0.5)/epsilon+0.5;

  Mat_<COR> sai; converte(entf,sai);
  imp(sai,argv[2]);
}
```

```
//brilconc.cpp - ajusta brilho/contraste no sistema CieLab
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=5) {
    printf("BrilConC: Ajusta brilho/contraste de Mat_<COR> no sistema CieLab\n");
    printf("BrilConC ent.ppm sai.ppm brilho contraste\n");
    printf("  brilho e contraste no intervalo -1.0 a +1.0\n");
    erro("Erro: Numero de argumentos invalido");
  }

  double brilho;
  convArg(brilho,argv[3]);
  if (brilho<-1.0 || 1.0<brilho) xerro1("Erro: Brilho fora do intervalo");

  double contraste;
  convArg(contraste,argv[4]);
  if (contraste<-1.0 || 1.0<contraste) xerro1("Erro: contraste fora do intervalo");

  Mat_<COR> ent; le(ent,argv[1]);
  Mat_<COR> lab; cvtColor(ent,lab,CV_BGR2Lab);
  vector< Mat_<GRY> > lab_planes(3);
  split(lab, lab_planes);  // now we have the L image in lab_planes[0]
```

```
  Mat_<FLT> light; converte(lab_planes[0], light);
  for (unsigned i=0; i<light.total(); i++)
    if (contraste<0.0) // contraste negativo
      light(i)=(light(i)+brilho-0.5)*(1.0+contraste)+0.5;
    else if (contraste<1.0) // contraste positivo ou zero
      light(i)=(light(i)+brilho-0.5)*(1.0/(1.0-contraste))+0.5;
    else // contraste = 1
      light(i)=(light(i)+brilho-0.5)/epsilon+0.5;
  converte(light,lab_planes[0]);

  merge(lab_planes,lab);
  Mat_<COR> sai;
  cvtColor(lab,sai,CV_Lab2BGR);
  imp(sai,argv[2]);
}
```

Imagem original (img3 do Leuven de Mikolajczyk)

Ajuste nas bandas RGB

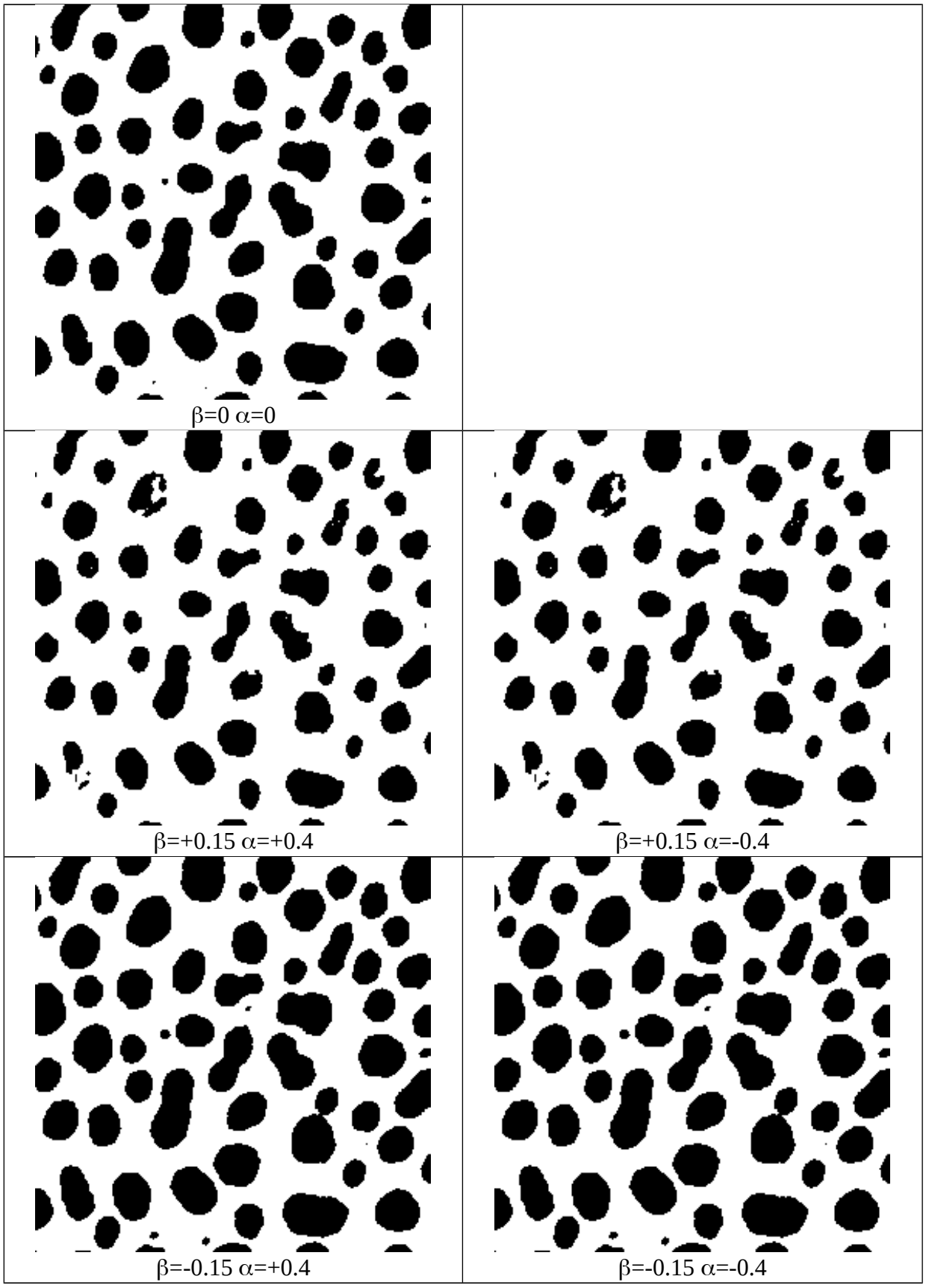
Ajuste convertendo para CieLab

Limiarização (thresholding)

A limiarização converte uma imagem em níveis de cinza ou colorida numa imagem binária, baseado somente no valor do pixel.

O método de limiarização mais simples substitui cada pixel na imagem por um pixel preto se $f(i,j)$ for menor que um limiar T, ou por um pixel branco caso contrário.

