

Bildverarbeitung

Jan Fässler

5. Semester (HS 2013)

Inhaltsverzeichnis

1 Einführung	1
1.1 Elektromagnetische Strahlung	1
1.2 Sensor-Arrays	1
1.3 Farbbilderzeugung	1
1.4 Digitales Rasterbild	2
1.5 Raster- vs. Vektordaten	2
1.6 Wertebereiche von Pixeln	3
1.7 Region of Interest	3
2 Bildspeicherung	4
2.1 Bildformate	4
2.2 PGM-Beispiel	4
2.3 TIFF-Struktur	4
2.4 JPEG-Prozesskette	5
2.5 Bildkompression	5
2.6 Codierung	5
2.6.1 mittlere Codelänge einer Symbolfolge	5
2.6.2 Huffman-Code	6
2.7 Unterabtastung	6
2.8 quantisierung	6
3 Punktoperationen	7
3.1 Definition	7
3.1.1 homogen	7
3.1.2 nicht homogen	7
3.2 Kontrast und Helligkeit	7
3.3 Automatische Kontrastanpassung	7
3.4 Schwellwertoperation	8
3.5 Histogrammausgleich	8
3.6 Gammakorrektur	8
3.7 Bildqualität	9
4 Farben	10
4.1 Farbsysteme	10
4.1.1 lineare Farbsysteme	10
4.1.2 nicht lineare Farbsysteme	10
4.2 Farträume	11
4.2.1 RGB	11
4.2.2 HSV und HLS	11
4.2.3 CIExyz	11
4.2.4 CIExy	11
4.3 Umwandlungen	12
4.3.1 CMY und CMYK	12
4.3.2 CIExy	12
4.3.3 Transformation sRGB → XYZ	12
4.4 Weisspunkte	12
5 Filter	14
5.1 Anwendung	14
5.2 Gauss-Filter	14
5.3 Laplace-Filter	14
5.4 Faltung	15
5.4.1 diskrete Faltung	15
5.4.2 Eigenschaften der Faltung	15
5.5 Separierbarkeit	15

6 Morphologie	16
6.1 Schrumpfen und Wachsen	16
6.2 Grundoperationen	16
6.3 Zusammengesetzte Operationen	16
6.4 Grauwert-Morphologie	17
7 Binärbildanalyse	18
7.1 Prozesskette	18
7.2 Flood filling	18
7.3 Rekursives Flood Filling	18
7.4 Hough-Transformation	18
8 Mustererkennung (Pattern Matching)	19
8.1 Definitionen	19
8.2 Ähnlichkeitsmasse	19
8.3 Chamfer-Algorithmus	19
9 Fourier-Transformation	20
9.1 Frequenz, Amplitude, Phase	20
9.2 Fourierreihe	20
9.3 Fourierintegral und -spektrum	20
9.4 Fouriertransformation	20
9.5 Fourier-Transformationspaare	21
9.6 Beispiel: Rücktransformation	21
9.7 FT von diskreten Signalen	22
9.8 Aliasing und Abtasttheorem	22
9.9 Diskrete Fouriertransformation	22
9.10 FFT und DCT	23
9.11 DFT in 2D	24
9.12 Implementierung der 2D-DFT	24
9.13 SD Algorithmus	25
10 Kanten und Ecken	26
10.1 Übersicht	26
10.2 Gradient	26
10.2.1 Ableitungsfilter	26
10.2.2 Kantendetektierung	27
10.2.3 1. und 2. Ableitung	27
10.3 Laplacian of Gaussian	27
10.3.1 Kantenschärfung	28
10.3.2 Anwendung des Laplace-Filters	28
10.4 Eckpunkte	28
10.4.1 Anforderungen	29
10.4.2 Strukturmatrix	29
10.4.3 Eigenwerte und Eigenvektoren	29
10.4.4 Ähnliche Matrizen	29
10.4.5 Diagonalisierte Strukturmatrix	29
10.4.6 Spezifische Interpretation	30
10.4.7 Corner Response Function (CRF)	30
10.5 Harris Algorithmus	31
11 Wavelet-Transformation	32
11.1 Einsatz der WT	32
11.2 Idee der Bildpyramide	32
11.3 L_2 -Norm	32
11.4 Mother Wavelet und Wavelet-Familien	33
11.5 Continuous Wavelet Transform	33
11.6 CWT, DWT und FWT	33

11.7 FWT	34
11.8 FWT als Filteroperation	34
11.9 Rücktransformation	35
11.10FWT Filterbank	35
11.11FWT in zwei Dimensionen	35
11.12Bilddaten-Dekorrelation	36
11.13Quantisierung	36
12 PGF	37
12.1 Features	37
12.2 Aufbau	37
12.3 Farbtransformation	37
12.4 5/3 Wavelet Filterbank	37
12.5 Skalare Quantisierung	38
12.6 Progressive Kodierung	38
12.7 Bitplane-Kodierung	39
12.8 Lauflängenkodierung (RLE)	39
12.9 Adaptives RLE	39
13 Uebungen	39
13.1 Uebung 1	39
13.2 Uebung 2	41
13.3 Uebung 3	47
13.4 Uebung 5	50
13.5 Uebung 6	53
13.6 Uebung 7	57
13.7 Uebung 8	59
13.8 Uebung 10	61

1 Einführung

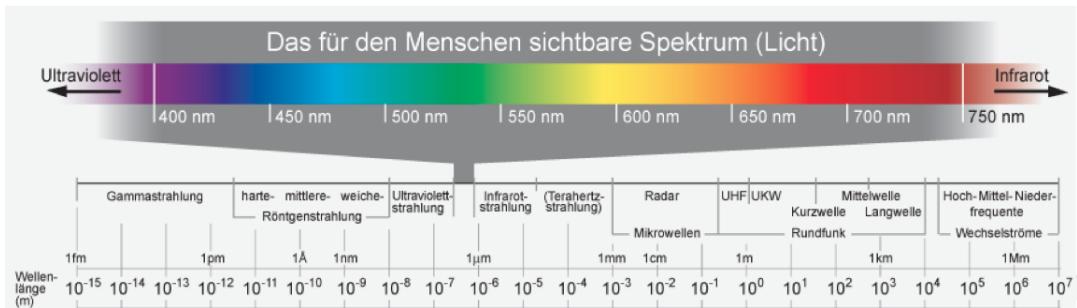
1.1 Elektromagnetische Strahlung

Wellenlänge

$$\lambda = \frac{c}{f} [m], \text{ mit } c = \text{Lichtgeschwindigkeit und } f = \text{Frequenz}$$

Energie der elektromagnetischen Strahlung

$$E = h * f [eV], \text{ mit } h = 6.626 * 10^{-34} \text{ Js} = 4.136 * 10^{-15} \text{ eVs}$$



1.2 Sensor-Arrays

Funktionsprinzip

- Ladungen werden in einem Kondensator gesammelt
- die Ladungsmenge ist proportional zur eingestrahlten Lichtmenge, wenn rechtzeitig ausgelesen wird, bevor die Leerlaufspannung der Fotodiode erreicht ist

CCD

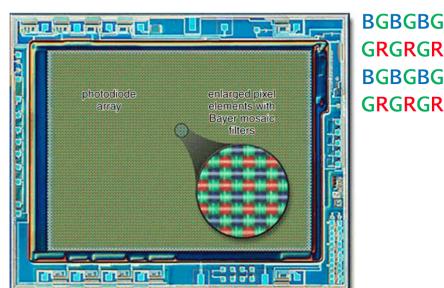
- Zeilen werden der Reihe nach ausgelesen
- pro Zeile werden die Ladungen serialisiert
- alle Ladungsmengen werden durch einen einzigen Ausleseverstärker geleitet

CMOS

- zu jeder Fotodiode gehört ein eigener Verstärker, welcher die Kondensatorspannung dem Analogsignalprozessor direkt zur Verfügung stellt

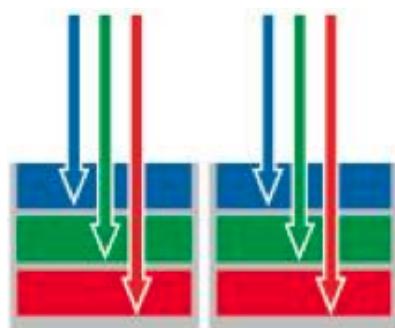
1.3 Farbbilderzeugung

Bayer-Maske

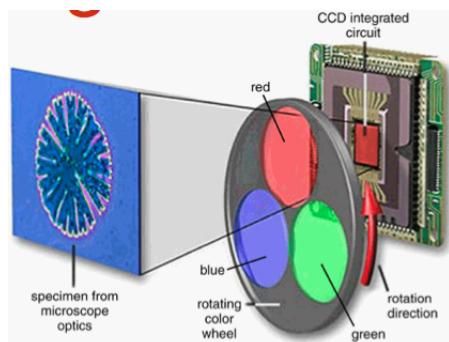


Andere

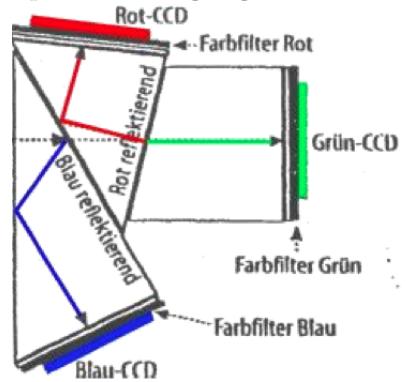
Foveon



Filterrad mit RGB-Filter



Spektralzerlegung



1.4 Digitales Rasterbild

Entstehung eines digitalen Rasterbildes

1. räumliche Abtastung

Übergang von einer räumlichen zu einer diskreten Lichtverteilung durch Geometrie des Aufnahmesensors (z.B. CCD-Chip)

2. zeitliche Abtastung

Steuerung der Zeit, über die die Lichtmessung stattfindet

3. Quantisierung der Pixelwerte

gemessene Lichtwerte werden auf eine endliche Menge von Zahlenwerten abgebildet

1.5 Raster- vs. Vektordaten

Rasterbilder

- regelmässige Matrix (mit diskreten Koordinaten) von Pixelwerten
- typische Formate: BMP, JFIF, TIFF, PNG, PGF usw.

Vektorgrafiken

- geometrische Beschreibung mit kontinuierlichen Koordinaten
- Rasterung (falls notwendig) erfolgt erst bei der Ausgabe
- typische Formate: SVG, DXF usw.

Metaformate

- oft werden Vektorgrafiken und Rasterbilder kombiniert
- typische Formate: CGM, AI, PICT, WMF, PS, EPS, PDF

1.6 Wertebereiche von Pixeln

Grauwertbilder (Intensitätsbilder):

Kanäle	Bit/Pixel	Wertebereich	Anwendungen
1	1	[0...1]	Binärbilder: Dokumente, Illustration, Fax
1	8	[0...255]	Universell: Foto, Scan, Druck
1	12	[0...4095]	Hochwertig: Foto, Scan, Druck
1	14	[0...16383]	Professionell: Foto, Scan, Druck
1	16	[0...65535]	Höchste Qualität: Medizin, Astronomie

Farbbilder:

Kanäle	Bits/Pixel	Wertebereich	Anwendungen
3	24	[0...255] ³	RGB, universell: Foto, Scan, Druck
3	36	[0...4095] ³	RGB, hochwertig: Foto, Scan, Druck
3	42	[0...16383] ³	RGB, professionell: Foto, Scan, Druck
4	32	[0...255] ⁴	CMYK, digitale Druckvorstufe

Spezialbilder:

Kanäle	Bits/Pixel	Wertebereich	Anwendungen
1	16	-32768...32767	Pos./neg. short integers, interne Verarbeitung
1	32	$\pm 3.4 \cdot 10^{38}$	Gleitkomma: Medizin, Astronomie
1	64	$\pm 1.8 \cdot 10^{308}$	Gleitkomma: interne Verarbeitung

1.7 Region of Interest

Grundidee

- nicht immer interessieren alle Teile eines Bild gleichermassen
- der oder die Bildbereiche, welche von Interesse sind, werden als ROIs bezeichnet

Bildformate mit ROI-Unterstützung

- JPEG 2000 (ROIs werden weniger stark komprimiert)
- PGF (nur ROI wird dekomprimiert)

Bestimmung von ROIs

- manuell (Benutzerin wählt den ROI selber aus)
- automatisch
 - Regionen mit Pixel einer bestimmten Intensität/Farbe
 - Regionen mit erhöhter Dichte
 - Regionen mit bestimmten Mustern (Mustererkennung)
 - Regionen die visuell hervorragen (Saliency Detection)

2 Bildspeicherung

2.1 Bildformate

ohne Kompression

- PNM: Portable Bitmap/Grayscale/Color Format
- RAS: Sun Raster Format
- BMP: Windows Bitmap

mit verlustloser Kompression

- GIF: Graphics Interchange Format
- PNG: Portable Network Graphics
- TIFF: Tagged Image File Format

nur mit verlustloser Kompression

- JFIF: JPEG File Interchange Format
- EXIF: Exchangeable Image File Format (Variante von JFIF)

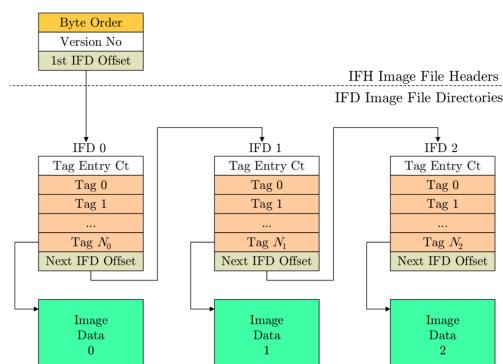
mit verlustloser/verlustbehafteter Kompression

- JPEG-2000
- PGF: Progressive Graphics File

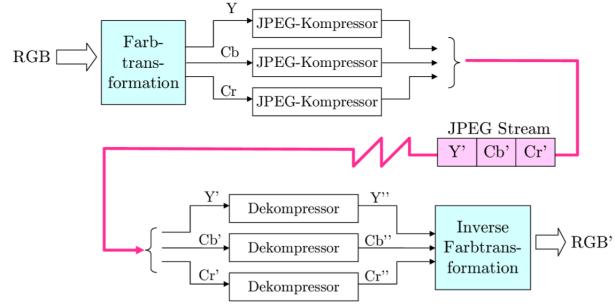
2.2 PGM-Beispiel

```
P2
# oie.pgm
17 7
255
0 13 13 13 13 13 13 13 0 0 0 0 0 0 0 0 0
0 13 0 0 0 0 0 13 0 7 7 0 0 81 81 81 81
0 13 0 7 7 7 0 13 0 7 7 0 0 81 81 0 0 0
0 13 0 7 0 7 0 13 0 7 7 0 0 81 81 81 0
0 13 0 7 7 7 0 13 0 7 7 0 0 81 0 0 0
0 13 0 0 0 0 0 13 0 7 7 0 0 81 81 81 81
0 13 13 13 13 13 13 13 0 0 0 0 0 0 0 0 0
```

2.3 TIFF-Struktur



2.4 JPEG-Prozesskette



2.5 Bildkompression

Das Ziel ist ein Bild auf dem Speichermedium so zu verdichten, dass es möglichst wenig Speicher benötigt, unter der Nebenbedingung, dass die Bildqualität möglichst gut ist.

$$\text{Kompressionsrate } r = \frac{\text{mem}_{\text{orig}}}{\text{mem}_{\text{comp}}}$$

verlustlos (Codierung)

- Entropiecodierung (Huffman, arithmetisch usw.)
- Präcodierung (RLE, LZ usw.)

verlustbehaftet (Datenreduktion)

- Farbreduktion (skalare Quantisierung / Vektorquantisierung)
- Transformation (Fourier / Wavelet) und Quantisierung der Koeffizienten

2.6 Codierung

$$\text{Codierungsredundanz } \Delta R_{\text{cod}} = S - H$$

N_A	Anzahl Symbole im Signal
N_B	gesamte Datenmenge des Signals in Bit
$S = N_B/N_A$	mittlere Anzahl Bits pro Symbol
K	Anzahl verschiedener Symbole
p_i	Symbol-Wahrscheinlichkeit
$H = -\sum_{i=1}^K p_i * ld(p_i)$	Entropie (mittlerer Symbolinformationsgehalt)
$H_{\text{tot}} = n * H$	Totaler Informationsgehalt bei n Symbolen

2.6.1 mittlere Codelänge einer Symbolfolge

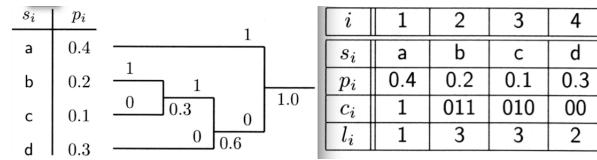
- Alphabet $\{s_1, s_2, \dots, s_K\}$
- mit $p_i = \text{Auftretenswahrscheinlichkeit des Symbols } s_i$

$$l = \sum_{i=1}^K p_i * ||c_i|| \text{ [Bit/Symbol]}$$

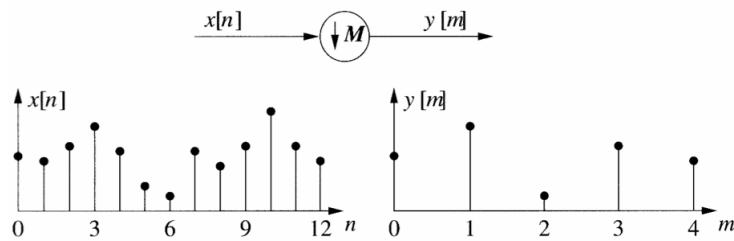
Untere Schranke $H \leq l \leq H + 1$ Obere Schranke

2.6.2 Huffman-Code

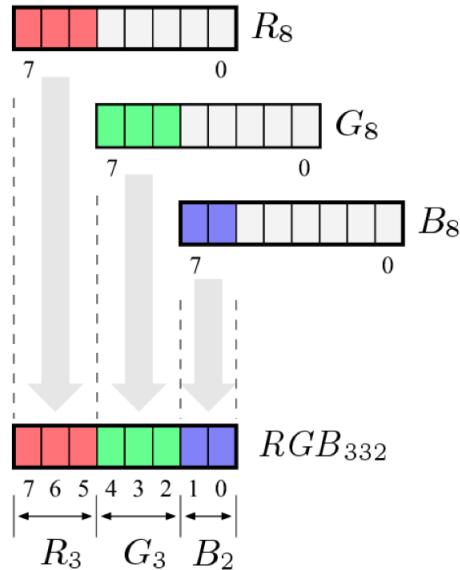
1. Betrachte alle Symbole als Blätter eines Codebaums und trage ihre Wahrscheinlichkeiten ein.
2. Fasse die beiden geringsten Wahrscheinlichkeiten zu einem Knoten zusammen und weise ihre Summe dem Knoten zu.
3. Beschrifte die neuen Äste mit 0 und 1.
4. Fahre bei Schritt 2 fort, so lange die Wurzel mit Wahrscheinlichkeit $p = 1$ noch nicht erreicht worden ist.



