Formatos de imagens que Cekeikon lê/imprime:

Formatos a serem usados no curso:

Para simplificar, utilizaremos preferencialmente os 4 tipos de imagens abaixo no curso.

1) Formatos sem compactação:
a) PPM para imagens coloridas (binário ou ASCII).
b) PGM para imagens em níveis de cinzas (binário ou ASCII).
c) PBM para imagens binárias (binário ou ASCII). Não interessa, pois OpenCV não consegue trabalhar com imagens binárias.

```
#include <cekeikon.h>
int main()
{ Mat_<COR> a; le(a,"lenna.jpg");
  imp(a,"lenna_bin.ppm");  // grava no formato PPM binario
  imp(a,"lenna_asc.ppm0"); // grava no formato PPM ASCII
}
```

```
#include <cekeikon.h>
int main()
{ Mat_<GRY> a; le(a,"lenna.jpg");
  imp(a,"lenna_bin.pgm");  // grava no formato PGM binario
  imp(a,"lenna_asc.pgm0"); // grava no formato PPM ASCII
}
```

Exemplo de imagem ppm ASCII com 2x2 pixels:

```
P3
# Created by Paint Shop Pro 7
2 2
255
255 0 0 0 0 255 0 255 0 255 0 255
```

A mesma imagem no formato binário:

```
[0000] 50 36 0A 23 20 43 72 65 [] 61 74 65 64 20 62 79 20 [P6.# Cre][ated by ]
[0010] 50 61 69 6E 74 20 53 68 [] 6F 70 20 50 72 6F 20 37 [Paint Sh][op Pro 7]
[0020] 0A 32 20 32 0A 32 35 35 [] 0A FF 00 00 00 00 FF 00 [.2 2.255][. .... .]
[0030] FF 00 FF 00 FF [] [] [] [] [] [] [] [] [] [] [] [] [ . . ...][........]
```

```cpp
//gray.cpp grad2015
#include <cekeikon.h>
int main() {
  Mat_<GRY> a = ( Mat_<GRY>(2,2) << 0, 64, 128, 255 );
  imp(a,"gray_bin.pgm");
  imp(a,"gray_asc.pgm0");
}
```

```cpp
//cor.cpp grad2015
#include <cekeikon.h>
int main() {
  Mat_<COR> a = ( Mat_<COR>(2,2) << COR(0,0,255), COR(0,255,0),
                                    COR(0,0,255), COR(255,255,255) );
  imp(a,"cor_bin.ppm");
  imp(a,"cor_asc.ppm0");
}
```

2) Formato com compactação e sem perdas:
PNG: Bom para gravar imagens “artificiais”. Compacta sem perdas. As imagens artificiais (como a tela do computador capturada) podem ser compactadas da ordem de 50 vezes. Há formato para imagens em níveis de cinza e coloridas. Não consegue compactar substancialmente imagens naturais.

```
#include <cekeikon.h>
int main() {
  Mat_<COR> a; le(a,"tela.ppm");
  imp(a,"tela0.png0");
  imp(a,"tela3.png");
  imp(a,"tela5.png5");
  imp(a,"tela9.png9");
}
```

```
Imagem capturada da tela do computador:
31/03/2016  09:58         2.884.846 tela.ppm
31/03/2016  10:06           266.395 tela0.png
31/03/2016  10:06           268.191 tela3.png
31/03/2016  10:06           177.312 tela5.png
31/03/2016  10:06           177.312 tela9.png
```

O número após .png indica o nível de compressão: quanto maior, melhor a compressão mas demora mais. O padrão é nível 3. Só funciona em Cekeikon.

3) Formato com compactação e com perdas:
JPG: Bom para imagens naturais. Compacta com perdas, isto é, a imagem gravada será diferente da imagem sem compactação. Há formatos para imagens em níveis de cinza e coloridas.

```
#include <cekeikon.h>
int main(int argc, char** argv)
{ if (argc!=3) {
    printf("jpeg ent.ppm sai.jpg\n");
    printf("  Ex: sai.jpgnn sai.jpg99 sai.jpg70...\n");
    printf("  nn de 0 a 100. default = 95\n");
    printf("  nn maior tem maior qualidade\n");
    erro("Erro: NUmero de argumentos invalido");
  }
  Mat_<COR> a; le(a,argv[1]);
  imp(a,argv[2]);
}
```

```
  5.730 lenna01.jpg  - compactação com perdas quali=01
  9.566 lenna10.jpg
 17.651 lenna30.jpg
 24.329 lenna50.jpg
 69.214 lenna90.jpg  - compactou umas 4 vezes
203.865 lenna99.jpg  - compactação com perdas quali=99
475.639 lenna9.png   - compactação sem perdas.
                       ficou maior do que sem compactação.
262.189 lennag.pgm   - imagem não compactada
```

Os dois números nn após .jpg indicam a qualidade da imagem compactada. Só funciona em Cekeikon.