```
//threshg.cpp
#include <cekeikon.h>

int main(int argc, char** argv)
{ if (argc!=4) {
    printf("ThreshG ent.tga sai.bmp limiar\n");
    printf("  if (ent(l,c)<=limiar) sai(l,c)=0;\n");
    erro("Erro: Numero invalido de argumentos");
  }

  int limiar=0;
  if (sscanf(argv[3],"%d",&limiar)!=1) erro("Erro: limiar deve ser inteiro");
  if (limiar<0 || 255<limiar) erro("Erro: limiar deve pertencer a [0,255]");

  Mat_<GRY> ent; le(ent,argv[1]);
  Mat_<GRY> sai(ent.size());

  for (int l=0; l<ent.rows; l++)
    for (int c=0; c<ent.cols; c++) {
      if (ent(l,c)<=limiar) sai(l,c)=0;
      else sai(l,c)=255;
    }
  imp(sai,argv[2]);
}
```



Limiarização em 127 após ajuste de brilho e contraste. A saída depende do brilho.

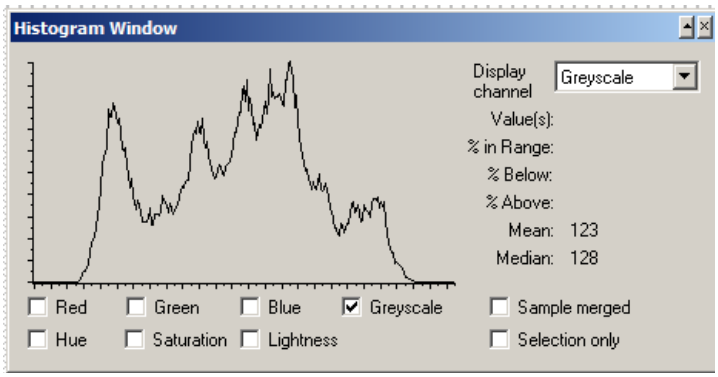
Histograma

Histograma representa distribuição de frequências por meio de retângulos, cujas larguras representam intervalos de classe e cujas áreas são proporcionais às frequências (absolutas ou relativas). A altura de cada retângulo é a frequência dividida pelo tamanho do intervalo.

