Bildverarbeitung SoSe 2012
Übung 6

Max Michels, Sebastian Kürten

1 Aufgabe 1

Listing 1: Hauptdatei a1.m

```
1  LENA = imread("lena-bw.png");
2  %LENA = imread("lena64.png");
3
4  %%% part a
5  % create gaussian pyramide
6
7  P = gaussPyramide(LENA);
8
9  % save all images of the pyramide in files
10 PI = P;
11 while(1)
12     B = PI.b;
13     [h,w] = size(B);
14     name = sprintf("lena.gauss.pyramid.%d.png", w);
15     printf("size: %d,%d, name: %s\n", h, w, name);
16     imwrite(uint8(B), name);
17     if (!isfield(PI, "next"))
18         break;
19     end;
20     PI = PI.next;
21 end;
22
23 %%% part b
24 % create laplacian pyramide
25
26 L = laplacePyramide(LENA);
27
28 % save all images of the pyramide in files
29 LI = L;
30 while(1)
31     B = LI.b;
32     [h,w] = size(B);
33     name = sprintf("lena.laplace.pyramid.%d.png", w);
34     printf("size: %d,%d, name: %s\n", h, w, name);
35     imwrite(normalize0255(B), name);
36     if (!isfield(LI, "next"))
37         break;
38     end;
39     LI = LI.next;
40 end;
41
42 %%% part c
43 % reconstruct image and save reconstructed images
44
45 Li = invert(L); % we need the pyramide starting with 1px
46 G = Li.b; % base image
47 LI = Li.next; %iterator
48 while(1)
49     B = LI.b;
50     [h,w] = size(B);
```

```
51      name = sprintf("lena.reconstructed.%d.png", w);
52      printf("size: %d,%d, name: %s\n", h, w, name);
53      % compute reconstructed image
54      R = enlarge2(G) + B;
55      imwrite(uint8(R), name);
56      G = R;
57      if (!isfield(LI, "next"))
58          break;
59      end;
60      LI = LI.next;
61  end;
```

1.1 Teil a: Gauß-Pyramide

Die Gaußpyramide wurde mit einer rekursiven Funktion implementiert, die eine verkettete Liste als Ergebnis aufbaut. Dabei steht in jedem Element der Liste im Attribut 'b' das Bild und im Attribut 'next' ein Zeiger auf den nächsten Listeneintrag. Solange das jeweilige Bild groß genug ist, wird zunächst ein 5x5-Gaußfilter per Konvolution auf das Bild angewendet und dieses dann mit einer 2x2-Mittelwertsreduktion auf die halbe Größe skaliert.

Listing 2: Gauß-Pyramide: gaussPyramide.m

```
1   function P = gaussPyramide(B)
2       % create a gaussian pyramide and store it
3       % in a linked list
4       [h, w] = size(B);
5       %printf("%d,%d\n", h, w);
6       if (h == 1 || w == 1)
7           % stop at size of 1
8           P = struct("b", B);
9           return;
10      end;
11      % otherwise, create a thumbnail and recurse
12      % first, apply gaussian blurr if possible
13      C = B;
14      if (h > 2)
15          C = falte(B, ones(5)/25);
16      end;
17      % then shrink the image
18      thumb = shrink2x2(C);
19      % recurse with thumbnail
20      R = gaussPyramide(thumb);
21      % append to the tail of the list
22      P = struct("b", B, "next", R);
23  end;
```

1.2 Teil b: Laplace-Pyramide

Die Laplace-Pyramide wurde mit der gleichen Struktur wie die Gauß-Pyramide implementiert. Das verkleinerte Bild wird durch duplizieren jeweils eines Pixels auf 3 weitere hochskaliert, um es vom Originalbild abziehen zu können und die Differenz zu bilden.

