Bildverarbeitung SoSe 2012
Übung 4

Max Michels, Sebastian Kürten

1 Aufgabe 1

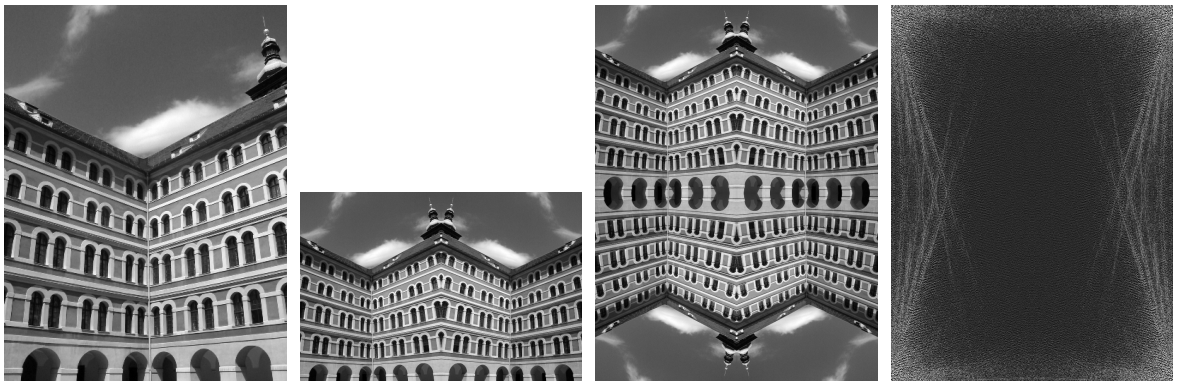
2 Aufgabe 2

(a) Original (b) Spiegelung 1 (c) Spiegelung 2 (d) Spektrum

Figure 1: lena-bw.png

(a) Original (b) Spiegelung 1 (c) Spiegelung 2 (d) Spektrum

Figure 2: skyline-bw.png



(a) Original (b) Spiegelung 1 (c) Spiegelung 2 (d) Spektrum

Figure 3: building-bw.png

Hauptdatei a2.m

```
[A, A1, A2, AK, AKN, AR, Aok] = kosinusTest("building-bw.png");
[B, B1, B2, BK, BKN, BR, Bok] = kosinusTest("lena-bw.png");
[C, C1, C2, CK, CKN, CR, Cok] = kosinusTest("skyline-bw.png");
```

Ausgabe

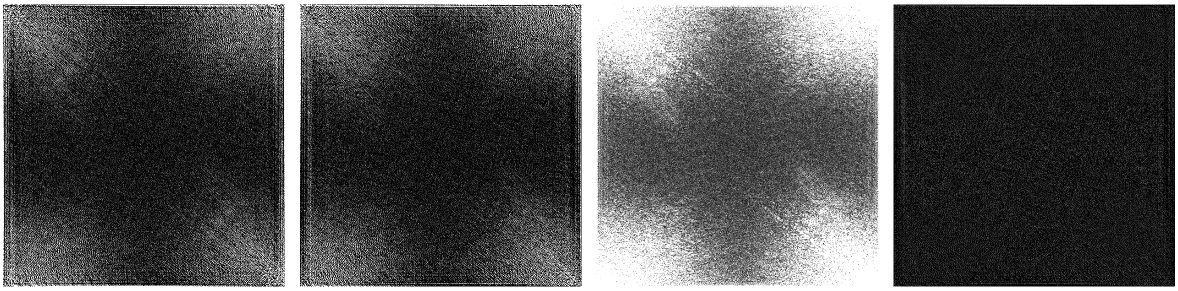
```
ok =  1
ok =  1
ok =  1
```

Hilfsfunktionen

```
function [B, B1, B2, K, KN, C, ok] = kosinusTest(name)
% read image to B
% create mirrored images in B1, B2
% cosine transformation of B2 in K
% back-transformed image in C
% ok is true if imaginary part of K is about zero
% KN contains a kind-of normalized version of
    B = imread(name);
    % mirror
    B1 = horzcat(B, fliplr(B)(:,1:end-1));
    B2 = vertcat(B1, flipud(B1)(1:end-1,:));
    % size
    [h, w] = size(B2);
    % fourier matrices
    fLeft = ffn(h);
    fRight = ffn(w);
    % transform
    K = fLeft * double(B2) * fRight;
    % transform back
    C = conj(fLeft) * K * conj(fRight);
    % test result
    ok = nearreal(K, 0.00001)
    % normalize result to grayscale
    KN = uint8(normalize(real(K), -10, 50) * 255);
    % creat some image files
    basename = name([1:1:end-4]);
    imwrite(B1, ["a2.", basename, ".mirror1.png"]);
    imwrite(B2, ["a2.", basename, ".mirror2.png"]);
    imwrite(KN, ["a2.", basename, ".spektrum.png"]);
end;
```

```
function A = ffn(n)
% create a fourier matrix with the specified dimension, optimized version ...
    which is a little faster
    gen = einheitswurzel(n, 1);
    rootofn = 1 / sqrt(n);
    A = arrayfun(@(x) rootofn * power(gen, x), [0:(n-1)]' * [0:(n-1)]);
end;
```