2.7 Unterabtastung



2.8 quantisierung



3 Punktoperationen

3.1 Definition

3.1.1 homogen

$$I'(u, v) \leftarrow f(I(u, v))$$

Typische Beispiele:

- Änderungen von Kontrast und Helligkeit
- Invertieren von Bildern
- Quantisieren der Helligkeit (Poster-Effekt)
- Schwellwertbildung
- Gammakorrektur
- Farbtransformation

3.1.2 nicht homogen

$$I'(u, v) \leftarrow g(I(u, v), u, v)$$

Typisches Beispiel:

- selektive Kontrast - oder Helligkeitsanpassung

3.2 Kontrast und Helligkeit

Erhöhung des Kontrasts um 50%:

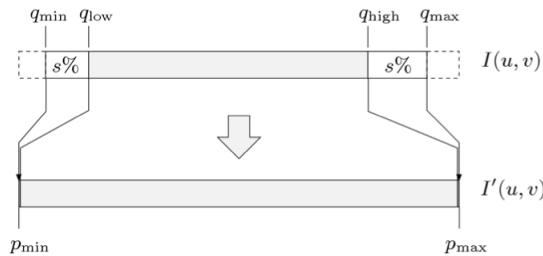
$$I'(u, v) \leftarrow f(I(u, v)) * 1.5$$

Anheben der Helligkeit um 10 Stufen:

$$I'(u, v) \leftarrow f(I(u, v)) + 10$$

3.3 Automatische Kontrastanpassung

$$I'(u, v) \leftarrow \begin{cases} p_{min}, & \text{für } I(u, v) \leq q_{low} \\ p_{min} + (I(u, v) - q_{low}) * \frac{p_{max} - p_{min}}{q_{high} - q_{low}}, & \text{für } q_{low} < I(u, v) < q_{high} \\ p_{max}, & \text{für } I(u, v) \geq q_{high} \end{cases}$$

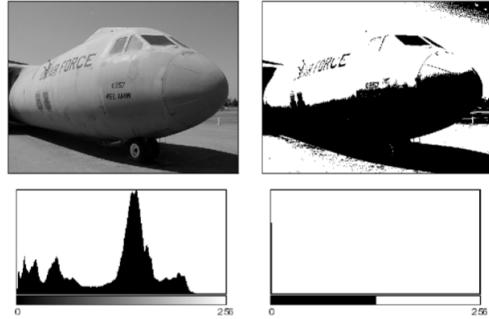


3.4 Schwellwertoperation

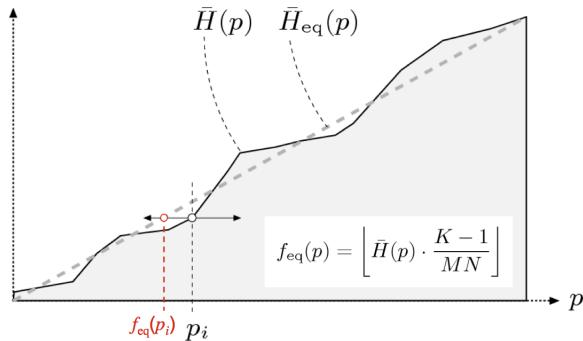
Thresholding

- Reduktion auf zwei Intensitätswerte → Binarisierung
- Spezialfall der Quantisierung (Graustufenreduktion)

$$I'(u, v) \leftarrow f_{th}(I(u, v)) \begin{cases} p_0, & \text{für } I(u, v) < p_{th} \\ p_1, & \text{für } I(u, v) \geq p_{th} \end{cases}$$



3.5 Histogrammausgleich



3.6 Gammakorrektur

einfache Gammakorrektur

$$q = GC(p, \gamma) = \left(\frac{p}{p_{max}} \right)^\gamma * p_{max}$$

Probleme

- Anstieg der Gammafunktion in der Nähe des Nullpunktes
- starke Rauschanfälligkeit wegen der extrem hohen Verstärkung

Modifizierte Gammafunktion

$$s = \frac{\gamma}{x_0(\gamma - 1) + x_0^{1-\gamma}} \quad d = \frac{1}{x_0^\gamma(\gamma - 1) + 1} - 1$$

Gammakorrektur:

$$f_{(\gamma, x_0)}(x) = \begin{cases} s * x, & \text{für } 0 \leq x \leq x_0 \\ (1 + d) * x^\gamma - d, & \text{für } x_0 < x \leq 1 \end{cases}$$

Invers Gammakorrektur:

$$f_{(\gamma, x_0)}(y) = \begin{cases} \frac{y}{s}, & \text{für } y \leq s \cdot x_0 \\ \left(\frac{y+d}{1+d}\right)^{\frac{1}{\gamma}}, & \text{für } y > s \cdot x_0 \end{cases}$$

3.7 Bildqualität

- technische Bildqualität ist ein Mass für die Abweichung zwischen Original (o) und Kopie (k)
- eine subjektive Qualitätsmessung eines Betrachters könnte anders ausfallen

root mean squared error

Bildkanal mit n Pixeln:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (o[i] - k[i])^2}{n}}$$

peak signal-to-noise ratio in dB

Bildkanal mit maximaler Intensität 255:

$$PSNR = 20 * \log_{10} \left(\frac{255}{RMSE} \right)$$

4 Farben

4.1 Farbsysteme

4.1.1 lineare Farbsysteme

RGB:

- Rot, Grün, Blau
- additive Farbmischung

CMY:

- Cyan (Türkis), Magenta (Purpur), Yellow (Gelb)
- subtraktive Farbmischung

CMY:

- Cyan (Türkis), Magenta (Purpur), Yellow (Gelb)
- subtraktive Farbmischung

YUV, YIQ und YC_bC_r :

- Fernseh-Komponentenfarbräume

CIEXYZ:

- CIE Standard-Farbraum

4.1.2 nicht lineare Farbsysteme

CMYK:

- Farbmischung für den 4-Farbendruck: CMY und Schwarz (K)

HSV/HSB/HSI/HLS:

- Hue (Farbton), Saturation (Sättigung), Value/Brightness/Intensity/Luminance (Helligkeit)

CIExy:

- Koordinatensystem des CIE-Farbdigramms
- einfache Umrechnung zwischen CIEXYZ und CIExy
- nicht intuitiv zu verstehen

CIE L*a*b*:

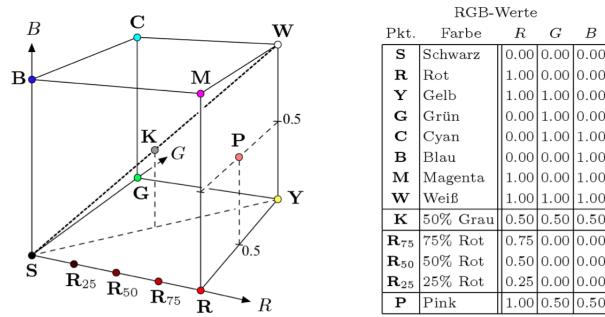
- intuitiv verständlich
- möglichst linear gegenüber dem menschlichen Sehen

sRGB:

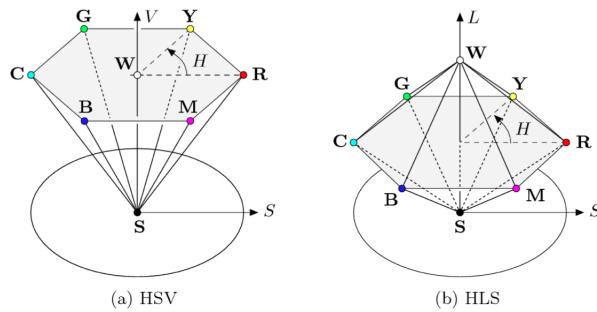
- Standard RGB (präzise definiert, basierend auf CIE xy)
- relativ kleines Gamut (für digitale Anwendungen genügend)

4.2 Farbräume

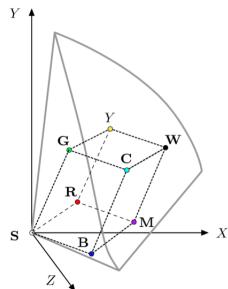
4.2.1 RGB



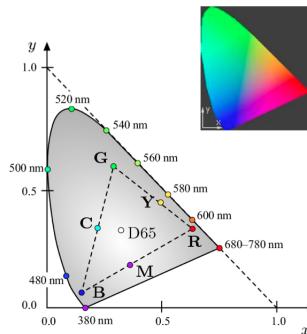
4.2.2 HSV und HLS



4.2.3 CIExyz



4.2.4 CIExy



4.3 Umwandlungen

4.3.1 CMY und CMYK

RGB → CMY:

$$C = 1 - R$$

$$M = 1 - G$$

$$Y = 1 - B$$

CMY → CMYK:

$$C' = C - K$$

$$M' = M - K$$

$$Y' = Y - K$$

$$K' = K$$

4.3.2 CIExy

XYZ → xy:

$$x = \frac{X}{X + Y + Z} \quad y = \frac{Y}{X + Y + Z}$$

xy → XYZ:

$$X = \frac{x}{y} \quad Y = 1 \quad Z = \frac{z}{y} = \frac{1 - x - y}{y}$$

4.3.3 Transformation sRGB → XYZ

Nichtlineare sRGB-Komponenten:

$$R = f_2(R') \quad G = f_2(G') \quad B = f_2(B')$$

Inverse modifizierte Gammakorrektur

Lineare Transformation von RGB nach CIE XYZ:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Rücktransformation:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

4.4 Weisspunkte

D50

- Farbtemperatur von 5000 Kelvin
- ähnlich dem direkten Sonnenlicht
- Referenzbeleuchtung für die Betrachtung von reflektierenden Bildern wie z.B. von Druckern

D65

- Farbtemperatur von 6500 Kelvin
- durchschnittliche indirekte Tageslichtbeleuchtung
- Normweisslicht für emittierende Wiedergabegeräte z.B. von Bildschirme

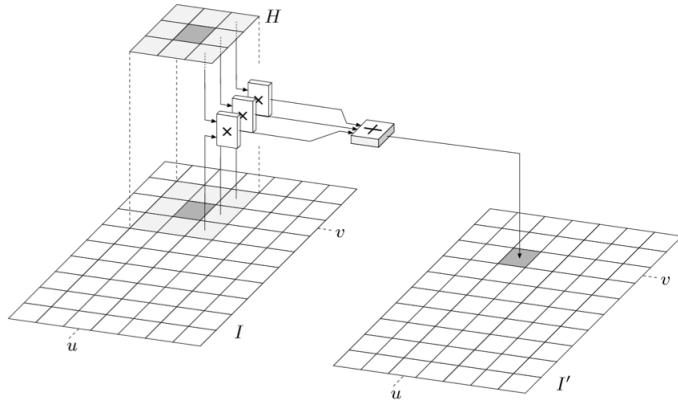
Chromatische Adaptierung:

$$\begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix} = M_{65|50} \cdot \begin{pmatrix} X_{50} \\ Y_{50} \\ Z_{50} \end{pmatrix} = \begin{pmatrix} 0.955556 & -0.023049 & 0.063197 \\ -0.028302 & 1.009944 & 0.021018 \\ 0.012305 & -0.020494 & 1.330084 \end{pmatrix} \cdot \begin{pmatrix} X_{50} \\ Y_{50} \\ Z_{50} \end{pmatrix}$$

$$\begin{pmatrix} X_{50} \\ Y_{50} \\ Z_{50} \end{pmatrix} = M_{50|65} \cdot \begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix} = \begin{pmatrix} 1.047840 & 0.0228961 & -0.0501482 \\ 0.0295561 & 0.990482 & -0.0170559 \\ -0.00923843 & 0.0150496 & 0.752033 \end{pmatrix} \cdot \begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix}$$

5 Filter

5.1 Anwendung



$$I'(u, v) \leftarrow \sum_{(i,j) \in \mathbb{R}} I(u+i, v+j) * H(i, j)$$

5.2 Gauss-Filter

diskrete, zweidimensionale Gauss-Funktion:

$$G_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \quad G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Glättungsfilter (Tiefpassfilter):

- weil alle Filterkoeffizienten positiv sind
- annähernd isotrop ab Filtergrößen von 5 x 5 (nach allen Richtungen hin gleichförmig arbeiten)
- gutmütiges Frequenzverhalten (stetig, differenzierbar)

5.3 Laplace-Filter

Interpretation als Differenz von zwei Summen:

$$I'(u, v) = \sum_{(i,j) \in \mathbb{R}^+} I(u+i, v+j) * |H(i, j)| - \sum_{(i,j) \in \mathbb{R}^-} I(u+i, v+j) * |H(i, j)|$$

$\mathbb{R}^+/\mathbb{R}^-$ Teil der Filterregion mit positiven/negativen Koeffizienten

Verstärken von Kanten und Konturen:

- örtliche Intensitätsunterschiede werden verstärkt
- richtungsunabhängige Detektion von Kanten
- Hochpassfilter

5.4 Faltung

5.4.1 diskrete Faltung

$$\begin{aligned}
 I'(u, v) &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u - i, v - j) \cdot H(i, j) \\
 I' &= I * H \\
 I'(u, v) &= \sum_{(i,j) \in \mathbb{R}} I(u - i, v - j) \cdot H(i, j) \\
 &= \sum_{(i,j) \in \mathbb{R}} I(u + i, v + j) \cdot H(-i, -j)
 \end{aligned}$$

5.4.2 Eigenschaften der Faltung

Kommutativität:

$$I * H = H * I$$

Linearität:

$$\begin{aligned}
 (a \cdot I) * H &= I * (a \cdot H) = a \cdot (I * H) \\
 (I_1 + I_2) * H &= (I_1 * H) + (I_2 * H)
 \end{aligned}$$

Assoziativität:

$$A * (B * C) = (A * B) * C$$

5.5 Separierbarkeit

$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$

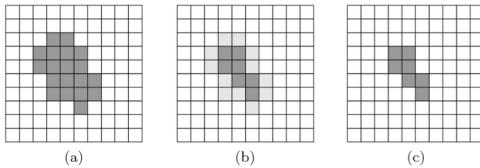
$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$H_x^P = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad H_y^P = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

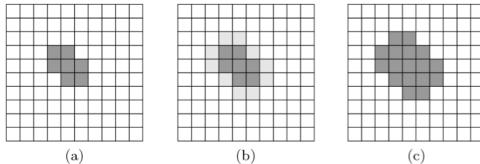
6 Morphologie

6.1 Schrumpfen und Wachsen

Schrumpfen



Wachsen lassen



Art der Nachbarschaft

$$\begin{array}{|c|c|c|} \hline & & \\ \hline & N_2 & \\ \hline N_1 & \times & N_4 \\ \hline & N_3 & \\ \hline \end{array}$$

N_2	N_3	N_4
N_1	\times	N_5
N_8	N_7	N_6

6.2 Grundoperationen

Strukturelement: binäre Matrix mit Hot-Spot als Ursprung des Koordinatensystems (analog zu einem Filter mit binären Koeffizienten)

Strukturelement als Punktmenge:

$$P_H = \{(i, j) | H(i, j)\}1\}$$

Bild als Punktmenge:

$$P_I = \{(u, v) | I(u, v)\}1\}$$

Erosion (Schrumpfen):

$$I \ominus H = \{(x, y) | (x + i, y + j) \in P_i, \forall (i, j) \in PH\}$$

$$\begin{array}{c} \begin{array}{|c|c|c|c|} \hline & 0 & 1 & 2 & 3 \\ \hline 0 & & \bullet & \bullet & \\ \hline 1 & & & & \\ \hline 2 & & & \bullet & \\ \hline 3 & & & & \\ \hline \end{array} & \oplus & \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline 0 & \bullet & \bullet \\ \hline 1 & & \\ \hline \end{array} & = & \begin{array}{|c|c|c|c|} \hline & 0 & 1 & 2 & 3 \\ \hline 0 & & \bullet & & \\ \hline 1 & & & \bullet & \\ \hline 2 & & & & \\ \hline 3 & & & & \\ \hline \end{array} \\ I & & H & & I \oplus H \end{array}$$

Dilation (Wachsen lassen):

$$I \oplus H = \{(x, y) = (u + i, v + j) \mid (u, v) \in P_i, \forall (i, j) \in PH\}$$

$$\begin{array}{c} \text{Table } I \\ \oplus \\ \text{Table } H \\ = \\ \text{Table } I \oplus H \end{array}$$

6.3 Zusammengesetzte Operationen

Opening

zuerst schrumpfen, dann wachsen lassen:

$$I \circ H = (I \ominus H) \oplus H$$

- sehr kleine Vordergrundstrukturen werden eliminiert

- glättet Kanten von grösseren Elementen
- Elemente mit kleinem Berührungs punkt werden getrennt

Closing

zuerst wachsen lassen, dann schrumpfen:

$$I \bullet H = (I \oplus H) \ominus H$$

- füllt kleine Löcher
- glättet Kanten von grösseren Elementen
- Elemente mit kleinem Berührungs punkt werden stärker verbunden

6.4 Grauwert-Morphologie

Grauwert-Dilation

$$(I \oplus H)(u, v) = \max_{(i,j) \in H} \{I(u+i, v+j) + H(i, j)\}$$

$$\begin{array}{c}
 I \\
 \begin{array}{|c|c|c|c|} \hline
 6 & 7 & 3 & 4 \\ \hline
 5 & 6 & 6 & 8 \\ \hline
 6 & 4 & 5 & 2 \\ \hline
 6 & 4 & 2 & 3 \\ \hline
 \end{array}
 \end{array}
 \oplus
 \begin{array}{c}
 H \\
 \begin{array}{|c|c|c|} \hline
 1 & 1 & 1 \\ \hline
 1 & 2 & 1 \\ \hline
 1 & 1 & 1 \\ \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 I \oplus H \\
 \begin{array}{|c|c|c|} \hline
 & & \\ \hline
 & 8 & 9 \\ \hline
 & 7 & 9 \\ \hline
 & & \\ \hline
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 I + H \\
 \begin{array}{|c|c|c|} \hline
 7 & 8 & 4 \\ \hline
 6 & 8 & 7 \\ \hline
 7 & 5 & 6 \\ \hline
 \end{array}
 \end{array}
 \text{ max}$$

Grauwert-Erosion

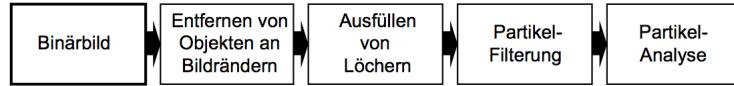
$$(I \ominus H)(u, v) = \min_{(i,j) \in H} \{I(u+i, v+j) - H(i, j)\}$$

$$\begin{array}{c}
 I \\
 \begin{array}{|c|c|c|c|} \hline
 6 & 7 & 3 & 4 \\ \hline
 5 & 6 & 6 & 8 \\ \hline
 6 & 4 & 5 & 2 \\ \hline
 6 & 4 & 2 & 3 \\ \hline
 \end{array}
 \end{array}
 \ominus
 \begin{array}{c}
 H \\
 \begin{array}{|c|c|c|} \hline
 1 & 1 & 1 \\ \hline
 1 & 2 & 1 \\ \hline
 1 & 1 & 1 \\ \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 I \ominus H \\
 \begin{array}{|c|c|c|} \hline
 & & \\ \hline
 & 2 & 1 \\ \hline
 & 1 & 1 \\ \hline
 & & \\ \hline
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 I - H \\
 \begin{array}{|c|c|c|} \hline
 5 & 6 & 2 \\ \hline
 4 & 4 & 5 \\ \hline
 5 & 3 & 4 \\ \hline
 \end{array}
 \end{array}
 \text{ min}$$