Calcular histograma de uma imagem em níveis de cinza em 8 “bins”:

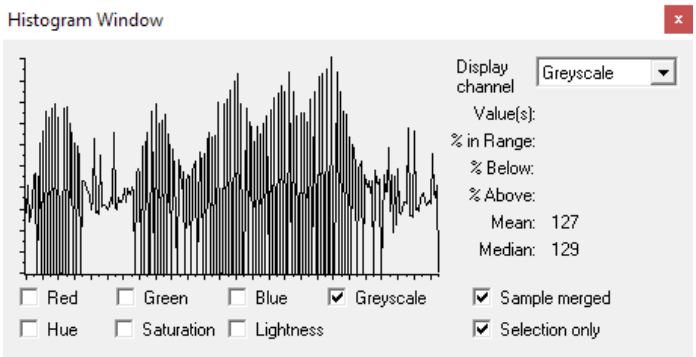
```
//histog.cpp - pos2012
#include <cekeikon.h>
int main()
{ Mat_<GRY> a; le(a,"lennag.tga");
  vector<int> histog(8,0);
  // vetor de 8 posicoes preenchida com zero
  for (int l=0; l<a.rows; l++)
    for (int c=0; c<a.cols; c++) {
      int i=a(l,c)/32; // i pertence [0,8[
      histog[i]++;
    }
  cout << histog << endl;
}

saída:
[415, 42487, 31329, 54730, 73740, 35218, 23815, 410]
```

lennag



histograma da imagem original

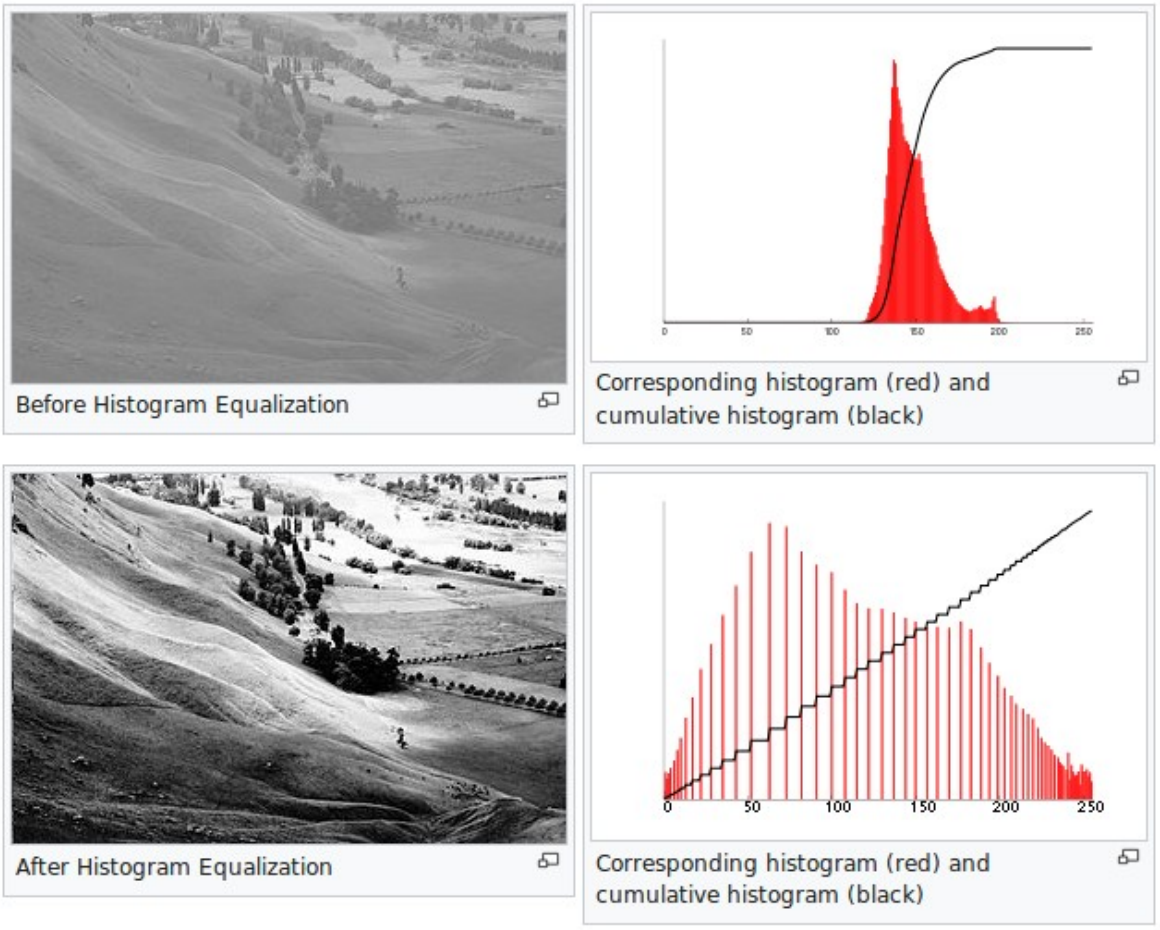
histograma equalizada



histograma equalizada

Veja:
https://en.wikipedia.org/wiki/Histogram_equalization
http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html

Equalização de histograma procura deixar histograma o mais uniforme possível ou histograma acumulado o mais "uniformemente crescente" possível. A imagem fica mais fácil de se visualizar. Não é possível deixar histograma completamente uniforme, devido ao número discreto de níveis de cinza e do número finito de pixels.



[Retirado de Wikipedia]