Listing 3: Laplace-Pyramide: laplacePyramide.m

```
1   function P = laplacePyramide(B)
2       % create a gaussian pyramide and store it
3       % in a linked list
4
5       % ensure we don't have uint8 anymore
```

```
    B = double(B);
    [h, w] = size(B);
    printf("%d,%d\n", h, w);
    % easy case, size of 1, break
    if (h == 1 || w == 1)
        P = struct("b", B);
        return;
    end;
    % prepare the shrinked image
    % with gaussian blurr if possible
    C = B;
    if (h > 2)
        C = falte(B, ones(5)/25);
    end;
    % create the thumbnail
    thumb = shrink2x2(C);
    % sample up again
    reenlarged = enlarge2(thumb);
    % subtract from original image
    L = B - reenlarged;
    % recurse
    R = laplacePyramide(thumb);
    P = struct("b", L, "next", R);
end;
```

(a) size 512 (b) size 256 (c) size 128 (d) size 64

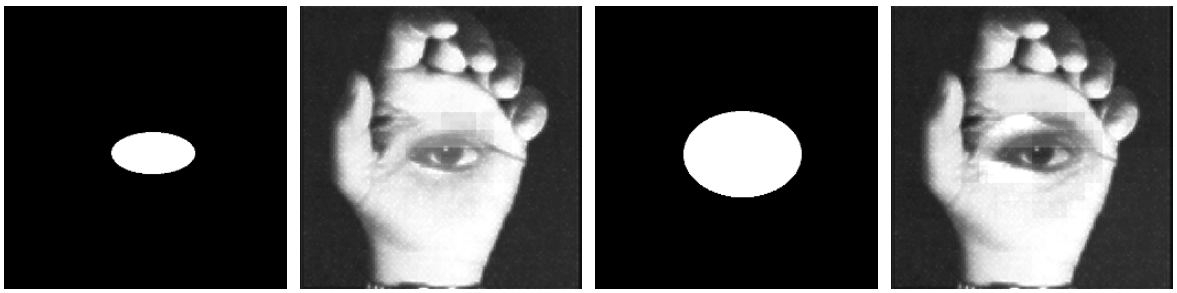
Figure 1: Teile der Laplace-Pyramide

2 Aufgabe 2

Listing 4: Hauptdatei a2.m

```
MASK = imread("eye-mask.png");
HAND = imread("hand-bw.png");
EYE = imread("eye-bw.png");

Pmask = gaussPyramide(MASK);
Lhand = laplacePyramide(HAND);
Leye = laplacePyramide(EYE);

M = invert(Pmask);
A = invert(Lhand);
B = invert(Leye);

%% create mixed image's laplace pyramid
% G
GA = A.b;
GB = B.b;
GX = M.b/255 * GA + (1-M.b/255) * GB;
X = struct("b", GX);
% iterators
MI = M.next;
AI = A.next;
BI = B.next;
% loop
while(1)
    LX = (MI.b/255) .* BI.b + (1-MI.b/255) .* AI.b;
    X = struct("b", LX, "next", X);
    if (!isfield(MI, "next"))
        break;
    end;
    MI = MI.next;
    AI = AI.next;
    BI = BI.next;
end;

%% reconstruct image from laplace pyramid
X = invert(X);
G = X.b;
LI = X.next;
while(1)
    B = LI.b;
    [h,w] = size(B);
    name = sprintf("mixed.%d.png", w);
    printf("size: %d,%d, name: %s\n", h, w, name);
    R = enlarge2(G) + B;
    imwrite(uint8(R), name);
    G = R;
    if (!isfield(LI, "next"))
        break;
    end;
    LI = LI.next;
end;
```

Die Mischung der Bilder wurde wie oben stehend implementiert. Im Grunde genommen wird nur parallel durch die drei Bildpyramiden iteriert und dabei eine neue Bildpyramide aufgebaut (dabei gemäß der jeweiligen Maske gemischt). Die entandende Laplace-Pyramide wird dann wie schon in Aufgabe 1 gesehen, dazu benutzt, ein Bild zu rekonstruieren. Das Ergebnis ist leider nicht ganz so schön, denn es sind kachelartige Artefakte sichtbar. Dies liegt eventuell daran, dass die simple, mittelwertsbasierte Implementierung der Pyramiden gewählt wurde.



(a) Maske 1 (b) Mischung 1 (c) Maske 2 (d) Mischung 2

Figure 2: Erebnisse des Mischens mit Laplace-Pyramide