3 Aufgabe 3



(a) real (b) imag (c) abs (d) angle

Figure 4: Spektrum

Hauptdatei a3.m

```
% load lena image
LENA = imread("lena-bw.png");
%LENA = imread("lena64.png");

mwOrt = 1/9 * [1 1 1; 1 1 1; 1 1 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% execute mean filter in spatial domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
B = padMirror(LENA);
tic;
B1 = uint8(falte(B, mwOrt));
tOrt = toc;
printf("Zeit im Ortsraum: %.1f Sekunden\n", tOrt);
B2 = crop(B1, 1);
imwrite(B2, "ausgabe.a3.ort.png");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare some filters within the frequency domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare mean filter
mwFreq = scaleFilter(mwOrt, size(LENA, 1));
filter = fft2d(mwFreq, 32);

% prepare identity filter
nop = scaleFilter([1], size(LENA, 1));
nopFilter = fft2d(nop, 32);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% execute mean filter in frequency domain, with fft
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;
S1 = fft2d(double(LENA), 32);
S2 = S1 .* filter;
R1 = ifft2d(S2, 32);
tFreq1 = toc;
printf("Zeit im Frequenzraum mit FFT: %.1f Sekunden\n", tFreq1);
R2 = real(R1) * size(S1, 1);
imwrite(uint8(R2), "ausgabe.a3.freq.fft.png");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% execute mean filter in frequency domain, with dft
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%load("f64");
%df = f64;
load("f512");
```

```
df = f512;
tic;
S3 = df * double(LENA) * df;
S4 = S3 .* filter;
R3 = df * S4 * df;
tFreq2 = toc;
printf("Zeit im Frequenzraum mit DFT: %.1f Sekunden\n", tFreq2);
R4 = real(R1) * size(S3, 1);
imwrite(uint8(R2), "ausgabe.a3.freq.dft.png");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create plots of spectral image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imwrite(uint8(real(S1) * 32), "ausgabe.a3.a.real.png");
imwrite(uint8(abs(S1) * 32), "ausgabe.a3.a.abs.png");
imwrite(uint8(imag(S1) * 32), "ausgabe.a3.a.imag.png");
imwrite(uint8(angle(S1) * 32), "ausgabe.a3.a.angle.png");
```

Ausgabe

```
Zeit im Ortsraum: 24.3 Sekunden
Zeit im Frequenzraum mit FFT: 5.9 Sekunden
Zeit im Frequenzraum mit DFT: 3.4 Sekunden
```