```
//equaliza.cpp - grad2018
//codigo adaptado de
//stackoverflow.com/questions/34126272/histogram-equalization-without-opencv
#include <cekeikon.h>

Mat_<GRY> equaliza(Mat_<GRY> a) {
  Mat_<GRY> b(a.size());  //b(a.rows,a.cols)
  // Compute histogram
  vector<int> hist(256, 0);
  for (unsigned i=0; i<a.total(); i++) hist[a(i)]++;

  // Find first non-zero bin
  int i=0;
  while (hist[i]==0) i++;
  if (hist[i]==a.total()) { // Imagem com uma unica cor
    b=a.clone(); return b;
  }

  //scale = 255.0 / Soma de todos bins menos first non-zero bin
  double scale = 255.0/(a.total()-hist[i]);

  vector<int> lut(256, 0); // cria LUT e preenche de zeros
  i++; // Primeiro bin depois do first non-zero bin. Antes, vira preto

  int sum=0; // soma excluindo first non-zero bin
  for (; i<256; i++) {
    sum+=hist[i];
    lut[i] = saturate_cast<GRY>(cvRound(sum*scale));
  }

  // Apply equalization
  for (unsigned i=0; i<a.total(); i++) b(i)=lut[a(i)];
  return b;
}

int main() {
  Mat_<GRY> a; le(a,"lennag.png");
  Mat_<GRY> b=equaliza(a);
  imp(b,"equaliza.png");
}
```

Intuição:

Suponha que o histograma seja:

nível de cinza	2	3	4
número de pixels	100	100	100

Então, o look-up-table deve ser:

níveis de cinza	0-2	3	4-255
saída	0	127	255

	
lennag.png	equaliza.png

OpenCV possui a função equalizeHist.

```
//histeqg.cpp
#include <cekeikon.h>
int main( int argc, const char** argv ) {
  Mat_<GRY> img; le(img,"lennag.png");
  Mat_<GRY> histeq;
  equalizeHist(img, histeq);
  imp(histeq,"histeqg.png");
}
```

Em Python:

```
# histeqg.py
import cv2
img = cv2.imread('lennag.png',0)
histeq=cv2.equalizeHist(img)
cv2.imwrite('histeqg_py.png',histeq)
```

Para fazer equalização de histograma de uma imagem colorida, pode-se fazer:
1) Equalizar cada uma das bandas RGB.
2) Converter imagem RGB para um outro sistema de cores onde a intensidade e coloração estão separados (como CieLAB, YCbCr, etc), equalizar a banda da intensidade (sem mexer na coloração) e depois reconverter para sistema RGB.

"Histogram equalization can also be used on color images by applying the same method separately to the Red, Green and Blue components of the RGB color values of the image. However, applying the same method on the Red, Green, and Blue components of an RGB image may yield dramatic changes in the image's color balance since the relative distributions of the color channels change as a result of applying the algorithm. However, if the image is first converted to another color space, Lab color space, or HSL/HSV color space in particular, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image [retirado de Wikipedia]".

Imagem original (img3 do Leuven de Mikolajczyk)

Equalização nas bandas RGB

Equalização de L do CieLAB.

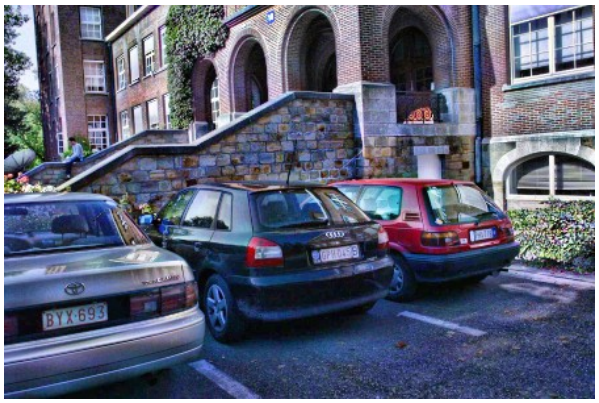
```cpp
//histrgb.cpp - Equaliza histograma de cada plano RGB
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=3) {
    printf("HistRGB: Equaliza histograma de cada plano RGB\n");
    printf("HistRGB ent.ppm sai.ppm\n");
    erro("Erro: Numero de argumentos invalido");
  }
  Mat_<COR> ent; le(ent,argv[1]);

  vector< Mat_<GRY> > bgr_planes(3);
  split(ent, bgr_planes);  // agora temos imagem R em bgr_planes[0], etc

  equalizeHist(bgr_planes[0], bgr_planes[0]);
  equalizeHist(bgr_planes[1], bgr_planes[1]);
  equalizeHist(bgr_planes[2], bgr_planes[2]);
  Mat_<COR> sai;
  merge(bgr_planes,sai);

  imp(sai,argv[2]);
}


//histeqc.cpp - equaliza histograma convertendo para CieLab
#include <cekeikon.h>
int main(int argc, char** argv) {
  if (argc!=3) {
    printf("HistEqC: Equaliza histograma no sistema CieLab\n");
    printf("HistEqC ent.ppm sai.ppm\n");
    erro("Erro: Numero de argumentos invalido");
  }
  Mat_<COR> ent; le(ent,argv[1]);
  Mat_<COR> lab; cvtColor(ent,lab,CV_BGR2Lab);
  vector< Mat_<GRY> > lab_planes(3);
  split(lab, lab_planes);  // now we have the L image in lab_planes[0]

  equalizeHist(lab_planes[0], lab_planes[0]);
  merge(lab_planes,lab);

  Mat_<COR> sai;
  cvtColor(lab,sai,CV_Lab2BGR);
  imp(sai,argv[2]);
}
```

Equalização de histograma adaptativa:

Equalização de histograma adaptativa (AHE) consiste em aplicar a equalização de histograma em cada janela móvel da imagem. Equalização de histograma adaptativa de contraste limitado (CLAHE) consiste em limitar a altura do histograma antes de fazer a equalização. Veja:

https://en.wikipedia.org/wiki/Adaptive_histogram_equalization

A função otimizada CLAHE está disponível no OpenCV3 (não disponível no OpenCV2).

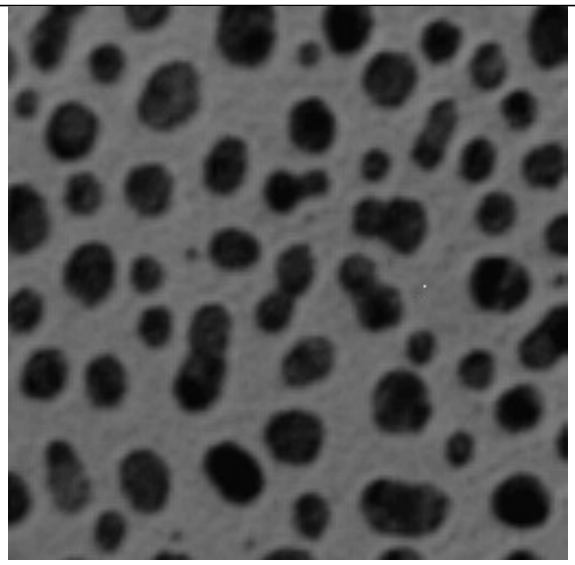
<table>
<tr><td> 
Imagem original níveis de cinza</td><td> 
Saída CLAHE clip=5 tile=16</td></tr>
<tr><td> 
Imagem original colorida</td><td> 
Saída aplicando CLAHE nas bandas RGB (c=5 t=16)</td></tr>
<tr><td> 
Saída aplicando CLAHE na banda L do CieLab</td><td></td></tr>
</table>