7 Binärbildanalyse

7.1 Prozesskette



7.2 Flood filling

```

1: REGIONLABELING( $I$ )
2: Initialize  $m \leftarrow 2$  (the value of the next label to be assigned).
3: Iterate over all image coordinates  $(u, v)$ .
4:   if  $I(u, v) = 1$  then
5:     FLOODFILL( $I, u, v, m$ )
6:   Increment  $m$ .
7: return
  
```

- **3 Varianten**

- **rekursiv**

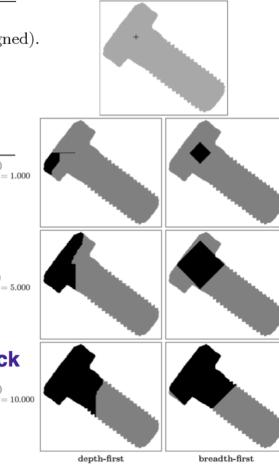
- meist untauglich, weil die Stackgrösse nicht ausreicht

- **iterativ mit Stack: depth-first**

- wie rekursiv, aber mit eigenem Stack

- **iterativ mit Queue: breadth-first**

- oft am speichereffizientesten



7.3 Rekursives Flood Filling

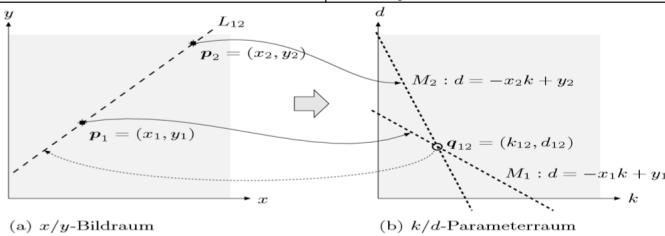
```

FloodFill( $I, u, v, label$ )
if  $(u, v)$  is within image boundaries  $\wedge I(u, v) = 1$  then
   $I(u, v) := label$ 
  FloodFill( $I, u + 1, v, label$ )
  FloodFill( $I, u, v + 1, label$ )
  FloodFill( $I, u, v - 1, label$ )
  FloodFill( $I, u - 1, v, label$ )
endif
end
  
```

7.4 Hough-Transformation

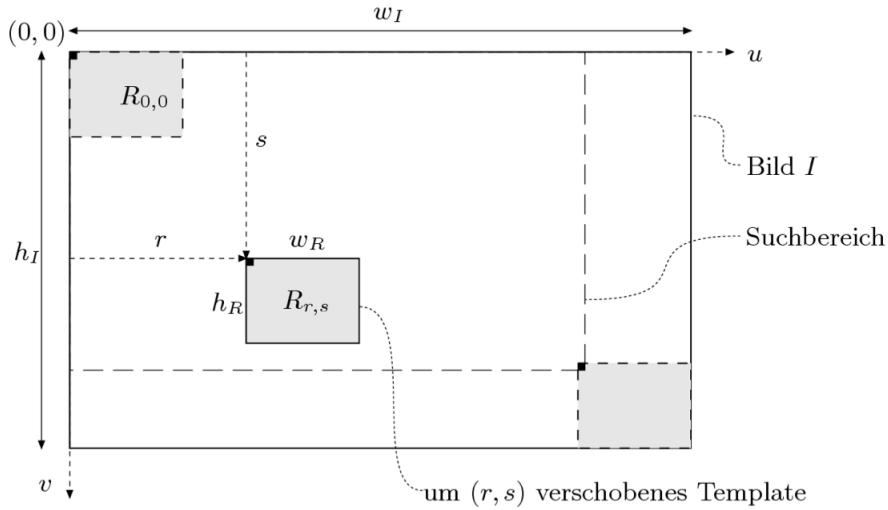
Die Grundidee ist, dass wir durch alle Kantenpixen in einem binären Kantenbild gehen. Ein solches Kantenpixel entspricht einem Punkt im Bildraum. Durch einen solchen Punkt im Bildraum können unendliche viele verschiedenen Geraden im Bildraum laufen. Diese können durch eine einzige Gerade im Parameterraum repräsentiert werden. Liegen mehrere Punkte im Bildraum auf derselben Geraden im Bildraum, so schneiden sich die entsprechenden Geraden im Parameterraum in einem einzigen Punkt.

Bildraum (x, y)		Parameterraum (k, d)	
Punkt	$p_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$	Gerade
Gerade	$L_j : y = k_j x + d_j$	$q_j = (k_j, d_j)$	Punkt



8 Mustererkennung (Pattern Matching)

8.1 Definitionen



8.2 Ähnlichkeitsmasse

Summe der Differenzbeträge:

$$d_A(r, s) = \sum_{(i,j) \in \mathbb{R}} |I(r+i, s+j) - R(i, j)|$$

Maximaler Differenzbetrag

$$d_M(r, s) = \max_{(i,j) \in \mathbb{R}} |I(r+i, s+j) - R(i, j)|$$

Summe der quadratischen Differenzbeträge (euklidische Distanz):

$$d_E(r, s) = \sqrt{\sum_{(i,j) \in \mathbb{R}} (I(r+i, s+j) - R(i, j))^2}$$

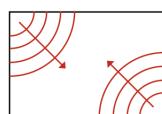
globale lineare Kreuzkorrelation:

$$(I \otimes R)(r, s) = \sum_{(i,j) \in \mathbb{R}} I(r+i, s+j) \cdot R(i, j)$$

8.3 Chamfer-Algorithmus

Idee

- wellenförmige Ausbreitung der Distanzwerte



Initialisierung

- $D(u, v) := (\#u, v) = 1 ? 0 : \infty$

Algorithmus

- erste Welle breitet sich von links oben nach rechts unten aus

if $D(u, v) > 0$ then $D(u, v) := \min(D(u+i, v+j) + M^L(i, j))$

$$M^L = \begin{bmatrix} m_2^L & m_3^L & m_4^L \\ m_1^L & \times & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

- zweite Welle breitet sich von rechts unten nach links oben aus

if $D(u, v) > 0$ then $D(u, v) := \min(D(u+i, v+j) + M^R(i, j))$

$$M^R = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & 0 & m_1^R \\ m_4^R & m_3^R & m_2^R \end{bmatrix}$$

$$\text{Ähnlichkeitsmasse } Q(r, s) = \frac{1}{K} \sum_{(i,j) \in FG(R)} D(r+i, s+j)$$

```

1: CHAMFERMATCH ( $I, R$ )                                ▷ binary image  $I(u, v)$  of size  $w_I \times h_I$ 
   ▷ binary template  $R(u, v)$  of size  $w_R \times h_R$ 
2: STEP 1 – INITIALIZE:
3:  $D \leftarrow \text{DISTANCETRANSFORM}(I)$                   ▷ Alg. 17.1
4:  $K \leftarrow$  number of foreground pixels in  $R$ 
5: Set up a match map  $Q$  of size  $(w_I - w_R + 1) \times (h_I - h_R + 1)$ 
6: STEP 2 – COMPUTE MATCH FUNCTION:
7: for  $r \leftarrow 0 \dots (w_I - w_R)$  do                   ▷ set origin of model to  $(r, s)$ 
8:   for  $s \leftarrow 0 \dots (h_I - h_R)$  do
9:     EVALUATE MATCH FOR TEMPLATE POSITIONED AT  $(r, s)$ :
10:     $q \leftarrow 0$ 
11:    for  $i \leftarrow 0 \dots (w_R - 1)$  do
12:      for  $j \leftarrow 0 \dots (h_R - 1)$  do
13:        if  $R(i, j) = 1$  then                            ▷ foreground pixel in model
14:           $q \leftarrow q + D(r+i, s+j)$ 
15:     $Q(r, s) \leftarrow q/K$ 
16: return  $Q$ 

```

9 Fourier-Transformation

9.1 Frequenz, Amplitude, Phase

$$\begin{aligned}
 \text{Beispiel: } & a \cdot \sin(\omega \cdot x - \varphi) \\
 \text{Kreisfrequenz: } & \omega = 2 \cdot \pi \cdot f \\
 \text{Frequenz: } & f = \frac{1}{T} = \frac{\omega}{2 \cdot \pi} \\
 \text{Periodenlänge: } & T \\
 \text{Amplitudde: } & a \\
 \text{Phase: } & \varphi
 \end{aligned}$$

9.2 Fourierreihe

Jede periodische Funktion $g(x)$ mit einer Grundfrequenz ω_0 kann als unendliche Summe von harmonischen Schwingungen dargestellt werden:

$$g(x) = \sum_{k=0}^{\infty} [A_k \cdot \cos(k \cdot \omega_0 \cdot x) + B_k \cdot \sin(k \cdot \omega_0 \cdot x)]$$

Fourieranalyse: Berechnung der Fourerkoeffizienten (A_k, B_k) aus einer gegebenen Funktion $g(x)$.

9.3 Fourierintegral und -spektrum

Eine nicht periodische Funktion $g(x)$ kann als Summe von unendlich vielen Sinus- und Kosinusschwingungen dargestellt werden. Das bedarf nicht nur Vielfache von der Grundfrequenz (\rightarrow Fourierintegral) sondern auch unendlich viele dicht aneinander liegende Frequenzen.

$$g(x) = \int_0^{\infty} A_{\omega} \cdot \cos(\omega \cdot x) + B_{\omega} \cdot \sin(\omega \cdot x) d\omega$$

Bestimmung des Fourierspektrums (Fourerkoeffizientenfunktionen):

$$\begin{aligned}
 A_{\omega} = A(\omega) &= \frac{1}{\pi} \cdot \int_{-\infty}^{\infty} g(x) \cdot \cos(\omega \cdot x) dx \\
 B_{\omega} = B(\omega) &= \frac{1}{\pi} \cdot \int_{-\infty}^{\infty} g(x) \cdot \sin(\omega \cdot x) dx
 \end{aligned}$$

9.4 Fouriertransformation

Übergang von Fourierintegral zu Fouriertransformation:

$$\begin{aligned}
 G(\omega) &= \sqrt{\frac{\pi}{2}} \cdot [A(\omega) - i \cdot B(\omega)] \\
 &= \sqrt{\frac{\pi}{2}} \cdot \left[\frac{1}{\pi} \cdot \int_{-\infty}^{\infty} g(x) \cdot \cos(\omega \cdot x) dx - i \cdot \frac{1}{\pi} \cdot \int_{-\infty}^{\infty} g(x) \cdot \sin(\omega \cdot x) dx \right] \\
 &= \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} g(x) \cdot [\cos(\omega \cdot x) - i \cdot \sin(\omega \cdot x)] dx
 \end{aligned}$$

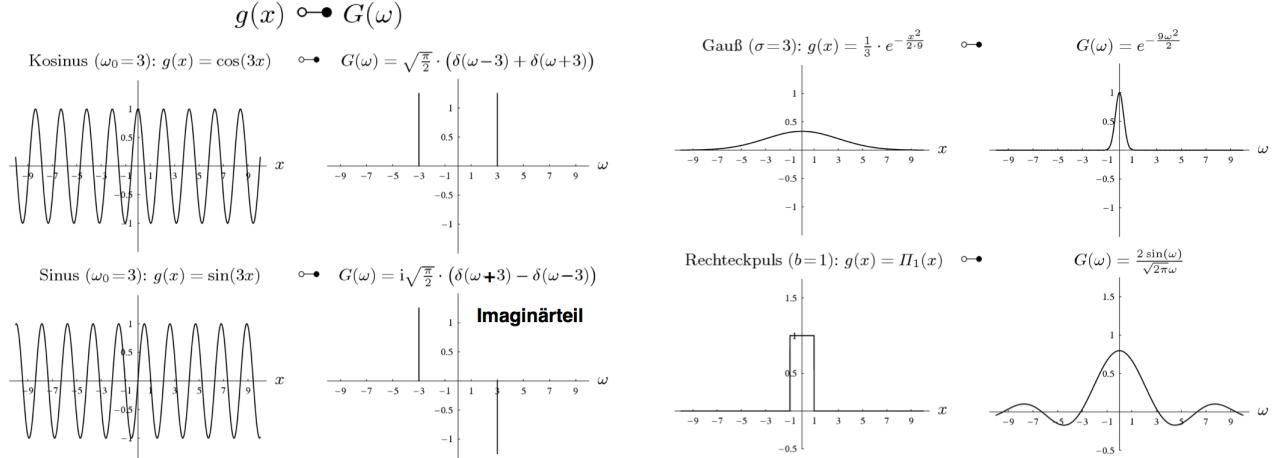
Vorwärtstransformation

$$G(\omega) = \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} g(x) \cdot e^{-i\omega x} dx$$

Rücktransformation

$$G(\omega) = \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} g(x) \cdot e^{i\omega x} dx$$

9.5 Fourier-Transformationspaare

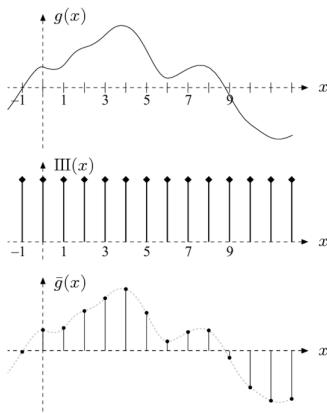


9.6 Beispiel: Rücktransformation

$$\begin{aligned}
 G(\omega) &= \sqrt{\frac{\pi}{2}}(\delta(\omega - 3) + \delta(\omega + 3)) \\
 f(x) &= \frac{\int_{-\infty}^{\infty} G(\omega) e^{j\omega x} d\omega}{\sqrt{2\pi}} \\
 &= \frac{\int_{-\infty}^{\infty} \delta(\omega - 3) e^{j\omega x} d\omega + \int_{-\infty}^{\infty} \delta(\omega + 3) e^{j\omega x} d\omega}{2} \\
 &= \frac{e^{j3x} + e^{-j3x}}{2} \quad |e^{ix}| = \cos(x) + j \sin(x) \\
 &= \frac{\cos(3x) + j \sin(3x) + \cos(3x) - j \sin(3x)}{2} \\
 &= \cos(3x)
 \end{aligned}$$

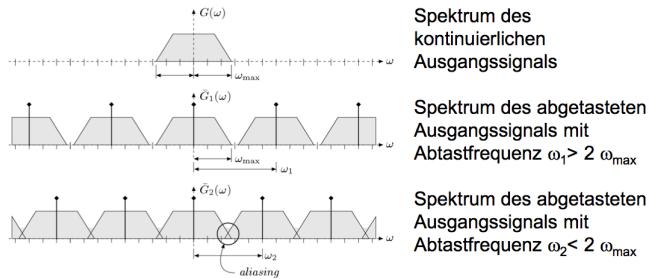
9.7 FT von diskreten Signalen

- Abtastung (Sampling)**
- Abtastung = Multiplikation mit Kammfunktion
 - durch Abtastung wird aus einer kontinuierlichen Ausgangsfunktion $g(x)$ eine diskrete Funktion
- Auswirkungen**
- Diskretisierung im Ortsraum führt zu Periodizität im Frequenzraum (Fourierspektrum)
 - Invers zu: Periodizität im Ortsraum führt zu diskretem Fourierspektrum (\rightarrow Fourierreihe)



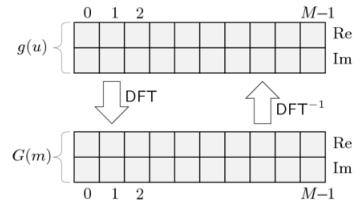
9.8 Aliasing und Abtasttheorem

- Diskretisierung im Ortsraum \rightarrow Periodizität im Frequenzraum
- falls die sich wiederholenden Spektralkomponenten im Frequenzraum nicht überschneiden, so ist eine verlustlose Rücktransformation möglich
- maximal zulässige Signalfrequenz ω_{max} ist von der Abtastfrequenz ω_s abhängig



9.9 Diskrete Fouriertransformation

$$\begin{aligned}
 \text{Periodenlänge } t_0: & M \text{ Abtastwerte im Abstand } t_s \\
 \text{Frequenz } f_0 = & \frac{1}{M \cdot t_s} \\
 \text{Abtastfrequenz } f_s = & \frac{1}{t_s} = M \cdot f_0 \\
 \text{Wellenzahl } m: & 0 \leq m < M \\
 \text{Kreisfrequenz } \omega & = m \cdot \omega_0 = 2\pi \cdot m \cdot f_0
 \end{aligned}$$



Leistungsspektrum

$$|G(m)| = \sqrt{G_{Re}^2(m) + G_{Im}^2(m)}$$

Phasenspektrum

$$Pha(m) = \arctan \left(\frac{G_{Im}(m)}{G_{Re}(m)} \right)$$

Implementation

```
Complex[] DFT(Complex[] g, boolean forward) {
    int M = g.length;
    double s = 1 / Math.sqrt(M); //common scale factor
    Complex[] G = new Complex[M];
    for (int m = 0; m < M; m++) {
        double sumRe = 0;
        double sumIm = 0;
        double phim = 2 * Math.PI * m / M;
        for (int u = 0; u < M; u++) {
            double gRe = g[u].re;
            double gIm = g[u].im;
            double cosw = Math.cos(phim * u);
            double sinw = Math.sin(phim * u);
            if (!forward) // inverse transform
                sinw = -sinw;
            //complex mult: (gRe + i gIm) * (cosw + i sinw)
            sumRe += gRe * cosw - gIm * sinw;
            sumIm += gIm * cosw + gRe * sinw;
        }
        G[m] = new Complex(s * sumRe, s * sumIm);
    }
    return G;
}
```

9.10 FFT und DCT

Zeitkomplexität der DFT

- zwei verschachtelte for-Schleifen von 0 bis M
- $O(M^2)$

Fast Fourier Transform (FFT)

- z.B. Algorithmus von Cooley und Tukey
- Optimierung auf Signallängen von $M = 2^k$
- Reduktion der Zeitkomplexität auf $O(M \cdot \log M)$

Discrete Cosine Transform (DCT)

- nur für reelle Signale geeignet
- Spektrum ist auch reell
- Transformation verwendet nur Kosinusfunktionen

9.11 DFT in 2D

Vorwärtstransformation

$$\begin{aligned} G(m, n) &= \frac{1}{\sqrt{MN}} \cdot \sum_{u=0}^{M-1} g(u, v) \cdot e^{-i2\pi \frac{m \cdot u}{M}} \cdot e^{-i2\pi \frac{n \cdot v}{N}} \\ &= \frac{1}{\sqrt{MN}} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} g(u, v) \cdot e^{-i2\pi \left(\frac{m \cdot u}{M} + \frac{n \cdot v}{N} \right)} \end{aligned}$$

Rücktransformation

$$\begin{aligned} g(u, v) &= \frac{1}{\sqrt{MN}} \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g(u, v) \cdot e^{i2\pi \frac{m \cdot u}{M}} \cdot e^{i2\pi \frac{n \cdot v}{N}} \\ &= \frac{1}{\sqrt{MN}} \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g(u, v) \cdot e^{i2\pi \left(\frac{m \cdot u}{M} + \frac{n \cdot v}{N} \right)} \end{aligned}$$

