

BFH-TI: Fachbereich Informatik
Vertiefung: Computer Perception & Virtual Reality

Skript

Grundlagen der Computer Vision



Copyright © 2007-2016
Marcus Hudritsch
Kirchraint 18
CH-2572 Sutz
Tel.: 076 405 15 00
marcus.hudritsch@bfh.ch

Inhaltsverzeichnis

1 Überblick.....	11
1.1 Definition von Computer Vision (CV).....	11
1.2 Literaturempfehlungen.....	12
2 Grundlagen.....	13
2.1 Grundlagen der visuellen Wahrnehmung.....	13
2.1.1 Das Licht.....	13
2.1.1.1 Strahlenmodell.....	13
2.1.1.2 Photonenmodell.....	15
2.1.1.3 Wellenmodell.....	17
2.1.2 Das Auge.....	20
2.1.2.1 Retina.....	21
2.1.2.2 Farbsehen.....	23
2.1.2.3 Tristimulustheorie.....	23
2.1.2.4 Gegenfarben-Theorie.....	24
2.1.2.5 Kontrastempfindlichkeit.....	25
2.2 Farbmodelle.....	26
2.2.1 RGB-Farbmodell.....	26
2.2.2 CMY(K)-Farbmodell.....	26
2.2.3 HSV-Farbmodell.....	28
2.2.4 HSL-, HSB- und HSI-Farbmodelle.....	29
2.2.5 YUV- und YCbCr-Farbmodelle.....	29
2.2.6 CNS-Farbmodell.....	31
2.3 Werkzeuge für die Computer Vision.....	32
2.3.1 ImagePlay.....	32
2.3.2 Matlab.....	33
2.3.3 ImageJ.....	33
2.3.3.1 Eingebaute ImageJ Werkzeuge.....	34
2.3.3.2 Plugins für ImageJ.....	34
2.3.3.3 Entwicklungsumgebungen für Plugins.....	35
2.3.3.4 Fiji .. 35	
2.4 Übungen zum Kapitel Grundlagen.....	35
3 Bildaufnahme.....	37
3.1 Optik.....	37
3.1.1 Camera Obscura.....	37
3.1.2 Objektivkennzahlen.....	37
3.1.2.1 Brennweite und Abbildungsmassstab.....	37
3.1.2.2 Blende und Schärfentiefe.....	38
3.1.3 Abbildungsfehler (Aberration).....	39
3.1.4 Objektivtypen.....	41
3.1.5 Objektivgewinde.....	41
3.2 Bildsensor.....	42
3.2.1 Sensorgröße und Auflösung.....	42
3.2.2 Spektrale Empfindlichkeit.....	42
3.2.3 Sensorkennzahlen.....	43

3.2.4 Rauschquellen.....	44
3.2.4.1 Defekte Pixel.....	44
3.2.4.2 Sensorrauschen (Noise).....	44
3.2.4.3 Blooming.....	45
3.2.5 Dunkelbild- und Weissbildkorrektur	45
3.2.6 Signal to Noise Ratio (SNR).....	46
3.2.7 CCD-Bildsensor.....	47
3.2.8 CMOS-Bildsensor.....	49
3.2.9 Farbaufnahmeprinzipien.....	50
3.2.9.1 3-CCD-Verfahren.....	50
3.2.9.2 Bayermusterfilter.....	51
3.2.9.3 Multilayer-Farbsensor.....	52
3.3 Kameraschnittstellen.....	53
3.4 Übung zum Kapitel Billaufnahme mit ImageJ.....	54
 4 Statistische Bilddauswertung.....	56
 4.1 Histogramm.....	56
4.1.1 Wahrscheinlichkeitsdichtefunktion.....	57
4.1.2 Kumulatives Histogramm.....	57
4.1.3 Mehrdimensionales Histogramm.....	58
 4.2 Statistische Kennzahlen.....	59
4.2.1 Minimum und Maximum.....	59
4.2.2 Dynamik.....	59
4.2.3 Kontrast.....	59
4.2.4 Mittelwert (Mean).....	60
4.2.5 Modalwert (Mode).....	60
4.2.6 Standardabweichung (Standard Deviation).....	60
4.2.7 Schiefe (Skewness).....	60
4.2.8 Wölbung (Kurtosis).....	60
4.2.9 Entropie (Entropy).....	60
4.2.10 Vergleiche.....	61
 5 Punktoperatoren.....	62
 5.1 Lineare und nichtlineare Abbildungsfunktionen.....	63
 5.2 Invertierung.....	64
 5.3 Binarisierung.....	64
 5.4 Grauwertreduktion.....	65
 5.5 Lineare Helligkeitskorrektur.....	65
 5.6 Nichtlineare Helligkeitskorrektur, Gammakorrektur.....	66
 5.7 Kontrastkorrektur.....	68
5.7.1 Lineare Kontrastkorrektur.....	68
5.7.2 Automatische Kontrasterhöhung.....	69
5.7.3 Kontrasterhöhung durch Histogrammausgleich.....	69
5.7.4 Kontrasterhöhung durch Farben.....	71
 5.8 Bildarithmetik.....	72
 5.9 Logische Operationen.....	74
 5.10 Punktoperationen in ImageJ-Plug-ins.....	75

5.10.1 Standard-Punktoperationen.....	75
5.10.2 Punktoperation mit Lookup-Table.....	75
5.10.3 Punktoperationen mit zwei Bildern.....	76
5.11 Übung: Punktoperationen mit Matlab.....	77
6 Lokale Operatoren.....	78
6.1 Lineare Faltung.....	79
6.1.1 Eigenschaften der Faltung.....	80
6.1.1.1 Separierbarkeit von Filtern.....	80
6.1.2 Tiefpassfilter.....	81
6.1.2.1 Rechteckfilter.....	81
6.1.2.2 Gaussfilter.....	81
6.1.2.3 Übung: Faltungsfilter mit Excel.....	83
6.1.3 Hochpassfilter.....	83
6.1.3.1 Sobelfilter.....	86
6.1.3.2 Laplace Filter als 2. Ableitung.....	87
6.1.3.3 Laplace of Gaussian Filter.....	89
6.1.4 Punktspreizfunktion (PSF).....	89
6.1.5 Anwendung Bildschärfung.....	90
6.1.6 Faltung in ImageJ.....	92
6.1.7 Übung: Lokale Operationen mit Matlab.....	93
6.2 Morphologische Operatoren.....	94
6.2.1 Morphologische Operatoren auf Binärbildern.....	94
6.2.1.1 Binäre Dilatation.....	94
6.2.1.2 Binäre Erosion.....	95
6.2.1.3 Eigenschaften von Dilatation und Erosion.....	96
6.2.1.4 Binäres Closing	96
6.2.1.5 Binäres Opening.....	96
6.2.1.6 Eigenschaften von Opening und Closing.....	97
6.2.1.7 Binäre Kantenextraktion.....	97
6.2.1.8 Hit-or-Miss-Operator.....	98
6.2.1.9 Schnelle Implementation durch Bildverschiebung.....	99
6.2.2 Morphologische Operatoren auf Graustufenbildern.....	100
6.2.2.1 Maximum- und Minimumfilter.....	101
6.2.2.2 Medianfilter.....	102
6.2.2.3 Closing und Opening auf Graustufenbildern.....	103
6.2.2.4 Kantenextraktion auf Graustufenbilder.....	104
6.2.2.5 Bot-Hat- und Top-Hat-Operation.....	104
6.2.3 Morphologische Operatoren in ImageJ.....	105
6.2.4 Morphologische Operatoren in Matlab.....	105
6.2.5 Übung zu morphologischen Operationen.....	106
7 Globale Operatoren.....	107
7.1 Fourier-Transformation.....	107
7.1.1 Sinus- und Kosinusfunktionen als Kreisbewegungen.....	108
7.1.2 1D-Fourier-Transformation.....	110
7.1.3 Übung 1D-Fourier-Transformation mit Excel.....	112
7.1.4 2D-Fourier-Transformation.....	114
7.1.4.1 Interpretation des Amplitudenspektrums.....	116
7.1.5 Anwendungen.....	118
7.1.5.1 Filterung im Frequenzraum.....	118

7.1.5.2 Inverse Filterung im Frequenzraum.....	119
7.1.5.3 Direkte Manipulation im Amplitudenspektrum.....	121
7.1.6 Fourier Transformation in ImageJ.....	122
7.1.7 Fourier Transformation in Matlab.....	122
7.1.8 Übung: 2D-Fourier-Transformation mit Matlab und ImageJ.....	123
7.2 Wavelet-Transformation.....	125
7.2.1 Probleme der Fourier-Transformation.....	125
7.2.2 Wavelets.....	126
7.2.3 Kontinuierliche Wavelet-Transformation.....	127
7.2.4 Diskrete Wavelet-Transformation.....	128
7.2.5 Haar-Transformation.....	129
7.2.6 Anwendungen der Wavelet-Transformation.....	132
7.2.7 Übung Wavelet.....	133
7.3 Hough-Transformation.....	134
7.3.1 Hough-Transformation für Geraden.....	134
7.3.1.1 Bildung des Hough-Raums.....	135
7.3.1.2 Geraden im Hough-Raum bestimmen.....	136
7.3.2 Hough-Transformation für Kreise.....	136
7.3.3 Hough-Transformation für Ellipsen.....	137
7.3.4 Hough-Transformation in Matlab.....	138
7.3.5 Übungen zur Hough-Transformation.....	138
7.4 Hauptkomponententransformation.....	139
7.4.1 Varianz, Kovarianz und Kovarianzmatrix.....	140
7.4.2 Eigenvektoren und Eigenwerte.....	142
7.4.3 Einfaches Beispiel mit Matlab.....	143
7.4.4 Anwendungen in der Bildverarbeitung und Computergrafik.....	147
7.4.4.1 Bestimmung der Ausrichtung von 2D- und 3D-Daten.....	147
7.4.4.2 Reduktion der Dimensionalität.....	148
7.4.4.3 Statistische Klassifikation.....	149
7.4.4.4 Mustererkennung.....	149
8 Segmentierung.....	150
8.1 Schwellwertbasierte Segmentierung.....	150
8.1.1 Globaler Schwellwert.....	150
8.1.1.1 Mittelwert.....	151
8.1.1.2 k-Means-Clustering.....	151
8.1.1.3 Otsu's optimale Methode.....	151
8.1.1.4 Prozentmethode.....	151
8.1.1.5 Globale Schwellwerte in ImageJ und Matlab	152
8.1.2 Lokaler Schwellwert	152
8.1.3 Tiefpassfilterung vor Schwellwertsegmentierung.....	153
8.2 Regionenbasierte Segmentierung.....	153
8.2.1 Region Growing.....	154
8.2.2 Split and Merge.....	155
8.3 Kantenbasierte Segmentierung.....	156
8.3.1 Gradientenbasierte Kantenfilter.....	157
8.3.2 Morphologische Kantenfilter.....	157
8.3.3 Canny Kantendetektor.....	157
8.3.4 Übung zu Kantendetektoren.....	161

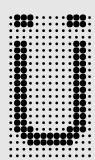
8.4 Modellbasierte Segmentierung.....	161
8.4.1 Hough Transformation.....	161
8.4.2 Aktive Konturen.....	161
8.4.2.1 Parameterkurve.....	162
8.4.2.2 Energiefunktion.....	163
8.4.2.3 Algorithmus von Williams und Shah.....	164
8.4.2.4 Snake Test Applikation in C#	165
8.4.2.5 Übung Snakes.....	167
8.4.2.6 Snakes in ImageJ und Matlab	167
8.5 Bewegungsbasierte Segmentation.....	168
8.5.1 Bewegung vor statischem Hintergrund.....	168
9 Repräsentation.....	169
 9.1 Region Labeling.....	169
 9.2 Masken.....	170
 9.3 Lauflängen.....	171
 9.4 Baum- und Graphstrukturen.....	171
 9.5 Konturen.....	171
9.5.1 Chain Code.....	171
9.5.2 Crack Code.....	172
9.5.3 Signaturen.....	173
9.5.4 Fourier Deskriptoren.....	175
 9.6 Skelett.....	178
9.6.1 Distanztransformation.....	178
9.6.2 Ausdünnung (Thinning).....	179
10 Merkmale.....	182
 10.1 Geometrische Merkmale.....	183
10.1.1 Breite und Höhe.....	183
10.1.2 Perimeter.....	183
10.1.3 Fläche.....	183
10.1.4 Kompaktheit oder Kreisförmigkeit.....	184
10.1.5 Konvexität, Dichte und Durchmesser.....	185
 10.2 Statistische Merkmale.....	186
10.2.1 Histogrammbasierte Merkmale.....	186
10.2.2 Momentbasierte Merkmale.....	186
10.2.2.1 Zentrale Momente (translationsinvariant).....	187
10.2.2.2 Normalisierte zentrale Momente (skalierungsinvariant).....	188
10.2.2.3 Invariante Momente (rotationsinvariant).....	189
 10.3 Topologische Merkmale.....	190
10.3.1 Anzahl Löcher.....	191
 10.4 Texturmerkmale.....	192
10.4.1 Statistische Texturmerkmale.....	192
10.4.1.1 Grauwertmatrix (Co-occurrence Matrix).....	193
10.4.2 Spektrale Texturmerkmale.....	194
 10.5 Punktmerkmale.....	195
10.5.1 Harris Corners.....	196
10.5.1.1 Lokale Strukturmatrix.....	196

10.5.1.2 Corner Response Function.....	197
10.5.2 SIFT-Deskriptoren	199
10.5.2.1 Detektion der Extremwerte im Multiskalenraum.....	199
10.5.2.2 Reduktion der Extrempunkte zu Keypoints.....	201
10.5.2.3 Bestimmung der Orientierung der Keypoints.....	202
10.5.2.4 Bestimmung eines Deskriptors für jeden Keypoint.....	203
10.5.3 SIFT-Nachfahren.....	204
10.5.3.1 Deskriptoren in Matlab.....	204
10.5.3.2 Deskriptoren in Fiji.....	204
10.6 Beispiel & Übung.....	205
10.6.1 In ImageJ mit ijblob.....	205
10.6.2 Übung in Matlab: Merkmale mit regionprops.....	206
10.6.3 Übung in ImageJ: Hu's invariante Momente.....	206
10.6.4 Übung: Rotationsinvarianz von Harris Corners.....	206
10.6.5 Übung: Skalierungsinvarianz von SIFT-Punktmerkmalen.....	206
11 Klassifikation.....	207
11.1 Merkmale und Klassen.....	207
11.1.1 Merkmale und Merkmalsraum.....	207
11.1.1.1 Reduktion der Merkmale mit der Hauptkomponentenanalyse.....	210
11.1.1.2 Gütekriterium für ein Merkmal.....	213
11.1.2 Bestimmung der Klassen.....	214
11.1.2.1 k-Means Clustering.....	214
11.2 Klassifikatoren.....	216
11.2.1 Distanzklassifikatoren.....	216
11.2.1.1 Minimale Distanz zum Cluster-Zentrum.....	216
11.2.1.2 Minimale Distanz zu k Nachbarn.....	219
11.2.2 Statistische Klassifikatoren.....	230
11.2.2.1 Bayes-Klassifikator.....	230
11.2.2.2 Mahalanobis-Distanz zum Cluster-Zentrum.....	230
11.2.3 Beurteilung von Klassifikatoren.....	234
11.3 Template Matching.....	236
11.3.1 Hohe Musterähnlichkeit durch kleine Differenz.....	236
11.3.1.1 Block Matching Algorithmen (BMA).....	236
11.3.2 Hohe Musterähnlichkeit durch grosse Korrelation.....	238
11.3.2.1 Beispiel mit normierter Kreuzkorrelation.....	239
11.3.2.2 Beispiel mit nicht normierter Korrelation.....	240
11.4 Übungen.....	242
11.4.1 Übung: k-Means Clustering.....	242
11.4.2 Übung: Eigenfaces.....	242
11.4.3 Übung: Face Recognition Contest.....	242
12 Bild- und Videokompression.....	243
12.1 Signaltheorie.....	243
12.1.1 Digitalisierung des Bildsignals.....	243
12.1.1.1 Nyquist-Frequenz.....	243
12.1.1.2 Shannon Theorem.....	244
12.1.2 Informationstheorie.....	244
12.1.3 Bildkompression = Datenreduktion.....	246
12.2 Verlustfreie Kompression.....	246

12.2.1 Run Length Encoding (RLE).....	246
12.2.2 Huffman-Code.....	247
12.2.3 LZ-Kodierung.....	249
12.2.4 Übungen verlustfreie Kompression.....	251
12.3 Verlustbehaftete Kompression.....	252
12.3.1 JPEG-Kompression.....	252
12.3.1.1 Schritt 1: Konvertierung ins YCrCb-Format.....	252
12.3.1.2 Schritt 2: Diskrete Kosinustransformation.....	253
12.3.1.3 Schritt 3: Quantisierung der DCT-Koeffizienten.....	255
12.3.1.4 Schritt 4. Zick-Zack-Codierung der Koeffizienten.....	256
12.3.1.5 Schritt 5. Kompression mit Lauflängen- und Entropiecodierung.....	256
12.3.1.6 JPEG Dekompression.....	257
12.3.1.7 Schwächen von JPEG.....	258
12.3.2 JPEG2000-Kompression.....	259
12.3.3 Neuere Formate.....	262
12.3.3.1 WebP.....	262
12.3.3.2 BPG.....	262
12.3.4 Übung DCT mit Excel.....	263
12.4 Videokompression.....	266
12.4.1 Grundbegriffe.....	266
12.4.1.1 Kompressionsstandard.....	266
12.4.1.2 Video-Codec.....	266
12.4.1.3 Video-Formate.....	266
12.4.1.4 Video-Player.....	266
12.4.2 Ziele der Videokompression.....	267
12.4.3 Kompression als Reduktion der Redundanz.....	267
12.4.3.1 Zeitliche Redundanz.....	268
12.4.4 3D-Transformkodierung.....	268
12.4.5 Bewegungskompensation: MPEG.....	269
12.4.6 MPEG-1.....	269
12.4.6.1 Schichtenmodell.....	270
12.4.6.2 Bildtypen.....	271
12.4.6.3 Bewegungskompensation.....	272
12.4.6.4 Sequenzanalyse.....	273
12.4.6.5 Übung MPEG-1.....	275
12.4.7 MPEG-2.....	275
12.4.7.1 Profiles & Levels.....	276
12.4.8 MPEG-4.....	276
12.4.8.1 MPEG-4 Szenengraph.....	276
12.4.8.2 MPEG-4 Visual Standard.....	278
12.4.8.3 MPEG-4 Profile.....	281
12.4.9 MPEG-4/AVC, H.264.....	282
12.4.9.1 Verbesserungen zum MPEG-4 Part-2 Standard.....	282
12.4.9.2 H.264-Profile.....	283
12.4.9.3 H.264-Level.....	283
12.4.10 Codec-Vergleich.....	284
13 Anhänge.....	286
13.1 Masseinheiten.....	286
13.2 Griechisches Alphabet.....	286
13.3 Einheiten des internationalen Einheitssystems.....	287

14 Stichwortverzeichnis.....	288
15 Literaturverzeichnis.....	291

1 Überblick



berblick: In diesem Kapitel erfahren Sie, was Sie in diesem Skript erwarten dürfen und worum es in der Computer Vision (CV) generell geht. Das Skript ist grob in fünf Bereiche unterteilt: Die **Kapitel 1 und 2** dienen der Einführung und der Behandlung einiger Grundlagen. Zu Letzterem gehören grundlegende Kenntnisse über Licht, Auge und Farbmodelle. Zum Schluss werden einige Werkzeuge vorgestellt. Das **Kapitel 3** behandelt die digitale Bildaufnahme, beginnend bei der Optik bis zu den verschiedenen Kameraschnittstellen zum Computer. Der Hauptteil dreht sich um digitale Bildsensoren und ihren Eigenschaften. Das **Kapitel 4-7** führt die grundlegenden Operationen ein, die auf einem Bild durchgeführt werden können. Nachdem wir die wichtigsten statistischen Kennzahlen kennengelernt haben, unterteilen wir das Kapitel nach deren Einflussbereichen in punktbezogene, lokale und globale Operatoren. Das **Kapitel 8-11** behandelt die klassischen Bereiche der Bildanalyse, die Segmentierung, die Repräsentationen, die Merkmalsextraktion und die Klassifikation. Im **Kapitel 12** behandeln wir zum Abschluss die Kompression von Bild- und Videodaten. Nach einigen Grundlagen der Signaltheorie lernen Sie, wie man Bilder verlustfrei oder verlustbehaftet komprimieren kann. Wenn wir viele ähnliche Bilder hintereinander haben, wie das beim Video der Fall ist, so können wir zusätzlich die zeitliche Redundanz zu einem grossen Teil eliminieren.

1.1 Definition von Computer Vision (CV)

Gemäss der englischen Wikipedia ist die Computer Vision:

*„Computer vision is a field that includes methods for **acquiring, processing, analyzing, and understanding images** and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.“*

Es geht also um die Aufnahme, Verarbeitung und Analyse von digitalen Bildern sowie der Gewinnung von Informationen aus diesen Bildern. Die Abgrenzung der Untergebiete *Bildverarbeitung*, *Bildanalyse* und *Bildsynthese* (*Computergrafik*) kann anhand des Prozessmodells gemacht werden. Ein Prozess wird dabei durch seine Eingabe (Input) und seine Ausgabe (Output) bestimmt:

- Ist sowohl die Eingabe als auch die Ausgabe ein Bild, so spricht man von *Bildverarbeitung (BV)* oder *Image Processing*.
- Ist ein Bild die Eingabe und eine Beschreibung die Ausgabe, so handelt es sich um eine *Bildanalyse* oder *Image Analysis*.
- Ist eine Beschreibung die Eingabe und ein Bild die Ausgabe, so spricht man von einer *Bildsynthese*. Der Begriff der *Computergrafik* entspricht am ehesten diesem Bereich.

Output Input	Image	Description
Image	Image Processing	Image Analysis
Description	Image Synthesis (Computer Graphics)	All other IT

Abb. 1: Bildverarbeitung, Bildanalyse und Bildsynthese als Verarbeitungsschritte.

Die Abgrenzung zur Bildsynthese (Computergrafik) ist jedoch nicht klar abgegrenzt:

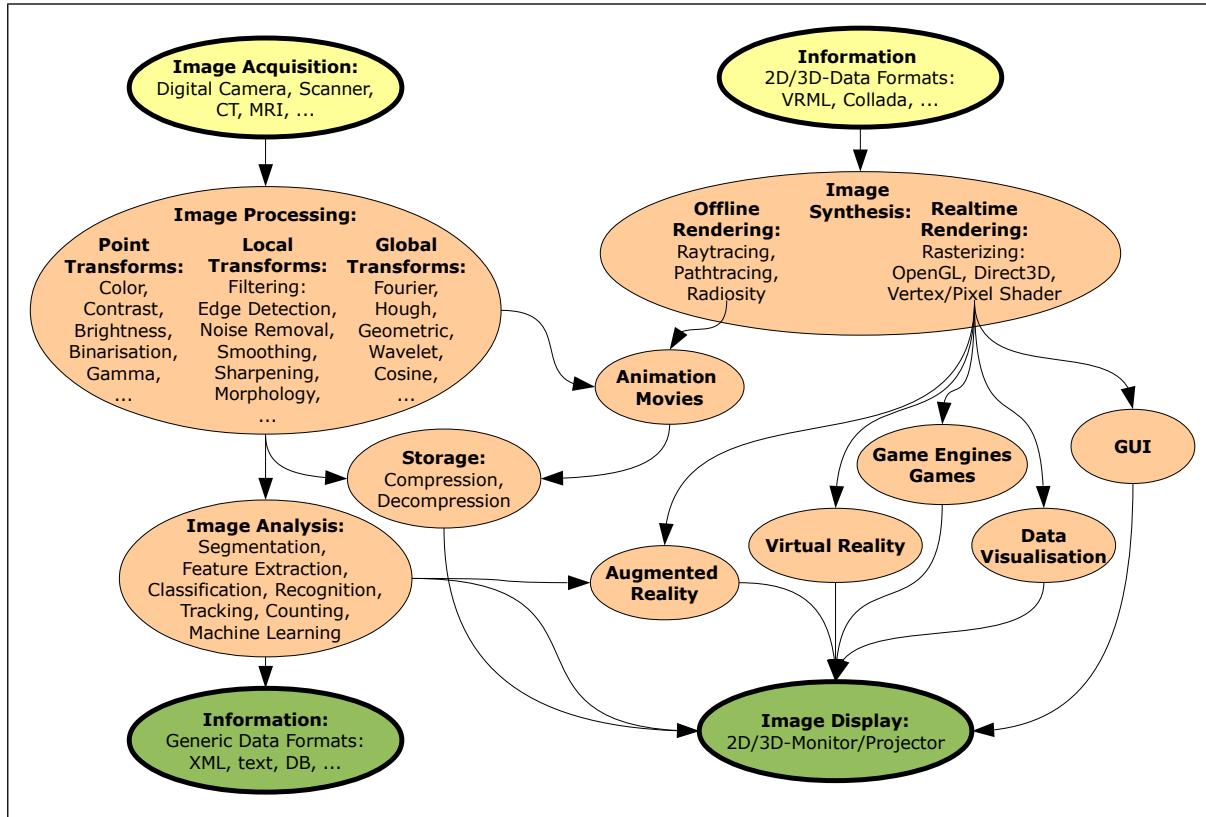


Abb. 2: Viele interessante Gebiete bedienen sich sowohl der Bildverarbeitung (Image Processing) als auch der Computergrafik (Image Synthesis).

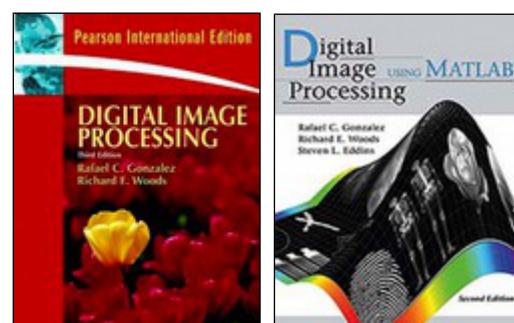
1.2 Literaturempfehlungen

Nachfolgend sind einige der Quellen aufgeführt, die der Autor für die weitere Lektüre empfiehlt, die aber für den Unterricht nicht erforderlich sind.

- **Digitale Bildverarbeitung, eine Einführung mit Java und ImageJ** von Wilhelm Burger und Mark James Burge, 2. Auflage, Springer Verlag. Das Buch ist eine Übersetzung des englischen Originals vom selben Verlag. Webseite: <http://www.imagingbook.com/>. Im Verlaufe des Skriptes wird mit dem Literaturverweis [Burger06] des öfteren auf dieses Buch verwiesen.



- **Digital Image Processing** von Rafael C. Gonzales und Richard E. Woods in Englisch ist eines der am meisten verwendeten Büchern an Universitäten und Fachhochschulen.
- **Digital Image Processing using Matlab** ist eine Version des oberen Buches mit vielen Matlab-Beispielen. Alle Matlab-Beispiele und Funktionen sind in der [DIPUM-Toolbox](#) erhältlich.



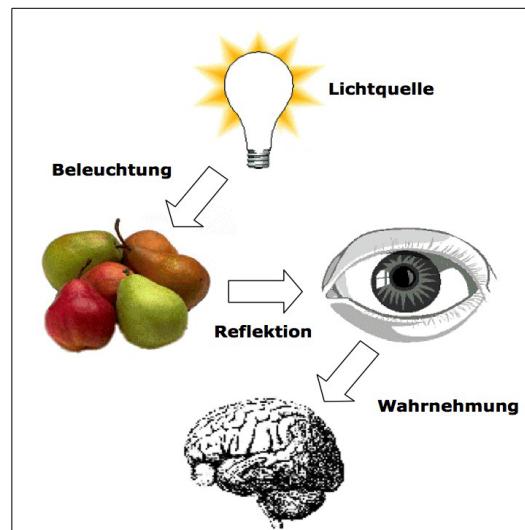
2 Grundlagen



rundlagen: Bevor wir uns mit der eigentlichen Computer Vision beschäftigen wollen, müssen wir uns zuerst mit einigen physikalischen und biologischen Grundlagen des Sehens vertraut machen. Der Weg des Lichts von der Lichtquelle über ein reflektierendes Objekt bis in unser Auge weist einige interessante aber auch anspruchsvolle physikalische Eigenschaften auf. Bevor wir in unserem Hirn einen visuellen Reiz empfinden, führt das Auge bereits eine Signalverarbeitung durch, von der wir einiges für unsere Bildverarbeitung lernen können. Insbesondere die Farbe und die Modelle davon haben sehr viel mit unserem Sehapparat zu tun. Am Schluss des Kapitels sollen einige Werkzeuge vorgestellt werden, die wir in diesem Modul für die Bildverarbeitung verwenden wollen.

2.1 Grundlagen der visuellen Wahrnehmung

Sehen ist die Wahrnehmung der Sinnesreize, die unser Hirn vom Auge empfängt. Die Lichtstrahlen, welche in unser Auge gelangen, haben ihren Ursprung bei aktiven und passiven Strahlern. Eine aktive Lichtquelle strahlt eine elektromagnetische Strahlung mit einer bestimmten Frequenzverteilung aus. Trifft diese Strahlung auf ein Objekt auf, so wird ein Teil davon reflektiert. Welche Frequenzanteile reflektiert werden, hängt von der Materialbeschaffenheit des Objekts ab. Schliesslich erreichen die reflektierten Strahlen die Netzhaut in unserem Auge, wo deren Energie in Nervensignale umgewandelt wird, die dann ans Gehirn weitergeleitet werden.



2.1.1 Das Licht

Die Natur des Lichtes ist sehr komplex und die Physik bedient sich verschiedener Modelle, um alle Eigenschaften plausibel zu erklären.

2.1.1.1 Strahlenmodell

In diesem Modell wird Licht durch unabhängige, unendlich dünne Strahlen und deren geometrische Interaktion mit optischen Materialien beschrieben. Trifft ein Lichtstrahl auf ein Medium, so kann er reflektiert und/oder durch das Medium hindurch geleitet (Transmission) und/oder von diesem absorbiert werden. Die ersten beiden Effekte, die Reflexion und Refraktion (Brechung), können durch das Strahlenmodell gut erklärt werden. Der dritte Effekt kann besser im Photonenmodell erklärt werden.

Reflexion

Licht wird an einer perfekt reflektierenden Oberfläche genauso umgelenkt, wie eine Billardkugel an einer Bande: Der Einfallswinkel ist gleich dem Ausfallwinkel und beide sind in derselben Ebene rechtwinklig zur reflektierenden Fläche. Dies stimmt wohl-gemerkt nur für perfekte Spiegel, die in der Realität selten vorkommen.

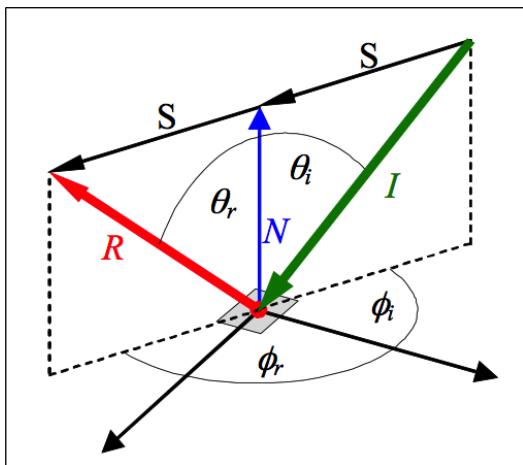


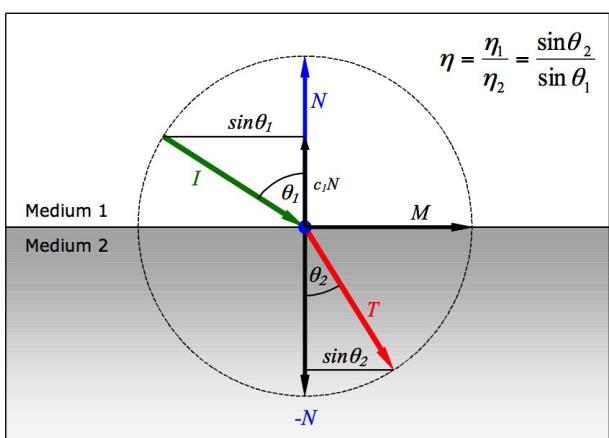
Abb. 3: Perfekte Strahlenreflexion

Refraktion

Ist ein Objekt transparent, so können Lichtstrahlen ein transparentes Medium passieren. Bedingt durch die unterschiedlichen optischen Dichten zweier Medien, kommt es am Übergang zu einer Geschwindigkeits- und Richtungsänderung die auch **Refraktion** genannt wird. Jedes Medium besitzt einen **Brechungsindex** η (eta), der das Verhältnis der Lichtgeschwindigkeit in diesem Medium zur Lichtgeschwindigkeit im Vakuum angibt.

Medium	Brechungsindex η
Vakuum:	1.0
Luft:	1.0003
Wasser:	1.333
Hornhaut:	1.37
Glas:	1.5-1.6
Diamant:	2.417

Am Übergangspunkt von einem Medium 1 mit einem niedrigeren Index η_1 zu einem Medium 2 mit einem höheren Index η_2 wird der Lichtstrahl zur Normalen hin gebrochen. Die unterschiedlichen Brechungswinkel θ_1 (theta_1) und θ_2 zur Normalen hin gehorchen dem Gesetz von [Snell](#). [Willebrord Snell](#) war ein holländischer Mathematiker und entdeckte dieses Gesetz 1621. Der Transmissionsvektor T berechnet sich mit folgendem Ausdruck:



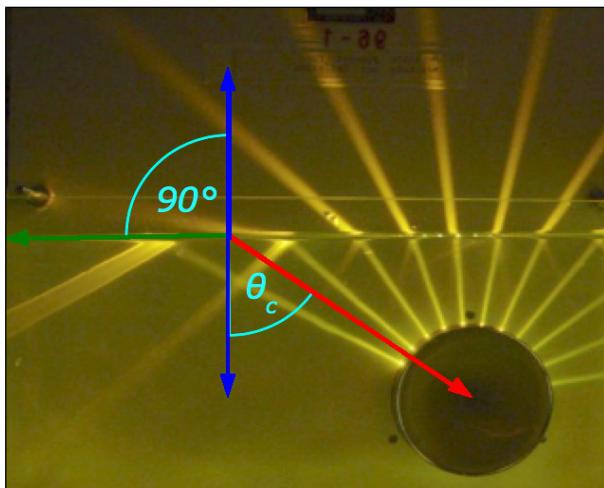
$$\begin{aligned} \eta &= \frac{\eta_1}{\eta_2} = \frac{\sin \theta_2}{\sin \theta_1} \\ T &= \eta I + (\eta c_1 - \sqrt{c_2}) N \\ \text{mit } c_1 &= \cos \theta_1 = -I \cdot N \\ c_2 &= 1 - \eta^2 (1 - c_1) \\ \eta &= \frac{\eta_1}{\eta_2} = \frac{\sin \theta_2}{\sin \theta_1} \quad (\text{Gesetz von Snell}) \end{aligned}$$

Abb. 4: Situation bei der gebrochenen Strahlberechnung

Totale interne Reflexion (TIR)

Wenn der Term c_2 in der Berechnung vom Transmissionsvektor T negativ ist, dann gäbe die Wurzel daraus eine imaginäre Zahl. Dies ist nicht ein Problem dieser Berechnung, sondern entspricht einem Effekt, der als totale interne Reflexion (TIR) bezeichnet wird. Wenn ein Lichtstrahl unter einem flachen Winkel von einem dichten Medium an ein

weniger dichtetes Medium gelangt, so wird es nicht unter einem gebrochenen Winkel das Medium verlassen, sondern wird unter dem gleichen Winkel ins dichte Medium zurückgespiegelt. Auf diesem Phänomen basiert die Fortbewegung der Lichtstrahlen in einem Glasfaserkabel.



$$\frac{\eta_1}{\eta_2} = \frac{\sin \theta_c}{\sin \frac{\pi}{2}}$$

$$\eta_1 \sin \theta_c = \eta_2 \sin \frac{\pi}{2} = \eta_2$$

$$\sin \theta_c = \frac{\eta_2}{\eta_1}$$

$$\arcsin\left(\frac{1.003}{1.33}\right) = 48.95^\circ$$

Abb. 5: Der maximale Brechungswinkel θ_2 entsteht, wenn der Luftstrahl parallel zur Wasseroberfläche ist. Ist der Winkel θ_2 grösser, so wird der Lichtstrahl innerhalb des dichteren Mediums total reflektiert.

2.1.1.2 Photonenmodell

Das Photonenmodell geht hauptsächlich auf Albert Einstein zurück, der für seine Formulierung den Nobelpreis erhalten hat. Licht wird in diesem Modell als reines Energiephänomen betrachtet, bei dem einzelne Energiepakete mit Lichtgeschwindigkeit vom Sender zum Empfänger transportiert werden. So ein Energiepaket wird Photon genannt (griech. Lichtquantum). Mit diesem Modell können wir am besten die Interaktion zwischen Licht und Materie erklären. Photonen können nur ausgestrahlt (emittiert) oder absorbiert werden. Da Photonen keine elektrische Ladung und kein magnetisches Moment besitzen, können sie von elektromagnetischen Feldern nicht abgelenkt werden.

Lumineszenz

Licht hat seinen Ursprung im atomaren Bereich. Nach dem *Bohrschen Modell* (Niels Bohr 1914) kreisen negativ geladene Elektronen um den positiv geladenen Atomkern, so wie Planeten um einen Stern kreisen. Je weiter ein Elektron vom Atomkern entfernt ist, umso höher ist seine Energie. Der Gesamtprozess, bei dem ein Atom nun Energie aufnimmt (absorbiert) und durch Photonenemission wieder abgibt, heisst *Lumineszenz*:

- **Absorption:** Ein Atom absorbiert ein Photon, wenn das Photon auf ein Elektron trifft und dieses dadurch auf eine, vom Atomkern weiter entfernt liegende, Umlaufbahn versetzt wird. Dieser instabile Zustand bleibt dann mehr oder weniger lang bestehen, bis das Elektron wieder auf seine ursprüngliche Lage zurückfällt.
- **Emission:** Hat ein Elektron eine energetisch erhöhte Umlaufbahn, so möchte es wieder auf seine angestammte und stabile Umlaufbahn zurückfallen. Die dabei frei werdende Energie wird in Form eines Photons mit Lichtgeschwindigkeit weggeschleudert.

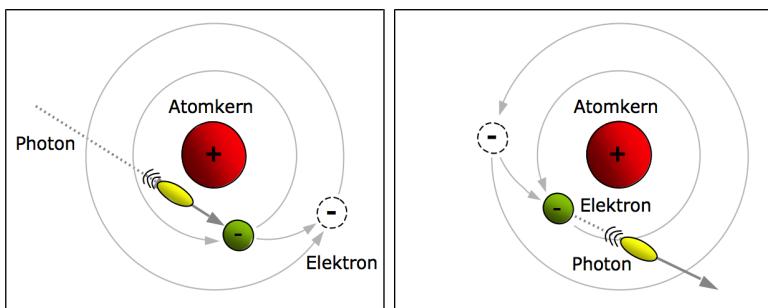


Abb. 6: Absorption und Emission von Photonen

Lichtquellen

Man unterscheidet die Lichtquellen in zwei Gruppen:

- **Wärmestrahler:** Diese Lichtquellen nehmen die Energie nicht durch Photonen auf, sondern durch Wärmenergie. Diese entsteht z. B. bei Verbrennungsprozessen oder durch Zuführen von elektrischer Energie. Die abgegebene Strahlung ist zu einem grossen Teil unsichtbare Wärmestrahlung im Infrarotbereich und nur zu einem kleinen Teil sichtbares Licht.
 1. *Natürliche Wärmestrahler*: Sonne, Blitze, alle Feuer durch gasförmige und fossile Brennstoffe.
 2. *Künstliche Wärmestrahler*: Glühlampen, Halogenlampen (95% Wärmestrahlung)
- **Lumineszenzstrahler:** Diese Lichtquellen leuchten, ohne dass sie brennen oder glühen, weshalb man sie auch *kalte Lichtquellen* oder nichtthermische Strahler nennt. Je nach Anregungsenergie unterscheidet man verschiedene Lumineszenzarten:
 3. *Photolumineszenz*: Anregung durch Photonen. Dabei unterscheidet man je nach Zeitdauer zwischen Anregung und Emission des Lichtes zwischen:
 - *Fluoreszenz*: Unmittelbare Emission nach der Anregung.
 - *Phosphoreszenz*: Verzögerte Emission nach der Anregung.
 4. *Elektrolumineszenz*: Anregung durch elektrischen Strom:
 - *Entladungslampen*: Sie beruhen auf einer Gasentladung (z. B. Neon). Sie erzeugen Ultraviolettsicht, das durch eine fluoreszierende Leuchtschicht auf dem Glasbehälter in sichtbares Licht gewandelt wird.
 - *Licht emittierende Dioden (LED)*: Diese Diode produzieren Licht in einem engen Spektralbereich und praktisch keine Wärme bei sehr hoher Lebensdauer.
 - *Licht emittierende Kondensatoren (LEC)*: Durch elektrische Felder können bestimmte Materialien zum Leuchten gebracht werden. (z. B. Leuchtfolien in Flachbildschirmen)
 - *Laser (Light Amplification by Stimulated Emission of Radiation)*: Gebündelte Lichtausstrahlung.
 5. *Kathodolumineszenz*: Anregung durch Elektronenbeschuss (z. B. Kathodenstrahldiagon).
 6. *Chemolumineszenz*: Anregung durch chemische Reaktion (z. B. Leuchtstäbe).
 7. *Radiolumineszenz*: Anregung durch radioaktive Strahlung. Früher wurden die Zifferblätter mit einer Radiumfarbe bemalt, damit man auch nachts die Ziffern noch lesen konnte. Heute wird ein ungefährliches Prometium (Betastrahler) verwendet.
 8. *Biolumineszenz*: Anregung durch chemische Reaktion in lebenden Organismen (z. B.: Oxidation von *Luciferin* im Leuchtkäfer).

Lichtenergie

Die Energie E eines Photons und seine Sendefrequenz sind proportional zueinander. Je höher die Frequenz v , desto höher die Energie. Die Proportionalitätskonstante heisst *Planck'sches Wirkungsquantum* h :

$$E = h \cdot v \quad \text{in eV}$$

1 eV (Elektronen Volt) entspricht $1.6 \cdot 10^{-19}$ J, der Energie, die ein Elektron aufnimmt, wenn es um die Potenzialdifferenz von 1 Volt beschleunigt wird. Die Wellenlänge λ der Lichtwelle ist umgekehrt proportional zur Energie:

$$E = h \cdot \frac{c}{\lambda} \quad \text{worin } c \text{ die Lichtgeschwindigkeit im Vakuum ist } 299'792'458 \text{ m/s}$$

2.1.1.3 Wellenmodell

Licht wird in diesem Modell als elektromagnetische Welle betrachtet, die sich im freien Raum mit Lichtgeschwindigkeit ausbreitet. Die elektrische und die magnetische Feldkomponente stehen dabei senkrecht zueinander. Mit diesem Modell können wir Phänomene wie das Farbspektrum und Polarisation erklären.

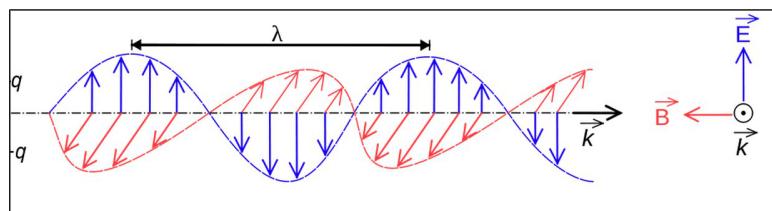


Abb. 7: Elektrisches Feld (E, blau) und magnetisches Feld (B, rot) einer sich im Vakuum nach rechts ausbreitenden Welle mit der Wellenlänge λ .

Farbspektrum

Farben sind kulturelle Bezeichnungen für elektromagnetische Wellen mit Wellenlängen zwischen 400 und 700 nm. Im Photonenmodell würde man von der Energieverteilung der eintreffenden Photonen sprechen. Dieses Spektrum entspricht einem Frequenzband von $750 \cdot 10^{12}$ Hz bis $450 \cdot 10^{12}$ Hz. Dies ist sehr hoch im Vergleich zu den Radio-, TV- oder Handyfrequenzen. Das Prinzip der Wellenausbreitung ist jedoch genau dasselbe. Insgesamt deckt der sichtbare Bereich weniger als 1% der möglichen elektromagnetischen Strahlung ab.

Lange Lichtwellen mit 650-700 nm werden als "Rot", Mittlere bei 540-550 nm als "Grün", und Kurze bei 430-440 nm als "Blau" bezeichnet. Auffallend am sichtbaren Spektrum ist, dass die Veränderungen nicht gleichmäßig sind. Die Wellenlängen von Rot und Gelb unterscheiden sich um 120 nm, während diejenigen von Gelb und Grün nur um 40 nm (= 0.00004 mm) differieren. Das Auge muss also ungeheuer sensibel sein, damit es dazwischen noch ein duzend Zwischentöne wahrnehmen kann.

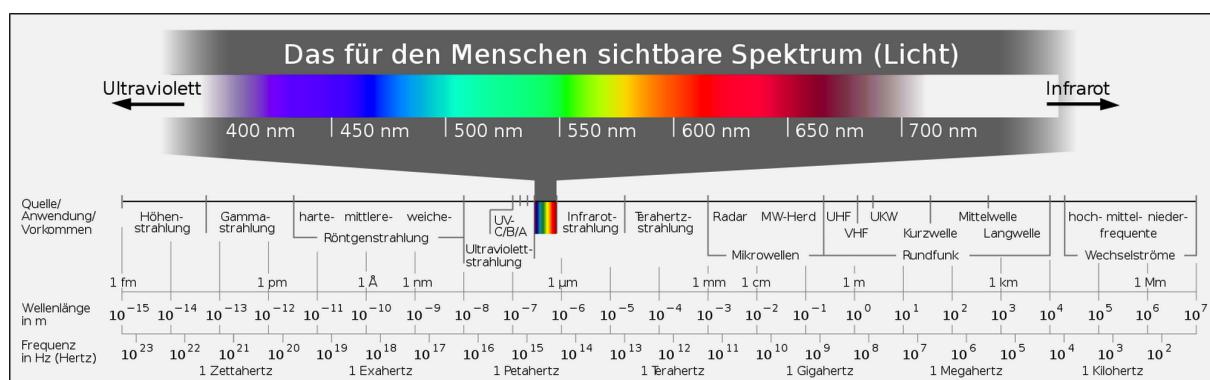
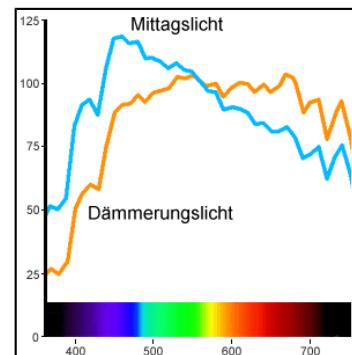


Abb. 8: Frequenzspektrum des Lichtes innerhalb des elektromagnetischen Spektrums

Der kurzwelligere Bereich jenseits des Blaulichts wird Ultraviolettbereich genannt und ist nicht mehr sichtbar. Wegen der hohen Energie verursachen diese Strahlen jedoch zahlreiche anderweitig wahrnehmbare Wirkungen, wie z. B. Hautrötung, Verbrennung oder Ozongeruch. Im langwelligen Bereich unterhalb des Rotlichts befindet sich die Infrarotstrahlung, die ebenfalls unsichtbar, durch Erwärmung aber ebenfalls wahrnehmbar ist.

Einige elektronische Apparate, wie z. B. Fernbedienungen arbeiten mit Infrarotlicht. Achtung für Experimentierfreudige: Die Infrarot-LEDs in diesen Apparaten sind äusserst effiziente Lichtquellen, die die Augen durch Fokuserwärmung schädigen können (fehlender Lidschlussreflex!)

So genannt weisses Licht ist eine Mischung aus allen sichtbaren Spektralbereichen. Dass Weiss aber nicht gleich Weiss ist, wissen wir von unseren Lampen. Bei Glühlampen z. B. muss eine sehr hohe Drahttemperatur erreicht werden, um alle Spektralbereiche (auch den Blaugrünen) genügend abzudecken. FL-Lampen enthalten Mischungen von Leuchtpigmenten, die Weiss annähernd reproduzieren. Auch das Tageslicht ist nicht einfach Weiss, sondern ändert sich konstant durch den Tag, je nach Sonnenstand und Bewölkung.



Brechung im Wellenmodell

Wie bereits im Strahlenmodell erwähnt ist der Brechungsindex das Verhältnis der Lichtgeschwindigkeit im Vakuum zur Geschwindigkeit des Lichts in einem Medium. Die Lichtgeschwindigkeit in einem Diamanten ist demnach $c/2.417$. Newton hat gezeigt, dass der Brechungsindex auch von der Wellenlänge abhängig ist, in dem er weisses Licht mit einem Prisma in seine Spektralfarben zerlegt hat:

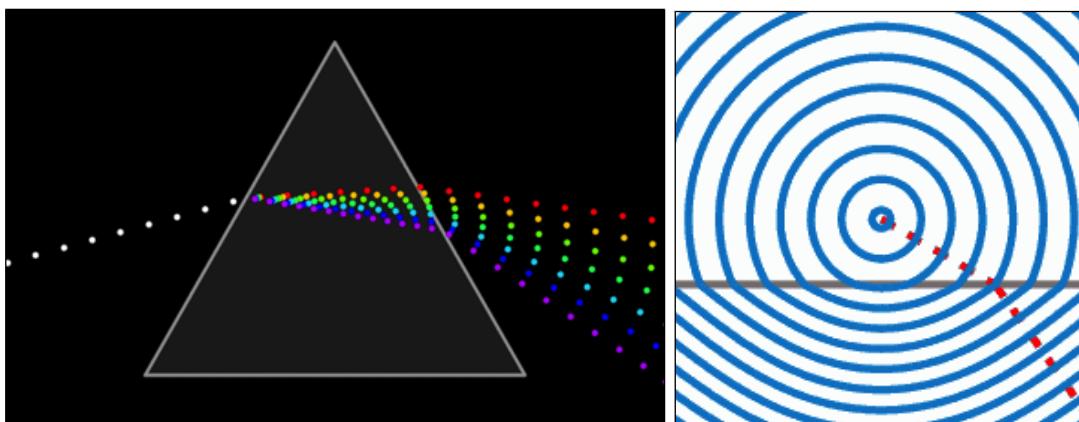


Abb. 9: Links: Weisses Licht wird in seine Spektralfarben aufgefächert, weil je nach Wellenlänge die Strahlen mehr oder weniger stark gebrochen werden. Diese unterschiedliche Brechung wird durch die unterschiedliche Phasengeschwindigkeit der verschiedenen Wellenlängen verursacht. Die Abbildung rechts zeigt auf, dass es zu einer Richtungsänderung kommen muss, wenn die Ausbreitungsrichtung rechtwinklig zur Welle steht.

Phasengeschwindigkeit und Gruppengeschwindigkeit

Wenn Licht aus verschiedenen Frequenzen besteht, unterscheidet man zwei verschiedene Ausbreitungsgeschwindigkeiten. Die Phasengeschwindigkeit ist die Geschwindigkeit, mit der sich eine monochromatische Welle gemäss ihrer Frequenz und Periodenlänge ausbreitet. Sie ist gleich der Geschwindigkeit, mit der sich der Wellenberg oder das Wellental fortbewegt. Sie berechnet sich aus der Wellenlänge λ über der Periodenlänge T oder aus Wellenlänge mal Frequenz f : $v_p = \lambda/T = \lambda \cdot f$

Die Gruppengeschwindigkeit ist die Geschwindigkeit, mit der sich ein Wellenpaket fortbewegt. Für elektromagnetische Wellen ist die Phasengeschwindigkeit und die Gruppengeschwindigkeit im Vakuum gleich der Lichtgeschwindigkeit c . In einem Medium ist die Phasengeschwindigkeit abhängig von der Wellenlänge.

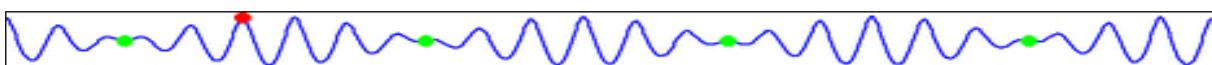
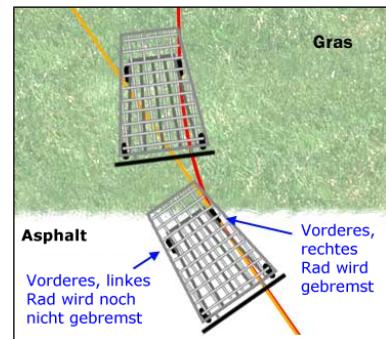


Abb. 10: Der rote Punkt bewegt sich mit der Phasengeschwindigkeit und die grünen Punkte bewegen sich mit der Gruppengeschwindigkeit. Dieses Bild muss unbedingt als [Animation in Wikipedia](#) betrachtet werden.

Eine schöne, nichtmathematische Erklärung für die Lichtbrechung geht wie folgt. Fährt man schräg mit einem Einkaufswagen von einem feinen Asphaltbelag auf eine Grasfläche, so werden die vier Räder nicht gleichzeitig abgebremst. Weil die vier Räder unterschiedliche Widerstände erfahren, wird der Wagen gedreht. Je schräger ich auf die Rasenfläche zufahre, umso länger dauert die asymmetrische Bremsung und umso mehr wird der Wagen gedreht. Die beiden Flächen stellen ebenfalls zwei unterschiedliche Medien dar. Fahre ich genau rechtwinklig auf den Rasen, so erfolgt, wie bei der Lichtbrechung, keine Richtungsänderung. Die totale interne Reflexion lässt sich mit dieser Erklärung auch verstehen.



Warum entsteht ein Regenbogen?

Vielleicht haben sie sich auch schon gefragt, warum ein Regenbogen entsteht, warum er rund, warum er an einer bestimmten Stelle erscheint oder vielleicht warum es einen doppelten Regenbogen gibt. Die Antworten dazu haben wir dem [René Descartes](#) zu verdanken.

Wenn man Sonnenstrahlen verfolgt, die auf die obere Hälfte der Regentropfen auftreffen und diese dann weiterverfolgt mit einer Brechung in den Tropfen hinein, mit einer Reflexion an der Rückseite zurück und wieder mit einer Brechung aus dem Tropfen heraus, so stellte man fest, dass es einen maximalen Austrittswinkel von 42° an der Unterseite gibt. Um diesen Winkel verdichten sich die Lichtstrahlen, was zu einem helleren Lichtstreifen überall dort führt, wo das zurückgeworfene Licht einen Winkel von 42° zum einfallenden Sonnenlicht einnimmt. Dies ist der Grund, warum jeder seinen eigenen Regenbogen sieht. Der zweite, schwächere Regenbogen kann mit demselben Sachverhalt begründet werden. Er resultiert durch die Sonnenstrahlen, die in die untere Regentropfenhälfte einfallen und in einem Winkel von 54° reflektiert werden. Er ist schwächer, weil die Strahlen einmal mehr gespiegelt werden im Regentropfen. Warum wir an der Stelle des Bogens alle Farben, eben die Regenbogenfarben sehen, hat Newton 35 Jahre später herausgefunden. Je nach Wellenlänge verlassen die Strahlen den Tropfen zwischen 40 und 42° beim inneren Hauptbogen und mit $52-54^\circ$ beim äusseren Bogen.

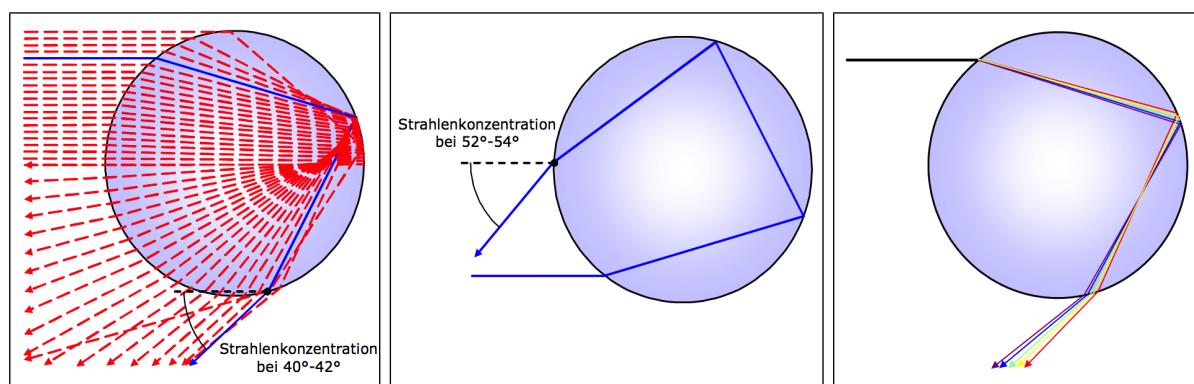
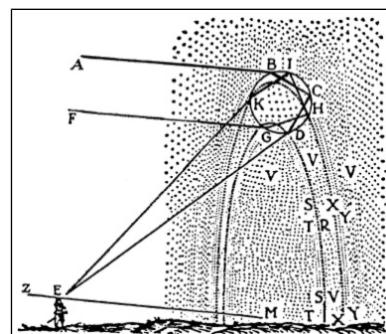


Abb. 11: Maximaler Ausfallwinkel der oberen (links) und der unteren Sonnenstrahlen (Mitte), sowie unterschiedliche Brechungswinkel von weißem Licht.

Gerade Strahlen

Lichtstrahlen breiten sich, abseits von schwarzen Löchern und anderen schweren Massen immer geradlinig aus. Das tönt sehr lapidar aber darauf beruht die geometrische Ähnlichkeit von Objekt und Bild in Abbildungssystemen. Dies ist die fundamentale Basis der Geometrie, die wir uns bei der Darstellung von Objekten in der Computergrafik zunutze machen werden.

Kleine Wellenlänge

Die Wellenlänge beim sichtbaren Licht liegt im Bereich von Millionsteln von Metern. Diese wiederum lapidare Tatsache ist der Grund, warum wir sehr kleine Gegenstände überhaupt sehen und warum optische Instrumente sehr präzise geschliffen und aufgebaut sein müssen. Aus demselben Grund können wir kleinere Strukturen, wie Elektronen und Atome auch mit dem besten Lichtmikroskop nicht sehen.

Polarisation

Elektromagnetische Wellen sind Transversalwellen, d. h., sie schwingen senkrecht zur Ausbreitungsrichtung. Diese Ebene wird auch Polarisationsebene genannt. Für einen Lichtstrahl ist sie konstant, kann aber beliebig um die Ausbreitungsrichtung herum angeordnet sein. Lichtstrahlen lassen sich also aufgrund ihrer Polarisation unterscheiden bzw. filtern. Wird ein Lichtstrahl an einem transparenten Material gebrochen, so wird nur eine bestimmte Polarisationsebene durchgelassen. Der Rest wird reflektiert oder absorbiert. Bereits polarisiertes Licht kann durch einen sogenannten Polarisationsfilter komplett blockiert werden. In LCD-Displays wird dieser Effekt eingesetzt, um ein Pixel ein- oder auszuschalten.

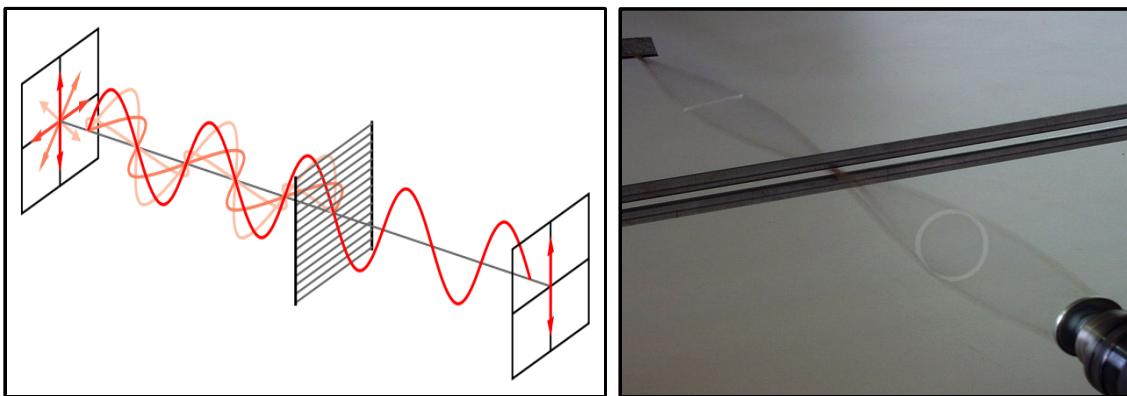


Abb. 12: Der Polarisationsfilter lässt nur eine Richtung des unpolarisierten Lichts passieren. Rechts: Ein rotierend geschwungenes Gummiband schwingt nach dem Schlitz linear.

Polarisationsfilter werden in jeder besseren Sonnenbrille eingesetzt, um den blauen Himmel abzudunkeln. In der Fotografie kann ein Polfilter eingesetzt werden, um störende Spiegelungen bei Gläsern wegzufiltern.

Das Polarisationsverfahren eignet sich aber auch gut für Projektionen. Man benötigt zwei Projektoren, die zusätzlich mit je einem Polarisationsfilter ausgestattet sind, die zueinander senkrecht stehen. Die beiden Bilder des Stereobildpaars werden übereinander projiziert, lassen sich aber dank der verschiedenen Polarisationsebenen wieder über solche Filter trennen. Jedes Auge des Betrachters bekommt so nur das für es bestimmte Bild zu Gesicht, und der Betrachter erhält dadurch einen Tiefeneindruck.

2.1.2 Das Auge

Das menschliche Auge ist ähnlich wie eine Fotokamera aufgebaut. Es besteht im Wesentlichen aus dem optischen Apparat (=Objektiv) und der Netzhaut. Das Auge ist kugelförmig und hat im Schnitt einen Durchmesser von 22mm.