Para imagem em níveis de cinza e qualidade 90, a compactação diminuiu o tamanho da ordem de 4 vezes.

```cpp
//jpg.cpp pos-2016
#include <cekeikon.h>
int main() {
  Mat_<COR> a; le(a,"mandrill.tga");
  imp(a,"mandrill01.jpg01");
  imp(a,"mandrill20.jpg20");
  imp(a,"mandrill50.jpg50");
  imp(a,"mandrill80.jpg80");
  imp(a,"mandrill90.jpg90");
  imp(a,"mandrill95.jpg95");
  imp(a,"mandrill99.jpg99");
  imp(a,"mandrill9.png9");
  imp(a,"mandrill.ppm");
  imp(a,"mandrill.txt");
  imp(a,"mandrill.jp2");
}
```

```
    592.594 mandrill.jp2
    786.447 mandrill.ppm     - sem compressão.
    786.450 mandrill.tga
  3.933.193 mandrill.txt
      6.462 mandrill01.jpg
     27.409 mandrill20.jpg
     50.757 mandrill50.jpg
     89.057 mandrill80.jpg
    627.837 mandrill9.png    - compactação sem perdas quase não compactou nada.
    134.004 mandrill90.jpg   - compactou aproximadamente 6 vezes
    191.212 mandrill95.jpg
    310.225 mandrill99.jpg
```

OpenCV lê/imprime os seguintes formatos:

- Windows bitmaps - *.bmp, *.dib
- JPEG files - *.jpeg, *.jpg, *.jpe
- JPEG 2000 files - *.jp2
- Portable Network Graphics - *.png
- Portable image format - *.pbm, *.pgm, *.ppm
- Sun rasters - *.sr, *.ras
- TIFF files - *.tiff, *.tif
- Outras imagens gravadas como .xml

Além disso, Cekeikon lê/imprime os seguintes formatos:

- *.tga não-compactada para Mat_<GRY> e Mat_<COR>.
- *.txt para Mat_<GRY>, Mat_<COR>, Mat_<SHT>, Mat_<FLT>, Mat_<DBL> e Mat_<CPX>
- *.img para Mat_<SHT>, Mat_<FLT>, Mat_<DBL> e Mat_<CPX>

1. Imagens grayscale (Mat_<GRY>):

Mat_<GRY> g;

- TGA não-compactado. imp(g,"nomearq.tga");
- PNG compactado sem perdas. imp(g,"nomearq.png");
  - PNG é bom para gravar imagens como tela de computador e "gibi".
- TIF compactado sem perdas. imp(g,"nomearq.tif");
- PGM binário imp(g,"nomearq.pgm");
  ascii imp(g,"nomearq.pgm0"); // 0 indica ascii
  - PGM é bom para gravar imagens sem compactação, tanto binário como texto, pois o formato é bem simples.
- TXT texto imp(g,"nomearq.txt");
- JPG compactado com perdas. imp(g,"nomearq.jpg95"); // qualidade 95
  - JPG é bom para gravar imagens de fotos "naturais".
- JP2 compactado com perdas. imp(g,"nomearq.jp2");

2. Imagens coloridas (Mat_<COR>):

Mat_<COR> a;

- TGA não-compactado. imp(a,"nomearq.tga");
- PNG compactado sem perdas. imp(a,"nomearq.png");
- TIF compactado sem perdas. imp(a,"nomearq.tif");
- PPM binário imp(a,"nomearq.ppm");
  ascii imp(a,"nomearq.ppm0"); // 0 indica ascii
- TXT texto imp(a,"nomearq.txt");
- JPG compactado com perdas. imp(a,"nomearq.jpg95"); // qualidade 95
- JP2 compactado com perdas. imp(a,"nomearq.jp2");

3. Imagens onde cada pixel é float, complexo, short, etc (Mat_<FLT>, Mat_<DBL>, Mat_<CPX>, Mat_<SHT>, etc).

Nota: FLT=float, DBL=double, CPX=complex<float>, SHT=short int

A extensão é .img

Mat_<FLT> a; imp(a,”nomearq.img”);

Também podem ser impressas como .txt

Só funciona em Cekeikon.