```
#!/bin/bash
set -v
kcek mconvert g pgm img3.ppm
claheg img3.pgm cla3.pgm -c=5 -t=16
clahec img3.ppm rgb3.ppm -c=5 -t=16 -r=true
clahec img3.ppm lab3.ppm -c=5 -t=16 -r=false
set +v
```

```
// CLAHEG = CLAHE para imagens em niveis de cinzas
// Linke com OpenCV3
// compila claheg -c -v3
#include <cekeikon.h>

int main(int argc, char** argv) {
  const char* keys =
    " curto| longo    |default| explicacao\n"
    " -c   | -clip    | 40    | clip histogram limit\n"
    " -t   | -tile    |  8    | use 8x8 tile grid size\n";

  if (argc<3) {
    printf("claheg: Contrast Limited Adaptive Histogram Equalization (GRY)\n");
    printf("claheg ent.pgm sai.pgm [opcoes]\n");
    printf("%s",keys);
    erro("Erro: Numero de argumentos invalido");
  }

  ArgComando cmd(argc,argv);
  string nomeent=cmd.getCommand(0);
  string nomesai=cmd.getCommand(1);
  double clip=cmd.getDouble("-c","-clip",40.0);
  int tile=cmd.getInt("-t","-tile",8);
  cmd.leuTodos();

  Mat_<GRY> ent; le(ent,nomeent);

  Ptr<CLAHE> clahe = createCLAHE();
  clahe->setClipLimit(clip);
  clahe->setTilesGridSize(Size(tile,tile));

  Mat_<GRY> sai;
  clahe->apply(ent,sai);
  imp(sai,nomesai);
}
```

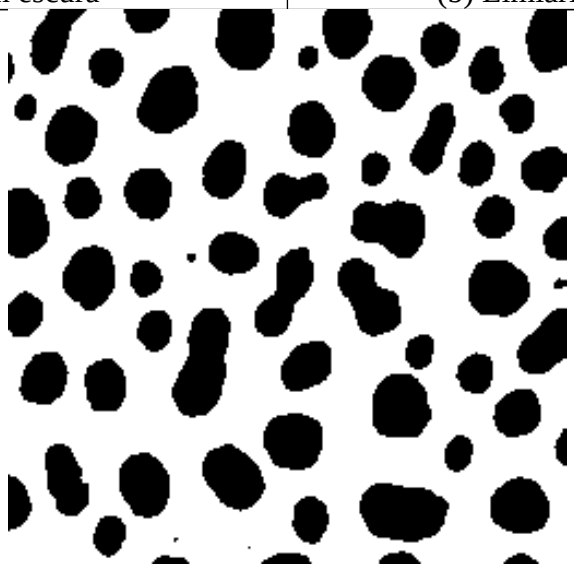
```
// CLAHEC = CLAHE para imagens coloridas
// Linke com OpenCV3
// compila clahec -c -v3
#include <cekeikon.h>

int main(int argc, char** argv) {
  if (argc<3) {
    printf("clahec: Contrast Limited Adaptive Histogram Equalization (COR)\n");
    printf("clahec ent.ppm sai.ppm [opcoes]\n");
    printf("%s",
      " curto| longo    |default| explicacao\n"
      " -c   | -clip    | 40    | Clip histogram limit\n"
      " -t   | -tile    |  8    | Use 8x8 tile grid size\n"
      " -r   | -rgb     | false | Equaliza cada banda RGB\n"
      "   Se nao especificar -r, equaliza L de CieLab\n");
    erro("Erro: Numero de argumentos invalido");
  }
  ArgComando cmd(argc,argv);
  string nomeent=cmd.getCommand(0);
  string nomesai=cmd.getCommand(1);
  double clip=cmd.getDouble("-c","-clip",40.0);
  int tile=cmd.getInt("-t","-tile",8);
  bool rgb=cmd.getBool("-r","-rgb",false);
  cmd.leuTodos();