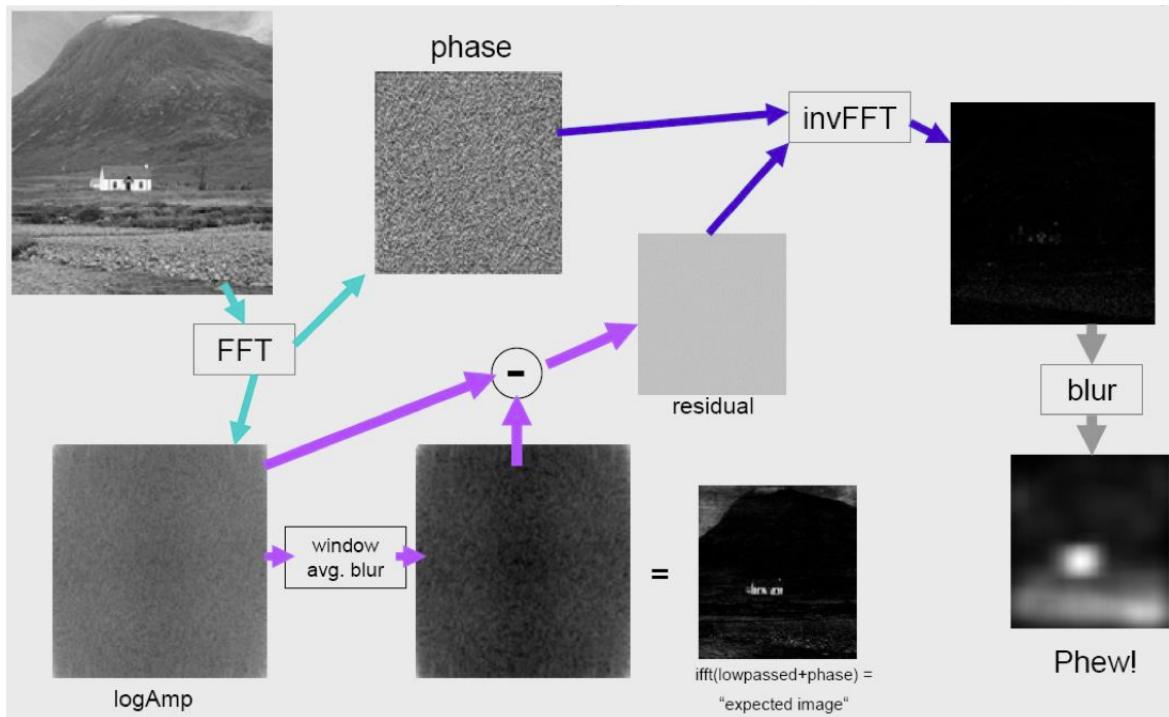
9.12 Implementierung der 2D-DFT

$$G(m, n) = \frac{1}{\sqrt{N}} \cdot \sum_{v=0}^{N-1} \underbrace{\left[\frac{1}{\sqrt{M}} \sum_{u=0}^{M-1} g(u, v) \cdot e^{-i2\pi \frac{m \cdot u}{M}} \right]}_{\text{1-dim. DFT der Zeile } g(\cdot, u)} \cdot e^{-i2\pi \frac{n \cdot v}{N}}$$

eindimensionale DFT genügt:

- zuerst alle Zeilen eines Bildes mit der DFT transformieren
- dann alle transformierten Zeilen spaltenweise mit DFT transformieren

9.13 SD Algorithmus



10 Kanten und Ecken

10.1 Übersicht

Gradientbasierte Kantendetektion

- Faltungsoperationen basierend auf der 1. Ableitung
- typische Vertreter: Prewitt, Sobel

Verfahren basierend auf der 2. Ableitung

- verbesserte Kantenlokalisierung gegenüber der Gradientverfahren
- typische Vertreter: Laplacian of Gaussian

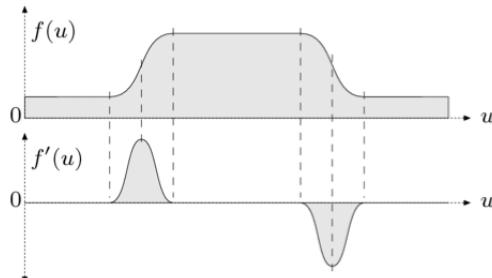
Canny Edge Detector

- basierend auf Gradientverfahren unter Ausnutzung der 2. Ableitung für Kantenlokalisierung
- gilt als eins der besten Verfahren

Morphologische Verfahren

- Ermittlung inneren oder äusseren Kontur

10.2 Gradient



Diskrete Approximation der Ableitung:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 * (f(u+1) - f(u-1))$$

Partielle Ableitungen:

$$\frac{df}{du}(u, v) \quad \frac{df}{dv}(u, v)$$

Gradient:

$$\nabla I(u, v) = \begin{bmatrix} \frac{df}{du}(u, v) \\ \frac{df}{dv}(u, v) \end{bmatrix} \quad |\nabla I(u, v)| = \sqrt{\left(\frac{df}{du}(u, v)\right)^2 + \left(\frac{df}{dv}(u, v)\right)^2}$$

10.2.1 Ableitungsfilter

Die Diskrete Approximation der ersten Ableitung lässt sich einfach mit einer diskreten Faltung realisieren:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 * (f(u+1) - f(u-1)) = f(u) * \left[-\frac{1}{2} \quad 0 \quad \frac{1}{2} \right]$$

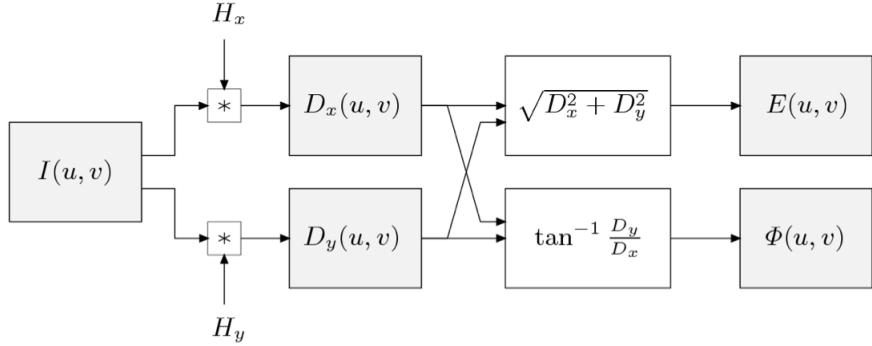
Resultierende Ableitungsfilter:

$$H_x^D = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$H_y^D = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

10.2.2 Kantendetektierung

Mit den Gradientenfiltern H_x und H_y werden zwei Gradientenbilder D_x und D_y erzeugt. Anschliessend wird die Kantenstärke E und die Kantenrichtung Φ pro Bildposition (u, v) berechnet:

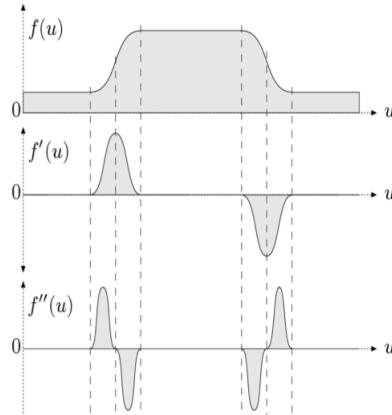


10.2.3 1. und 2. Ableitung

Problematik der Gradientenverfahren:

- detektierte Kanten sind so breit wie die Steigerungsverläufe
- schlechte Kantenlokalisierung

Mit der Verwendung der zweiten Ableitung können die Übergänge zwischen verschiedenen Steigerungsverläufen detektiert werden:



10.3 Laplacian of Gaussian

Grundidee:

- Bild zuerst mit Gauss-Filter glätten (kleine scharfe Störungen verwischen)
- Zweite Ableitung des geglätteten Bildes bestimmen
- Pder: Bild mit 2. Ableitung von Gauss falten

Gauss-Funktion:

$$G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

1. partielle Ableitung:

$$\frac{\partial G(x, y)}{\partial x} = -\frac{x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = -\frac{x}{\sigma^2} G(x, y)$$

2. partielle Ableitung:

$$\frac{\partial^2 G(x, y)}{\partial^2 x} = -\frac{x^2 - \sigma^2}{\sigma^4} G(x, y)$$

LoG:

$$-\nabla^2 G(x, y) = -\frac{\partial^2 G(x, y)}{\partial^2 x} - \frac{\partial^2 G(x, y)}{\partial^2 y}$$

10.3.1 Kantenschärfung

Verwendung des Laplace-Operators:

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f(x, y)}{\partial^2 x} + \frac{\partial^2 f(x, y)}{\partial^2 y}$$

Approximation der 2. Ableitungen:

$$\frac{\partial^2 f}{\partial^2 x} \approx H_x^L = [1 \quad -2 \quad 1] \quad \frac{\partial^2 f}{\partial^2 y} \approx H_y^L = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

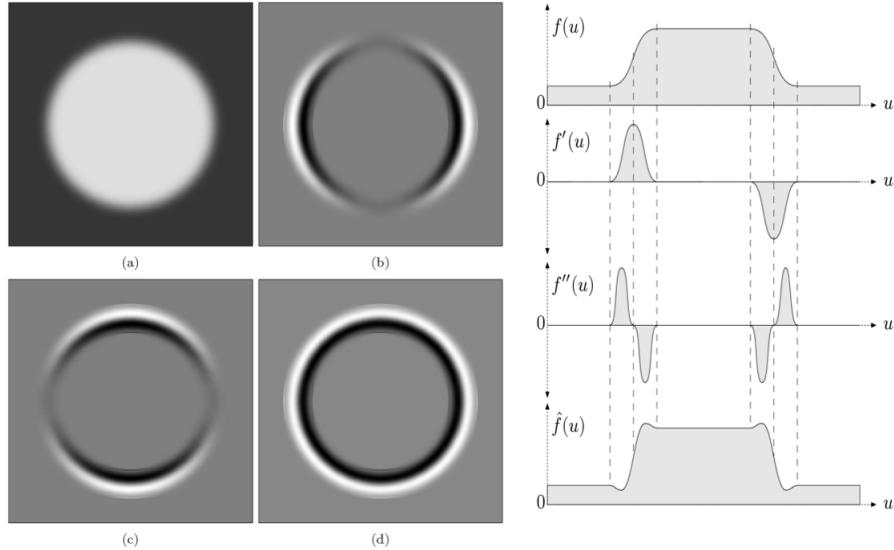
Laplace-Filter:

$$H^L = H_x^L + H_y^L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Kantenschärfung:

$$I' \leftarrow I - \omega \cdot (H^L * I)$$

10.3.2 Anwendung des Laplace-Filters



10.4 Eckpunkte

Eckpunkte weisen Intensitätsänderung in x- und y-Richtung auf.

10.4.1 Anforderungen

- lokale Einzigartigkeit
- invariant gegenüber linearen Abbildungen
- unempfindlich gegenüber Rauschen

10.4.2 Strukturmatrix

Autokorrelation:

$$cor(u, v) = \sum_{i,j \in \mathbb{R}} [I(u+i, v+j) - I(u+i+\Delta u, v+j+\Delta v)]^2$$

Taylor-Approximation des Verschobenen Bildes pro Bildpunkt berechnen:

$$A(u, v) = I_x^2(u, v) \quad B(u, v) = I_y^2(u, v) \quad c(u, v) = I_x^2(u, v) \cdot I_y^2(u, v)$$

Konzeptionell zusammengefasst zur Strukturmatrix:

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$$

Strukturmatrix glätten:

$$\bar{M} = \begin{pmatrix} A * H^{G,\sigma} & C * H^{G,\sigma} \\ C * H^{G,\sigma} & B * H^{G,\sigma} \end{pmatrix} = \begin{pmatrix} \bar{A} & \bar{C} \\ \bar{C} & \bar{B} \end{pmatrix}$$

10.4.3 Eigenwerte und Eigenvektoren

- Ein Vektor $x \neq 0$ heisst Eigenvektor einer quadratischen Matrix A zum reellen Eigenwert λ , falls $Ax = \lambda x$. $(A - \lambda E)x = 0$, wobei E die Einheitsmatrix ist.
- Die Eigenwerte der Matrix A sind alle reellen λ für die ein Eigenvektor existiert.
- Die Matrix A ist eine lineare Abbildung, welche den Eigenvektor x nicht rotiert, sondern nur um den Faktor λ skaliert.
- Wenn A eine Diagonalmatrix ist, dann sind die Diagonalelemente die Eigenwerte und die Einheitsvektoren sind die Eigenvektoren.

10.4.4 Ähnliche Matrizen

- 2 quadratische Matrizen A und D heissen ähnlich, falls es eine invertierbare Matrix S gibt, mit $A = S^{-1}DS$
- eine quadratische Matrix A heisst symmetrisch falls $A^T = A$
- Jede symmetrische Matrix A ist ähnlich zu einer Diagonalmatrix D . Die Diagonalelemente der Diagonalmatrix D sind die Eigenwerte von A .
- Die Transformationsmatrix S wird aus den orthonormierten Eigenvektoren x_i gebildet, $S = [x_1, x_2, \dots, x_n]$

10.4.5 Diagonalisierte Strukturmatrix

Ähnliche Matrix zur geglätteten Strukturmatrix:

- die Diagonalelemente sind gleich den Eigenwerten λ

$$\bar{M}' = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

Generelle Interpretation:

- Matrix entspricht einer linearen Abbildung basierend auf einem Koordinatensystem
- durch geschickte Variation des Koordinatensystems kann die Abbildungsmatrix vereinfacht (diagonalisiert) werden
- die Eigenvektoren entsprechen den Einheitsvektoren des neuen Koordinatensystems

10.4.6 Spezifische Interpretation

Eigenwerte:

$$\lambda_{1,2} = \frac{\text{trace}(\bar{M})}{2} \pm \sqrt{\left(\frac{\text{trace}(\bar{M})}{2}\right)^2 - \det(\bar{M})} = \frac{1}{2} \left(\bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2} \right)$$

- beide sind null in flachen Bildregionen
- bei einer Kante in beliebiger Richtung ist der kleinere der beiden Eigenwerte fast null
- nur bei Eckpunkten sind beide Eigenwerte gross

Eigenvektoren:

- geben die Richtung der Kanten an

10.4.7 Corner Response Function (CRF)

Die Differenz der Eigenwerte kann als Gütemass für einen Eckpunkt gewertet werden: je kleiner die Differenz, desto eher ist es einen Eckpunkt:

$$\lambda_1 - \lambda_2 = 2 \cdot \sqrt{\frac{1}{4} \cdot (\text{trace}(\bar{M})^2 - \det(\bar{M}))}$$

CRF:

$$Q(u, v) = \det(\bar{M}) - \alpha \cdot (\text{trace}(\bar{M})^2) = (\bar{A}\bar{B} - \bar{C}^2) - \alpha \cdot (\bar{A} + \bar{B})^2$$

- mit dem Parameter α wird die Empfindlichkeit gesteuert (z.B. 0.05)
- $Q(u, v) >$ Schwellwert (z.B. 10'000 bis 1 Mio)

10.5 Harris Algorithmus

```

1: HARRISCORNERS( $I$ )
    Returns a list of the strongest corners found in the image  $I$ .
2: STEP 1—COMPUTE THE CORNER RESPONSE FUNCTION:
3:     Prefilter (smooth) the original image:  $I' \leftarrow I * H_p$ 
4:     Compute the horizontal and vertical image derivatives:
         $I_x \leftarrow I' * H_{dx}$ 
         $I_y \leftarrow I' * H_{dy}$ 
5:     Compute the local structure matrix  $M(u, v) = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$ :
         $A(u, v) \leftarrow I_x^2(u, v)$ 
         $B(u, v) \leftarrow I_y^2(u, v)$ 
         $C(u, v) \leftarrow I_x(u, v) \cdot I_y(u, v)$ 
6:     Blur each component of the structure matrix:  $\bar{M} = \begin{pmatrix} \bar{A} & \bar{C} \\ \bar{C} & \bar{B} \end{pmatrix}$ :
         $\bar{A} \leftarrow A * H_b$ 
         $\bar{B} \leftarrow B * H_b$ 
         $\bar{C} \leftarrow C * H_b$ 
7:     Compute the corner response function:
         $Q \leftarrow (\bar{A} \cdot \bar{B} - \bar{C}^2) - \alpha \cdot (\bar{A} + \bar{B})^2$ 
8: STEP 2—COLLECT CORNER POINTS:
9:     Create an empty list:
         $Corners \leftarrow []$ 
10:    for all image coordinates  $(u, v)$  do
11:        if  $Q(u, v) > t_H$  and IsLOCALMAX( $Q, u, v$ ) then
12:            Create a new corner:
                 $c_i \leftarrow \langle u_i, v_i, q_i \rangle = \langle u, v, Q(u, v) \rangle$ 
13:            Add  $c_i$  to  $Corners$ 
14:    Sort  $Corners$  by  $q_i$  in descending order (strongest corners first)
15:    GoodCorners  $\leftarrow$  CLEANUPNEIGHBORS( $Corners$ )
16:    return  $GoodCorners$ .
17: IsLOCALMAX( $Q, u, v$ )      ▷ determine if  $Q(u, v)$  is a local maximum
18:     Let  $q_c \leftarrow Q(u, v)$  (center pixel)
19:     Let  $\mathcal{N} \leftarrow Neighbors(Q, u, v)$       ▷ values of all neighboring pixels
20:     if  $q_c \geq q_i$  for all  $q_i \in \mathcal{N}$  then
21:         return true
22:     else
23:         return false.
24: CLEANUPNEIGHBORS( $Corners$ )  ▷  $Corners$  is sorted by descending  $q$ 
25:     Create an empty list:
         $GoodCorners \leftarrow []$ 
26:     while  $Corners$  is not empty do
27:          $c_i \leftarrow REMOVEFIRST(Corners)$ 
28:         Add  $c_i$  to  $GoodCorners$ 
29:         for all  $c_j$  in  $Corners$  do
30:             if  $Dist(c_i, c_j) < d_{min}$  then
31:                 Delete  $c_j$  from  $Corners$ 
32:     return  $GoodCorners$ .