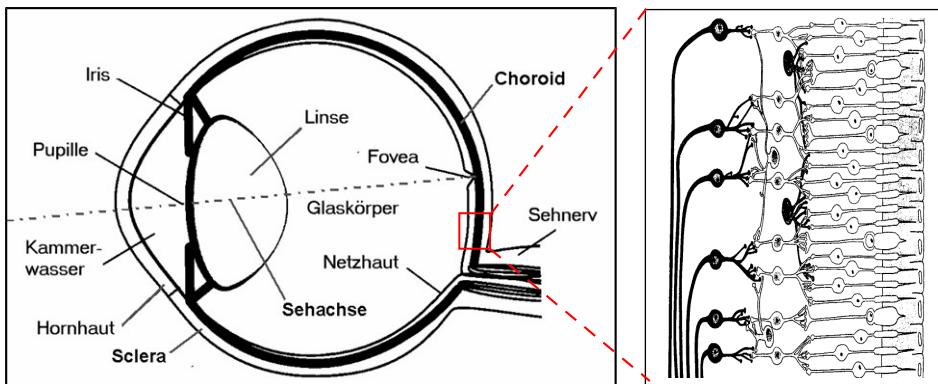
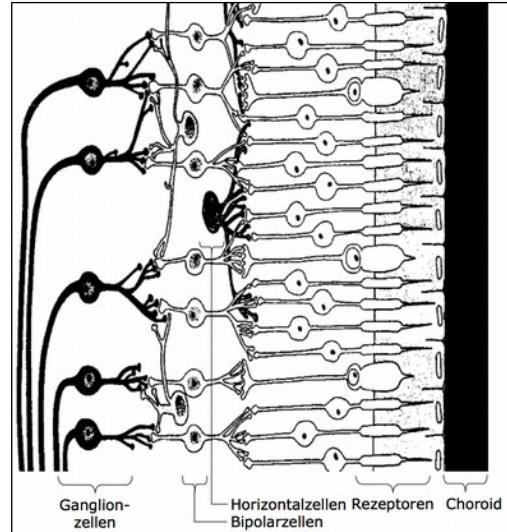


Abb. 13: Das Auge im Längsschnitt mit einem vergrößerten Ausschnitt aus der Retina

Das Licht passiert zuerst die **Hornhaut (Cornea)**, die das erste und stärkste lichtbrechende Element des Auges ist. Sie ist nicht durchblutet und daher transparent. Sie enthält aber viele kleine Nerven, weshalb sie sehr schmerzempfindlich ist. Danach tritt das Licht in das **Kammerwasser**, das dem Ausgleich des Augeninnendrucks dient. Dahinter folgt die **Pupille (Iris)**, die das eintretende Licht reguliert, wie die Blende bei der Kamera. Der Durchmesser kann zwischen 2 bis 8 mm variieren. Die nachfolgende **Linse** fokussiert die Lichtstrahlen auf die Netzhaut und hat trotz ihrer starken Krümmung wenig Brechkraft. Die Linse besteht aus ca. 65% Wasser. Die restlichen Fette und Proteine haben u.a. die Aufgabe Infrarot- und UV-Licht aufzuhalten. Das Augeninnere ist ausgefüllt mit dem transparenten **Glaskörper**. Schliesslich trifft das Licht auf die **Netzhaut (Retina)**, wo es absorbiert und in Nervenimpulse umgesetzt wird. Bevor die Signale das Auge im **Sehnerv** verlassen, werden sie einer starken Komprimierung und Umkodierung unterzogen.

2.1.2.1 Retina

Die Retina ist ein mehrschichtiges System von Nervenzellen. Zuvor liegen die **Ganglionzellen**, deren Ausläufer zum blinden Fleck, der Mündung des Sehnervs, streben. In der mittleren Schicht sind die **Bipolar-** und die **Horizontalzellen**. Sie bilden Differenzsignale für eine Kontrast- und Kantenverstärkung und nehmen gleichzeitig eine Farbumwandlung vor. Beides dient der Daten- und Rauschreduktion. Erst jetzt, nachdem das Licht einiges an Zellen und Blutgefäßen passiert hat, kommen die lichtempfindlichen **Rezeptoren**. Hinter den Rezeptoren befindet sich eine glatte schwarze Schicht, **Choroid** genannt, welche die noch nicht umgewandelten Photonen absorbiert und damit verhindert, dass sie zurückreflektiert werden.



Rezeptoren

Die Rezeptoren enthalten den Sehfarbstoff **Rhodopsin** als aktive Substanz. Der Farbstoff wird laufend neu gebildet, wobei das Vitamin A eine wichtige Rolle spielt. Lichteinfall spaltet den Farbstoff in seine Bestandteile, wobei Nervenimpulse an die nächsthöheren Zellen abgegeben werden. Je intensiver der Lichtreiz, desto höher die Frequenz der Nervenimpulse. Es gibt 2 Typen von Sehzellen.

Die **Stäbchen** sind hochempfindlich auf Photonen in einem breiten Wellenlängenbereich und deshalb für das Schwarzweiss-Sehen zuständig. Sie sind durch den Tag abgeschaltet und kommen erst mit der Dämmerung und bei Nacht zum Einsatz.

Die *Zapfen* sind empfindlich auf Photonen in schmalen Wellenlängenbereichen und für das Farbsehen zuständig. Sie sind bei grosser Helligkeit durch den Tag aktiv und werden bei der Dämmerung deaktiviert.

Verteilung der Rezeptoren

Der Mensch hat etwa 110-130 Mio. Stäbchen und ca. 6-8 Mio. Zapfen. Die Mitte des Gesichtsfeldes liegt in der *Fovea Centralis*, auch *gelber Fleck* genannt. Auf dieser Fläche von ca. 3 mm Durchmesser sind praktisch alle Zapfen konzentriert. In der Fovea erreichen die Zapfen eine Dichte von etwa 150'000 pro mm². Ein Bussard besitzt dort eine Dichte von 1 Mio. Zapfen pro mm²). Ein 5-Frankenstück auf Armlänge vor den Augen gehalten, gibt etwa den Bereich an, wo der grösste Teil unserer Wahrnehmung geschieht.

Die Stäbchen liegen breiter um die optische Achse verteilt und reichen bis an den Rand des Gesichtsfeldes. Bewegungen können dort rasch, wenn auch nur schwarz-weiss, wahrgenommen werden und lösen eine Änderung der Blickrichtung dorthin aus. Der Rest des Gesichtsfeldes ist sozusagen ein Kurzzeitspeicher, der mittels Augenbewegungen mehr oder weniger häufig abgetastet wird. Darum ist es z. B. anstrengend, sich einen fremdsprachigen Film mit Untertiteln anzusehen.

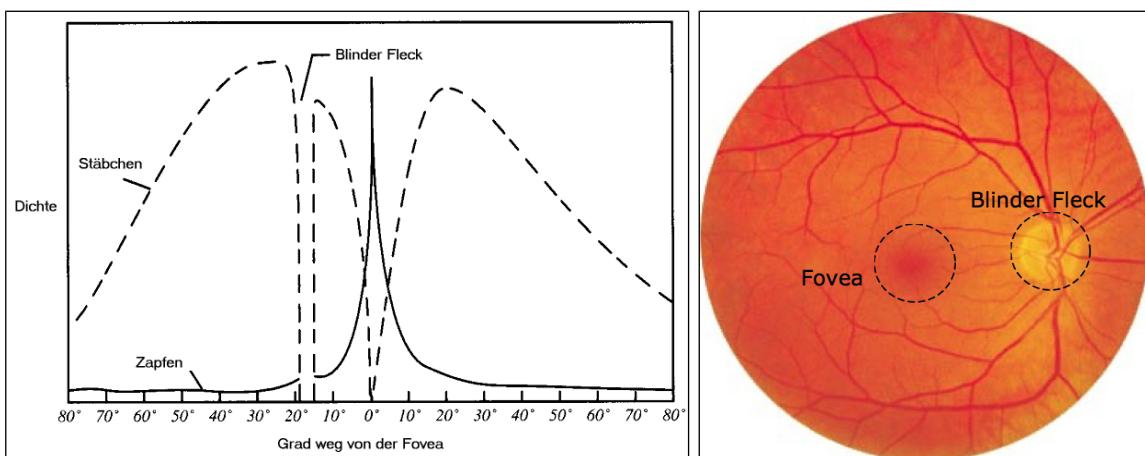


Abb. 14: Links: Zapfen- und Stäbchenverteilung der Netzhaut Abb. 28: Rechts: Fotografie der Netzhaut

Neben der Fovea gibt es den *blinden Fleck*, wo etwa 1 Mio. Sehnerven das Auge verlassen und viele Blutgefäße hereinkommen. Die roten Bahnen in der obigen Fotografie sind Blutgefäße und nicht etwa Nervenbahnen. Das Hirn ergänzt den blinden Fleck, wo es keine Rezeptoren hat, mit Information, die er aus der Umgebung interpoliert.



Abb. 15: Unser Auflösungsvermögen ist stark von der Blickrichtung abhängig. Originalbild von Louis Boilly (1761-1845) „Thirty-Six Faces of Expression“ links und eine logarithmische Polartransformation davon rechts. Bildquelle [[Peters07](#)].

Wenn ca. 130 Mio. Rezeptoren Signale produzieren aber nur etwa 1 Mio. Sehnerven das Auge verlassen, dann muss schon innerhalb des Auges eine starke Signalverarbeitung stattfinden. Wie dies geschieht, werden sie in den beiden nachfolgenden Kapiteln über das Farb- und Kontrastsehen erfahren.

2.1.2.2 Farbsehen

Seit Jahrhunderten beschäftigen sich Physiker, Physiologen und Psychologen mit dem Phänomen Farbe. Bis heute jedoch ist dieser Wahrnehmungsaspekt nicht vollständig verstanden. Meilensteine für unser Farbverständnis wurden erarbeitet durch:

- **Newton 1672:** Zerlegung von weissem Licht in seine Spektralfarben
- **Goethe:** Über die naturgemäße Ordnung der Farben
- **Maxwell, Young, Helmholtz 1800-1870:** Tristimulustheorie
- **Hering 1895:** Gegenfarbentheorie

Farbton, Leuchtdichte und Sättigung

In der Wahrnehmungslehre werden Farben durch die Begriffe *Farbton*, *Leuchtdichte* und *Sättigung* beschrieben.

- **Farbton (Hue):** Er entspricht der dominanten Wellenlänge des Frequenzspektrums, die sich allerdings nicht immer klar bestimmen lässt (z. B. hat ein Rosa keine dominante Wellenlänge).
- **Helligkeit (Lightness):** Sie entspricht der physikalischen Eigenschaft der Strahlungs-dichte und misst die im Licht enthaltene Energie. Sie ist proportional zur Fläche unter der Spektrumskurve.
- **Sättigung (Saturation):** Sie entspricht der Erregungsreinheit, die sich aus dem Ver-hältnis der reinen Farbe exklusive dem Weißanteil zur reinen Farbe berechnet.

2.1.2.3 Tristimulustheorie

Es gibt drei Zapfentypen mit unterschiedlichen spektralen Empfindlichkeitskurven. Sie werden als L-Typ (*long wave*), M-Typ (*medium wave*) und S-Typ (*short wave*) bezeichnet. Die Empfindlichkeitskurven haben endliche Breiten und überlappen sich. Die Empfindlichkeiten des S- und des M-Typs liegen ungefähr im Bereich des blauen und des grünen Lichts. Der L-Typ hat seine höchste Empfindlichkeit aber nicht im roten, sondern im gelben Bereich. Der Farbeindruck resultiert aus der relativen Aktivierung der 3 Zapfentypen. Die relative Gesamtempfindlichkeit ergibt sich durch die Addition der 3 Kurven mit einem Maximum um 550 nm.

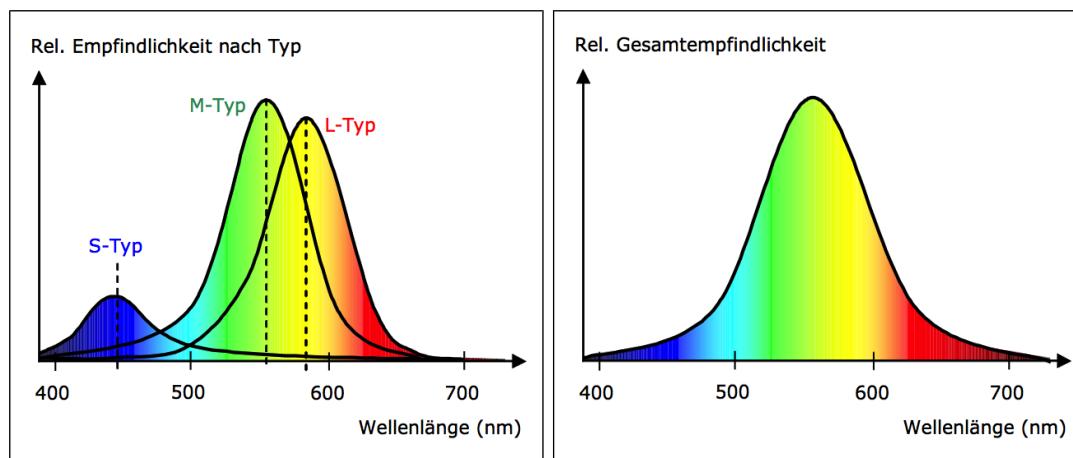


Abb. 16: Zapfenempfindlichkeit und Gesamtempfindlichkeit

Das Wissen um diese drei Zapfentypen wird *Tristimulus-Theorie* genannt und wurde von Thomas Young um 1801 und von Hermann von Helmholtz 1866 unabhängig voneinander entwickelt. Sie haben diese Theorie nur aufgrund des Sehverhaltens entwickelt. Dass es wirklich drei Typen mit unterschiedlichen, durch die molekulare Struktur bedingte, Farbempfindlichkeiten gibt, wurde erst vor Kurzem nachgewiesen.

Häufigkeit und Verteilung

Wir sehen im Blaubereich relativ unempfindlich, nicht weil der S-Typ schwach reagiert, sondern weil er weniger häufig vorkommt als die anderen beiden Zapfentypen. Es gibt rund 10 % S-, 48 % M- und 42 % L-Typen. Die unregelmäßige Verteilung der Zapfen dient der Vermeidung von sogenanntem Aliasing. Was das genau ist, werden sie in nachfolgenden Kapiteln genauer erklärt bekommen.

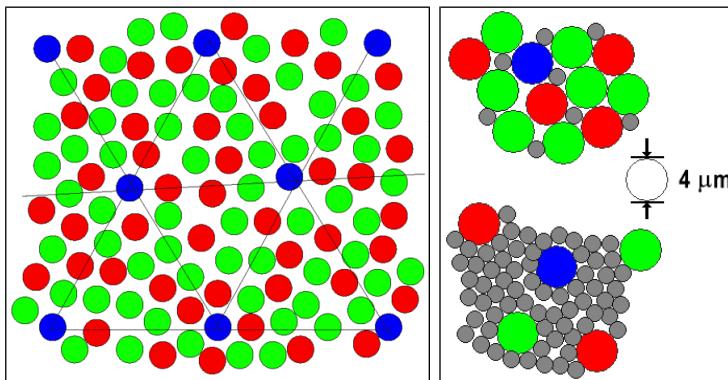


Abb. 17: Zapfenverteilung in der Fovea (links u. rechts oben) sowie 5 mm von der Fovea weg (rechts unten)

Metamerie

Der gleiche Farbeindruck kann aus verschiedenen Spektren (*Metameren*) erzeugt werden. An der Brown Universität gibt es dazu ein gutes [Java-Applet](#).

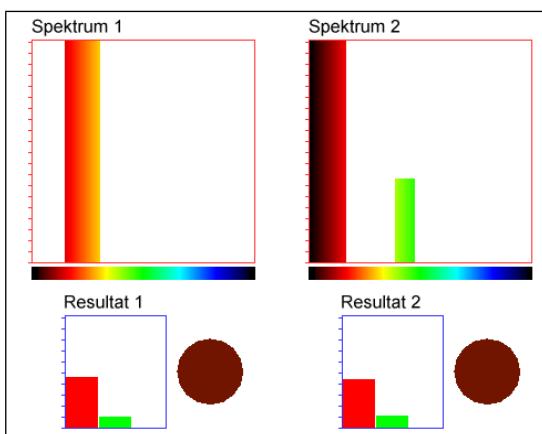


Abb. 18: Unterschiedliche Spektren mit demselben Eindruck

2.1.2.4 Gegenfarben-Theorie

Wenn man gefragt wird, welches die wichtigsten Grundfarben sind, so wird nach Rot, Grün und Blau meistens Gelb angegeben. Gelb erscheint uns als eigenständige Farbe. Bei den Farbmischungen können wir uns gelbrot, gelbgrün, blaugrün und rotblau sehr leicht vorstellen. Gelbblau und Rot-Grün können wir uns intuitiv aber nicht vorstellen, obwohl wir in der Schule lernen, dass Gelb und Blau im Farbtopf zusammen Grün ergeben.

Hering entwickelte daraus die [Gegenfarben-Theorie](#) um 1895, in der er vermutete, dass im Auge ein Gelbblau- und Rotgrün-Signal sowie ein Hell-dunkel-Signal entstehen. Mit seinem Gegenfarbenkreis können alle gesättigten Farbmischungen gebildet werden.

Wie bei der Tristimulus-Theorie fand man diese Vermutung 1966 exakt bestätigt, als man die Signale messen konnte, die das Auge verlassen. Die Umkodierung der S-, M- und L-Signale in ein R/G-, Y/B- und Hell/Dunkel-Signal findet in der 2. und 3. Verarbeitungsschicht der Netzhaut mit den Bipolar-, Ganglionzellen statt.

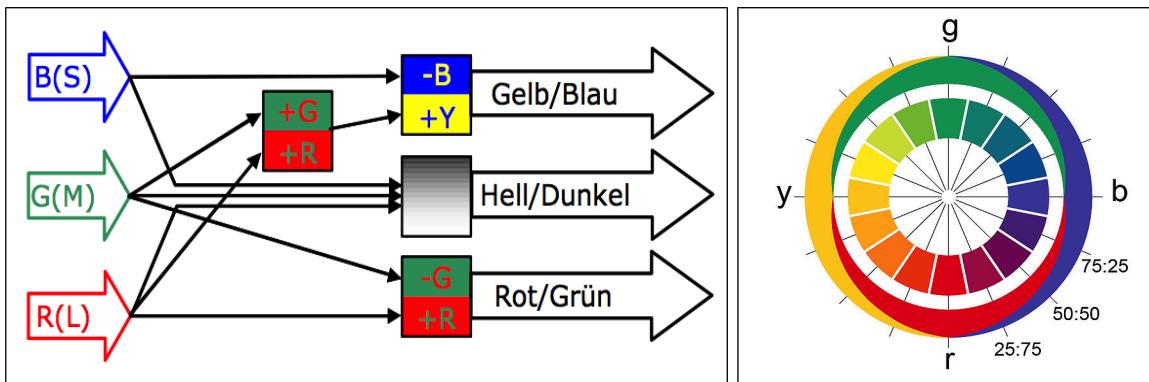


Abb. 19: Schema der Gegenfarbenumkodierung und Herings Gegenfarbenkreis

2.1.2.5 Kontrastempfindlichkeit

Die Kontrastempfindlichkeit der Netzhaut führt dazu, dass die subjektiv empfundene Helligkeit einer Fläche von deren Nachbarschaft beeinflusst wird. Z.B. werden die nachfolgenden Kreuzungen dunkler gesehen, weil der aufhellende Einfluss der benachbarten dunklen Quadrate schwächer ist als entlang der Linien.

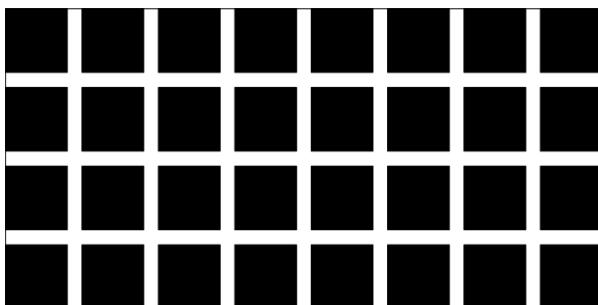


Abb. 20: Optische Täuschungen durch Kontrastbildung in der Netzhaut

Warum das? Ganglionzellen der Netzhaut definieren rezeptive Felder. Hier werden rund 130 Mio. Rezeptoren zu rund 1 Mio. Ganglionzellen zusammengefasst. Der im Zentrum eines Feldes registrierte Reiz wird durch gleichartige Reize am Rand des Feldes geschwächt. Das ans Gehirn weitergeleitete Signal entspricht der Summe der registrierten Signalwerte multipliziert mit den Gewichtsfaktoren des rezeptiven Feldes. Mit der Annahme, dass schwarze Flächen dem Signalwert 0 und die weißen Flächen dem Signalwert 1 entsprechen, ergibt sich folgende Rechnung:

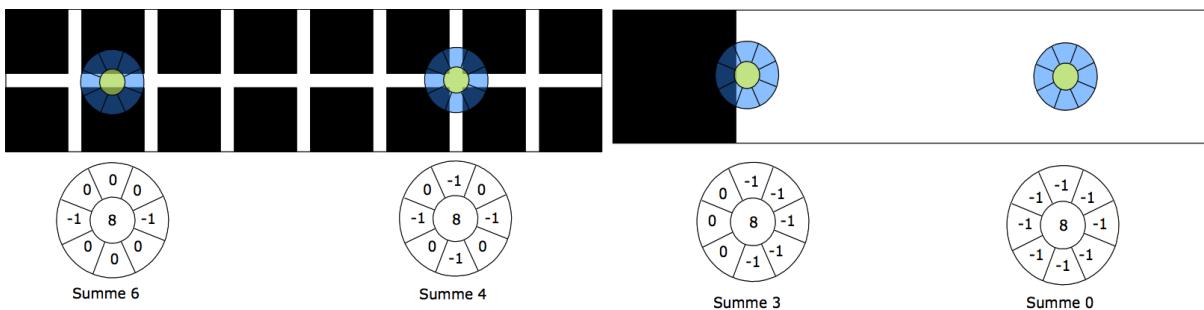


Abb. 21: Resultierende Intensitätswerte an Kanten oder auf der Fläche.

Und der Zweck des Ganzen? Eine gleichmäig helle Fläche produziert keinen Reiz, nur am Rand entsteht die Information: "Hier beginnt eine Fläche". Wie wir später im Kapitel über lokale Filter lernen werden, entspricht diese lokale Umgewichtung genau einer Kantendetektion. Unser Auge schickt also nur Kanten weiter ans Hirn.

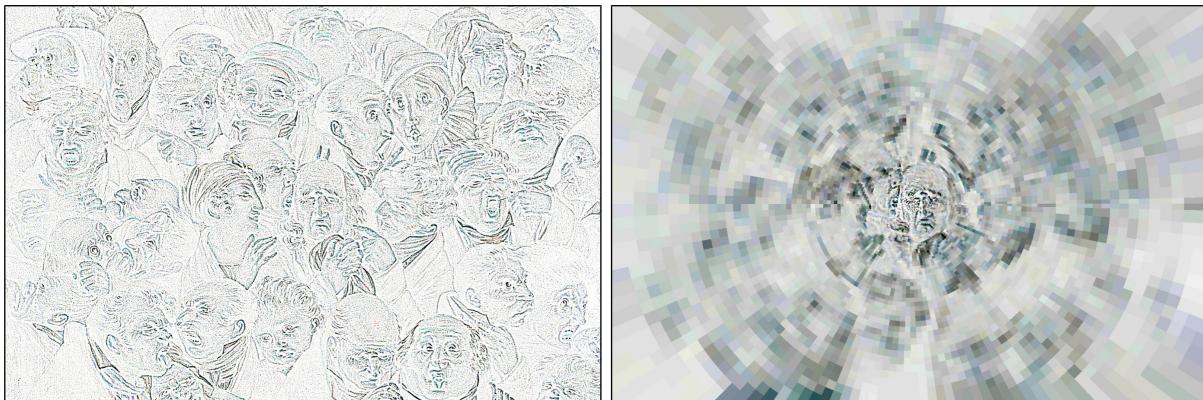


Abb. 22: Kantenbild von Louis Boilly's Thirty-Six Faces links und wiederum die logarithmische Polartransformation davon. (Bildquelle: Peters07]

2.2 Farbmodelle

2.2.1 RGB-Farbmodell

Das RGB-Farbsystem kann man sich als 3D-Koordinatensystem vorstellen, mit einer Rot-, einer Grün-, und mit einer Blauachse. Jedem Punkt in diesem Farbraum ist eine Mischfarbe zugeordnet. Farben können wie im Vektorraum durch Vektoralgebra berechnet werden.

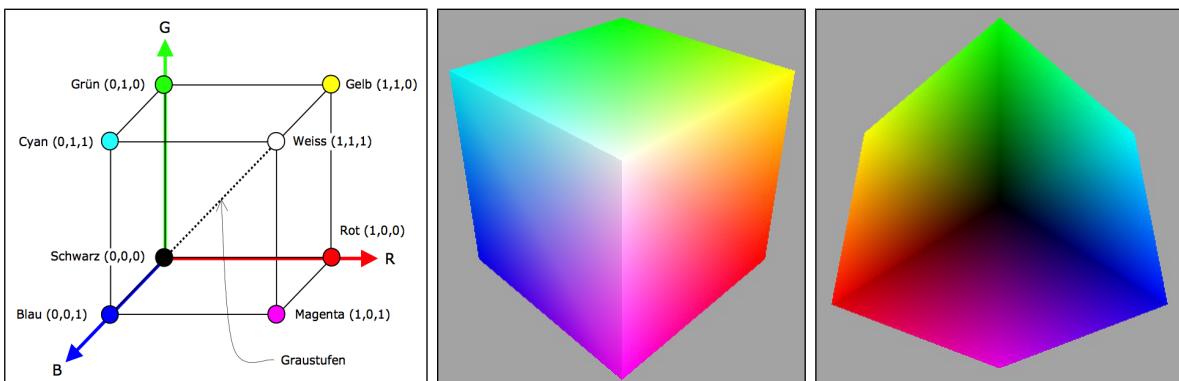


Abb. 23: RGB-Würfel mit Farbkoordinaten, RGB-Würfel (Sicht auf Weiss), RGB-Würfel (Sicht auf Schwarz)

Aus Rot und Grün entsteht Gelb, aus Blau und Grün wird *Cyan (Türkis)* und aus Rot und Blau wird *Magenta (Purpur)*. Alle drei Farben zusammen ergeben Weiss: Wenn alle Zapfentypen etwa gleichmässig gereizt werden, schliesst unser visuelles System auf ein ausgewogenes Lichtspektrum und ordnet dieser Mischung den Eindruck Weiss oder neutral zu. In der Abb. 25 sehen wir 7 Farben: die 3 Grundfarben Rot, Grün und Blau, die drei 2er-Kombinationen Cyan, Magenta und Gelb und Weiss in der Mitte. Mischfarben werden immer durch Addition von Lichtkomponenten erzeugt. Daher der verwenden wir den Begriff der *additiven Farbmischung*, wenn wir im RGB-Farbsystem arbeiten.

Gegenüberliegende Farben werden als *Komplementärfarben* bezeichnet, deren Addition zueinander immer Weiss ergibt. Eine Grundfarbe liegt immer einer Mischfarbe gegenüber. Die Mischfarbe besteht demnach aus Weiss, dem die Spektralkomponente der gegenüberliegenden Grundfarbe fehlt.

2.2.2 CMY(K)-Farbmodell

Wir können Rot, Grün und Blau nicht zum Drucken brauchen, da sie übereinander gedruckt unansehnliche Grün- und Brauntöne produzieren.

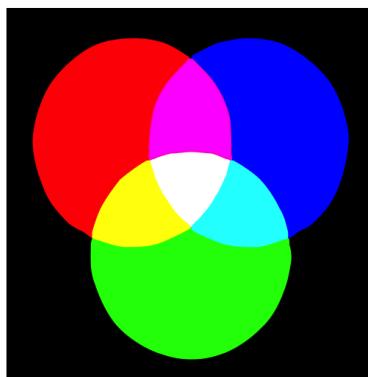


Abb. 24: RGB mit Licht

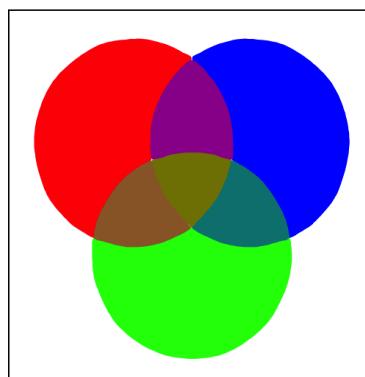


Abb. 25: RGB als Farben gedruckt

Die Lösung heisst hier *subtraktive Farbmischung*. Die Idee dahinter besteht darin, dass wir den Rot-, Grün- und Blauanteil des vom bedruckten Papier zurückreflektierten Lichts steuern, indem wir die entsprechenden Gegenfarben absorbieren.

- Cyan absorbiert den Rotanteil.
- Magenta absorbiert den Grünanteil.
- Gelb absorbiert den Blauanteil.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Der Umrechnung aus dem RGB-Farbmodell ist denkbar einfach:

Wir erinnern uns an das Beispiel mit Magenta: Magenta ist Weiss, dem der Grünanteil fehlt. Also ist Magenta der Farbstoff, den wir brauchen, um Grün aus dem weissen Umgebungslicht zu absorbieren.

Bei der subtraktiven Farbmischung fragen wir, wie viel Rot, Grün und Blau wir sehen wollen. Dann entfernen wir die überschüssigen Lichtanteile, indem wir die richtige Menge der jeweiligen Komplementärfarben aufs Papier drucken. Beim Zusammenmischen von Farbstoffen addieren sich die Absorptionen. Daher können wir einfach die Gegenfarben der Grundfarben nehmen und zusammenmischen. Alle drei Farben zusammen ergeben Schwarz, die Gegenfarbe zu Weiss, d. h., alles Licht wird absorbiert.

Druckfarbe	absorbiert	reflektiert
Cyan	Rot	Blau, Grün
Magenta	Grün	Blau, Rot
Gelb	Blau	Rot, Grün
Schwarz	alles	nichts
Weiss	nichts	alles

An der Brown Universität gibt es ein wunderbares [Java-Applet](#) dazu.

Das CMY-Farbsystem hat ein kleines Problem:

Das Schwarz, das beim übereinander drucken aller 3 Farben entsteht, ist nie ganz dunkel, da es schwierig ist, die 3 Druckfarben zuverlässig so aufeinander abzustimmen, dass wirklich das ganze Spektrum eliminiert wird. Meist bleibt ein sehr dunkler Grünton erhalten, der z. B. beim Ausdruck von Text auf einem farbigen Dokument stört. Farbdrucker der ersten Generation z. B. HP500C hatten dieses Problem.

Um Schwarz zuverlässig zu drucken, wird es als vierte Druckfarbe zum CMY-Farbsystem hinzugefügt. Damit haben wir das CMYK-Farbsystem (k für *key(note)* Grundton), wie es für den Vierfarbendruck auf Offset-Maschinen eingesetzt wird. Auch Computerdrucker verwenden inzwischen dieses Farbsystem. Dazu werden eine 3-Farben-Patrone und eine Schwarzpatrone eingesetzt. Dies ist natürlich auch viel ökonomischer, da oft auch nur schwarz-weiss Text auf Farbdruckern gedruckt wird.

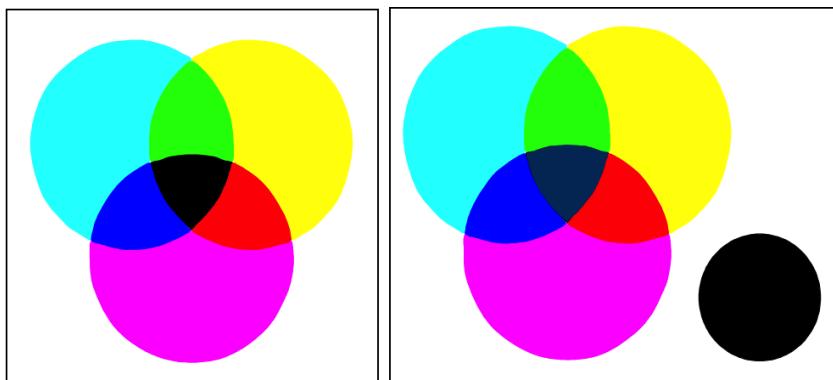


Abb. 26: CMY-Farbmodell und CMYK-Farbmodell

Der mathematische Zusammenhang mit dem RGB-Farbmodell ist wie folgt:

$$\begin{bmatrix} C_K \\ M_K \\ Y_K \\ K \end{bmatrix} = \begin{bmatrix} C \\ M \\ Y \\ 0 \end{bmatrix} + \begin{bmatrix} -\min(C, M, Y) \\ -\min(C, M, Y) \\ -\min(C, M, Y) \\ \min(C, M, Y) \end{bmatrix}$$

Beim Vierfarbendruck kann Schwarz eingesetzt werden, um gleiche Anteile der Buntfarben (Grau) zu ersetzen. Dabei verschwindet die Buntfarbe mit dem kleinsten Anteil und wird zusammen mit den entsprechenden Anteilen der anderen beiden Buntfarben durch Schwarz ersetzt. Dieses Verfahren wird *Undercolor Removal* genannt. Damit müssen immer nur maximal 3 Farben auf eine Stelle gedruckt werden, da immer eine der 3 Buntfarben aus der Kombination verschwindet.

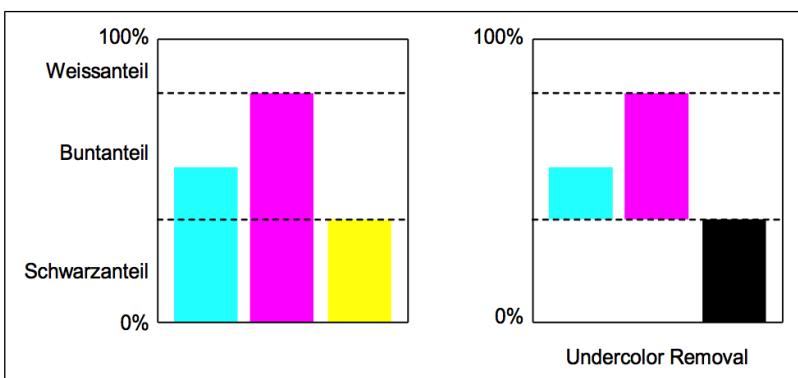


Abb. 27: Undercolor Removal

2.2.3 HSV-Farbmodell

Die RGB- und CMY(K)-Farbmodelle orientieren sich an der Biologie des Auges bzw. an deren Nachbildung in der Computerhardware. Für einen intuitiven Umgang mit Farbe sind sie nicht besonders geeignet. Dafür wurde das *HSV*-Modell entwickelt. Es beschreibt den Farbraum mit einem Farbton (engl. *Hue*, H), mit einer Sättigung (engl. *Saturation*, S), und mit einem Leuchtdichtewert (engl. *Value*, V).

Das HSV-Modell wurde 1978 von Ray Smith entwickelt. Er beschreibt es, ausgehend vom RGB-Raum, als eine sechseckige, auf dem Kopf stehende Pyramide mit der Höhe = 1.0 und Grundkantenlänge = 1.0. Man kann dabei den RGB-Würfel bzw. seine Teilwürfel entlang der Raumdiagonalen von Weiss nach Schwarz betrachten.

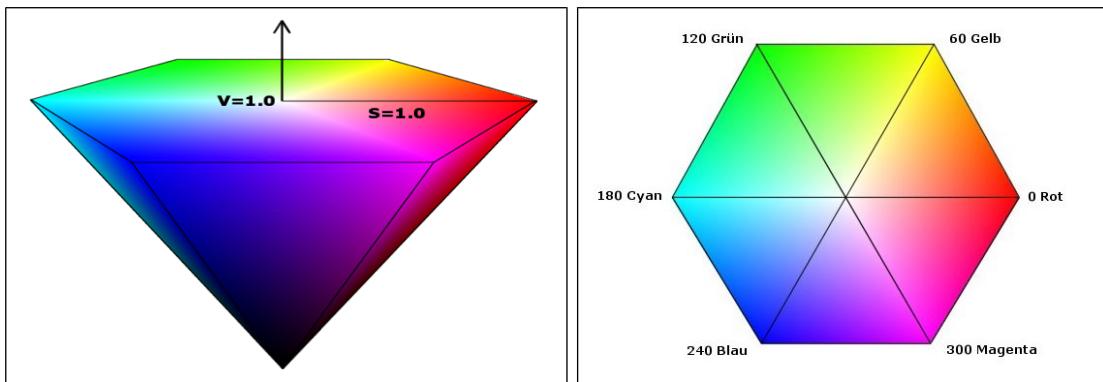


Abb. 28: HSV Pyramide nach Ray Smith in der Ansicht und von oben

Dabei sind: $V = \max(R, G, B)$

$$S = V - \min(R, G, B) / V$$

H wird als Winkel angegeben und ist z. B. im Bereich von 0°-60° auf den Fall $R = \max(R, G, B)$ festgelegt durch:

$$H = (G - B) / (R - \min(R, G, B)) * 60^\circ.$$

V entspricht der Höhe gemessen an der Achse der Pyramide. S entspricht der Distanz zur Achse. Ist S=0, so haben wir eine Graustufe zwischen Schwarz (V=0.0) und Weiss (V=1.0) und H darf nicht definiert sein.

2.2.4 HSL-, HSB- und HSI-Farbmodelle

Ähnlich zum HSV-Modell gibt es noch das HSL-, HSB- und HSI-Farbmodell. Die ersten beiden Komponenten *H* und *S* sind in allen gleich definiert. Nur die dritten Komponenten *V* (*value* bei HSV), *L* (*luminance* bei HSL), *B* (*brightness* bei HSB) und *I* (*intensity* bei HSI) sind unterschiedlich. Sie sind aber immer ein gewichteter Mittelwert aller drei Farbkomponenten R, G und B, also ein Mass der Helligkeit.

2.2.5 YUV- und YCbCr-Farbmodelle

Das *YUV*-Farbmodell entstand bei der Entwicklung des europäischen *PAL (Phase Alternating Line)* Standard. Beim amerikanischen Farbfernsehstandards *NTSC (National Television Standards Committee)* heisst ein ähnliches Farbmodell *YIQ*. Die wichtigsten Ziele dieser Farbmodelle ist eine Reduzierung der Bandbreite für den Übertragungskanal und die Kompatibilität zum S/W-Fernsehstandard.

Wie auch immer, rohe, ungefilterte Farbvideodaten brauchen eine enorme Bandbreite im Übertragungskanal. Fernsehsender können eine solche Kapazität nicht bieten. Zudem ist das menschliche Auge gar nicht in der Lage, bewegte Farbbilder derart scharf und detailliert zu erfassen wie stehende oder Schwarzweissbilder (s. Verteilung der Sinneszellen auf der Netzhaut). Eine bewegte farbige Szene wird eher wie ein grob kolorierter Schwarzweissfilm wahrgenommen, ausser man schaut genau hin (Fovea!) und betrachtet ein stehendes Bild (Computermonitor!) über längere Zeit.

Dabei werden die Bilddaten in ein Helligkeitssignal (*Luminanz*) Y und 2 Farbsignale (*Chrominanz*) U und V umgerechnet. Das Y-Signal repräsentiert die mittlere Helligkeit eines Bildes und setzt sich aus den gewichteten Anteilen der Rot- Grün- und Blautönen zusammen. Die Gewichtung entspricht der empfundenen Helligkeit der Grundfarben. Die U-Komponente ist die Differenz vom Blauanteil zur Helligkeit und die V-Komponente ist die Differenz vom Rotanteil zur Helligkeit:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

$$U = (B - Y) \cdot 0.493$$

$$V = (R - Y) \cdot 0.877$$

Die Y-Komponente wird mit hoher Bandbreite (d. h. mit voller Auflösung) übertragen und kann am Empfangsort mit einem Schwarzweissmonitor dargestellt werden. Die U- und V-Komponenten werden mit geringer Bandbreite (d. h. z. B. mit halber Auflösung) mit gesendet und werden am Empfangsort wieder mit dem Y-Signal kombiniert. Die Farbmerkmale des so übertragenen Bildes sind etwas verschwommen gegenüber den Konturen des unterliegenden Schwarzweissbildes, was aber den Betrachter nicht stört, weil sein Auge für ihn die Interpretation und Assoziation sehr schnell vornimmt.

In der Literatur und im Web finden sich mehrere verschiedene Umrechnungsvarianten. Nach de.wikipedia.org berechnet sich die

RGB-YUV- bzw. YUV-RGB-Konvertierung für 8-Bit RGB wie folgt:

```

Y = round( 0.257*R + 0.504*G + 0.098*B) + 16
U = round(-0.148*R - 0.291*G + 0.439*B) + 128
V = round( 0.439*R - 0.368*G - 0.071*B) + 128

C = 1.164*(Y-16); D = U-128; E = V-128
R = clip(round(C           + 1.596*E))
G = clip(round(C - 0.391*D - 0.813*E))
B = clip(round(C + 2.018*D           ))

```

Die Funktion `clip()` macht dabei Werte <0 zu 0 und Werte >255 zu 255. Eine schnellere Umrechnung (Annäherung) ohne runden und mit Ganzzahloperationen geht wie folgt:

```

Y = (( 66*R + 129*G + 25*B + 128)>>8) + 16
U = ((-38*R - 74*G + 112*B + 128)>>8) + 128
V = ((112*R - 94*G + 18*B + 128)>>8) + 128

C = 298*(Y-16)+128; D = U-128; E = V-128
R = clip((C           + 409*E) >> 8)
G = clip((C - 100*D - 208*E) >> 8)
B = clip((C + 516*D           ) >> 8)

```

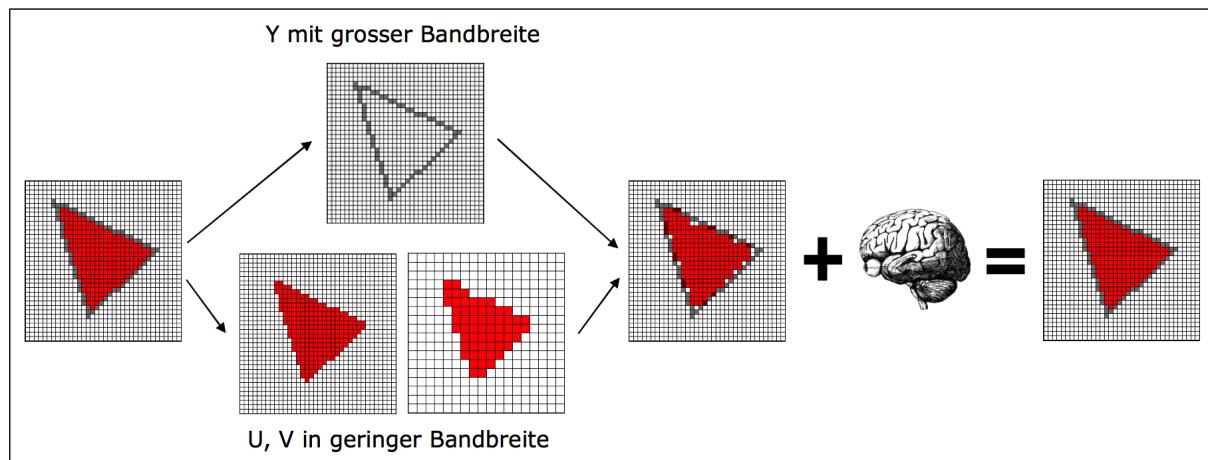


Abb. 29: Signalübertragung mit YUV-Farbmodell

YCbCr-Farbmodell

Das analoge YUV-Farbformat wurde für das Digitalfernsehen im Format **YCbCr** angepasst. Es wird neben der TV-Signalübertragung in praktisch allen Video- und Rasterbildformaten verwendet. Dazu gehören z. B. die Videoformate DV, MPEG1-4 und das Rasterbildformat JPEG. Das YCbCr-Modell teilt die Farbinformation in die Grundhelligkeit Y und die zwei Farbkomponenten Cb (*Blue-Yellow Chrominance*) und Cr (*Red-Green Chrominance*) auf.

Liegen die RGB-Daten bereits digital im Wertebereich [0-255] vor, so erfolgt die Umrechnung nach folgenden Formeln:

```

Y = round( 0.299*R + 0.587*G + 0.114*B)
Cb = round(-0.169*R - 0.331*G + 0.500*B) + 128
Cr = round( 0.500*R - 0.419*G - 0.081*B) + 128

```

2.2.6 CNS-Farbmodell

Zur verbalen Beschreibung einer Farbe eignet sich das *CNS*-Modell (*Color Name System*).

Der Farbton wird beschrieben mit:

Red, Orange, Yellow, Green, Blue, Purple

Zwischen zwei benachbarten Grundtönen können noch drei Zwischenfarbtöne beschrieben werden. Zwischen *Yellow* und *Green* wären das:

Yellowish-Green, Yellow-Green, Greenish-Yellow

Die Sättigung wird beschrieben mit:

Grayish, Moderate, Strong, Vivid

Die Helligkeit wird beschrieben mit:

Very Dark, Dark, Medium, Light, Very Light

Die achromatische Skala besteht aus den sieben Grautönen:

Black, Very Dark Gray, Dark Gray, Gray, Light Gray, Very Light Gray, White

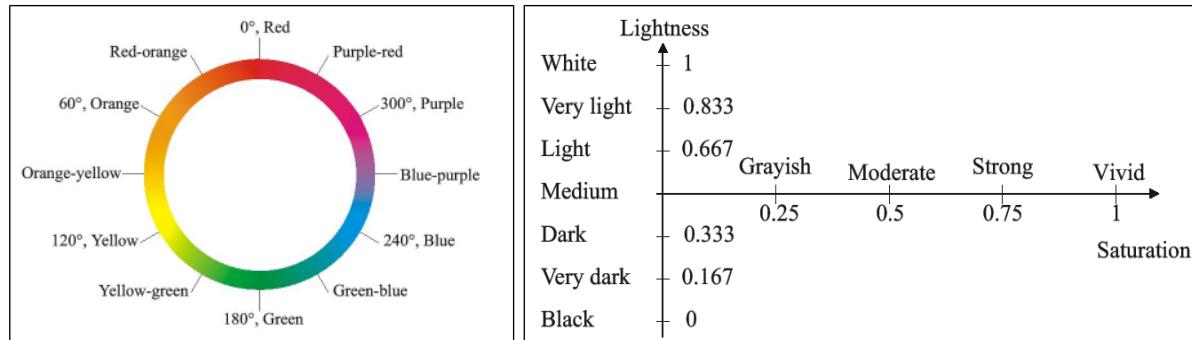


Abb. 30: Farbtondefinition beim CNS-Farbmodell

2.3 Werkzeuge für die Computer Vision

Es gibt viele Werkzeuge für die digitale Bildverarbeitung, was die Auswahl für ein bestimmtes Projekt nicht einfach macht. Schlussendlich wird meistens das Projektziel die Werkzeugwahl bestimmen. Manche Ziele lassen verschiedene Werkzeuge zu, öfter aber ist auch das schnellste Werkzeug noch zu langsam. Die Wahl des Werkzeugs hängt von einigen Kriterien ab:

- **Wahl der Programmiersprache:** Wer nur in einer Programmiersprache entwickeln kann oder will, für den stellt sich diese Auswahl nicht. Man kann in fast jeder Programmiersprache mehr oder weniger performante Bildverarbeitungsprogramme entwickeln. In fast jeder Sprache gibt es auch Anbindungen an performante Bibliotheken, die meistens in C/C++ geschrieben sind.
- **Wahl der Bildverarbeitungsbibliothek:** Selten können oder müssen alle Bildverarbeitungsaufgaben selbst entwickelt werden. Es stehen viele Softwarebibliotheken zur Verfügung, die sowohl kostenpflichtig als auch gratis sein können. Diese Bibliotheken lösen klassische Aufgaben meist besser und schneller, als wenn man sie selber programmieren würde. Zu den bekannteren Bibliotheken gehören z. B. [OpenCV](#), [CImg](#), [ImageMagick](#), [aforge](#) oder [ImageJ](#). Leider besteht fast kein Projekt nur aus klassischen Teilschritten, sodass meistens auch eigene, ganz spezifisch auf die Aufgabenstellung zugeschnittene Algorithmen entwickelt werden müssen.
- **Entwicklungsgeschwindigkeit:** Die Entwicklungsgeschwindigkeit nimmt tendenziell von kompilierten Sprachen wie C/C++, über managed Sprachen wie Java oder C# bis hin zu Skriptsprachen wie z. B. Python zu. Umgekehrt nimmt die reine Rechen-Performance ab. Zu den Skriptsprachen zählen auch wissenschaftliche Werkzeuge wie z. B. [Matlab](#), [Octave](#) oder [SciLab](#). Noch schneller lassen sich Bildverarbeitungslösungen mit sogenannten Rapid Prototyping Werkzeugen entwickeln wie z. B. [LabView](#) oder unserem [ImagePlay](#). Damit lassen sich Lösungen als Prozesskette über eine rein visuelle Konfiguration realisieren.
- **Performance:** Eine hohe Ausführungsgeschwindigkeit ist nicht immer notwendig. Will man nur ein Einzelbild verarbeiten, so kommt es nicht drauf an, ob dies eine Zehntelsekunde oder 2 Sekunden dauert. Will man aber die Bilder einer Videokamera in Echtzeit verarbeiten, so hat man in der Regel nur wenige Millisekunden dazu. Je höher die Performance sein muss, um so mehr muss man sich mit der Hardware auseinandersetzen oder ein hardwarenahes Werkzeug wie z. B. [OpenCV](#), [IPP](#), [CUDA](#) oder [OpenCL](#) wählen. Wenn auch diese Tools noch zu langsam sind, so bleibt nur noch eine Umsetzung in Hardware übrig, z. B. mit [FPGA](#)- oder [DSP](#)-Programmierung.

Auch mit diesen Kriterien fällt die Entscheidung über das Werkzeug der Wahl nicht einfach. In Industrieprojekten wird deshalb oft zuerst die Machbarkeit einer Lösung mit einem Rapid Prototyping Werkzeug überprüft, um sie danach in einer hardwarenäheren Sprache zu beschleunigen.

Um eine möglichst breite Sicht zu erhalten, möchten wir deshalb im Unterricht mit **ImagePlay** ein Rapid Prototyping Werkzeug, mit **Matlab** ein Werkzeug aus der Kategorie der wissenschaftlichen Werkzeuge, mit **ImageJ** eine managed Library in Java und mit **OpenCV** ein auf Performance optimierte Bibliothek kennenlernen.

2.3.1 ImagePlay



[ImagePlay](#) ist ein Rapid Prototyping Bildverarbeitungswerkzeug, das in unserer Gruppe cpvrLab entwickelt wird. Es wurde ursprünglich von Roger Cattin einem ehemaligen Bildverarbeitungsdozenten der BFH speziell für den Unterricht entwickelt. Sie finden den Quellcode unter [Github](#) sowie die aktuellsten Setups und [imagplay.io](#). Im [Wiki von ImagePlay](#) finden Sie zusätzliche Informationen wie Sie eigene Plugins für ImagePlay erstellen können.

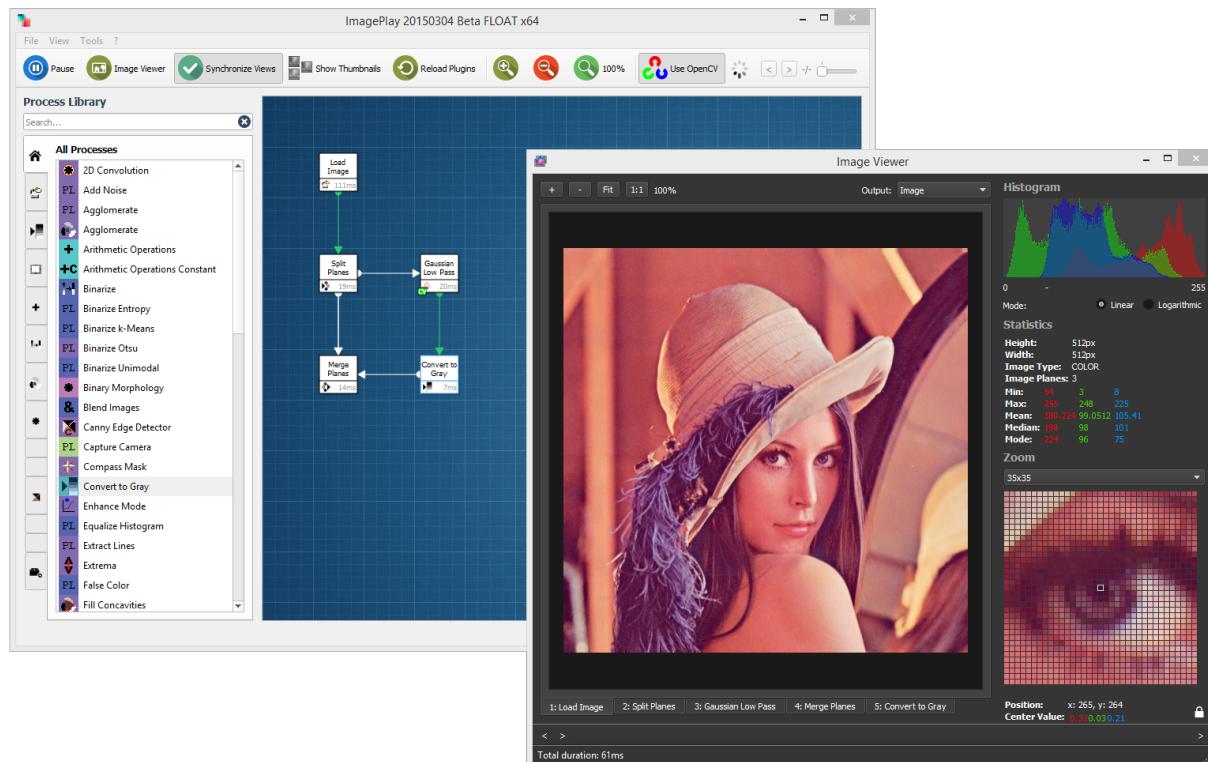


Abb. 31: ImagePlay Prozess Editor im Hintergrund und der Image Viewer im Vordergrund.

2.3.2 Matlab



[Matlab](#) ist eine Abkürzung für *Matrix Laboratory* und ist ein Softwarepaket für numerische Berechnungen mit Vektoren und Matrizen. Weil Bilder auch als Matrizen interpretiert werden können, eignet sich Matlab mit seiner speziellen [Image Processing Toolbox](#) auch gut für die Bildverarbeitung. Weitergehende Lösungen im Bereich der Bildanalyse und der Computer Vision finden sich in der [Computer Vision System Toolbox](#).

Matlab ist mittlerweile gratis für Studenten der BFH. Sie müssen aber ein Studentenkonto bei [Mathworks](#) erstellen. Eine genaue Anleitung für die Installation finden Sie auf dem [Intranet der BFH](#). **Installieren Sie mindestens folgende Toolboxen:**

- [Image Processing Toolbox](#)
- [Computer Vision System Toolbox](#)
- [Statistic and Machine Learning Toolbox](#)

Dieses Skript ist keine Einführung in die grundlegenden Arbeitstechniken mit Matlab. Es gibt jedoch viele Tutorials im Internet und eine sehr gute Matlab-Dokumentation. Eine Zusammenfassung in Deutsch finden Sie mit den Suchbegriffen „Matlab elementare Bildverarbeitung“.

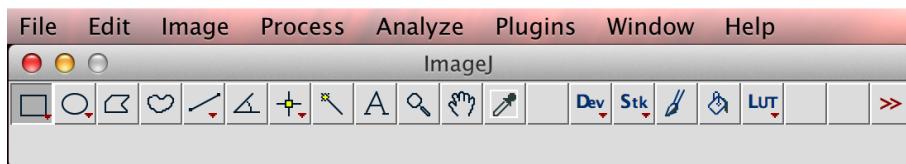
2.3.3 ImageJ



[ImageJ](#) bietet einerseits bereits fertige Werkzeuge zur Darstellung und interaktiven Manipulation von Bildern, andererseits lässt es sich sehr einfach durch eigene Softwarekomponenten erweitern. ImageJ ist komplett in Java geschrieben und ist damit weitgehend plattformunabhängig. Die dynamische Struktur von Java erlaubt es, eigene Plugins in Form eigener Java Klassen zu erstellen und im laufenden System zu übersetzen und auch sofort auszuführen, ohne ImageJ neu starten zu müssen. Dieser Ablauf macht ImageJ zu einer idealen Basis, um neue Bildverarbeitungsalgorithmen zu entwickeln und damit zu experimentieren.

2.3.3.1 Eingebaute ImageJ Werkzeuge

ImageJ als eigenständiges Programm bietet von Haus aus folgende Funktionalität:



- **File**: Funktionen zum Öffnen, Speichern, Importieren und Drucken von Bildern.
- **Edit**: Klassische Bearbeitungsfunktionen wie Copy, Paste sowie Löschen, Füllen und Selektieren von Bildbereichen. ImageJ Optionen.
- **Image**: Funktionen für Bildtypenumwandlung, Bildinformationen, Beschneidung, Skalierung sowie Werkzeuge für Helligkeits-, Kontrast-, Farbton- und Farbsättigungs-korrektur.
- **Process**: Typische Bildverarbeitungsfunktionen für Punktoperationen sowie lokale und globale Filteroperationen.
- **Analyze**: Statistische Bildauswertung über Histogramm sowie spezielle Darstellungen.
- **Plugins**: Funktionen zum Bearbeiten, Erstellen, Verwalten und Starten von Plugins.

2.3.3.2 Plugins für ImageJ

ImageJ Plugins sind Java-Klassen, die ein vorgegebenes Interface implementieren. Es gibt zwei Arten von Plugins:

- **PlugIn** benötigt keinerlei Argumente, kann daher auch ohne Bild ausgeführt werden.
- **PlugInFilter** wird beim Start immer ein Bild (das aktuelle Bild) übergeben.

Für die Übungen verwenden wir fast ausschliesslich das Filter-Plugin. Das Interface hat folgende Methoden:

- **int setup (String arg, ImagePlus img)**: Diese Methode wird beim Start von ImageJ als erstes aufgerufen, um zu prüfen, ob die Spezifikationen des Plugins mit dem übergebenen Bild zusammenpassen. Die Methode liefert einen Integerwert zurück, der die Eigenschaften des Plugins beschreibt.
- **void run (ImageProcessor ip)**: Diese Methode erledigt die eigentliche Arbeit des Plugins. Der einzige Parameter *ip* vom Typ *ImageProcessor* enthält das zu bearbeitende Bild und alle relevanten Informationen dazu.