  Mat_<COR> ent; le(ent,nomeent);
  Mat_<COR> sai;

  if (rgb==false) { // Processa em CieLab
    Mat_<COR> lab; cvtColor(ent,lab,CV_BGR2Lab);
    vector< Mat_<GRY> > lab_planes(3);
    split(lab, lab_planes);  // now we have the L image in lab_planes[0]

    Ptr<CLAHE> clahe = createCLAHE();
    clahe->setClipLimit(clip);
    clahe->setTilesGridSize(Size(tile,tile));

    clahe->apply(lab_planes[0],lab_planes[0]);
    merge(lab_planes,lab);

    cvtColor(lab,sai,CV_Lab2BGR);
  } else { // Processa cada banda
    vector< Mat_<GRY> > bandas(3);
    split(ent, bandas);  // Aqui, temos BGR em bandas[0], bandas[1] e bandas[2]

    Ptr<CLAHE> clahe = createCLAHE();
    clahe->setClipLimit(clip);
    clahe->setTilesGridSize(Size(tile,tile));

    clahe->apply(bandas[0],bandas[0]);
    clahe->apply(bandas[1],bandas[1]);
    clahe->apply(bandas[2],bandas[2]);

    merge(bandas,sai);
  }
  imp(sai,nomesai);
}
```

Se a imagem for muito escura ou muito clara (como na subfigura-a abaixo), fazer limiarização em 127 pode não dar bom resultado (subfigura-b). O método de Otsu calcula automaticamente um limiar adequado (subfigura-c).



(a) Imagem escura

(b) Limiarização em 127



(c) Limiarização usando método de Otsu. Escolheu automaticamente o limiar 68.

Método de Otsu está descrito (por exemplo) em:

https://en.wikipedia.org/wiki/Otsu%27s_method

O método de Otsu escolhe limiar que minimiza a soma ponderada da variância intra-classe.

In Otsu's method we exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t and σ_0^2 and σ_1^2 are variances of these two classes.

The class probability $\omega_{0,1}(t)$ is computed from the L histograms:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance:[2]

$$\begin{aligned}\sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2\end{aligned}$$

which is expressed in terms of class probabilities ω and class means μ.

while the class mean $\mu_{0,1,T}(t)$ is:

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0}$$

$$\mu_1(t) = \sum_{i=t}^{L-1} i\frac{p(i)}{\omega_1}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

Está diferente do artigo original do Otsu

The following relations can be easily verified:

$$\begin{aligned}\omega_0\mu_0 + \omega_1\mu_1 &= \mu_T \\ \omega_0 + \omega_1 &= 1\end{aligned}$$

The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm.

Algorithm [edit]

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1, \ldots$ maximum intensity
    1. Update ω_i and μ_i
    2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$

Artigo original de Otsu maximiza:

$$\sigma_b^2(t)=\frac{(\mu_t\omega_0(t)-\mu_0(t))^2}{\omega_0(t)[1-\omega_0(t)]}$$

onde $\mu_0(t)$ é a média da classe preta vezes $\omega_0(t)$.

Exemplo:

Vamos supor que uma imagem possui o seguinte histograma:

nível de cinza	1	2	3	4
frequencia	0.25	0.25	0.25	0.25

Evidentemente, neste caso, a melhor binarização é tornar {1,2} preto e {3,4} branco.

Se fizer limiarização em 2,5 (deixando {1,2} pretos e {3,4} brancos), temos o seguinte "weighted sum of variances":

$$\sigma_w^2(t)=w_0(t)\sigma_0^2(t)+w_1(t)\sigma_1^2(t) \quad \Rightarrow \quad \sigma_w^2(2.5)=0.5\times 0.25+0.5\times 0.25=0.25$$

Se fizer limiarização em 1,5 (deixando {1} preto e {2,3,4} brancos), temos o seguinte "weighted sum of variances":

$$\sigma_w^2(1.5)=0.25\times 0.0+0.75\times 0.67=0.50$$

Portanto, a limiarização em 2,5 resulta numa menor variância intra-classe $\sigma_w^2(t)$.

Para acelerar o algoritmo, em vez de buscar a menor variância intra-classe, procura-se a maior variância inter-classe.

Se fizer limiarização em 2,5 (deixando {1,2} pretos e {3,4} brancos), temos a seguintevariância inter-classe:

$$\sigma_b^2(t)=\frac{(\mu_t\omega_0(t)-\mu_0(t))^2}{\omega_0(t)[1-\omega_0(t)]} \quad \Rightarrow \quad \sigma_b^2(0.5)=\frac{[0.75\times 0.5-0.75]^2}{0.5[1-0.5]}=1$$

Se fizer limiarização em 1,5 (deixando {1} preto e {2,3,4} brancos), temos o seguinte resultado:

$$\sigma_b^2(t)=\frac{(\mu_t\omega_0(t)-\mu_0(t))^2}{\omega_0(t)[1-\omega_0(t)]} \quad \Rightarrow \quad \sigma_b^2(0.5)=\frac{[0.25\times 0.25-0.25]^2}{0.25[1-0.25]}=0.75$$

Implementação do método de Otsu em C++:

```
//otsug.cpp
#include <cekeikon.h>

void ConstroiHist(const Mat_<GRY>& ent, vector<double>& hist) {
  for (int i=0; i<256; i++) hist[i]=0.0;
  for (int i=0; i<ent.total(); i++) hist[ent(i)]++;
  for (int i=0; i<256; i++) hist[i]=hist[i]/ent.total();
}

int main(int argc, char** argv) {
  if (argc!=3) {
    printf("OtsuG: Binariza imagem usando limiar de Otsu global\n");
    printf("OtsuG ent.pgm sai.pgm\n");
    erro("Erro: Numero de argumentos invalido");
  }

  Mat_<GRY> ent; le(ent,argv[1]);
  Mat_<GRY> sai(ent.size());

  vector<double> hist(256);
  ConstroiHist(ent,hist);

  double ut=0.0; // media total
  for (int i=0; i<256; i++) ut=ut+i*hist[i];

  double w0=0.0; // probabilidade de preto
  double u0=0.0; // media dos pixels pretos
  double maxvar=0.0;
  int maxind=0;
  for (int i=0; i<255; i++) {
    // Nota Se final for 256, nao funciona - Note que ha somente 255 possiveis cortes
    w0=w0+hist[i];
    u0=u0+i*hist[i];
    if (w0==0.0 || w0==1.0) continue;
    double varian=elev2(ut*w0-u0)/(w0*(1-w0));
    printf("i=%d w=%g m=%g varian=%g\n",i,w0,u0,varian);
    if (varian>maxvar) { maxvar=varian; maxind=i; }
  }
  printf("ind_max=%d (p<=ind_max -> preto), var_max=%lg\n",maxind,maxvar);
  for (int i=0; i<ent.total(); i++)
    if (ent(i)<=maxind) sai(i)=0;
    else sai(i)=255;
  imp(sai,argv[2]);
}
```

Em OpenCV, há a função:

```
double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)
```

- src– input array (single-channel, 8-bit or 32-bit floating point).
- dst– output array of the same size and type as src.
- thresh– threshold value.
- maxval – maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types.
- type– thresholding type (see the details below).

Tipos (type) principais:

- THRESH_BINARY
- THRESH_BINARY | THRESH_OTSU

Currently, the Otsu's method is implemented only for 8-bit images.

```
//otsu-ocv.cpp
#include <cekeikon.h>
int main(int argc, char** argv) {
   if (argc!=3) {
      printf("Otsu-ocv: Binariza imagem usando limiar de Otsu global\n");
      printf("Otsu-ocv ent.pgm sai.pgm\n");
      erro("Erro: Numero de argumentos invalido");
   }
   Mat_<GRY> ent; le(ent,argv[1]);
   Mat_<GRY> sai(ent.size());
   double t=threshold(ent, sai, 0, 255, THRESH_BINARY | THRESH_OTSU);
   cout << t << endl;
   imp(sai,argv[2]);
}
```