
Bildverarbeitung SoSe 2012
Übung 4

Max Michels, Sebastian Kürten

1 Aufgabe 1

2 Aufgabe 2

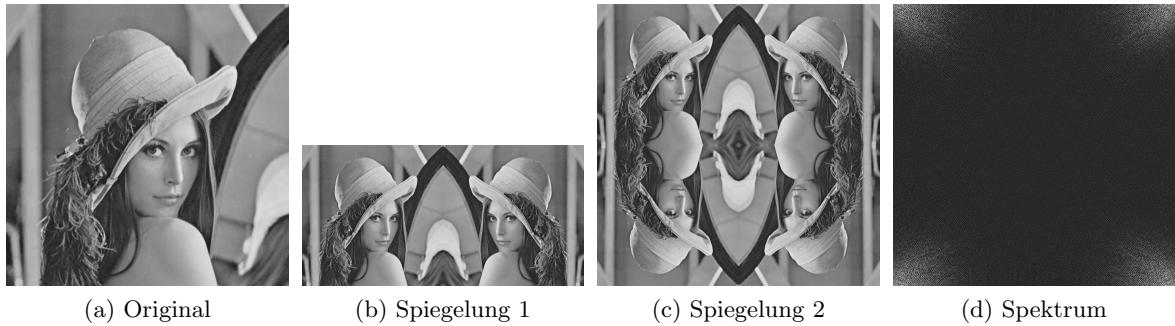


Figure 1: lena-bw.png

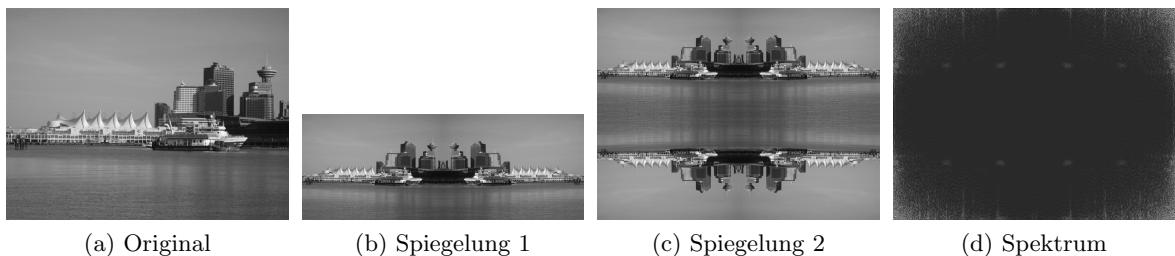


Figure 2: skyline-bw.png

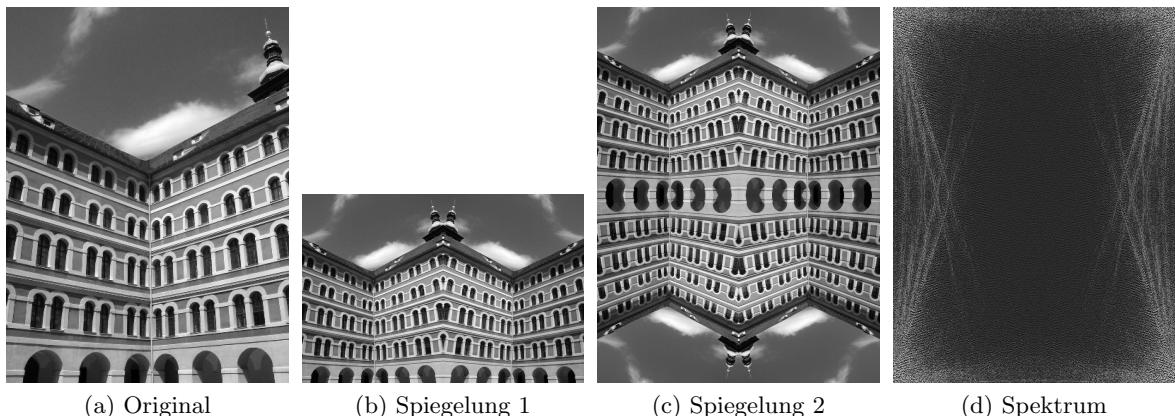


Figure 3: building-bw.png

Hauptdatei a2.m

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

Ausgabe

```
1 ok = 1
2 ok = 1
3 ok = 1
```

Hilfsfunktionen

```
1 function [B, B1, B2, K, KN, C, ok] = kosinusTest(name)
2 % read image to B
3 % create mirrored images in B1, B2
4 % cosine transformation of B2 in K
5 % back-transformed image in C
6 % ok is true if imaginary part of K is about zero
7 % KN contains a kind-of normalized version of
8 B = imread(name);
9 % mirror
10 B1 = horzcat(B, fliplr(B) (:,1:end-1));
11 B2 = vertcat(B1, flipud(B1)(1:end-1,:));
12 % size
13 [h, w] = size(B2);
14 % fourier matrices
15 fLeft = ffn(h);
16 fRight = ffn(w);
17 % transform
18 K = fLeft * double(B2) * fRight;
19 % transform back
20 C = conj(fLeft) * K * conj(fRight);
21 % test result
22 ok = nearreal(K, 0.00001)
23 % normalize result to grayscale
24 KN = uint8(normalize(real(K), -10, 50) * 255);
25 % creat some image files
26 basename = name([1:1:end-4]);
27 imwrite(B1, ["a2.", basename, ".mirror1.png"]);
28 imwrite(B2, ["a2.", basename, ".mirror2.png"]);
29 imwrite(KN, ["a2.", basename, ".spektrum.png"]);
30 end;
```

```
1 function A = ffn(n)
2 % create a fourier matrix with the specified dimension, optimized version ...
3 % which is a little faster
4 gen = einheitswurzel(n, 1);
5 rootofn = 1 / sqrt(n);
6 A = arrayfun(@(x) rootofn * power(gen, x), [0:(n-1)]' * [0:(n-1)]);
7 end;
```

3 Aufgabe 3

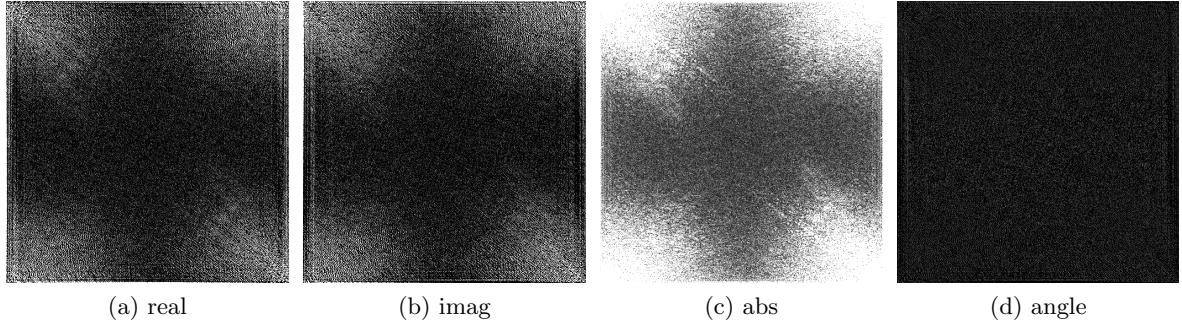


Figure 4: Spektrum

Hauptdatei a3.m

```