Nachfolgend ein kleines Beispiel eines Filter-Plugins, welches das Input-Bild invertiert. Der Klassename eines ImageJ Plugins muss immer mindestens ein `_` enthalten:

```

import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class Inverter_ implements PlugInFilter
{
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G; // Allows only 8-Bit grayscale images
    }

    public void run(ImageProcessor ip)
    {
        // Get the byte array of pixels
        byte[] pixels = (byte[]) ip.getPixels();

        // invert the pixel values
        for (int i = 0; i < pixels.length; i++)
            pixels[i] = (byte) (255 - pixels[i]);
    }
}
  
```

2.3.3.3 Entwicklungsumgebungen für Plugins

Es gibt verschiedene Möglichkeiten Plugins für ImageJ zu entwickeln:

- **In ImageJ mit dem eingebauten Editor:** Der integrierte Editor ([Menü Plugins > New](#)) eignet sich gut für kleine Plugins, um schnell etwas zu testen. Er bietet kein Syntax-Highlighting und man kann auch nicht debuggen. Mit [Plugin > Compile and Run](#) kann das Plugin kompiliert und gestartet werden. Das Plugin wird dabei im Ordner Plugins des ImageJ Installationspfads installiert.
- **In Eclipse mit ImageJ-Start:** Wenn man sein Plugin debuggen will, kann man es auch in der Entwicklungsumgebung [Eclipse](#) entwickeln. Die Methode, die im [ImageJ-Wiki](#) beschrieben ist, bindet ImageJ als Applikation ein. Mit einem Ant-Skript wird das Plugin nach dem Kompilieren in den ImageJ/Plugin-Ordner kopiert und ImageJ wird aus Eclipse heraus gestartet.
- **In Eclipse ohne ImageJ-Start:** Wer ohne das Aufstarten von ImageJ arbeiten möchte, kann ImageJ auch als JAR-Datei einbinden. Sie können dies tun, indem Sie die [ij.jar](#) Datei aus dem ImageJ-Verzeichnis über das Kontextmenü beim Projekt hinzufügen. Der Befehl im Kontextmenü auf dem Projekt lautet: [Build Path > Add External Archives ...](#). Es braucht so kein Ant-Skript und die Quelldatei kann in einem beliebigen Verzeichnis sein. Zum obigen Beispiel braucht es dann nur noch die statische main-Funktion:

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class MyPlugIn_ implements PlugInFilter
{
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    public void run(ImageProcessor ip)
    {
        byte[] pixels = (byte[]) ip.getPixels();

        // invert the pixel values
        for (int i = 0; i < pixels.length; i++)
            pixels[i] = (byte) (255 - pixels[i]);
    }

    public static void main(String[] args)
    {
        MyPlugIn_ plugin = new MyPlugIn_();
        ImagePlus im = new ImagePlus("../Images/acc.png");
        im.show();
        plugin.setup("", im);
        plugin.run(im.getProcessor());
    }
}
```

2.3.3.4 Fiji



[Fiji](#) ist eine ImageJ-Distribution. Es verhält sich etwa so wie Ubuntu zu Linux und ist eine ImageJ-Version mit z.T. stark erweiterten Funktionsumfang. Sie können problemlos Fiji und ImageJ parallel installieren.



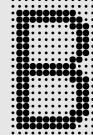
2.4 Übungen zum Kapitel Grundlagen

1. Installieren Sie die aktuellste Version von [ImageJ](#) und [Fiji](#).
2. Lesen Sie Kapitel 1 bis und mit 4 des [Tutorials „Writing ImageJ Plugins“](#).
3. Schreiben Sie ein einfaches Filter-Plugin mit den verschiedenen Methoden der Entwicklungsumgebung.



4. Falls Sie Matlab noch nicht installiert haben: Erstellen Sie ein Studentenkonto bei [Mathworks](#). Eine genaue Anleitung für die Installation finden Sie auf dem [Intranet der BFH](#). Installieren Sie mindestens die *Image Processing* und die *Statistics Toolbox*.
5. Wenn Sie noch nie mit Matlab gearbeitet haben, arbeiten Sie ein Anfänger-Tutorial für Matlab durch, z.B. [Getting Started with Matlab](#). Die offizielle Dokumentation zur *Image Processing Toolbox* als PDF finden sie [hier](#).
6. Arbeiten Sie ein Anfänger-Tutorial für die *Image Processing Toolbox* durch.

3 Bildaufnahme



Bilder in digitaler Form entstehen meistens, indem Licht auf einem lichtempfindlichen Sensor eingefangen wird und danach in einem Quantisierungsprozess digitalisiert wird. In der Wissenschaft und insbesondere in der Medizin werden aber auch bildgebende Systeme eingesetzt, die nicht sichtbare Strahlung auswerten. In diesem Kapitel beschränken wir uns aber auf den klassischen, lichtempfindlichen Flächensensor, so wie er in den meisten Kameras eingesetzt wird. Bevor das Licht auf den Sensor gelangt, passiert es allerdings immer eine Optik, die einen mehr oder weniger starken Einfluss auf die Bildqualität haben kann. Einen guten Überblick sowie detailliertere Information zur Objektiven und Sensoren finden Sie auch im [Stemmer Handbuch der Bildverarbeitung](#).

3.1 Optik

3.1.1 Camera Obscura

Obwohl das Prinzip der Vergrösserungslupe bereits um das Jahr 1000 vom islamischen Gelehrten [Alhazen](#) entdeckt wurde, erfand man die ersten optischen Apparate, wie das Fernrohr oder das Mikroskop, erst gegen Ende des 16. Jahrhunderts. Die perspektivische Abbildung mit der [Camera Obscura](#) entdeckte aber bereits Aristoteles um 350 v. Chr. Der Nachteil bei der Camera Obscura ist die lange Belichtungszeit, die notwendig ist, um genügend Licht auf der Rückseite der Kamera zu erhalten. Sobald man das Loch vergrössert, kommt zwar genug Licht rein aber man verliert die Schärfe, weil die Strahlen nicht mehr gebündelt werden. Genau dazu kann man die Sammellinse verwenden, allerdings mit dem Nachteil, dass nur Objekte in einer bestimmten Schärfeebebe fokussiert werden.

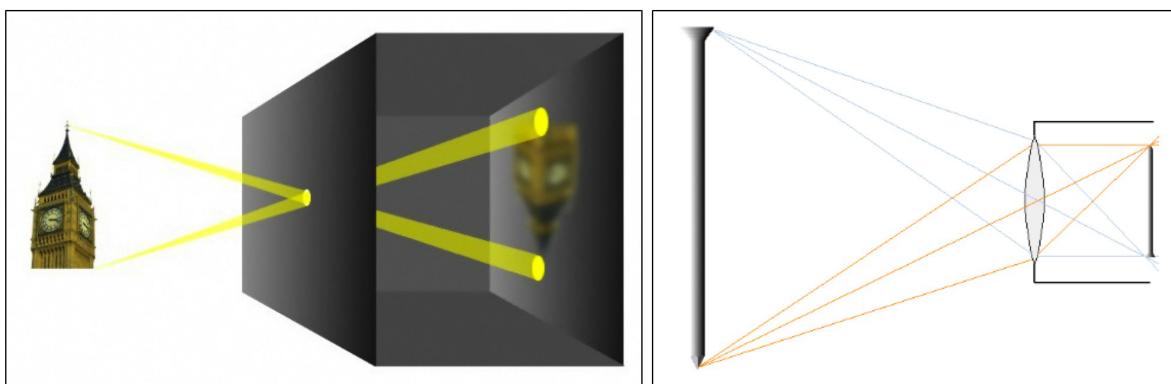


Abb. 32: Links: Camera Obscura mit der punktgespiegelten Abbildung auf der Rückseite. Rechts: Scharfe Abbildung in der Schärfeebebe, wenn das Licht durch eine Sammellinse wieder gebündelt wird.

3.1.2 Objektivkennzahlen

3.1.2.1 Brennweite und Abbildungsmassstab

- Der **Brennpunkt F** ist der Punkt, wo sich parallel einfallende Lichtstrahlen nach der Linse treffen. Hinter dem Brennpunkt befindet sich die Bildebene, in der sich auch der Sensor befinden muss.
- Die **Brennweite f** einer Optik ist der Abstand zwischen der Hauptebene der Linse und dem Brennpunkt F.
- Der **Abbildungsmassstab A** ist das Verhältnis zwischen Gegenstandsgröße G und Bildgröße B bezeichnet.

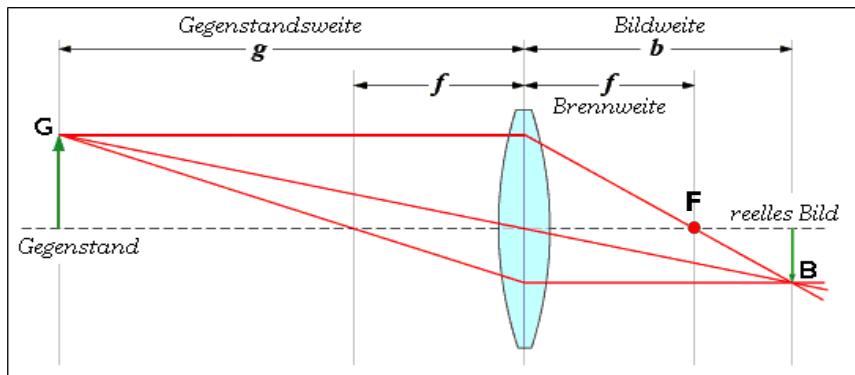


Abb. 33: Abbildung des Punktes G mit einer (dünnen) Sammellinse im Punkt P.

Der **Abbildungsmassstab** ist definiert als: $A = \frac{B}{G} = \frac{b}{g}$

Mit dem Strahlensatz kann folgende Gleichung formuliert werden: $\frac{B}{G} = \frac{b}{g} = \frac{b-f}{f}$

Durch Umformung erhalten wir die **Linsengleichung** (für dünne Linsen): $\frac{1}{b} + \frac{1}{g} = \frac{1}{f}$

Daraus lassen sich verschiedene Berechnungsformeln herleiten:

- **Brennweite bei gegebener Sensorgrösse und Gegenstandsweite:** $f = \frac{g}{\frac{G}{B} + 1}$

Beispiel: Welche Brennweite ist nötig, um bei einem 1/2" Sensor (Höhe B=6.4mm) und 300mm Gegenstandsweite ein Bildfeld von 150mm zu erfassen?
 $f = 300/(150/6.4+1) = 12.28\text{mm}$

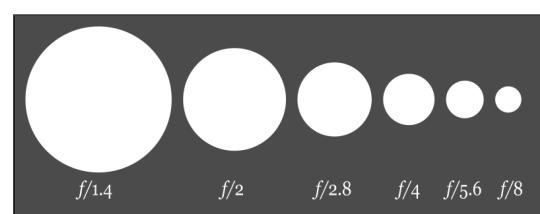
- **Gegenstandsweite bei gegebener Brennweite und Sensorgrösse:** $g = f \cdot \left(\frac{G}{B} + 1\right)$

Beispiel: Wie weit muss eine Kamera mit 12mm Brennweite und einem 1/2" Sensor (Höhe B=6.4mm) von einem 150mm grossen Objekt entfernt sein?
 $g = 12 \cdot (150/6.4+1) = 293.25\text{mm}$

3.1.2.2 Blende und Schärfentiefe

- Mit der **Blendenzahl** k wird angegeben, wie viel Licht ein Objektiv passieren kann. Sie wird berechnet aus der Brennweite f dividiert durch den Öffnungsdurchmesser D in mm:

$$k = \frac{f}{D}$$



Die Blende wird häufig als Bruch f/k angegeben und entspricht damit dem Öffnungsdurchmesser. Die Schritte zwischen den Standardblendenzahlen 1.0, 1.4, 2.0, 2.8, 4, 5.6, 8, 11, 16 und 22 reduzieren die Lichtmenge jeweils um 50%. Vorsicht: Je kleiner die Blendenzahl umso grösser ist die Lichtmenge.

- Die **Schärfentiefe** ist der Tiefenbereich, indem Objekte als scharf angesehen werden. Je kleiner die Blende (= grosse Blendenzahl), umso grösser ist die Schärfentiefe und umgekehrt. Die Schärfentiefe berechnet sich aus der Brennweite f , der Blendenzahl k , den gewünschten maximalen Zerstreuungskreisdurchmesser Z und der Gegenstandsweite g zum fokussierten Punkt P. Daraus berechnet man die Distanzen zum nahesten scharfen Punkt (Nahpunkt Q) und zum fernsten scharfen Punkt (Fernpunkt

R). Die Distanz zwischen Nahpunkt und Fernpunkt ist die Schärfentiefe. Die Grösse des Zerstreuungskreis wird bei digitalen Sensoren gleich der Pixelgrösse gesetzt. Die Herleitung der Berechnungsformeln finden Sie bei [Wikipedia](#). Als Schätzwert für die Schärfentiefe dient die Formel kann der Abbildungsmassstab herangezogen werden:

$$\text{Schärfentiefe} \approx \frac{1}{A} = \frac{G}{B} = \frac{g}{b}$$

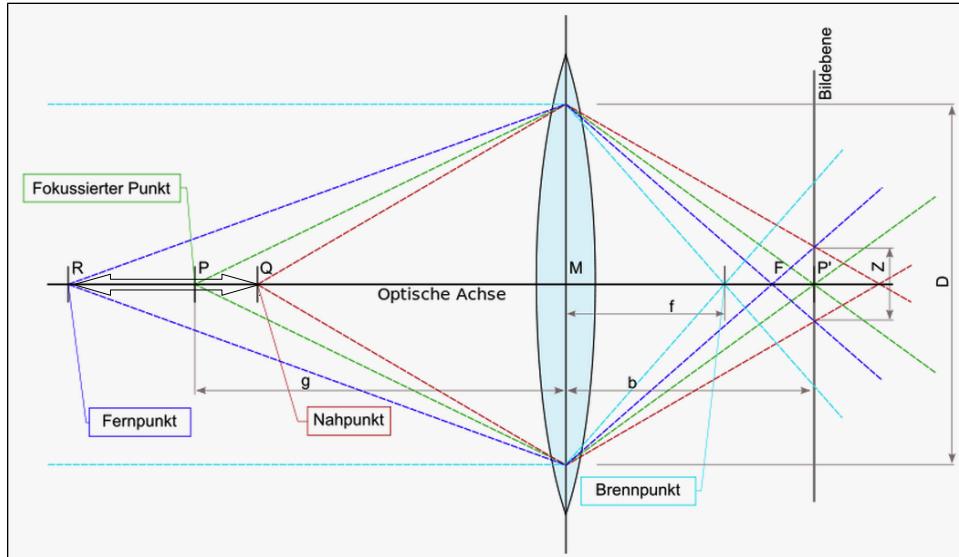


Abb. 34: Schärfentiefe zwischen Nahpunkt und Fernpunkt

3.1.3 Abbildungsfehler (Aberration)

- **Chromatische Aberration** entsteht durch die unterschiedlichen Brechungswinkel von Lichtstrahlen mit verschiedener Wellenlänge. Siehe auch Kapitel 2.1.1.3 über Refraktion. Mit farbkorrigierten Objektiven kann diesem Abbildungsfehler entgegen gewirkt werden.

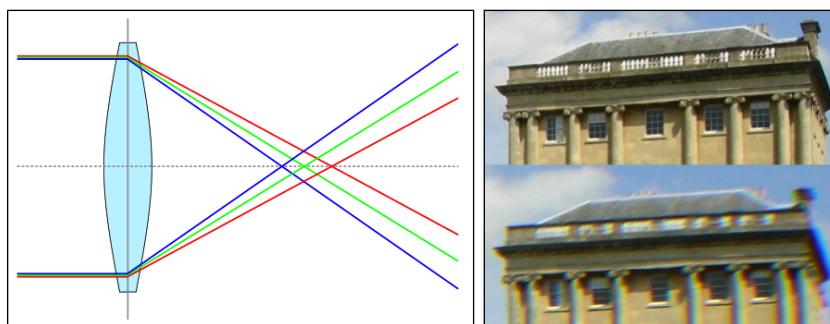


Abb. 35: Links: Die gebrochenen Strahlen der roten, grünen und blauen Strahlen treffen sich in verschiedenen Brennpunkten. Rechts oben: Aufnahme mit farbkorrigiertem Objektiv und rechts unten mit nichtkorrigiertem Objektiv.

- **Sphärische Aberration** entsteht bei genau sphärisch geschliffenen Linsen. Dies ist zwar einfach und günstig, führt aber eben zu einem ungenauen Brennpunkt. Genaue Brennpunkte können deshalb nur mit asphärisch geschliffenen Linsen erreicht werden.

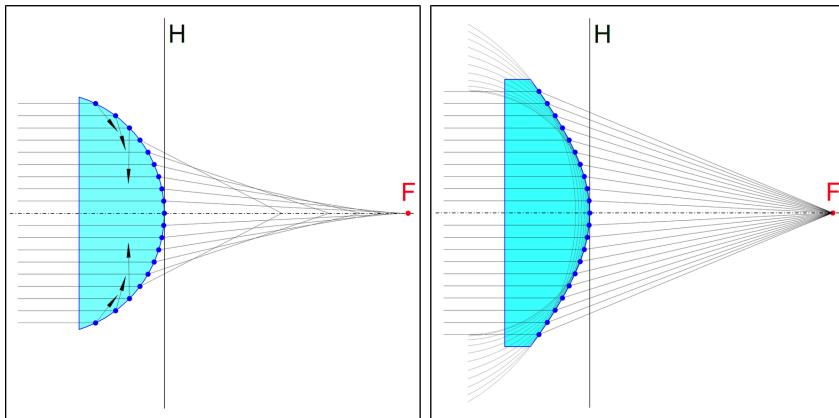


Abb. 36: Sphärisch und asphärisch geschliffene Linsen

- **Verzeichnung (Distortion)** bedeutet, dass rechtwinklige Strukturen nicht mehr rechtwinklig abgebildet werden. Je kleiner die Brennweite ist, umso grösser ist die Verzeichnung. Dies geht bis zu einem Abbildungskreis bei einem Fischaugobjektiv.

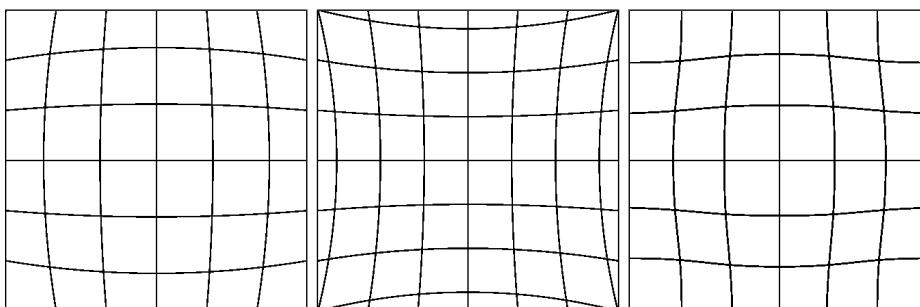


Abb. 37: Tonnenförmige (barrel), kissenförmige (pincushion) und schnauzförmige (mustashe) Verzeichnung.

- **Vignettierung** bezeichnet die Lichtabnahme zum Rand des Sensors. Dies kann durch die Optik und den Sensor verursacht sein.
 - **Mechanische Vignettierung** durch Fassungsteile des Objektivs
 - **Cos⁴-Vignettierung** entsteht durch die natürliche Lichtabnahme, wenn das Licht nicht mehr rechtwinklig auf den Sensor trifft (s. [Cos⁴-Gesetz](#)). Sie tritt deshalb umso stärker auf, je kleiner die Brennweite ist.
 - **Sensorabschattung** entsteht bei Sensoren, bei denen der lichtempfindliche Bereich in einem quadratischen Pixel selbst nicht quadratisch ist. Die Lichtabnahme ist dadurch nicht radial, sondern in x- oder y-Richtung.

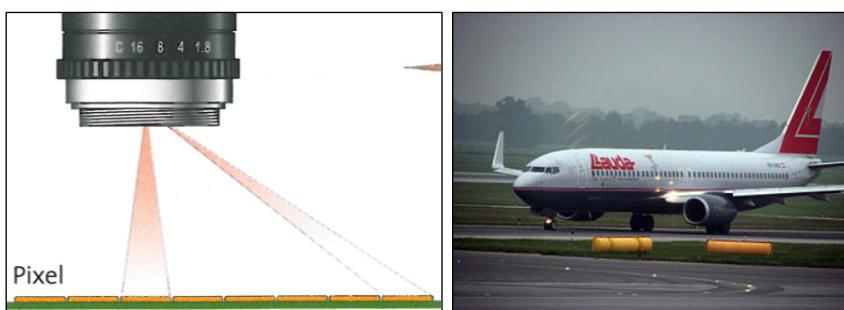
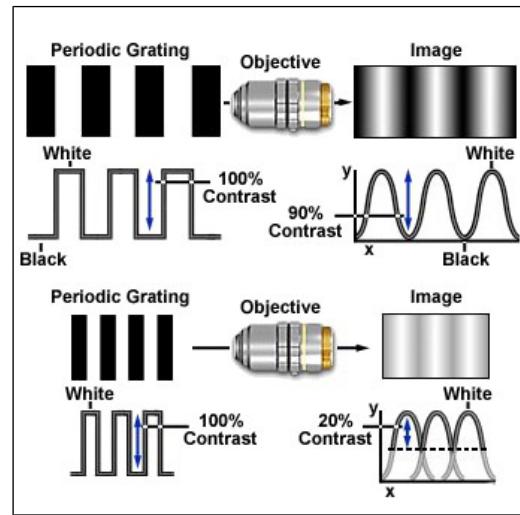


Abb. 38: Lichtabnahme gegen den Rand des Bildes

- Die **Modulationsübertragungsfunktion** (MTF: [Modulation Transfer Function](#)) ist die quantitative Beschreibung der Abbildungsgüte eines Objektivs. Zur Bestimmung werden schwarz-weiße Linienpaare mit abnehmendem Abstand dargestellt. Wenn der Kontrast der Linienpaare Null erreicht ist die **Auflösungsgrenze** erreicht. Das Mass wird in Linienpaaren pro mm (lp/mm) angegeben. Die Auflösungsgrenze nimmt wegen der verschiedenen Abbildungsfehler vom Zentrum zum Bildrand um den Faktor 2-3 ab. Grundsätzlich ist die Abbildungsgüte umso besser, je kleiner die Pixel eines Sensors sind und umso besser die Qualität des Objektivs ist. Man kann auch sagen, dass je kleiner die Pixel sind, umso besser und teurer muss das Objektiv sein.
- Bildquelle: [microscopyu.com](#)



3.1.4 Objektivtypen

- Festbrennweitenobjektive** sind Objektive mit fixer Brennweite. Sie haben z.T. nicht einmal eine Fokus- und Blendeneinstellung und können dadurch die beste Abbildungsqualität anbieten.
9. Standardobjektive haben eine MTF von 70-90 lp/mm.
10. Präzisionsobjektive haben eine MTF > 120 lp/mm.
- Zoomobjektive** bieten eine variable Brennweite mit Fokus- und Blendeneinstellung.
- Telezentrische Objektive** haben im Gegensatz zu den entozentrischen Objektiven keine perspektivische, sondern eine orthografische Projektion. Sie eignen sich deshalb gut für Vermessungsaufgaben, weil die Bildgröße damit unabhängig ist von der Gegenstandsweite.

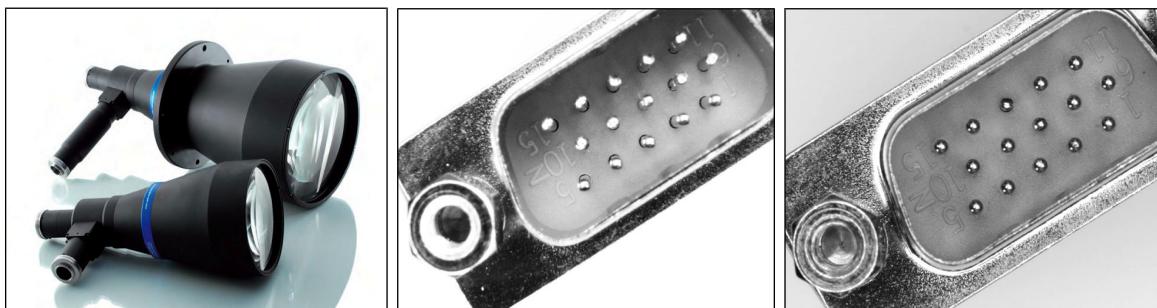


Abb. 39: Links: Telezentrische Objektive, Mitte: Abbildung mit normalem entozentrischen und rechts mit telezentrischem Objektiv.

3.1.5 Objektivgewinde

Die allermeisten Objektive und Kameras im Maschine Vision Bereich haben eines der drei Standardgewinde, um das Objektiv auf die Kamera zu schrauben:

- S-Mount:** Das M12x0.5-Gewinde (12mm) wird für die meisten Platinenkameras verwendet. Die Tiefe des Gewindes ist nicht definiert.
- C-Mount:** Das 1" x 32 TPI Gewinde (1"=1 Zoll=2.54cm mit 1/32 Steigung) ist das am meisten verwendete Gewinde. Der Sensor muss dabei 17.5mm tief im Kameragehäuse liegen.
- CS-Mount:** Gleiches Gewinde wie beim C-Mount allerdings muss der Sensor nur 12.5mm tief im Gehäuse liegen.

- F-Mount:** Dieses Gewinde kann für Kleinformatobjektive von Nikon verwendet werden. Es kommt nur selten zum Einsatz bei sehr grossen Sensoren.

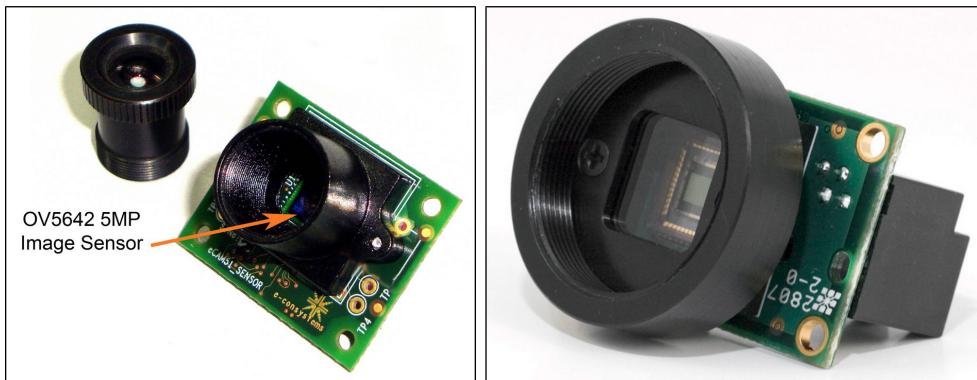


Abb. 40: Links: S-Mount-Linse und Gewinde einer Platinenkamera. Rechts: C-Mount Gewinde.

3.2 Bildsensor

In der Schärfenebene der Kamera befindet sich ein Flächensensor, der durch eine Vielzahl im Raster angeordneten, lichtempfindlichen Sensorelementen das Bildsignal auffängt. Die einfallenden Photonen werden bei der Absorption im Halbleitermaterial in Elektronen umgewandelt. In jedem dieser Potenzialtöpfe bildet sich während der Belichtungszeit ein Ladungspotenzial, das der absorbierten Lichtmenge entspricht.

3.2.1 Sensorgroesse und Auflösung

Die Grösse eines Sensors wird meistens in Zoll (1 Zoll = 1 inch = 2.54 cm) angegeben. Diese Grössenangabe kommt aus den Tagen der Röhrentechnologie der alten Fernseher und hat mit der eigentlichen Grösse nichts zu tun. Aus der nachfolgenden Tabelle können Sie die korrespondierenden Sensorgrössen entnehmen:

Sensorgrösse & Auflösung	Auflösung:	VGA	PAL	XGA	SXGA	2 MP	5MP
	Breite (px):	640	768	1024	1280	1628	2588
	Höhe (px):	480	576	768	960	1236	1958
	Pixelanzahl	307'200	442'368	786'432	1'228'800	2'012'208	5'067'304
Sensor	Breite (mm)	Höhe (mm)	Diag. (mm)	Pix.(µm)	Pix.(µm)	Pix.(µm)	Pix.(µm)
1/4"	3.65	2.74	4.56	5.7	4.8	3.6	2.9
1/3"	4.80	3.60	6.00	7.5	6.3	4.7	3.8
1/2"	6.40	4.80	8.00	10.0	8.3	6.3	5.0
2/3"	8.80	6.60	11.00	13.8	11.5	8.6	6.9
1"	12.70	9.50	15.86	19.8	16.5	12.4	9.9
						2.2	1.4
						2.9	1.9
						3.9	2.5
						5.4	3.4
						7.8	4.9

Abb. 41: Das Auge hat in der Fovea etwa 150'000 farbempfindliche Zapfen pro mm². Dies entspricht einer Pixelgrösse von etwa 2.6 µm. Das heisst, wir haben ab 2 MP auf 1/4" die menschliche Auflösung überschritten.

Eine hohe Auflösung muss nicht zwangsläufig ein schärferes und damit besseres Bild bedeuten. Kleine Pixel verursachen ein grösseres Rauschen und bedingen auch eine hochqualitative Optik.

3.2.2 Spektrale Empfindlichkeit

Die spektrale Empfindlichkeit ist bei einem Standardsensor breiter als beim Menschen. Diese Sensoren werden deshalb mit einem Filterglas abgedeckt, um die unerwünschten UV- und IR-Bereiche herauszufiltern. Für Kameras, die extra in diesen Bereichen empfindlich sein sollen, fehlen diese Filter natürlich oder es werden sogar spezielle Sensoren dafür hergestellt.

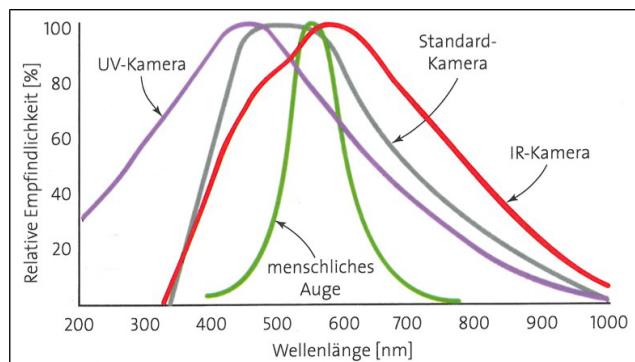


Abb. 42: Spektrale Empfindlichkeit einer UV-, IR- und Standardkamera gegenüber der menschlichen Empfindlichkeit. (Quelle: Stemmer)

3.2.3 Sensorkennzahlen

Der Standard zur Charakterisierung von Bildsensoren ([EMVA-Standard 1288](#)) formuliert einige physikalische Kennzahlen und leitet daraus ein idealisiertes mathematisches Modell ab.

- Während der Belichtungszeit kommt eine Anzahl Photonen n_p auf dem Sensor an.
- **Quantum Efficiency:** In den lichtempfindlichen Photodioden wird bei der Absorption des Photons ein Elektron erzeugt. Es werden jedoch nicht alle Photonen absorbiert. Die Wahrscheinlichkeit, dass ein einfallendes Photon absorbiert wird, wird *Quanteneffizienz* genannt. Diese Effizienz hängt von der Wellenlänge des Lichts ab und liegt bei modernen Sensoren um die 50% (fotografischer Film zum Vergleich: 10%).
- **Full Well Capacity:** Aus den Elektronen bildet sich elektrische Ladung. Eine Sensorzelle wird deshalb auch Potentialtopf (Engl. *well*) genannt. Je grösser ein Potenzialtopf ist, umso grösser ist die Sättigungskapazität. Moderne Sensoren von DSLR-Kameras (*DSLR = Digital Single Lens Reflex*) haben eine Sättigungsgrenze von bis zu 80'000 Elektronen.
- **Dynamic Range:** Das Verhältnis von Sättigungskapazität zur minimalsten detektierbaren Elektronenmenge wird Dynamikumfang bezeichnet.
- **Gain:** Die elektrische Ladung, die als Spannung ausgelesen werden kann, ist jedoch so schwach, dass sie verstärkt werden muss. Mit der Verstärkung (*Gain*) kann die Empfindlichkeit des Sensors stark beeinflusst werden. Je grösser die Verstärkung jedoch ist, umso grösser wird auch das Verstärkerrauschen.
- **A/D-Conversion:** Das verstärkte Signal wird mit einem Analog-Digital-Wandler (*A/D-Wandler*) in einen diskreten Messwert umgewandelt. Die Sättigungskapazität und die kleinste unterscheidbare Spannung bestimmen, wie viele unterscheidbare Graustufen möglich sind. Aktuelle Sensoren benötigen einen A/D-Wandler von 12-14 Bit Auflösung (4096 - 16384 Graustufen). Der Mensch kann mit seinem Auge nur etwa 60 Graustufen unterscheiden.
- **Noise:** Bei jedem Umwandlungsschritt wird dem Signal ein Rauschen hinzugefügt, das nicht im ankommenden Photonenstrom enthalten war. Im Verhältnis zum inhärenten Photonenrauschen ist das Rauschen welches durch die Elektronik hinzugefügt relativ klein.

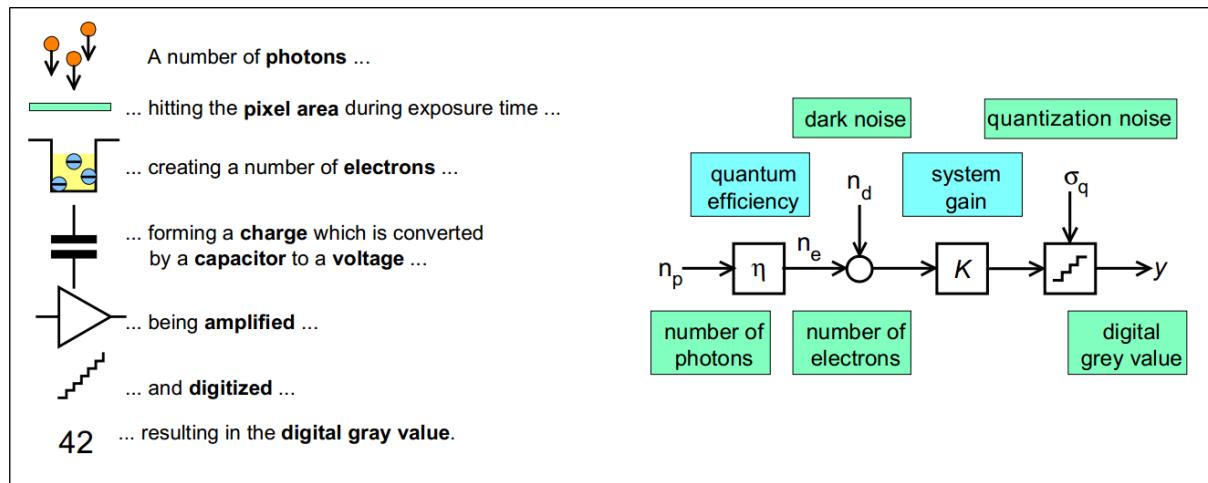


Abb. 43: Vereinfachtes physikalisches Modell zu Charakterisierung eines Bildsensors gemäss EMVA-Standard.

3.2.4 Rauschquellen

3.2.4.1 Defekte Pixel

Bei der Herstellung eines Sensors können defekte Pixel entstehen. Man unterscheidet folgende Pixeldefekte:

- **Tote Pixel** liefern kein oder ein sehr schwaches Signal, egal wie viel Licht einfällt.
- **Warne Pixel** integrieren zwar Licht auf. Es steht aber in keinem Zusammenhang mit der eingefallenen Lichtmenge.
- **Heisse Pixel** erreichen die Sättigung, auch wenn kein Licht einfällt.

3.2.4.2 Sensorrauschen (Noise)

- **Photon Noise (Photonenrauschen)**: Dieses Rauschen kommt vom Licht selbst. Der Photonenstrom, der auf den Sensor trifft, kommt nicht gleichmäßig, sondern mit einer [Poisson-Verteilung](#). Das Photonenrauschen hat den grössten Anteil am Gesamt- rauschen. Durch die natürliche Poisson-Verteilung bedingt ergibt sich ein Signal- Rausch-Verhältnis (SNR) von $\sqrt{N_{electrons}}$.
- **Dark Current Noise (Dunkelstrom)**: Dieses Rauschen entsteht z. B. durch thermische Prozesse im Sensor selbst und steigt mit der Belichtungszeit auch ohne Lichteinfall. Sehr empfindliche Sensoren, wie sie z. B. in der Astro-Fotographie eingesetzt werden, müssen deshalb gekühlt.
- **Readout Noise (Ausleserauschen)**: Beim Auslesen und Umwandeln der Photonen- ladung in Spannung entsteht sein Rauschen, weil nicht alle Zeilen beim CCD oder alle Pixel beim CMOS-Sensor gleichmäßig umwandeln.
- **Quantization Noise (Quantisierungsrauschen)**: Bei der Umwandlung der ana- logen Spannung in ein digitales Signal entsteht ein Rauschen.
- **Fixed Pattern Noise**: Dieses Rauschen entsteht, weil die Pixel nicht genau gleich auf den Photoneneinfall reagieren und die Ladung unterschiedlich aufsummieren.
- **Gain Noise (Verstärkerrauschen)**: Rauschen, dass durch die unterschiedliche Signalverstärkung entsteht.

Bei modernen Sensoren ist das Rauschen, das durch den Sensor erzeugt wird, mittler- weile sehr gering. Mehr zu Informationen zu Eigenschaften von aktuellen DSLR-Sensoren finden Sie bei [ClarkVision](#).

3.2.4.3 Blooming

Ist die Sättigungsgrenze (*Full Well Capacity*) eines Pixels erreicht, wird das Element überfüllt und die Ladung schwappt auf benachbarte Pixel des Sensors über. Diese können dadurch ebenfalls überfüllt werden, wodurch die Ladung wiederum auf deren Nachbarn überfliesst, usw. Dieser unerwünschte Effekt wird *Blooming* genannt. Ein Grossteil der modernen Sensoren ist bereits hardwareseitig mit sogenannten *Anti-Blooming-Gates* ausgestattet. Diese sind zwischen den einzelnen Pixelreihen angeordnete Gräben, die überschwappende Ladung aufnehmen und so die Weitergabe auf benachbarte Pixel verhindern.

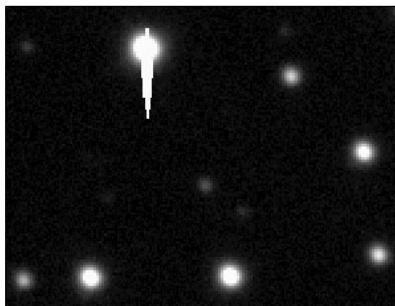


Abb. 44: Sternenbild mit Blooming

3.2.5 Dunkelbild- und Weissbildkorrektur

Zur Beseitigung des Dunkelstromrauschen kann bei Langzeitaufnahmen (z. B. in der Astrofotografie) ein Dunkelbild abgezogen werden (*Dark Field Correction*).

Helligkeitsunterschiede, die auf Verunreinigungen auf dem Sensor, durch *Fixed Pattern Noise* oder auf die verwendete Optik (z. B. Vignettierung) zurückzuführen sind, können ausgleichen werden. Dazu wird das aufgenommene Bild durch ein Weissbild dividiert (*Flat Field Correction*) und mit dem Mittelwert des Weissbilds multipliziert.

Im nachfolgenden Beispiel wird eine astronomische Aufnahme durch Dunkelbild- und Weissbildkorrektur entrauscht:

- Das **Dunkelbild** wurde bei gleicher Zeitdauer und Betriebstemperatur wie das Rohbild, allerdings ohne Licht aufgenommen. Es erfasst somit den während der entsprechenden Zeit aufgelaufenen Dunkelstrom.
- Das **Weissbild** wurde aufgenommen, während das Instrument auf eine gleichmässig erhelle Fläche ausgerichtet war. Es erfasst Unregelmäßigkeiten in der Ausleuchtung des Bildes (zum Beispiel durch Staub) und in der Empfindlichkeit der einzelnen Pixel.

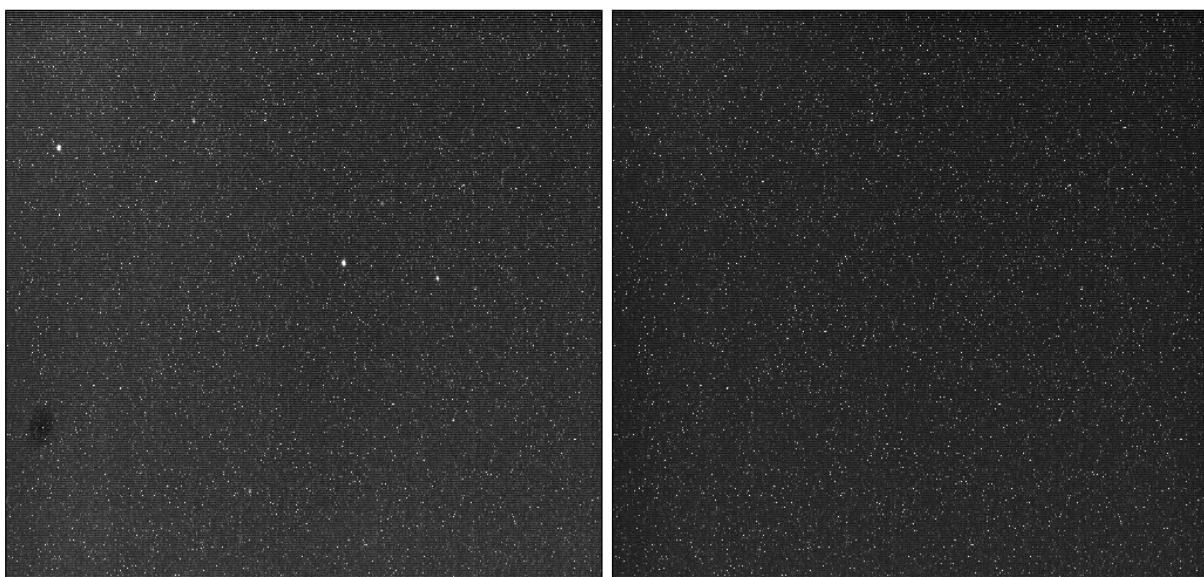


Abb. 45: Links: Unbrauchbares Rohbild, wo die Sterne kaum vom Rauschen zu unterscheiden sind. Rechts: Dunkelbild bei gleicher Belichtungszeit und Sensortemperatur aber geschlossenem Objektiv.

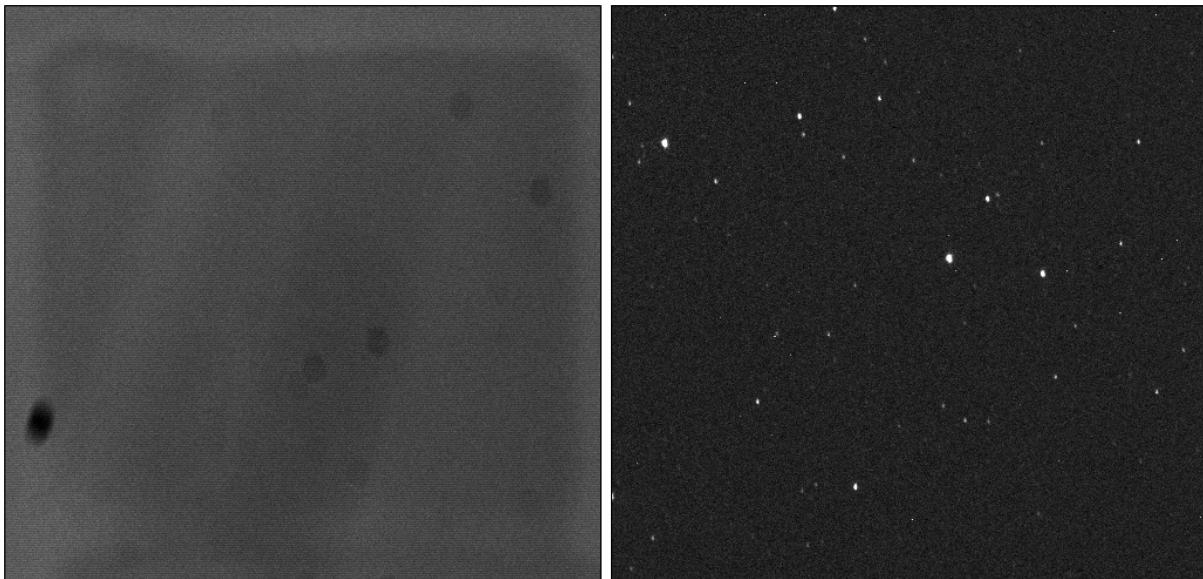


Abb. 46: Links: Weissbild zeigt eine gleichmässig weisse Fläche. Dadurch werden Staub und unterschiedlich empfindliche Pixel sichtbar. Rechts: Resultat nach der Weissbild- und Dunkelbildkorrektur. Quelle: ([Wikipedia](#))

3.2.6 Signal to Noise Ratio (SNR)

Das Signal-Rausch-Verhältnis (auch Signal-Rausch-Abstand) bezeichnet das Verhältnis zwischen einem Nutzsignal zu dem darüber gelagerten Rauschsignal. Für einen Sensor ist das SNR ein Qualitätsmaß für die Signalverarbeitung über alle Verarbeitungsschritte im Sensor. Je grösser dieses Mass, umso besser der Sensor.

$$SNR = \frac{Signal}{Noise} = \frac{electrons_{signal}}{electrons_{noise}}$$

In einem idealisierten Sensor kommt zum inhärenten Photonenrauschen kein zusätzliches Rauschen mehr hinzu. Ist der Sensor sauber (kein Staub), kalt (kein thermisches Rauschen) und das Signal nur minimal verstärkt, so bleibt alleine das Photonenrauschen übrig, welches sich aufgrund der [Poisson-Verteilung](#) wie folgt berechnen lässt:

$$SNR = \sqrt{N_{electrons}}$$

Das bedeutet, dass der Anteil des Photonenrauschens mit abnehmender Lichtmenge stark zunimmt. Dieses Rauschen wird meist nochmals durch das Verstärkerrauschen intensiviert, das ja genau dann zum Einsatz kommt.

Photons	SNR	Noise
9	3	33%
100	10	10%
900	30	3%
10000	100	1%
90000	300	0.3%

Weil die Signalleistung meist um mehrere Größenordnungen grösser ist als die Rauschleistung, wird das Signal-Rausch-Verhältnis oft im logarithmischen Massstab dargestellt. Man benutzt dazu die Pseudoeinheit [Dezibel \(dB\)](#):

$$SNR = 20 \cdot \log \left(\frac{Signal}{Noise} \right) \text{ [dB]}$$

Peak Signal to Noise Ratio

Wenn wir zwei Bilder A und B haben, von denen A das fehlerfreie Signal enthält und das Bild B denselben Inhalt jedoch mit Rauschen überlagert enthält, so können wir das PSNR-Mass als Qualitätsmaß heranziehen.

Das **PSNR-Mass** (*Peak Signal to Noise Ratio*) ist eine vom Signal-Rausch-Verhältnis abgeleitete Grösse, die in der Signalechnik verwendet wird, um den Übertragungsfehler einer Leitung zu messen. Zuerst wird das Rauschen als mittlerer quadratischer Fehler (= Summe der quadrierten Pixeldifferenzen) berechnet:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{mn} \sum_{x=1}^m \sum_{y=1}^n (A(x,y) - B(x,y))^2}$$

Das PSNR-Mass ist dann der zwanzigfache 10er-Logarithmus vom Verhältnis des maximalen Pixelwertes 255 (bei 8 Bit) zum **RMSE** (*Root Mean Squared Error*):

$$PSNR = 10 \cdot \log_{10} \left(\frac{255^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{255}{RMSE} \right) \quad [dB]$$

Um ein Gefühl zu bekommen für mögliche Werte beim PSNR hilft die nachfolgende Tabelle. Mit jeder 10er-Potenz, die der Fehler abnimmt, nimmt das PSNR-Mass um 20 dB zu:

100.000% Fehler > PSNR = $20 \cdot \log(255 / 255.00000)$	= 0 dB
10.000% Fehler > PSNR = $20 \cdot \log(255 / 25.50000)$	= 20 dB
1.000% Fehler > PSNR = $20 \cdot \log(255 / 2.55000)$	= 40 dB
0.100% Fehler > PSNR = $20 \cdot \log(255 / 0.25500)$	= 60 dB
0.010% Fehler > PSNR = $20 \cdot \log(255 / 0.02550)$	= 80 dB
0.001% Fehler > PSNR = $20 \cdot \log(255 / 0.00255)$	= 100 dB

3.2.7 CCD-Bildsensor

CCD steht für *Charge Coupled Device*, was so viel bedeutet wie ladungsgekoppelte Verschiebeeinheit. Ein CCD funktioniert wie eine Eimerkette. Anstatt eines Eimers mit Wasser reicht eine CCD-Zelle die in ihr gespeicherte elektrische Ladung an die nächste Zelle weiter und wird selber mit der Ladung aus ihrem zweiten Nachbarn aufgefüllt.

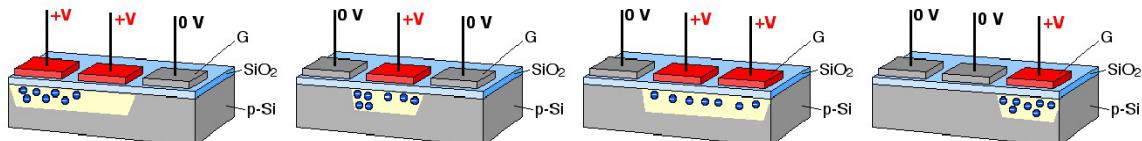


Abb. 47: Die Ladungspotenziale verschieben sich beim Auslesen in einer Zeile von einem Potenzialtopf zum nächsten. (Quelle: Wikipedia)

Somit entspricht das CCD einem analogen Schieberegister, bei dem der „Inhalt“ einer Speicherzelle in die benachbarte Zelle verschoben wird. Ein Auslesen eines einzelnen Pixels ist nicht möglich. Von der Speicherzone werden die Ladungen **zeilenweise, sequenziell** an den Verstärker weitergeleitet, der daraus das analoge Videosignal erzeugt.

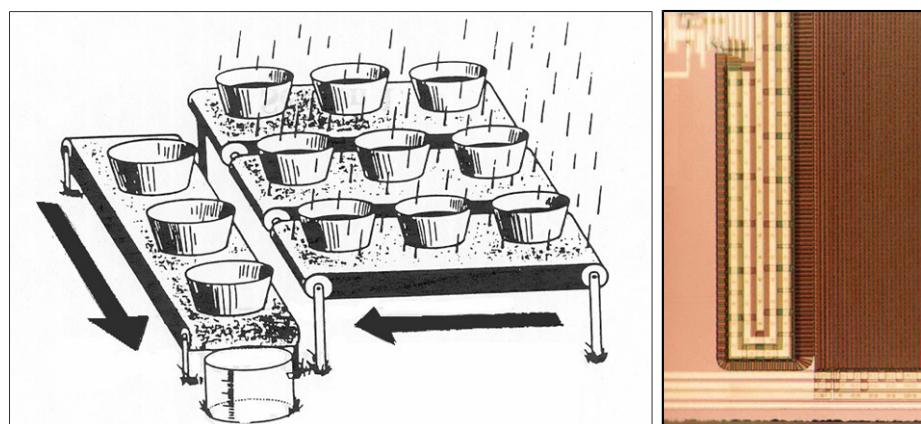


Abb. 48: Links eine bildliche Darstellung der Eimerketten und rechts die entsprechende Stelle auf einem CCD.

Interlaced oder Progressive Scan

Der *Interlaced CCD-Chip* nimmt nacheinander je ein Halbbild auf und kommt hauptsächlich in Videokameras mit DV-Format zum Einsatz. Beim Aufnehmen schneller Bewegungen kann daher eine sägezahnartige Ausfransung auftreten, da die vom Sensor kurz nacheinander gelesenen Bilddaten der zwei getrennt belichteten Halbbilder nicht mehr identisch sind. Der Interlaced Modus entstand in der Fernsehtechnologie, um ein stabileres Bild zu erzeugen (2×25 Hz).

Beim *Progressive-Scan-CCD-Chip* wird das ganze Bild auf einmal ausgelesen, da jede Zeile eine eigene Verschiebeeinheit besitzt. Damit fällt mit einer Aufnahme die doppelte Datenmenge an. Es braucht deshalb mehr Speicher und eine grössere Bandbreite bei der Übertragung.

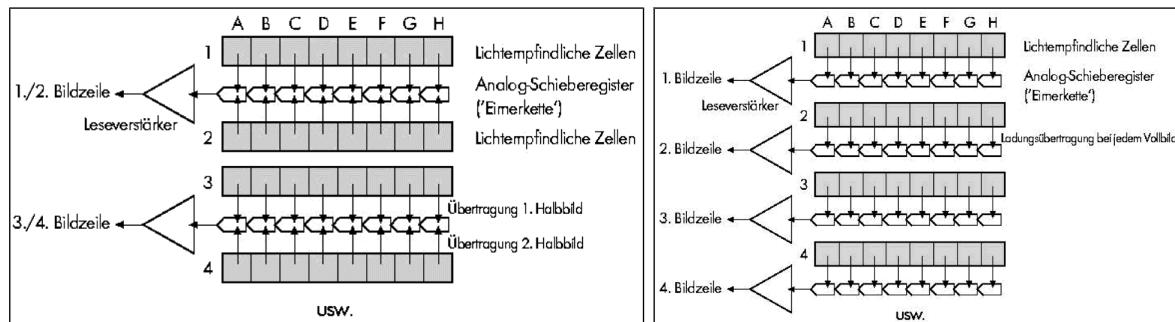


Abb. 49: Verfahren beim interlaced CCD-Chip. (Quelle: c't)



Abb. 50: Sägezahnartefakte zwischen den geraden und ungeraden Zeilen.

Vorteile von CCD-Sensoren

- Hohe Empfindlichkeit:** Weil sich ausser Töpfen kaum etwas anderes auf dem Sensor befindet, können diese grösstmöglich sein, was wiederum zu einer hohen Sättigungsgrenze führt.
- Geringes Rauschen:** Grosse Pixel führen zu einem geringeren Rauschanteil.

Nachteile von CCD-Sensoren

- Höherer Integrationsaufwand:** Am Ausgang besteht erst ein analoges Signal, welches durch zusätzliche Komponenten (Verstärker, A/D-Wandler etc.) verarbeitet werden muss.
- Geringere Auslesegeschwindigkeit:** Dies trifft nur bei Fullframe-CCDs zu. Mehr Informationen über verschiedene CCD-Typen finden Sie bei [Wikipedia](#).
- Höherer Stromverbrauch.**
- Keine X/Y-Adressierung** und damit nur Voll- oder Halbbildzugriff.

3.2.8 CMOS-Bildsensor

In **CMOS**-Bildsensoren (*Complementary Metal Oxide Semiconductor*) ist jede Fotozelle mit einem eigenen Verstärker ausgestattet. Die Auswertung der während des Belichtungsvorgangs entstandenen Ladungspotenziale geschieht durch eine Adressierung, die mit dem Ansprechen von RAM-Adressen vergleichbar ist.

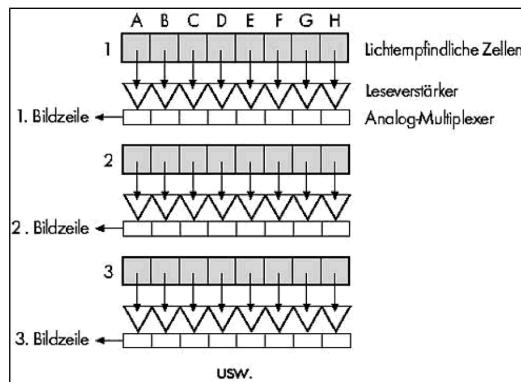


Abb. 51: X/Y-Adressierbarkeit beim CMOS Sensor (Quelle: c't)

Rolling Shutter und Global Shutter bei CMOS-Sensoren

- Beim **Rolling Shutter** (rollender Verschluss) werden die Pixel zeilenweise belichtet und weitergeleitet. Es sind damit sehr kurze Belichtungszeiten mögliche. Sensoren mit nur dieser Belichtungsmöglichkeit können einen Transistor pro Pixel sparen und sind deshalb günstiger. Bei bewegten Objekten entstehen durch die zeilenweise Belichtung Verzerrungen. Eine Beleuchtung durch Blitzen ist nicht möglich.
- Beim **Global Shutter** werden alle Pixel zusammen belichtet.



Abb. 52: Fahrender Bus mit rolling Shutter aufgenommen (Quelle: Stemmer)

Vorteile von CMOS-Sensoren

- **Schnell:** Weil die Messelektronik in jedem Pixel integriert ist, können CMOS-Sensoren schneller arbeiten. Bei einer 1.3 MP (1280x1024) Auflösung können bis zu 500 FPS erreicht werden.
- **Bildausschnitte:** Durch die Matrixadressierung können nur interessante Ausschnitte (*Region of Interest = ROI*) ausgelesen werden. Die Bildrate ist dabei umgekehrt proportional zur Ausschnittshöhe.
- **Tiefere Produktionskosten:** Da mit klassischer Waver-Fertigung produziert werden kann, sind die Produktionskosten tiefer als bei CCDs.
- **Geringerer Stromverbrauch:** 50% weniger als CCDs.
- **All in One Cameras:** Integration von zusätzlicher Verarbeitung (z. B. DAC).
- **Hohe Auflösung:** Da auf Wavern produziert wird, sind höhere Auflösungen möglich.

Nachteile von CMOS-Sensoren

- **Geringere Empfindlichkeit:** Die Messelektronik bei jedem Pixel braucht Platz. Daher sind die eigentlichen Messtöpfe kleiner, was wiederum in einem kleineren Dynamikumfang resultiert.

- Rauschanfälliger:** Kleinere Pixel = geringere Dynamik = grössere Verstärkung = mehr Photonens- und Verstärkerrauschen.

3.2.9 Farbaufnahmeprinzipien

Man kennt 4 Verfahren, um farbige Digitalbilder aufzuzeichnen:

- Farbaufnahme mit drei s/w CCD Chips parallel
- Farbaufnahme mit einem s/w CCD seriell
- Farbaufnahme mit einem Mosaik-Farb-CCD Chip
- Farbaufnahme mit Multilayer-Chip

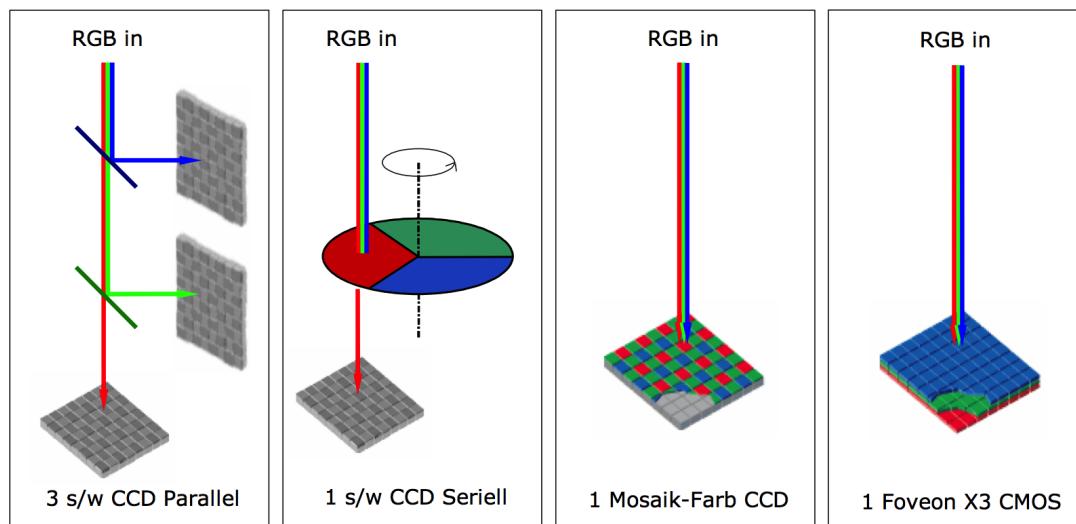


Abb. 53: Farbaufnahmeprinzipien

3.2.9.1 3-CCD-Verfahren

Um sämtliche für das Auge relevanten Farbinformationen eines Bildes zu erhalten, genügt es, eine rot-, grün- und blau-gefilterte S/W-Aufnahme anzufertigen. Zur Wiedergabe brauchte man demzufolge auch drei synchron laufende Projektoren, wiederum mit einem entsprechenden Farbfilter vor der Linse.

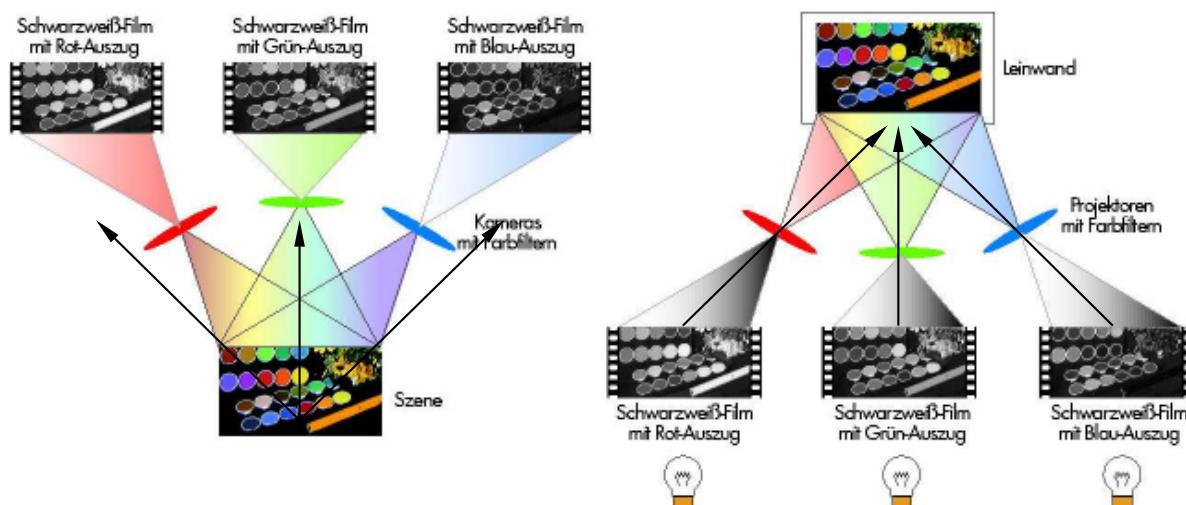


Abb. 54: Farbbild aus 3 gefilterten S/W-Bildern (links) und Farbprojektion mit 3 gefilterten S/W-Bildern (rechts)

Wir benötigen also drei CCD-Chips mit jeweils einer Farbfilterscheibe in einer der Grundfarben davor. Jeder der drei ausschliesslich helligkeitsempfindlichen CCD-Sensoren erhält

über ein System von halb durchlässigen Spiegeln das gleiche Bild der aufgenommenen Szene, allerdings jeweils durch einen eigenen Farbfilter.

Beschränkt man sich auf die Fotografie von unbewegten Objekten (*Stills*), ist eine Vereinfachung der Kamera möglich: Man macht mit einem einzigen s/w-CCD drei Aufnahmen hintereinander, jeweils durch eine rote, grüne und blaue Filterscheibe gesehen, so spart man sich die aufwendige Spiegeloptik und zwei der kostspieligen CCD-Sensoren.

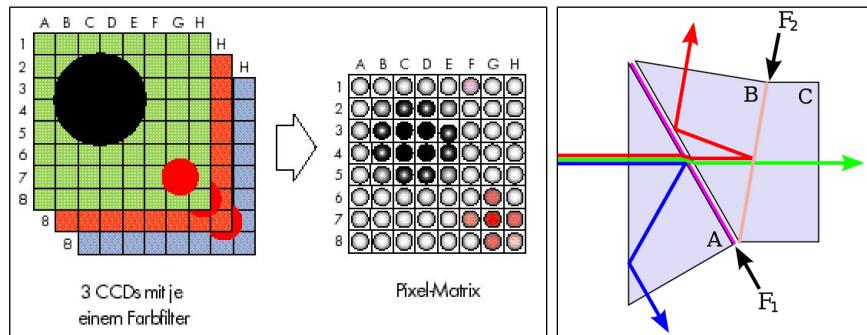


Abb. 55: Links: 3-CCD-Aufnahmen haben die volle Farbinformation auf jedem Pixel (Quelle: c't). Rechts: Prisma zum Filtern der RGB-Farben (Quelle: Wikipedia)

3.2.9.2 Bayermusterfilter

Die meisten Kameras arbeiten aber nur mit einem Sensor und unterschiedlichen Farbfiltern vor den einzelnen Zellen. Das RGBG-Muster wurde von Bruce Bayer von der Firma Eastman Kodak entwickelt und patentiert.

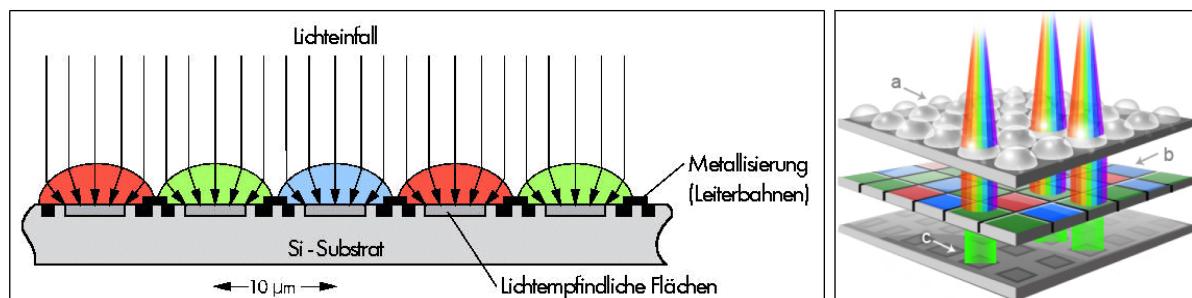


Abb. 56: Farbfiltermaske eines Farb-CCD (Quelle: c't), rechts: Bündelung und Filterung über den lichtempfindlichen Pixeln (Quelle: digital.pho.to).

Die matrixartig angeordneten, quadratischen Zellen sind so verteilt, dass an einem Kreuzungspunkt ihrer Grenzen immer drei Farben aneinanderstoßen. Wegen der erhöhten Empfindlichkeit des Auges im Spektralbereich um 555 nm wählt man als doppelte Farbe Grün.

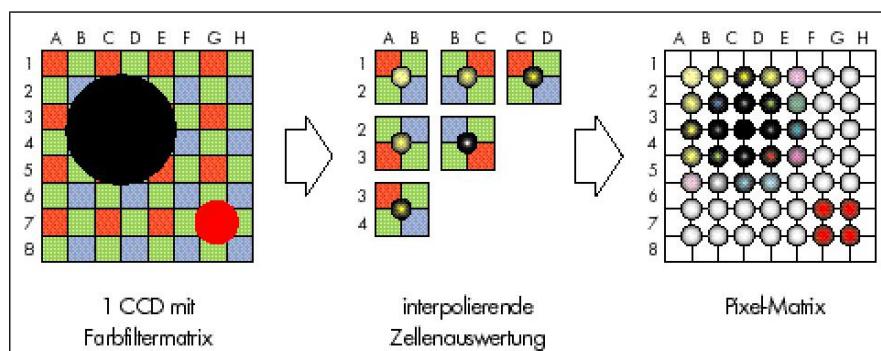


Abb. 57: Aufnahmeprinzip eines Sensors mit Bayer Pattern. Objekte, wie der rote Punkt, die kleiner sind als ein Viererquadrat, werden vom Farb-CCD nur korrekt erfasst, wenn sie auf dem entsprechenden Farbpixel liegen. Ein weiteres Problem sind die Farbverschiebungen an den Objektkanten, bedingt durch die teilweise Abdeckung der angeschnittenen Viererquadrate. (Quelle: c't)

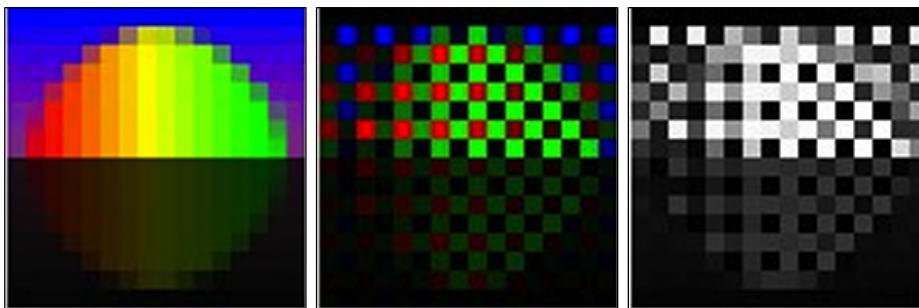
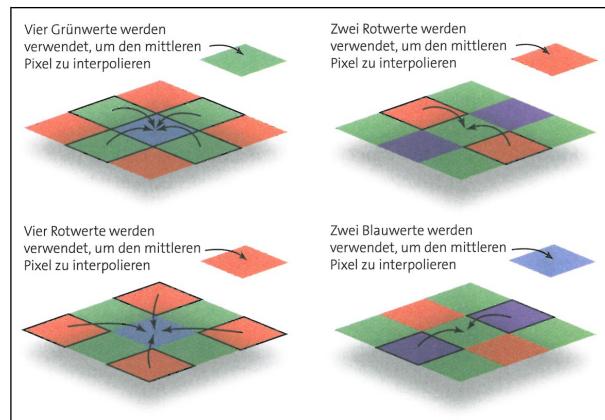


Abb. 58: Farbbild links, das entsprechende Bayermuster in der Mitte und die eigentlichen Pixelintensitäten.

Demosaicing, Debayering

Der Interpolationsprozess um vom Bayermosaik die RGB-Information auf jedem Pixel zu erhalten wird im Englischen *Demosaicing* oder *Debayering* genannt. Im einfachsten Fall, wie im Bild rechts dargestellt, wird bei jedem Pixel die fehlenden Farbinformationen aus den benachbarten Pixeln bilinear interpoliert. Bei kontrastreichen Kanten kommt es dabei zu Interpolationsfehlern, die als Farbrauschen sichtbar werden. Mit aufwendigeren Debayering Algorithmen lässt sich das Rauschen reduzieren.



Einen guten Überblick über einfache bis sehr aufwendige Debayering-Algorithmen finden Sie in [Nischwitz11].

Standardmäßig liefern die meisten Kameras RGB-Bilder, die meist mit dem JPEG-Format komprimiert sind. Wer mit dem Debayering und/oder der JPEG-Kompression nicht zufrieden ist, kann in vielen Kameras die Bilder im sogenannten RAW-Format abspeichern. Das ist ein herstellerabhängiges Rohformat, indem die ursprünglichen Sensordaten abgespeichert sind. Mit RAW-Konverter-Programmen wie *Adobe Lightroom* oder *Apple Aperture* kann eine aufwendige Interpolation erst auf dem Computer gerechnet werden.

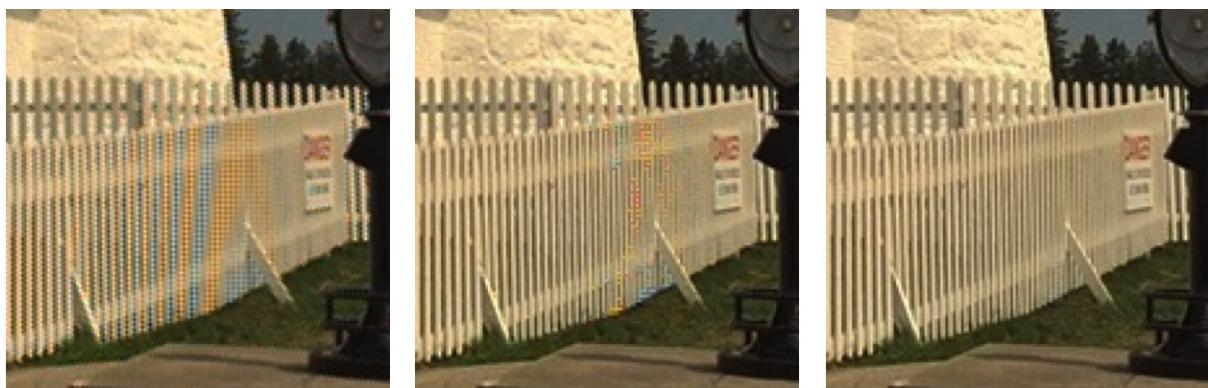


Abb. 59: Beispiel einer einfachen, einer mittleren und einer sehr aufwendigen Debayering-Interpolation.

3.2.9.3 Multilayer-Farbsensor

Die Firma [Foveon](#) hat 2002 einen neuartigen CMOS Farbbildsensor vorgestellt. Das Grundprinzip basiert darauf, dass Photonen je nach Wellenlängen unterschiedlich tief in das Silizium eindringen. Drei übereinander liegende Ebenen werden dabei für die Absorption der Grundfarben eingesetzt. Damit ist es möglich in einem Pixel zur gleichen Zeit, die rote, grüne und blaue Farbkomponente aufzuzeichnen. Dies erhöht nicht nur die

Auflösung gegenüber dem mosaikartigen Farb-CCD, sondern vermeidet auch jegliche Interpolationsartefakte.

Nach demselben Prinzip funktionierte auch der 3-Schicht-Farbfilm, der von der Firma Kodak entwickelt wurde.

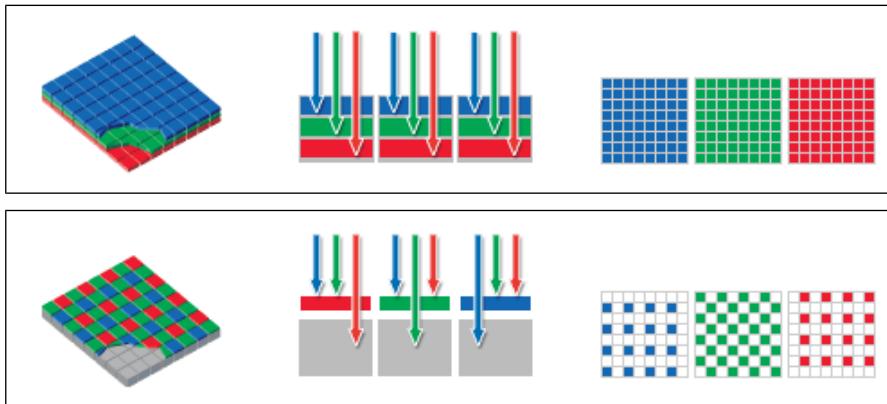


Abb. 60: Vergleich des X3-Prinzips (oben) und des Farb-CCDs (unten) (Quelle: Foveon)

3.3 Kameraschnittstellen

Nachfolgend sehen Sie eine Liste der aktuellen Kameraschnittstellen:

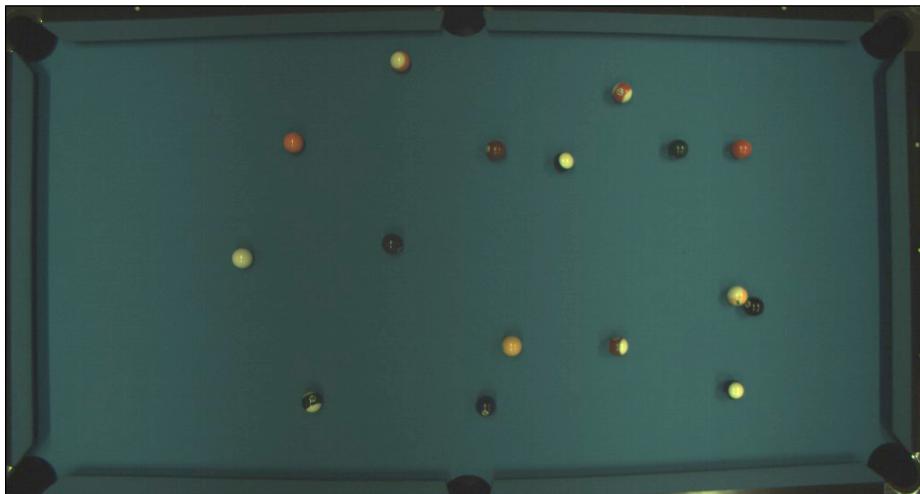
Schnittstelle	MByte/s	max. Kabel-länge	Frame-grabber	CPU-Ge-bräuch	Vor- und Nachteile
USB 2.0	30	5m	Nein	hoch	Sehr verbreitet, mittelschnell, nur kurze Kabel. Wird abnehmen.
Firewire IEEE 1394a	50	15m	Nein	gering	Sehr verbreitet, mittelschnell, benötigt meistens den grossen, stromführenden Firewire-Anschluss
Firewire IEEE 1394b	80	15m 100m bei 100 MBit	Nein	gering	Wenig verbreitet, schnell, benötigt meistens den grossen, stromführenden Firewire-Anschluss
GigE Vision Gigabit Ethernet	100	100m	Nein	mittel	In industrieller BV rel. verbreitet, schnell, mehrere Kameras möglich.
USB 3.0	400	5m	Nein	gering	Neuer Standard seit 2011, noch wenig verbreitet.
Camera Link (CL)	680	10m	Ja	mittel	In industrieller BV wenig verbreitet, schnell, braucht spez. Frame Grabber, mehrere Kameras möglich.
CoaXPress	6250 3125	40m 100m	Ja	?	Neuer Standard seit 2010, noch wenig verbreitet.



3.4 Übung zum Kapitel Bildaufnahme mit ImageJ

Mit einer Kamera soll ein Billardtisch überwacht werden:

- Die Masse des Tisches sind 2.54m x 1.27m.
- Die Kamera wird über dem Zentrum des Tisches in 1.27m angebracht.
- Die Kamera besitzt einen Bayermuster-Farbsensor mit einer Auflösung von 2048 x 1088 Pixeln. Gemäss Datenblatt ist ein Pixel $5.5\mu\text{m} \times 5.5\mu\text{m}$.
- Die Kamera liefert die Bilder unkomprimiert im RAW-Format mit 8 Bit pro Pixel.
- In Farbe sieht eine Aufnahme wie folgt aus:



1. Welche Brennweite muss das Objektiv haben?
2. Importieren Sie das RAW-Bild [Billard2048x1088x1.raw](#) mit ImageJ über das Menü File > Import Raw.
3. Speichern Sie das Bild im PNG-Format mit dem Namen [Billard2048x1088x1.png](#).
4. Schreiben Sie einen DeBayering-Filter, der ein RGB-Bild in halber Auflösung (1024 x 544) erzeugt. Fassen Sie dabei jeweils 4 Pixel des RAW-Bildes zu einem RGB-Pixel zusammen. Speichern Sie das RGB-Bild als [Billard1024x544x3.png](#).
5. Um die Billardkugeln im Bild zu finden, verwenden wir am Besten den Farbton. Dadurch sollten wir jegliche Schattierungen auf dem Tisch und an den Kugeln verlieren. Konvertieren Sie die RGB-Werte mit [Color.RGBtoHSB](#). Speichern Sie sowohl den Farbton (H-Wert) und den Helligkeitswert (B-Wert) als Graustufen Bild ab.

Die Funktionalität, um Bilder mit ImageJ zu erstellen und abzuspeichern können Sie aus dem nachfolgenden Code entnehmen:

```

import java.awt.Color;
import ij.ImagePlus;
import ij.gui.NewImage;
import ij.plugin.PNG_Writer;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;

public class Billard_Tracker implements PlugInFilter
{
    @Override
    public int setup(String arg, ImagePlus imp)
    {
        return DOES_8G;
    }

    @Override
    public void run(ImageProcessor ip1)
    {
        int w1 = ip1.getWidth();
        int h1 = ip1.getHeight();
    }
}

```

```
byte[] pix1 = (byte[]) ip1.getPixels();

ImagePlus imgGray = NewImage.createByteImage("GrayDeBayered", w1/2, h1/2,
                                             1, NewImage.FILL_BLACK);
ImageProcessor ipGray = imgGray.getProcessor();
byte[] pixGray = (byte[]) ipGray.getPixels();
int w2 = ipGray.getWidth();
int h2 = ipGray.getHeight();

ImagePlus imgRGB = NewImage.createRGBImage("RGBDeBayered", w1/2, h1/2,
                                             1, NewImage.FILL_BLACK);
ImageProcessor ipRGB = imgRGB.getProcessor();
int[] pixRGB = (int[]) ipRGB.getPixels();

long msStart = System.currentTimeMillis();

ImagePlus imgHue = NewImage.createByteImage("Hue", w1/2, h1/2,
                                             1, NewImage.FILL_BLACK);
ImageProcessor ipHue = imgHue.getProcessor();
byte[] pixHue = (byte[]) ipHue.getPixels();

int i1 = 0, i2 = 0;

for (int y=0; y < h2; y++)
{
    // ???
}

long ms = System.currentTimeMillis() - msStart;
System.out.println(ms);

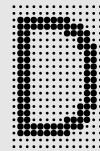
PNG_Writer png = new PNG_Writer();
try
{
    png.writeImage(imgRGB, "../../Images/Billard1024x544x3.png", 0);
    png.writeImage(imgHue, "../../Images/Billard1024x544x1H.png", 0);
    png.writeImage(imgGray, "../../Images/Billard1024x544x1B.png", 0);

} catch (Exception e)
{
    e.printStackTrace();
}

imgGray.show();imgGray.updateAndDraw();
imgRGB.show(); imgRGB.updateAndDraw();
imgHue.show(); imgHue.updateAndDraw();
}

public static void main(String[] args)
{
    Billard_Tracker plugin = new Billard_Tracker();
    ImagePlus im = new ImagePlus("../../Images/Billard2048x1088x1.png");
    im.show();
    plugin.setup("", im);
    plugin.run(im.getProcessor());
}
}
```

4 Statistische Bildauswertung



a die Bildaufnahme ein normaler Messvorgang ist, kann die Bildverarbeitung auch als Messdatenverarbeitung aufgefasst werden. Diese Sicht führt unmittelbar zu einer statistischen Beschreibung digitaler Rasterbilder und erlaubt, grundlegende Begriffe der Statistik für die Beurteilung digitaler Bilddaten zu nutzen. Im Unterschied zur Fotobearbeitung soll der Mensch in der Computer Vision möglichst wenig subjektiv eingreifen, sodass wir Bilder meistens einer statistischen Auswertung unterziehen, bevor wir weitere automatische Verarbeitungsschritte durchführen können.

4.1 Histogramm



Die geläufigste Statistik ist die diskrete Häufigkeitsverteilung der Grauwerte eines Bildes, das sogenannte *Histogramm*. Das Histogramm kann für Grauwerte, als auch für die Farbkomponenten Rot, Grün und Blau erstellt werden.

Für das 4-Bit-Graustufenbild nebenan werden alle Pixel von allen 16 Graustufen ausgezählt und in einem Balkendiagramm dargestellt.

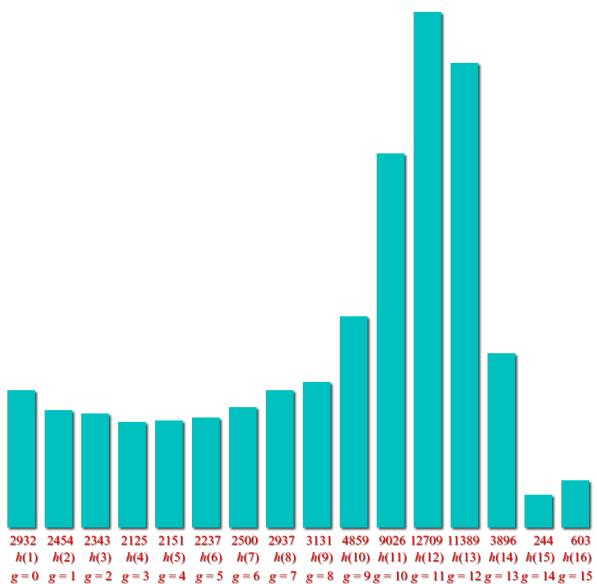
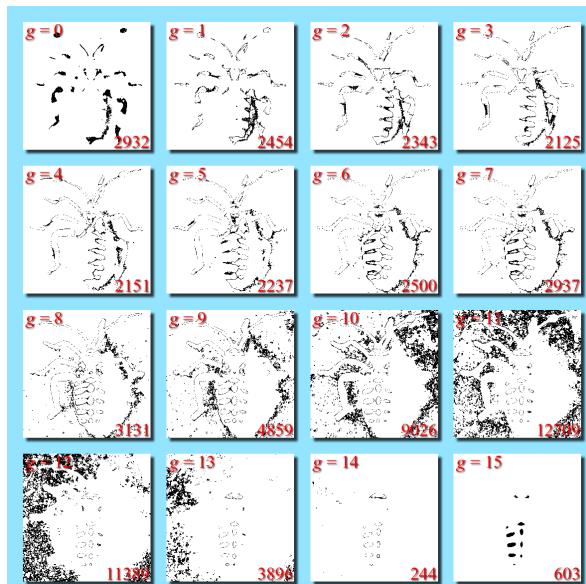


Abb. 61: Beispiel: Ein 4-Bit-Graustufenbild mit seinen 16 Graustufen einzeln dargestellt. Rechts das Balkendiagramm des Histogramms (Bildquelle [Peters07]).

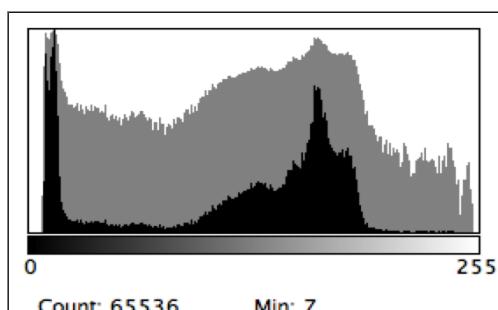


Abb. 62: Beispiel: Ein 8-Bit-Graustufenbild und sein Histogramm in ImageJ, bei dem für jede Graustufe zwischen 0 und 255 ein schwarzer Strich gezeichnet wird. Das hellgraue Histogramm im Hintergrund hat eine logarithmische Skalierung.

Das Grauerthistogramm stellt die Verteilung der Häufigkeiten $h(g)$ aller Grauwerte dar:

$$h(g) = n_g \quad \text{mit } g \text{ als Grauwerte (0-255) und } n_g \text{ der Anzahl Pixel mit dem Grauwert } g$$

Die Summe aller Häufigkeiten ist folglich die Summe aller Pixel N :

$$\sum_{g=0}^{255} h(g) = N$$

In **ImageJ** lässt sich das Histogramm über das Menü [Analyze > Histogram](#) erzeugen und anzeigen. In einem ImageJ-Filter lässt sich das Histogramm-Array mit der Funktion [getHistogramm](#) erzeugen:

```
public void run(ImageProcessor ip)
{
    float[] h = ip.getHistogram();
    ...
}
```

In **Matlab** lässt sich das Histogramm mit dem Befehl [imhist](#) erzeugen und anzeigen. Wer nur die Histogrammwerte braucht, kann sie mit dem Befehl [hist](#) erzeugen.

4.1.1 Wahrscheinlichkeitsdichtefunktion

Normiert man das Histogramm, indem alle Häufigkeiten durch die Summe aller Pixel dividiert werden, so erhält man die [Wahrscheinlichkeitsdichtefunktion](#) $p(g)$ (Engl. [probability density function \(pdf\)](#)). Die Summe aller Wahrscheinlichkeiten ist dann 1 (100%).

$p(g)$ besagt, wie wahrscheinlich ein Pixel den Grauwert g hat.

$$p(g) = \frac{n_g}{N}$$

$$\sum_{g=0}^{255} p(g) = 1$$

4.1.2 Kumulatives Histogramm

Das kumulative Histogramm $H(g)$ ist eine Variante des normalen Histogramms, bei dem zur Häufigkeit einer Graustufe zusätzlich alle tieferen Graustufen aufsummiert sind. Das kumulative Histogramm wird vor alle bei den automatischen Kontrastanpassungen eingesetzt.

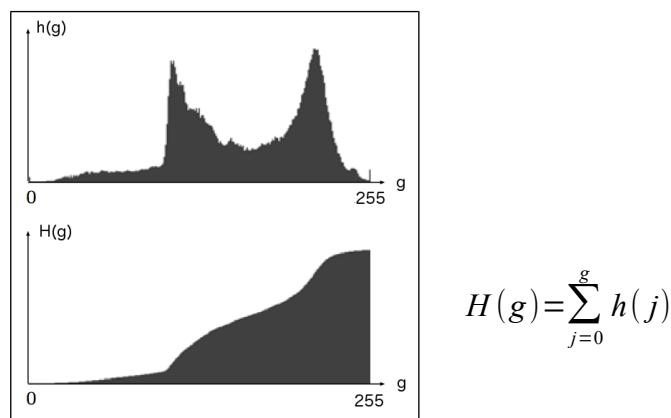


Abb. 63: Das normale und das kumulative Histogramm übereinander.

Analog zum Histogramm entspricht das normierte kumulative Histogramm der [kumulativen Wahrscheinlichkeitsdichtefunktion](#) $P(g)$ (Engl. [Cumulative Density Function \(CDF\)](#)).

$P(g)$ besagt, wie wahrscheinlich ein Pixel den Grauwert g oder kleiner hat.

$$P(g) = \frac{H(g)}{N}$$

4.1.3 Mehrdimensionales Histogramm

Die Erstellung eines Histogramms ist nicht auf ein Graustufenbild beschränkt. Das N-dimensionale Histogramm für ein N-kanaliges Bild ist wie folgt definiert:

$$h(g_0, h_1, \dots, h_{N-1}) = a_{g_0 g_1 \dots g_{N-1}}$$

worin $a_{g_0 g_1 \dots g_{N-1}}$ die Häufigkeit der Graustufenkombination mit dem Grauwert g_n im Bildkanal n ist. Ein 2D-Histogramm ist somit eine 3D-Balkendiagramm worin die Häufigkeiten in der Höhe der Balken dargestellt wird. Es lässt sich auch als Bild darstellen, wenn wir die Häufigkeit mit einer Konstante C skalieren:

$$s(x, y) = s(g_0, g_1) = C \cdot h(g_0, h_1)$$

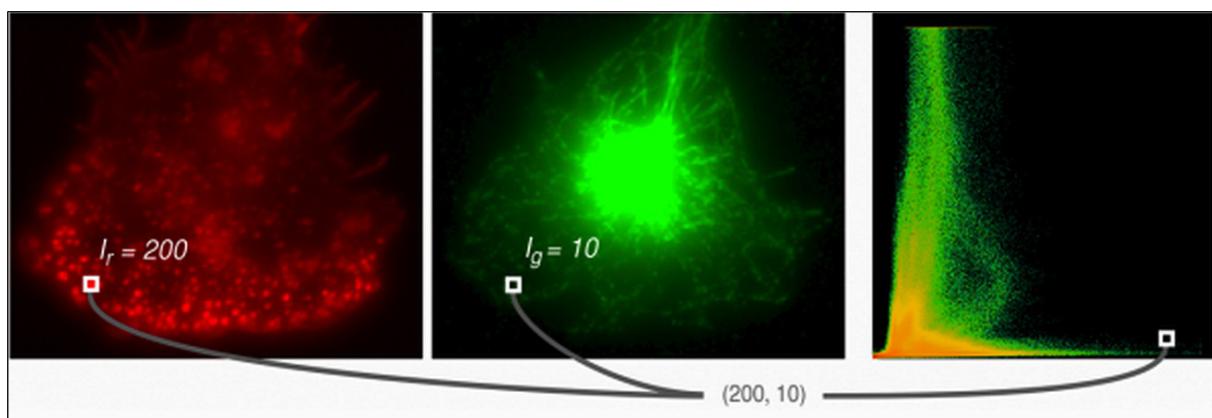


Abb. 64: 2D-Histogramm (rechts) vom Rotkanal und Grünkanal. An der markierten Stelle bei der Bildkoordinate [30,20] hat das Rotkanalbild links einen Wert 200 und das Grünkanalbild in der Mitte einen Wert von 10. Im 2D-Histogramm der Grösse 256 x 256 wird dadurch der Zähler bei [200,10] um eins erhöht. Das 2D-Histogramm wird mit einer Farbskala (die 3. Dimension) dargestellt, um feine Grauwertunterschiede besser ersichtlich zu machen. (Bildquelle: [Scientific Volume Imaging](#))

Aus einem 2D-Histogramm lassen sich einige Eigenschaften aus zwei gleich grossen Bildern ablesen:

- **Überblendung:** Werden bei einer Achse die Häufigkeiten abgeschnitten (im Histogramm in Abb. 64), so weisst dies auf eine Überblendung und Informationsverlust in diesem Kanal hin.
- **Korrelation:** Sind die meisten Werte um Diagonale von unten links nach oben rechts verteilt, so haben die beiden Kanäle eine hohe Korrelation oder Überlappung. Der quadratische Abstand zur Diagonalen kann als Mass für die Überlappung verwendet werden.
- **Verteilung des Dynamikumfangs:** Sind die Häufigkeiten entlang einer Achse verteilt, so hat dieser Kanal einen deutlich höheren Dynamikumfang.
- **Heisse Pixel:** Gibt es vereinzelte Häufigkeiten, so ist dies ein Hinweis auf einzelne defekte Pixel in beiden Kanälen.
- **Klassifikation:** Im Kapitel 11 über Klassifikation (S. 207) wird die sogenannte Cluster-Analyse in 2D- oder N-dimensionalen Histogrammen durchgeführt. Cluster sind Punktewolken im Histogramm, die für eine bestimmte Klasse von Objekten im Bild stehen. Die verschiedenen Methoden der Klassifikation versuchen jede Pixel einer Klasse zuzuweisen.

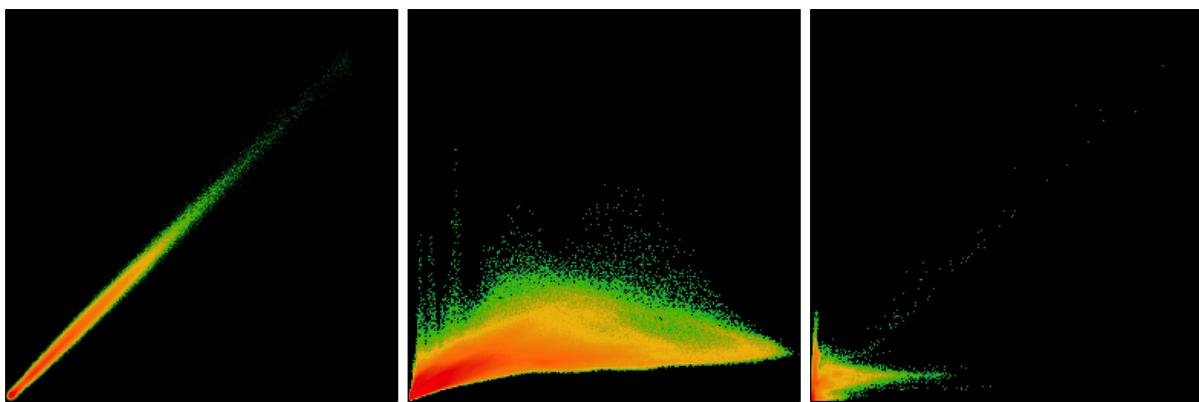


Abb. 65: 2D-Histogramm mit hoher Korrelation (links), einer einseitigen Dynamik in der Mitte und von einzelnen heißen Pixeln im rechten Histogramm. (Bildquelle: [Scientific Volume Imaging](#))

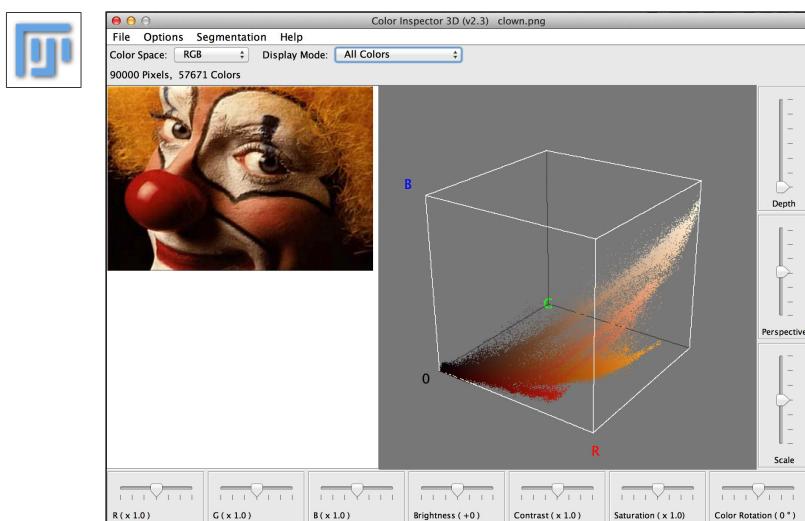


Abb. 66: *3D Color Inspector* in Fiji

In **Fiji** lassen sich 2D-Histogramme über das Plugin [*Plugins > Analyze > 2D Histogram*] erstellen.

Über das Plugin *Color Inspector 3D* lassen sich ebenfalls 3D-Histogramme dynamisch anzeigen. Sie können dabei Farbräume sowie Anzeigemodi auswählen.

4.2 Statistische Kennzahlen

Neben der Häufigkeitsverteilung können aber auch die klassischen, statistischen Kenngrößen zur Beurteilung der Bilddaten herangezogen werden:

4.2.1 Minimum und Maximum

g_{min} und g_{max} bezeichnen den minimalen resp. maximalen Grauwert in einem Bild. Sie sind wichtig die Beurteilung der Dynamik und des Kontrastes.

4.2.2 Dynamik

Als Dynamik kann man die Differenz zwischen dem hellsten und dunkelsten Grauwert verwendet werden.

$$D = g_{max} - g_{min}$$

Alternativ kann auch die Anzahl der verwendeten Graustufen (N_c) als Dynamikwert verwendet werden. Sie ist normalerweise $\leq D$. Eine hohe Dynamik ist meistens wünschenswert, da mit einer geringen Dynamik in nachfolgenden Verarbeitungsschritten ein Qualitäts- und Informationsverlust einhergeht.

4.2.3 Kontrast

Ein ähnlicher aber generellerer Wert ist der Kontrastwert:

$$C = \frac{g_{max} - g_{min}}{g_{max} + g_{min}}$$

4.2.4 Mittelwert (Mean)

Der arithmetische Mittelwert entspricht der mittleren Helligkeit eines Bildes:

$$\bar{g} = \frac{1}{N} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} g(x, y)$$

4.2.5 Modalwert (Mode)

Der Modalwert oder Modus ist der am häufigsten vorkommende Grauwert:

$$\dot{g} = \max(h(g))$$

4.2.6 Standardabweichung (Standard Deviation)

Die Standardabweichung entspricht der Wurzel aus der Varianz. Die Varianz entspricht der quadratischen Differenz aller Grauwerte zum Mittelwert und kann ebenfalls als Mass für den Kontrast verwendet werden.

$$s = \sqrt{Var}$$

$$var = \frac{1}{N} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} (g(x, y) - \bar{g})^2$$

4.2.7 Schiefe (Skewness)

Die Schiefe ist ein Mass für die Asymmetrie der Verteilung im Histogramm.

$$v = \frac{1}{N} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \left(\frac{g(x, y) - \bar{g}}{s} \right)^3$$

4.2.8 Wölbung (Kurtosis)

Die Wölbung ist ein Mass für die Abweichung der Verteilung von der Normalverteilung. Untersucht man die Oberfläche eines Werkstückes, die einfarbig und unstrukturiert sein soll, so machen sich Kratzer oder Verunreinigungen durch eine Verbreiterung und Asymmetrie der Verteilung bemerkbar, sodass Varianz, Schiefe und Wölbung als Indikatoren verwendet werden können.

$$w = \frac{1}{N} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \left(\frac{g(x, y) - \bar{g}}{s} \right)^4$$

4.2.9 Entropie (Entropy)

Die Entropie H ist ein Mass für den Informationsgehalt eines Bildes. Wir brauchen dieses Mass für die Bildkompression den es gibt an, wie viele Bits im Minimum notwendig sind, um das Bild zu speichern. Mehr dazu erfahren Sie im Kapitel 12.1.2.

$$H = - \sum_0^{255} h(g) \log_2(h(g))$$

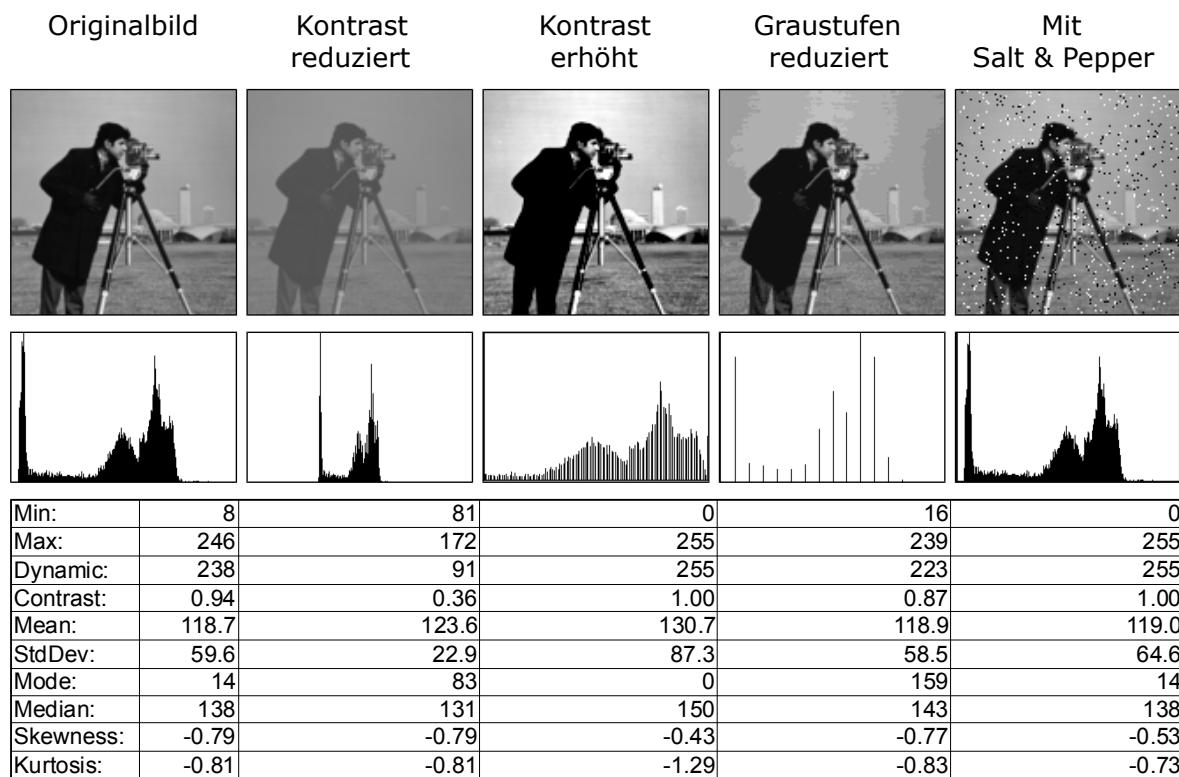
mit $h(g)$ aus dem Histogramm als Wahrscheinlichkeit einer Graustufe. Die Entropie H kann auch interpretiert werden als mittlere Anzahl Bits, die notwendig sind, um ein Bild ohne Informationsverlust zu speichern:

- Ein homogenes Bild mit einer einzigen Graustufe hat eine Entropie $H=0$.
- Ein Bild mit gleichmäßig verteilten Graustufen ($h(g)=1/256$) hat die Entropie $H=8$.

4.2.10 Vergleiche



Nachfolgend sehen Sie diese statistischen Kennzahlen von verschiedenen Varianten von einem Bild, die jeweils mit den ImageJ Befehl [Analyze > Measure] in Erfahrung gebracht wurden:



In Matlab stehen folgende Statistikfunktionen zur Verfügung:

```
Imin = min(I(:)); % Berechnet Minimum
Imax = max(I(:)); % Berechnet Maximum
Imea = mean(I(:)); % Berechnet Durchschnitt
Imed = median(I(:)); % Berechnet Median
Istd = std(I(:)); % Berechnet Standardabweichung
Ivar = var(I(:)); % Berechnet Varianz
Iskw = skewness(I(:)); % Berechnet Schiefe
Ikur = kurtosis(I(:)); % Berechnet Exzess
```

5 Punktoperatoren



Punktoperatoren ist die erste Gruppe von Grundoperationen, mit denen digitale Bilder verarbeitet werden. Die Bildverarbeitung und die nachfolgende Bildanalyse setzen sich meist aus einer Vielzahl von einzelnen Verarbeitungsschritten zusammen. Einzelne Grundoperationen, wie z. B. die Kontrastanpassung kennen Sie sicher auch aus der privaten Fotobearbeitung. Wir gehen in den nächsten 3 Kapiteln noch nicht auf übergeordnete Anwendungsziele ein, sondern beschränken uns auf die verschiedenen Grundoperationen. Wir gliedern diese Operationen aus technischer Sicht in punktbezogene (Kapitel 5), lokale (Kapitel 6) und globale Operatoren (Kapitel 7). Diese Kategorien unterscheiden damit die Menge der Pixel, die im Quellbild notwendig sind, um ein Pixel im Zielbild zu berechnen.

Wir unterscheiden die Operationen der Bildverarbeitung in folgende Gruppen:

- **Punkt Operatoren** sind Operationen, die jeweils nur auf dem Pixelwert selbst operieren. Sie brauchen und tangieren keine Pixel um das zu bearbeitende Pixel und sind deshalb meist einfach und sehr schnell. Die Operationen bedingen allerdings häufig eine vorgängige statistische Auswertung des Bildes, was wiederum auch nicht gratis ist. Zu den bekanntesten Punktoperatoren zählen:
 11. Binarisierung
 12. Grauwertreduktion
 13. Kontrast- und Helligkeitsmanipulation
 14. Histogramm-Ausgleich
 15. Arithmetische und logische Operationen
 16. Geometrische Transformationen (Skalierung, Rotation)
- **Lokale Operatoren** beziehen für die Berechnung eines Pixels im Zielbild eine Region aus dem Quellbild in die Operation ein:
 17. Tiefpassfilter (Mittelwert, Gauss)
 18. Hochpassfilter (Sobel, Laplace, Laplace of a Gaussian)
 19. Morphologische Operationen (Dilatation, Erosion, Opening, Closing)
 20. Rangordnungsfilter (Median, Min, Max)
- **Globale Operatoren** transformieren die gesamte Bildinformation in einen anderen Raum, in dem neue oder schnellere Möglichkeiten der Verarbeitung bestehen. Das gesamte Bild oder die gewonnenen Informationen müssten dann zurück transformiert werden.
 21. Fourier-Transformation
 22. Wavelet-Transformation
 23. Hough-Transformation
 24. Principal Component Analysis

5.1 Lineare und nichtlineare Abbildungsfunktionen

Das Histogramm bildet die Grundlage für viele Punktoperationen, die dessen Verteilung über lineare und nichtlineare Abbildungsfunktionen modifizieren:

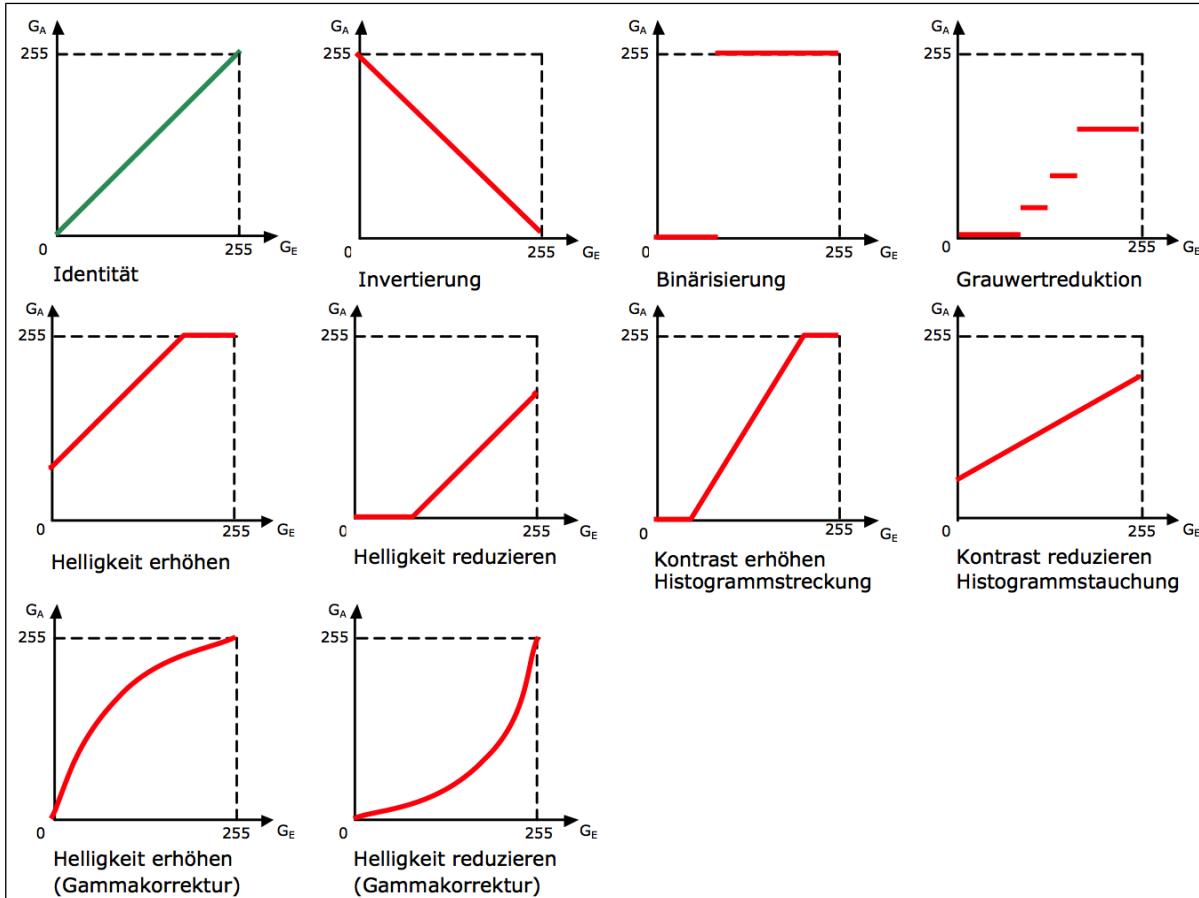


Abb. 67: Lineare und nichtlineare Abbildungsfunktionen: G_E : Eingangsgrauwert, G_A : Ausgangsgrauwert

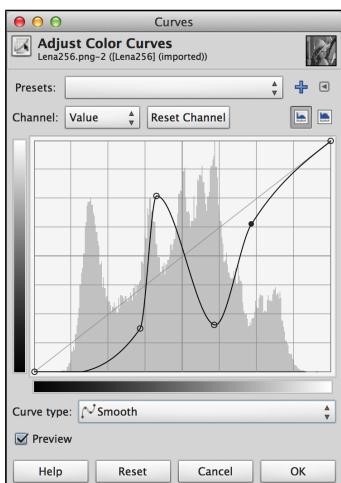


Abb. 68: Der Abbildungsfunktions-Editor von Gimp

Lookup-Tabellen

Generell lassen sich alle Punktoperationen, die mit einer Abbildungsfunktion formuliert werden können, mittels einer Lookup-Tabelle implementieren. Man wendet dazu die Abbildungsfunktion auf alle Graustufen an und speichert die Resultate in einer Tabelle.

Danach kann man den Output-Wert nur noch über den Input-Wert nachschlagen und braucht so pro Pixel keinerlei Berechnungen mehr durchzuführen:

Input	0	1	2	3	4	5	6	7	...	250	251	252	253	254	255
Output	255	254	253	252	251	250	249	248	...	5	4	3	2	1	0

Abb. 69: Lookup-Tabelle für eine Invertierung

5.2 Invertierung

Das Negativbild wird durch eine der einfachsten Punktoperationen erstellt, bei der die Grauwerte invertiert werden:

$$g_A(x, y) = 255 - g_E(x, y)$$



Abb. 70: Links: Abbildungsfunktion der Invertierung, Mitte und Rechts: Bild vor und nach der Invertierung.

5.3 Binarisierung

Bei der Binarisierung werden alle Grauwerte unterhalb einer geeigneten Schwelle t (Englisch: *Threshold*) zu Weiss und alle darüber liegenden Grauwerte zu Schwarz gemacht:

$$g_A(x, y) = \begin{cases} g_1: g_E(x, y) < t \\ g_2: g_E(x, y) \geq t \end{cases}$$

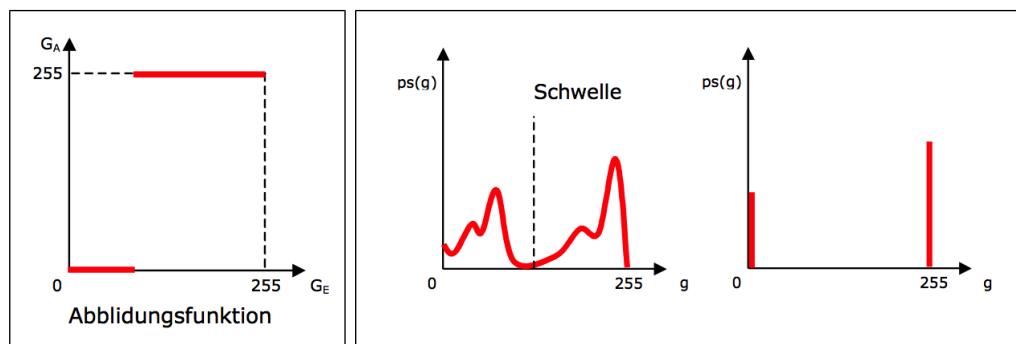


Abb. 71: Links: Abbildungsfunktion für eine Binarisierung. Rechts: Histogramm vor und nach der Binarisierung.

Die Schwierigkeit bei der Binarisierung liegt in der automatischen Festlegung des Schwellwerts. Für einige Applikationen kann es zwar möglich sein, den Schwellwert manuell mit einer Benutzerinteraktion oder sogar bei einem konstanten Wert festzulegen.



Abb. 72: Manuelle Binarisierung in ImageJ (Menü *Image > Adjust > Threshold*), wo der Schwellwert auch automatisch durch eine Vielzahl an Algorithmen festgelegt werden kann.

Meistens muss der Schwellwert aber durch Berechnung festgelegt werden. Diese *Bi-modalitätsprüfung* kann mehr oder weniger aufwendig sein und die Einfachheit der eigentlichen Punktoperation schnell zu Nichte machen. Im schlimmsten Fall gibt es keinen globalen Schwellwert, der für das ganze Bild funktioniert. In diesem Fall müssen lokale, an die Regionen des Bildes angepasste Schwellwerte bestimmt werden. Wir betrachten die verschiedenen Methoden, um den Schwellwert zu berechnen im Kapitel 8.1.

5.4 Grauwertreduktion

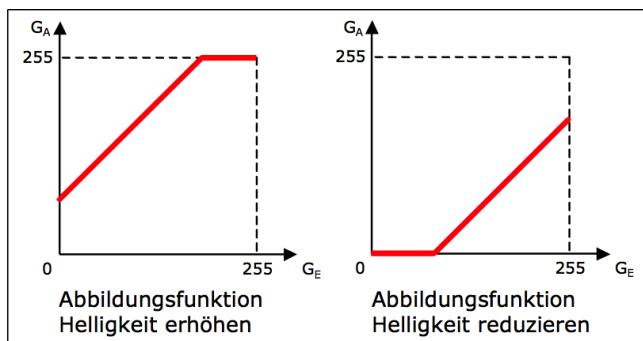
In Anlehnung an die Fotografie wird diese Technik auch als Äquidensiten-Technik (*gray level slicing*) bezeichnet. In der Farbbildverarbeitung wird diese Methode Plakatierung (*Posterizing*) genannt. Auch die Grauwertreduktion wird mittels einer *lookup table* implementiert. Die Erstellung der *lookup table* ist allerdings keine triviale Sache, wenn man sie aufgrund des analysierten Bildes erstellt und nicht statisch vorgibt. Der bekannteste Algorithmus hierzu ist der [Median Cut Algorithmus von Paul Heckbert](#), der auch für die Farbreduktion von Farbbildern eingesetzt wird. Die Grauwertreduktion wird u.a. zur Reduktion der Bittiefe für bestimmte Speicherformate (z. B. GIF, PNG und BMP) verwendet.



Abb. 73: Links: Abbildungsfunktion einer Grauwertreduktion. Mitte und Rechts: Bild & Histogramm nach der Grauwertreduktion.

5.5 Lineare Helligkeitskorrektur

Die lineare Helligkeitskorrektur bewirkt eine Verschiebung der Häufigkeitsverteilung im Histogramm und entspricht der Konstanten b in der linearen Gleichung.



$$g_A(x, y) = c \cdot g_E(x, y) + b$$

Abb. 74: Abbildungsfunktionen für eine lineare Aufhellung (links) und eine lineare Abdunklung (rechts).

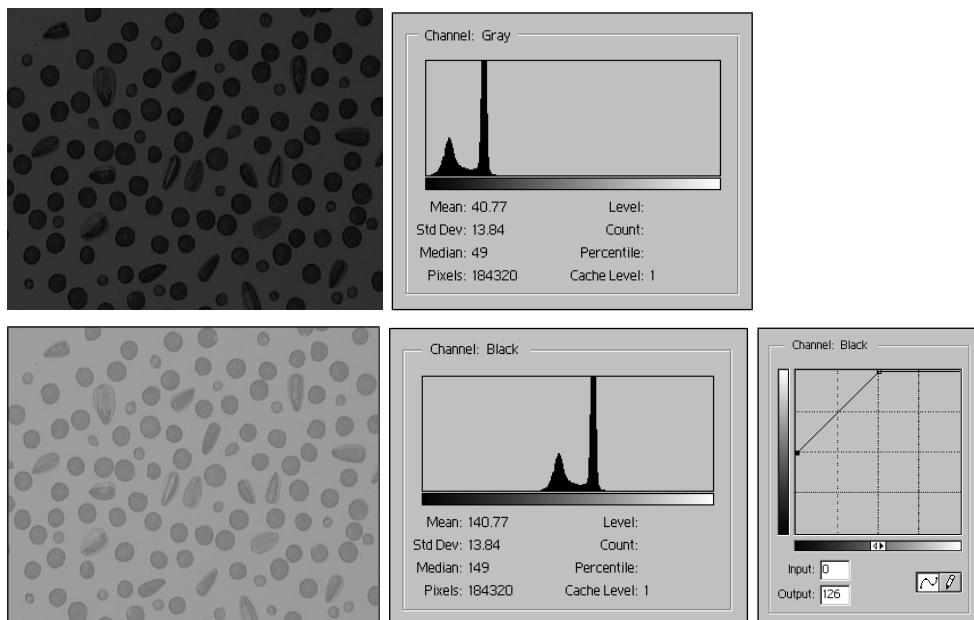
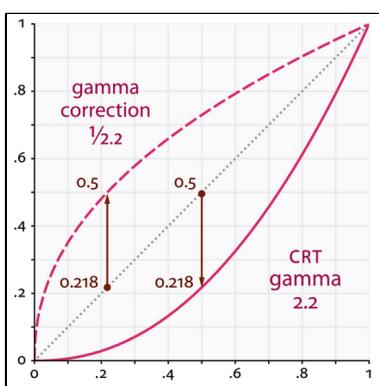


Abb. 75: Bild und Histogramm vor (oben) und nach der linearen Aufhellung (unten).

5.6 Nichtlineare Helligkeitskorrektur, Gammakorrektur

Im Gegensatz zur linearen Helligkeitskorrektur ist die *Gammakorrektur* eine nichtlineare Funktion, worin ein Exponent $\gamma < 1$ eine Aufhellung und ein Exponent $\gamma > 1$ eine Abdunklung bewirken.



$$g_A(x, y) = 255 \cdot \left(\frac{g_E(x, y)}{255} \right)^\gamma$$

Abb. 76: Nichtlineare Abbildungsfunktionen einer Gammakorrektur mit einem Gamma von 1/2.2 (Aufhellung) und 2.2 (Abdunklung). Bildquelle: http://en.wikipedia.org/wiki/Gamma_correction

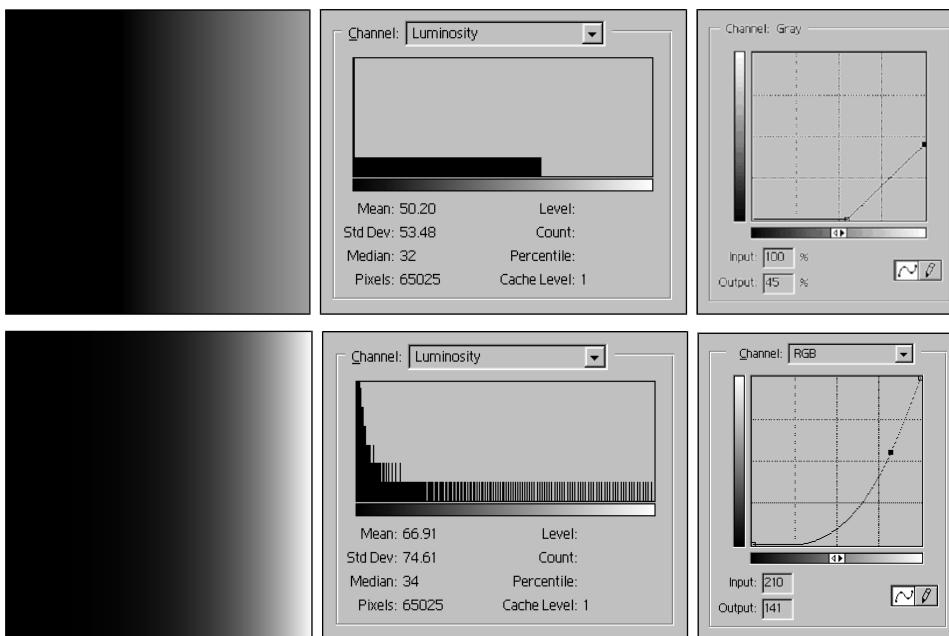


Abb. 77: Vergleich von zwei abgedunkelten Grauverläufen mit linearer Verschiebung (oben) und mit Gammakorrektur (unten).

Die Gammakorrektur hat den Vorteil, dass sie bei Bildern mit gleichmässig verteilten Graustufen einen geringeren Informationsverlust nach sich zieht. Durch die nichtlineare Abbildungsfunktion werden helle Partien weniger stark aufgehellt als Dunkle. Die Gammakorrektur wird deshalb auch eingesetzt, um bei Monitoren die Helligkeit und die Farbtemperaturen zu kalibrieren.

Gammakorrektur bei Monitoren

Computermonitore, egal ob Kathodenstrahl- oder LCD-Monitor, benötigen eine Gammakorrektur, da sie kein lineares Helligkeitsspannungsverhältnis haben. Um eine einheitliche Helligkeit unter den Monitoren zu gewährleisten, haben sich die Monitorhersteller auf ein Gamma von 2.5 geeinigt. Das bedeutet, dass die Helligkeit aus Spannung hoch 2.5 ergibt. Hätte man demnach ein Bild mit folgender gleichmässiger Grauwertverteilung, so würde ohne Gammakorrektur das Bild abgedunkelt auf dem Monitor wiedergegeben:



Abb. 78: Ausgangsbild mit gleichmässiger Helligkeitsverteilung



Abb. 79: Monitordarstellung ohne Gammakorrektur. Helligkeit $L = V^{2.5}$

Um also eine korrekte Darstellung zu erreichen, muss also vorgängig das Bildsignal mit einer Gammakorrektur aufgehellt werden:



Abb. 80: Aufhellung mit Gammakorrektur: Helligkeit $L' = L^{1/2.5}$



Abb. 81: Korrekte Darstellung am Monitor

Leider konnten sich die Monitorhersteller nun aber nicht plattformübergreifend einigen, ob und wie stark sie diese Gammakorrektur bereits in die Hardware einbauen und wie viel sie der Software, sprich dem Betriebssystem überlassen. Z.B. hat Apple sich entschlossen in ihren Monitoren das Signal um die Potenz $1/1.4$ anzuheben. Demnach muss das Mac OS nur noch eine Gammakorrektur von $1.8 (=2.5/1.4)$ vornehmen. Monitore für PCs oder Sun Rechner werden nicht hardwareseitig gammakorrigiert, weshalb dort das Betriebssystem die ganze Korrektur von $1/2.5$, manchmal auch $1/2.2$ vornimmt.

Mehr Informationen über die Hintergründe der Gammakorrektur finden Sie im Kapitel 5.6.2 von [Burger06].

5.7 Kontrastkorrektur

5.7.1 Lineare Kontrastkorrektur

Lineare Skalierungen bewirken eine gleichmässige Verschiebung, Streckung oder Stauchung des Grauwerteumfangs, was visuell in einer Erhöhung oder Verminderung des Kontrastes resultiert. Die Skalierung kann durch eine lineare Gleichung dargestellt werden. Wenn $b = 0$ und $c = 1$, resultiert daraus die identische Abbildung. Die Konstante c beeinflusst den Kontrast, während b die Helligkeit bestimmt.

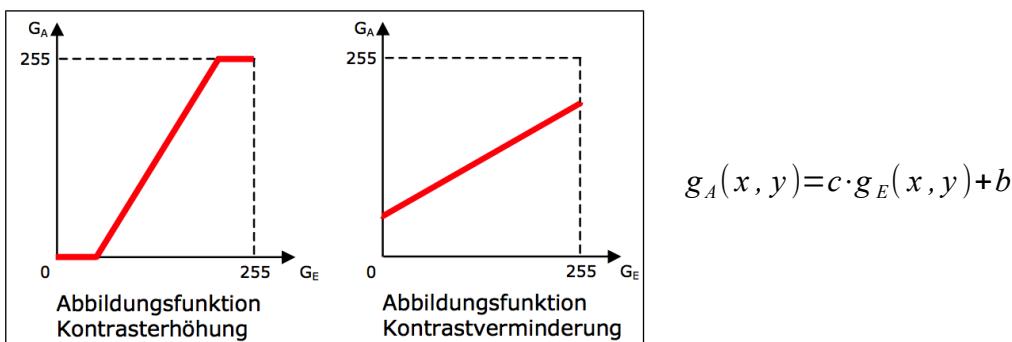


Abb. 82: Abbildungsfunktionen für lineare Kontrastkorrektur.

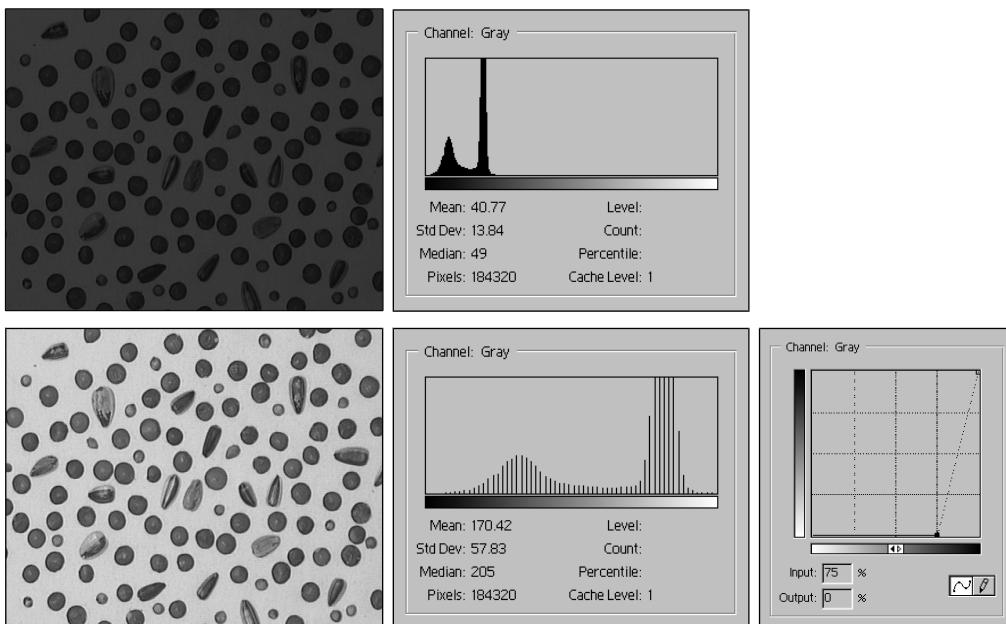


Abb. 83: Blutzellbild und Histogramm vor (oben) und nach der Kontrastkorrektur (unten) und die dazugehörige Abbildungsfunktion

5.7.2 Automatische Kontrasterhöhung

Eine automatische maximale Kontrasterhöhung kann wie folgt berechnet werden:

$$g_A(x, y) = (g_E(x, y) - g_{min}) \cdot \frac{255}{g_{max} - g_{min}}$$

Hat es im Bild aber ein ganz schwarzes und ein ganz weisses Pixel, so ist keine Kontrasterhöhung mehr möglich. Man umgeht das Problem, indem ganz oben und ganz unten einen gewissen Prozentsatz s der Pixel in Sättigung gehen lässt.

$$g_A(x, y) = \begin{cases} g'_{min} & \text{für } g < g'_{min} \\ (g_E(x, y) - g'_{min}) \frac{255}{g'_{max} - g'_{min}} & \text{für } g'_{min} < g < g'_{max} \\ g'_{max} & \text{für } g > g'_{max} \end{cases}$$

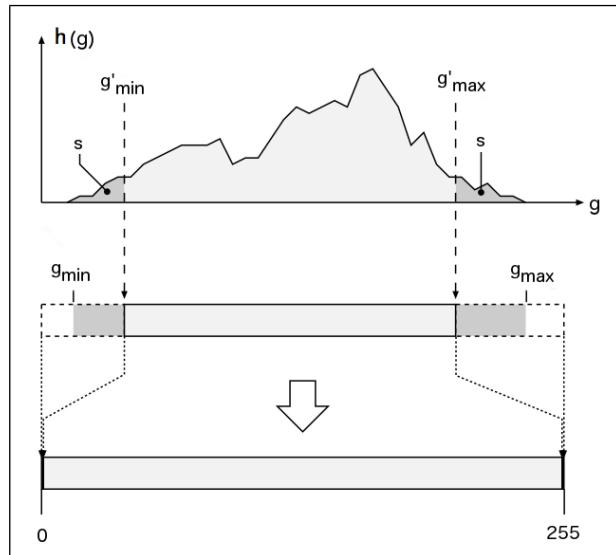


Abb. 84: Automatische Kontrasterhöhung mit einem Prozentsatz an Sättigung. (Bildquelle: [Burger06])

5.7.3 Kontrasterhöhung durch Histogrammausgleich

Während mit einer Skalierung die Form der Histogrammkurve nur gestreckt oder gestaucht wird, ist das Ziel eines Histogrammausgleichs mit annähernd gleichen Häufigkeiten $h(g)$ für alle Grauwertintervalle zu erreichen.

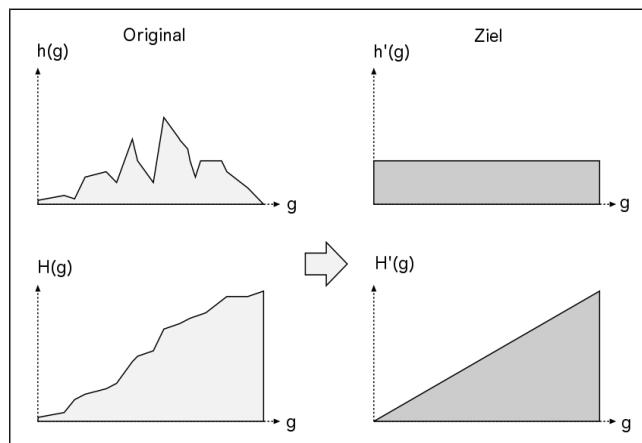


Abb. 85: Histogrammausgleich oben als gleichmäßige Verteilung über alle Graustufen. Im kumulativen Histogramm ergibt sich durch den Ausgleich ein gleichmäßig ansteigender Keil. (Bildquelle: [Burger06])

Als Abbildungsfunktion wird das kumulative Histogramm verwendet:

$$f_{eq}(g) = H(g) \cdot \frac{(K-1)}{N}$$

mit K als grösstmöglichen Grauwert und N als Summe aller Pixel. Eine Herleitung, warum das kumulative Histogramm als Umverteilungsfunktion verwendet werden kann, finden Sie in [Burger06] bzw. bei [Gonzales08]. Die kumulative Histogrammkurve steigt bei den Graustufen an, wo wir viele Pixel haben. Verwenden wir diese als Abbildungsfunktion, so werden diese Graustufen auseinandergerissen und in der Breite verteilt. Da zwischen entstehen aber keine neuen Graustufen, sondern Lücken.

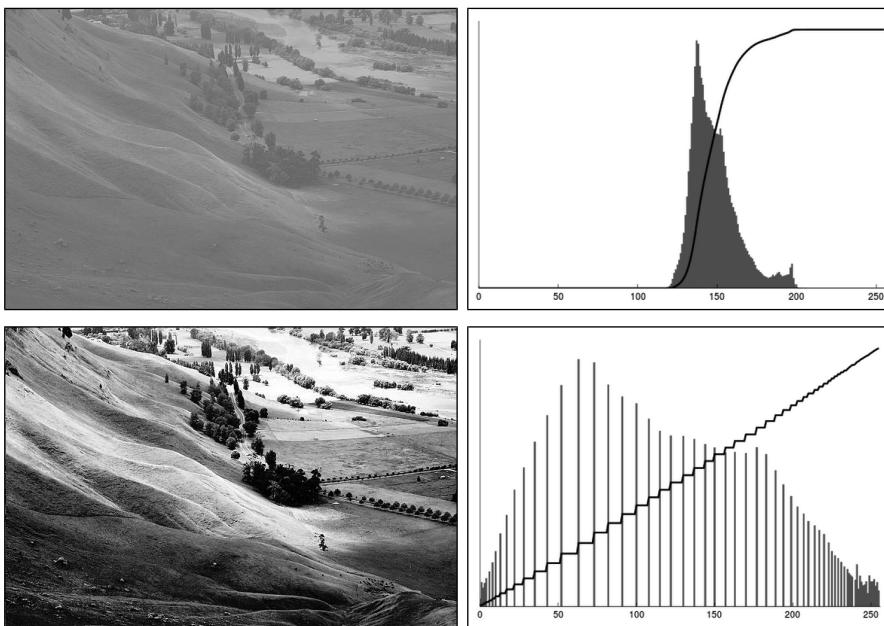
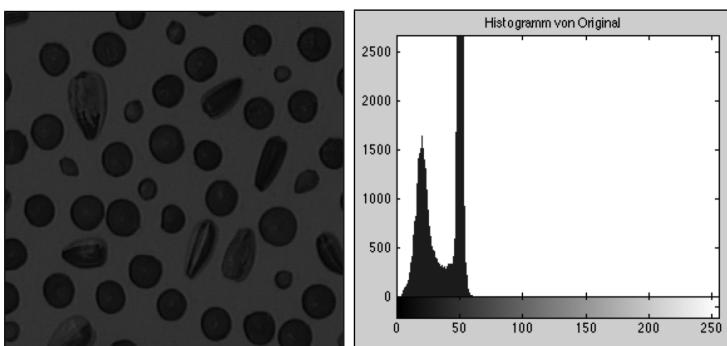


Abb. 86: Bild und Histogramm vor dem Ausgleich und unten nach dem Ausgleich. (Bildquelle: Wikipedia)

Im transformierten Bild ist die Zahl der verschiedenen Grauwerte meist kleiner als im Ausgangsbild. Der mittlere Informationsgehalt pro Pixel ist somit im ausgeglichenen Bild kleiner als im Original. Da unser Auge aber nur etwa 30 verschiedene Graustufen sicher auflösen kann, gewinnen wir, wegen des höheren Kontrastes, trotzdem den Eindruck, das Bild sei informationsreicher.

Der Histogrammausgleich bringt aber nicht immer das gewünschte Resultat. Insbesondere bei Bildern mit grossen uniformen Bereichen (= viele Pixel in wenigen Graustufen) kann es passieren, dass durch die Umverteilung ein starkes Rauschen entsteht, weil die wenigen Graustufen stark auseinander verteilt werden:



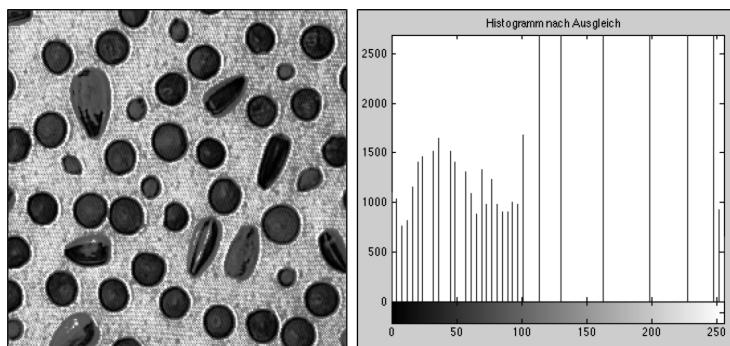


Abb. 87: Bild und Histogramm vor dem Ausgleich und unten nach dem Ausgleich. Zwischen den Blutzellen entstand starkes Rauschen.

In **ImageJ** finden Sie den linearen Histogrammausgleich unter dem Menü *Process > Enhance Contrast*.

5.7.4 Kontrasterhöhung durch Farben

Durch Farb-Lookup-Tabellen können aus Graustufenbildern feinste Grauunterschiede hervorgehoben werden. Da der Mensch nur etwa 60 Graustufen unterscheiden kann, kann er z. B. in einem 8-Bit-Bild mit 256 Graustufen nur etwa nach je 5 Graustufen einen Unterschied feststellen. Nachfolgend sehen Sie einen Ausschnitt aus der Schweizer Höhenmap, wo der tiefste Punkt bei Basel die Graustufe 0 hat und die höchsten Berge den Wert 255.

In **ImageJ** stehen unter dem Menü *Image > Lookup Tables* an die 30 verschiedenen Farbtabellen für 8-Bit-Bilder zur Verfügung. Je nach Farbtabelle können ganz unterschiedliche Aspekte visualisiert werden:

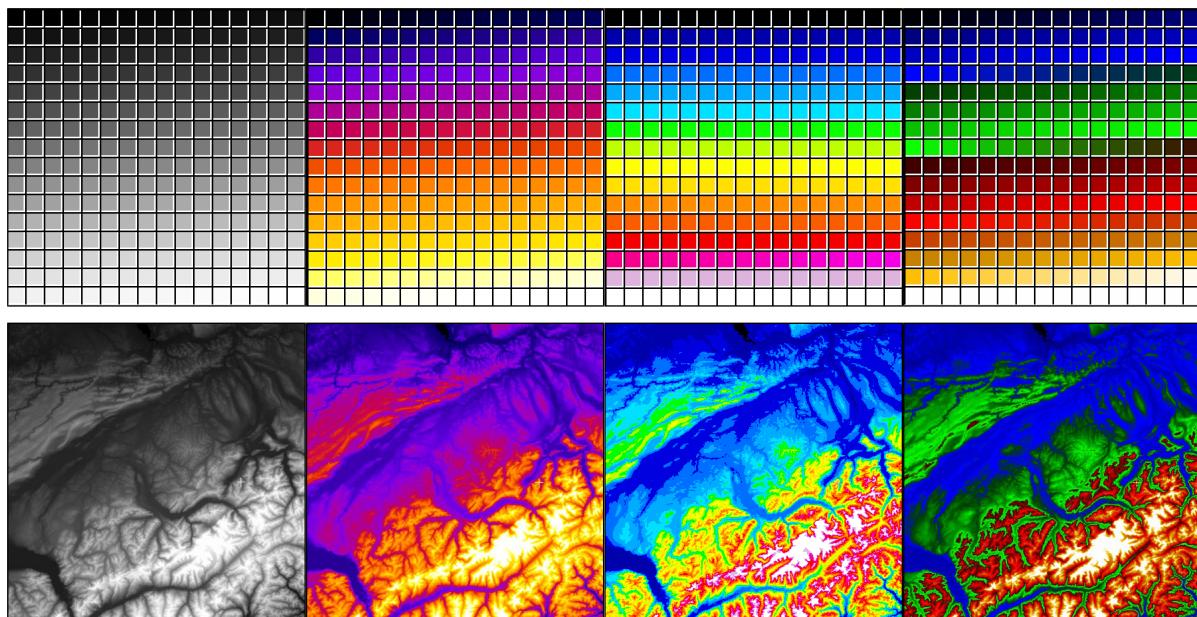
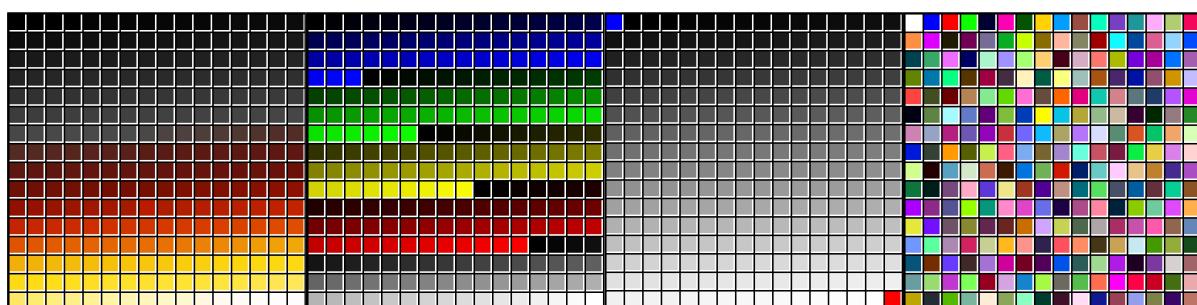


Abb. 88: V.l.n.r.: Original mit 256 Graustufen, Fire LUT, 16 Colors LUT, Thal LUT



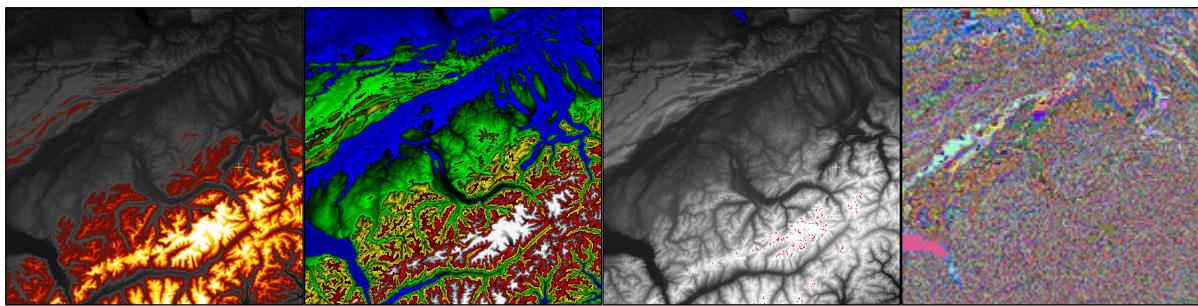


Abb. 89: V.l.n.r.: Smart LUT, 5 Ramps LUT, HiLow LUT, Glasbey Lut



In **Matlab** kann die Lookup-Tabelle mit dem Befehl [colormap](#) gesetzt werden.

5.8 Bildarithmetik

Bei der linearen Helligkeits- und Kontrastkorrektur haben wir bereits Bildpunktarithmetik betrieben, indem wir zum Pixelwert einen konstanten Wert b hinzugefügt oder ihn mit einem konstanten Wert c multipliziert haben, um den Kontrast zu verändern:

$$g_A(x, y) = c \cdot g_E(x, y) + b$$

Grundsätzlich können wir solche arithmetischen Operationen auch mit nicht konstanten Werten durchführen. D. h., wir addieren (bzw. subtrahieren) oder multiplizieren (bzw. dividieren) ganze Bilder, sodass jedes Pixel mit dem korrespondierenden Pixel eines zweiten Input-Bildes verarbeitet wird. Dabei ist natürlich immer zu beachten, dass die resultierenden Pixelwerte im gültigen Bereich von 0-255 (für 8-Bit-Bilder) liegen.



Fiji bietet gegenüber ImageJ im Menü *Process* einen verbesserten *Calculator Plus Dialog* mit folgenden Arithmetikoperationen:

Addition: $g_A(x, y) = (g_{E1}(x, y) + g_{E2}(x, y)) \cdot k1 + k2$

Subtraktion: $g_A(x, y) = (g_{E1}(x, y) - g_{E2}(x, y)) \cdot k1 + k2$

Multiplikation: $g_A(x, y) = (g_{E1}(x, y) \cdot g_{E2}(x, y)) \cdot k1 + k2$

Division: $g_A(x, y) = (g_{E1}(x, y) / g_{E2}(x, y)) \cdot k1 + k2$

Beispiel: Hintergrunds-Subtraktion oder Division

Bildarithmetik eignet sich z. B. gut um Beleuchtungsvariationen aus einem Bild zu entfernen, bevor man mit einer Binarisierung Vorder- und Hintergrund auftrennt:

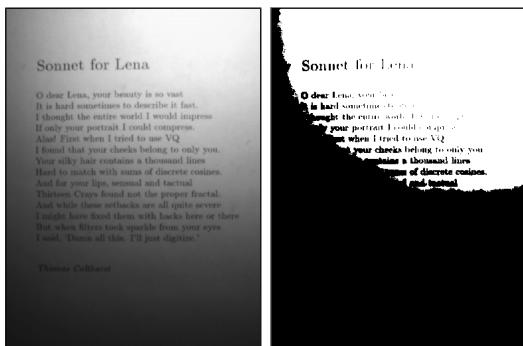


Abb. 90: Ein Bild mit starkem Beleuchtungsabfall lässt sich nicht mit globalem Schwellwert binarisieren.

Wir können die Beleuchtungsvariation entweder durch Subtraktion oder Division aus dem Bild rechnen.

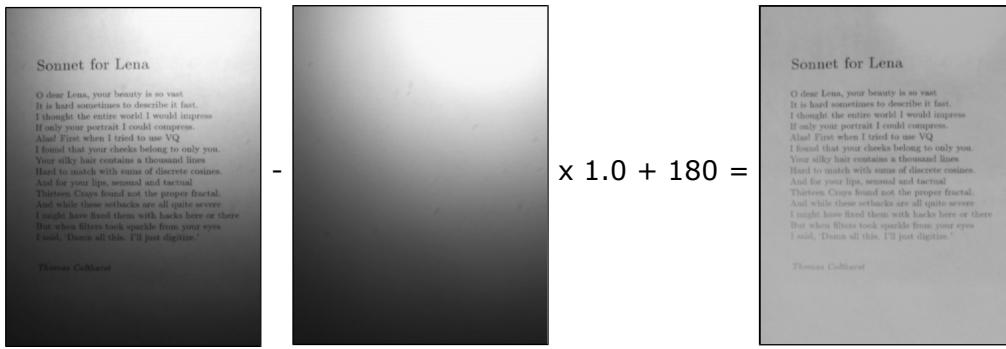


Abb. 91: Entfernen der Beleuchtungsvariation durch Bildsubtraktion

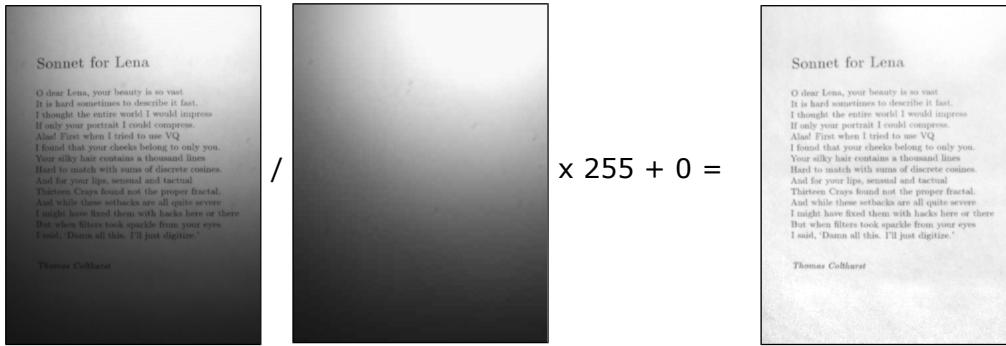


Abb. 92: Entfernen der Beleuchtungsvariation durch Bilddivision.

Die Frage bleibt, wie wir in diesem Beispiel zum Bild mit der variierenden Beleuchtung ohne den Text kommen. Dieses liesse sich z. B. durch einen Gaussfilter (s. Kapitel 6.1.2.2) oder Medianfilter (s. Kapitel 6.2.2.2) erzeugen. Diese Filter werden wir im Kapitel über lokale Operatoren kennenlernen. Vorab sei hier nur noch angemerkt, dass solche lokale Operatoren deutlich aufwendiger sind als Punktoperationen. Eine Alternative für dieses Anwendungsproblem wird im Kapitel 8.1.2 vorgestellt.

Beispiel: Absolute Differenz

Eine andere klassische Anwendung von Bildarithmetik ist die absolute Differenz, um verschobene Objekte zu detektieren.



ImageJ bietet dieses Werkzeug unter *Process > Image Calculator*:

$$g_A(x, y) = |g_{E1}(x, y) - g_{E2}(x, y)|$$



Abb. 93: Durch die absolute Differenz zweier Bilder werden die verschobenen Teile sichtbar.



Fiji bietet zudem im Menü *Process* den *Image Expression Parser*. Damit lassen sich fast beliebige Operationen auch mit mehr als zwei Bildern und vordefinierten ImageJ-Funktionen formulieren.

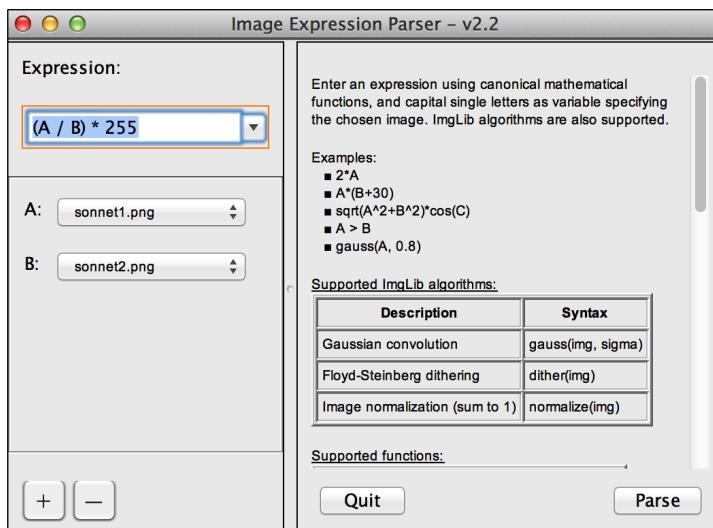
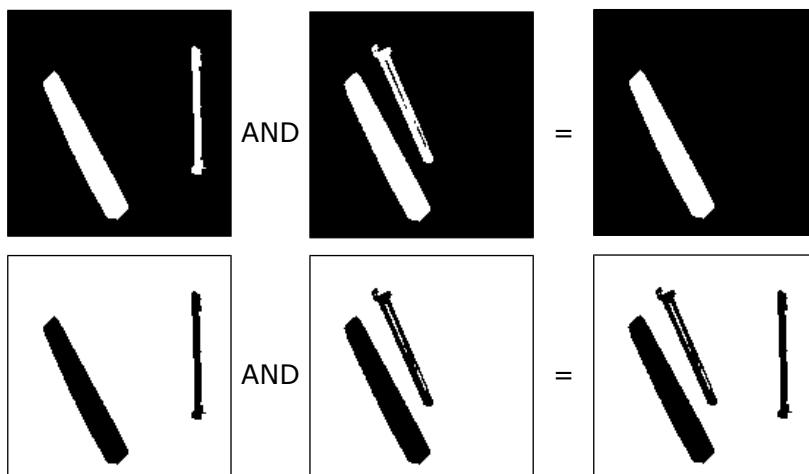


Abb. 94: Image Expression Parser Dialog in Fiji

5.9 Logische Operationen

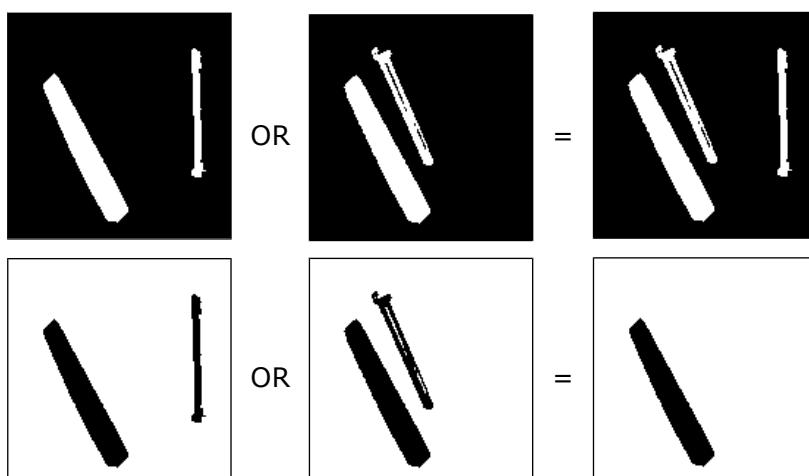
Logische Operationen wie AND, OR oder XOR eignen sich um mehrere Bilder zu kombinieren. Normalerweise werden diese Operationen mit Binärbildern verwendet, um bestimmte Regionen zu selektieren.

ImageJ bietet diese Operationen im Menü *Process > Image Calculator*:



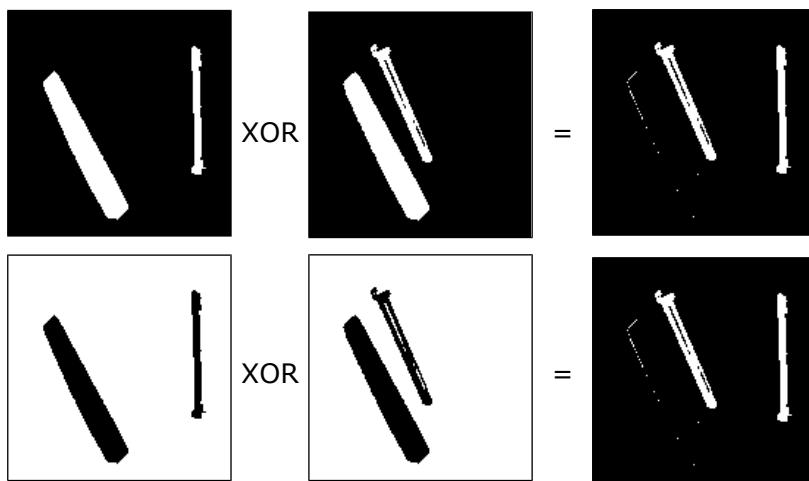
$$\begin{aligned} 0 \text{ AND } 0 &= 0 \\ 0 \text{ AND } 1 &= 0 \\ 1 \text{ AND } 0 &= 0 \\ 1 \text{ AND } 1 &= 1 \end{aligned}$$

Abb. 95: AND-Verknüpfungen



$$\begin{aligned} 0 \text{ OR } 0 &= 0 \\ 0 \text{ OR } 1 &= 1 \\ 1 \text{ OR } 0 &= 1 \\ 1 \text{ OR } 1 &= 1 \end{aligned}$$

Abb. 96: OR-Verknüpfungen



0 XOR 0=0
 0 XOR 1=1
 1 XOR 0=1
 1 XOR 1=0

Abb. 97: XOR-Verknüpfungen

5.10 Punktoperationen in ImageJ-Plug-ins

5.10.1 Standard-Punktoperationen

Folgende Punktoperationen ohne Parameter oder mit konstantem Parameter stehen in der ImageJ-Klasse *ImageProcessor* zur Verfügung:

void invert()	Inverts the image
void add(int value)	Adds 'value' to each pixel in the image
void subtract(double value)	Subtracts 'value' from each pixel in the image
void multiply(double value)	Multiplies each pixel in the image by 'value'
void and(int value)	Binary AND of each pixel in the image with 'value'
void or(int value)	Binary OR of each pixel in the image with 'value'
void xor(int value)	Binary XOR of each pixel in the image with 'value'
void gamma(double value)	Performs gamma correction of the image
void log()	Performs a log transform on the image
void exp()	Performs a exponential transform on the image
void sqr()	Performs a square transform on the image
void sqrt()	Performs a square root transform on the image
void abs()	Performs an absolute value transform
void min(double value)	Pixels less than 'value' are set to 'value'
void max(double value)	Pixels less than 'value' are set to 'value'

5.10.2 Punktoperation mit Lookup-Table

Eine beliebige Punktoperation werden üblicherweise mit einer **Lookup-Table** implementiert. D. h., man wendet zuerst die Abbildungsfunktion auf alle Grauwerte an und speichert das Resultat bei 8-Bit-Graustufen in einer 256 Elemente grossen Tabelle. Mit der Funktion *applyTable* kann diese Tabelle dann sehr schnell auf das ganze Bild angewendet werden. Im nachfolgenden Beispiel ist eine Gammakorrektur mit einem Exponenten von 2.8 implementiert:

```

public void run(ImageProcessor ip)
{ int K = 256;
  int aMax = K-1;
  double GAMMA = 2.8;

  // create and fill the lookup table
  int[] lut = new int[K];
  for(int a= 0; a<K ; a++)
  {   double aa = (double) a / aMax;           // scale to [0, 1]
      double bb = Math.pow(aa, GAMMA);          // gamma function int b = (int)
      Math.round(bb * aMax);                    // scale back to [0, 255] lut[a] = b;
  }
  ip.applyTable(lut);
}

```

```
}
```

5.10.3 Punktoperationen mit zwei Bildern

Folgende Standard-Punktoperationen mit zwei Bildern stehen über die *ImageProcessor*-Methode *copyBits* zur Verfügung. Ein Zielbild (*dst*) kann damit die Bits eines Quellbildes (*src*) in sich hineinkopieren. Wie die Pixel miteinander verrechnet werden, kann mit dem Parameter *mode* beeinflusst werden. Folgende Modi stehen zur Verfügung:

```
dst.copyBits(ImageProcessor src, int x, int y, int mode)

Modes:
ADD          dst = dst + src
AND          dst = dst AND src
AVERAGE      dst = (dst + src) / 2
COPY          dst = src
COPY_INVERTED dst = 255 - src (8-bits and RGB)
DIFFERENCE   dst = abs(dst - src)
DIVIDE       dst = dst / src
MAX          dst = max(dst, src)
MIN          dst = min(dst, src)
MULTIPLY     dst = src * src
OR           dst = dst OR src
SUBTRACT    dst = dst - src
XOR          dst = dst XOR src
```

Die Kopiermethode führt bei allen Operationen eine Begrenzung (*clamping*) auf den gültigen Wertebereich durch. Wem diese Modi nicht ausreichen, kann eine komplexere Operation ein mehrere Standardoperationen aufteilen oder die Operation mit einer eigenen for-Schlaufe implementieren.

Im Kapitel 5.7.4 von [Burger06] finden Sie ein Anwendungsbeispiel für ein Plugin, das mit zwei Bildern arbeitet. Standardmäßig arbeiten Plugins ja immer mit einem (dem aktiven) Bild. Für den Fall, dass Sie mit zwei oder mehreren Bildern arbeiten wollen, müssen Sie einen Auswahldialog anzeigen, mit dem weitere Bilder von nichtaktiven Fenstern ausgewählt werden können.



5.11 Übung: Punktoperationen mit Matlab

Aufgaben:

- 1) Lassen Sie folgendes Matlab-Skript *PunktOperationen.m* laufen und studieren Sie es. Lesen Sie in der Matlab-Hilfe nach, welche Optionen die Befehle *imread* und *imshow* haben.

```

X = imread('Blood2.bmp','bmp');           % liest BMP in Matrix X
I = im2single(X);                      % Konvertierung in Mat. I mit floats (0-1)
imshow(I);                             % Bild anzeigen
title('Original');
pause;

fprintf ('\n\nBerechne Statistiken ...');
Imin = min(I(:));                     % Berechnet Minimum
Imax = max(I(:));                     % Berechnet Maximum
Imea = mean(I(:));                    % Berechnet Durchschnitt
Istd = std(I(:));                     % Berechnet Standardabweichung
Ivar = var(I(:));                     % Berechnet Varianz
Iskw = skewness(I(:));                % Berechnet Schiefe
Ikur = kurtosis(I(:));               % Berechnet Exzess

fprintf ('\nMinimum : %f', Imin);
fprintf ('\nMaximum : %f', Imax);
fprintf ('\nMittelwert: %f', Imea);
fprintf ('\nStd.abw : %f', Istd);
fprintf ('\nVarianz : %f', Ivar);
fprintf ('\nSchiefe : %f', Iskw);
fprintf ('\nKurtosis : %f', Ikur);

fprintf ('\n\nHistogramm Bearbeitung ...');
[B,map] = gray2ind (I,256);          % Float Graustufen zu 256 Integer-Graustufen
imhist(B);                           % Histogramm anzeigen
title('Histogramm von Original');
pause;

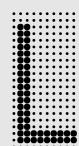
I2 = histeq(I);                      % Histogrammausgleich
[B,map] = gray2ind(I2,256);          % Float Graustufen zu 256 Integer-Graustufen
imshow(I2);
title('Nach Histogrammausgleich');
pause;
imhist(B);
title('Histogramm nach Ausgleich');
pause;

I2 = imadjust(I,[Imin Imax],[0 1]); % Kontrasterhöhung durch Spreizung
[B,map] = gray2ind(I2,256);
imshow(I2);
title('Nach Kontrastspreizung');
pause;
imhist(B);
title('Histogrammausgleich nach Kontrastspreizung');
pause;

```

- 2) Probieren Sie eigene Bilder aus. Wandeln Sie ihre Bilder in Graustufenbilder um.
- 3) Führen Sie eine lineare Helligkeitskorrektur durch. Vergleichen Sie das Histogramm vorher und nachher.
- 4) Führen Sie eine Gammakorrektur durch. Vergleichen Sie das Histogramm vorher und nachher.
- 5) Binarisieren Sie ein Bild mit der Funktion *im2bw* und der automatischen Schwellwertfunktion *graythresh*. Welche Methode verwendet *graythresh*?

6 Lokale Operatoren



lokale Operatoren beziehen im Unterschied zu den Punktoperatoren auch die Umgebung des Bildpunktes in die Berechnung ein. Lokale Operatoren werden oft auch als *Filter* bezeichnet. Wie die Punkt-Operatoren werden sie im Ortsraum angewendet und hauptsächlich bei der Bildvorverarbeitung eingesetzt, um eine visuelle Beurteilung des Bildmaterials zu erleichtern oder um eine nachfolgende Bildanalyse zu ermöglichen.

Die wichtigste Gruppe der lokalen Operatoren ist die der *linearen lokalen Operatoren*. Sie ermitteln den Pixelwert im Zielbild als Summe der Pixel in der durch eine Maske definierten Umgebung. Die Maske, manchmal auch *Filterkernel* genannt, ordnet dabei jedem Bildpunkt in der Umgebung einen Gewichtsfaktor zu und berechnet damit den Zielwert des neuen Pixels im Zentrum der Maske. Man lässt die Filtermaske mit dem Zentrumspixel über alle Pixel des Quellbilds laufen und schreibt das Resultat ins Pixel im Zielbild. Aus Filtermasken sind meistens quadratisch und von ungerader Größe (3x3, 5x5 etc.).

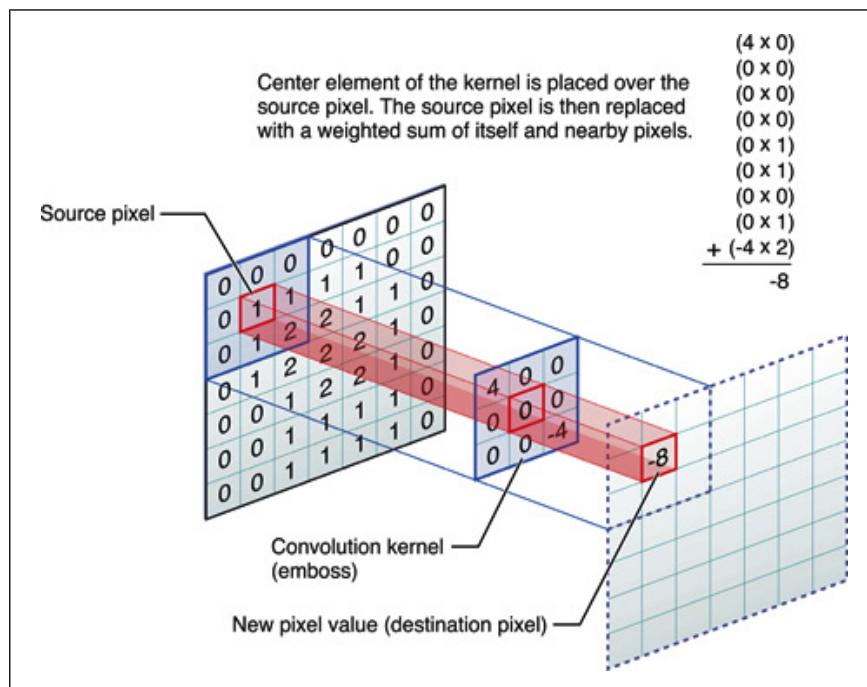


Abb. 98: Die gewichtete Summe aller Faktoren der Filtermaske h mit den darunterliegenden Pixeln des Quellbilds bildet den neuen Pixelwert im Zielbild. (Bildquelle: Apple Mac Developer Library)

Die lineare Filterung eines Graustufenbildes g mit einer quadratischen Filtermaske h der Größe $2k+1$ sieht wie folgt aus:

$$f(x, y) = g * h = \sum_{u=-k}^k \sum_{v=-k}^k g(x+u, y+v) \cdot h(u, v)$$

Der Punkt bei X,Y berechnet sich demnach wie folgt:

$$\begin{aligned}
 f(X, Y) = & g(X-1, Y-1) \cdot h(-1, -1) + g(X, Y-1) \cdot h(0, -1) + g(X+1, Y-1) \cdot h(1, -1) + \\
 & g(X-1, Y) \cdot h(-1, 0) + g(X, Y) \cdot h(0, 0) + g(X+1, Y) \cdot h(1, 0) + \\
 & g(X-1, Y+1) \cdot h(-1, +1) + g(X, Y+1) \cdot h(0, +1) + g(X+1, Y+1) \cdot h(1, +1)
 \end{aligned}$$

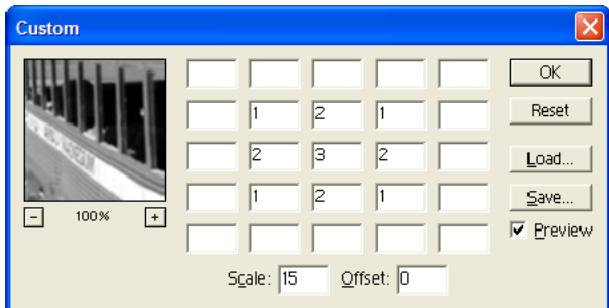


Abb. 99: Die meisten Bildverarbeitungsprogramme besitzen einen Dialog wie hier in Photoshop, um eine beliebige Filtermaske zu definieren.

Randproblem

An den Rändern des Bildes sind besondere Massnahmen zu treffen, damit die Maske nur in definierten Bereichen arbeitet. Angenommen man will ein gleich grosses Ausgangsbild wie das Eingangsbild, so müssen die angrenzenden Pixel mit Werten aufgefüllt werden:

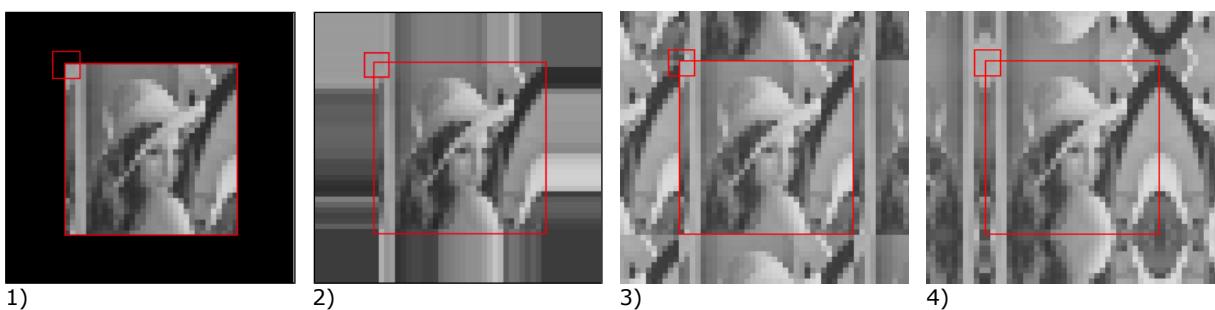


Abb. 100: Lösungen für das Randproblem:

Beim 1. Bild wird um das Bild mit 0 ergänzt. Dies führt zu einem klar sichtbaren Rand.

Beim 2. Bild wird der Bildrand nach aussen kopiert. Im Ausgangsbild tritt der Rand so kaum in Erscheinung.

Beim 3. Bild wird das Bild repliziert. Es entsteht ein periodisches Bildsignal

Beim 4. Bild wird das Bild gespiegelt repliziert. Es entsteht ebenfalls ein periodisches Bildsignal aber mit weniger starken Kanten.

6.1 Lineare Faltung

Mathematisch gesehen handelt es sich bei der lokalen Filterung um eine **diskrete lineare Faltung (Konvolution)** zweier Funktionen g und h , die in 1D und 2D wie folgt definiert ist:

$$f(x) = g(x) * h(u) = \sum_{u=-\infty}^{\infty} g(x+u) \cdot h(-u)$$

$$f(x, y) = g(x, y) * h(u, v) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} g(x+u, y+v) \cdot h(-u, -v)$$

In dieser Definition sehen Sie, dass der Anwendungsbereich von $-\infty$ bis ∞ geht, also nicht auf die Maskengröße beschränkt ist und das die eine Funktion $h(u, v)$ mit negativen Vorzeichen also gespiegelt angewendet wird. In der Praxis wird allerdings diese Spiegelung meistens weggelassen, da die Filtermasken meistens symmetrisch sind. Wir werden später sehen, dass eine Faltung ohne diese Spiegelung, der verwandten Operation der *Korrelation* (s. Kapitel 11.3.2) entspricht.

Die Operation der eindimensionalen Faltung kann gut mit einer Animation visualisiert werden. Eine ähnliche Faltungsanimation finden Sie im [Wikipediaartikel](#) über die Faltung:

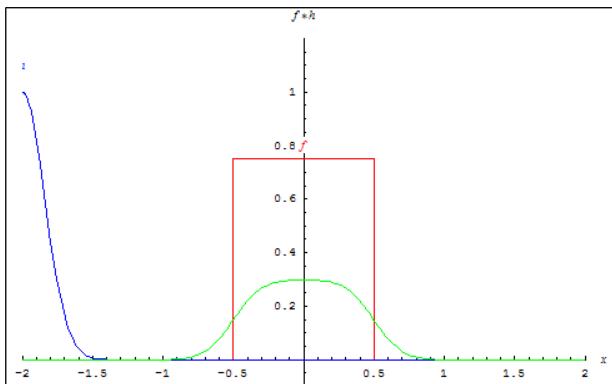


Abb. 101: Eindimensionale Faltung als gemittelte Multiplikation zweier Funktionen. [Quelle und Animation: Wikipedia](#).

6.1.1 Eigenschaften der Faltung

Für die Faltung gelten folgende Eigenschaften:

- **Kommutativgesetz:** $g * h = h * g$
- **Assoziativgesetz:** $g * h * i = g * (h * i) = (g * h) * i$
- **Distributivgesetz:** $g * h + i * h = (g + i) * h$

Das Distributivgesetz besagt also, dass ein Filter h auf zwei Bilder g und i vor oder nach einer Addition angewendet werden kann. Die Eigenschaft trifft nur für die *lineare Faltung* zu und bedeutet, dass die Reihenfolge der Operationen im Ergebnis keine Auswirkungen haben darf.

Als Beispiel wählen wir einen Filterkern h mit $[1 \ 2 \ 1]$, als ersten Bildausschnitt $g[\dots 0 \ 0 \ 1 \ 0 \ 0 \dots]$ und als zweiten Bildausschnitt $i[\dots 0 \ 0 \ 0 \ 1 \ 0 \ 0]$.

Als Erstes führen wir die Faltung und danach die Addition aus:

$$\begin{aligned} [1 \ 2 \ 1] * [\dots 0 \ 0 \ 1 \ 0 \ 0 \ 0 \dots] + [1 \ 2 \ 1] * [\dots 0 \ 0 \ 0 \ 1 \ 0 \ 0 \dots] &= \\ [\dots 0 \ 1 \ 2 \ 1 \ 0 \ 0 \dots] + [\dots 0 \ 0 \ 1 \ 2 \ 1 \ 0 \ 0 \dots] &= \\ [\dots 0 \ 1 \ 3 \ 3 \ 1 \ 0 \dots] \end{aligned}$$

Nun führen wir erst die Addition, dann die Faltung aus:

$$\begin{aligned} [1 \ 2 \ 1] * ([\dots 0 \ 0 \ 1 \ 0 \ 0 \ 0 \dots] + [\dots 0 \ 0 \ 0 \ 1 \ 0 \ 0 \dots]) &= \\ [1 \ 2 \ 1] * [\dots 0 \ 0 \ 1 \ 1 \ 0 \ 0 \dots] &= \\ [\dots 0 \ 1 \ 3 \ 3 \ 1 \ 0 \dots] \end{aligned}$$

Diese Eigenschaft gilt nicht für nichtlineare Filter wie z. B. den Medianfilter, den wir später behandeln werden.

Wegen dieser Ortsunabhängigkeit der Filtermaske bezeichnet man die linearen Operatoren auch als homogene lineare Operatoren oder LSI-Operatoren (engl. *linear shift-invariant*). *Shift-invariant* oder ortsunabhängig bedeutet, wenn man das Ausgangsbild der Filterung um X Pixel in die Richtung Y verschiebt, so verschiebt sich auch das gefilterte Bild um genau X Pixel in Richtung Y. Ansonsten ist das Ergebnis der Filterung identisch.

6.1.1.1 Separierbarkeit von Filtern

Aus dem Assoziativgesetz können wir eine wichtige Eigenschaft der linearen Faltung ableiten. Wir können eine Faltungsmaske zerlegen in zwei Teilmasken h_x und h_y und das Bild g einzeln und nacheinander damit Falten:

$$g * h_{xy} = g * (h_x * h_y) = (g * h_x) * h_y$$

$$h_{xy} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = h_x * h_y = [1 \ 1 \ 1 \ 1 \ 1] * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$g * h_{xy} = g * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = (g * [1 \ 1 \ 1 \ 1 \ 1]) * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Wenn wir ein Bild mit der Filtermaske h_{xy} falten, so ergeben sich $5 \times 3 = 15$ Operationen (Multiplikationen und Additionen) pro Pixel im Bild. Wenn wir das Bild aber zuerst mit h_x und dann mit h_y Falten entstehen nur $5 + 3 = 8$ Operationen pro Pixel. Grundsätzlich wächst der Aufwand ohne Separierung im Quadrat, während er mit Separierung nur linear wächst mit der Filtergrösse.

6.1.2 Tiefpassfilter

Tiefpassfilter schwächen die hochfrequenten Teile des Bildinhaltes ab. Hohe Frequenzen sind überall dort, wo entweder feine Details oder abrupte Änderungen vorkommen.

Typische Eigenschaften des Tiefpassfilters sind:

- Der visuelle Eindruck des Bildes wird weicher.
- Grauwertkanten werden verwischt.
- Details und Rauschen werden abgeschwächt.
- In homogenen Bereichen haben Tiefpassfilter keinen Einfluss.

6.1.2.1 Rechteckfilter

Der einfachste Filter dieses Typs ist der Rechteckfilter (*box filter*). Der Rechteckoperator berechnet als neuen Wert für einen Bildpunkt das arithmetische Mittel aus den Umgebungspixeln. Die Mittelung der Umgebungspixel mit gleichen Gewichtsfaktoren hat eine Glättung von Rauschstörungen zur Folge. Die kleinste Filtermaske ist eine 3x3-Maske mit allen Einsen und einem Gewichtsfaktor von 1/9.

$$R = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

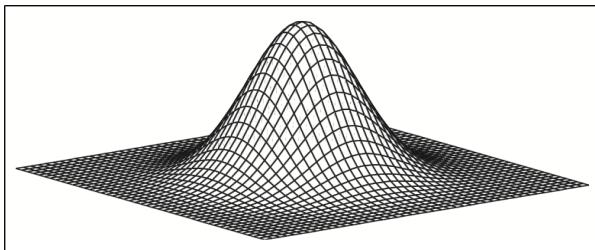


Abb. 102: Bild vor und nach der Anwendung des Rechteckfilters.

6.1.2.2 Gaussfilter

Ein besserer Tiefpassfilter, insbesondere für grössere Filtermasken ist der Gaussfilter, der einer Gaussglocke entspricht. Für einen quadratischen Gaussfilter berechnet sich der Filterkoeffizient an der Stelle $[i,j]$ mit der 2D-Funktion der [Normalverteilung](#):

$$G(i, j) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\left(\frac{i^2+j^2}{2\sigma^2}\right)}$$



Die ungerade Filtergrösse L berechnet sich aus dem σ der Normalverteilung wie folgt:

$$L = 2 \cdot \text{ceil}\left(\frac{\sigma - 0.8}{0.3} + 1\right) + 1$$

Die Gauss-Filtermatrix kann auch aus der Faltung zweier eindimensionaler [Binomialverteilungen](#), die eine gute Näherung für die Gaussfunktion im diskreten Definitionsbereich darstellt. Binomialverteilungen ermittelt man mit dem [Pascal'schen Dreieck](#) und der dazugehörigen Gewichtung f für die Normalisierung der Fläche unter der Gaussglocke:

Pascalsches Dreieck: Bildung der Gaussmaske:

Size	Weight	Sigma
1	1/1	
2	1/2	
3	1/4	
4	1/8	
5	1/16	0.8
6	1/32	1.1
7	1/64	1.4
8	1/128	1.7
9	1/256	
10	1/512	
11	1/1024	2.0

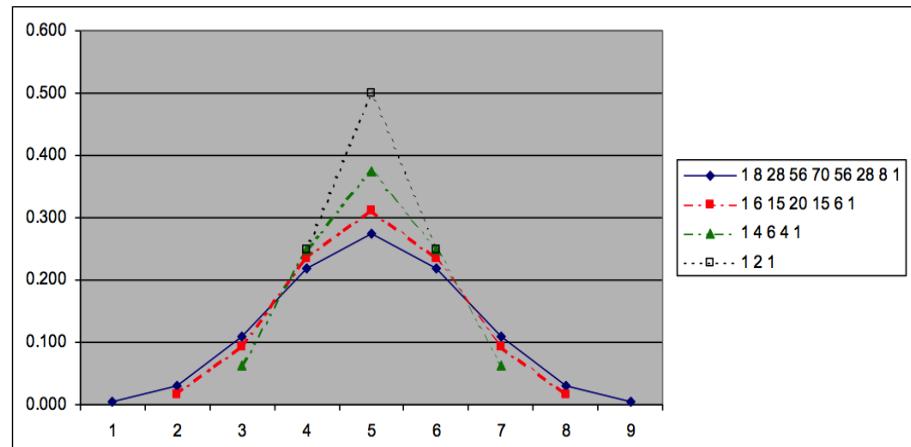


Abb. 103: Ungerade Binomialverteilungen als Annäherung an die Gausskurve

Das σ einer Gauss-Filtermatrix von ungerader Grösse L berechnet sich aus:

$$\sigma = 0.3 \cdot \left(\frac{L-1}{2} - 1\right) + 0.8$$

Die kleinste 3x3-Maske des Gaussfilters mit einem σ von 0.5 sieht wie folgt aus:

$$G = \frac{1}{4} \cdot [1 \ 2 \ 1] * \frac{1}{4} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Die 5x5-Maske einem σ von 1.0:

$$G = \frac{1}{16} \cdot [1 \quad 4 \quad 6 \quad 4 \quad 1] * \frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

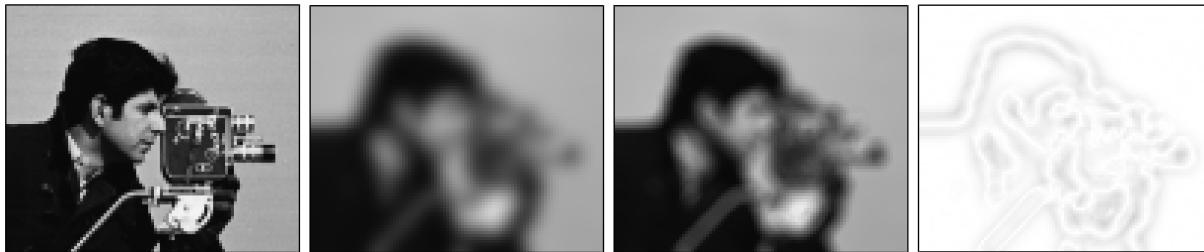


Abb. 104: v.l.n.r.: Original, mit 7x7 Rechteckfilter, mit 7x7 Gaussfilter und negative Differenz davon.

6.1.2.3 Übung: Faltungsfilter mit Excel

Implementieren Sie eine Faltung mit einer 3x3 Filtermaske in *Excel* oder *OpenOffice Calc*. Wählen Sie die kleinstmögliche Schrift in quadratischen Zellen. Das Gewicht der Filtermaske soll in einer separaten Zelle angegeben werden. Implementieren sie die Filterformel in der jeweiligen Zielzelle. Beim Kopieren der Formel in andere Zellen werden die Indizes automatisch mitgeführt. Um die Zellenindizes der Filtermaske zu fixieren, müssen zwei \$-Zeichen angefügt werden (z. B. D\$15\$). Die Einfärbung der Zellen kann über die bedingte Formatierung mit Farbskalen erreicht werden.

Abb. 105: Faltung mit Gaussfilter in Excel

6.1.3 Hochpassfilter

Der Hochpassfilter filtert die niedrigen Frequenzen eines Bildes heraus. Typische Eigen-
schaften dieses Filtertyps sind die Folgenden:

- Grauwertübergänge (Kanten) werden weiss hervorgehoben.
 - Homogen Bildbereiche werden zu Schwarz.
 - Das Rauschen wird verstärkt.

Geeignete Operatoren für die Hochpassfilterung ergeben sich aus der Überlegung, dass Kanten als Grauwertänderungen sich ausdrücken in der ersten und zweiten Ableitung des Bildsignals. Grauwertänderungen werden in diesem Zusammenhang auch als *Grauwertgradienten* oder einfach als *Gradienten* bezeichnet.

Eine Filtermaske lässt sich leicht aus der diskreten 1. Ableitung durch Differenzgleichungen herleiten. Getrennt nach ihren Richtungen ist die 1. Ableitung eines zweidimensionalen Grauwertbildes wie folgt definiert:

$$G'(x) = \frac{dG}{dx}(x) = G(x+dx) - G(x) = -1 \cdot G(x) + 1 \cdot G(x+1)$$

$$G'(y) = \frac{dG}{dy}(y) = G(y+dy) - G(y) = -1 \cdot G(y) + 1 \cdot G(y+1)$$

Daraus können wir direkt den einfachsten und schnellsten *Differenzoperator* oder *Gradientenoperator* für die 1. Ableitung eines Bildes machen:

$$D_x = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad D_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

In diagonaler Richtung heisst dieser Filter *Roberts Operator* oder *Roberts Cross Operator*:

$$D_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad D_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Als 3x3-Maske heisst dieser Filter *Prewitt Operator*:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad D_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

G(x):	9	9	9	9	9	8	7	6	5	4	3	2	2	2	2	2	3	5	8	10	11	11	11	11	11
G'(x):	0	0	0	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0
G''(x):	0	0	0	-1	0	0	0	0	0	0	1	0	0	0	1	1	1	-1	-1	-1	0	0	0	0	0

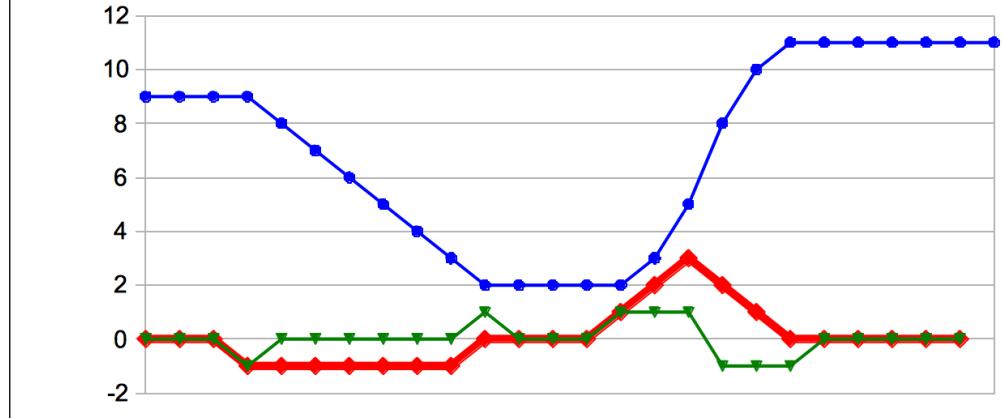


Abb. 106: 1. Ableitung (Rot) eines diskreten 1D-Signals (Blau) mit den Gewichten $[-1, 1]$ in x-Richtung. Die 2. Ableitung (Grün) wird im nächsten Kapitel hergeleitet. Die 1. Ableitung hat ihre Maxima bei der grössten Steigung und ihre Minima bei der grössten Senkung. Die 2. Ableitung hat an diesen Stellen einen negativen bzw. einen positiven Nulldurchgang aufweist.

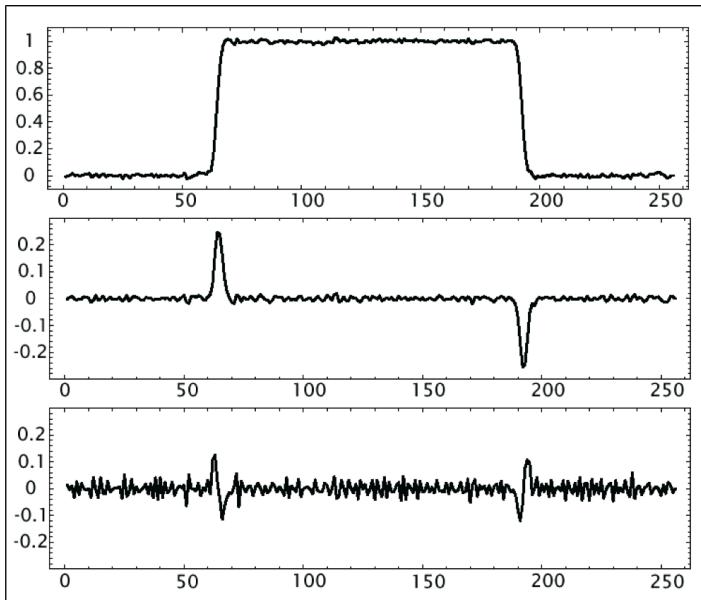


Abb. 107: 1D-Signal oben sowie seine 1. und 2. Ableitung darunter. In diesem grösseren Beispiel wird klar ersichtlich, dass das Rauschen mit jeder Ableitung zunimmt.

Um alle Kanten richtungsunabhängig zu erhalten, berechnen wir den *Gradientenbetrag* aus zwei rechtwinklig zueinanderstehenden Gradientenbildern:

$$|G'| = \sqrt{G'_x^2 + G'_y^2}$$

Vereinfacht aber weniger genau können auch die Beträge der Gradientenkomponenten summiert werden:

$$|G'| = |G'_x| + |G'_y|$$

Die *Gradientenrichtung* berechnet sich wie folgt:

$$\theta = \text{atan} \left(\frac{G'_y}{G'_x} \right)$$

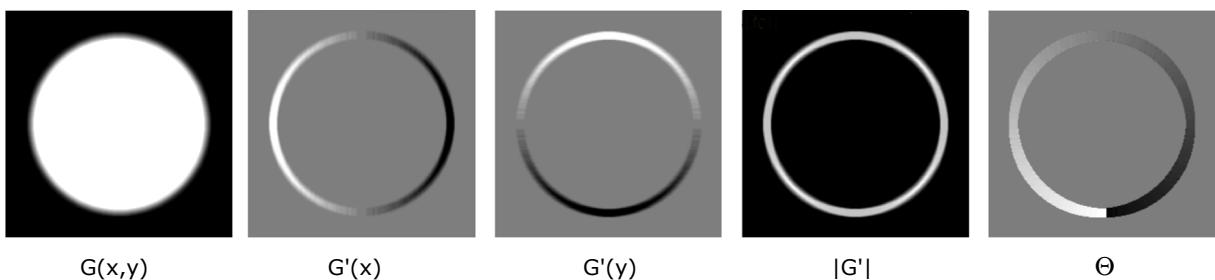


Abb. 108: $G(x,y)$: Originalgraustufenbild

$G'(x)$: Gradient in x-Richtung, in der Helligkeit angehoben um die negativen Werte in Schwarz darzustellen.

$G'(y)$: Gradient in y-Richtung, in der Helligkeit angehoben um die negativen Werte in Schwarz darzustellen.

$|G'|$: Gradientenbetrag aus Addition der Beiträge der beiden Gradienten.

Θ : Visualisierung der Gradientenrichtung von Weiss bis Schwarz.



Abb. 109: v.l.n.r.: Original, Gradienten in x-Richtung, Gradienten in y-Richtung. Wie Sie sehen, verschwinden die negativen Kanten, weil nur positive Werte in einem 8-Bit-Graustufenbild gespeichert werden können.

6.1.3.1 Sobelfilter

Der Sobelfilter ist die kleinste diskrete Approximation an die 1. Ableitung des Gaussfilters und soll damit möglichst wenig zusätzliches Rauschen erzeugen.

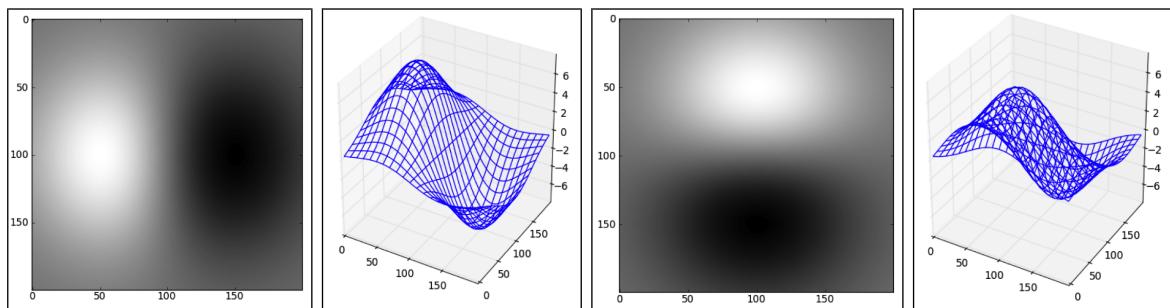


Abb. 110: v.l.n.r.: 1. Ableitung der Gaussglocke in x- und y-Richtung.

Nachfolgend die Filtermasken des Sobelfilters in X- und Y-Richtung sowie eine Diagonallrichtung:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad S_{dl} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$



Abb. 111: v.l.n.r.: Sobelfilter in x-Richtung, in y-Richtung und diagonal. Für die entgegengesetzten Richtungen müssten einfach die Vorzeichen geändert werden.

Wie Sie sehen, verschwinden die negativen Kanten auch beim Sobelfilter, weil nur positive Werte in einem 8-Bit-Graustufenbild gespeichert werden können.

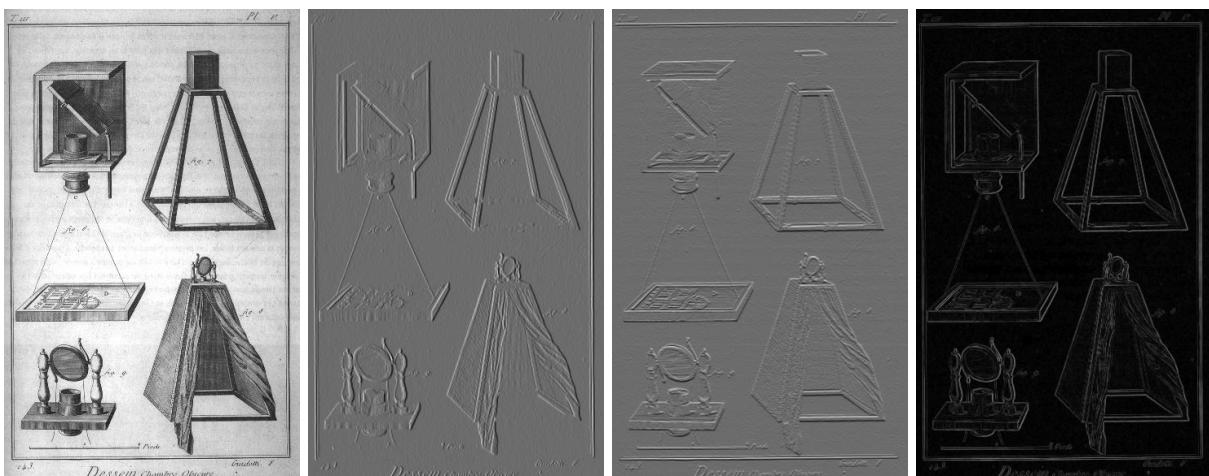


Abb. 112: v.l.n.r.: Originalbild, nach Sobel in X-Richtung, nach Sobel Y-Richtung und nach Sobel in beide Richtungen kombiniert durch Addition der Absolutwerte. Bei den mittleren Bildern wurde der Grauwert um 128 angehoben, um die negativen Werte sichtbar zu machen.

Weitere Vertreter von Kantendetektoren finden Sie im [Wikipediaartikel](#).

Abb. 113: Sobelfilter in X-Richtung in Excel

6.1.3.2 Laplace Filter als 2. Ableitung

Zur richtungsunabhängigen Detektion von Kanten eignen sich Filter, die auf einer diskreten Approximation der 2. Ableitung des Bildsignals beruhen. Die 2. Ableitung ist die Ableitung der 1. Ableitung:

$$G'(x) = \frac{dG}{dx}(x) = G(x+dx) - G(x)$$

$$G''(x) = \frac{d^2 G}{d^2 x}(x) = \frac{dG}{dx}(x+dx) - \frac{dG}{dx}(x)$$

$$G''(x) \equiv (G(x+1) - G(x)) - (G(x) - G(x-1))$$

$$G'''(x) \equiv 1 \cdot G(x-1) - 2 \cdot G(x) + 1 \cdot G(x+1)$$

und dito für die y-Richtung:

$$G''(v) \equiv 1 \cdot G(v-1) - 2 \cdot G(v) + 1 \cdot G(v+1)$$

Daraus können wir wiederum direkt die Gewichte für die Masken erstellen. Durch die Summe beider Richtungen ergibt sich zudem eine rotationsinvariante Maske für den als [Laplace-Operator](#) bekannten Filter:

$$L = L_x + L_y = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Die Summe mit den zusätzlichen Diagonalrichtungen ergibt einen stärkeren Laplace-Operator, der aber ebenfalls das Rauchen verstärkt:

$$L = L_x + L_y + L_{dl} + L_{d2} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Abb. 114: v.l.n.r.: Original, mit [1 -4 1]-Laplace-Filter, mit [1 -8 1]-Laplace-Filter

Die Laplace-Filtermaske mit $[1 \ -8 \ 1]$ erinnert uns auch an die Verarbeitung in unserem Auge. Dies sind genau die Gewichtsfaktoren, welche in den rezeptiven Feldern der Netzhaut für das Kontrastsehen verantwortlich sind.

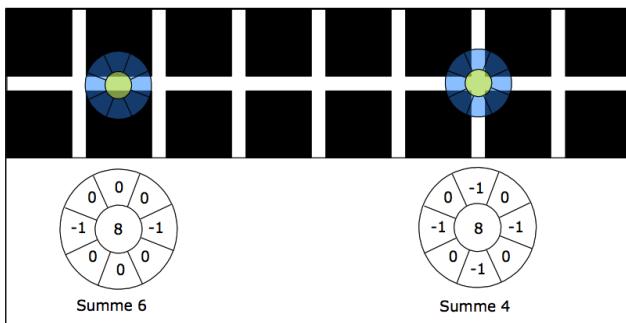


Abb. 115: Rezeptive Felder in der Netzhaut entsprechen einem Laplace-Filter.

Abb. 116: Laplace Filter in Excel

6.1.3.3 Laplace of Gaussian Filter

Der Laplace-Filter ist aber anfällig auf Rauschen im Bild, da er ja auch ein Hochpassfilter ist. Um dieses Problem zu reduzieren, hat man ihn mit dem Gaussfilter kombiniert. Man nimmt die Laplace-Funktion von der Gaussfunktion, was der 2. Ableitung von der Gaussfunktion entspricht. Deshalb wird dieser Filter manchmal auch als *Laplace of a Gaussian (LoG)* bezeichnet.

$$G(x) = e^{-\frac{x^2}{2\sigma^2}}$$

Gaussfunktion mit σ als Standardabweichung

$$G'(x) = \frac{-1}{\sigma^2} x \cdot e^{-\frac{x^2}{2\sigma^2}}$$

1. Ableitung der Gaussfunktion

$$G''(x) = \frac{1}{\sigma^4} \left(\frac{x^2}{\sigma^2} - 1 \right) \cdot e^{-\frac{x^2}{2\sigma^2}}$$

2. Ableitung der Gaussfunktion

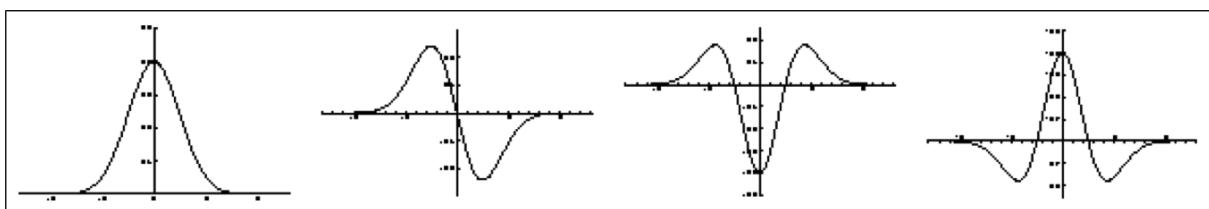


Abb. 117: Gaussfunktion,

1. Ableitung,

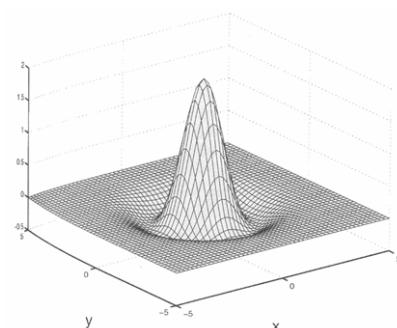
2. Ableitung (LoG)

2. Ableitung invertiert

Als wirkungsvolle Filtermaske muss mindestens eine 5×5 Matrix gewählt werden. In der Praxis werden oft Masken der Grösse 11×11 oder noch grösser gewählt:

$$L_{LoG3x3} = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix} \quad L_{LoG5x5} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$L_{LoG13x13} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -2 & -2 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & -2 & -3 & -3 & -4 & -3 & -3 & -2 & -2 & 0 & 0 \\ 0 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -3 & -3 & -2 & -1 & 0 \\ 0 & -1 & -3 & -3 & -1 & 4 & 6 & 4 & -1 & -3 & -3 & -1 & 0 \\ -1 & -2 & -3 & -3 & 4 & 14 & 19 & 14 & 4 & -3 & -3 & -2 & -1 \\ -1 & -2 & -4 & -2 & 6 & 19 & 24 & 19 & 6 & -2 & -4 & -2 & -1 \\ -1 & -2 & -3 & -3 & 4 & 14 & 19 & 14 & 4 & -3 & -3 & -2 & -1 \\ 0 & -1 & -3 & -3 & -1 & 4 & 6 & 4 & -1 & -3 & -3 & -1 & 0 \\ 0 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -3 & -2 & -1 & 0 & 0 \\ 0 & 0 & -2 & -2 & -3 & -3 & -4 & -3 & -3 & -2 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -2 & -2 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Wegen seiner Form wird der invertierte LoG-Operator auch *Mexican-Hat* genannt.

6.1.4 Punktspreizfunktion (PSF)

Zu guter Letzt sei noch auf eine spezielle Filtermaske hingewiesen. Man nennt sie auch *Punktspreizfunktion* (*Point Spread Function*) und sie ist eine Faltungsmaske mit überall Nullen und einer 1 im Zentrum. Diese Funktion wird auch Impulsantwort- oder *Dirac-Funktion* genannt.

$$PSF = \delta(i, j) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

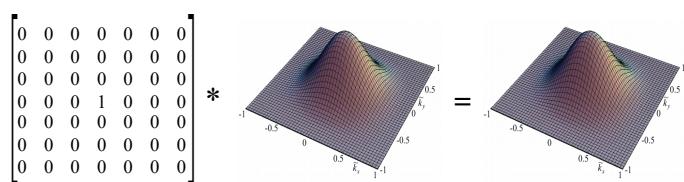
Faltet man ein Bild g mit einer PSF, so resultiert das identische Bild g :

$$g * \delta = \delta * g = g$$



Diese Unveränderbarkeit trifft natürlich auch auf eine Faltung einer PSF mit einer Faltungsmaske h zu:

$$h * \delta = \delta * h = h$$



Dies ist dann von Nutzen, wenn wir eine Faltungsmaske gar nicht kennen. Wenn wir uns die Verunschärfung vorstellen, welche durch ein Objektiv verursacht wird, so können wir diese durch eine Faltung modellieren. Lassen wir einen unendlich dünnen Lichtstrahl durch ein Objektiv, so entsteht dabei die Impulsantwort.



Abb. 118: Ein unendlich dünner Lichtimpuls durch eine Optik ergibt deren PSF auf dem Sensor. Anhand dieser PSF ermöglicht eine inverse Filterung (s. Kapitel 7.1.5.2) noch stärkere Verunschärfungen zu korrigieren.

6.1.5 Anwendung Bildschärfung

Die Bildschärfung ist eine spezielle Anwendung des Hochpassfilters. Man erstellt zuerst eine Kantendetektion mit dem LoG-Filter und subtrahiert diese dann vom Ausgangsbild hinzu. Mathematisch entspricht dies einer Subtraktion der 2. Ableitung von seiner Funktion. In der chemischen Fotografie entspricht dies einem einfachen Verfahren namens [Unscharfmaskierung](#). Will man dort ein unscharfes Dia schärfen, so erstellt man davon ein noch unschärferes Negativ, legt die beiden übereinander und erhält so eine scharfe Vergrößerung.

$$G_{SharpLoG} = b(x, y) - b(x, y) * L_{\log}$$

$$G_{SharpUSM} = b(x, y) + (b(x, y) - u(x, y))$$

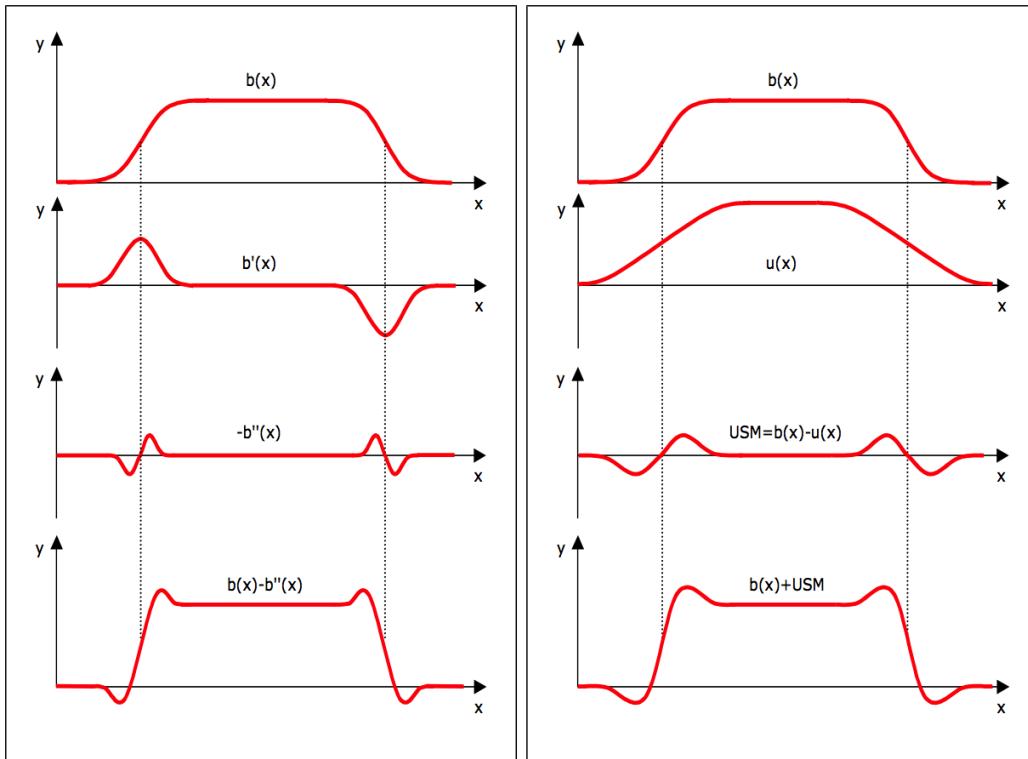


Abb. 119: Links Bildschärfung als Subtraktion der 2. Ableitung oder rechts als Unscharfmaskierung

Bei der Schärfung mit dem Laplace-Filter können wir die Subtraktion (gemäßes Distributivgesetz) direkt in eine Schärfungsfiltermaske integrieren, indem wir eine Laplace-Maske von einem Impulsfilter abziehen:

$$G_{Sharp} = g * i - g * h = g * (i - h)$$

Die 3 nachfolgenden Masken erzeugen jeweils eine schwache, mittlere und starke Schärfung:

$$G_{Sharp1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$G_{Sharp2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$G_{Sharp3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

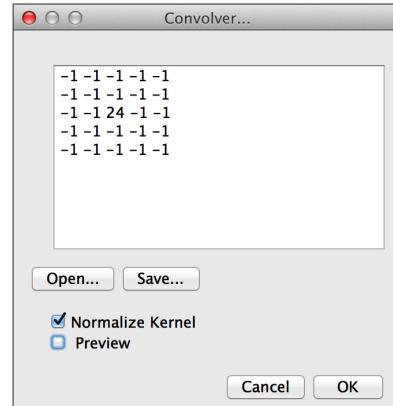


Abb. 120: v.l.n.r.: Original sowie Schärfungen mit den 3 Schärfungsfiltermasken $G_{\text{Sharp}1}$ - $G_{\text{Sharp}3}$.

6.1.6 Faltung in ImageJ

Wie die meisten Bildverarbeitungsprogramme bietet auch ImageJ einen eigenen Dialog unter dem Menü *Process > Filter > Convolve*, um eine beliebige Faltungsmaske zu definieren. Im selben Untermenü finden Sie einen Dialog für einen Gaussfilter mit einstellbarem Radius.

Möchte man eine Faltung in einem eigenen Plugin einsetzen, so kann man diese natürlich ausprogrammieren. Sie finden dazu in [Burger06] ein Beispiel, das allerdings nur zum didaktischen Verständnis gedacht ist. Da die Faltung bei grossen Bildern und insbesondere bei noch zusätzlich grossen Faltungsmasken eine teure Operation ist, sollte auf eine optimale Implementation zurückgegriffen werden.



In ImageJ gibt es dazu die Klasse *Convolver*:

```
import ij.plugin.filter.Convolver;
public void run(ImageProcessor I)
{
    float[] H = {0.075f, 0.125f, 0.075f,
                 0.125f, 0.200f, 0.125f,
                 0.075f, 0.125f, 0.075f};
    Convolver cv = new Convolver();
    cv.setNormalize(false); // turn off filter normalization
    cv.convolve(I, H, 3, 3); // do the filter operation
}
```

Um einen Gaussfilter zu kreieren, gibt es die Klasse *GaussianBlur*:

```
import ij.plugin.filter.GaussianBlur;
public void run(ImageProcessor I)
{
    GaussianBlur gb = new GaussianBlur();
    gb.blur(I, 2.5);
}
```

6.1.7 Übung: Lokale Operationen mit Matlab



Studieren Sie das Matlab-Skript *LokaleOperationen.m*:

```

clear all; close all; clc;           %clear matrices, close figures & clear cmd
wnd.

X = imread('../images/Blood2.bmp'); %Load image into matrix X
I = im2single(X);                 %Convert to single floats [0-1]
imshow(I); title('Original');
pause;

Imin = min(I(:));                %Calculate min of matrix I
Imax = max(I(:));                %Calculate max of matrix I

I2 = imadjust(I,[Imin Imax],[0 1]);
imshow(I2); title('After contrast spread'); pause;

h = [1 1 1 1 1;
      1 1 1 1 1;
      1 1 1 1 1;
      1 1 1 1 1;
      1 1 1 1 1];
h

I4 = conv2(I3,h);
imshow(I4); title('After average box filter');
pause;

imshow(I2); title('Original');
pause;
h = [1 4 6 4 1;
      4 16 24 16 4;
      6 24 32 24 4;
      4 16 24 16 4;
      1 4 6 4 1] / 256;
h
I3 = conv2(I2,h);
imshow(I3); title('After gaussian filter');
pause;

imshow(I2); title('Original');
pause;

hx = [-1 0 1;
       -2 0 2;
       -1 0 1];
hx
I3 = conv2(I2,hx);
imshow(I3); title('After X-sobel filter');
pause;

hy = hx';          % transpose matrix
I4 = conv2(I2,hy);
imshow(I4); title('After Y-sobel filter');
pause;

imshow(I2); title('Original');
pause;
h = fspecial ('log');
h
I3 = conv2(I2,h);
imshow (I3); title('After LoG-Filter');

```

Aufgaben:

- Schlagen Sie die beiden Befehle *conv2* und *imfilter* in der Matlab-Dokumentation nach. Was sind die Unterschiede? Welche Lösungen für das Randproblem werden angeboten?
- Welche Filter können mit der Filterkreationsfunktion *fspecial* erzeugt werden?
- Suchen & lösen Sie folgende Aufgaben im Skript:
 - Step 1: Create horizontal Sobel filter matrix hx

- Step 2: Convolve image I2 with hx and show the result
- Step 3: Create vertical Sobel filter matrix hy by transposing the horizontal one
- Step 4: Convolve image I2 with hy and show the result
- Step 5: Combine the the horizontal & vertical sobel to one image and show it

6.2 Morphologische Operatoren

Die *Morphologie* (aus dem Griechischen: die Gestalt betreffend) ist die Lehre der Formen und ist in vielen Wissenschaften ein Teilgebiet. In der Bildverarbeitung können morphologische Operationen auf Binär- und Graustufenbilder angewendet werden und sind für vielfältig Bildverarbeitungsaufgaben wie Kantenextraktion, Skelettierung oder Segmentierung geeignet.

Die *mathematische Morphologie* wurde durch die Franzosen *Georges Matheron* und *Jean Serra* in den 70er-Jahren begründet. In den Standardwerken [Serr82] und [Serr88] wird die mathematische Morphologie sehr allgemein und mit anspruchsvollen Notationen aus der Mengenlehre formuliert.

6.2.1 Morphologische Operatoren auf Binärbildern

Dabei wird ein lokales *Strukturelement* (SE) pixelweise mit einem Quellbild verrechnet. Das Strukturelement hat immer einen Ankerpunkt (Hotspot). Strukturelemente müssen nicht unbedingt quadratisch sein, noch muss der Ankerpunkt im Zentrum sein:

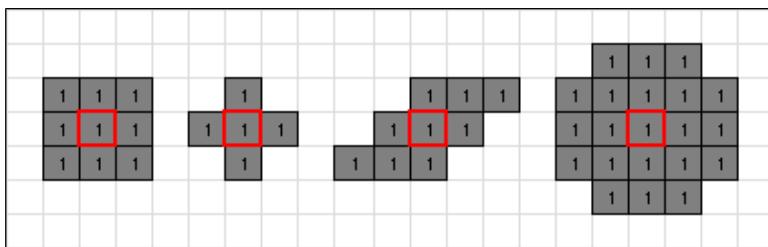


Abb. 121: Typische Strukturelemente mit ihrem Ankerpunkt. Das SE ganz links wird meistens 8er- und das 2. 4er-Nachbarschatsfs-SE genannt.

Die beiden wichtigsten morphologischen Basisoperationen sind die *Dilatation* und die *Erosion*. Die Kombination von diesen beiden wird je nach Reihenfolge *Opening* und *Closing* genannt.

6.2.1.1 Binäre Dilatation

Die Dilatation, die im Englischen meistens *dilation* und nicht *dilatation* heißt, kann bei binären Bildern auf 3 Arten definiert werden:

- **Mengenoperation:** Ist die Schnittmenge aller Zellen des Strukturelements ungleich der leeren Menge aller überlagerten Pixel im Quellbild, so resultiert ein gesetztes Pixel im Zielbild:

$$B_{dilation} = B \oplus S = \{ B \cap S \neq \emptyset \text{ für alle Pos. } S \text{ in } B \}$$

- **Logische Operation:** Das Zielpixel entspricht der ODER-Verknüpfung aller Pixel im Quellbild unter dem Strukturelement.
- **Rangordnungsoperation:** Das Zielpixel entspricht dem Maximum aller Pixel im Quellbild unter dem Strukturelement.

Im resultierenden Bild sind alle Strukturen aufgeblättert und kleine Löcher verschwunden:

Abb. 122: Quellbild B links mit Strukturelement S in der Mitte und nach der Dilatation rechts. Für roten Ausschnitt oben links resultiert eine 1, weil nur ein Pixel im Quellbild gesetzt ist. Im Ausschnitt unten rechts ergibt sich eine 0, weil kein einziges Pixel im Quellbild gesetzt ist ($B(x,y) \cap S = \emptyset$).

6.2.1.2 Binäre Erosion

Die Erosion bei binären Bildern kann auf 3 Arten definiert werden:

- **Mengenoperation:** Ist die Schnittmenge aller Zellen des Strukturelements gleich der Menge aller überlagerten Pixel im Quellbild, so resultiert ein gesetztes Pixel im Zielbild:

$$B_{erosion} = B \ominus S = \{ B \cap S = S \text{ für alle Pos. } S \text{ in } B \}$$

- **Logische Operation:** Das Zielpixel entspricht der UND-Verknüpfung aller Pixel im Quellbild unter dem Strukturelement.
 - **Rangordnungsoperation:** Das Zielpixel entspricht dem Minimum aller Pixel im Quellbild unter dem Strukturelement.

Im resultierenden Bild sind alle Strukturen geschrumpft. Kleine Strukturen verschwinden ganz:

The diagram illustrates the erosion operation. On the left, a 3x3 kernel labeled Θ is applied to a 10x10 input matrix. The input matrix has a central gray region. The kernel slides over the input, and the result is shown on the right. The result matrix has a red box highlighting the output of the central element of the kernel.

Abb. 123: Quellbild links mit Strukturelement in der Mitte und nach der Erosion rechts. Im roten Ausschnitt oben links resultiert eine 1, weil alle überlappten Pixel im Quellbild gesetzt sind. Im Ausschnitt unten rechts ergibt sich eine 0, weil nicht alle überlappten Pixel gesetzt sind.

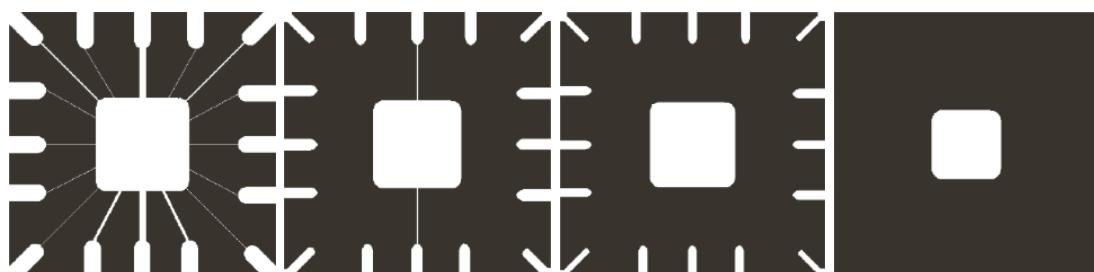


Abb. 124: v.l.n.r.: Originalbild 486x486 Pixel, Erosion mit 11x11, Erosion mit 15x15 und Erosion mit 45x45 (Bildquelle: [Gonzales08])

6.2.1.3 Eigenschaften von Dilatation und Erosion

- **Dilatation und Erosion sind unumkehrbar.** Ein durch Erosion geschrumpftes Bild kann nicht durch Dilatation zurückgewonnen werden. Das Umgekehrte gilt ebenso.
- **Dilatation und Erosion sind dual zueinander:** D. h., dass eine Dilatation des Vordergrunds durch eine Erosion des Hintergrunds durchgeführt werden kann. Das Gleiche gilt auch umgekehrt: $\overline{B} \oplus S = (\overline{B} \ominus S)$ und $\overline{B} \ominus S = (\overline{B} \oplus S)$
- **Dilatation ist kommutativ:** $B \oplus S = S \oplus B$: Dadurch können wie bei der Faltung Bild und Maske ausgetauscht werden.
- **Dilatation ist assoziativ:** $(B \oplus S_1) \oplus S_2 = B \oplus (S_1 \oplus S_2)$: D. h. die Reihenfolge von aufeinanderfolgenden Dilatationen ist nicht relevant.
- **Erosion ist nicht kommutativ:** $B \ominus S \neq S \ominus B$

6.2.1.4 Binäres Closing

Die Closing-Operation besteht aus einer Dilatation gefolgt von einer Erosion. Für asymmetrische Strukturelemente muss die Erosion mit gespiegeltem Strukturelement S' durchgeführt werden, um die Verschiebung durch die Dilatation wieder rückgängig zu machen:

$$B_{Closing} = B \bullet S = (B \oplus S) \ominus S'$$

Die obigen beiden Beispiele zusammen sind eine Closing-Operation. Durch diese Methode werden Löcher geschlossen und konkave Ränder aufgefüllt. Es können Brücken zu benachbarten Strukturen entstehen.

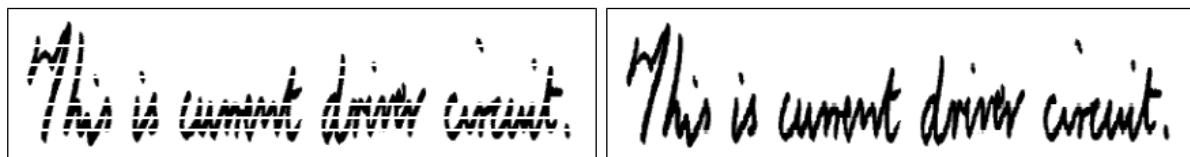


Abb. 125: Closing Beispiel mit 1x5 Pixel grossen SE. Links: Gestörtes Original und rechts nach Closing.

6.2.1.5 Binäres Opening

Die Opening-Operation besteht aus einer Erosion gefolgt von einer Dilatation mit gespiegeltem Strukturelement S' :

$$B_{Opening} = B \circ S = (B \ominus S) \oplus S'$$

Durch diese Methode verschwinden kleine Strukturen ganz und konvexe Ränder werden abgetragen.

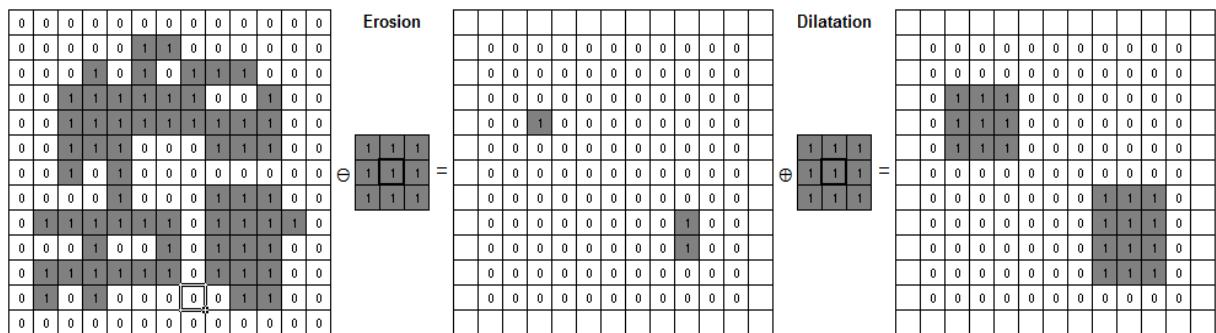


Abb. 126: Opening-Operation als Erosion gefolgt von einer Dilatation

Nachfolgend ein Beispiel aus [Gonzales08] mit einer Abfolge von Erosionen und Dilatation mit einem 8er-Nachbarschafts-SE die zusammengefasst eine

$$B' = (((B \ominus S_8) \oplus S_8) \oplus S_8) \ominus S_8 = (B \circ S_8) \bullet S_8$$

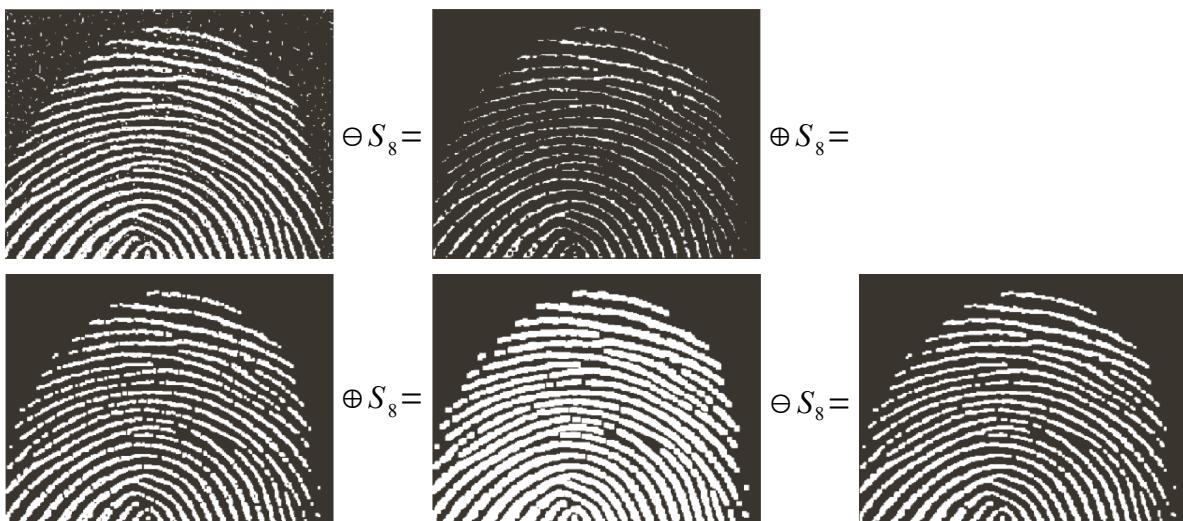


Abb. 127: Rauschentfernung durch ein Opening gefolgt von einem Closing (Quelle: [Gonzales08])

6.2.1.6 Eigenschaften von Opening und Closing

- **Opening und Closing sind final:** Beide Operationen sind *final* im Sinne, dass eine weitere Anwendung derselben Operation keine Auswirkung mehr hat:

$$(B \circ S) \circ S = B \circ S \text{ und } (B \bullet S) \bullet S = B \bullet S$$
 - **Opening und Closing sind dual:** Die beiden Operationen sind zueinander *dual* im Sinn, dass ein Opening auf den Vordergrund gleich ist wie ein Closing des Hintergrunds und umgekehrt.

$$B \circ S \equiv \overline{\overline{B} \bullet S} \quad \text{und} \quad B \bullet S \equiv \overline{\overline{B} \circ S}$$

6.2.1.7 Binäre Kantenextraktion

Eine klassische Anwendung der Morphologie ist die Kantenextraktion durch Differenz mit dem Original. Durch Erosion mit nachfolgender Differenz zum Original erhält man die internen Kantenpixel und durch Dilatation mit nachfolgender Differenz (\setminus) die externen Kantenpixel. Eine dicke Kante aus internen und externen Kanten erhält man durch Subtraktion der Erosion von der Dilatation.

$$B_{\text{edge, internal}} = (B \ominus S) \setminus B = B - (B \ominus S)$$

$$B_{\text{adjacent}} = (B \oplus S) \setminus B = (B \oplus S) - B$$

$$B_{-1-d+1} \equiv (B \oplus S) - (B \ominus S)$$

Nachfolgend zwei Beispiele mit dem 8er-Nachbarschafts-SE. Man beachte, dass die Kantenpixel immer über die Pixelseiten verbunden sind:

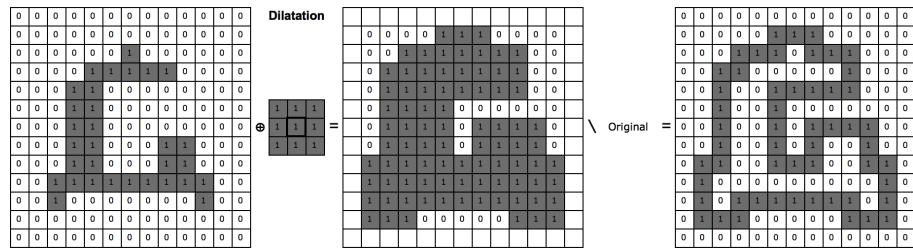


Abb. 128: Open innenseitige und unten aussenseitige Kantenextraktion mit 8er-Dilatation und nachfolgender Differenz vom Original.

Und jetzt zwei Beispiele mit dem 4er-Nachbarschafts-SE. Die Kanten können nun auch über die Pixelecken verbunden sein:

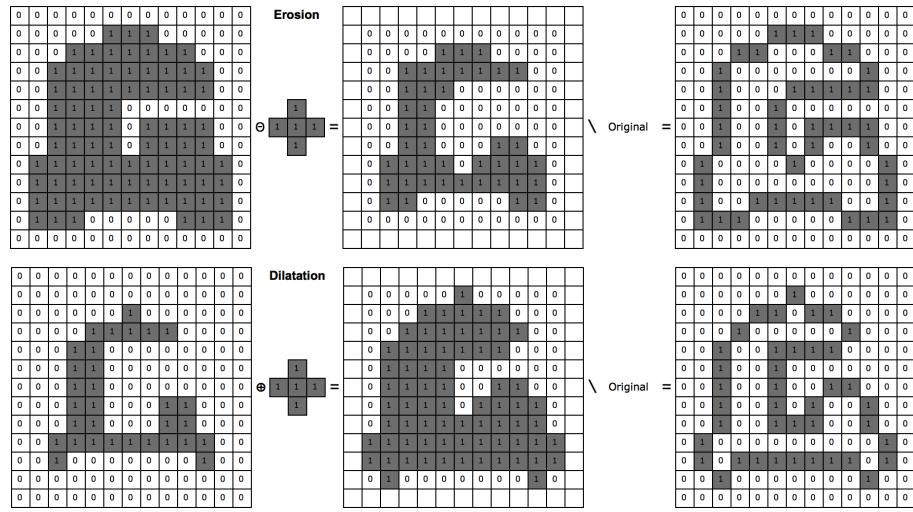


Abb. 129: Oben innenseitige und unten aussenseitige Kantenextraktion mit 4er-Dilatation und nachfolgender Differenz vom Original.

6.2.1.8 Hit-or-Miss-Operator

Mit dem Hit-or-Miss-Operator können wir nach ganz bestimmten Pixelmustern suchen. Er kann durch die beiden Schritte Hit und Miss oder nur durch den Hit-Schritt definiert werden. Im Hit-Schritt wird durch Erosion nach dem Pixelmuster des SE gesucht. Im nachfolgenden Beispiel wird nach zwei nebeneinanderliegenden Vordergrundspixeln (1) gesucht.

Im Miss-Schritt wird darauf im invertierten Bild B^C mit einem SE erodiert, das der Umgebung des Hit-SE's entspricht. Die beiden resultierenden Bilder werden dann UND-verknüpft und ergeben so die Positionen der gesuchten Struktur.

Der Hit-or-Miss-Operator wird mit einem Stern im Kreis symbolisiert:

$$B \circledast S = (B \ominus S_{Hit}) \cap (B^C \ominus S_{Miss}) \quad \text{mit } B^C = \bar{B}$$

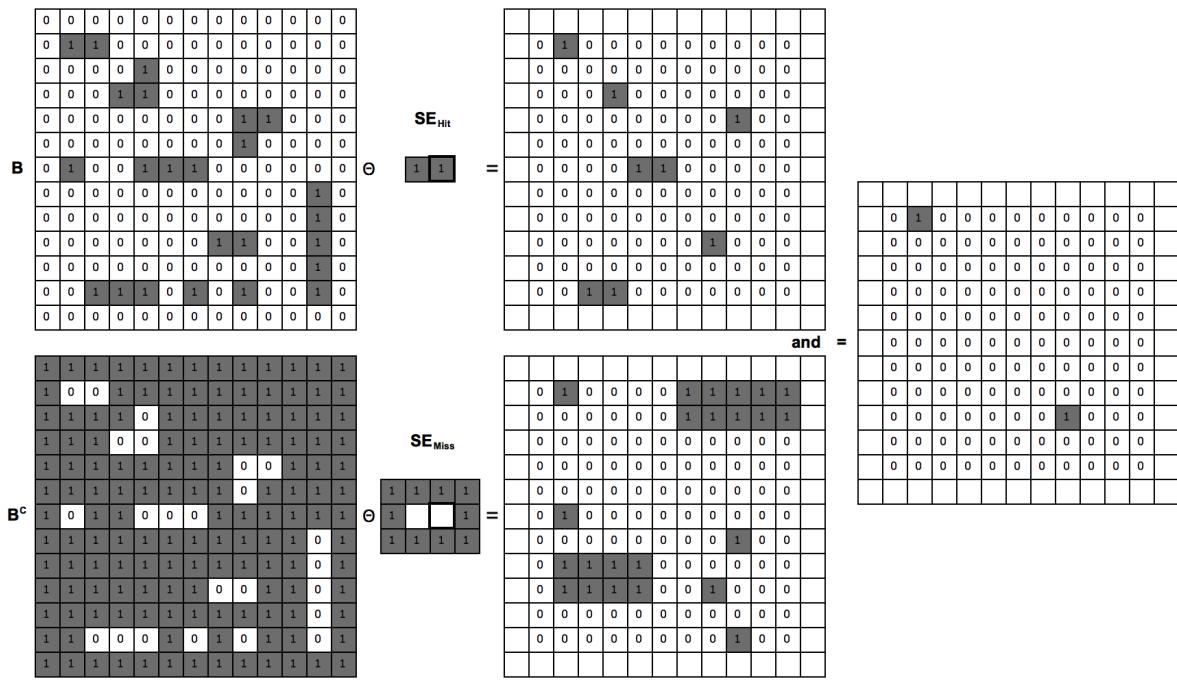


Abb. 130: Hit-or-Miss-Operator in zwei Schritten

Wenn wir SE ermöglichen, die nicht rechteckig sind und auch Nullen enthalten können, so können wir den Hit-or-Miss-Operator auf einen Hit-Operator reduzieren. Durch eine UND-Verknüpfung von mehreren Hit-Operatoren mit jeweils verschiedenen SE können wir z. B. alle Ecken in einem Bild identifizieren.

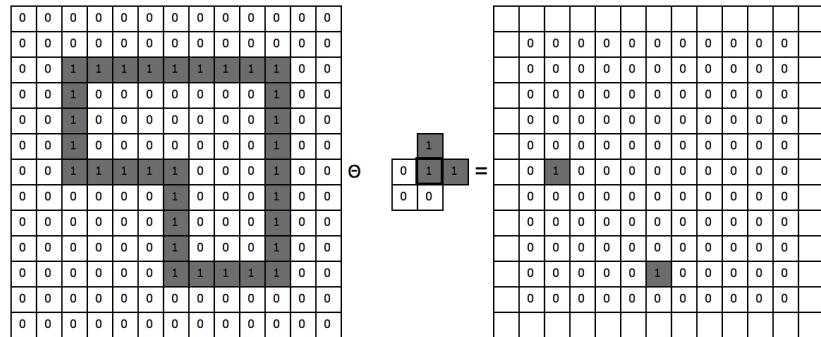


Abb. 131: Hit-Operator mit Nullen und Einsen zum Suchen von Links-Unten-Ecken

6.2.1.9 Schnelle Implementation durch Bildverschiebung

Eine schnellere Implementation der morphologischen Operationen als durch Verschieben des Strukturelementes ergibt sich durch Verschieben des Quellbildes an alle Positionen des Strukturelementes. Kombiniert man danach alle Verschiebungen des Quellbildes mit einem logischen ODER, so ergibt sich eine Dilatation und mit einem logischen UND eine Erosion.

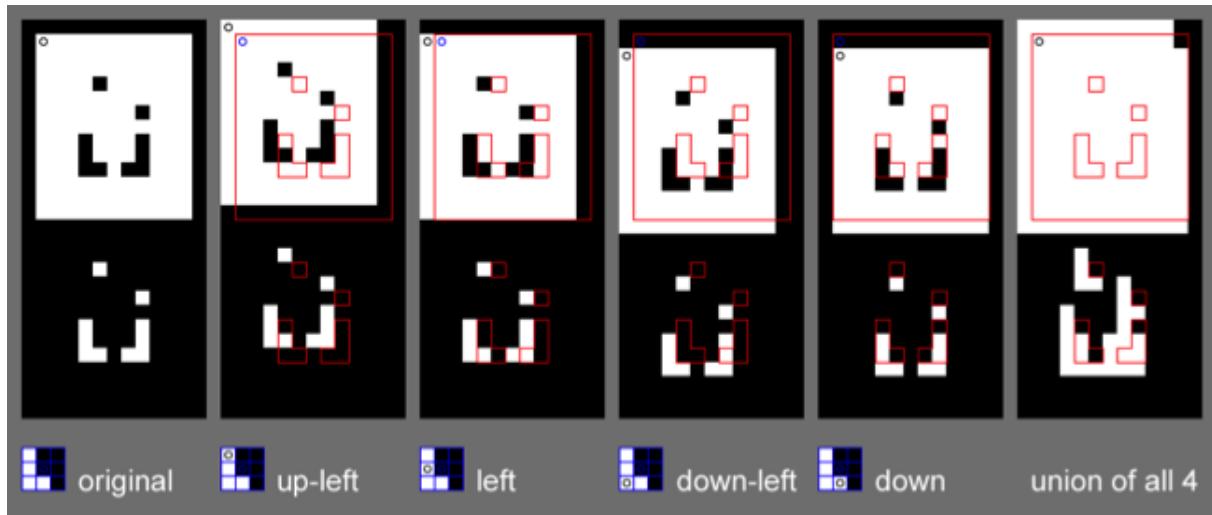


Abb. 132: Vereinigung aller verschobenen Quellbilder mit einem logischen ODER ergeben eine Dilatation. (Bildquelle: [Peters07])

6.2.2 Morphologische Operatoren auf Graustufenbildern

Bei der Morphologie mit Graustufenbildern bestehen die Strukturelemente nicht nur aus einer Menge an gesetzten Pixeln, sondern sie enthalten reelle Zahlen, die auch null sein können. Graustufenbilder lassen sich deshalb auch als 3D-Strukturen darstellen:

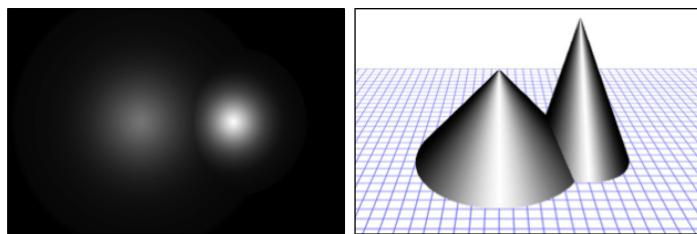


Abb. 133: Ein Graustufenbild links als 3D-Visualisierung rechts (Bildquelle: [Peters07])

Analog lassen sich morphologische Operationen mit Graustufenbildern besser mit Mengenoperationen im 3D-Raum darstellen. Ob eine Menge Teilmenge einer andern ist, hängt somit explizit von der Höhe beider Mengen ab:

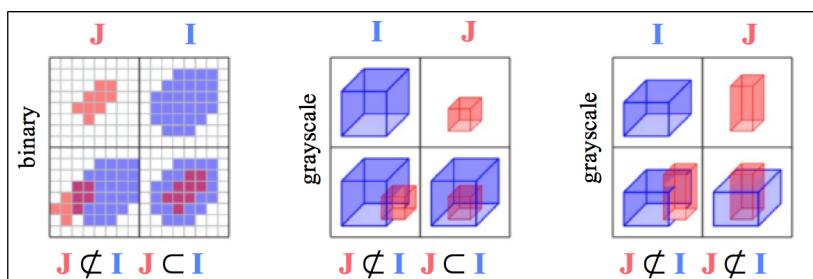


Abb. 134: Ob J Teilmenge von der Menge I ist, hängt von der Höhe von J ab (Bildquelle: [Peters07]).

Strukturelemente enthalten nun reelle Zahlen und müssen wiederum nicht rechteckig und auch nicht symmetrisch sein:

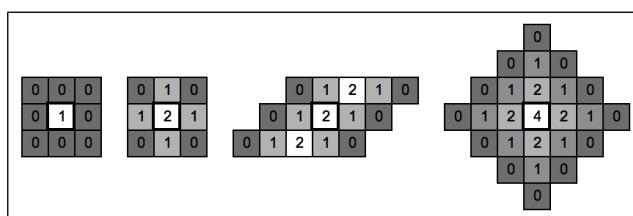


Abb. 135: Strukturelement mit reellen Zahlenwerten für morphologische Operationen in Graustufenbildern.

Die Dilatation und die Erosion werden nur noch mit einer *Rangordnungsoperation* definiert: Bei der Dilatation addiert man die entsprechenden Pixelwerte des Strukturelements $S(i,j)$ zu den Quellpixeln hinzu und nimmt von allen Resultaten das Maximum für das Zielpixel. Bei der Erosion subtrahiert man die entsprechenden Pixelwerte des Strukturelements zu den Quellpixeln hinzu und nimmt von allen Resultaten das Minimum für das Zielpixel. Natürlich muss man mit einem *Clamping* gewährleisten, dass die Wertebereiche des Pixels eingehalten werden.

Dilatation: $G'(x, y) = G(x, y) \oplus S(i, j) = \max(G(x+i, y+j) + S(i, j))$

Erosion: $G'(x, y) = G(x, y) \ominus S(i, j) = \min(G(x+i, y+j) - S(i, j))$

Sind alle Werte im Strukturelement null, so gelten diese Formulierungen auch für binäre morphologische Operationen.

Sind die Werte im Strukturelement nicht null, so kann es sein, dass neue Graustufen hinzukommen, die vorher nicht im Bild vorhanden waren.

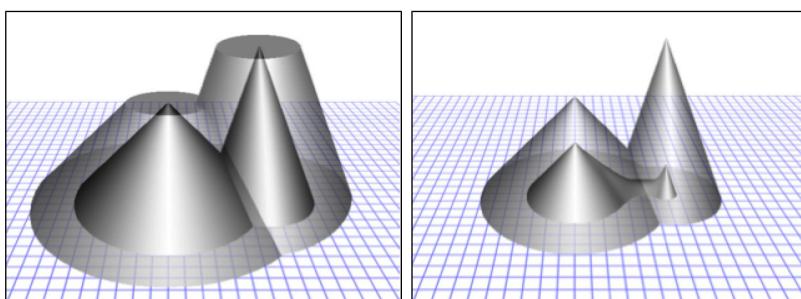


Abb. 136: Bei einem runden Maximumfilter (links) wird die Helligkeit auf die höchste Helligkeit unter der Maske angehoben. Bei einem runden Minimumfilter (rechts) wird die Helligkeit auf die tiefste Helligkeit unter der Maske abgesenkt (Bildquelle: [Peters07]).

6.2.2.1 Maximum- und Minimumfilter

Rangordnungsoperatoren, wie diese Gruppe auch noch genannt wird, bringen die Pixel unterhalb des Strukturelements in eine geordnete Reihe und ermitteln daraus einen neuen Grauwert. Sind alle Werte im Strukturelement null und ist das Strukturelement symmetrisch, so ist die Graustufen-Dilatation ein Maximumfilter und die Graustufen-Erosion entspricht einem Minimumfilter:

$$\text{Dilatation = Maximumfilter: } \quad G'(x, y) \equiv G(x, y) \oplus S(i, j) \equiv \max(G(x+i, y+j))$$

Erosion = Minimumfilter: $G'(x, y) = G(x, y) \ominus S(i, j) = \min(G(x+i, y+j))$

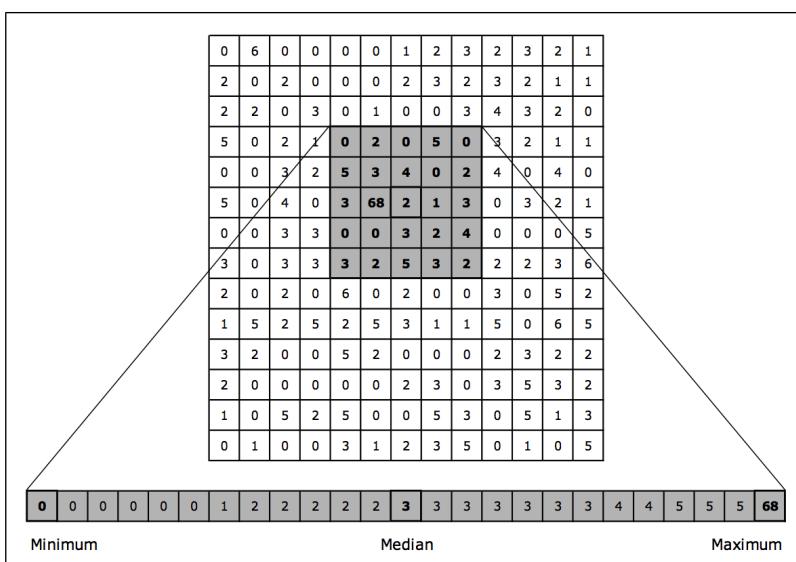


Abb. 137: Filterbereich unter dem SE und sortiert nach Grösse.



Abb. 138: v.l.n.r.: Originalbild, Filterung mit einem 3x3-Maximumfilter (Dilatation), Filterung mit einem 3x3-Minimumfilter (Erosion).

6.2.2.2 Medianfilter

Der Medianfilter ist ebenfalls ein Rangordnungsoperator, der aber nicht den max. oder min. Pixelwert unter dem Strukturelement verwendet, sondern den Wert in der Mitte des sortierten Wertebereichs (s. Abb. 137).

Der Medianwert ist interessant, weil er im Gegensatz zum Durchschnittswert beim Rechteck- und Gaussfilter, nicht durch statistische Ausreisser beeinflusst wird. In einem Bild können damit einzelne Ausreisserpixel, auch *Salt and Pepper* genannt, herausgefiltert werden, ohne dass eine generelle Weichzeichnung erfolgt.

Bei natürlichen Rauschen, welches das Bild mit einer Gauss-Verteilung überlagert, wirkt der Median immer noch Kanten erhaltend aber nicht mehr so perfekt wie bei Salt & Pepper-Rauschen.

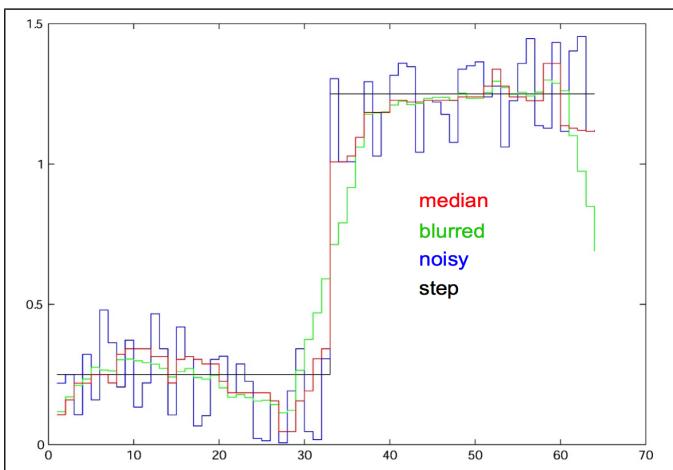


Abb. 139: Eine verrauschte (blau) S/W-Kante (schwarz) bleibt mit Median gefiltert (rot) relativ gut erhalten, während sie mit Gauss gefiltert abgeschwächt wird (grün). (Bildquelle: [Peters07])

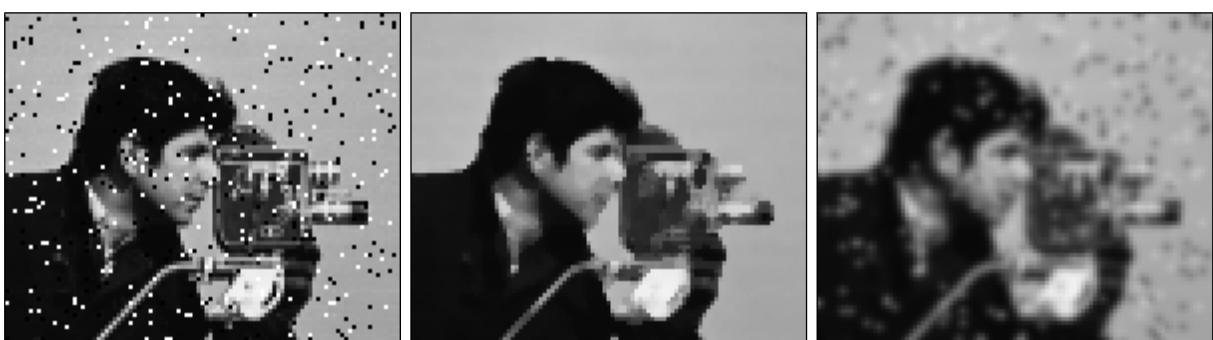


Abb. 140: v.l.n.r.: Mit Salt & Pepper-Rauschen, mit einem 3x3-Medianfilter, mit einem 3x3-Gaussfilter.



Abb. 141: v.l.n.r.: Mit Gauss-Rauschen mit einem Sigma von 10.0, einmal und zweimal mit 3x3-Medianfilter.



Abb. 142: v.l.n.r.: Mit Gauss-Rauschen mit einem Sigma von 10.0, einmal und zweimal mit 3x3-Gaussfilter.

6.2.2.3 Closing und Opening auf Graustufenbildern

Wie bei binären Operationen sind eine Abfolge von Dilatation und Erosion als Closing und eine Erosion gefolgt von einer Dilatation als Opening definiert:

$$G_{\text{Closing}} = G \bullet S = (G \oplus S) \ominus S'$$

$$G_{\text{Opening}} = G \circ S = (G \ominus S) \oplus S'$$

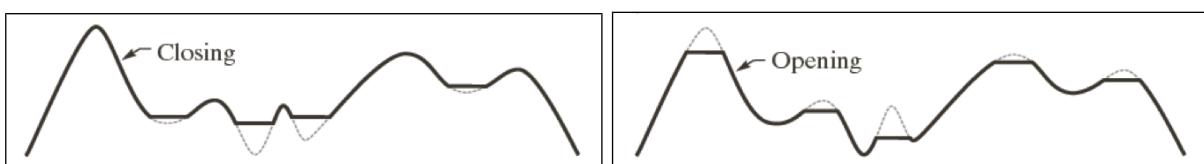


Abb. 143: Graustufenquerschnitt der Closing- (links) und Opening-Operation (Bildquelle: [Gonzales08]).

Die Closing-Operation belässt helle Details und lässt dunkle Details verschwinden.

Die Opening-Operation belässt dunkle Details und lässt helle Details verschwinden.



Abb. 144: v.l.n.r.: Original, nach Closing-Operation und nach Opening (Bildquelle: [Peters07])

6.2.2.4 Kantenextraktion auf Graustufenbilder

Gleich wie bei der binären Morphologie ist die Kantenextraktion durch Differenzbildung möglich. Durch Erosion mit nachfolgender Differenz zum Original erhält man die internen Kantenpixel und durch Dilatation mit nachfolgender Subtraktion des Originals die externen Kantenpixel. Eine dicke Kante aus internen und externen Kanten erhält man durch Subtraktion der Erosion von der Dilatation. Die Kantenstärke hängt ebenfalls vom Durchmesser des SE ab.

Im Gegensatz zu den Differenzoperatoren, die grundsätzlich anfällig sind auf Rauschen, eignen sich die morphologischen Filter gut zur Kantenextraktion, weil sowohl die Dilatation als auch die Erosion glättend wirken. Noch etwas rauscharmer wirkt das Minimum aus internen und externen Kanten.

$$G_{\text{edge intern}} = G - (G \ominus S)$$

$$G_{\text{edge extern}} = (G \oplus S) - G$$

$$G_{\text{edge thick}} = (G \oplus S) - (G \ominus S)$$

$$G_{\text{edge thin}} = \min(G_{\text{edge intern}}, G_{\text{edge extern}})$$

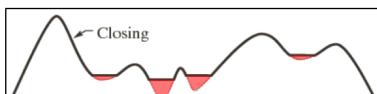


Abb. 145: Kantenextraktion aus Dilatation minus Erosion, jeweils mit dem 4er-Nachbarschaftselement.

6.2.2.5 Bot-Hat- und Top-Hat-Operation

Wie bei den binären Operationen kann man auch Differenzen bilden zwischen dem Closing- und Opening-Resultat mit dem Original.

Die Bot-Hat-Operation (manchmal auch Black Top-Hat genannt) macht kleine Details, welche dunkler sind als ihre Umgebung, zu weiss.



$$G_{\text{Bothat}} = G_{\text{Closing}} - G = (G \bullet S) - G$$

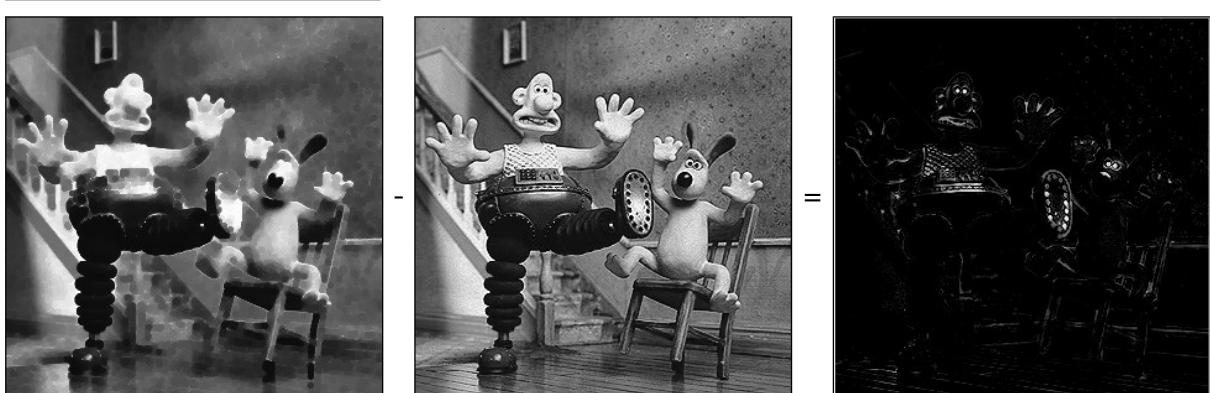
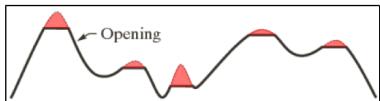


Abb. 146: Bot-hat-Operation (Bildquelle: [Peters07])

Die Top-Hat-Operation macht kleinen Details, die heller sind als ihre Umgebung, zu weiss.



$$G_{\text{Tothat}} = G - G_{\text{Opening}} = G - (G \circ S)$$

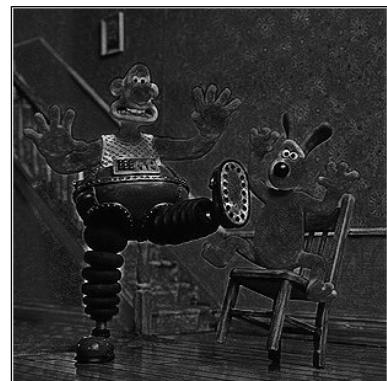


Abb. 147: Top-hat-Operation (Bildquelle: [Peters07])

6.2.3 Morphologische Operatoren in ImageJ



ImageJ bietet über das Menü *Process > Binary* die Operatoren *Erode*, *Dilate*, *Open*, *Close* und *Outline* an. Alle Operationen werden mit dem 8er-Nachbarschafts-SE durchgeführt. Fiji bietet zusätzlich im Menü *Process > Morphology > Gray Morphology* einen Dialog für Graustufen-Morphologie mit einstellbaren SE.

Zum Programmieren stellt die Klasse *ImageProcessor* einige Methoden für einfache morphologische Filter zur Verfügung: *dilate()*, *erode()*, *open()* und *close()*. Diese Methoden verwenden ebenfalls das 8er-Nachbarschafts-SE, die abhängig vom Bildinhalt entweder binäre oder Grauwert-Operationen durchführen.

Die Klasse *ColorProcessor* stellt diese Methoden auch für Farbbilder zur Verfügung, wobei die morphologischen Operationen individuell auf die einzelnen Farbkanäle angewendet werden.

6.2.4 Morphologische Operatoren in Matlab



Matlab bietet sehr generische aber auch spezifische [morphologische Funktionen](#) an. Ein Strukturelement wird mit der *strel* definiert. Sie können damit freie SE, wie auch SE mit vordefinierter Geometrie definieren.

Nachfolgend ein Beispiel (*MorphologicalOperators.m*), um mit einer Opening Operation die Bauteile von den Leiterbahnen zu trennen:

```
clear all; close all; clc; %clear matrices, close figures & clear cmd wnd.