OpenCV permite gravar como .XML

As imagens digitais podem ter diferentes características, por exemplo:

- Podem ser em níveis de cinza ou coloridas. No último caso, podem usar diferentes sistemas de cores (RGB, HSI, CMYK, YCbCr, etc).
- Resolução da coloração pode ser menor que a resolução da luminância (ex: vídeo).
- Número de bits por pixel (Ex: 1, 4, 8, 16, 24, 32). Quando o número de bits/pixel for baixo, é comum usar palete.
- Compactado com perdas (JPG), compactado sem perdas (PNG) ou não-compactado (PBM, PGM, PPM).

Por exemplo, numa impressora jato de tinta preto-e-branco, em cada pixel só é possível escolher entre jogar (ou não jogar) tinta preta. É necessário representar imagens em níveis de cinza apenas com essas escolhas.

A impressora laser preto-e-branco também é binária, com a restrição adicional de que só consegue imprimir amontoado de pixels (não consegue imprimir pixels pretos e brancos finamente intercalados).

Alguns formatos de imagens

Os formatos de imagens BMP, TIF e TGA possuem muitos subformatos. Podem representar imagens:

- Coloridas, em níveis de cinza ou binárias.
- Não-compactado ou compactado sem perdas.
- Diferentes números de bits por pixel.
- Com ou sem palette.

Eles são utilizados quando deseja armazenar imagens sem perda e não há necessidade de grande compressão.

O formato GIF pode ter no máximo 256 cores diferentes. É um formato antigo. É bom para armazenar imagens tipo “quadrinhos”, gráficos, etc. Pode armazenar vídeos simples. Pode ter cor de transparência.

O formato PNG compacta as imagens sem perdas (colorida, níveis de cinza e binária). É bom para armazenar imagens artificiais tipo “quadrinhos”, gráficos, cópia da tela de computador, etc. Pode ter cor de transparência.

Os formatos de imagens JPG ou JPEG (antigo e 2000) armazenam imagens com perdas, usando respectivamente as transformadas DCT (discrete cosine transform) e wavelet. O tamanho da imagem torna-se extremamente pequena, com a introdução de pequenos erros. São bons para armazenar imagens “naturais” mas são ruins para armazenar imagems artificiais (gráficos, halftones, quadrinhos, cópias da tela do computador, etc.), pois perdem os componentes de alta freqüência.

O formato JBIG (1 e 2) foram projetados especialmente para armazenar imagens binárias, com ou sem perdas.

Os formatos PBM, PGM e PPM são usados respectivamente para armazenar imagens binárias, em níveis de cinzas e coloridas. Eles são muito simples e por isso é fácil escrever rotinas de leitura/escrita para eles. Cada um deles há versão em arquivo texto e arquivo binário.

A imagem 1a é “true color”, isto é, todas as cores que podem ser mostras na tela podem ser representadas nela (a tela não consegue mostrar todas as cores visíveis a um ser humano). O conjunto de todas as cores possíveis de serem representadas está na figura 1b. Cada pixel é representado com 24 bits, 8 bits para cada cor primária, e cada pixel pode assumir uma entre $2^{24} \cong 16$ milhões de cores diferentes possíveis. Cada uma das bandas R, G e B podem representar níveis de cinza que vai de 0 a 255.

Alguns formatos de imagens utilizam palete para diminuir a memória necessária para armazenar uma imagem. Por exemplo, o formato GIF (figura 2a) escolhe 256 cores mais representativos da imagem, armazena essas cores numa tabela chamada palete (figura 2b), e cada pixel da imagem passa a conter os índices da palete (em vez de “true color”).

A escolha boa da palete é importante para que a imagem seja representada com acuidade. Figura 3a mostra a imagem lenna.tga representada com 16 cores, onde a palete uniforme (figura 3b), padrão do Windows antigo, foi utilizada. A qualidade da imagem é sofrível.