1 % load lena image
2 LENA = imread("lena-bw.png");
3 %LENA = imread("lena64.png");
4
5 mwOrt = 1/9 * [1 1 1; 1 1 1; 1 1 1];
6
7 %%%%%%%%%%%%%%
8 % execute mean filter in spatial domain
9 %%%%%%%%%%%%%%
10 B = padMirror(LENA);
11 tic;
12 B1 = uint8(falte(B, mwOrt));
13 tOrt = toc;
14 printf("Zeit im Ortsraum: %.1f Sekunden\n", tOrt);
15 B2 = crop(B1, 1);
16 imwrite(B2, "ausgabe.a3.ort.png");
17
18 %%%%%%%%%%%%%%
19 % prepare some filters within the frequency domain
20 %%%%%%%%%%%%%%
21 % prepare mean filter
22 mwFreq = scaleFilter(mwOrt, size(LENA, 1));
23 filter = fft2d(mwFreq, 32);
24
25 % prepare identity filter
26 nop = scaleFilter([1], size(LENA, 1));
27 nopFilter = fft2d(nop, 32);
28
29 %%%%%%%%%%%%%%
30 % execute mean filter in frequency domain, with fft
31 %%%%%%%%%%%%%%
32 tic;
33 S1 = fft2d(double(LENA), 32);
34 S2 = S1 .* filter;
35 R1 = ifft2d(S2, 32);
36 tFreq1 = toc;
37 printf("Zeit im Frequenzraum mit FFT: %.1f Sekunden\n", tFreq1);
38 R2 = real(R1) * size(S1, 1);
39 imwrite(uint8(R2), "ausgabe.a3.freq.fft.png");
40
41 %%%%%%%%%%%%%%
42 % execute mean filter in frequency domain, with dft
43 %%%%%%%%%%%%%%
44 %load("f64");
45 %df = f64;
46 load("f512");

```

```

47 df = f512;
48 tic;
49 S3 = df * double(LENA) * df;
50 S4 = S3 .* filter;
51 R3 = df * S4 * df;
52 tFreq2 = toc;
53 printf("Zeit im Frequenzraum mit DFT: %.1f Sekunden\n", tFreq2);
54 R4 = real(R1) * size(S3, 1);
55 imwrite(uint8(R2), "ausgabe.a3.freq.dft.png");
56
57 %%%%%%%%%%%%%%
58 % create plots of spectral image
59 %%%%%%%%%%%%%%
60
61 imwrite(uint8(real(S1) * 32), "ausgabe.a3.a.real.png");
62 imwrite(uint8(abs(S1) * 32), "ausgabe.a3.a.abs.png");
63 imwrite(uint8(imag(S1) * 32), "ausgabe.a3.a.imag.png");
64 imwrite(uint8(angle(S1) * 32), "ausgabe.a3.a.angle.png");