```

11 Wavelet-Transformation

11.1 Einsatz der WT

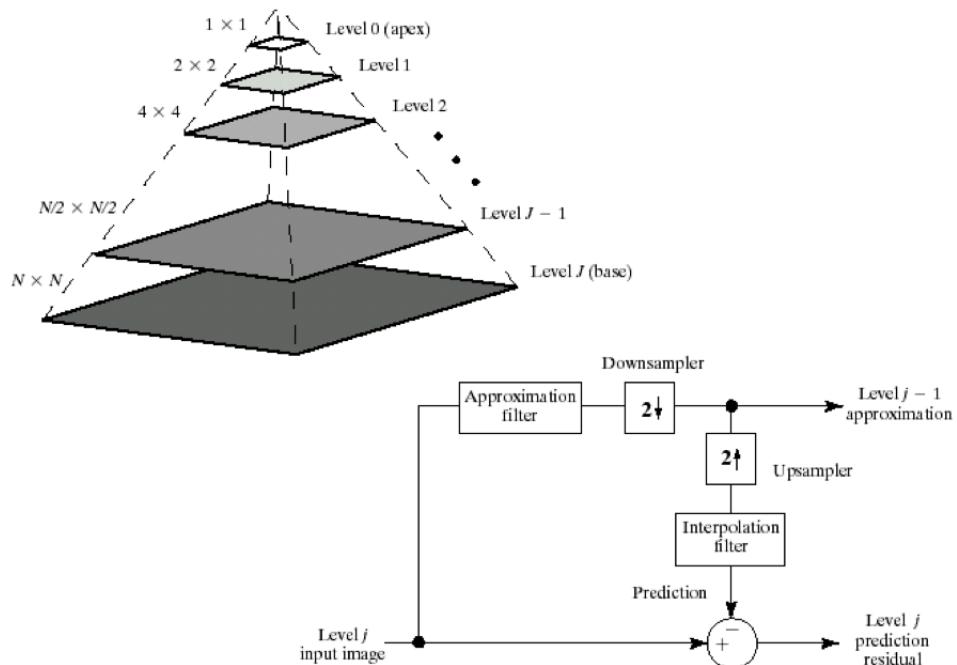
Datenverdichtung:

- Bilder: JPEG2000, PGF, MPEG-4
- Audio

Multi Resolution Models:

- Daten in verschiedenen Auflösungsstufen zur Verfügung stellen
- alle Auflösungsstufen in einem Kompakten Format zusammenhalten
- keine disjunkten Datensätze, sondern alle Daten werden benötigt, um die höchste Auflösungsstufe zu rekonstruieren

11.2 Idee der Bildpyramide



11.3 L_2 -Norm

Ist eine Vektornorm:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad |x| = \sqrt{\sum_{k=1}^n |x_k|^2}$$

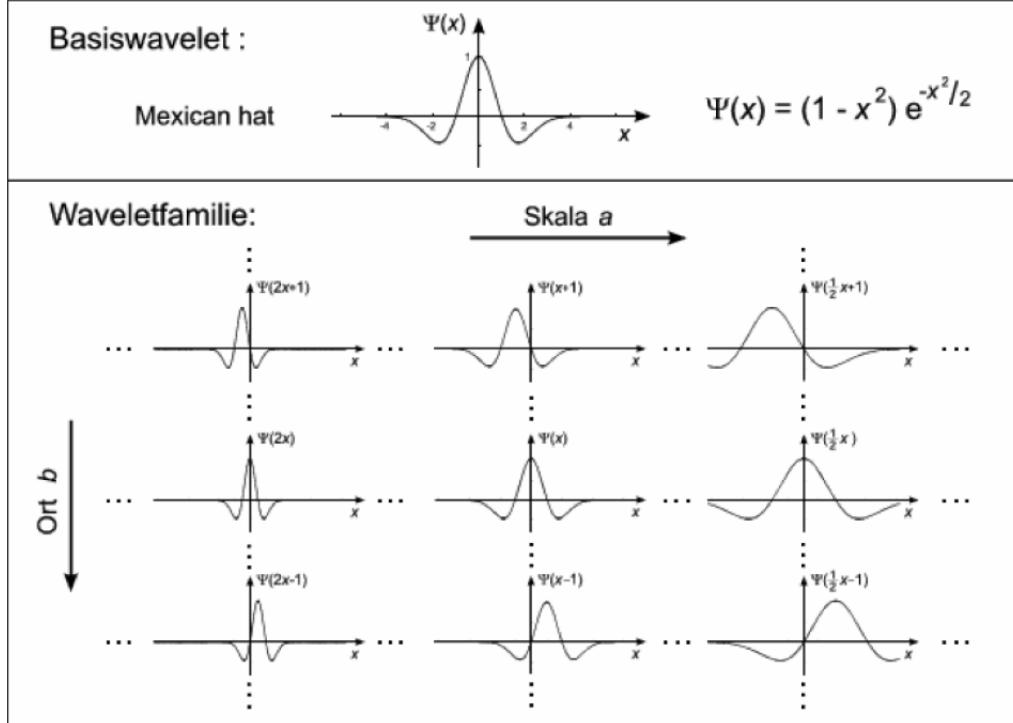
Ist eine Norm für Funktionen $\Phi(x)$:

$$\|\Phi\|^2 \equiv \Phi \circ \Phi \equiv \langle \Phi | \Phi \rangle \equiv \int |\Phi(x)|^2 dx$$

11.4 Mother Wavelet und Wavelet-Familien

Aus Mother Wavelets, auch Basis-Wavelets genannt, kann durch Skalierung ($a > 0$) und Verschiebung (b) eine ganze Wavelet-Familie erzeugt werden:

$$\Psi_{a,b}(x) = \frac{1}{\sqrt{a}} \Psi\left(\frac{x-b}{a}\right)$$



11.5 Continuous Wavelet Transform

- Ausgangssignal $f(x)$
- wähle beliebiges Wavelet Ψ
- berechne Skalarprodukt von $\Psi_{a,b}$ mit $f(x)$ für alle möglichen Paare von a und b
- die berechneten Werte sind die Wavelet-Koeffizienten $CWT_{a,b}$

Vorwärtstransformation

$$CWT_{a,b}(f) = \langle f(x) | \Psi_{a,b} \rangle \equiv \int_{-\infty}^{\infty} f(x) \overline{\Psi_{a,b}(x)} dx$$

Rücktransformation

$$f(x) = \langle CWT_{a,b}(f) | \Psi_{a,b} \rangle \equiv \int_{\mathbb{R}^* \times \mathbb{R}} CWT_{a,b}(f) \overline{\Psi_{a,b}(x)} \frac{da db}{|a|^2}$$

11.6 CWT, DWT und FWT

Problem der CWT:

- es gibt unendliche viele Parameterpaare (a, b)

- es ist unklar, welche Parameterpaare (a,b) erforderlich sind, um eine vollständige Rekonstruktion zu ermöglichen

Lösungsansatz DWT:

- üblicherweise sind diskrete Signale mit begrenzter Abtastung gegeben
- für diskrete Signale muss eine endliche Zahl von Parameterpaaren (a,b) ausreichen, so dass eine Rücktransformation ohne Verlust möglich ist
→ Discrete Wavelet Transform (DWT)

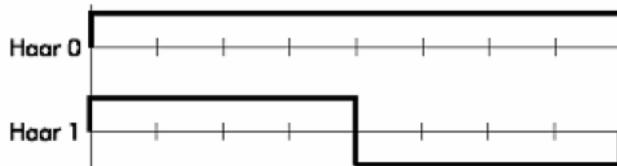
Durchbruch: FWT:

- Multiresolution Analysis führte zum Algorithmus der schnellen DWT → Fast Wavelet Transform (FWT)

11.7 FWT

Input	A	B	C	D	E	F	G	H
Detail 1	$A/2 - B/2$	$B/2 - A/2$	$C/2 - D/2$	$D/2 - C/2$	$E/2 - F/2$	$F/2 - E/2$	$G/2 - H/2$	$H/2 - G/2$
Durchschnitt 1	$AB := A/2 + B/2$		$CD := C/2 + D/2$		$EF := E/2 + F/2$		$GH := G/2 + H/2$	
Detail 2	$AB/2 - CD/2$		$CD/2 - AB/2$		$EF/2 - GH/2$		$GH/2 - EF/2$	
Durchschnitt 2	$ABCD := AB/2 + CD/2$				$EFGH := EF/2 + GH/2$			
Detail 3	$ABCD/2 - EFGH/2$				$EFGH/2 - ABCD/2$			
Durchschnitt 3					$ABCD/2 + EFGH/2$			
Rekonstruktion								
Input	12	4	6	8	4	2	5	7
Detail 1	4	-4	-1	1	1	-1	-1	1
Durchschnitt 1	8		7		3		6	
Detail 2	0.5		-0.5		-1.5		1.5	
Durchschnitt 2			7.5				4.5	
Detail 3			1.5				-1.5	
Durchschnitt 3					6			
Rekonstruktion	12	4	6	8	4	2	5	7

11.8 FWT als Filteroperation



- **Betrachtungsweise**
 - Haar0 und Haar1 skaliert auf ein Intervall der Länge 2
 - Amplitude von Haar0 wird zu $\frac{1}{2}$ (nicht normalisiert)
 - Amplituden von Haar1 werden zu $\frac{1}{2}$ und $-\frac{1}{2}$ (nicht normalisiert)
- **Schreibweise als Filterkoeffizienten**
 - Haar0 = [$\frac{1}{2}$, $\frac{1}{2}$] =: low (Skalierungsfunktion, Tiefpass)
 - Haar1 = [$\frac{1}{2}$, $-\frac{1}{2}$] =: high (Waveletfunktion, Hochpass)
- **Anwendung**
 1. Ausgangssignal jeweils mit low und high falten
 2. Schritt 1 mit dem zuvor skalierten (low) Signal wiederholen

11.9 Rücktransformation

Generelle Rekonstruktion des Ausgangssignals:

- im Wesentlichen wieder ein Skalarprodukt mit einem Wavelet
- Analyse- und Synthese-Wavelet müssen nicht identisch sein

Bei der FWT:

- Verwendung eines Synthese-Filterpaars, passend zum Analyse-Filterpaar und Anwendung der Faltung

11.10 FWT Filterbank

Bi-orthogonale Filterbank:

- (h_L, h_H) : Filterpaar der Vorwärtstransformation (Low, High)
- (g_L, g_R) : Filterpaar der Rückwärtstransformation (Left, Right)
- Rücktransformationsfilter können aus der Vorwärtstransformationsfilter abgeleitet werden:
 $\langle h_L | g_R \rangle = 0$ und $\langle h_H | g_L \rangle = 0$

Beispiel

– Daubechies 5/3, bi-orthogonal [PGF, JPEG2000]

$$h_L = \frac{1}{4\sqrt{2}} [-1, 2, \bar{6}, 2, -1] \quad h_H = \frac{1}{2\sqrt{2}} [-\bar{1}, 2, -1]$$

$$g_L = \frac{1}{2\sqrt{2}} [1, \bar{2}, 1] \quad g_R = \frac{1}{4\sqrt{2}} [-1, \bar{-2}, 6, -2, -1]$$

		Transformationsmatrizen für Signalvektor der Länge 8
Vorwärtstransformation	$\begin{bmatrix} 6 & 4 & -2 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 6 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 6 & 2 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 2 & 5 \\ 8 & -2 & 4 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 4 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 4 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4 & 4 \end{bmatrix}$	Rücktransformation
		$\begin{bmatrix} 4 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 5 & -1 & 0 & 0 \\ 0 & 4 & 0 & 0 & 2 & -2 & 0 & 0 \\ 1 & 0 & 2 & 2 & 0 & -1 & 6 & -1 \\ 4 & 0 & 0 & 4 & 0 & 0 & -2 & -2 \\ 0 & 0 & 2 & 2 & 0 & -1 & 6 & -1 \\ 0 & 0 & 0 & 4 & 0 & 0 & -2 & -2 \\ 0 & 0 & 0 & 4 & 0 & 0 & -2 & 6 \end{bmatrix}$

11.11 FWT in zwei Dimensionen

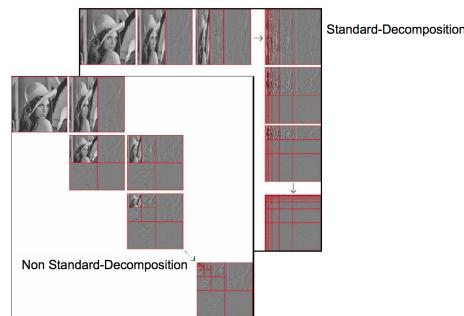
Die Grundidee ist wie bei der DFT wird für mehrere Dimensionen auf die eindimensionale Transformation zurückgegriffen. Die Filterpaare müssen auch normalisiert sein (L_2 -Norm = 1)

Standard Dekomposition:

- mehrstufige FWT für jede Zeile des Bildes
- mehrstufige FWT für jede Spalte des zeilentransformierten Bildes

Nicht-Standard Dekomposition:

- 1 Filterschritt für jede Zeile eines Bildes durchführen
- 1 Filterschritt für jede Spalte des zeilentranformierten Bildes
 → vier Quadranten entstehen: LL, HL, LH, HH
- rekursiv den Quadranten LL wieder filtern



11.12 Bilddaten-Dekorrelation

Innerhalb eines Farbkanals gibt es noch weitere Redundanz (z.B. Flächen mit gleicher Farbe). Um die Redundanz zu reduzieren findet eine Transformation vom Bild- in den Frequenzraum statt.

DCT (discrete cosine transform):

- schnell, einfach
- Blockbildung
- Einsatz in JPEG und MPEG

DWT (discrete wavelet transform):

- schnell, einfach
- keine Blockbildung
- verlustlose Integer-Arithmetik möglich
- Einsatz in PGF, JPEG 2000, MPEG-4

11.13 Quantisierung

Ziel:

- möglichst kleine Anzahl von unterschiedlichen Wavelet-Koeffizienten
- Koeffizienten mit kleinen Absolutwerten (viele Nullen)

Ansatz:

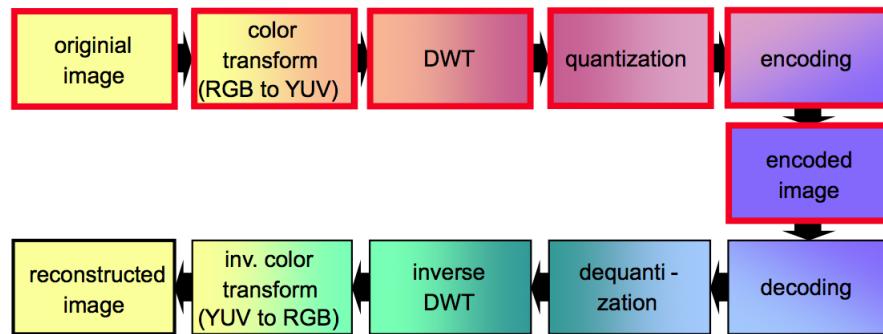
- Zusammenfassen von ähnlich grossen Wavelet- Koeffizienten in Gruppen → Quantisierung
- anstatt des Wertes eines Wavelet-Koeffizienten wird seine Gruppennummer abgespeichert
- für jede Gruppe muss die Spannweite abgespeichert werden → Quantisierungstabelle
- je grösser die Spannweite einer Gruppe, desto grösser der Informationsverlust → ohne Quantisierung kein Informationsverlust

12 PGF

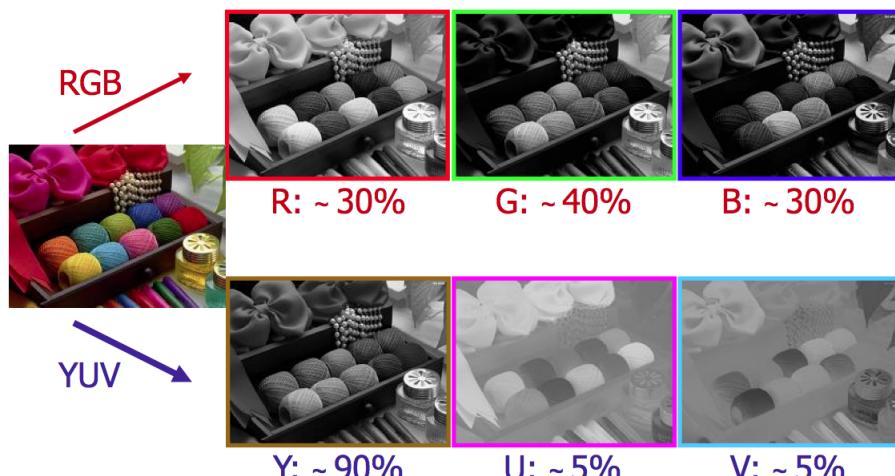
12.1 Features

- sehr hohe verlustlose Kompressionsrate
- sehr gute verlustbehaftete Bildqualität/Kompressionsrate
- kontinuierlicher Datenstrom erlaubt schrittweise Erhöhung der Auflösung

12.2 Aufbau



12.3 Farbtransformation



12.4 5/3 Wavelet Filterbank

- kann ganzzahlig implementiert werden, obwohl Faktor $\sqrt{2}$ vorkommt
- ist relativ kurz (kleine Anzahl von Filterkoeffizienten)
- gehört zu den besten Filtern für Bildverarbeitung

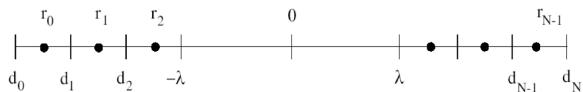
12.5 Skalare Quantisierung

Grundschema

- uniform: die Spannweiten aller Gruppen sind gleich lang (Zweierpotenzen)
→ keine Quantisierungstabelle notwendig
 - sehr einfach, alle Koeffizienten durch die Spannweite dividieren

dead zone

- grösseres Intervall um Null herum, in welchem alle Koeffizienten auf Null gesetzt werden
 - die meisten Wavelet-Koeffizienten sind um Null herum
→ sehr grosse Gruppe mit wenig Informationsverlust



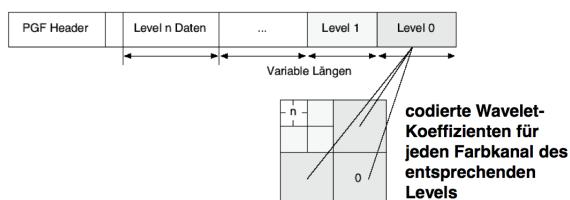
12.6 Progressive Kodierung

Ziele:

- eingebunder Bitstrom fürs sequentielles LEsen
 - schnelles, progressives Kodierungsschema
 - gute Komprimierungsrate

Ansatz:

- Wavelet-Koeffizienten getrennt nach Level abspeichern
 - Umordnen der Wavelet-Koeffizienten, so dass Koeffizienten mit ähnlich grossen Absolutwert nahe beieinander liegen



Umordnen der Koeffizienten:

Koeffizienten in den zusammengehörigen LH- und HL-Subbands an der gleichen Bildposition ähnlich gross

0	LL	1	3	10	12	...	46	48	...
		5	7	18	LH				
2	4	9	HH						
6	8				40				
11	13	...		42	43				
19				44	45				
		...	41						
47	49	...				174	175	176	177
						178	179	180	181
						182	183	184	185
						186	187	188	189

12.7 Bitplane-Kodierung

- **Verfahren**
 - umgeordnete, quantisierte Absolutwerte der Koeffizienten werden in Makroblöcke gleicher Länge aufgeteilt
 - pro Makroblock Bitplane-Kodierung anwenden
→ Aufteilung in signifikante Bits und Verfeinerungsbits
 - signifikante Bits enthalten sehr lange Folgen von Nullen und werden deshalb adaptiv lauflängenkodiert

MSB	
LSB	

12.8 Lauflängenkodierung (RLE)

- **Annahme**
lange Sequenzen von Nullen
- **statisches Verfahren**

Codewort	Inputsequenz
0	2^k Nullen
1n0	$0 < n < 2^k$ Nullen gefolgt von einer 1, positives Vorzeichen an der Stelle i
1n1	$0 < n < 2^k$ Nullen gefolgt von einer 1, negatives Vorzeichen an der Stelle i

n wird mit k Bits abgespeichert

12.9 Adaptives RLE

Problem:

- der bestmögliche Wert für k hängt vom Input ab
- wird k zu gross gewählt, so werden zu viele Bits verschwendet, um n abzuspeichern
- wird k zu klein gewählt, so werden zu viele Codeworte erzeugt

Ansatz (adaptiv):

- k nach jedem Codewort anpassen
- k mit 0 initialisieren
- nach einem Codewort 0: $k + +$
- nach einem anderem Codewort: $k - -(k > 0)$

13 Uebungen

13.1 Uebung 1

Gegeben ist das Bild Schlittentest.raw der Grösse 800 mal 600 Bildpunkte. Es enthält einen Header der Länge 1 KiByte in einem proprietären Format und die Bildinformation von einem CMOS-Bildsensor mit Bayer-Maske.

Pro Bildpunkt (Pixel) ist jeweils nur eine der drei Farb- komponenten (Rot, Grün, Blau) in einem Byte abgespeichert. Die an der ersten Stelle des Bildes gegebene Farbkomponente ist Blau. Sie können dieses Bild in ImageJ mit File/Import/Raw öffnen.

