DCT: Discrete Cosine Transform.

Escrever a definição de DCT.

DCT é amplamente usado na compressão de imagens e vídeos (JPEG, MPEG, etc). O olho humano é pouco sensível às altas freqüências. Assim, as altas freqüências podem ser armazenadas com menos bits (quantização mais grosseira).

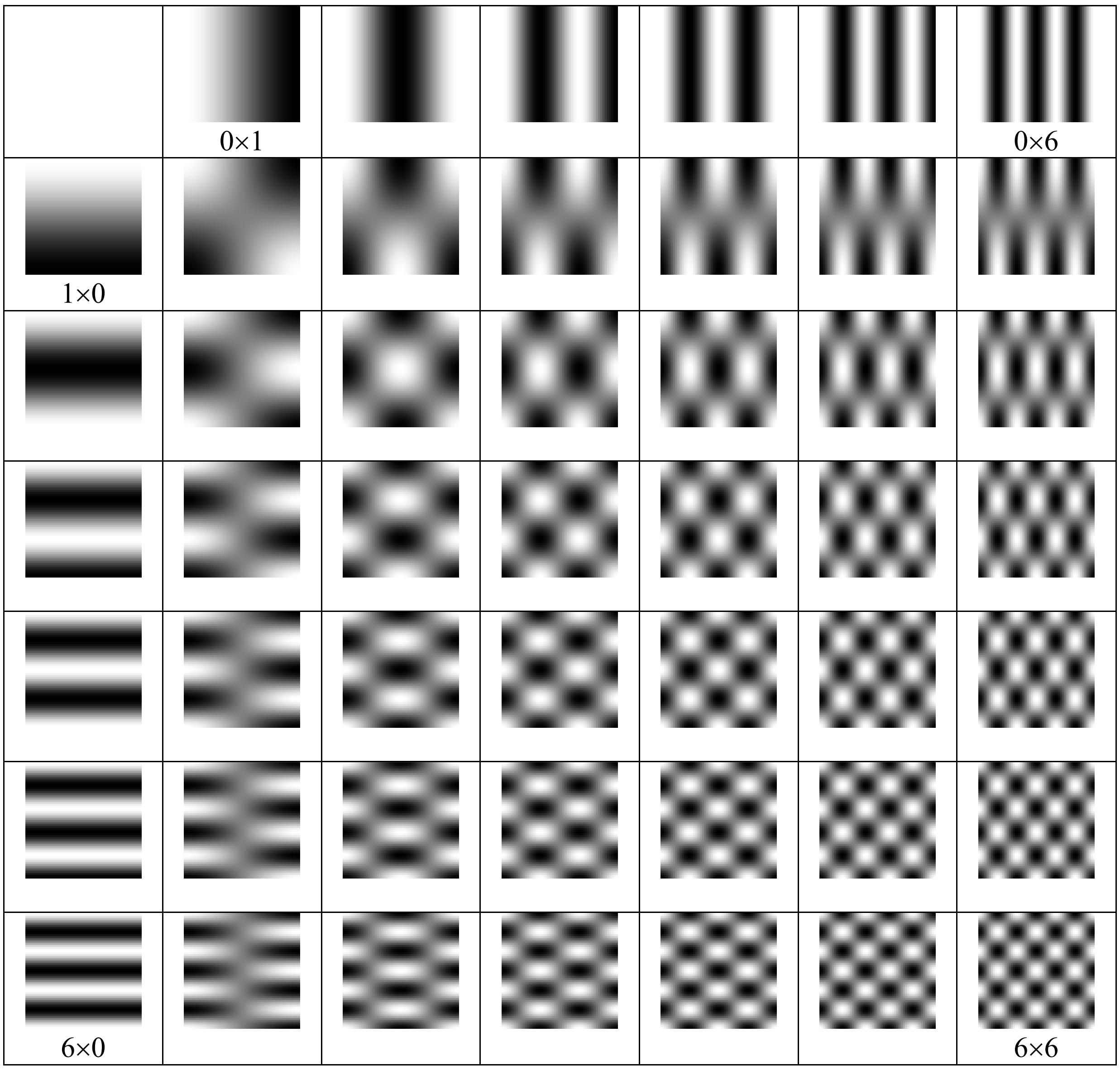
DCT pode ser calculada rapidamente a partir da rotina que calcula FFT.