A figura 4a também está representada utilizando apenas 16 cores. Mas a qualidade visual é bem melhor do que a figura 3a por utilizar palete escolhido pelo algoritmo “median cut”. Este algoritmo é um exemplo de aprendizagem de máquina não-supervisionada. A difusão de erro intercala cores diferentes para representar cores inexistentes na palete.

A figura 5a também está representada utilizando as mesmas 16 cores da figura 4a. Mas a qualidade visual é melhor do que 4a por utilizar a técnica de difusão de erro.

Fig. 1a: lenna.tga, 24 bits/pixel, 16 milhões de cores

Fig. 1b: Todas as cores

Fig. 2a: lenna.gif (palete com 256 cores,8 bits/pixel)

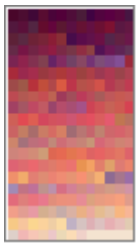


Fig. 2b: Palete com 256 cores

Fig. 3a: l-win-nn.tga
(palete com 16 cores, 4 bits/pixel, palete padrão do windows, vizinho mais próximo).

Fig. 3b: Palete uniforme com 16 cores, padrão do Windows

Fig. 4a: l-med-nn.tga
(palete com 16 cores, 4 bits/pixel, palete escolhido pelo median cut, vizinho mais próximo)

Fig. 4b: Palete escolhido pelo median cut

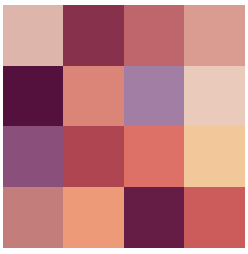
Fig. 5a: l-med-ed.tga
(palete com 16 cores, 4 bits/pixel, palete escolhido pelo median cut, difusão de erro)

Fig. 5b: Palete escolhido pelo median cut

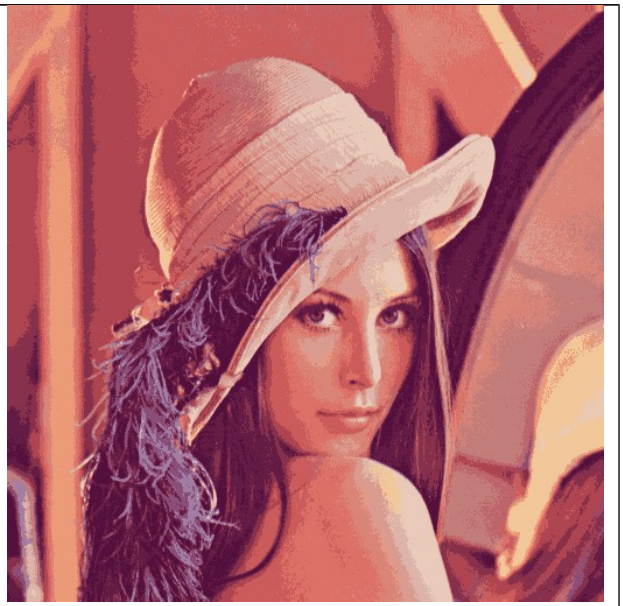
Nota: O algoritmo median cut pode escolher o palete rapidamente.
Nota: O algoritmo k-means (k-médias) poderia ser usada para escolher palete.

Escolher 16 cores do palette usando k-means:

```
//kmeans3.cpp - pos2014
#include <cekeikon.h>

int main()
{ Mat_<COR> a; le(a,"lenna.jpg");

  Mat_<FLT> temp(1,3);
  Mat_<FLT> m; // Entrada com todas as cores da imagem
  for (unsigned j=0; j<a.total(); j++) {
    COR cor=a(j);
    temp(0)=cor[0]; temp(1)=cor[1]; temp(2)=cor[2];
    m.push_back(temp);
  }

  int k=16;
  Mat_<int> bestLabels(m.rows,1);
  Mat_<FLT> centers(k,3);
  TermCriteria criteria(TermCriteria::COUNT | TermCriteria::EPS, 3, 2.0);
  kmeans(m,k,bestLabels,criteria,1,KMEANS_PP_CENTERS,centers);

  Mat_<COR> palete(4,4);
  for (unsigned i=0; i<16; i++)
    palete(i)=COR( cvRound(centers(i,0)),
                   cvRound(centers(i,1)),
                   cvRound(centers(i,2)) );
  imp(palete,"palette.ppm");

  Mat_<COR> b(a.size());
  for (unsigned i=0; i<b.total(); i++) {
    int k=bestLabels(i);
    assert(0<=k && k<16);
    COR cor;
    cor[0]=saturate_cast<GRY>(centers(k,0));
    cor[1]=saturate_cast<GRY>(centers(k,1));
    cor[2]=saturate_cast<GRY>(centers(k,2));
    b(i)=cor;
  }
  imp(b,"kmeans3.ppm");
}
```