Hilfsfunktionen

```
function A = scaleFilter(filter, newSize)
% create a scaled version of a filter that may be used in frequency domain

A = zeros(newSize);

h = size(filter, 1);
w = size(filter, 2);

ys = 0;
xs = 0;
ye = ys + h;
xe = xs + w;

A([ys+1:ye],[xs+1:xe]) = filter;

% put the filter in the top left cornder and then
% shift the matrix so that the center of the filter is at the
% first entry of the matrix (at position (1, 1))
vshift = -floor(h / 2);
hshift = -floor(w / 2);
A = shift(shift(A, vshift), hshift, 2);

end;
```

```
function A = fft2d(B, b)
% 2-dimensional fast fourier transform
    A = rowFFT(colFFT(B, b), b);
end;
```

```
function A = ifft2d(B, b)
% inverse 2-dimensional fast fourier transform
    A = irowFFT(icolFFT(B, b), b);
end;
```

```
function A = rowFFT(B, b)
% simple wrapper method to apply fast rowwise fourier transform
% recursion base is b
    fb = fn(b);
    A = complicatedRowFFT(B, fb, b, false);
end;
```

```
function A = colFFT(B, b)
% simple wrapper method to apply fast colwise fourier transform
% recursion base is b
    fb = fn(b);
    A = complicatedColFFT(B, fb, b, false);
end;
```

```
function A = irowFFT(B, b)
    fb = conj(fn(b));
    A = complicatedRowFFT(B, fb, b, true);
end;
```

```
function A = icolFFT(B, b)
    fb = conj(fn(b));
    A = complicatedColFFT(B, fb, b, true);
end;
```

```
function A = complicatedRowFFT(B, fb, b, isInverse)
    % size information
    n = size(B, 1);
    n2 = n / 2;
    % base case
    if n == b
        A = B * fb;
    % recurive case
    else
        % swap columns
        B = horzcat(B(:,[1:2:n]), B(:,[2:2:n]));
        % partition input into submatrices
        B11 = B([1:n2],[1:n2]);
        B12 = B([1:n2],[n2+1:n]);
        B21 = B([n2+1:n],[1:n2]);
        B22 = B([n2+1:n],[n2+1:n]);
        % recurse for submatrices
        FB11 = complicatedRowFFT(B11, fb, b, isInverse);
        FB12 = complicatedRowFFT(B12, fb, b, isInverse);
        FB21 = complicatedRowFFT(B21, fb, b, isInverse);
        FB22 = complicatedRowFFT(B22, fb, b, isInverse);
        % diagonal helper matrix with roots of unity
        H = ffthelper(n2);
        if isInverse
            H = conj(H);
        end;
        HFB12 = FB12 * H;
        HFB22 = FB22 * H;
        % calculate target submatrices
        A11 = FB11 + HFB12;
        A12 = FB11 - HFB12;
        A21 = FB21 + HFB22;
        A22 = FB21 - HFB22;
        % stitch together the submatrices
        Atop = horzcat(A11, A12);
```

```
        Abot = horzcat(A21, A22);
        A = vertcat(Atop, Abot);
        % scale
        A = A / sqrt(2);
    end;
end;
```

```
function A = complicatedColFFT(B, fb, b, isInverse)
    % size information
    n = size(B, 1);
    n2 = n / 2;
    % base case
    if n == b
        A = fb * B;
    % recurive case
    else
        % swap rows
        B = vertcat(B([1:2:n],:), B([2:2:n],:));
        % partition input into submatrices
        B11 = B([1:n2],[1:n2]);
        B12 = B([1:n2],[n2+1:n]);
        B21 = B([n2+1:n],[1:n2]);
        B22 = B([n2+1:n],[n2+1:n]);
        % recurse for submatrices
        FB11 = complicatedColFFT(B11, fb, b, isInverse);
        FB12 = complicatedColFFT(B12, fb, b, isInverse);
        FB21 = complicatedColFFT(B21, fb, b, isInverse);
        FB22 = complicatedColFFT(B22, fb, b, isInverse);
        % diagonal helper matrix with roots of unity
        H = ffthelper(n2);
        if (isInverse)
            H = conj(H);
        end;
        HFB21 = H * FB21;
        HFB22 = H * FB22;
        % calculate target submatrices
        A11 = FB11 + HFB21;
        A12 = FB12 + HFB22;
        A21 = FB11 - HFB21;
        A22 = FB12 - HFB22;
        % stitch together the submatrices
        Atop = horzcat(A11, A12);
        Abot = horzcat(A21, A22);
        A = vertcat(Atop, Abot);
        % scale
        A = A / sqrt(2);
    end;
end;
```

```
function A = ffthelper(n)
% shifting function needed within the fft
    A = zeros(n);
    for (i = 1 : n)
        A(i, i) = einheitswurzel(2*n,i-1);
    end;
end;
```

Korrektheitstest für die FFT (correct.m):

```
% test the correctness of all FFT implementations by comparing to the
% results of the discrete fourier transformation.

% all tests should return 1

% original data
LENA64 = imread("lena64.png");
B = double(LENA64);
% comparism values
RS = B * fn(64);
CS = fn(64) * B;
S = fn(64) * B * fn(64);

% test 1-dimensional transformation with different recursion bases
nearzero(rowFFT(B, 2) - RS, 0.000001)
nearzero(colFFT(B, 2) - CS, 0.000001)

nearzero(rowFFT(B, 4) - RS, 0.000001)
nearzero(colFFT(B, 4) - CS, 0.000001)

nearzero(rowFFT(B, 8) - RS, 0.000001)
nearzero(colFFT(B, 8) - CS, 0.000001)

% test 1-dimensional inverse transformation with different recursion bases
nearzero(irowFFT(RS, 2) - B, 0.000001)
nearzero(icolFFT(CS, 2) - B, 0.000001)

nearzero(irowFFT(RS, 4) - B, 0.000001)
nearzero(icolFFT(CS, 4) - B, 0.000001)

nearzero(irowFFT(RS, 8) - B, 0.000001)
nearzero(icolFFT(CS, 8) - B, 0.000001)

% test 2-dimensional transformation with different recursion bases
nearzero(fft2d(B, 2) - S, 0.000001)
nearzero(fft2d(B, 4) - S, 0.000001)
nearzero(fft2d(B, 8) - S, 0.000001)

% test 2-dimensional inverse transformation with different recursion bases
nearzero(ifft2d(S, 2) - B, 0.000001)
nearzero(ifft2d(S, 4) - B, 0.000001)
nearzero(ifft2d(S, 8) - B, 0.000001)
```

```
function o = nearzero(X, eps)
% test whether all values length within the matrix is below a certain epsilon
    if max(max(abs(X))) > eps
        o = 0;
    else
        o = 1;
    end
end
```