Bildverarbeitung SoSe 2012
Übung 7

Max Michels, Sebastian Kürten

1 Aufgabe 1

Bildpyramiden - Wavelet: Bildpyramiden sind ein Spezialfall der Wavelettransformation.

Wavelet - DFT: bei beidem werden die Bilder in einene Frequenzraum transformiert, der Hauptunterschied ist, dass die DFT global arbeitet und die Wavelettransformation vor allem auch lokale Frequenzen enthält.

2 Aufgabe 2

Es wurde die 2D-Wavelet-Transformation mit Haar-Wavelets implementiert. Dabei wurde zur Implementierung das Lifting-Schema verwendet. Kern der Implementierung ist die Version für Hin- und Rücktransformation einer einzelnen Spalte, auf der die Gesamttransformation aufgebaut ist.

In der Hauptdatei werden dann für die beiden Eingabebilder im Wavelet-Raum kleine Werte im Sinne des Absolutbetrags auf Null gesetzt. Dies geschieht in der Funktion 'compress'. Es wurden anteilig die kleinsten vorkommenden absoluten Werte im Bild auf Null gesetzt, d.h. bspw. die Angabe 0.9 bedeutet, dass 90 Prozent aller Werte auf Null gesetzt wurden. Unten sind die Abbildungen der auf diese Art gewonnen Bilder zu sehen.

Listing 1: Hauptdatei a2.m

```
values = [0.5 0.6 0.7 0.8 0.9 0.95 0.97];

for pic = {"lena", "old-lady"}
%for pic = {"lena64"}
    pic = pic{1};
    src = sprintf("%s-bw.png", pic)

    B = normalize(imread(src));
    w = size(B, 2);
    C = compress(B, values);

    for i = [1:size(values, 2)]
        value = values(i);
        name = sprintf("%s.compressed.%.2f.png", pic, value)
        p = ((i-1) * w);
        Ci = uint8(denormalize(C(:,[p + 1:p + w])));
        imwrite(Ci, name);
    end;
end;
```

Listing 2: Kompression

```
function A = compress(B, values)
    A = B;
    H = haar(B);
```

```
4
5       for i = values
6           cutOff = percentile(H, i);
7           HM = setToZeroAbs(H, cutOff);
8           M = ihaar(HM);
9           A = horzcat(A, M);
10      end;
11
12  end;
```

Listing 3: Haar transformation with Lifting schema

```
1  function A = haar(B)
2      A = haarCols(haarRows(B));
3  end;
```

Listing 4: Haar for rows (based on column version)

```
1  function A = haarRows(B)
2      A = haarCols(B')';
3  end;
```

Listing 5: Haar for columns

```
1  function A = haarCols(B)
2      [h, w] = size(B);
3      A = zeros(h, w);
4      for i = [1 : w]
5          A(:,i) = haarSingleCol(B(:,i));
6      end;
7  end;
```

Listing 6: Haar transform for a single column

```
1  function A = haarSingleCol(B)
2  % apply the haar transform to a single column
3      h = size(B, 1);
4      A = zeros(h, 1); % overall result
5      gpu = zeros(h / 2, 1); % result of add
6      gmu = zeros(h / 2, 1); % result of sub
7      for i = [1 : h/2] % calculate add
8          gpu(i) = (B((i-1)*2 + 1) + B((i-1)*2 + 2)) / sqrt(2);
9      end;
10     for i = [1 : h/2] % calculate sub
11         gmu(i) = (B((i-1)*2 + 1) - B((i-1)*2 + 2)) / sqrt(2);
12     end;
13     A(h/2+1:end) = gmu;
14     if h == 2
15         % stop for length 2
16         A(1) = gpu;
17     else
18         % recurse into one half
19         A(1:h/2) = haarSingleCol(gpu);
20     end;
21 end;
```

Listing 7: Inverse haar transformation

```
function A = ihaar(B)
    A = ihaarCols(ihaarRows(B));
end;
```

Listing 8: Inverse haar transformation (based on columns)

```
function A = ihaarRows(B)
    A = ihaarCols(B')';
end;
```