palette.ppm

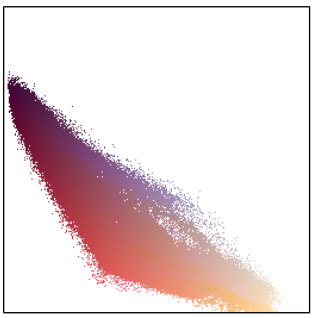


kmeans3.ppm

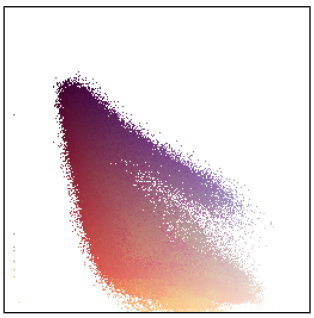
```
//kmeandiferr.cpp - pos2015
#include <cekeikon.h>

int diferenca(COR a, COR b) {
  return abs(a[0]-b[0])+abs(a[1]-b[1])+abs(a[2]-b[2]);
}

COR maisProximo(COR a, Mat_<COR> palete) {
  COR maispx; int mindif=maxint;
  for (unsigned i=0; i<palete.total(); i++) {
    int dif=diferenca(a,palete(i));
    if (dif<mindif) { maispx=palete(i); mindif=dif; }
  }
  return maispx;
}

int main() {
  Mat_<COR> a;
  le(a,"lenna.jpg");

  Mat_<FLT> temp(1,3);
  Mat_<FLT> m; // Entrada com todas as cores da imagem
  for (unsigned j=0; j<a.total(); j++) {
    COR cor=a(j);
    temp(0)=cor[0]; temp(1)=cor[1]; temp(2)=cor[2];
    m.push_back(temp);
  }

  int k=16;
  Mat_<int> bestLabels(m.rows,1);
  Mat_<FLT> centers(k,3);
  TermCriteria criteria(TermCriteria::COUNT | TermCriteria::EPS, 3, 2.0);
  kmeans(m,k,bestLabels,criteria,1,KMEANS_PP_CENTERS,centers);

  Mat_<COR> palete(k,1);
  for (unsigned i=0; i<k; i++)
    palete(i)=COR( round(centers(i,0)),
                   round(centers(i,1)),
                   round(centers(i,2)) );
  //imp(palete,"palette.ppm");

  Img<COR> d(a.size());
  for (int l=0; l<d.rows; l++)
    for (int c=0; c<d.cols; c++) {
      COR cor=maisProximo(a(l,c),palete);
      d(l,c)=cor;
      Vec3i err=a(l,c)-cor;
      a(l+1,c-1)   += err/4;
      a(l+1,c)     += err/4;
      a(l  ,c+1)   += err/4;
      a(l+1,c+1)   += err/4;
    }
  imp(d,"kmeandiferr.ppm");
}
```

kmeandiferr.ppm - não ficou muito boa

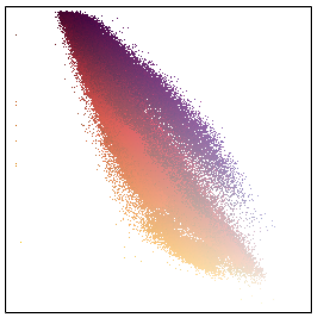
O programa IMG PROJCOR (ou kcek projcor) projeta as cores que aparecem na imagem nos planos RG, RB e GB. Rodando esse programa para lenna, obtemos:



RG



RB



GB

Note que o espaço das cores não está completamente cheia. Isto é, há muitas cores que não aparecem na imagem Lenna.