I = imread('../images/circuit.png');           % Load image into matrix X
J = imcomplement(I);                         % Invert image
BW1 = im2bw(J,0.8);                          % make black & white
SE1 = strel('rectangle',[30 20]);             % 40x30 rectangle SE
BW2 =imerode(BW1,SE1);                      % erode w. 40x30 rectangular SE
BW3 = imdilate(BW2,SE1);                     % dilate w. the same SE

imshow(I); title('Original');                 % Show images
figure; imshow(BW2); title('After erosion');
figure; imshow(BW1); title('After binarisation');
figure; imshow(BW3); title('After dilation');
```

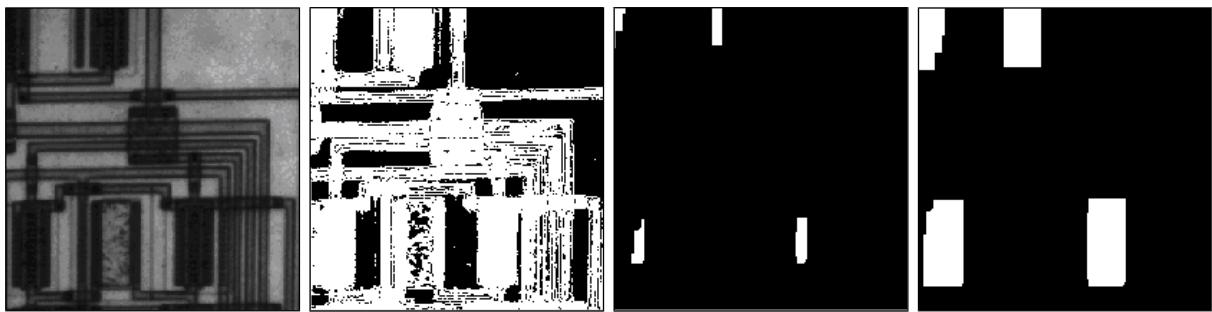


Abb. 148: Separierung der Bauteile von den Leiterbahnen.

6.2.5 Übung zu morphologischen Operationen

- Implementieren Sie die Dilatation und die Erosion in *MS Excel* oder *Open Office Calc*.
- Fügen Sie einem Bild einmal Salt & Pepper-Rauschen und einmal Gauss-Rauschen hinzu. Entrauschen Sie die Bilder mit ein- oder mehrfacher Anwendung des Medianfilters oder des Gaussfilters. Probieren Sie verschiedene Größen der Filtermasken aus.
- Optional: Implementieren Sie die Operationen des vorangegangenen Matlab-Skripts in *ImageJ*.

7 Globale Operatoren

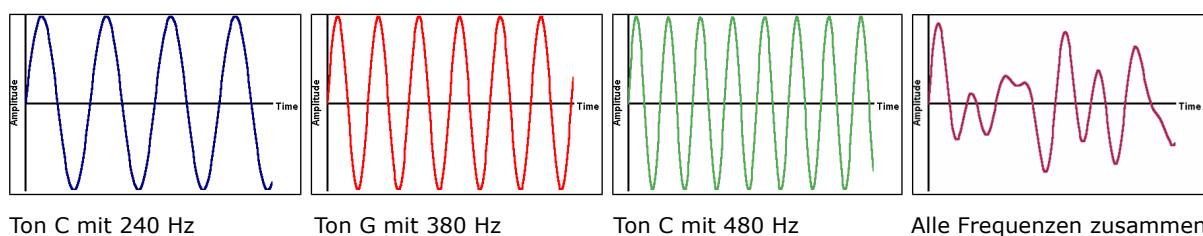


lobalen Operatoren beziehen im Unterschied zu den lokalen Operatoren die das gesamte Bild in die Berechnung ein. Jedes Signal, also auch ein digitales Bild, kann grundsätzlich auf verschiedene, ja sogar unendlich viele Arten dargestellt werden. Um es von einer Darstellung in eine andere zu verwandeln, braucht man eine *Abbildung* oder *Transformation*. Bei der Transformation wie bei der Rücktransformation geht theoretisch keine Information verloren. Sie wird nur anders dargestellt. Nur die wenigsten, der möglichen Transformationen ergeben eine Darstellung, die für uns einen Gewinn an Information oder Verarbeitungsmöglichkeiten bietet. Die beiden Schritte der Transformation und der Rekonstruktion werden auch *Analyse* und *Synthese* genannt.

Eine ihnen bekannte Transformation ist die Rotation eines Vektors in einen anderen Vektor. Ein anderes Beispiel ist die Multiplikation zweier römischer Zahlen. Will man die beiden römischen Zahlen *LXXXVI* und *XLI* multiplizieren, so transformiert man diese zuerst in arabische Zahlen, multipliziert sie ($86 \cdot 41 = 3526$) und transformiert das Ergebnis zurück in römische Zahlen: *LXXXVI * XLI = MMMDXXVI*.

7.1 Fourier-Transformation

Die Fourier-Transformation ist eine der meistverwendeten Transformationen und verwandelt ein Signal von der Darstellung im **Ortsraum** (*Space Domain oder Time Domain*) in die Darstellung im **Frequenzraum** (*Frequency Domain*). Die erste Darstellung umfasst die uns bekannte Darstellung von Farbwerten der räumlich lokalisierten Pixel oder bei Tonsignalen deren Wellendarstellung. Bei der Frequenzraumdarstellung werden die Frequenzanteile des Signals oder des Bildes dargestellt. Der Frequenzraum eines Signals kann anhand eines akustischen Signals am anschaulichsten (oder besser anhörlichsten) erklärt werden. Man stelle sich folgende 3 Töne in der Ortsraumdarstellung vor:

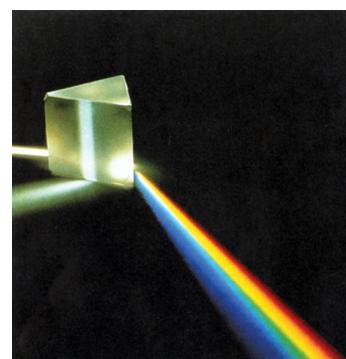


Spielt man nun alle drei Töne gleichzeitig, ergibt sich das Signal links im Ortsraum. Transformiert man dieses Signal nun in den Frequenzraum, so erhielt man die drei Zahlen 240, 380 und 480. Die Fourier-Transformation besitzt also die Fähigkeit eines guten Musikers, ein Signal in seine Frequenzanteile zu zerlegen.

Ein anderes sinnbildliches Pendant ist das Lichtprisma: Es spaltet weisses Licht in seine Spektralfarben auf und kann auch verschiedene Spektren wieder zu einem Gesamtspektrum vereinen.



Die Fourier-Transformation geht auf *Jean-Baptiste Joseph Baron de Fourier* (französischer Ägyptologe, Mathematiker und Physiker, 1768-1830) zurück. Er hat gezeigt, dass sich jede Funktion als Summe (bei periodischen Funktionen) bzw. Integrale (bei nicht periodischen Funktionen) einfacher Sinus- und Kosinus-Kurven beschreiben lässt.



7.1.1 Sinus- und Kosinusfunktionen als Kreisbewegungen

Die Funktion $\sin(\omega x)$ beschreibt eine Sinusfunktion mit der Kreisfrequenz ω an der Position x . Die periodische Funktion hat die *Kreisfrequenz* ω und damit die *Periode* oder *Wellenlänge* $T=2\pi/\omega$. Für $\omega=1$ ist die Periode $T_1=2\pi$. Für $\omega=3$ ist die Periode $T_3=2\pi/3$. Das Gleiche gilt für die Kosinusfunktion mit dem Unterschied, dass die Sinuskurve durch den Ursprung geht ($\sin(0)=0$). Die Zusammenhang zwischen Kreisfrequenz und „normaler“ Frequenz ist: $f=1/T=\omega/2\pi$.

Der Ausdruck „Kreisfrequenz“ und die Tatsache, dass wir auf der x -Achse die Winkeleinheiten angeben, verdeutlicht den oft vergessenen Hintergrund, dass diese Kurven durch *Kreisbewegungen* eines Punktes um den Kreismittelpunkt entstehen. Wie schnell der Punkt sich dreht, ist eben durch die Kreisfrequenz bestimmt. Ein weiterer Vorteil der Kreisbewegung ist, dass wir dafür ein Gefühl für die Energie haben. Jeder hat schon einen Gegenstand an einer Schnur geschwungen. Je kürzer die Schnur ist (= Amplitude), umso weniger Kraft braucht es für eine bestimmte Drehzahl.

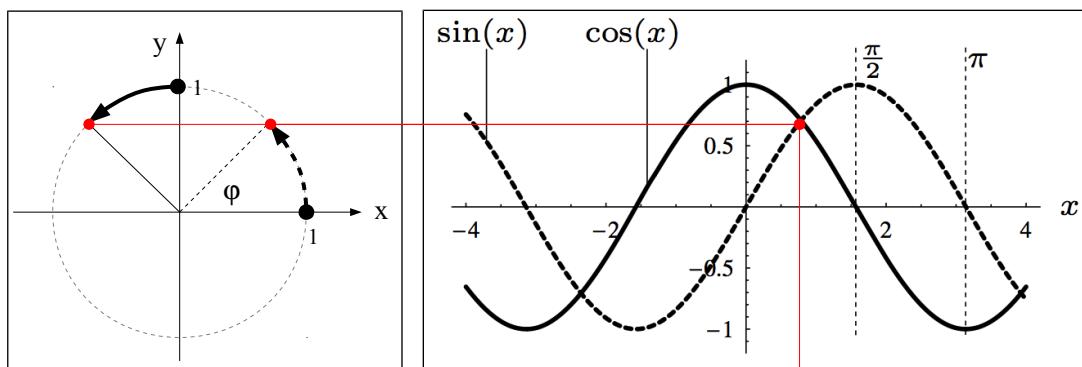


Abb. 149: Links: 1/8 Kreisbewegung der Kosinus- und Sinusdrehung. Rechts die dazugehörige Kosinus- und Sinuskurve mit der Kreisfrequenz 1.

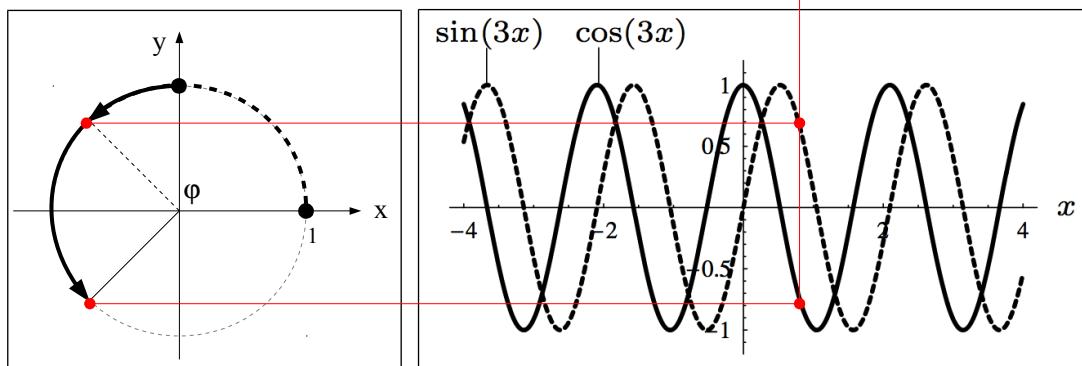


Abb. 150: In derselben Zeit drehen $\sin(3x)$ und $\cos(3x)$ dreimal schneller als $\sin(x)$ und $\cos(x)$. Rechts die Kosinus- und Sinuskurve mit der Kreisfrequenz 3.

Die Kosinus- und Sinuskurven oszillieren zwischen +1 und -1. Multipliziert man die Funktionen mit $\pm a$, so ändert man die *Amplitude* der Kurve und sie oszilliert nun zwischen $\pm a$. Die Amplitude entspricht bei der Kreisbewegung dem Drehradius.

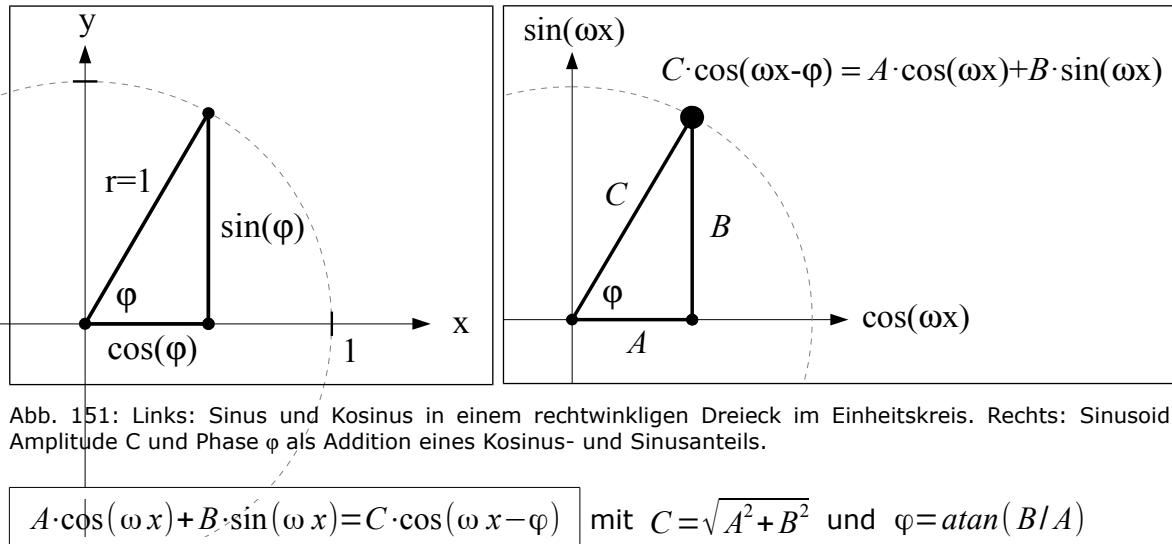
$$a \cdot \cos(\omega x) \text{ bzw. } a \cdot \sin(\omega x)$$

Um die Kurve entlang der x -Achse zu verschieben, addieren oder subtrahieren wir einen Phasenwinkel φ zu x hinzu. φ entspricht in der Kreisbewegung dem Startwinkel. Wir sehen dabei auch, dass wir die Kosinusfunktion eine um 90° ($\pi/2$) verschobene Sinusfunktion ist.

$$\cos(\omega x) = \sin(\omega x - \pi/2)$$

Man sagt, die Sinus- und Kosinusfunktion stehen rechtwinklig oder orthogonal zueinanderstehen. Wir erkennen das auch an der Darstellung der Sinus- und Kosinus-

komponenten im Einheitskreis. Wir können damit einen beliebigen Sinusoid mit Amplitude C und Phase φ durch seine Kosinus- und Sinuskomponente zusammensetzen. Man beachte, dass dabei der Phasenwinkel verschwindet:



Komplexwertige Sinusfunktion in der Eulerschen Notation

Die obige Summe aus zwei rechtwinkligen Komponenten erinnert an eine komplexe Zahl mit ihrem realen und imaginären Anteil:

$$z = a_{real} + i \cdot b_{imag} \text{ worin } i \text{ die imaginäre Einheit ist } (i^2 = -1).$$

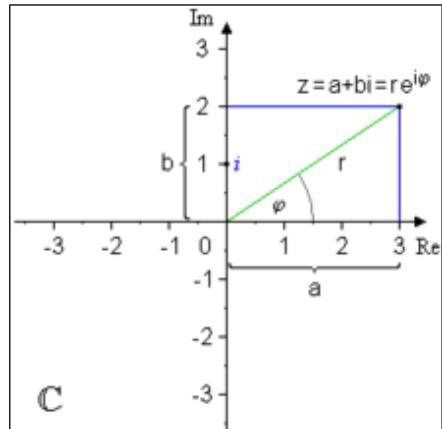


Abb. 152: Komplexe Zahlen können auch in einem zweidimensionalen Zahlenraum dargestellt werden. Die horizontale Achse entspricht dem realen Anteil und die vertikale Achse entspricht dem imaginären Anteil. Bildquelle: [Wikipedia](#).

Die sogenannte Eulersche Notation einer komplexen Zahl ist praktisch identisch zur komponentenweisen Addition aus Kosinus- und Sinuskomponente. Darin entsprechen die Kosinuskomponente der realen Komponente und der Sinusanteil der imaginären Komponente. Die Zahl e steht für die Eulersche Zahl $e = 2.71828$:

$$e^{i\varphi} = e^{i\omega x} = \cos(\omega x) + i \cdot \sin(\omega x)$$

Periodische Signale als Summe von Sinus- und Kosinusfunktionen

Wie Fourier nun herausgefunden kann jedes periodische Signal als Summe von gewichteten Sinus- und Kosinusfunktionen dargestellt werden. Diese Summe können wir in der Kurvendarstellung von mehreren Signalen leicht nachvollziehen. Es werden schlicht alle Kurvenpunkte aufaddiert.

Um das Signal als Summe von mehreren Kreisbewegungen zu verstehen, reicht die statische Darstellung in einem Bild nicht mehr aus. Lucas Vieira hat dazu eine wunderbare Flash-Animation erstellt. Sie finden diese auf folgender Webseite:

<http://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform>

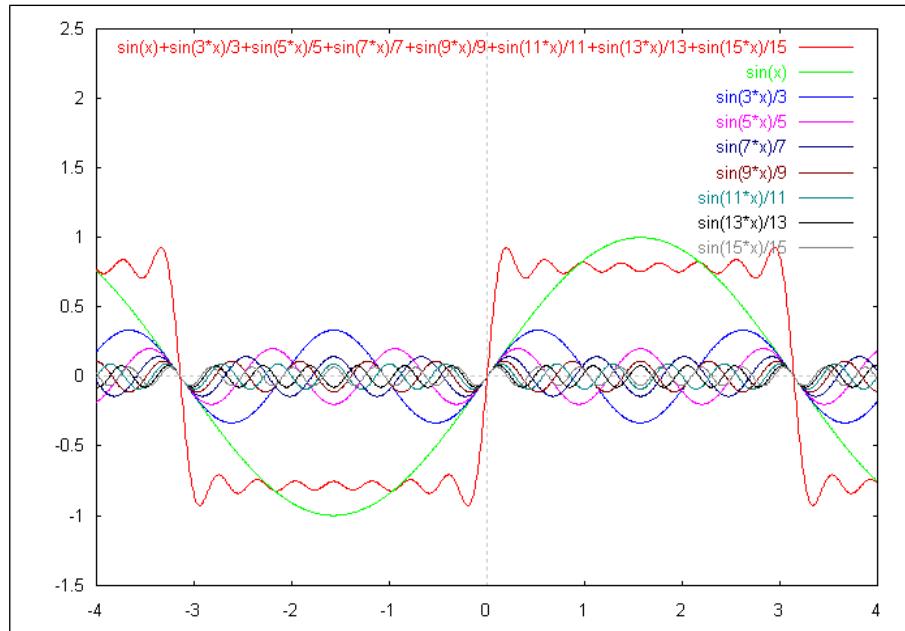


Abb. 153: Annäherung an ein Rechtecksignal als Summe (rote Kurve) von 6 Sinusfrequenzen.

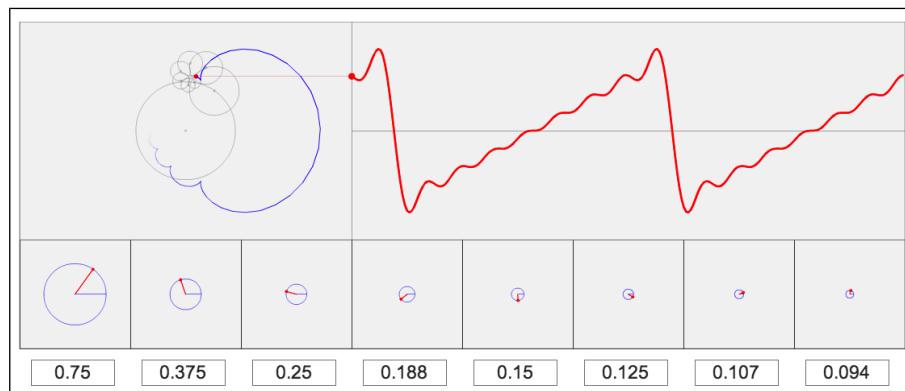


Abb. 154: Annäherung eines Sägezahnsignals als Summe von 8 Sinusfrequenzen. Die blaue Kurve links oben zeigt die Summe aller 6 Kreisbewegungen mit ihren verschiedenen Radien.

7.1.2 1D-Fourier-Transformation

Die Fourier-Transformation zerlegt ein Signal in eine Folge von Kosinus und Sinus Komponenten. Sie kann also für die beiden vorangegangenen Beispiele die Faktoren der einzelnen Sinus- und/oder Kosinusfrequenzen alleine aus den roten Signalen berechnen.

Die **kontinuierliche Fourier-Transformation (CFT oder FT)** für jede Frequenz ω wird mit folgendem Integral dargestellt:

$$G(\omega) = \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{\infty} g(x) \cdot e^{-i\omega x} dx \quad \text{worin: } e^{i\omega x} = \cos(\omega x) + i \cdot \sin(\omega x)$$

Die **Inverse Fourier-Transformation (IFT)** also die Rücktransformation vom Frequenzraum in den Ortsraum sieht bis auf ein Vorzeichen gleich aus:

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

Die **Diskrete Fourier-Transformation (DFT)** stellt die diskrete Version der kontinuierlichen FT dar. Für ein Signal $g(u)$ der Länge M ($u=0\dots M-1$) sieht die diskrete FT wie folgt aus:

$$G(m) = \frac{1}{\sqrt{M}} \cdot \sum_{u=0}^{M-1} g(u) \cdot e^{-i2\pi \frac{mu}{M}}$$

Die **Inverse Diskrete Fourier-Transformation (IDFT)** sieht bis auf das Vorzeichen im Exponenten und den Faktor $1/N$ ebenfalls gleich aus:

$$g(u) = \frac{1}{\sqrt{M}} \cdot \sum_{m=0}^{M-1} G(m) \cdot e^{i2\pi \frac{mu}{M}}$$

Sowohl das Signal $g(u)$ als auch das Frequenzspektrum $G(m)$ sind komplexwertige Vektoren der Länge M ($u,m=0\dots M-1$). Beim Signal $g(u)$ steckt das eigentliche Eingangssignal nur im Realteil, der imaginäre Teil ist immer 0:

$$g(u) = g_{real}(u) + i \cdot g_{imag}(u)$$

$$G(m) = G_{real}(m) + i \cdot G_{imag}(m)$$

Ersetzen wir die Eulersche Notation wieder durch die Sinus- und Kosinuskomponenten, so erhalten wir in der Summe eine Multiplikation von zwei komplexen Zahlen:

$$G(m) = \frac{1}{\sqrt{M}} \cdot \sum_{u=0}^{M-1} [g_{real}(u) + i \cdot g_{imag}(u)] \cdot \left[\cos\left(2\pi \frac{mu}{M}\right) - i \cdot \sin\left(2\pi \frac{mu}{M}\right) \right]$$

$$g(u) = \frac{1}{\sqrt{M}} \cdot \sum_{m=0}^{M-1} [G_{real}(m) + i \cdot G_{imag}(m)] \cdot \left[\cos\left(2\pi \frac{mu}{M}\right) + i \cdot \sin\left(2\pi \frac{mu}{M}\right) \right]$$

Sowohl bei der kontinuierlichen als auch bei der diskreten FT sehen Sie vor dem Integral resp. vor der Summe einen Skalierungsfaktor. Je nach Quelle kann dieser anders sein oder ganz weggelassen sein. Die Formeln in diesem Skript sind gleich gehalten wie bei [Burger06].

Wir interessieren uns aber nicht so sehr für den realen- oder imaginären Teil, sondern für die daraus abzuleitende Amplitude und Phase der einzelnen Frequenzen:

$$\text{Amplitude} = |G(m)| = \sqrt{G_{real}(m)^2 + G_{imag}(m)^2} \quad \text{Phase} = \text{atan}\left(\frac{G_{imag}(m)}{G_{real}(m)}\right)$$

 Diese Formulierung der diskreten Fouriertransformation können wir in einer Programmiersprache direkt ausprogrammieren. Die Vorwärts- und die Rückwärtstransformation sind in einer Routine **DFT_1D** implementiert:

```
class Complex
{
    double re, im;
    Complex(double re, double im)
    {
        this.re = re;
        this.im = im;
    }
}

Complex[] DFT_1D(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] . [cos() + i sin()]
        sumRe += gRe * cosw + gIm * sinw;
        sumIm += gIm * cosw - gRe * sinw;
    }
    G[m] = new Complex(s * sumRe, s * sumIm);
}
return G;
}

```

Fast Fourier Transformation (FFT)

Die direkte Implementation führt also in der inneren der beiden Schlaufen je eine teure Sinus- und Kosinusfunktion aus. In der 2D-Version der FT wären es 4 verschachtelte Schlaufen. Dies ist der Grund, warum die FT so aufwendig ist und damit praktisch nicht brauchbar war für längere Signale. Wir können den obigen Algorithmus natürlich beschleunigen, indem wir die Sinus- und Kosinusaufzüge in einer Lookup-Tabelle auslagern, aber es bleiben immer noch die $8 \times N^4$ Fließkommamultiplikationen.

1965 entwickelten die Herren [Cooley](#) und [Tukey](#) die [Fast Fourier Transformation](#), welche das Laufzeitverhalten zum einem $O(n \log n)$ Algorithmus verbesserte. Auf diesem Algorithmus basieren heute praktisch alle Implementierungen, u. a. auch die freie Implementation der FFTW ([Fastest Fourier Transform in the West](#)).



7.1.3 Übung 1D-Fourier-Transformation mit Excel

In dieser Übung geht es darum herauszufinden, ob die Fourier-Analyse-Funktion von Excel genau die Frequenzen herausfindet, die wir vorher gezielt in einem Signal zusammengesetzt haben.

1. Konstruieren Sie dazu in einem Excel-Blatt ein diskretes Sinussignal mit 128 Werten, das sich aus folgenden Sinuswellen zusammensetzt:

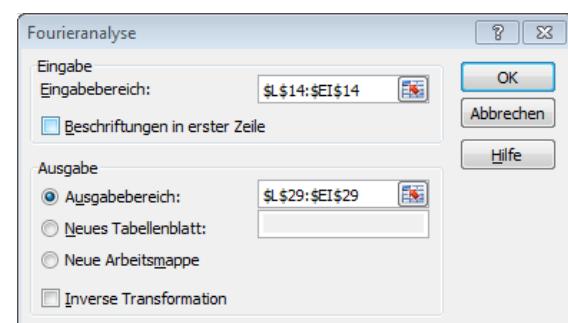
$$f(x) = \sin(x) + \frac{1}{3} \sin(3x) + \frac{1}{5} \sin(5x) + \frac{1}{7} \sin(7x)$$

2. Visualisieren Sie die einzelnen Wellen sowie die Summe davon in einer Grafik.

3. Die FT in Excel befindet sich in den Analysefunktionen, einem Add-In, dass speziell eingebunden werden muss (Menü [Extras/Add-Ins Manager](#)). In Excel ab Version 2007 müssen die Add-Ins zuerst über Datei > Optionen eingeschaltet werden. Nach dem einschalten stehen die Analyse Funktionen im Reiter „Daten“ zur Verfügung.

Die Excel-Fourier-Analyse berechnet aus einer 2^n -grossen Zahlenserien, die ebenso grosse fouriertransformierte Serie mit komplexen Zahlen. Die Amplitude und die Phase können daraus mit folgenden Excel-Funktionen berechnet werden:

$$\text{Amplitude} = \sqrt{(\text{IMREALTEIL}(Xn))^2 + (\text{IMAGINÄRTEIL}(Xn))^2}$$

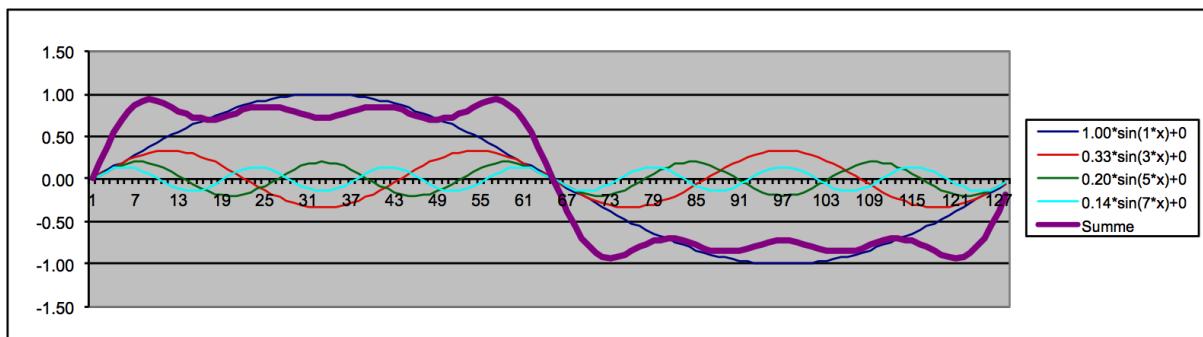


Phase = WENN(IMAGINÄRTEIL(Xn)=0;0;ARCTAN2(IMREALTEIL(Xn);IMAGINÄRTEIL(Xn)))

4. Stimmen die detektierten Frequenzen und Amplituden mit denen überein, die Sie ins Signal reingesteckt haben?
5. Verschieben Sie eine oder mehrere Frequenzen noch oben. Erscheint er nach der Analyse im Spektrum an der Stelle 0?
6. Welches ist die höchste Frequenz, die Sie noch detektieren können?

Fourier-Transformation eines aus 4 Frequenzen bestehenden Signals mit N = 128 Abtastungen (Samples)

N	=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Abtastung bei PI/64 =	x =	0.00	0.05	0.10	0.15	0.20	0.25	0.29	0.34	0.39	0.44	0.49	0.54	0.59	0.64	0.69	0.74
1) 1.0 * sin(1 *x) + 0.0	=	0.00	0.05	0.10	0.15	0.20	0.24	0.29	0.34	0.38	0.43	0.47	0.51	0.56	0.60	0.63	0.67
2) 0.3 * sin(3 *x) + 0.0	=	0.00	0.05	0.10	0.14	0.19	0.22	0.26	0.29	0.31	0.32	0.33	0.33	0.31	0.29	0.27	
3) 0.2 * sin(5 *x) + 0.0	=	0.00	0.05	0.09	0.13	0.17	0.19	0.20	0.20	0.18	0.16	0.13	0.09	0.04	-0.01	-0.06	-0.10
4) 0.1 * sin(7 *x) + 0.0	=	0.00	0.05	0.09	0.12	0.14	0.14	0.13	0.10	0.05	0.01	-0.04	-0.09	-0.12	-0.14	-0.14	-0.13
		0.00	0.19	0.38	0.55	0.69	0.80	0.87	0.92	0.93	0.92	0.89	0.85	0.80	0.76	0.73	0.71

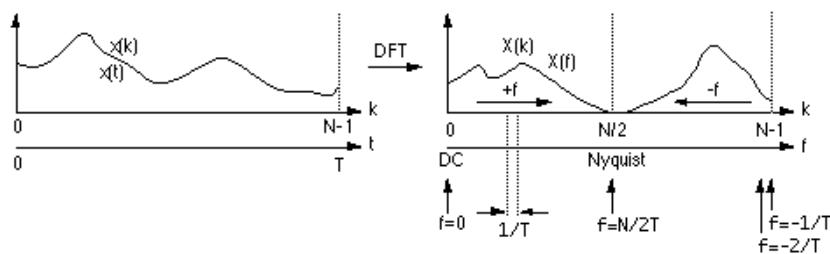


Fourieranalyse	=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Amplitude = (real^2+imag^2)^0.5 =		0	-64i	0	-21.33i	0	-12.8i	0	-9.142i	0	0	0	0	0	0	0	0
Amplitude geshiftet		0	64	0	21.3	0	12.8	0	9.14	0	0	0	0	0	0	0	0
Phase = arctan2(real,imag) =		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Amplitudenspektrum der FDFT		0	-1.6	0	-1.6	0	-1.6	0	-1.6	0	0	0	0	0	0	0	0

Abb. 155: 1D-Fourier Transformation mit Excel

Bemerkungen zum Excel-Beispiel

- Die FT liefert für 128 Samples, 128 komplexe Zahlen, deren Betrag (Länge des Vektors) der Frequenzamplitude entspricht.
- Die erste komplexe Zahl enthält den DC-Wert (*Direct Current*), der dem Durchschnitt der Signalwerte entspricht. In unserem Beispiel ist er 0, da das Signal um die 0-Linie alterniert.
- Die Frequenzanteile sind symmetrisch um die *Nyquist-Frequenz* (s. auch Kapitel 12.1.1.1) angeordnet. Unterhalb sind die positiven und oberhalb die negativen Frequenzen. Die Nyquist-Frequenz ist die höchste Frequenz, für die eine verlustfreie Transformation durchgeführt werden kann. Meist werden die Frequenzanteile um den 0-Punkt angeordnet (*geshiftet*).



7.1.4 2D-Fourier-Transformation

Die Fourier-Transformation kann jedes periodische Signal in seine Kosinus- und Sinusfrequenzanteile zerlegen. Ist ein Signal nicht periodisch, wie es ein Bild im Normalfall ist, so macht man es einfach periodisch, indem man es sich mehrfach dupliziert vorstellt:



Abb. 156: Die 2D-Fourier-Transformation interpretiert ein Bild als ein sich periodisch wiederholendes Signal. Wenn das Signal einfach repetiert wird, so können an den Bildrändern hohe Frequenzen entstehen. Dies könnte reduziert werden, indem das Signal gespiegelt repetiert wird.

So wie wir die Frequenzen in einem eindimensionalen Signal feststellen können, so können wir mit der zweidimensionalen Fourier-Analyse die Frequenzen in einem zweidimensionalen Signal feststellen. Eine Sinusfrequenz in einem diskreten $M \times N$ Signale ist wie definiert mit:

$$\sin[2\pi(Um+Vn)]$$

worin U und V die horizontale resp. vertikale Frequenzkomponente darstellen. U und V können auch als Richtungsvektor aufgefasst werden, in die die Frequenz am schnellsten schwingt. Die Amplitude der Frequenz wird demnach als Länge des Vektors $[U,V]$ bestimmt und die Phase als dessen Richtung:

$$\text{Amplitude: } \Omega = \sqrt{U^2 + V^2} \quad \text{Phase: } \theta = \tan^{-1}\left(\frac{U}{V}\right)$$

Wenn wir für jede Zeile eines Bildes (eindimensionales Signal) mit N Pixel die DFT durchführen, so erhalten wir für jede Zeile N Frequenzanteile mit Amplitude und Phase. Wenn wir diese Ausgabe wieder als "Bild" auffassen und nun für jede der M -Spalte die DFT durchführen, so erhalten wir ein zweites "Bild", das als Fourier-Transformation des ursprünglichen Bildes bezeichnet wird.

Die **diskrete 2D-Fourier-Transformation** ist wie folgt definiert:

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

Wird also ein Signal MxN-mal abgetastet, so ergibt die DFT M x N komplexe Zahlen:

$$G(u, v) = G_{real}(u, v) + i \cdot G_{imag}(u, v) = \begin{bmatrix} G_{0,0} & G_{0,1} & \cdots & G_{0,N-1} \\ G_{1,0} & G_{1,1} & \cdots & G_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ G_{M-1,0} & G_{M-1,1} & \cdots & G_{M-1,N-1} \end{bmatrix}$$

Die **inverse, diskrete 2D-Fourier-Transformation** sieht wiederum bis auf das Vorzeichen im Exponenten ebenfalls gleich aus:

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

Genau wie bei der eindimensionalen DFT werden aus dem Real- und dem Imaginärteil ein Amplitudenspektrum und ein Phasenspektrum berechnet.



Abb. 157: Lena im Ortsraum

DFT
→

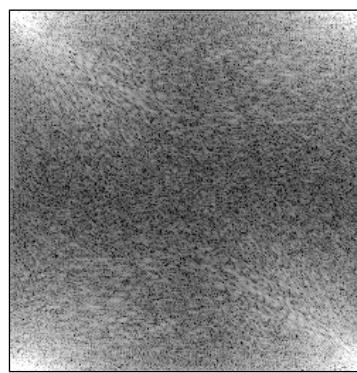
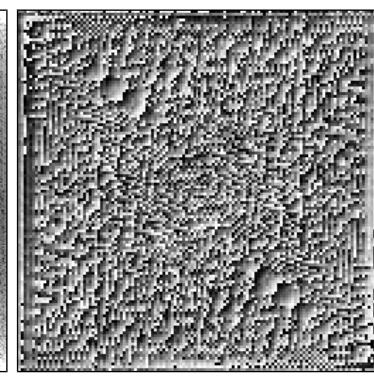
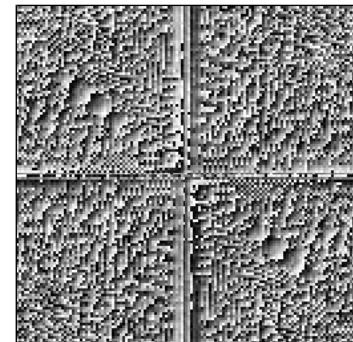
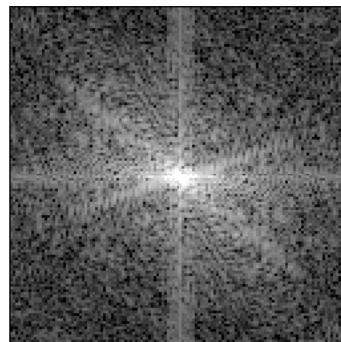
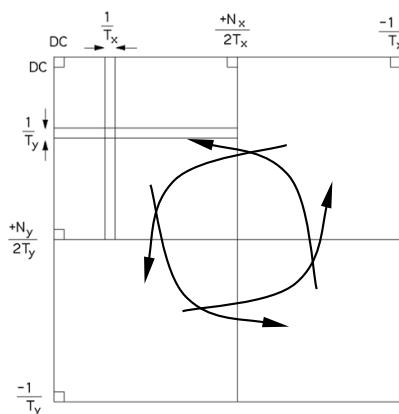


Abb. 158: Amplituden- und Phasenspektrum der Lena nach der DFT



Wie bei der eindimensionalen DFT besitzt die Ausgabe ein bestimmtes Format.

- Die Ausgabe einer 2D-DFT eines M x N grossen Bildes ist ein M x N grosser Array mit komplexen Zahlen.
- Die erste komplexe Zahl enthält den DC-Wert (*Direct Current*), der dem Durchschnitt der Signalwerte entspricht.
- Die Frequenzanteile sind punktsymmetrisch um die Nyquist-Frequenz angeordnet. Unterhalb sind die positiven und oberhalb die negativen Frequenzen. Meist werden die vier Quadranten um den Mittelpunkt vertauscht, damit die tiefen Frequenzanteile um den Mittelpunkt zentriert sind.



So unglaublich es erscheinen mag, doch die beiden Darstellungen des Amplitudenspektrums und des Phasenspektrums stellen zusammen die gleichen Informationen dar, die im Bild der Lena stecken. Obwohl das Phasenspektrum nichtssagend ist, so enthält die Phase doch die

wichtige Information über den Ort einer Frequenz, während die Amplitude nur über deren Stärke Auskunft gibt. Man kann dies einfach nachvollziehen, wenn man zwei komplett unterschiedliche Bilder in den Frequenzraum transformiert und sie mit vertauschten Phasenspektren rekonstruiert:

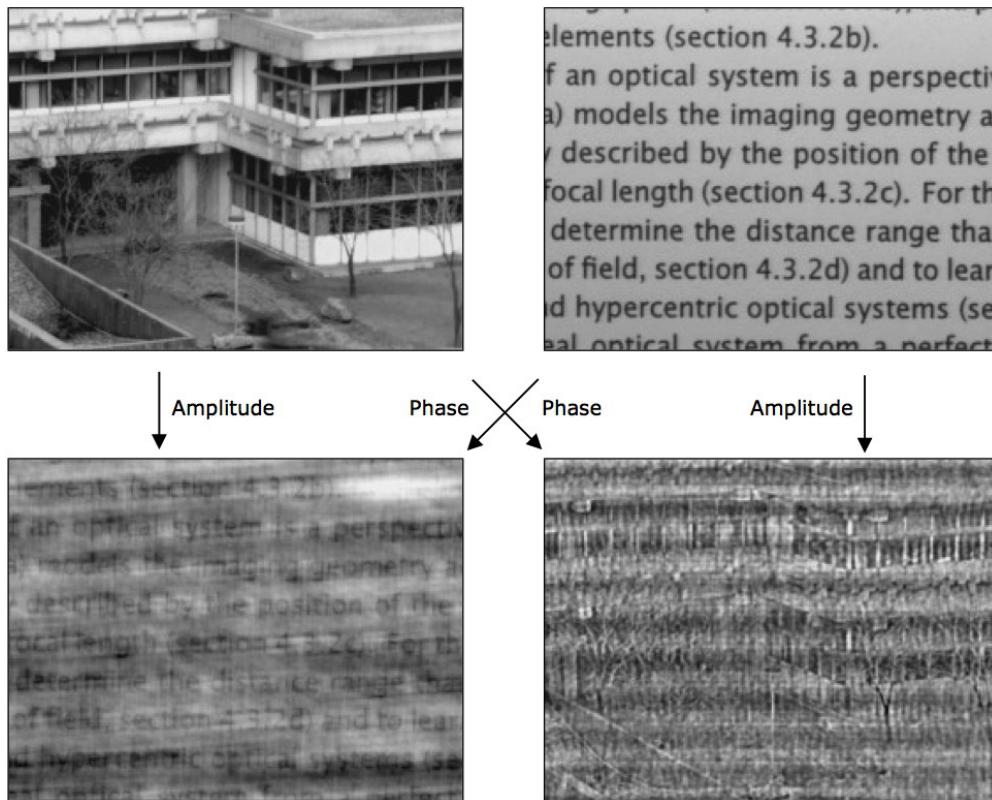


Abb. 159 Rekonstruktion mit vertauschten Phasenspektren [Jähne05]. Wir erkennen darin, dass die Phase die wichtige Information über den Ort enthält.

7.1.4.1 Interpretation des Amplitudenspektrums

Um ein besseres Verständnis für das 2D-Amplitudenspektrum zu erhalten, werden in der Folge Bilder mit einem ganz bestimmten Frequenzspektrum erzeugt und überprüft, ob das Amplitudenspektrum nach der FT auch die entsprechenden Peaks enthält.

Bei allen nachfolgenden Beispielen sehen Sie zuerst das Eingangsbild gefolgt von dessen Amplituden- und Phasenspektrum im Frequenzraum:

$$I(u, v) = \frac{1}{2} \sin\left(\frac{2\pi}{128}(1u + 0v)\right) + \frac{1}{2}$$

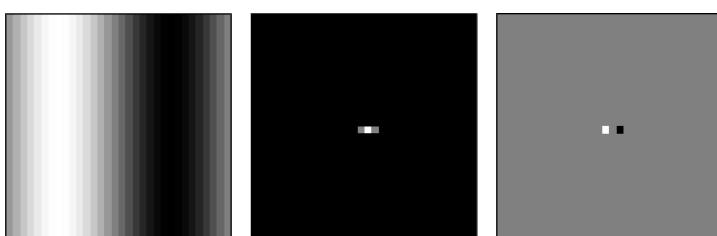


Abb. 160: Eine einfache horizontale Sinusschwingung erzeugt neben dem DC in der Mitte genau zwei Peaks.

$$I(u, v) = \frac{1}{2} \sin\left(\frac{2\pi}{128}(4u+4v)\right) + \frac{1}{2}$$

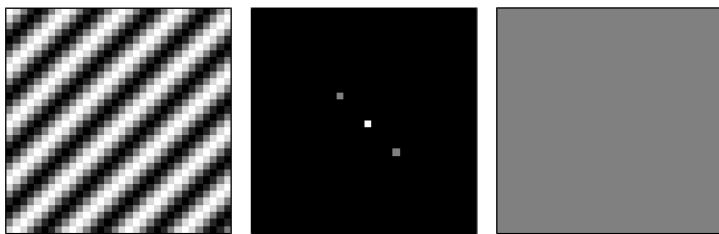


Abb. 161: Eine vierfache diagonale Sinusschwingung erzeugt auch zwei Peaks, vier Pixel diagonal vom Zentrum entfernt.

$$I(u, v) = \frac{1}{2} \sin\left(\frac{2\pi}{128}(4u)\right) \sin\left(\frac{2\pi}{128}(4v)\right) + \frac{1}{2}$$

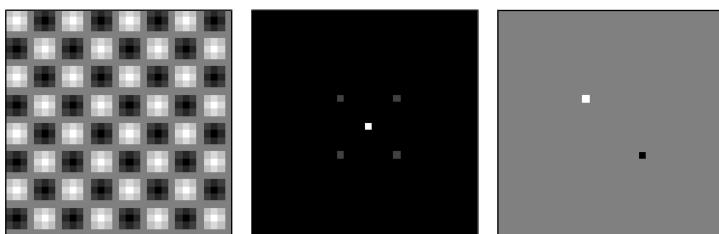


Abb. 162: Eine vierfache horizontale und vertikale Sinusschwingung erzeugt 4 Peaks, 4 Pixel vom Zentrum entfernt, je einen pro Quadranten.

$$I(u, v) = \sum_{i=1}^{i=7; i+=2} \frac{1}{2i} \sin\left(\frac{2\pi}{128}(iu+iv)\right) + \frac{1}{2}$$

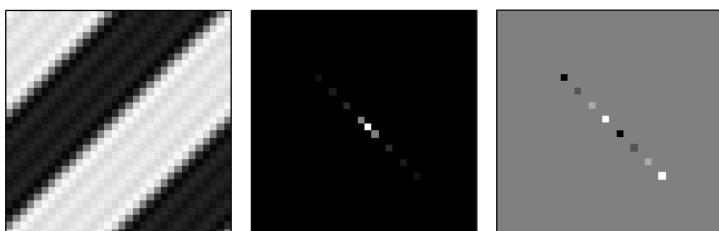


Abb. 163: Eine ein-, drei-, fünf- und siebenfache diagonale Sinusschwingung ergeben aufsummiert einen angenäherten diagonalen Schwarz/Weiss-Balken und erzeugen je 4 Peaks in zwei Quadranten.

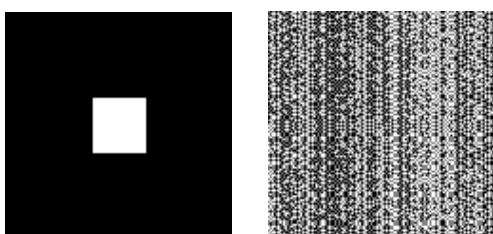


Abb. 164: Ein weisses Quadrat auf schwarzem Hintergrund hat unendlich hohe Frequenzen, sodass im Amplitudenspektrum auf der Horizontalen und Vertikalen alle Frequenzen auftreten.

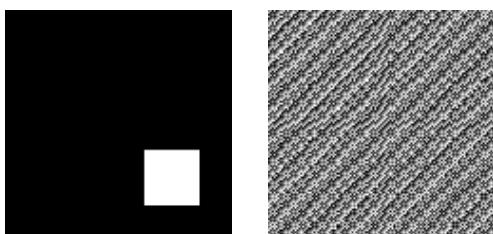


Abb. 165: Dasselbe Quadrat verschoben bewirkt nur ein anderes Phasenspektrum.

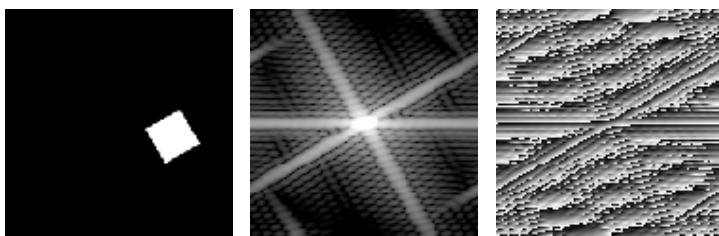


Abb. 166: Ein rotiertes Quadrat erzeugt ein um denselben Winkel gedrehtes Amplitudenspektrum.

7.1.5 Anwendungen

7.1.5.1 Filterung im Frequenzraum

Auch wenn für die Rekonstruktion beide Spektren wichtig sind, so ist doch das Amplitudenspektrum der Beweggrund für die aufwendige Fourier-Transformation. In ihm lassen sich nun die lokalen Filter, die wir im Ortsraum kennengelernt haben, ganz einfach durch Skalierung der Amplituden realisieren. Eine Filterfunktion wird im Frequenzraum *Transferfunktion* genannt und kann erzeugt werden, indem der *Filterkernel* ebenfalls durch die DFT geschickt wird.

Der Faltungssatz der Fourier-Theorie besagt, dass eine Faltung eines Signals g mit einer Faltungsmaske h im Ortsraum einer Multiplikation der Fourier-Transformierten von g mit der Fourier-Transformierten von h im Frequenzraum entspricht.

$$f = g * h = IFT(G \cdot H) = IFT(FT(g) \cdot FT(h))$$

Dies erlaubt uns, eine Filterung mit demselben Resultat auch im Frequenzraum durchzuführen. Sie hat zudem den Vorteil, dass das Randproblem nicht auftritt und das bei grösseren Bildern, der Rechenaufwand geringer ist, als bei der Filterung im Ortsraum. Dies trifft allerdings nur zu, wenn die *Fast Fourier Transformation (FFT)* verwendet wird.

Der Faltungssatz gilt in der Praxis allerdings nur, wenn wir bei der Faltung das Randproblem durch Replizieren gelöst haben. Nur diese Randergänzung entspricht der periodischen Signalinterpretation der 2D-DFT. Es können dadurch hohe Frequenzen hinzukommen, wie z. B. im nachfolgend Bild der Lena am linken und rechten Rand, die im eigentlichen Bild der Lena nicht vorhanden sind.

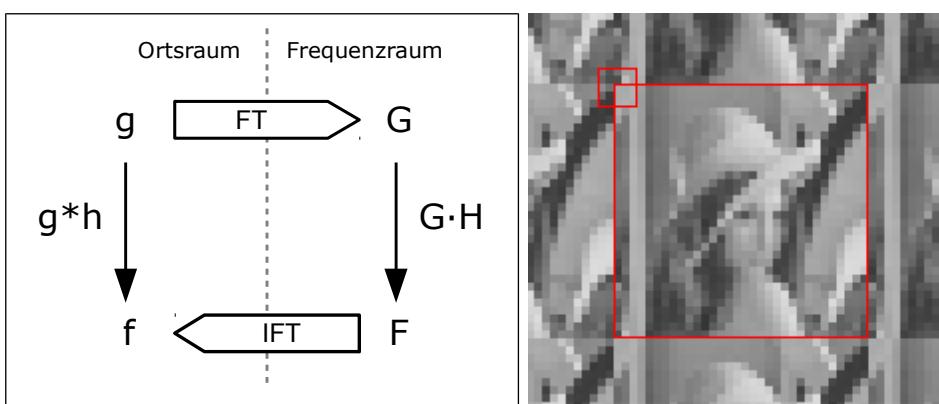


Abb. 167: Links: Eine Faltung im Ortsraum entspricht einer komponentenweisen Multiplikation beider Transformierten im Frequenzraum, allerdings nur, wenn das Randproblem bei der Faltung durch Replikation (Bild rechts).

Die Darstellung des Gaussfilters als Transferfunktion im Frequenzraum macht die Vorteile gegenüber dem Rechteckfilter deutlich. Multipliziert man das Amplitudenspektrum mit der Gaussglocke, so werden die hohen Frequenzen am Rande des Spektrums gleichmässig reduziert. Die Gestalt der Transferfunktion des Rechteckfilters im Frequenzraum erklärt, warum verschiedene Frequenzen damit ungleichmässig behandelt werden.

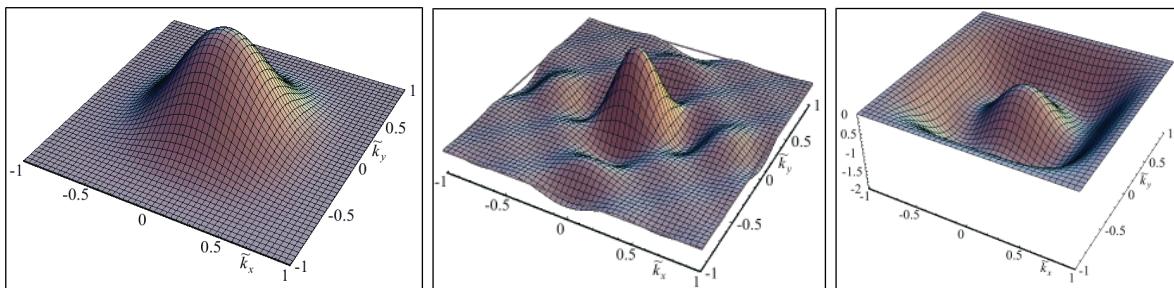


Abb. 168: Gaussfilter, Rechteckfilter und LoG-Filter als Transferfunktion nach der DFT [Jähne05]

Beispiel: Laplace-Filter im Ortsraum und Frequenzraum:

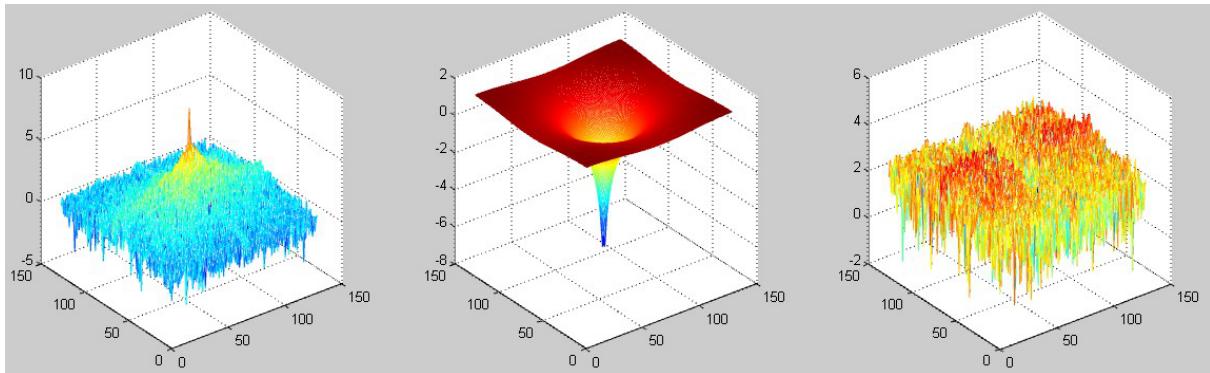


Abb. 169: Links das Amplitudenspektrum des Lena-Bildes, in der Mitte das Amplitudenspektrum des Laplace-Filter und rechts das Amplitudenspektrum nach der komponentenweisen Multiplikation der beiden. Zurück transformiert ergibt Letzteres bis auf ein paar Randpixel das identische Resultat wie eine Faltung mit einem lokalen Laplace-Filter.

7.1.5.2 Inverse Filterung im Frequenzraum

Wenn eine komponentenweise Multiplikation im Frequenzraum einer Filterung im Ortsraum entspricht, so muss eine komponentenweise Division zu einer inversen Filterung führen. Inverse Filterung wird auch *Deconvolution (Entfaltung)* genannt und ist hauptsächlich im Frequenzraum möglich.

Point Spread Function (PSF)

Fokussierungs- und Bewegungsunschärfen können mit lokalen Filtermasken erzeugt werden. Diese Degradierungsfunktion wird auch Punktantwort oder Punktspreizfunktion (*Point Spread Function (PSF)*) genannt (s. Kapitel 3.1.3). Gewisse Aberrationen, wie z. B. die sphärische Aberration kann durch eine PSF beschrieben werden.

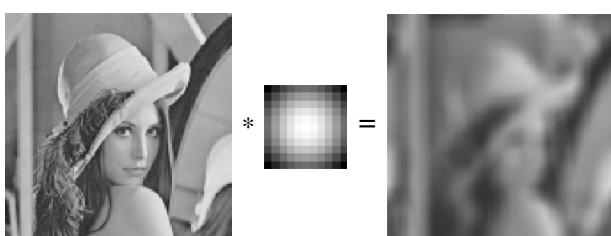


Abb. 170: Verunschärfung durch eine 7x7 Gaussfilter PSF.

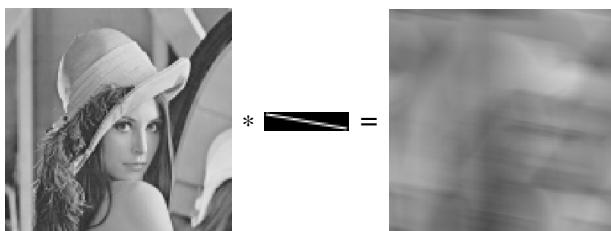


Abb. 171: Bewegungsunschärfe erzeugt mit einer 9x41 Pixel grossen PSF Filtermaske.

Kennt man also die genaue PSF, so kann man die Bildstörung durch inverse Filterung beheben, indem man durch die Fourier-transformierte PSF (H) dividiert:

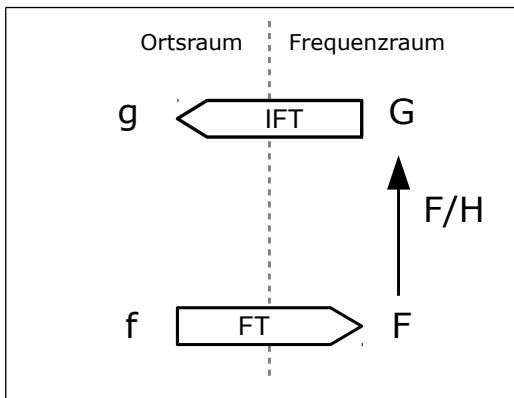


Abb. 172: Inverse Filterung im Frequenzraum macht eine Filterung wieder rückgängig.

Bei der Division im Frequenzraum muss darauf geachtet werden, dass nicht durch 0 dividiert wird. Das heisst, dass der absolute Betrag von H z. B. grösser als 0.00001 ist. Andernfalls muss $G/H = 0$ sein.

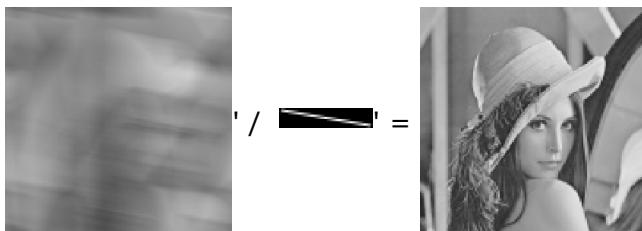


Abb. 173: Bei bekannter PSF lässt sich die Verunschärfung im Frequenzraum (' $/$ ') rückgängig machen.

Leider ist die inverse Filterung mit bekannter PSF von geringem Wert, weil schon das geringste Rauschen eine akzeptable Rekonstruktion verunmöglicht. Die Verstärkung röhrt daher, dass ein Tiefpassfilter h bei hohen Frequenzen gegen 0 abfällt, während das Rauschen gerade dort Frequenzanteile enthält, die dann durch $1/H$ verstärkt werden.

$$f = g * h + noise$$

$$G = \frac{(f + noise)'}{H} = \frac{F}{H} + \frac{noise'}{H}$$

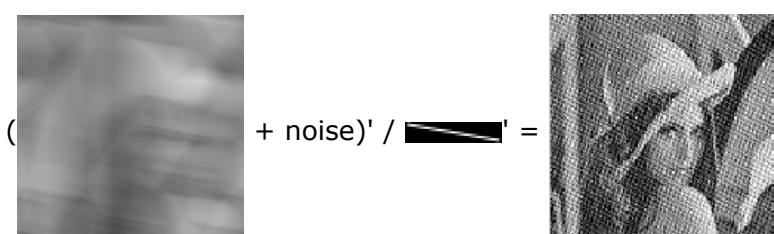


Abb. 174: Aber schon eine kleinste Menge an Rauschen verunmöglicht eine Wiederherstellung.

Diese hohe Rauschempfindlichkeit ist der Grund, warum eine Entfaltung im Ortsraum nicht möglich ist. Wir können zwar H in Ortsraum zurücktransformieren, aber durch das Randproblem bei lokalen Filtermasken entstehen zu hohe Frequenzen, welche eine Restaurierung verunmöglichen.

Blind Deconvolution



Die Bestimmung einer unbekannten PSF und die damit durchgeführte Restaurierung eines verunschärften Bildes wird *Blind Deconvolution* genannt. In der [Matlab Dokumentation](#) finden Sie einige Beispiele dazu.

7.1.5.3 Direkte Manipulation im Amplitudenspektrum

Die unerwünschten Frequenzbereiche können aber auch gezielt im Amplitudenspektrum gelöscht werden:

Beispiel: Tiefpassfilterung im Frequenzraum [Bourke98]

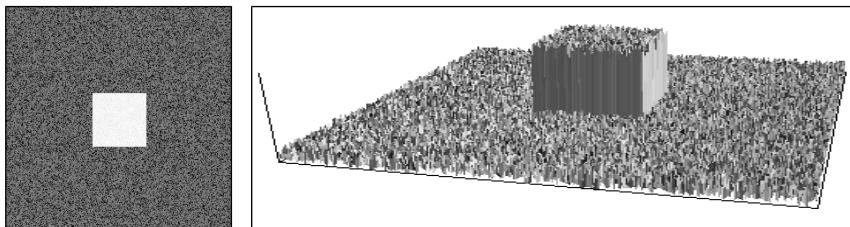


Abb. 175: Verrausches Ausgangsbild und seine 3D-Darstellung im Ortsraum

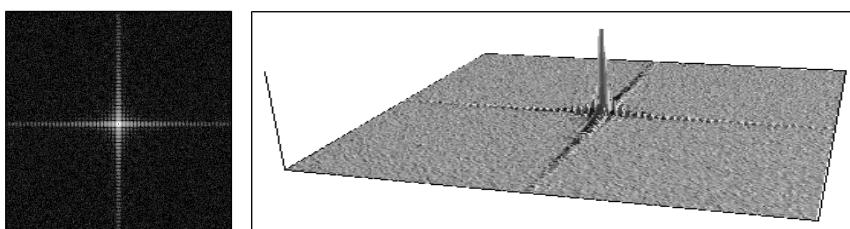


Abb. 176: 2D und 3D Darstellung des Amplitudenspektrums nach der DFT

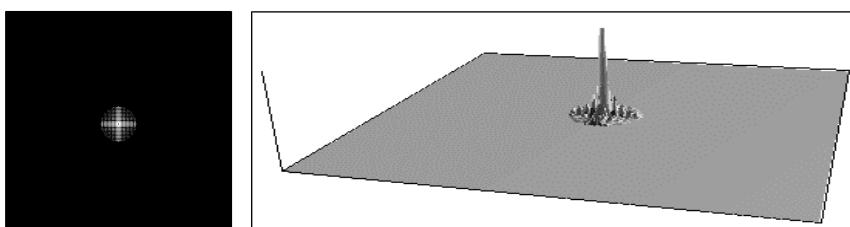


Abb. 177: Korrigiertes Amplitudenspektrum mit gelöschten, hohen Frequenzen um die Mitte

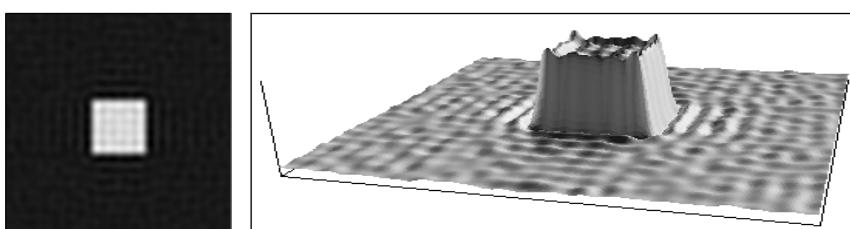


Abb. 178: Rücktransformiertes Bild nach der Tiefpassfilterung

Beispiel: Hochpassfilterung im Frequenzraum [Bourke98]

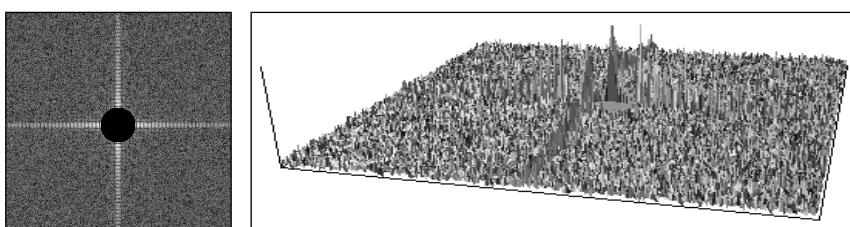


Abb. 179: Korrigiertes Amplitudenspektrum mit gelöschten, tiefen Frequenzen in der Mitte

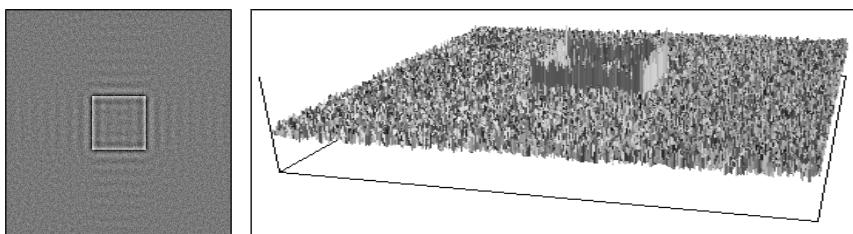


Abb. 180: Rücktransformiertes Bild nach der Hochpassfilterung

7.1.6 Fourier Transformation in ImageJ



Im Menü *Process > FFT* können Sie für die FFT, die IFFT, sowie einen Optionendialog aufrufen. Standardmäßig wird das quadrantengespiegelte und logarithmisch skalierte Amplitudenspektrum angezeigt. Dies ist jeweils nur eine Darstellung. Die komplexwertige FT wird im Hintergrund gehalten. Sie können mit den Selektionswerkzeugen Regionen markieren und sie entweder mit 0 (Schwarz) oder 255 (Weiss) füllen. Ersteres bewirkt ein Unterdrücken der jeweiligen Frequenzen und Letzteres ein Überleben. Alle anderen Grauwerte ausser 0 und 255 haben keinen Einfluss bei der IFT. Sie können die Farbe mit *ColorPicker* Dialog einstellen (Doppelklicken auf das Werkzeug).

Einen guten Überblick über die FT-Anwendung in ImageJ finden Sie im [Wiki der ImageJ Dokumentation](#).

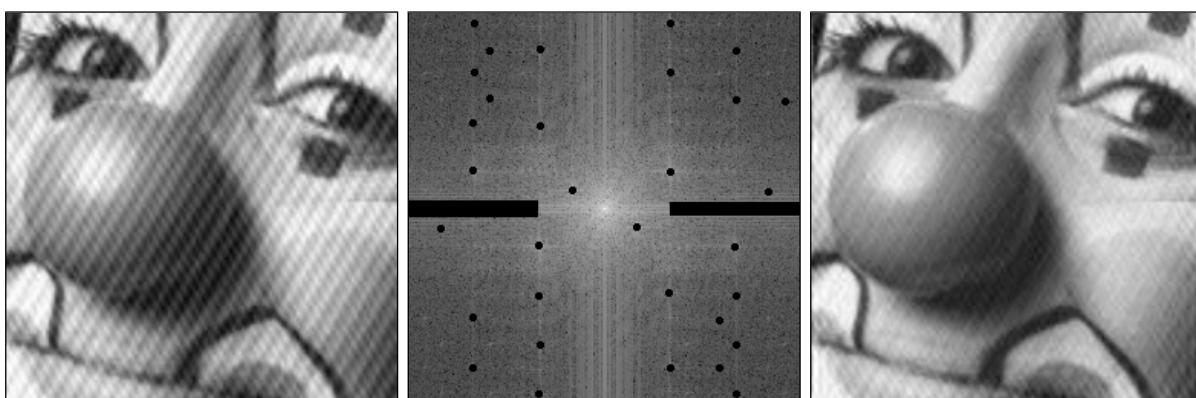


Abb. 181: Entfernen von strukturellem Rauschen durch Löschen von hohen Frequenzanteilen im Amplitudenspektrum in ImageJ.

7.1.7 Fourier Transformation in Matlab



Wie in Excel wird auf in Matlab die diskrete Fourier-Transformation mit der *Fast Fourier Transformation (FFT)* durchgeführt. Für die 1D-Transformation steht die Funktion *fft* und für die 2D-Transformation die Funktion *fft2* zur Verfügung, wobei die Letztere auch mit der 1D-Version dargestellt werden kann: *fft2(I) = fft(fft(I).')*.

Die Ausführungsgeschwindigkeit hängt von der Kantenlänge der Bilder ab. Am schnellsten werden Kantenlängen der Grösse 2^n transformiert.

Im Skript *Fourier2D.m* werden alle vorangegangenen Beispiele erzeugt. Damit das Amplitudenspektrum besser sichtbar wird, wird es meistens logarithmiert dargestellt.