Ergänzen Sie das ImageJ-Plugin BayerMask_.java an den entsprechenden Stellen, um die fehlenden Farbkomponenten durch Interpolation zu erzeugen. Testen Sie Ihr Programm mit dem Bild Schlittentest.raw. Eine gute Bayer-Mask-Interpolation sollte zu möglichst wenig Farbartefakten (z.B. Farb-Moiré im weissen Ärmel oder bei den weissen Drähten) führen (siehe nachfolgenden Bildausschnitt).

Listing 1: Bayer Mask

```

1 // BVERI
// bung 1.3

import ij.*;
import ij.process.*;
6 import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

public class BayerMask_ implements PlugInFilter {
11 static final int R = 0;
static final int G = 1;
static final int B = 2;
ImagePlus imp;

16 public int setup(String arg, ImagePlus imp) {
    this.imp = imp;
    return DOES_8G + NO_CHANGES;
}

21 public void run(ImageProcessor ip) {
    final int w = ip.getWidth();
    final int h = ip.getHeight();
    ImageProcessor ip2 = new ColorProcessor(w, h);
    int[] rgb = new int[3];
26    int g1, g2, g3, g4;
    int b1, b2, b3, b4;
    int r1, r2, r3, r4;

    // um die Aufgabe zu vereinfachen, fahren wir keine Randbehandlung durch und lassen am
    // Rand 2 Pixel schwarz
31    for (int v=2; v < h - 2; v++) {
        for (int u=2; u < w - 2; u++) {
            // hier die fehlenden Farbkomponenten interpolieren
            if (v%2 == 0) {
                if (u%2 == 0) {
36                    // b
                    rgb[B] = ip.getPixel(u, v);
                    g1 = ip.getPixel(u-1, v);
                    g2 = ip.getPixel(u+1, v);
                    g3 = ip.getPixel(u, v-1);
                    g4 = ip.getPixel(u, v+1);
                    rgb[G] = (g1 + g2 + g3 + g4) >> 2;
                    r1 = ip.getPixel(u-1, v-1);
                    r2 = ip.getPixel(u+1, v-1);
                    r3 = ip.getPixel(u-1, v+1);
                    r4 = ip.getPixel(u+1, v+1);
                    rgb[R] = (r1 + r2 + r3 + r4) >> 2;
                } else {
                    // g
                    rgb[G] = ip.getPixel(u, v);
                    r1 = ip.getPixel(u, v-1);
                    r2 = ip.getPixel(u, v+1);
                    rgb[R] = (r1 + r2) >> 1;
                    b1 = ip.getPixel(u-1, v);
                    b2 = ip.getPixel(u+1, v);
                    rgb[B] = (b1 + b2) >> 1;
                }
            } else {
                if (u%2 == 0) {
51
56

```

```

    // g
61   rgb[G] = ip.getPixel(u, v);
    r1 = ip.getPixel(u-1, v);
    r2 = ip.getPixel(u+1, v);
    rgb[R] = (r1 + r2) >> 1;
    b1 = ip.getPixel(u, v-1);
66   b2 = ip.getPixel(u, v+1);
    rgb[B] = (b1 + b2) >> 1;
} else {
    // r
    rgb[R] = ip.getPixel(u, v);
71   g1 = ip.getPixel(u-1, v);
    g2 = ip.getPixel(u+1, v);
    g3 = ip.getPixel(u, v-1);
    g4 = ip.getPixel(u, v+1);
    rgb[G] = (g1 + g2 + g3 + g4) >> 2;
76   b1 = ip.getPixel(u-1, v-1);
    b2 = ip.getPixel(u+1, v-1);
    b3 = ip.getPixel(u-1, v+1);
    b4 = ip.getPixel(u+1, v+1);
    rgb[B] = (b1 + b2 + b3 + b4) >> 2;
81 }
}
ip2.putPixel(u, v, rgb);
}
86 ImagePlus imp2 = new ImagePlus("RGB", ip2);
imp2.show();
}
91 }

```

13.2 Uebung 2

In dieser Aufgabe wollen wir den Huffman-Code aus Aufgabe 2 verwenden, um ein Graustufenbild entropiecodiert abzuspeichern. Erweitern Sie das Plugin so, dass Sie eine Binärdatei mittels eines ObjectOutputStreams erzeugen können. Der ObjectOutputStream eignet sich für die Objektserialisierung. Betrachten Sie die Methoden write und encodeImage. a) Schreiben Sie in den Header der Datei die Breite und Höhe des Bildes. Verwenden Sie dazu einfach zwei Integer.

- b) Für das Decodieren der Datei wird der Code-Baum aus Aufgabe 2 wiederum benötigt. Daher sollten Sie nach dem Header den ganzen Code-Baum serialisieren.
- c) Zum Schluss kommen noch die codierten Bilddaten. Der Einfachheit halber können Sie ein BitSet verwenden. Iterieren Sie durch alle Pixel des Bildes und schreiben Sie den entsprechenden Huffman-Code jedes Pixels hintereinander ins BitSet. Anschliessend serialisieren Sie das BitSet. Tipp: Schreiben Sie den Huffman-Code jeweils mit dem MSB zuerst ins BitSet.
- d) Denn dazu passenden Decoder finden Sie in Huffman_Dec.java. Falls Sie leicht anders vorgegangen sind, müssen Sie den Decoder unter umständen leicht anpassen.
- e) Codieren Sie ein Graustufenbild mit Ihrem Encoder (stimmt der erwartete Speicherbedarf mit dem wirklichen überein?) und decodieren Sie die entstandene huf-Datei mit dem Decoder. Verwenden Sie in ImageJ den ImageCalculator, um die Differenz zwischen dem Originalbild und dem decodierten Bild zu bestimmen.

Listing 2: Huffman Decodierung

```

/**
 * Huffman decoder
 * @author Christoph Stamm
4 *
 */
import java.io.*;

import ij.*;
9 import ij.io.*;
import ij.gui.*;

```

```

import ij.plugin.*;
import java.util.*;

14 public class Huffman_Dec implements PlugIn {
    ImagePlus img;

    public Huffman_Dec() {
        img = null;
    }

    public void run(String arg) {
        OpenDialog od = new OpenDialog("Open HUF encoded image ...", arg);
        String directory = od.getDirectory();
    24 String fileName = od.getFileName();
        if (fileName == null)
            return;

        IJ.showStatus("Opening: " + directory + fileName);
    29 read(directory, fileName);

        if (img == null)
            return;
    34 img.show();
    }

    39 protected void read(String dir, String filename) {
        int width, height;

        try {
            ObjectInputStream in = new ObjectInputStream(new FileInputStream(dir + filename));
    44 // read Header
            width = in.readInt();
            height = in.readInt();

            // read code tree
            Node root = (Node)in.readObject();

            // read compressed data
            BitSet data = (BitSet)in.readObject();
    49 // close file
            in.close();

            // create output image
            img = NewImage.createByteImage(filename, width, height, 1, NewImage.FILL_BLACK);

            // fill in data
            Node node;
            int index = 0;
    54 byte[] pixels = (byte[])img.getProcessor().getPixels();
            for (int i = 0; i < pixels.length; i++) {
                node = root;
                while(node.isInnerNode()) {
                    node = node.decodeBit(data.get(index++));
                }
                pixels[i] = ((Leaf)node).getIntensity();
            }
        }
    59 } catch (Exception e) {
        IJ.error("Huffman Decoder", e.getMessage());
        return;
    }
}
}

```

In dieser Aufgabe sollen Sie einen Huffman-Entropiecodierer als ImageJ-Plugin programmieren und auf ein Graustufenbild anwenden. Als Gerüst können Sie Huffman_Enc.java verwenden. Betrachten Sie die Methode createHuffmanTree().

- Bestimmen Sie zuerst die Wahrscheinlichkeiten, mit denen die einzelnen Intensitäten in einem Graustufenbild auftreten. Verwenden Sie dazu das Histogramm.
- Schätzen Sie mit Hilfe des Resultats aus a) die mittlere Codelänge und den resultierenden Speicherbedarf für das codierte Bild ab.
- Programmieren Sie die Erzeugung des Code-Baumes und speichern Sie die einzelnen Codewörter für die verschiedenen Intensitäten ab. Zur Erzeugung des Code-Baums eignet sich eine PriorityQueue<Node> besonders gut, weil damit auf sehr einfache Art die beiden Knoten mit kleinster Wahrscheinlichkeit gefunden werden können.
- Berechnen Sie die mittlere Codelänge Ihres Codes und überprüfen Sie Ihr Resultat mit demjenigen von b).

Listing 3: Huffman Encodierung

```

// BVERI
// bung 2.2

import ij.*;
import ij.process.*;
import ij.plugin.filter.*;
import ij.gui.*;
import ij.io.*;
import java.io.*;
import java.util.*;

12 /**
 * Huffman encoder
 * @author Christoph Stamm
 *
 */
17 public class Huffman_Enc implements PlugInFilter {
    ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G + NO_CHANGES;
    }

    public void run(ImageProcessor ip) {
        final int w = ip.getWidth();
        final int h = ip.getHeight();
        final int size = w*h;

        int[] hist = ip.getHistogram();
        Leaf[] codes = new Leaf[hist.length];
        Node root = createHuffmanTree(hist, codes, size);
        BitSet data = encodeImage(ip, codes);

        // compute mean code length
        //StringBuilder sb = new StringBuilder();
        37 double sum = 0;
        for(int i=0; i < codes.length; i++) {
            //sb.append(" " + i + ":" + codes[i].getProbability() + ", " + codes[i].getCodeLen()
            //        + ", " + codes[i].getCode() + "\n");
            sum += codes[i].getProbability()*codes[i].getCodeLen();
        }

        42 new MessageDialog(imp.getWindow(), "wirklicher Speicherbedarf",
            "mittlere Codelnge: " + sum + "\n" +
            "Speicherbedarf: " + sum*size/8 + " Byte");

        SaveDialog sd = new SaveDialog("Save image in HUF format", imp.getTitle(), ".huf");
        if (sd.getFileName() != null) {
            imp.startTiming();
            write(sd.getFileName(), sd.getDirectory(), ip, root, data);
        }
52    }
}

```

```

    /**
     * Build code tree
     * @param hist histogram of input image
     * @param codes code table
     * @param size number of pixels
     * @return root node of code tree
     */
57  private Node createHuffmanTree(int[] hist, Leaf[] codes, int size) {
    PriorityQueue<Node> pq = new PriorityQueue<Node>(hist.length);
    double ld = Math.log(2), H = 0, p;

    // compute probabilities and entropy
67  for(int i=0; i < hist.length; i++) {
        p = (double)(hist[i])/size;
        H -= ((p == 0) ? 0 : p*Math.log(p)/ld);
        codes[i] = new Leaf(p, (byte)i);
        pq.add(codes[i]);
    }

    // compute needed memory
    new MessageDialog(imp.getWindow(), "geschätzter Speicherbedarf",
        "gesetzte mittlere Codelänge: [" + (float)H + ", " + (float)(H + 1) + "]\n" +
        "gesetzter Speicherbedarf: [" + (float)H*size/8 + ", " + (float)(H + 1)*size/8 + "]
        Byte");
77

    // build Huffman tree
    while(pq.size() >= 2) {
        Node v1 = pq.poll();
        Node v2 = pq.poll();

        pq.add(new Node(v1, v2));
    }
    Node root = pq.poll();
87    //new MessageDialog(imp.getWindow(), "Probability", "p = " + root.getProbability());
    root.setCode(0, 0);

    return root;
}
92

    /**
     * Encode image
     * @param ip
     * @param codes code table
     * @return encoded data
     */
97  private BitSet encodeImage(ImageProcessor ip, Leaf[] codes) {
    final int w = ip.getWidth();
    final int h = ip.getHeight();
102  BitSet bs = new BitSet();
    int index = 0;
    Node node;
    long code;

    //encode image data:
    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
            node = codes[ip.getPixel(u, v)];
            code = node.getCode();
112            for (int i = node.getCodeLen() - 1; i >= 0; i--) {
                bs.set(index + i, code%2 == 1);
                code >>= 1;
            }
            index += node.getCodeLen();
        }
    }
117    return bs;
}

122    /**
     * Write file
     * @param filename

```

```

    * @param directory
    * @param ip
127   * @param root root of code tree
    * @param data encoded data
    */
    private void write(String filename, String dir, ImageProcessor ip, Node root, BitSet data
        ) {
        final int w = ip.getWidth();
        final int h = ip.getHeight();

132   try{
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(dir + filename))
            ;
            ;

137     // write Header
        out.writeInt(w);
        out.writeInt(h);

            //
142     out.writeObject(root);

            //
        out.writeObject(data);

            //
147     // close file
        out.close();

        IJ.showMessage("Huffman Encoder", "Image has been successfully saved.\n \n");
    } catch(Exception e){
152     IJ.error("Huffman Encoder", e.getMessage());
    }
}

}

```

Listing 4: Node

```

import java.io.Serializable;

/*
4  * Node of code tree
 * @author Christoph Stamm
 *
 */
class Node implements Comparable<Node>, Serializable {
9  static final long serialVersionUID = 1;
    protected transient double m_p;    // probability (not needed during decoding)
    protected transient long m_code;   // binary code; maximum 64 bits (not needed during
        decoding)
    protected transient byte m_codeLen; // binary code length (not needed during decoding)
    protected Node m_left, m_right;    // children
14
    public Node(double p) {
        m_p = p;
    }

19    public Node(Node left, Node right) {
        m_left = left;
        m_right = right;
        m_p = left.m_p + right.m_p;
    }

24    public double getProbability() {
        return m_p;
    }

29    public long getCode() {
        return m_code;
    }

```

```

    public byte getCodeLen() {
34        return m_codeLen;
    }

    public boolean isInnerNode() {
        return m_left != null;
39    }

    public int compareTo(Node v) {
        if (v != null) {
            if (m_p < v.m_p) {
                return -1;
            } else if (m_p == v.m_p) {
                return 0;
            } else {
                return 1;
            }
        } else {
            return -1;
        }
    }
54

    public void setCode(long code, int codeLen) {
        m_code = code;
        m_codeLen = (byte)codeLen;

        if (m_left != null) {
            // 0-Bit
            m_left.setCode(code << 1, codeLen + 1);
        }
        if (m_right != null) {
            // 1-Bit
            m_right.setCode((code << 1) + 1, codeLen + 1);
        }
    }
69

    public Node decodeBit(boolean bit) {
        return (bit) ? m_right : m_left;
    }
}

74 class Leaf extends Node {
    static final long serialVersionUID = 1;
    private byte m_intensity; // pixel intensity used during decoding

    public Leaf(double p, byte intensity) {
79        super(p);
        m_intensity = intensity;
    }

    public byte getIntensity() {
84        return m_intensity;
    }
}

```

Die aktuelle Version von ImageJ unterstützt das PGM-Dateiformat. PGM-Dateien im ASCII- und im Binärformat können eingelesen werden, aber beim Erstellen wird immer nur das Binärformat verwendet.

- Erstellen Sie ein neues Plugin ToPGM.java für 8-Bit-Grauwertbilder, welches ein Bild als PGM-Datei im ASCII-Format abspeichert.
- Wie gross ist die Kompressionsrate, wenn ein Bild im Binär- anstatt im ASCII-Format abgespeichert wird? Schätzen Sie die Kompressionsrate für ein beliebiges Bild theoretisch ab. Überprüfen Sie Ihre Abschätzung an konkreten Beispielen (bridge.gif).

Listing 5: To PGM

```

// BVERI
// bung 2.1

4 import ij.*;

```

```

import ij.process.*;
import ij.io.*;
import ij.gui.*;
import java.awt.*;
9 import ij.plugin.filter.*;
import java.io.*;

import ij.plugin.filter.*;

14 public class ToPGM_ implements PlugInFilter {
    ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
19        return DOES_8G + NO_CHANGES;
    }

    public void run(ImageProcessor ip) {
        SaveDialog sd = new SaveDialog("Save image as PGM in ASCII format", imp.getTitle(), ".pgm");
24        if (sd.getFileName() != null) {
            imp.startTiming();
            savePGM(sd.getFileName(), sd.getDirectory(), ip);
        }
    }

29    private void savePGM(String filename, String directory, ImageProcessor ip) {
        final int w = ip.getWidth();
        final int h = ip.getHeight();
        final int maxGrayVal = 255;
34        int p;

        try{
            BufferedWriter out = new BufferedWriter(new FileWriter(directory + filename));

            // Write Header:
            out.write("P2\n");
            out.write("# " + filename + "\n");
            out.write(w + " " + h + "\n");
            out.write(maxGrayVal + "\n");
44

            //Write image data:
            for (int v = 0; v < h; v++) {
                for (int u = 0; u < w; u++) {
                    p = ip.getPixel(u,v);
                    out.write(ip.getPixel(u, v) + " ");
                }
                out.write("\n");
            }
            out.close();
54

            IJ.showMessage("PGM Writer", "Image has been successfully saved.\n \n");
        } catch(Exception e){
            IJ.showMessage("PGM Writer", "An error occured writing the file.\n \n" + e.toString());
        }
59    }
}

```

13.3 Uebung 3

Programmieren Sie den linearen Histogrammausgleich gemäss Punktoperationen Folie 22. Verwenden Sie dazu eine Lookup-Table, die Sie als int-Array der Länge 256 für alle möglichen Intensitäten im Voraus berechnen. Mit der Instanzmethode applyTable() der Klasse ImageProcessor können Sie anschliessend die Lookup-Table auf das Bild anwenden.

Listing 6: Linearer Histogrammausgleich

```

import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
5 import ij.plugin.filter.*;

public class LinHistAusgleich_ implements PlugInFilter {
    ImagePlus imp;

10 public int setup(String arg, ImagePlus imp) {
    this.imp = imp;
    //return DOES_ALL;
    return DOES_8G;
}

15 public void run(ImageProcessor ip) {
    final int w = ip.getWidth();
    final int h = ip.getHeight();

20    // Histogramm erstellen
    int[] H = ip.getHistogram();
    int K = H.length; // Anzahl Intensitts-Stufen
    int k1 = K-1; // maximale Intensitt
    int n = w*h; // Anzahl Pixel
25    int[] LUT = new int[K]; // lookup table
    int hKum = 0; // kumuliertes Histogramm

    for (int i = 0; i < K; i++) {
        // kumuliertes Histogramm berechnen
30        hKum += H[i];

        // Histogrammausgleich auf LUT anwenden
        LUT[i] = hKum*k1/n;
    }

35    // LUT aufs Bild anwenden
    ip.applyTable(LUT);
}
}

```

Programmieren Sie die modifizierte Gammakorrektur mit variablen Werten für Gamma und x_0 gemäss Punktoperationen Folie 27 für ein Graustufenbild. Verwenden Sie wiederum eine Lookup-Table.