Bibliografia:

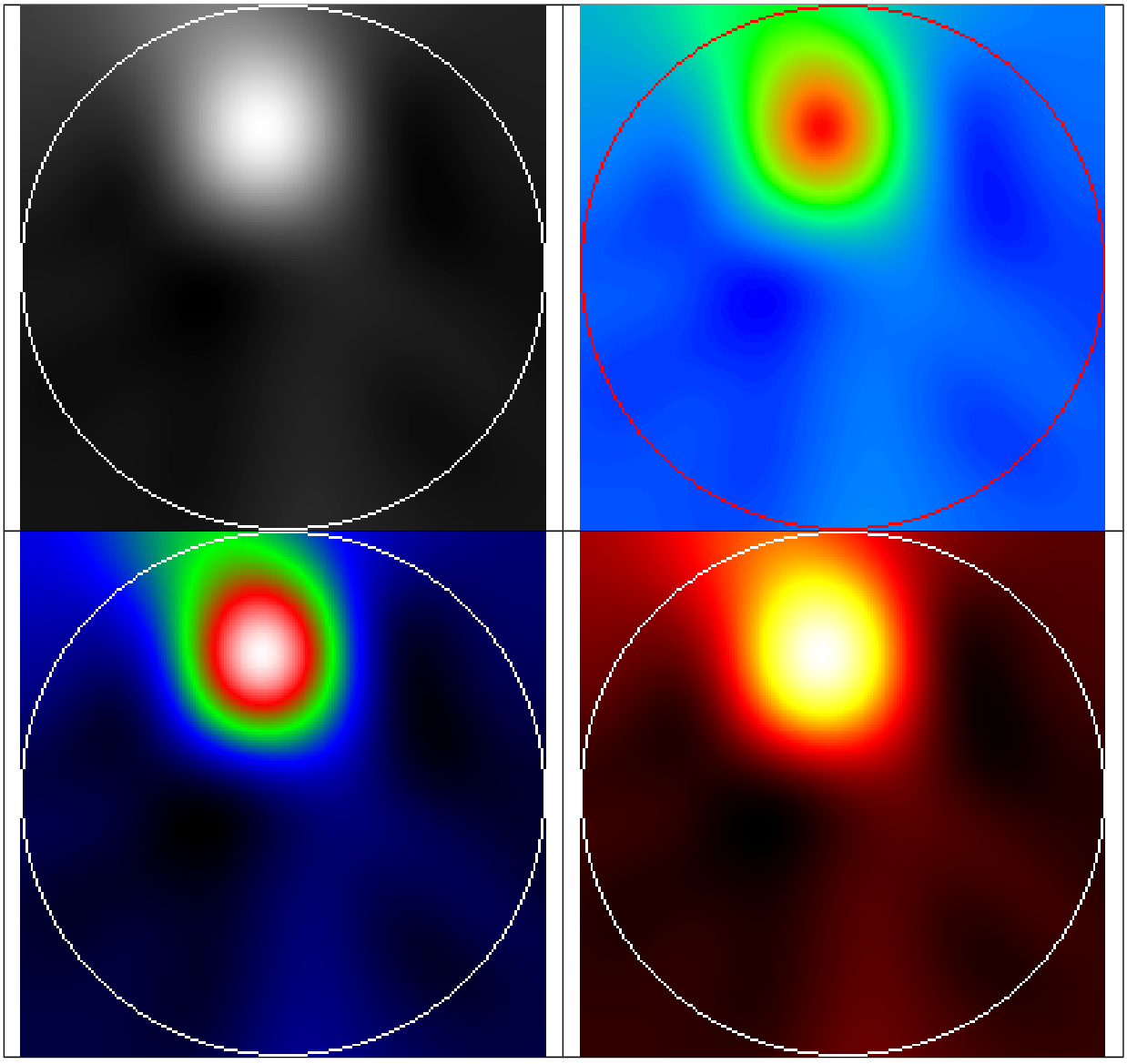
[Heckbert, 1982] P. Heckbert, “Color Image Quantization for Frame Buffer Display,” *Computer Graphics*, vol. 16, no. 3, pp. 297-307, 1982, available at
http://www.cs.cmu.edu/~ph/ciq.ps.gz
Descreve o algoritmo median-cut para escolha de palette, isto é, as cores representativas de uma imagem.