```

Ausgabe

```

1 Zeit im Ortsraum: 24.3 Sekunden
2 Zeit im Frequenzraum mit FFT: 5.9 Sekunden
3 Zeit im Frequenzraum mit DFT: 3.4 Sekunden

```

Hilfsfunktionen

```

1 function A = scaleFilter(filter, newSize)
2 % create a scaled version of a filter that may be used in frequency domain
3
4 A = zeros(newSize);
5
6 h = size(filter, 1);
7 w = size(filter, 2);
8
9 ys = 0;
10 xs = 0;
11 ye = ys + h;
12 xe = xs + w;
13
14 A([ys+1:ye], [xs+1:xe]) = filter;
15
16 % put the filter in the top left corner and then
17 % shift the matrix so that the center of the filter is at the
18 % first entry of the matrix (at position (1, 1))
19 vshift = -floor(h / 2);
20 hshift = -floor(w / 2);
21 A = shift(shift(A, vshift), hshift, 2);
22
23 end;

```

```

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

```

```

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

```

```

1 function A = rowFFT(B, b)
2 % simple wrapper method to apply fast rowwise fourier transform
3 % recursion base is b
4     fb = fn(b);
5     A = complicatedRowFFT(B, fb, b, false);
6 end;

```

```

1 function A = colFFT(B, b)
2 % simple wrapper method to apply fast colwise fourier transform
3 % recursion base is b
4     fb = fn(b);
5     A = complicatedColFFT(B, fb, b, false);
6 end;

```

```

1 function A = irowFFT(B, b)
2     fb = conj(fn(b));
3     A = complicatedRowFFT(B, fb, b, true);
4 end;

```

```

1 function A = icolFFT(B, b)
2     fb = conj(fn(b));
3     A = complicatedColFFT(B, fb, b, true);
4 end;

```

```

1 function A = complicatedRowFFT(B, fb, b, isInverse)
2     % size information
3     n = size(B, 1);
4     n2 = n / 2;
5     % base case
6     if n == b
7         A = B * fb;
8     % recursive case
9     else
10        % swap columns
11        B = horzcat(B(:, [1:2:n]), B(:, [2:2:n]));
12        % partition input into submatrices
13        B11 = B([1:n2], [1:n2]);
14        B12 = B([1:n2], [n2+1:n]);
15        B21 = B([n2+1:n], [1:n2]);
16        B22 = B([n2+1:n], [n2+1:n]);
17        % recurse for submatrices
18        FB11 = complicatedRowFFT(B11, fb, b, isInverse);
19        FB12 = complicatedRowFFT(B12, fb, b, isInverse);
20        FB21 = complicatedRowFFT(B21, fb, b, isInverse);
21        FB22 = complicatedRowFFT(B22, fb, b, isInverse);
22        % diagonal helper matrix with roots of unity
23        H = ffthelper(n2);
24        if isInverse
25            H = conj(H);
26        end;
27        HFB12 = FB12 * H;
28        HFB22 = FB22 * H;
29        % calculate target submatrices
30        A11 = FB11 + HFB12;
31        A12 = FB11 - HFB12;
32        A21 = FB21 + HFB22;
33        A22 = FB21 - HFB22;
34        % stitch together the submatrices
35        Atop = horzcat(A11, A12);

```

```

36         Abot = horzcat(A21, A22);
37         A = vertcat(Atop, Abot);
38         % scale
39         A = A / sqrt(2);
40     end;
41 end;

```

```

1 function A = complicatedColFFT(B, fb, b, isInverse)
2 % size information
3 n = size(B, 1);
4 n2 = n / 2;
5 % base case
6 if n == b
7     A = fb * B;
8 % recursive case
9 else
10    % swap rows
11    B = vertcat(B([1:2:n],:), B([2:2:n],:));
12    % partition input into submatrices
13    B11 = B([1:n2],[1:n2]);
14    B12 = B([1:n2],[n2+1:n]);
15    B21 = B([n2+1:n],[1:n2]);
16    B22 = B([n2+1:n],[n2+1:n]);
17    % recurse for submatrices
18    FB11 = complicatedColFFT(B11, fb, b, isInverse);
19    FB12 = complicatedColFFT(B12, fb, b, isInverse);
20    FB21 = complicatedColFFT(B21, fb, b, isInverse);
21    FB22 = complicatedColFFT(B22, fb, b, isInverse);
22    % diagonal helper matrix with roots of unity
23    H = ffthelper(n2);
24    if (isInverse)
25        H = conj(H);
26    end;
27    HFB21 = H * FB21;
28    HFB22 = H * FB22;
29    % calculate target submatrices
30    A11 = FB11 + HFB21;
31    A12 = FB12 + HFB22;
32    A21 = FB11 - HFB21;
33    A22 = FB12 - HFB22;
34    % stitch together the submatrices
35    Atop = horzcat(A11, A12);
36    Abot = horzcat(A21, A22);
37    A = vertcat(Atop, Abot);
38    % scale
39    A = A / sqrt(2);
40 end;
41 end;

```