Listing 7: Gamma

```

1 // Uebung 3.2

import ij.*;
import ij.process.*;
import ij.gui.*;
6 import java.awt.*;
import ij.plugin.filter.*;

public class ModGamma_ implements PlugInFilter {
    ImagePlus imp;

11 public int setup(String arg, ImagePlus imp) {
    this.imp = imp;
    return DOES_8G;
}

16 public void run(ImageProcessor ip) {
    final int K = 256; //Anzahl Intensitten
    final int iMax = K - 1;
    final double gamma = 1/2.4; // sRGB
21    final double x0 = 0.00304; // sRGB
    //final double gamma = 1/2.222; // ITU
    //final double x0 = 0.018; // ITU

    int[] LUT = new int[K]; //Lookup Table

```

26

```

// Vorberechnungen
final double s = gamma/(x0*(gamma - 1) + Math.pow(x0, 1 - gamma));
final double d = 1/(Math.pow(x0, gamma)*(gamma - 1) + 1) - 1;

31 // LUT erstellen
double x;
for(int i = 0; i < K; i++){
    x = (double)i/iMax;
    if (x <= x0) {
        LUT[i] = (int) Math.round(s*x*iMax);
    } else {
        LUT[i] = (int) Math.round(((1 + d)*Math.pow(x, gamma) - d)*iMax);
    }
}
41 ip.applyTable(LUT);
}
}

```

Bestimmen Sie die inverse Funktion zur modifizierten Gammakorrektur. Beachten Sie dabei, dass der Trennwert, wo zwischen linearer und Potenzfunktion unterschieden wird, auch angepasst werden muss. Programmieren Sie die inverse modifizierte Gammakorrektur mit variablen Werten für Gamma und x_0 . Verwenden Sie wiederum eine Lookup-Table.

Listing 8: Gamma Inverse

```

// Uebung 3.3

import ij.*;
import ij.process.*;
5 import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

public class ModGammaInverse_ implements PlugInFilter {
10 ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G;
    }

15    public void run(ImageProcessor ip) {
        final int K = 256; // Anzahl Intensitten
        final int iMax = K - 1;
        final double gamma = 1/2.4; // sRGB
        final double x0 = 0.00304; // sRGB
20        //final double gamma = 1/2.222; // ITU
        //final double x0 = 0.018; // ITU

        int[] LUT = new int[K]; // Lookup Table

25        // Vorberechnungen
        final double s = gamma/(x0*(gamma - 1) + Math.pow(x0, 1 - gamma));
        final double d = 1/(Math.pow(x0, gamma)*(gamma - 1) + 1) - 1;

30        // LUT erstellen
        double y;
        for(int i = 0; i < K; i++){
            y = (double)i/iMax;
            if (y <= s*x0) {
                LUT[i] = (int) Math.round(y/s*iMax);
            } else {
                LUT[i] = (int) Math.round(Math.pow((y + d)/(1 + d), 1/gamma)*iMax);
            }
        }

40        ip.applyTable(LUT);
    }
}

```

```
    }  
45 }
```

13.4 Uebung 5

Basierend auf der YUV-Farbtransformation der Aufgabe 1 wollen wir eine Umwandlung in ein Grauwertbild vornehmen. Als Grauwert werden wir den Luminanzwert des YUV-Farbraums verwenden.

Gegeben sei wiederum eine Eingabebild im sRGB-Farbraum mit einem effektiven Gamma γ von 1/2.2. a) Überführen Sie das nicht-lineare sRGB-Bild in den linearen RGB-Farbraum mit Hilfe der modifizierten Gammakorrektur.

b) Erstellen Sie ein erstes Grauwertbild, indem Sie den Y-Kanal der YUV-Transformation aus Aufgabe 1 berechnen.

c) Eine abgekürzte, nicht ganz korrekte Umrechnung erlaubt die Transformation in einem einzigen Schritt aus den sRGB-Komponenten. Dabei wird Y wie folgt berechnet: $Y = 0.309 * R' + 0.609 * G' + 0.082 * B'$

d) Bestimmen Sie die Bildqualität des zweiten Grauwertbildes bezüglich des ersten in dB. Berechnen Sie dazu die PSNR.

Listing 9: RGB zu Graustufenbild

```
// Uebung 5.2  
  
import ij.*;  
import ij.plugin.filter.PlugInFilter;  
5 import ij.process.*;  
import java.awt.*;  
import ij.gui.*;  
  
public class RGBtoGray_ implements PlugInFilter {  
10    ImagePlus imp;  
  
    public int setup(String arg, ImagePlus imp) {  
        this.imp = imp;  
        return DOES_ALL + NO_CHANGES;  
15    }  
  
    private int clamp(int v) {  
        if (v > 255) return 255;  
        else if (v < 0) return 0;  
20        return v;  
    }  
  
    private int[] createInverseGammaLUT() {  
        final int K = 256; //Anzahl Intensitten  
25        final int iMax = K - 1;  
        final double gamma = 1/2.4;  
        final double x0 = 0.00304;  
  
        int[] LUT = new int[K]; //Lookup Table  
30        // Vorberechnungen  
        final double s = gamma/(x0*(gamma - 1) + Math.pow(x0, 1 - gamma));  
        final double d = 1/(Math.pow(x0, gamma)*(gamma - 1) + 1) - 1;  
        //MessageDialog msg = new MessageDialog(imp.getWindow(), "LUT", "s: " + s + ", d: " + d  
        );  
35        // LUT erstellen  
        double y;  
        for(int i = 0; i < K; i++){  
            y = (double)i/iMax;  
40            if (y <= s*x0) {  
                LUT[i] = (int) Math.round(y/s*iMax);  
            } else {  
                LUT[i] = (int) Math.round(Math.pow((y + d)/(1 + d), 1/gamma)*iMax);  
            }  
45        }  
}
```

```

        return LUT;
    }

    public void run(ImageProcessor ip) {
50     // Bit-Masken
     // 00FF0000 = Rot
     // 0000FF00 = Grün
     // 000000FF= Blau

55     final int w = ip.getWidth();
     final int h = ip.getHeight();
     final int size = w*h;

     int color;
60     int Y1, Y2;
     int R, G, B;
     double psnr = 0;
     int[] LUT = createInverseGammaLUT(); // Aufgabe a

65     ImageProcessor ip2 = ip.duplicate();
     ImagePlus imp2 = imp.createImagePlus();
     imp2.setProcessor("Gammakorrigiertes RGB", ip2);

     ImageProcessor grayIp1 = new ByteProcessor(w, h);
     ImagePlus grayImg1 = imp.createImagePlus();
     grayImg1.setProcessor("Graustufen-Bild", grayIp1);

     ImageProcessor grayIp2 = new ByteProcessor(w, h);
     ImagePlus grayImg2 = imp.createImagePlus();
     grayImg2.setProcessor("Graustufen-Bild ohne Linearisierung", grayIp2);

     for (int x = 0; x < w; x++) {
         for (int y = 0; y < h; y++) {
             color = ip.get(x, y);
             R = (0x00FF0000 & color) >> 16;
             G = (0x0000FF00 & color) >> 8;
             B = (0x000000FF & color);

             // Aufgabe c
             Y2 = clamp((int)Math.round(0.309*R + 0.609*G + 0.082*B));
             grayIp2.putPixel(x, y, Y2);

             // Aufgabe a
             R = LUT[R];
             G = LUT[G];
             B = LUT[B];
             ip2.putPixel(x, y, (((R << 8) | G) << 8) | B));

             // Aufgabe b
95            Y1 = clamp((int)Math.round(0.299*R + 0.587*G + 0.114*B)); // YUV
             Y1 = clamp((int)Math.round(0.2125*R + 0.7154*G + 0.072*B)); // ITU
             grayIp1.putPixel(x, y, Y1);

             // Aufgabe d
100           psnr += (Y1 - Y2)*(Y1 - Y2);
         }
     }

     // Aufgabe d
105           psnr = 20*Math.log10(255/Math.sqrt(psnr/size));

     imp2.show();
     grayImg1.show();
     grayImg2.show();
110           MessageDialog msg = new MessageDialog(ip.getWindow(), "PSNR", "PSNR: " + psnr);
    }
}

```

YUV ist die Basis für die Farbkodierung im analogen Fernsehen, sowohl im nordamerikanischen NTSC- als auch im europäischen PAL-System. Die Luminanzkomponente Y wird aus den RGB-Komponenten in der Form $Y = 0.299 * R + 0.587 * G + 0.114 * B$

abgeleitet, wobei angenommen wird, dass die RGB-Werte bereits nach dem TV-Standard für die Wiedergabe gammakorrigiert sind ($\gamma_{NTSC} = 2.2$). Die UV-Komponenten sind als gewichtete Differenz zwischen dem Luminanzwert und dem Blau- bzw. Rotwert definiert, konkret als $U = 0.492 * (B - Y)$

und

$$V = 0.877 * (R - Y).$$

- a) Programmieren Sie die lineare RGB-YUV-Farbtransformation für das amerikanische Fernsehensystem NTSC. Das Eingabebild sei im nicht-linearen sRGB-Farbraum mit einem effektiven Gamma γ von 1/2.2.
- b) Programmieren Sie die lineare YUV-RGB-Rücktransformation. Die Koeffizienten der Rücktransformationsmatrix erhalten Sie durch Inversion der Matrix.
- c) Berechnen Sie die Qualität des rücktransformierten Bildes bezüglich des Originals. Berechnen Sie dazu die PSNR für jeden der drei Farbkanäle.

Listing 10: YUV Transformation

```
// Uebung 5.1
2
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*;
7 import ij.gui.*;

public class YUVTransform_ implements PlugInFilter {
    ImagePlus imp;

12    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_ALL + NO_CHANGES;
    }

17    private int clamp(int v) {
        if (v > 255) return 255;
        else if (v < 0) return 0;
        return v;
    }

22    public void run(ImageProcessor ip) {
        // Bit-Masken
        // 00FF0000 = Rot
        // 0000FF00 = Grn
27        // 000000FF= Blau

        final int w = ip.getWidth();
        final int h = ip.getHeight();
        final int size = w*h;

32        int color;
        int Y, U, V;
        int R, G, B;
        int oldR, oldG, oldB;
37        double psnrR = 0, psnrG = 0, psnrB = 0;

        ImageProcessor newIp = ip.duplicate();
        ImagePlus newImg = imp.createImagePlus();
        newImg.setProcessor("Nach YUV-Trans und back", newIp);
42

        for (int x = 0; x < w; x++) {
            for (int y = 0; y < h; y++) {
                color = ip.getPixel(x, y);
                oldR = (0x00FF0000 & color) >> 16;
47                oldG = (0x0000FF00 & color) >> 8;
                oldB = (0x000000FF & color);

                // Aufgabe a
                Y = (int) Math.round((0.299*oldR) + (0.587*oldG) +(0.114*oldB));
            }
        }
    }
}
```

```

52     U = (int)Math.round((-0.147*oldR) + (-0.289*oldG) +(0.436*oldB));
53     V = (int)Math.round((0.615*oldR) + (-0.515*oldG) +(-0.1*oldB));
54     //U= (int)Math.round(0.492*(oldB-Y));
55     //V =(int)Math.round(0.877*(oldR-Y));
56
57     Y = clamp(Y);
58     // kein clamping f r U und V, weil diese negative Werte haben drfen
59
60     // Aufgabe b
61     R = (int)Math.round(Y - 3.9457e-005*U + 1.1398*V);
62     G = (int)Math.round(Y - 0.39461*U - 0.5805*V);
63     B = (int)Math.round(Y + 2.032*U - 0.00048138*V);
64
65     //G=(int)Math.round(Y-(0.00003947313749*U)-(0.580809209*V));
66     //R=(int)Math.round((1.140250855*V)+Y);
67     //B=(int)Math.round((2.032520325*U)+Y);
68
69     R = clamp(R);
70     G = clamp(G);
71     B = clamp(B);
72
73     // Aufgabe c
74     psnrR += (oldR - R)*(oldR - R);
75     psnrG += (oldG - G)*(oldG - G);
76     psnrB += (oldB - B)*(oldB - B);
77
78     newIp.putPixel(x, y, (R << 16) | (G << 8) | B);
79   }
80 }
81
82 // Aufgabe c
83 psnrR = 20*Math.log10(255/Math.sqrt(psnrR/size));
84 psnrB = 20*Math.log10(255/Math.sqrt(psnrG/size));
85 psnrG = 20*Math.log10(255/Math.sqrt(psnrB/size));
86
87 newImg.show();
88
89 MessageDialog msg = new MessageDialog(imp.getWindow(), "PSNR", "Red: " + psnrR + ", "
90                                         "Green: " + psnrG + ", Blue: " + psnrB);
91 }

```

13.5 Uebung 6

Implementieren Sie ein Gauss'sches Glättungsfilter. Die Grösse des Filters (Radius ω) soll beliebig einstellbar sein. Ein sinnvoller Wert von ω wäre zum Beispiel 3. Erstellen Sie die zugehörige Filtermatrix dynamisch mit einer Grösse von mindestens 5ω in beiden Richtungen. Nutzen Sie die x-/y-Separierbarkeit der Gaussfunktion.

Listing 11: Gauss Filter

```

//BVER
// bung 6 - Aufgabe 2.3

4 import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

9 public class GaussFilter_ implements PlugInFilter {
    ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
14        this.imp = imp;
        return DOES_8G;
    }

    private void convolution(ImageProcessor ip, int[] filter, int quot, int hotSpot) {

```

```

19     final int w = ip.getWidth();
20     final int h = ip.getHeight();
21     final int len = filter.length;

22     ImageProcessor copy = ip.duplicate();
23     int sum, x, y;

24     // horizontale Filterung an der Position u,v
25     for (int v = 0; v < h; v++) {
26         for (int u = 0; u < w; u++) {
27             sum = 0;
28             for (int i = 0; i < len; i++) {
29                 // Randbehandlung
30                 x = u + i - hotSpot;
31                 if (x < 0) x = -x;
32                 if (x >= w) x = 2*w - 1 - x;
33
34                 // neue Intensität berechnen
35                 sum = sum + filter[i]*ip.getPixel(x, v);
36             }
37             sum /= quot;
38
39             // neue Intensität setzen
40             copy.putPixel(u, v, sum);
41         }
42     }

43     // vertikale Filterung an der Position u,v
44     for (int v = 0; v < h; v++) {
45         for (int u = 0; u < w; u++) {
46             sum = 0;
47             for (int j = 0; j < len; j++) {
48                 // Randbehandlung
49                 y = v + j - hotSpot;
50                 if (y < 0) y = -y;
51                 if (y >= h) y = 2*h - 1 - y;
52
53                 // neue Intensität berechnen
54                 sum = sum + filter[j]*copy.getPixel(u, y);
55             }
56             sum /= quot;
57
58             // clamping
59             if (sum < 0) sum = 0;
60             if (sum > 255) sum = 255;
61
62             // neue Intensität setzen
63             ip.putPixel(u, v, sum);
64         }
65     }

66     public void run(ImageProcessor ip) {
67         final int sigma = 3;
68         final int sigma2 = 2*sigma*sigma;
69         final int factor = 100;

70         int quot, x, len = 5*sigma;
71         int len2 = len/2;
72         len = 2*len2 + 1;
73
74         // Filtermatrix erzeugen
75         int[] filter = new int[len];
76         quot = 0;
77         for (int i=0; i < len; i++) {
78             x = i - len2;
79             filter[i] = (int)(factor*Math.exp(-x*x/(double)sigma2));
80             quot += filter[i];
81         }

82         convolution(ip, filter, quot, len2);
83     }

```

```
}
```

Implementieren Sie ein gewichtetes Medianfilter. Spezifizieren Sie die Gewichte als konstantes, zweidimensionales Integer-Array. Testen Sie das Filter und vergleichen Sie es mit einem gewöhnlichen Medianfilter.

Listing 12: Gewichteter Median Filter

```
//BVER
// bung 6 - Aufgabe 2.2

4 import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;
9 import java.util.*;

public class GewMedianFilter_ implements PlugInFilter {
    ImagePlus imp;

14 public int setup(String arg, ImagePlus imp) {
    this.imp = imp;
    return DOES_8G;
}

19 private void median(ImageProcessor ip, int[][] gewichte, int hotSpotX, int hotSpotY) {
    final int w = ip.getWidth();
    final int h = ip.getHeight();
    final int fh = gewichte.length;
    final int fw = gewichte[0].length;
24

    ImageProcessor copy = ip.duplicate();
    int gewSum, val, pos, x, y;

    // Summe der Gewichte bestimmen
29    gewSum = 0;
    for (int v = 0; v < fh; v++) {
        for (int u = 0; u < fw; u++) {
            gewSum += gewichte[v][u];
        }
34    }
    int[] values = new int[gewSum];

    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
39        // Anwendung des Filters an der Position u,v
            pos = 0;
            for (int j = 0; j < fh; j++) {
                for (int i = 0; i < fw; i++) {
44                    // Randbehandlung
                    x = u + i - hotSpotX;
                    y = v + j - hotSpotY;
                    if (x < 0) x = -x;
                    if (x >= w) x = 2*w - 1 - x;
49                    if (y < 0) y = -y;
                    if (y >= h) y = 2*h - 1 - y;

                    // Intensität im Array zwischenspeichern
                    for(int k = 0; k < gewichte[j][i]; k++) {
                        values[pos] = copy.getPixel(x, y);
44                    pos++;
                    }
                }
            }
59        // neue Intensität bestimmen
        Arrays.sort(values);
        pos = values.length/2;
        if (values.length%2 == 0) {
```

```

64          // gerade Anzahl: Median als arithmetischer Durchschnitt
65          val = (values[pos - 1] + values[pos])/2;
66      } else {
67          val = values[pos];
68      }
69
70          // clamping
71          if (val < 0)    val = 0;
72          if (val > 255) val = 255;
73
74          // neue Intensität setzen
75          ip.putPixel(u, v, val);
76      }
77  }
78
79  public void run(ImageProcessor ip) {
80      // Gewichtsmatrix
81      int[][] gewichte = {
82          {1, 2, 1},
83          {2, 4, 2},
84          {1, 2, 1}
85      };
86
87      median(ip, gewichte, 1, 1);
88  }
89

```

Programmieren Sie eine Prozedur zur linearen Filterung. Das Bild und die Filtermatrix sollen dabei als Parameter übergeben werden. Zur Behebung der Randproblematik verwenden Sie auf der einen Seite die Absolutwerte von negativen Pixelkoordinaten und auf der andern Seite das Entsprechende.

Listing 13: Lineares Filter

```

//BVER
// bung 6 - Aufgabe 2.1

import ij.*;
5 import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

10 public class LinearesFilter_ implements PlugInFilter {
    ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
15        return DOES_8G;
    }

    private void convolution(ImageProcessor ip, int[][] filter, int quot, int hotSpotX, int
        hotSpotY) {
        final int w = ip.getWidth();
        final int h = ip.getHeight();
20        final int fh = filter.length;
        final int fw = filter[0].length;

        ImageProcessor copy = ip.duplicate();
        int sum, x, y;

        for (int v = 0; v < h; v++) {
            for (int u = 0; u < w; u++) {

25            // Anwendung des Filters an der Position u,v
                sum = 0;
                for (int j = 0; j < fh; j++) {
                    for (int i = 0; i < fw; i++) {
                        // Randbehandlung
30                        x = u + i - hotSpotX;

```

```

        y = v + j - hotSpotY;
        if (x < 0) x = -x;
        if (x >= w) x = 2*w - 1 - x;
        if (y < 0) y = -y;
        if (y >= h) y = 2*h - 1 - y;

40      // neue Intensität berechnen
        sum = sum + filter[j][i]*copy.getPixel(x, y);
    }
}
sum /= quot;

// clamping
if (sum < 0) sum = 0;
if (sum > 255) sum = 255;

// neue Intensität setzen
ip.putPixel(u, v, sum);
}

55 }

public void run(ImageProcessor ip) {
    // filtermatrix H
60    int[][] filter = {
        {3, 5, 3},
        {5, 8, 5},
        {3, 5, 3}
    };
    int quot = 40;

    convolution(ip, filter, quot, 1, 1);
}
}

```

13.6 Uebung 7

Ausgehend von der Implementierung der linearen Faltung (Übung 6, Aufgabe 2.1) können Sie auf einfache Art die Grauwert-Dilation programmieren. Erlauben Sie Strukturmatrizen beliebiger Form, d.h. lassen Sie es zu, dass gewisse Elemente der Strukturmatrix nicht berücksichtigt werden.