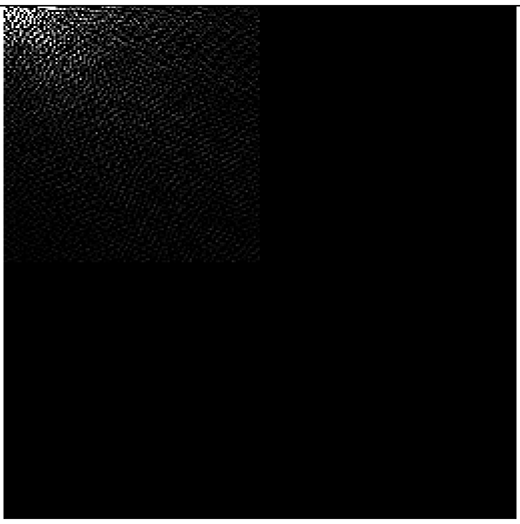
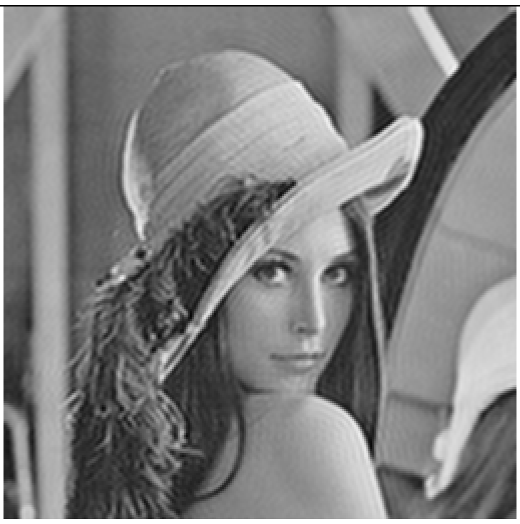
O seguinte programa calcula DCT da lennag.tga, e a reconstrói usando apenas *n* primeiros coeficientes.

// Versão 2006
```
#include <proeikon>

int main(int argc, char** argv)
{ if (argc!=2) {
    printf("DCT: Reconstroi lennag usando n primeiros coeficientes DCT\n");
    printf("DCT n\n");
    erro("Erro: Numero de argumentos invalido");
  }

  int n;
  sscanf(argv[1],"%d",&n);

  IMGFLT a; le(a,"lennag.tga");
  IMGFLT d=dct(a);

  VETOR<int> v; dctzigzag(a.nl(),a.nc(),v);

  IMGFLT e=d;
  for (int i=n; i<d.n(); i++)
    e(v(i))=0.0;

  a=idct(e);
  imp(a,"dct.tga");
}
```

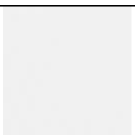
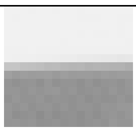
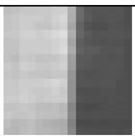
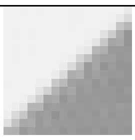
```
// Versao 2007 pós - usa 1/4 dos coeficientes de DCT.
#include <proeikon>

int main()
{ IMGFLT a; le(a,"lennag.tga");

  IMGFLT A=dct(a);

  for (int l=128; l<256; l++)
    for (int c=0; c<128; c++)
      A(l,c)=0.0;

  for (int l=128; l<256; l++)
    for (int c=128; c<256; c++)
      A(l,c)=0.0;

  for (int l=0; l<128; l++)
    for (int c=128; c<256; c++)
      A(l,c)=0.0;

  a=idct(A);
  imp(A,"dct.tga");
  imp(a,"compress.tga");
}
```

Base DCT



lenna	 dct
 compress	

Uso de DCT para detectar a direção da textura dominante.

 15.114706 -0.005217 0.005917 0.003244	 11.766275 0.040446 2.528655 0.011920	 9.089952 3.242645 0.308405 0.001655	 12.555147 1.419176 1.703597 -0.195742

```
#include <proeikon>
int main(int argc, char** argv)
{ if (argc!=2) erro("DCT entrada.tga");

  IMGFLT a; le(a,argv[1]);
  IMGFLT b=dct(a);
  printf("%10f %10f\n",b(0,0),b(0,1));
  printf("%10f %10f\n",b(1,0),b(1,1));
}
```

Haar Wavelet Transform

(Retirado de http://dmr.ath.cx/gfx/haar/)

To calculate the Haar transform of an array of n ($n=2^k$) samples:

1. Find the average of each pair of samples. ($n/2$ averages)
2. Find the difference between each average and the samples it was calculated from. ($n/2$ differences)
3. Fill the first half of the array with averages.
4. Fill the second half of the array with differences.
5. Repeat the process on the first half of the array.
   (The array length should be a power of two)