Listing 9: Inverse haar transformation for columns

```
function A = ihaarCols(B)
    [h, w] = size(B);
    A = zeros(h, w);
    for i = [1 : w]
        A(:,i) = ihaarSingleCol(B(:,i));
    end;
end;
```

Listing 10: Inverse Haar for a single Column

```
function A = ihaarSingleCol(B, depth = 0)
% revert the haar transform of a single column
    h = size(B, 1); % the size of the current vector
    if h == 2 % stop recursion if h = 2
        % the first two values are calculated from
        % the last two entries of the vector
        A = [B(1) + B(2); B(1) - B(2)] / sqrt(2);
    else
        % we deduct one part by adding and the
        % other by subtracting parts of the vector
        S = ihaarSingleCol(B(1:h/2), depth + 1);
        T = B(h/2+1:end);
        A = [S + T; S - T] / sqrt(2);
        % fancy permutation to reorder rows
        sel = [rem([0:h-2]*h/2, h-1) h-1] + 1;
        A = A(sel);
    end;
end;
```

3 Aufgabe 3

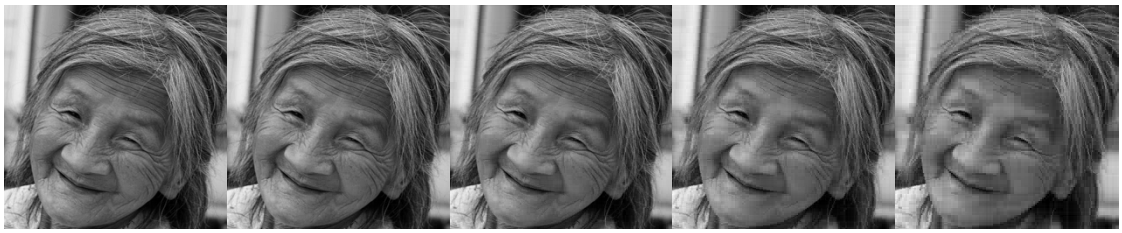
Wir würden einen Kantenfilter auf das Bild anwenden und anschließend überprüfen, ob sich eine Kante an der erwarteten Stelle im Bild befindet. Da die Flaschen immer an der gleichen Stelle stehen ist wenn die Aufnhame des Bildes gemacht wird, sollte, von kleinen Abweichungen abgesehen immer in etwa an der gleichen Stelle mit der Kante zu rechnen sein. Als Kantenfilter könnte man zum Beispiel einen Sobel-XY Filter verwenden. Um dann die Kante zu identifizieren könnte man prüfen ob im erwarteten Bereich eine Häufung von hellen Punkten zu finden ist. Dazu könnte man die Akkumulatortechnik verwenden und nach einigen Experimenten mit korrekt und schlecht befüllten Flaschen einen Schwellwert festlegen, der zur Erkennung der Kante geeignet ist. Da es sich immer um horizontale Kanten handeln muss (wegen der Schwerkraft) könnte man auch einen Sobel-Y oder einen Laplace-Filter verwenden.

Alternativ könnte man auch eine Hough-Transformation anwenden um die Kanten zu identifizieren. Auch hier würde sich nach einigen Experimenten ein recht enger Bereich identi-

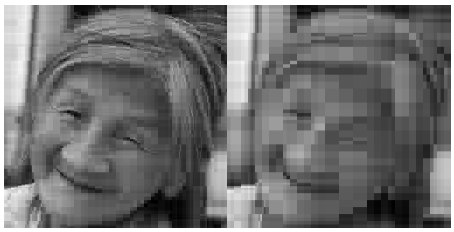
(a) 0.50 (b) 0.60 (c) 0.70 (d) 0.80 (e) 0.90

(f) 0.95 (g) 0.97

Figure 1: lena



(a) 0.50 (b) 0.60 (c) 0.70 (d) 0.80 (e) 0.90



(f) 0.95 (g) 0.97

Figure 2: old-lady

fizieren in dem sich im Akkumulatorbild die Häufungspunkte befinden müssten. Eventuell könnte man auch hier das Verfahren verbessern, indem lediglich auf horizontale Kanten getestet würde.