Listing 14: Grauwerterosion

```

1 //BVER
// bung 7.1

import ij.*;
import ij.process.*;
6 import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

public class GrauwertDilation_ implements PlugInFilter {
11   ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G;
    }

16    private void dilation(ImageProcessor ip, int[][] struct, int hotSpotX, int hotSpotY) {
        final int w = ip.getWidth();
        final int h = ip.getHeight();
21        final int sh = struct.length;
        final int sw = struct[0].length;

        ImageProcessor copy = ip.duplicate();
        int tmp, max, x, y;

```

```

26
    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {

            // Anwendung des Filters an der Position u,v
            max = -1;
            for (int j = 0; j < sh; j++) {
                for (int i = 0; i < sw; i++) {
                    if (struct[j][i] >= 0) {
                        // Randbehandlung
                        x = u + i - hotSpotX;
                        y = v + j - hotSpotY;
                        if (x < 0) x = -x;
                        if (x >= w) x = 2*w - 1 - x;
                        if (y < 0) y = -y;
                        if (y >= h) y = 2*h - 1 - y;

                        // neue Intensität berechnen
                        tmp = struct[j][i] + copy.get(x, y);
                        if (tmp > max) max = tmp;
                    }
                }
            }
            // clamping
            if (max < 0) max = 0;
            if (max > 255) max = 255;

            // neue Intensität setzen
            ip.putPixel(u, v, max);
        }
    }
}

public void run(ImageProcessor ip) {
    // Strukturmatrix H
    int[][] struct = {
        {-1, 1, -1},
        { 1, 2,  1},
        {-1, 1, -1}
    };
    dilation(ip, struct, 1, 1);
}
}

```

Analog zu Aufgabe 1, jetzt aber für die Grauwert-Erosion.

Listing 15: Grauwertdilation

```

1 //BVER
// bung 7.2

import ij.*;
import ij.process.*;
6 import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.*;

public class GrauwertErosion_ implements PlugInFilter {
11   ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_8G;
    }

    private void erosion(ImageProcessor ip, int[][] struct, int hotSpotX, int hotSpotY) {
21      final int w = ip.getWidth();
      final int h = ip.getHeight();
      final int sh = struct.length;
      final int sw = struct[0].length;

```

```

        ImageProcessor copy = ip.duplicate();
        int tmp, min, x, y;
26
        for (int v = 0; v < h; v++) {
            for (int u = 0; u < w; u++) {

                // Anwendung des Filters an der Position u,v
                min = Integer.MAX_VALUE;
                for (int j = 0; j < sh; j++) {
                    for (int i = 0; i < sw; i++) {
                        if (struct[j][i] >= 0) {
                            // Randbehandlung
                            x = u + i - hotSpotX;
                            y = v + j - hotSpotY;
                            if (x < 0) x = -x;
                            if (x >= w) x = 2*w - 1 - x;
                            if (y < 0) y = -y;
                            if (y >= h) y = 2*h - 1 - y;

                            // neue Intensität berechnen
                            tmp = copy.get(x, y) - struct[j][i];
                            if (tmp < min) min = tmp;
46
                        }
                    }
                }
                // clamping
                if (min < 0) min = 0;
51
                if (min > 255) min = 255;

                // neue Intensität setzen
                ip.putPixel(u, v, min);
            }
56
        }
    }

    public void run(ImageProcessor ip) {
        // Strukturmatrix H
61
        int[][] struct = {
            {-1, 1, -1},
            {1, 2, 1},
            {-1, 1, -1}
        };
66
        erosion(ip, struct, 1, 1);
    }
}

```

13.7 Uebung 8

Gegeben ist das Graustufenbild coins.png. Schreiben Sie ein Programm, welches die Anzahl der Münzen bestimmt und jede Münze mit einer eigenen Farbe einfärbt.

- Beachten Sie, dass das Graustufenbild zuerst binarisiert und allfällige Störungen eliminiert werden sollen, bevor Sie das Flood Filling Verfahren anwenden können. Schreiben Sie eine Methode, die das Bild und einen Schwellwert als Input haben und ein neues Binärbild generiert.
- Verwenden Sie für das anschliessende Flood Filling ein effizientes Verfahren, welches auch für grössere Bilder geeignet ist.
- Erzeugen Sie ein neues Falschfarbenbild, welches den Hintergrund schwarz und die Münzen in Falschfarben darstellt.

Listing 16: Region Labeling

```

1 import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*;

```

```

import java.util.*;
6 import java.util.Random;

public class RegionLabeling_ implements PlugInFilter {

    class Node {
11        int x, y;
        Node(int x, int y)
        {
            this.x = x;
            this.y = y;
16        }
    };

    public int setup(String arg, ImagePlus img)
    {
21        return DOES_8G + NO_CHANGES;
    }

    // Recursive implementation, just for small images.
    public void floodFillRecursive(ImageProcessor image, int u, int v, int regionLabel)
26    {
        if(u >= 0 && u < image.getWidth() && v >= 0 && v < image.getHeight() && image.get(u, v)
           == 0) {
            image.set(u, v, regionLabel);

            floodFill(image, u+1, v, regionLabel);
            floodFill(image, u, v+1, regionLabel);
            floodFill(image, u-1, v, regionLabel);
            floodFill(image, u, v-1, regionLabel);

        }
36    }

    // Breath first implementation with queue
    public void floodFill(ImageProcessor image, int u, int v, int regionLabel)
    {
41        final int M = image.getWidth();
        final int N = image.getHeight();
        Deque<Node> q = new LinkedList<Node>();

        q.addFirst(new Node(u, v));
46        while (!q.isEmpty()) {
            Node n = q.removeLast();
            if((n.x >= 0) && (n.x < M) && (n.y >= 0) && (n.y < N) && (image.get(n.x, n.y) == 0) )
            {
                image.set(n.x, n.y, regionLabel);

                q.addFirst(new Node(n.x+1, n.y));
                q.addFirst(new Node(n.x, n.y+1));
                q.addFirst(new Node(n.x, n.y-1));
                q.addFirst(new Node(n.x-1, n.y));
            }
56        }
    }

    public ByteProcessor convertToBinaryImage(ImageProcessor orig, int threshold)
    {
61        int M = orig.getWidth();
        int N = orig.getHeight();
        ByteProcessor image = new ByteProcessor(M, N);

        for(int v=0; v < N; v++) {
            for(int u = 0; u<M; u++) {
                if (orig.get(u, v) < threshold) {
                    image.set(u, v, 1);
                } else {
                    image.set(u, v, 0);
                }
            }
        }
71        return image;
    }
}

```

```

    }

76   public void run(ImageProcessor orig)
{
    int M = orig.getWidth();
    int N = orig.getHeight();

81   final int threshold = 100;
    int regionLabel = 2;

    // convert to binary image
86   ByteProcessor image = convertToBinaryImage(orig, threshold);

    // find and label regions
91   for(int v = 0; v < N; v++) {
        for(int u = 0; u < M; u++) {
            if (image.get(u, v) == 0) {
                floodFill(image, u, v, regionLabel);
                regionLabel++;
            }
        }
    }

96 }

    // create color table
Random random = new Random();
Color[] table = new Color[regionLabel - 1];
101  table[0] = Color.BLACK;
    for(int i=1; i < table.length; i++) {
        table[i] = new Color(random.nextInt(256), random.nextInt(256), random.nextInt(256));
    }

106 // create false-color image
int[] rgb = new int[3];
ImageProcessor color = new ColorProcessor(M, N);
for(int v = 0; v < N; v++) {
    for(int u = 0; u < M; u++) {
        Color c = table[image.get(u, v) - 1];
        color.set(u, v, c.getRGB());
    }
}
111 ImagePlus im2 = new ImagePlus("Output", color);
im2.show();
116 }
}

```

13.8 Uebung 10

a) Programmieren Sie die Kantendetektierung mittels verbessertem Sobel-Filtern:

$$H_x = \frac{1}{32} \begin{pmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{pmatrix} \text{ und } H_y = \frac{1}{32} \begin{pmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{pmatrix}$$

Geben Sie die Kantenstärke sowie die Kantenrichtung als Bilder aus.

Wenden Sie Ihren Kantendetektor auf windisch.png an.

b) Färben Sie die Kanten anhand ihrer Winkel ein und geben Sie sie in einem Bild aus. Wählen Sie zum Beispiel die Hue-Komponente des HSV oder HLS-Farbraumes für den winkelabhängigen Farbton.

Wenden Sie das angepasste Verfahren auf coins.png an.

Listing 17: Corner

```

import ij.process.ByteProcessor;
2
public class Corner implements Comparable{
    int u;
    int v;

```

```

float q;
7
Corner(int u, int v, float q){
    this.u = u;
    this.v = v;
    this.q = q;
12}
13
public int compareTo(Object obj){
    Corner c2 = (Corner) obj;
    if (this.q > c2.q) return -1;
17
    if (this.q < c2.q) return 1;
    else return 0;
}
18
double dist2 (Corner c2){
22
    int dx = this.u - c2.u;
    int dy = this.v - c2.v;
    return (dx * dx) + (dy * dy);
}
23
24
void draw(ByteProcessor ip){
    int paintvalue = 0;
    int size = 2;
    ip.setValue(paintvalue);
    ip.drawLine(u-size, v, u+size, v);
28
    ip.drawLine(u, v-size, u, v+size);
}
29
}

```

Vervollständigen Sie das ImageJ Plugin HarrisCornerPlugin_.java indem Sie die Corner Response Funktion makeCrf() der Klasse HarrisCornerDetector ausprogrammieren.

Adaptieren Sie zudem die Zeichnungsmethode der Eckpunkte, um die Eckpunktstärke graphisch darzustellen.

Listing 18: Harris Corner Detection

```

1 import ij.IJ;
import ij.ImagePlus;
import ij.plugin.filter.Convolver;
import ij.process.Blitter;
import ij.process.ByteProcessor;
6 import ij.process.FloatProcessor;
import ij.process.ImageProcessor;

import java.util.Arrays;
import java.util.Collections;
11 import java.util.Iterator;
import java.util.Vector;

public class HarrisCornerDetector {
    public static final float DEFAULT_ALPHA = 0.050f;
16    public static final int DEFAULT_THRESHOLD = 20000;

    float alpha = DEFAULT_ALPHA;
    int threshold = DEFAULT_THRESHOLD;
    double dmin = 10;
21
    final int border = 20;

    final float [] pfilt = {0.223755f, 0.552490f, 0.223755f};
    final float [] dfilt = {0.453014f, 0.0f, -0.453014f};
26    final float [] bfilt = {0.01563f, 0.09375f, 0.234375f, 0.3125f,
        0.234375f, 0.09375f, 0.01563f};
        // = {1, 6, 15, 20, 15, 6, 1}/64

    ImageProcessor ipOrig;
31    FloatProcessor A;
    FloatProcessor B;
    FloatProcessor C;
    FloatProcessor Q;

```

```

36     Vector corners;

    HarrisCornerDetector(ImageProcessor ip){
        this.ipOrig = ip;
    }

41    HarrisCornerDetector(ImageProcessor ip, float alpha, int threshold){
        this.ipOrig = ip;
        this.alpha = alpha;
        this.threshold = threshold;
    }

46 }

    void findCorners(){
        makeDerivatives();
        makeCrf();
        corners = collectCorners(border);
        corners = cleanupCorners(corners);
    }

    void makeDerivatives(){
56        FloatProcessor Ix = (FloatProcessor) ipOrig.convertToFloat();
        FloatProcessor Iy = (FloatProcessor) ipOrig.convertToFloat();

        Ix = convolveIh(convolveIh(Ix, pfilt), dfilt);
        Iy = convolveIv(convolveIv(Iy, pfilt), dfilt);

61        A = sqr((FloatProcessor) Ix.duplicate());
        A = convolve2(A, bfilt);

        B = sqr((FloatProcessor) Iy.duplicate());
        B = convolve2(B, bfilt);

        C = mult((FloatProcessor) Ix.duplicate(), Iy);
        C = convolve2(C, bfilt);
    }

71    void makeCrf(){      // Corner Response Function
        int w = ipOrig.getWidth();
        int h = ipOrig.getHeight();
        Q = new FloatProcessor(w, h);

76        float[] Apix = (float[]) A.getPixels();
        float[] Bpix = (float[]) B.getPixels();
        float[] Cpix = (float[]) C.getPixels();
        float[] Qpix = (float[]) Q.getPixels();

81        for (int v = 0; v < h; v++){
            for (int u = 0; u < w; u++){
                int i = v * w + u;
                float a = Apix[i];
                float b = Bpix[i];
                float c = Cpix[i];
                // Determinante berechnen
                // Spur berechnen
                // Corner strength Qpix berechnen
            }
        }

91        //////////// Lschen
        float det = a*b - c*c;
        float trace = a+b;
        Qpix[i] = det - alpha * (trace * trace);
    }

96    //////////// Ende Lschen
}

Vector collectCorners(int border){
    Vector<Corner> cornerList = new Vector<Corner>(1000);
    int w = Q.getWidth();
    int h = Q.getHeight();

106    float[] Qpix = (float[]) Q.getPixels();
}

```

```

        for (int v = border; v<h-border; v++){
            for (int u = border; u<w-border; u++){
                float q = Qpix[v*w + u];
                if (q>threshold && isLocalMax(Q, u, v)){
                    Corner c = new Corner(u, v, q);
                    cornerList.add(c);
                }
            }
        }
    }
    Collections.sort(cornerList);
    return cornerList;
}

Vector cleanupCorners(Vector corners){
    double dmin2 = dmin * dmin;
    Object[] cornerArray = corners.toArray();
    Vector<Corner> goodCorners = new Vector<Corner>(corners.size());
    for (int i=0; i<cornerArray.length; i++){
        if (cornerArray[i] != null){
            Corner c1 = (Corner) cornerArray[i];
            goodCorners.add(c1);
            for (int j = i+1; j<cornerArray.length; j++){
                if (cornerArray[j] != null){
                    Corner c2 = (Corner) cornerArray[j];
                    if (c1.dist2(c2)<dmin2){
                        cornerArray[j] = null;
                    }
                }
            }
        }
    }
    return goodCorners;
}
void printCornerPoints(Vector crf){
    Iterator it = crf.iterator();
    for (int i=0; it.hasNext(); i++){
        Corner ipt = (Corner) it.next();
        IJ.write(i + ":" + (int)ipt.q + " " + ipt.u + " " + ipt.v);
    }
}

ImageProcessor showCornerPoints(ImageProcessor ip){
    ByteProcessor ipResult = (ByteProcessor)ip.duplicate();

    int[] lookupTable = new int[256];
    for (int i = 0; i< 256; i++){
        lookupTable[i] = 128+(i/2);
    }
    ipResult.applyTable(lookupTable);

    Iterator it = corners.iterator();
    for (int i=0; it.hasNext(); i++){
        Corner c = (Corner) it.next();
        c.draw(ipResult);
    }
    return ipResult;
}

void showProcessor (ImageProcessor ip, String title){
    ImagePlus win = new ImagePlus(title, ip);
    win.show();
}

void dummy(){
    Corner[] cornerArray = new Corner[100];
    cornerArray[0] = new Corner(10, 20, 0.7f);
    cornerArray[2] = new Corner(10, 20, 1.7f);
    Arrays.sort(cornerArray);
}

```

```

static FloatProcessor convolve1h(FloatProcessor p, float[] h){
    Convolver conv = new Convolver();
    conv.setNormalize(false);
    conv.convolve(p, h, 1, h.length);
    return p;
}

186 static FloatProcessor convolve1v(FloatProcessor p, float[] h){
    Convolver conv = new Convolver();
    conv.setNormalize(false);
    conv.convolve(p, h, h.length, 1);
    return p;
}

191 }

static FloatProcessor convolve2(FloatProcessor p, float[] h){
    convolve1h(p, h);
    convolve1v(p, h);
    return p;
}

196 }

static FloatProcessor sqr(FloatProcessor fp1){
    fp1.sqr();
    return fp1;
}

201 }

static FloatProcessor mult(FloatProcessor fp1, FloatProcessor fp2){
    int mode = Blitter.MULTIPLY;
    fp1.copyBits(fp2, 0, 0, mode);
    return fp1;
}

206 }

static boolean isLocalMax(FloatProcessor fp, int u, int v){
    int w = fp.getWidth();
    int h = fp.getHeight();
    if (u <= 0 || u >= w-1 || v <= 0 || v >= h-1){
        return false;
    }
    else {
        float[] pix = (float[]) fp.getPixels();
        int i0 = (v-1)*w+u;
        int i1 = v*w+u;
        int i2 = (v+1)*w+u;
        float cp = pix[i1];
        return
            cp >= pix[i0-1] && cp >= pix[i0] && cp >= pix[i0+1] &&
            cp >= pix[i1-1] && cp > pix[i1+1] &&
            cp >= pix[i2-1] && cp >= pix[i2] && cp >= pix[i2+1];
    }
}
211 }

216 }

221 }

226 }
}

```

Listing 19: Harris Corner Plugin

```

1 import ij.IJ;
import ij.ImagePlus;
import ij.gui.GenericDialog;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

6 public class HarrisCornerPlugin_ implements PlugInFilter{
    ImagePlus imp;
    static float alpha = HarrisCornerDetector.DEFAULT_ALPHA;
    static int threshold = HarrisCornerDetector.DEFAULT_THRESHOLD;
11    static int nmax = 0;

    public int setup(String arg, ImagePlus imp){
        IJ.register(HarrisCornerPlugin_.class);
        this.imp = imp;
        if (arg.equals("about")){
16

```

```

        showAbout();
        return DONE;
    }
    return DOES_8G + NO_CHANGES;
}

21 }

public void run(ImageProcessor ip){
    if (!showDialog()) return;
    HarrisCornerDetector hcd = new HarrisCornerDetector(ip, alpha, threshold);
26    hcd.findCorners();
    ImageProcessor result = hcd.showCornerPoints(ip);
    ImagePlus win = new ImagePlus("Corners from " + imp.getTitle(), result);
    win.show();
}

31 void showAbout(){
    String cn = getClass().getName();
    IJ.showMessage("About " + cn+ " ...", "Harris Corner Detector");
}

36 private boolean showDialog() {
    GenericDialog dlg = new GenericDialog
        ("Harris Corner Detector", IJ.getInstance());

41     float def_alpha = HarrisCornerDetector.DEFAULT_ALPHA;
    dlg.addNumericField("Alpha (default; "+def_alpha+")", alpha, 3);

46     int def_threshold = HarrisCornerDetector.DEFAULT_THRESHOLD;
    dlg.addNumericField("Thresholt (default; "+def_threshold+")", threshold, 0);
    dlg.addNumericField("Max. points (0 = show all)", nmax, 0);
    dlg.showDialog();

51     if(dlg.wasCanceled()) return false;
    if(dlg.invalidNumber()){
        IJ.showMessage("Error", "Invalid input number");
        return false;
    }
    alpha = (float) dlg.getNextNumber();
56    threshold = (int) dlg.getNextNumber();
    nmax = (int) dlg.getNextNumber();

    return true;
}
61 }

```
