

Bildverarbeitung: Übung 02

von
Naja v. Schmude (4127652)

Vorbemerkung: Leider hatte ich nicht dran gedacht, dass Octave nicht auf meinem Laptop installiert ist und konnte deshalb jetzt keine Screenshots mehr machen ;-)

Aufgabe 1

In der ersten Aufgabe sollte die FFT implementiert werden. Dazu braucht man natürlich die Fourier-Matrix, die durch folgendes rekursive Verfahren berechnet wird (man beachte, dass zu der "normalen" Fourier-Matrix diese vertauschte Spalten hat):

$$F_n = \begin{pmatrix} F_{n/2} & w_1 \odot F_{n/2} \\ F_{n/2} & -w_1 \odot F_{n/2} \end{pmatrix}$$

Wobei w_1 ein Spaltenvektor der Länge $\frac{n}{2}$ ist, der in der i ten Spalte den Wert ω_n^i besitzt. Der Operator \odot multipliziert dabei komponentenweise jede Spalte von $F_{n/2}$ mit w_1 . Wenn man dann die Fourier-Matrix so berechnet hat, dann wendet man eigentlich ganz normal die Transformation auf das Bild an. Man muss bloß beachten, dass man bei dem Bild auch entsprechend die Spalten vertauscht (allerdings hat das bei mir nicht so richtig funktioniert, und ich bekomme falsche Ergebnisse, vermutlich hab ich komisch gedacht und was falsch verstanden ...).

```
% do the transformation
function fft = fastFourierTransformation(img)
    [m,n] = size(img);
    f = fastFourierMatrix(n);
    s = swap(img); % all columns with even index first, then the odd ones
    fft = f * s * f';
endfunction

% generate the fast fourier matrix
function f = fastFourierMatrix(dim)
    if (dim == 1)
        f = 1;
    else
        w1 = [];
        for i=1:1:(dim / 2)
            w1 = [w1; rootOfUnit(i-1,dim)];
        end

        fHalf = fastFourierMatrix(dim / 2);
        mult = multColsWithVec(fHalf,w1);
        f = [fHalf, mult ; fHalf, -mult];
    endif
endfunction
```

Aufgabe 2

Da meine FFT aus Aufgabe 1 nicht so funktional war, hab ich die Octave eigene `fft2` und `ifft2` verwendet ...

Die Konvolution funktioniert (nach dem Satz) wie folgt:

$$konvolution = F^{-1}(FKF) \otimes (FBF)F^{-1}$$

Wobei \otimes die komponentenweise Multiplikation darstellt. Als Kernel kann man unterschiedliche Dinge nehmen, ausprobiert habe ich einen Gaußkernel der Form $\begin{pmatrix} 1 & 2 & 1 \\ \frac{1}{16} & 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ (aufgefüllt dann mit 0en), einen Sobelfilter für die x -Richtung und einen Laplace-Filter. Die Bilder kann ich leider wie oben schon vermerkt jetzt nicht mitdrucken, aber schick ich per Mail nach, sind superschick :-)

```
gauss = [1, 2, 1; 2, 4, 2; 1, 2, 1] / 16;
sobelX = [1, 2, 1; 0, 0, 0; -1, -2, -1];
laplace = [1, 1, 1; 1, -8, 1; 1, 1, 1];
kernel = zeros(n,n);
kernel(1: 3, 1:3) = laplace;

% Konvolution with DFT
fourierDFT = fourierMatrix(n);
fourierDFTInv = conjugatedFourierMatrix(n);
konvolutionFreqDFT = (fourierDFT * complex(kernel) * fourierDFT) .* (fourierDFT * complex
konvolutionDFT = fourierDFTInv * konvolutionFreqDFT * fourierDFTInv;

%Konvolution with FFT (with the version from octave, because my version has bugs ...)
konvolutionFreqFFT = fft2(kernel) .* fft2(grayImg);
konvolutionFFT = ifft2(konvolutionFreqFFT);
```

Wie man sich natürlich schon denken kann, ist die FFT Variante schneller (wobei natürlich die schon von dem Octave-Team implementierte Version optimal ist und das somit etwas geschummelt).