Average / Difference

Two samples, l and r, can be expressed as an average, a, and a difference, d, like in mid-side coding:

$a = (l + r) / 2$
$d = a - l = r - a$

This is reversible:

$l = a - d$
$r = a + d$

Example

Eight elements:

7	1	6	6	3	-5	4	2

Averages:
```
(7 +  1) / 2 =  4
(6 +  6) / 2 =  6
(3 + -5) / 2 = -1
(4 +  2) / 2 =  3
```
Differences:
```
(7 -  4) = ( 4 -  1) = 3
(6 -  6) = ( 6 -  6) = 0
(3 - -1) = (-1 - -5) = 4
(4 -  3) = ( 3 -  2) = 1
```

4	6	-1	3	3	0	4	1

Four elements:

4	6	-1	3	3	0	4	1

Averages:
```
( 4 + 6) / 2 =  5
(-1 + 3) / 2 =  1
```

Differences:

```
( 4 - 5) = (5 - 6) = -1
(-1 - 1) = (1 - 3) = -2
```

5	1	-1	-2	3	0	4	1

Two elements:

5	1	-1	-2	3	0	4	1

Averages:

```
(5 + 1) / 2 = 3
```

Differences:

```
(5 - 3) = (3 - 1) = 2
```

3	2	-1	-2	3	0	4	1

We can't recurse any further. Note that the first value in the resulting array is the average value of all the samples in the original array:

```
(7 + 1 + 6 + 6 + 3 - 5 + 4 + 2) / 8 = 3
```

2D Transform

Given a two-dimensional array of values, we can perform a 2D Haar transform by first performing a 1D Haar transform on each row:

-	-	-	>

And then on each column:

|			
|			
|			
V			

Transformada de Haar 2D de A resulta em H:

$A = \begin{bmatrix} 40 & 80 \\ 30 & 10 \end{bmatrix}$	$H = \begin{bmatrix} 40 & -5 \\ 20 & -15 \end{bmatrix}$

H ser interpretado de duas formas. Primeira forma:

$A \cdot \frac{1}{4} \cdot \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix}$	$A \cdot \frac{1}{4} \cdot \begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix}$
$A \cdot \frac{1}{4} \cdot \begin{bmatrix} +1 & +1 \\ -1 & -1 \end{bmatrix}$	$A \cdot \frac{1}{4} \cdot \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$

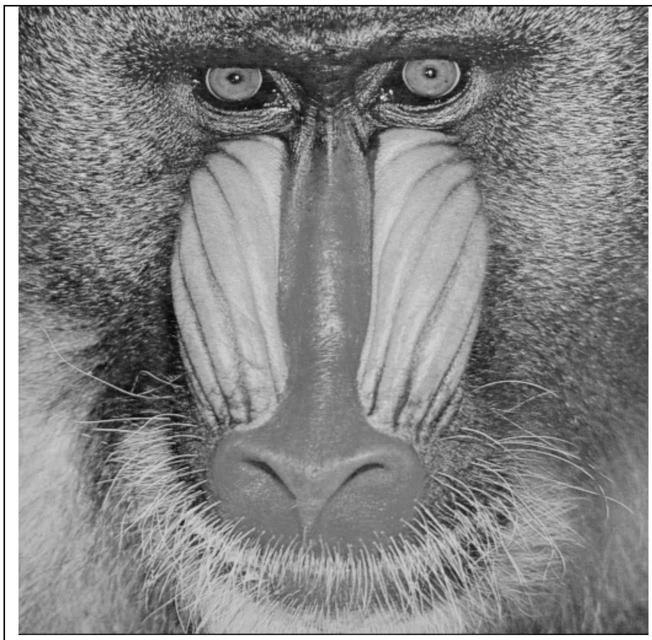
Segunda forma:

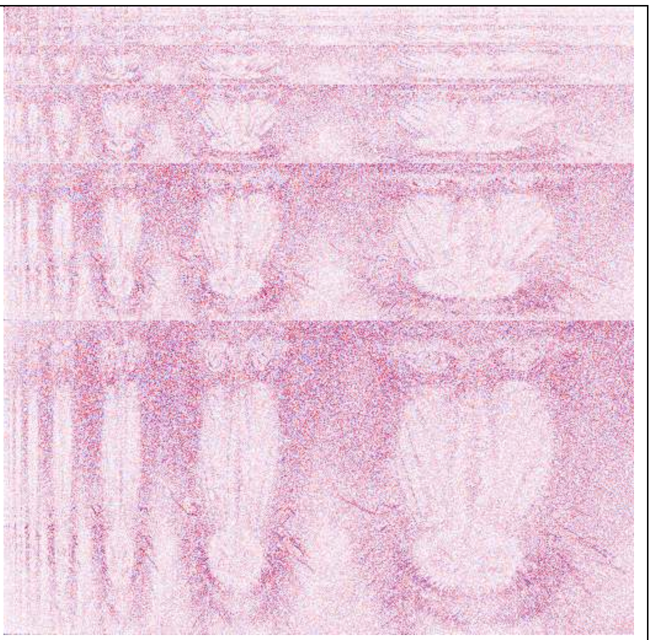
média de A	média da metade à esquerda de A - média de A
média da metade superior de A - média de A	média da diagonal principal de A - média de A

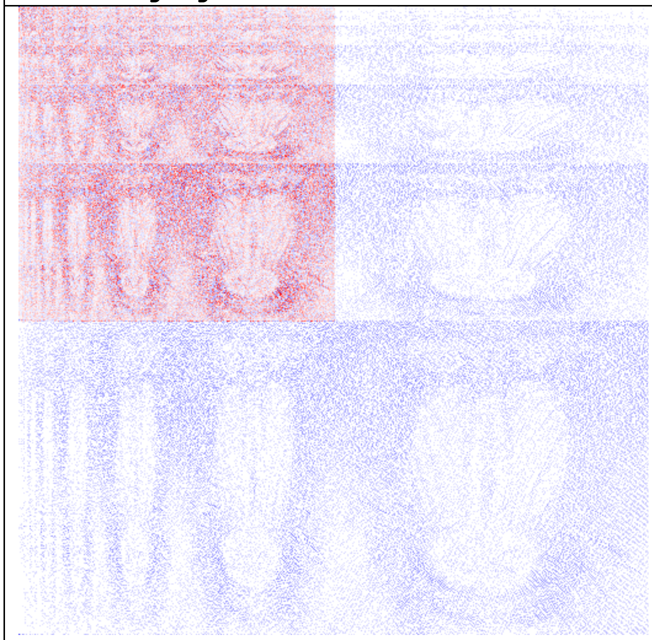
Para imagens maiores, este processo deve ser repetido recursivamete.