[Orchard and Bouman, 1991] M. T. Orchard and C. A. Bouman, “Color Quantization of Images,” *IEEE Transactions on Signal Processing*, vol. 39, no. 12, pp. 2677-2690, 1991, available at http://ee.www.ecn.purdue.edu/~bouman/publications/pdf/sp1.pdf

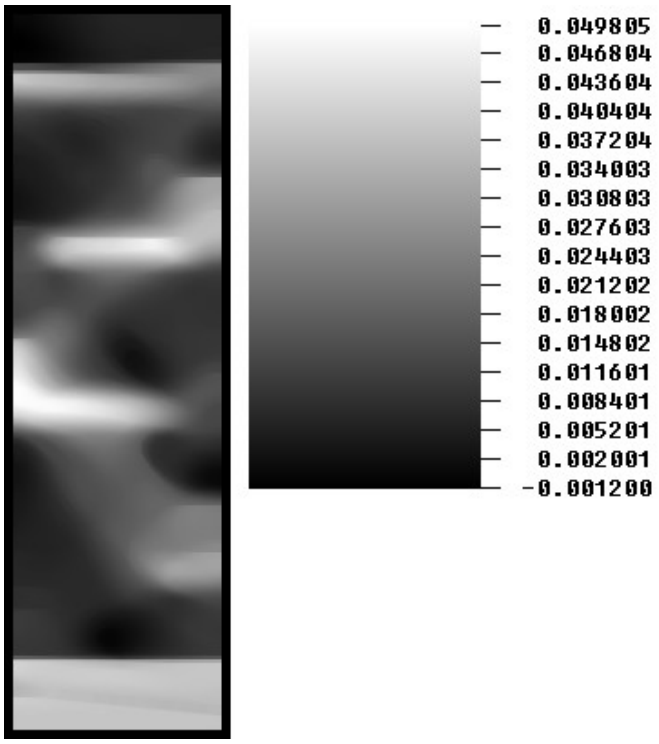
Palete para obter imagens com “falsa cor” ou “pseudo-cor”

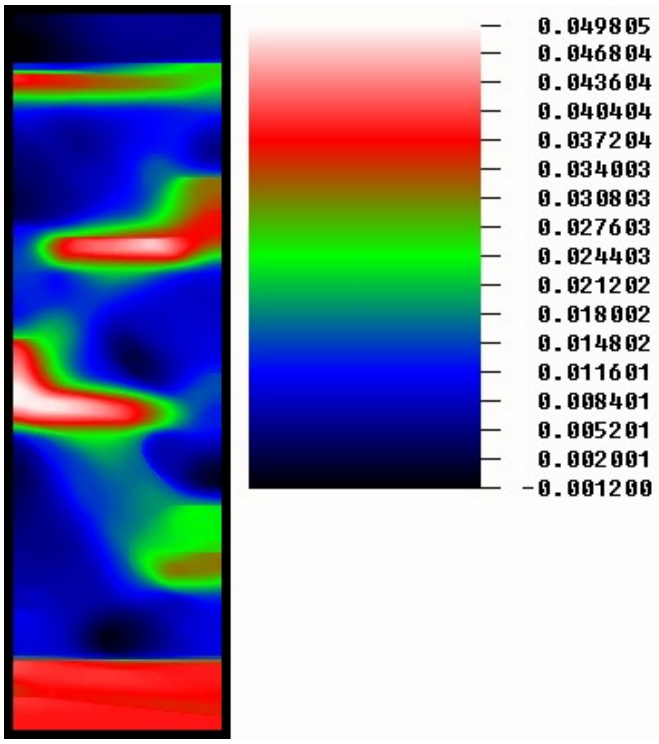
A “falsa cor” é muito usada em imagens médicas, imagens de sensoriamento remoto, imagens tomografia, etc para facilitar a visualização de uma imagem em níveis de cinza. [Gonzalez and Woods, 2002] mostra a vigilância de malas nos aeroportos com raio-x, onde os objetos são visualizados mais facilmente com pseudo-cor.

A figura abaixo mostra uma imagem de tomografia industrial com 4 diferentes pseudo-cores.



Uma imagem de tomografia industrial em níveis de cinza e a mesma imagem com pseudo-cores que facilitam a visualização.


0.049805
0.046804
0.043604
0.040404
0.037204
0.034003
0.030803
0.027603
0.024403
0.021202
0.018002
0.014802
0.011601
0.008401
0.005201
0.002001
-0.001200


0.049805
0.046804
0.043604
0.040404
0.037204
0.034003
0.030803
0.027603
0.024403
0.021202
0.018002
0.014802
0.011601
0.008401
0.005201
0.002001
-0.001200