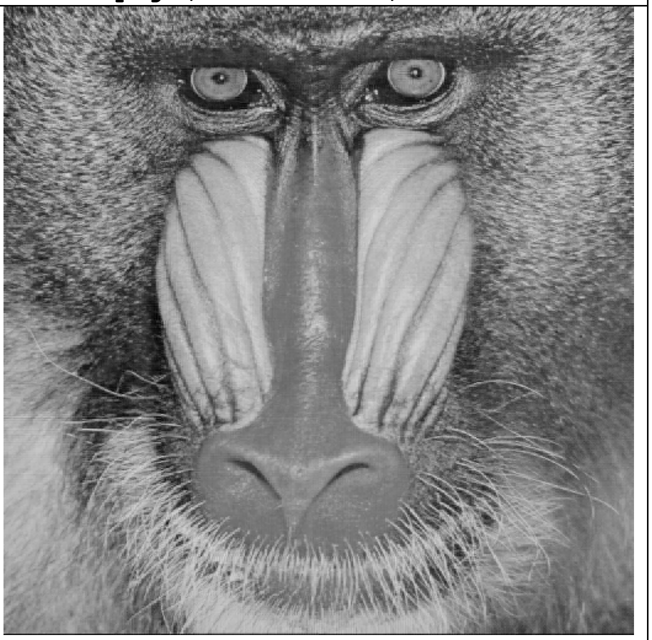
```
// Haar.cpp.
// Nota: funções haar, unhaar e haar2imc
// foram incluídas nas versões novas de Cekeikon
#include <cekeikon.h>

Mat_<FLT> haar(const Mat_<FLT> a)
{ assegura(nextexp2(a.cols)==a.cols && nextexp2(a.rows)==a.rows,
           "Erro dimensao 2");
  Mat_<FLT> d=a.clone();
  Mat_<FLT> e(1,max(a.rows,a.cols));
  for (int l=0; l<d.rows; l++)     //rows=512
    for (int m=d.cols; m>1; m/=2) { //m=512, 256, ..., 2
      int m2=m/2;
      for (int c=0; c<m2; c++) { //c=256, 128, ..., 1
        e(c) = (d(l,2*c)+d(l,2*c+1)) /2;
        e(m2+c) = d(l,2*c)-e(c);
      }
      for (int c=0; c<m; c++) d(l,c)=e(c);
    }
  //resultado em d
  for (int c=0; c<d.cols; c++)
    for (int m=d.rows; m>1; m/=2) {
      int m2=m/2;
      for (int l=0; l<m2; l++) {
        e(l) = (d(2*l,c)+d(2*l+1,c)) /2;
        e(m2+l) = d(2*l,c)-e(l);
      }
      for (int l=0; l<m; l++) d(l,c)=e(l);
    }
  return d;
}

Mat_<FLT> unhaar(const Mat_<FLT> a)
{ assegura(nextexp2(a.cols)==a.cols && nextexp2(a.rows)==a.rows,
           "Erro dimensao 2");
  Mat_<FLT> d=a.clone();
  Mat_<FLT> e(1,max(a.rows,a.cols));
  for (int l=0; l<d.rows; l++)     //rows=512
    for (int m=2; m<=d.cols; m*=2) { //m=2, 4, ..., 512
      int m2=m/2;
      for (int c=0; c<m2; c++) { //c
        e(2*c)   = d(l,c)+d(l,m2+c);
        e(2*c+1) = d(l,c)-d(l,m2+c);
      }
      for (int c=0; c<m; c++) d(l,c)=e(c);
    }
  // resultado em d
  for (int c=0; c<d.cols; c++)     //cols=512
    for (int m=2; m<=d.rows; m*=2) { //m=2, 4, ..., 512
      int m2=m/2;
      for (int l=0; l<m2; l++) { //l
        e(2*l)   = d(l,c)+d(m2+l,c);
        e(2*l+1) = d(l,c)-d(m2+l,c);
      }
      for (int l=0; l<m; l++) d(l,c)=e(l);
    }
 return d;
}

Mat_<FLT> zera(const Mat_<FLT> a, FLT limiar)
{ Mat_<FLT> d=a.clone();
```

```
  int conta=0;
  for (int l=0; l<d.rows; l++)
    for (int c=0; c<d.cols; c++) {
      int dist=max(l,c);
      if (d(l,c)<limiar && dist>a.rows/2) { d(l,c)=0; conta++; }
    }
  printf("zeros=%4.0f%% nao-zeros=%4.0f%%\n",
          100.0*conta/a.total(),100.0*(a.total()-conta)/a.total());
  return d;
}

Mat_<COR> haar2imc(const Mat_<FLT> f)
{ Mat_<COR> d(f.size());
  float max_neg=0.0, max_pos=0.0;
  for (int l=0; l<f.rows; l++)
    for (int c=0; c<f.cols; c++) {
        if (f(l,c) < max_neg) max_neg = f(l,c);
        if (f(l,c) > max_pos) max_pos = f(l,c);
      }
  for (int l=0; l<f.rows; l++)
    for (int c=0; c<f.cols; c++) {
      float v;
      if (f(l,c) < 0) v = f(l,c) / max_neg;
      else            v = f(l,c) / max_pos;
      GRY b = GRY(255 - sqrt(v)*255);
      if (f(l,c) < 0) d(l,c)=COR(b,b,255);
      else d(l,c)=COR(255,b,b);
    }
  return d;
}

int main()
{ Mat_<FLT> ent;
  assegura(nextexp2(ent.cols)==ent.cols && nextexp2(ent.rows)==ent.rows,
           "Erro dimensao 1");
  le(ent,"mandrillg.tga");
  Mat_<FLT> haa=haar(ent);
  Mat_<COR> imc=haar2IMC(haa);
  imp(imc,"haar4a.png");

  haa=zera(haa,0.005);
  imc=haar2imc(haa);
  imp(imc,"haar4b.png");

  Mat_<FLT> sai=unhaar(haa);
  imp(sai,"haar4c.png");
}
```


mandrillg.tga


haar4a.png (coeficientes)


haar4b.png (após zerar 50% dos coefs)


haar4c.png imagem compactada

haar4a: Coeficientes de haar
haar4b: Coeficientes de haar após zerar 50% dos coeficientes de menor valor nos 3 quadrantes sup-dir, inf-esq e inf-dir.

```
Diferença entre mandrillg.tga e haar4c.png:
      MAE (max=100%)...................:             3.21%
      RMSE (max=100%)..................:             4.70%
      PSNR considering highest=255.....:           26.5596 dB
```