```

X = imread('..../images/Lena128.png'); %Load image into matrix X
I = im2double(X); %Convert to double matrix I
F = fftshift(fft2(I)); %FFT2 shifted
Amp = log(abs(F)); %Amplitude abs(F)=sqrt(real(F)^2+imag(F)^2)
Pha = angle(F); %Phase angle(F)=atan2(imag(F),real(F))
subplot(1,3,1), imshow(I, 'InitialMagnification','fit');
subplot(1,3,2), imshow(Amp, []); [%=[min(min(Amp)) max(max(Amp))]];
subplot(1,3,3), imshow(Pha, []); [%=[min(min(Pha)) max(max(Pha))]];
pause;

%-----
fprintf ('\nTransform square ...');

```

```

size = 128;
I = zeros(size,size);
I(50:80,50:80) = 1;
F = fftshift(fft2(I, 128, 128));
subplot(1,3,1), imshow(I);
subplot(1,3,2), imshow(log(abs(F)),[]);
subplot(1,3,3), imshow(angle(F),[]);
pause;
%-----
fprintf ('\nTransform square translated ...');
I = zeros(size,size);
I(80:110,80:110) = 1;
F = fftshift(fft2(I, 128, 128));
subplot(1,3,1), imshow(I);
subplot(1,3,2), imshow(log(abs(F)),[]);
subplot(1,3,3), imshow(angle(F),[]);
pause;
%-----
fprintf ('\nTransform square rotated ...');
I = zeros(size,size);
I(80:110,80:110) = 1;
I2 = imrotate(I, 30, 'bicubic');
F = fftshift(fft2(I2, 128, 128));
subplot(1,3,1), imshow(I2);
subplot(1,3,2), imshow(log(abs(F)),[]);
subplot(1,3,3), imshow(angle(F),[]);
pause;
%-----
fprintf ('\nTransform I(v,u)=sin(2*pi/size*(1*u + 0*v))*0.5 + 0.5 ...');
size = 32;
I = zeros(size,size);
for v=1:size
    for u=1:size
        I(v,u) = 0.5 * cos(2*pi/size*(0.5*u + 0*v)) + 0.5;
    end
end
F = fftshift(fft2(I, size, size));
subplot(1,3,1), imshow(I);
subplot(1,3,2), imshow(log(abs(F)),[]);
subplot(1,3,3), imshow(angle(F),[]);
pause;
...

```

Rücktransformation nach manipuliertem Amplitutenspektrum

Wenn Sie das Amplitudenspektrum manipuliert haben, können Sie das FT wie folgt wieder zusammensetzen:

```

F = fft2(I);
amp = abs(F);
pha = angle(F);
... % entfernen der Störfrequenzen
O = amp .* exp(i*pha); % = amp.*cos(pha) + i*amp.*sin(pha)

```

7.1.8 Übung: 2D-Fourier-Transformation mit Matlab und ImageJ



Übung 1: Filterung im Frequenzraum

1. Lesen Sie die Matlab-Hilfe zu den Befehlen *fft2*, *abs* und *angle*.
2. Öffnen Sie das *Lena128.png* Bild
3. Kreieren Sie ein 3x3 Laplace-Filter
4. Transformieren Sie das Lena-Bild mit *fft2* in den Frequenzraum
5. Transformieren Sie die Filtermaske mit *fft2* in den Frequenzraum:

$$h_fft = fft2(h, size(I,1), size(I,2));$$
6. Führen Sie die Filterung im Frequenzraum durch, indem Sie die Bild und Maske im Frequenzraum komponentenweise multiplizieren.
7. Transformieren Sie das Resultat mit *ifft2* zurück in den Ortsraum.

8. Zeigen Sie alles an und überprüfen sie die Resultate der beiden Filterungen. Sind sie wirklich gleich?
9. Messen Sie die Zeit für die Filterung im Ortsraum und im Frequenzraum mit den Befehlen `tic` und `toc`. Welche ist schneller? Probieren Sie das Gleiche mit einem grösseren Bild (z. B. Lena512.png)

Übung 2: Inverse Filterung mit Rauschen

1. Studieren Sie das Matlab Skript `Fourier2DFilterInverse.m`. Beachten Sie, wie der inverse Filter `invH` erstellt wird.
2. Zeigen Sie das Amplitudenspektrum des Filters und des inversen Filters mit `mesh` an.
3. Fügen Sie mit dem verunschärften Bild Rauschen hinzu mit dem Befehl `imnoise(I,'gaussian',0, noiseVar)`. Was passiert mit der Rekonstruktion? Probieren sie verschiedene Rauschstärken aus. Probieren Sie auch `Salt & Pepper` Rauschen aus.

Übung 3: Strukturelle Störung entfernen

Im Matlab-Skript `Fourier2DDeNoise.m` wird dem Lena-Bild ein strukturelles Rauschen in Form einer spezifischen Sinusschwingung hinzugefügt.

1. Visualisieren und analysieren Sie das Frequenzspektrum.
2. Entfernen Sie die Störung im Bild mit einem eigenen Algorithmus. Überprüfen Sie, ob der Algorithmus auch mit anderen Störfrequenzen funktioniert.
3. Entfernen Sie die Störung manuell in ImageJ.

7.2 Wavelet-Transformation

Die mathematischen Grundlagen der *Wavelet-Theorie* stammen aus den 30er Jahren. Die heutige Wavelet-Transformation wurde jedoch erst in den späten 80er von den Physikern *Jean Morlet* und *Alex Grossmann* in Frankreich entwickelt. Die Motivation, die zu der Entwicklung führte, lag in den Problemen der Fourier-Transformation begründet. Eine sehr gute, auch für nicht Mathematiker geeignete Einführung in die Theorie und die Geschichte der Wavelets gibt Barbara Burke in ihrem Buch „*Wavelets: Die Mathematik der kleinen Wellen*“ [*Burke97*].

7.2.1 Probleme der Fourier-Transformation

Die Information über das zeitliche Auftreten der einzelnen Frequenzen eines Fourier transformierten Signals ist nicht direkt erkennbar. Diese Zeitinformation ist versteckt in den Phasen der zugrundeliegenden Sinus- und Kosinusschwingungen und wird erst durch deren Überlagerung sichtbar. Dieses Manko der Fourier-Transformation sollte durch die nachfolgend beschriebene, gefensterte Fourier-Transformation (*Windowed Fourier-Transform WFT*) behoben werden. Die Grundidee liegt darin, die im Signal enthaltenen Frequenzen, abschnittsweise mittels der klassischen Fourier-Analyse zu berechnen, um somit innerhalb dieser Abschnitte (Fenster) Aussagen über die auftretenden Frequenzen machen zu können. Diese sind dann abschnittsweise zeitlich lokalisiert. Das Fenster wird entsprechend entlang des Signals verschoben, bis das gesamte Signal analysiert ist. Die nachfolgende Abbildung zeigt zwei verschiedene Signalverläufe mit den dazugehörigen Amplitudenspektren:

- Links: zwei überlagerte Schwingungen, die periodisch sind über den gesamten Verlauf.
- Rechts: eine abrupte Frequenzänderung im Signal

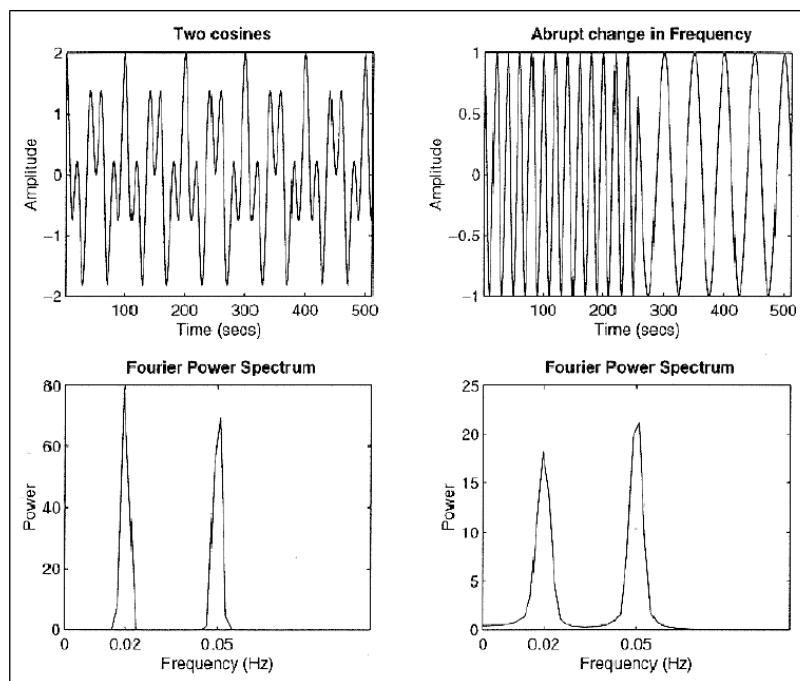


Abb. 182: Zwei Signale mit praktisch identischen Amplitudenspektren [*Rauch01*]

Wir sehen, dass beide Signale die gleichen Frequenzen enthalten. Mit der klassischen Fourier-Analyse kann jedoch nicht unterscheiden werden, wo bzw. wann im Signal die enthaltenen Frequenzen auftreten. Es stellt sich nun die Frage, wie gross soll nun das Analysefenster gewählt werden. Anhand dieser Fragen werden auch die Beschränkungen der *WFT* deutlich: Ein schmales Analysefenster bietet eine sehr gute Zeitauflösung, D. h. plötzliche Signaländerungen sowie Signalspitzen und Sprünge können gut lokalisiert

werden. Nachteil dabei ist dann jedoch, dass die Analyse unempfindlich gegenüber niedrigen Signalfrequenzen wird, welche nur in einem kleinen Bereich durch das Analysefenster abgedeckt sind. Umgekehrt bieten grosse Fenster eine gute Erfassung der vorhandenen Frequenzen, können diese jedoch nicht mehr entsprechend zeitlich lokalisieren.

Die Wavelet-Transformation ermöglicht nun, das Signal in mehreren Auflösungen zu analysieren. Damit können sowohl tiefe Frequenzen mit hoher Frequenzauflösung und schlechter Zeitauflösung als auch rasche Signaländerungen mit hoher Zeitauflösung und niedriger Frequenzauflösung in einer Analyse erfasst werden.

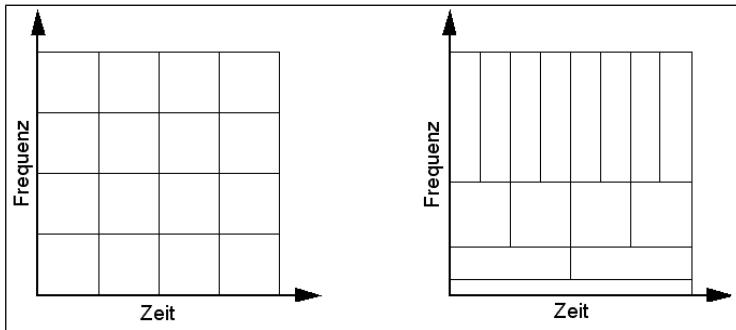


Abb. 183: Starre Frequenz-Zeit Auflösung (WFT) gegenüber der Wavelet-Transformation

7.2.2 Wavelets

Wavelets („kleine Wellen“) sind mathematische Funktionen, die, ähnlich den Sinus- und Kosinus Funktionen der Fourier-Transformation, dazu benutzt werden, eine gegebene Funktion (Signal) auf die in ihr enthaltenen Frequenzbestandteile hin zu untersuchen. Zu diesem Zweck können Wavelets gestreckt bzw. gestaucht (skaliert) werden oder an eine bestimmte Stelle des Signals verschoben werden. Ausgangspunkt einer Wavelet-Transformation ist ein **Basis-** bzw. **Mutter-Wavelet** Ψ (Psi). In der Transformation selbst kommen dann die entsprechend skalierten und örtlich verschobenen Varianten des Basis-Wavelets zum Einsatz. Diese Varianten werden auch allgemein als Wavelet-Familie bezeichnet. Die folgende Abbildung zeigt die **Mexican-Hat-Waveletfamilie**:

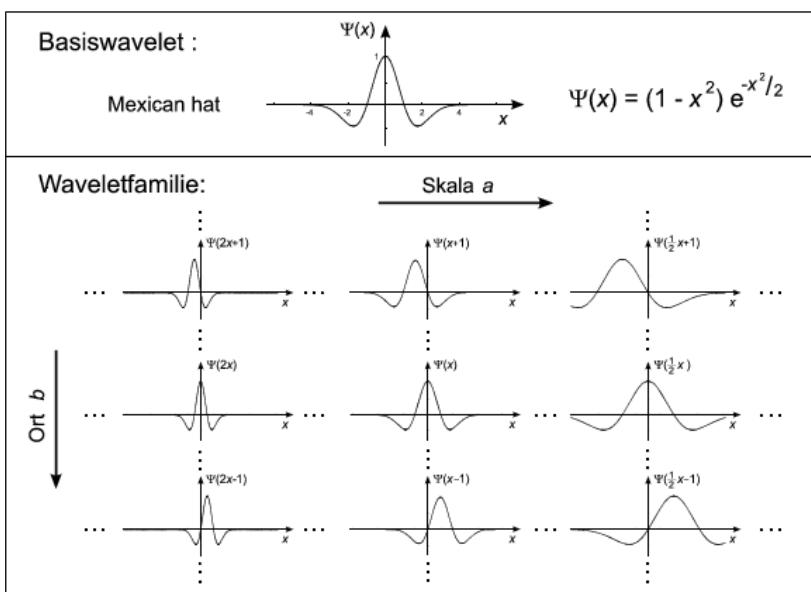


Abb. 184: Die Mexican-Hat-Wavelet Familie mit dem Mutter-Wavelet und seinen Kindern [Rauch99]

Basis-Wavelets Ψ müssen folgende Bedingungen erfüllen:

- Das Wavelet muss in einem begrenzten Intervall ungleich 0 sein.
- Das Integral über dem Wavelet muss null sein: $\int \psi(x) dx = 0$

Ein Wavelet $\psi(a,b)$ der Wavelet-Familie kann beschrieben werden durch:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \quad \text{wobei } a \text{ der Skalierungsfaktor und } b \text{ der Verschiebungsfaktor ist.}$$

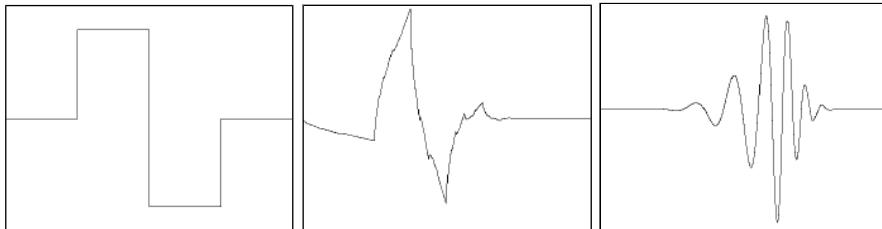


Abb. 185: Beispiele anderer Basis-Wavelets: Haar Wavelet, Daubechies 4 Wavelet und Daubechies 20 Wavelet benannt nach Ingrid Daubechies

7.2.3 Kontinuierliche Wavelet-Transformation

Bei der kontinuierlichen Wavelet-Transformation (*Continuous Wavelet-Transform CWT*) wird das Signal zunächst mit einem stark gestreckten Wavelet untersucht, um die tiefen Frequenzen zu bestimmen. Anschliessend komprimiert man das Wavelet schrittweise - es wird also in verschiedenen Auflösungen dargestellt - um immer feinere Frequenzen zu erfassen. Die durch vollständige Abtastung des Signals mit einem entsprechend skalierten Wavelet entstandene Funktion wird als *Wavelet-Transformierte* bezeichnet. Sie beschreibt also die Übereinstimmung des Signals an allen Stellen mit dem analysierenden Wavelet. Die Rekonstruktion des Ausgangssignals erfolgt analog zur Fourier-Transformation, durch die Addition der entsprechenden Wavelet-Transformierten.

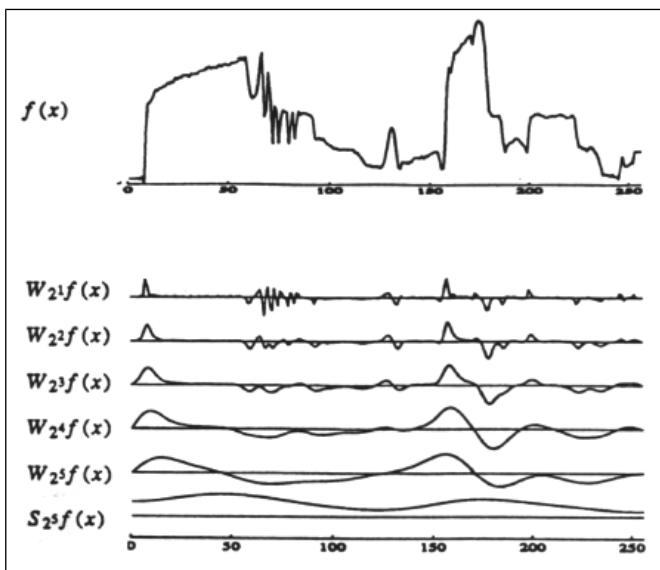


Abb. 186: Signal (oben) und seine Wavelet-Transformationsanteile [Burke97]

Auf Basis der Skalierungs- und Verschiebungs-Parameter a bzw. b , stellt sich die kontinuierliche Wavelet-Transformation als Funktion $CWT(a,b)$ wie folgt dar:

$$CWT(a,b) = \int_{-\infty}^{+\infty} x(t) * \psi_{a,b}(t) dt \quad \text{worin } x(t) \text{ der Wert des Ausgangssignals zur Zeit } t \text{ ist}$$

Wie berechnet man nun den Deckungsgrad eines skalierten und verschobenen Wavelets mit dem darunterliegenden Signal? Dazu multipliziert man die Wavelet-Funktion mit dem überdeckten Signalabschnitt. Diese Multiplikation ergibt eine neue Funktion, über welche man anschliessend das Integral berechnet. Den Wert dieses Integrals bezeichnet man auch als *Wavelet-Koeffizient*.

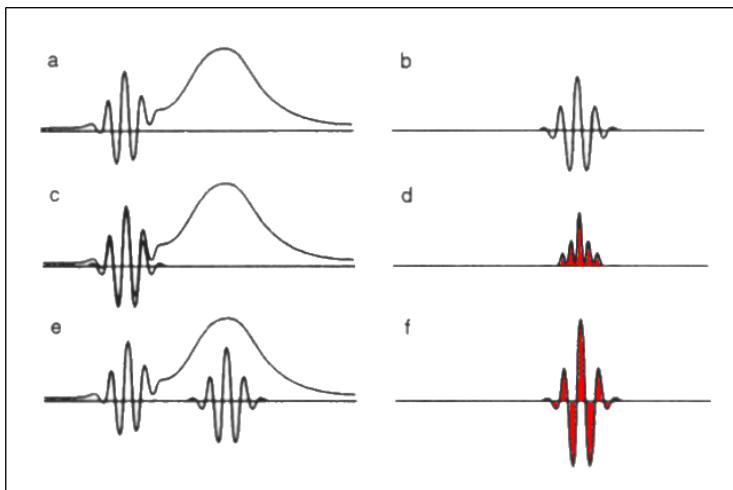


Abb. 187: Vergleich von Signal a mit Wavelet b, indem die beiden im entsprechenden Abschnitt miteinander multipliziert werden. [Burke97]

Die vorangehende Abbildung zeigt, wie ein Wavelet (b) nach und nach mit einer Ausgangsfunktion (a) verglichen wird. In (c) stimmt das Wavelet mit dem gerade überdeckten Abschnitt des Signals annähernd überein. Entsprechend ergibt die Multiplikation von Wavelet und dem Signal eine neue Funktion (d), deren Fläche relativ gross ist und somit ein Mass für gute Übereinstimmung darstellt. Verschiebt man das Wavelet entlang des Signals weiter nach rechts, so ergibt sich die Situation in (e). Die Multiplikation beider Funktionen erzeugt einen sehr kleinen Wavelet-Koeffizienten und zeigt, dass keine Übereinstimmung zwischen Wavelet und dem Signalabschnitt besteht.

Durch die theoretisch mögliche Bestimmung von unendlich vielen Wavelet-Koeffizienten ist die CWT natürlich wegen Überlappung einzelner Wavelets in höchstem Masse redundant und auch sehr rechenintensiv. In der Praxis schränkt man sich deshalb, wie bei der Fourier-Transformation, auf diskrete Abschnitte und Skalierungen ein.

7.2.4 Diskrete Wavelet-Transformation

Die Berechnung der diskreten Wavelet-Transformation (*Discrete Wavelet-Transform DWT*) erfolgt analog zu der CWT, jedoch mit dem Unterschied, dass nun für die Skalen und Zeit Parameter a bzw. b, jeweils diskrete Werte verwendet werden. Dabei gilt es a und b so geschickt zu wählen, dass zum einen nur eine sehr geringe Redundanz bei den Wavelet-Koeffizienten auftritt und zum anderen das Ursprungssignal trotzdem noch vollständig beschrieben und damit auch rekonstruierbar ist.

Auch die Diskretisierung der Wavelet Parameter in der DWT brachte in der Praxis nicht immer den gewünschten Erfolg der Transformationsvereinfachung. Der Durchbruch, der vor allem in der Bildverarbeitung eine stärkere Anwendung der Wavelet-Theorie ermöglichte, gelang 1986 mit der von *Stéphane Mallat* und *Yves Meyer* entwickelten *schnellen Wavelet-Transformation (FWT)*. Ihr Paper „*A theory for multiresolution signal decomposition: The wavelet representation*“ ist das am zweitmeisten zitierte Paper der Computer Vision Wissenschaften.

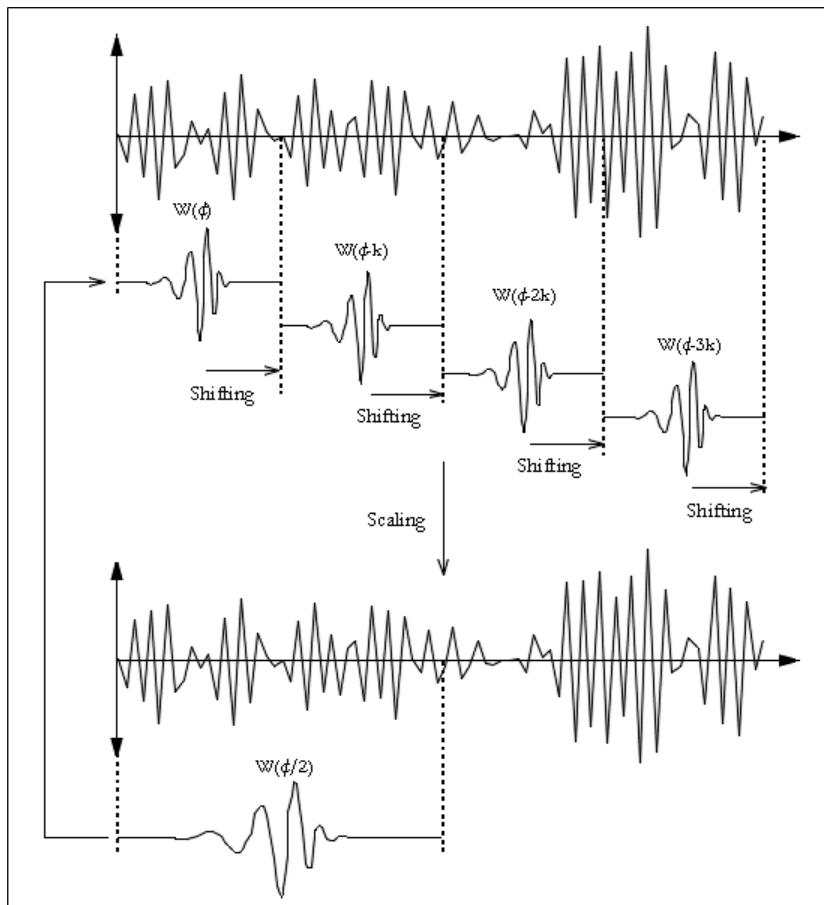


Abb. 188 Verschieben und Skalieren in diskreten Schritten [Altman96]

7.2.5 Haar-Transformation

Die einfachste Wavelet-Transformation ist die *Haar-Transformation* (benannt nach seinem Erfinder Alfred Haar 1909). Weil sie einfach ist, eignet sie sich gut, um das Prinzip zu erklären. Aus demselben Grund ist sie aber auch schnell, weil sie mathematisch eine einfache aber harte Funktion darstellt. Durch Skalierung und Aneinanderreihung des ersten *Haar-Wavelets* erhält man einen ganzen Satz weiterer Funktionen:

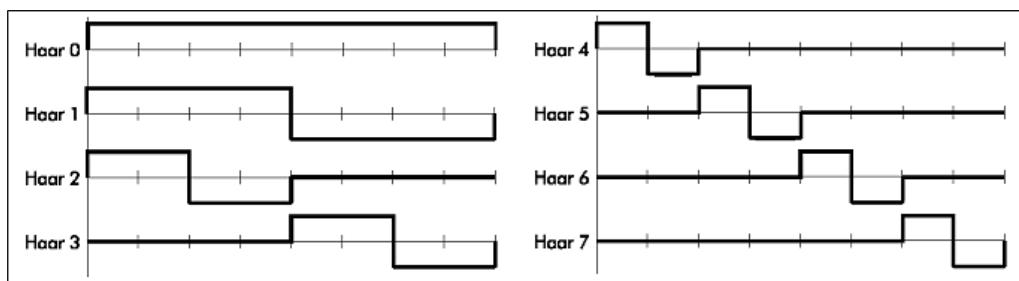
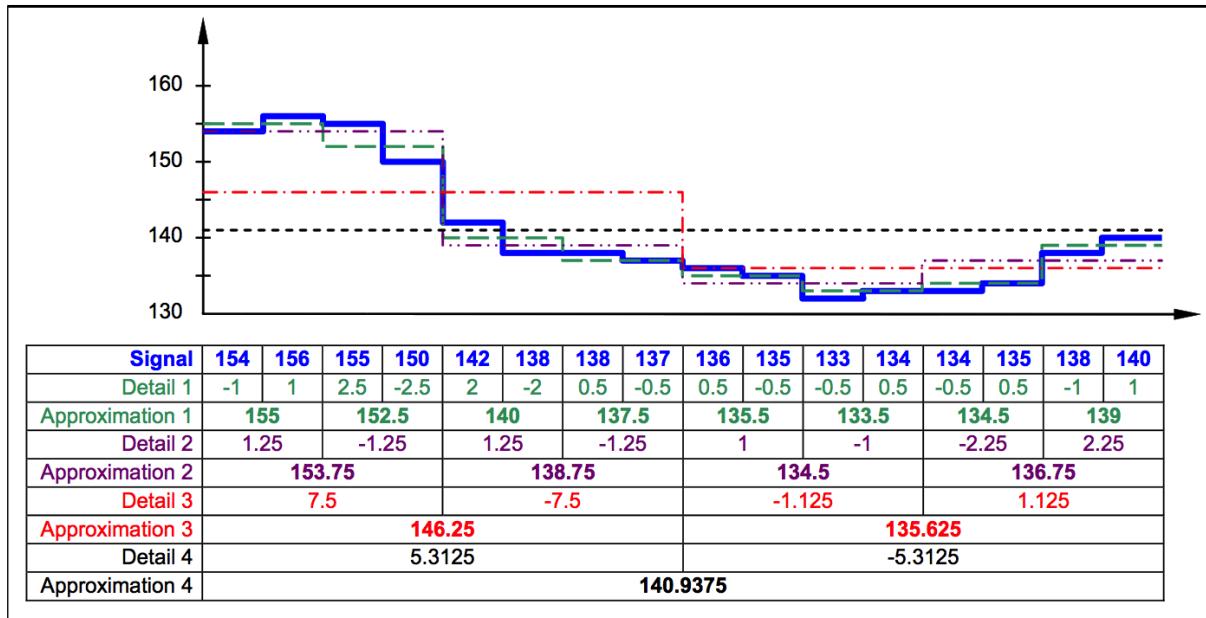


Abb. 189: Haar-Funktionen [Fiedler00]

Durch die Addition dieser sieben Rechteckimpulse mit jeweils verschiedenen Koeffizienten lässt sich jedes beliebige Signal aus acht diskreten Werten darstellen. Führt man die Haar-Funktionen weiter, sind auch noch mehr diskrete Werte darstellbar. Die Haar-Funktion mit der Nummer 0 stellt dabei erneut den Durchschnitt aller Werte dar. Zusammen mit der Funktion 1 beschreibt sie die Durchschnitswerte der 1. und 2. Signalhälfte. Zusammen mit der Funktion 2 und 3 werden diese Signalhälften erneut unterteilt, bis das Signal genau rekonstruiert ist. Der Vorteil der Haar-Transformation ist, dass sie sich lediglich aus Additionen, Subtraktionen und Multiplikationen berechnen lässt.

Man kann eine eindimensionale Haar-Transformation nach einem einfachen Schema berechnen, das *averaging and differencing* heißt. Im folgenden Beispiel soll ein Signal mit

16 Werten Haar-transformiert werden. Wir bilden dazu für jedes benachbarte Wertepaar den Durchschnitt und schreiben diesen in die übernächste Zeile (Approximation 1). In der Zeile dazwischen (Detail 1) schreiben wir jeweils die Differenzen zu den Ausgangswerten. Diesen Schritt können wir 4-mal wiederholen, bis wir nur noch einen Mittelwert von **140.9375** haben:



Die resultierende eindimensionale Haar-Transformation wird wie folgt notiert:

140.9375, 5.3125, 7.5, -1.125, 1.25, 1.25, 1, -2.25, -1, 2.5, 2, 0.5, 0.5, -0.5, -0.5, -1

Wir erhalten einen Durchschnittswert und 15 sogenannte Detailkoeffizienten. Da die Detailkoeffizienten symmetrisch sind, brauchen wir nur einen davon (z. B. den Linken), um die vorangegangene Approximation zu rekonstruieren. Wir erhalten also wieder gleichviel Werte wie das ursprüngliche Signal und können es damit zu 100% rekonstruieren. Wir haben somit keine Information verloren oder hinzugewonnen. Wir sehen auch, dass wir n rekursive Schritte brauchen, um ein Signal mit 2^n Werten zu transformieren. Dieser Prozess, bei dem das Signal in n Auflösungen dargestellt wird, wird auch *Multi-Resolution-Analysis (MRA)* genannt.

Haartransformation als Filteroperation

Die Approximation und Detailbildung kann genauso gut mit einer Filtermaske ausgedrückt werden. Bei diesen Vorgängen, wie bei den beiden Rekonstruktionsschritten handelt es sich um nichts anderes als um eindimensionale Faltungen (2 Multiplikationen + 1 Addition):

$$h_0 = \left[\frac{1}{2}, \frac{1}{2} \right] \text{ Tiefpassfilter (Approximation)} \quad h_1 = \left[\frac{1}{2}, -\frac{1}{2} \right] \text{ Hochpassfilter (Detail)}$$

Beispiel aus obigem Signal für Approximation:

$$[154, 156] * \left[\frac{1}{2}, \frac{1}{2} \right] = \left(154 \cdot \frac{1}{2} \right) + \left(156 \cdot \frac{1}{2} \right) = 155$$

Beispiel aus obigem Signal für Detail links:

$$[154, 156] * \left[\frac{1}{2}, -\frac{1}{2} \right] = \left(154 \cdot \frac{1}{2} \right) + \left(156 \cdot -\frac{1}{2} \right) = -1$$

Wie Sie vielleicht schon vermutet haben, ist der Approximationsfilter ein Tiefpassfilter und der Detailfilter ein Hochpassfilter. Die Rekonstruktion, auch *Synthese* genannt, besitzt folgende Filtermasken:

$$f_0 = [1, 1] \text{ Synthesefilter links}$$

$$f_1 = [1, -1] \text{ Synthesefilter rechts}$$

Beispiel aus obigem Signal für Synthesefilter links:

$$[155, -1] * [1, 1] = (155 \cdot 1) + (-1 \cdot 1) = 154$$

Beispiel aus obigem Signal für Synthesefilter rechts:

$$[155, -1] * [1, -1] = (155 \cdot 1) + (-1 \cdot -1) = 156$$

Zweidimensionale Haar-Transformation

Wendet man diese Haar-Transformation auf jede Zeile und danach auf jede Spalte eines Bildes an, so erhalten wir die *Standard-Dekomposition* der zweidimensionalen Haar-Transformation. Bei der *Non-Standard Dekomposition* werden die Bildzeilen und -spalten abwechselnd transformiert:

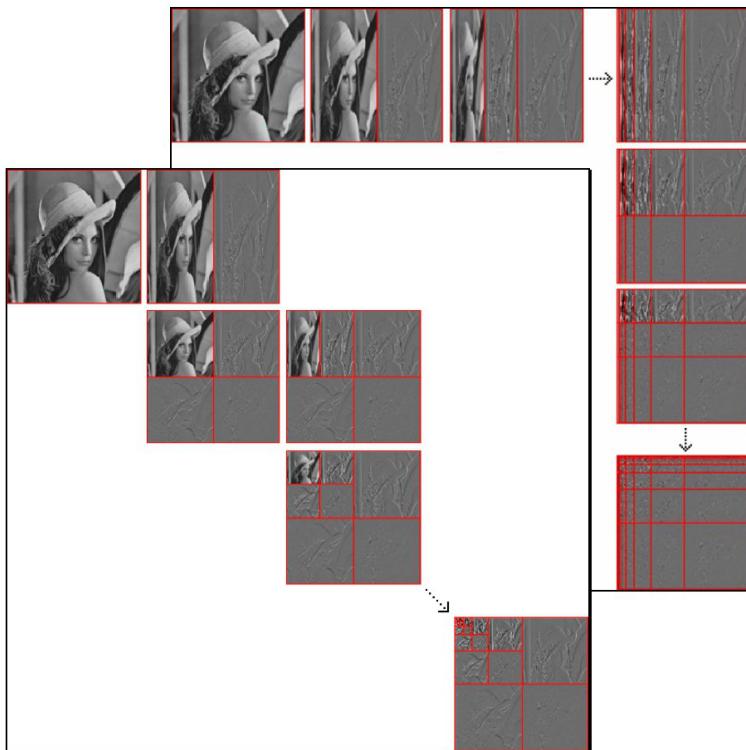


Abb. 190: Non-Standard Dekomposition im Vordergrund, Standard-Dekomposition im Hintergrund

Bilder, die wir nun mit dem Haar-Tiefpass- und Hochpassfilter transformieren neigen zu Blockbildung und starkem Rauschen. Dies kommt daher, dass wir bei der Mittelwertbildung durch 2 dividieren, und zwar zweimal, einmal horizontal und einmal vertikal. Es hat sich gezeigt, dass das Rauschen und die Blockbildung minimiert werden können, wenn wir durch die Wurzel aus 2 dividieren. Dies entspricht einem Skalierungsvektor der Länge 2. Die Filter dieser sogenannten *normalisierten Haar-Transformation* wären demnach:

$$h_0 = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right] \text{ Tiefpassfilter}$$

$$h_1 = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right] \text{ Hochpassfilter}$$

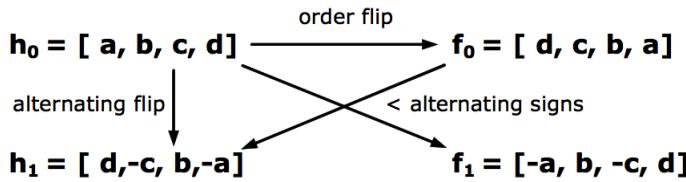
$$f_0 = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right] \text{ Synthesefilter links}$$

$$f_1 = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right] \text{ Synthesefilter rechts}$$

Andere Filtermasken

Andere Wavelet-Transformationen brauchen Filtermasken, die breiter sind als 2. Es braucht aber immer einen Tiefpass-, einen Hochpass- und zwei Synthesefilter. Dieses Set

an Filtern bezeichnet man auch als *Filterbank*. Zwischen den vier Filtern besteht ein einfacher Zusammenhang, sodass wir nur den Tiefpassfilter bestimmen müssen und die drei restlichen Filter nach einem einfachen Schema ableiten können. In einer Filterbank mit der Filterlänge 4 bestehen folgende Zusammenhänge:



In diesem Schema Namens *Alternating flip pattern* bedeuten:

Order flip: Umkehrung der Reihenfolge der Filter-Koeffizienten

Alternating flip: Umkehrung der Reihenfolge und abwechselnde Negation

Alternating signs: Abwechselnde Negation der Filter-Koeffizienten

Konkrete Filterbeispiele:

Haar, normalisiert: (Filterlänge =2)

```
h0 = [ 0.70710678118655,  0.70710678118655]
h1 = [ 0.70710678118655, -0.70710678118655]
f0 = [ 0.70710678118655,  0.70710678118655]
f1 = [ 0.70710678118655,  0.70710678118655]
```

Daubechies 4: (Filterlänge =8)

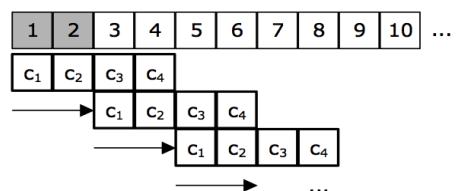
```
h0 = [ 0.23037781330886,   0.71484657055254,   0.63088076792959,  -0.02798376941698,
      -0.18703481171888,   0.03084138183599,   0.03288301166698,  -0.010597401785]

h1 = [-0.010597401785,   -0.03288301166698,   0.03084138183599,   0.18703481171888,
      -0.02798376941698,  -0.63088076792959,   0.71484657055254,  -0.23037781330886]

f0 = [-0.010597401785,   0.03288301166698,   0.03084138183599,  -0.18703481171888,
      -0.02798376941698,   0.63088076792959,   0.71484657055254,   0.23037781330886]

f1 = [-0.23037781330886,   0.71484657055254,  -0.63088076792959,  -0.02798376941698,
      0.18703481171888,   0.03084138183599,  -0.03288301166698,  -0.010597401785]
```

Bei allen Filtern gleich ist ebenfalls die Verschiebung um 2 Einheiten, auch wenn der Filter breiter ist. Bei allen Filtern mit Filterlänge > 2 entstehen dadurch aber Randprobleme, weil alle Werte mehrfach in die Mittelwertbildung einbezogen, werden außer die beiden Werte am linken und rechten Rand.



7.2.6 Anwendungen der Wavelet-Transformation

Bildkomprimierung

Sicher die wichtigste und bekannteste Anwendung der Wavelet-Transformation ist die Komprimierung von Stand- und Videobildern. Sie wurde Mitte 2001 als neuer Standard für verlustbehaftete Komprimierung vom ISO-Komitee verabschiedet. Wir besprechen diesen potenziellen Nachfolger des in die Jahre gekommenen JPEG Standard am Ende des Kapitels über Kompression.

Multi-Skalen-Auflösungen

Überall dort, wo mit unterschiedlichen Auflösungen gearbeitet werden muss, insbesondere wenn die Datenmenge ein Problem darstellt, bietet sich die Wavelet-Transformation an.

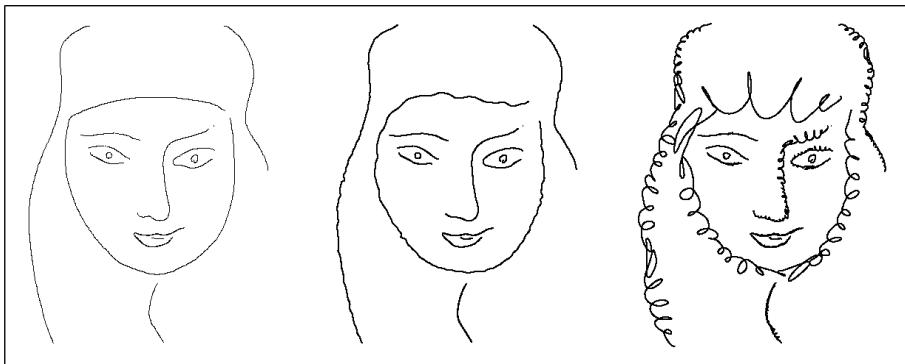


Abb. 191: Vektorgrafik in unterschiedlichen Auflösungen

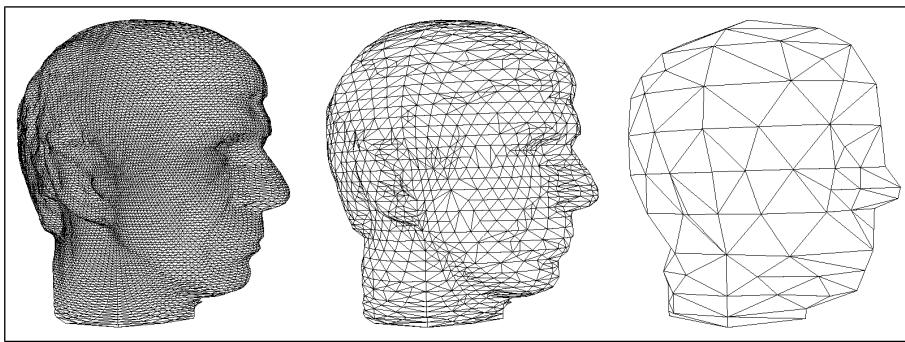


Abb. 192: 3D-Kopf-Modell (Mesh) in unterschiedlichen Auflösungen.

Audioanalyse

Die diskrete Wavelet-Transformation zerlegt ein Signal immer wieder in die Hälfte seiner Auflösung. Ein Mensch hat einen hörbaren Frequenzbereich von 20 Hz bis 20 kHz mit logarithmischer Empfindlichkeit. Der Frequenzbereich von 100 Hz bis 200 Hz wird als gleich gross mit dem Bereich 5000 Hz bis 10'000 Hz empfunden. Genau diese Unterschiede bildet die Wavelet-Transformation sehr schön nach.

7.2.7 Übung Wavelet

1. Nachfolgend sehen Sie die eindimensionale Haar-Transformation eines Signals:

89.6875, 1.6875, -5.875, 3.25, -4.5, -0.25, 2.25, 0.75, -1, -4 , -1, -0.5, 1.5, 1, 0.5, 0

Rekonstruieren Sie das Ausgangssignal mit den Synthesefiltern [1,1] (links) und [1,-1] (rechts).

2. Der erste Wert (89.6875) wird wie bei der Fourier-Transformation DC (Direct-Current = Gleichstrom) genannt. Erklären Sie warum.

7.3 Hough-Transformation

Die *Hough-Transformation (HT)* ist eine globale Transformation, die gebraucht wird, um geometrische Strukturen wie Geraden, Kreise und Ellipsen in einem bereits binarisierten Bild zu detektieren. Der Algorithmus geht auf Paul Hough zurück, der das Verfahren 1962 patentieren liess.

Wie bei allen globalen Transformationen üblich, ist auch die Hough-Transformation in ihrer einfachen Variante eine rechenintensive Aufgabe. Positiv ist, dass sie sehr robust und wenig anfällig gegen Rauschen ist.

7.3.1 Hough-Transformation für Geraden

Die Grundidee hinter der Hough-Transformation ist, dass wir für jedes Vordergrundspixel im Quellbild alle möglichen Geraden, die durch das Pixel gehen können, in einem sogenannten Hough-Raum einzutragen. Dieser Raum hat ebenfalls zwei Dimensionen und eine beschränkte Grösse, sodass wir nicht unendlich viele Geraden eintragen können. Es stellt sich dann die Frage, welche Parameter wir im Hough-Raum für die beiden Dimensionen wählen. Eine erste Variante wären die Konstanten s und b aus der Geradengleichung. Darin ist s ja bekanntlich die Steigung und b der y-Achsendurchgang bei $x=0$:

$$y = mx + b$$

Diese Geradendarstellung hat den Nachteil, dass eine Senkrechte eine unendlich grosse Steigung hat. Deshalb hat man sich für die Darstellung mit der sogenannten *Hessechen Normalform (HNF)* entschieden.

$$r = x \cos(\theta) + y \sin(\theta)$$

Dabei ist r der Abstand der Geraden zum Ursprung und θ den Winkel des Abstands zur x-Achse. Wenn wir diese Darstellung nach y auflösen, erhalten wir wieder die Geradengleichung mit der Steigung und der problematischen Division durch 0 bei $\sin(0)$:

$$y = \left(-\frac{\cos(\theta)}{\sin(\theta)} \right) x + \left(\frac{r}{\sin(\theta)} \right)$$

Für den roten Punkt im kartesischen Koordinatensystem (nachfolgend links) ergeben sich somit unendlich viele Geraden durch diesen roten Punkt. Überträgt man diese Geraden in den *Hough-Raum* mit ihren Parametern r und θ , so ergibt sich eine Sinusoid-Kurve, auf der ein Punkt einer Geraden entspricht:

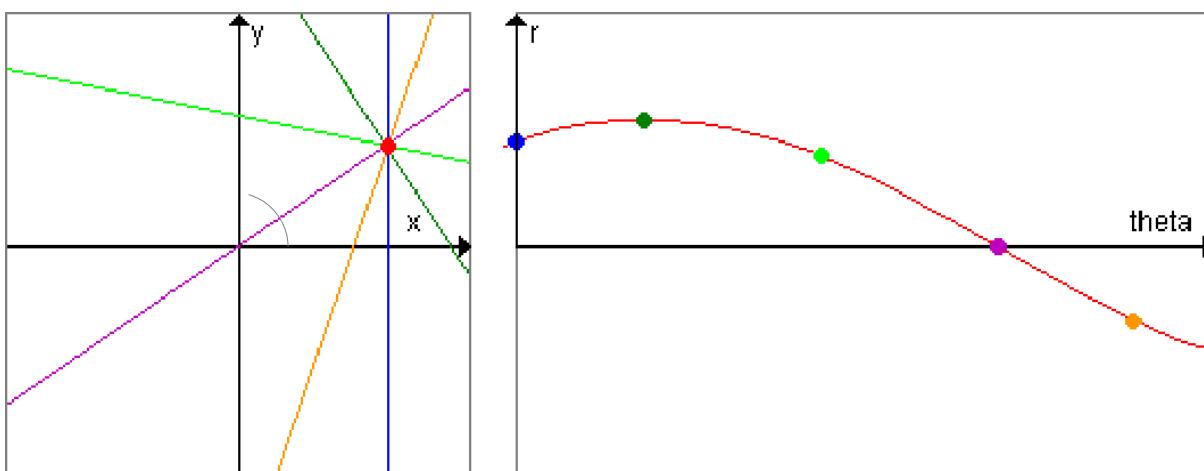


Abb. 193: Links: Kartesischer Raum mit mehreren Beispielgeraden durch einen Punkt. Rechts: Die Parameterkurve für diesen Punkt im Hough-Raum mit den entsprechenden Positionen der Beispielgeraden.
Bilder generiert auf: <http://www.vision.ee.ethz.ch/~cvcourse/brechbuehler/hough.html>

Umgekehrt entspricht ein Punkt im Hough-Raum einer Geraden im kartesischen Raum. Durch diesen Punkt führen unendlich viele Parameterkurven mit unterschiedlichen Radien r und Winkeln θ :

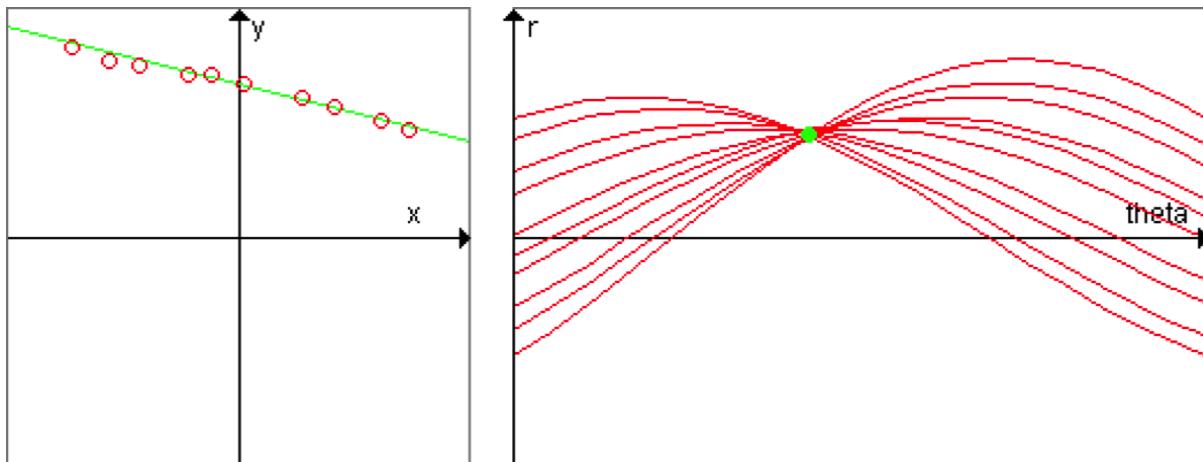


Abb. 194: Links: Kartesischer Raum mit einer Beispielgeraden mit mehreren Punkten, die im Hough-Raum je einer Parameterkurve entsprechen. Diese Bilder wurden mit einem [Java-Applet der ETHZ](#) generiert.

Um nun eine Gerade in einem Bild zu detektieren, wird in der Praxis für jeden schwarzen Punkt im Quellbild eine Kurve in einem diskreten Hough-Raum gezeichnet. Dabei werden die Punkte auf der Kurve nicht einfach gesetzt, sondern inkrementiert. Man nennt diesen diskreten Hough-Raum deshalb auch Akkumulator-Raum. Visualisiert man diesen Raum als Bild, so erhält man ein Graustufenbild mit ganz schwach sichtbaren Kurven. Nur dort, wo sich viele Kurven kreuzen, werden deutlichere Punkte sichtbar. Diese Maxima entsprechen den gesuchten Geraden im Quellbild. Wenn also 3 Geraden gesucht werden sollen, müssen nur die 3 maximalen Werte im Akkumulator-Bild ermittelt werden. Deshalb ist dieses Verfahren unempfindlich gegenüber Rauschen im Bild und Lücken in den Geraden.

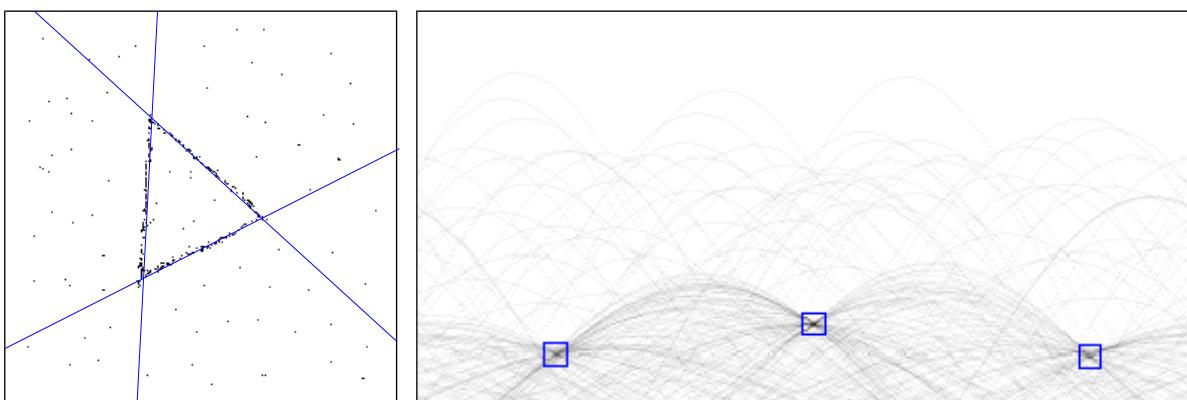


Abb. 195: Links: Quellbild mit 3 Geraden die im diskreten Hough-Raum (hier negativ dargestellt) den 3 Maxima entsprechen. Diese Bilder wurden mit einem [Java-Applet der Uni Braunschweig](#) generiert.

7.3.1.1 Bildung des Hough-Raums

Der Algorithmus um den Hough-Raum zu bilden ist einfach aber trotzdem rechenintensiv. Wir gehen dabei durch alle Pixel des Quellbildes und berechnen für jedes Vordergrundspixel und davon für jeden Winkel θ den Radius r . Für jedes Winkel-Radius-Paar inkrementieren wir die entsprechende Zelle in einem Akkumulator-Array.

```
// Set up Hough space
final int nAng = 256;
final int nRad = 256;
int xC = ip.getWidth()/2;
int yC = ip.getHeight()/2;
double dAng = (Math.PI/nAng);
// number of angels
// number of radii
// x-coordinate of image center
// y-coordinate of image center
// step size of angle
```

```

double rMax = Math.sqrt(xC*xC + yC*yC); // max. radius (center to corner)
double dRad = (2*rMax)/nRad;           // step size radius
int[][][] hough = new int[nAng][nRad];   // Hough accumulator space

// Fill Hough array
for (int y = 0; y < ip.getHeight(); y++)
{
    for (int x = 0; x < ip.getWidth(); x++)
    {
        if (ip.getPixel(x,y) > 0)
        {
            int cx = x-xC, cy = y-yC;

            // Calculate for all angles the radius r & increment the hough array
            for (int t = 0; t < nAng; t++)
            {
                double theta = dAng * t;
                int r = (int)((cx*Math.cos(theta) +
                               cy*Math.sin(theta)) / dRad) + nRad/2;
                if (r >= 0 && r < nRad) hough[t][r]++;
            }
        }
    }
}

```

Diese Dreifachschlaufe ist natürlich aufwendig. Sie könnte beschleunigt werden, indem man die Sinus- und Kosinusaufzüge vorgängig für alle möglichen Winkel theta vorausberechnet. Zudem liesse sie diese Schlaufe optimal parallelisieren.

7.3.1.2 Geraden im Hough-Raum bestimmen

Je mehr Pixel auf einer Geraden liegen, umso höher ist die entsprechende Zelle (Winkel/Radius). Um die n Geraden mit den meisten Pixeln zu finden, müssen wir also die n Zellen mit den höchsten Werten des Hough-Raums finden. Dabei entsteht das Problem, dass z. B. ein Nachbarpixel des höchsten Bergs einen höheren Wert hat als der zweit höchste Gipfel im Hough-Raum. Um das zu verhindern, müssen wir den Hough-Raum zuerst mit einem Verfahren namens **Non-Maximum-Suppression** bearbeiten. Dabei geht man mit einer Maske (z. B. der Größe 3x3) über das Bild und schreibt den Wert des Zentrumspixels in das Zielbild, wenn das Pixel den max. Wert innerhalb der Maske hat. Hat das Zentrumspixel nicht den max. Wert unterdrückt man das den Pixelwert und schreibt eine 0 ins Zielbild.

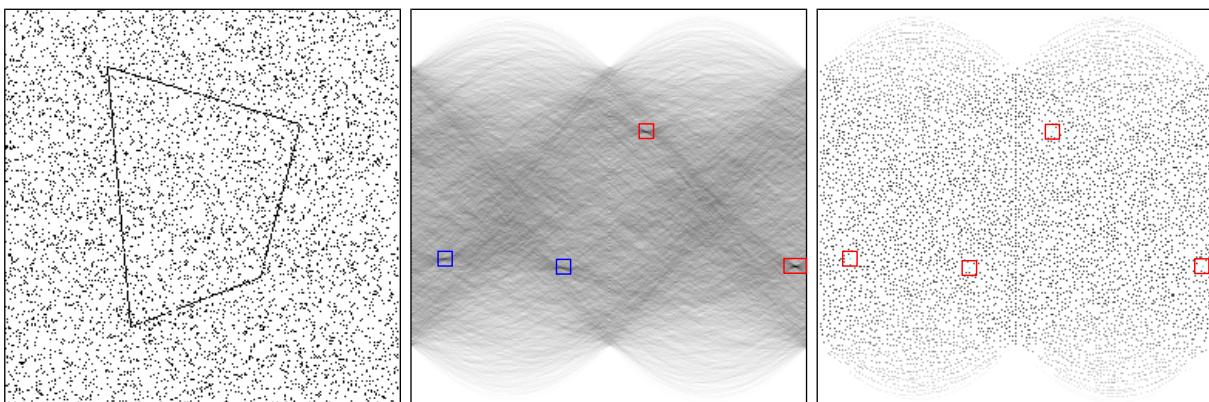
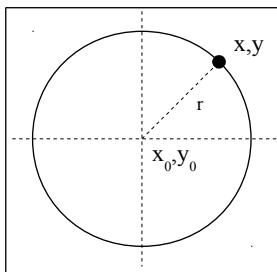


Abb. 196: v.l.n.r.: Polygon, Hough-Raum mit den dunkelsten vier Pixeln rot markiert. Diese entsprechen den beiden längsten Linien. Die beiden kürzeren wären bei den blau markierten Orten. Durch Non-Maximum-Suppression erhalten wir alle vier Maximalwerte.

7.3.2 Hough-Transformation für Kreise

Während Geraden in 2D durch zwei Parameter definiert sind, so braucht es für einen Kreis deren Drei. Die Koordinaten des Zentrums x_0 und y_0 sowie der Radius r . Die Hough-Transformation für Kreise (**CHT** = *Circular Hough Transform*) verwendet die übliche Kreisgleichung:



$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$$

Abb. 197: Kreispunkt bei x, y mit Kreiszentrum bei x_0, y_0 und Radius r

Bei unbekanntem Radius muss ein dreidimensionaler Akkumulatorraum aufgebaut werden, der gleich gross ist, wie das Bild und n Ebenen für jeden zu suchenden Radius aufweist. In einer Akkumulatorebene für einen bestimmten Radius wird für jedes schwarze Pixel ein Kreis gezeichnet. Dabei werden wiederum bestehende Werte nicht überschrieben, sondern auf akkumuliert.

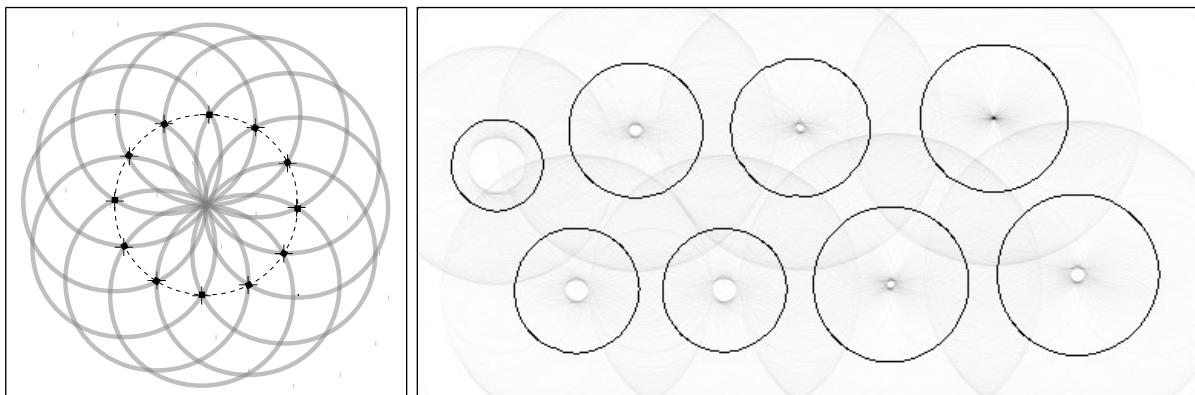
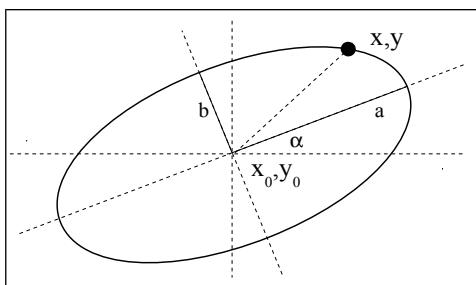


Abb. 198: Hough-Transformation für Kreise: Links: Bei 12 schwarzen Pixeln eines Kreises werden 12 Kreise (Grau) in der Akkumulatorebene aufakkumuliert. Stimmt der Radius der Akkumulatorebene mit dem Radius des Kreises überein, so ergibt sich der höchste Wert im Zentrum des Kreises. Rechts: Eingangsbild mit Akkumulatorbild überlagert. Gesucht wurde ein Kreis mit einem Radius wie vom Kreis oben rechts. Obwohl mehrere andere Kreise einen ähnlichen Radius aufweisen, ergibt sich nur für den gesuchten Kreis ein klares Maximum im Akkumulator-Bild.

Um die Zellen zu bestimmen, durch die ein Kreis verläuft, kann die [Kreisvariante des berühmten Bresenham-Algorithmus](#) verwendet werden.

7.3.3 Hough-Transformation für Ellipsen

Der Hough-Raum für Ellipsen muss sogar 5-dimensional sein, denn eine Ellipse ist durch ihr Zentrum x_0 und y_0 , die beiden Radien a und b sowie durch den Rotationswinkel α definiert:



$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} x_0 + a \cos t \cos \alpha - b \sin t \sin \alpha \\ y_0 + a \cos t \sin \alpha + b \sin t \cos \alpha \end{pmatrix} \text{ mit } 0 \leq t \leq 2\pi$$

Abb. 199: Ellipsenpunkt bei x, y mit Zentrum bei x_0, y_0 , den Radien a und b sowie dem Winkel α .

Ohne Einschränkung des Parameterraumes kann die hohe Dimensionalität schnell zu einem Speicherproblem führen. Bei einer Auflösung von nur $128 = 2^7$ Schritten in jeder Dimension ergeben sich bereits 2^{35} Akkumulator-Zellen, was bei einer Implementierung als 4-Byte-Integers einem Speicherbedarf von 2^{37} Bytes bzw. 128 Gigabytes entspricht.



7.3.4 Hough-Transformation in Matlab

Matlab kennt für die Hough-Transformation und deren Auswertung die 3 Befehle `hough`, `houghpeaks` und `houghlines`. Das nachfolgende Matlab-Skript `HoughLine1.m` extrahiert die stärksten 5 Geraden aus einem rotierten Bild und zeichnet die längsten zusammenhängenden Segmente:

```
I = imread('circuit.tif'); % Load image
rotI = imrotate(I,33,'crop'); % Rotate image 33 degrees
BW = edge(rotI,'canny'); % Edge detection with Canny algorithm
imshow(BW); % Show edge image

% Transform into hough space H with all thetas T and rhos R
[H,T,R] = hough(BW, 'ThetaResolution',0.5, 'RhoResolution', 1);

% Show the hough space with T & R as axis scales
figure, imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel ('\rho');
axis on, axis normal, hold on;
colormap(jet);

% Get the thetas & rhos of the 5 brightest peaks
P = houghpeaks(H,5);
x = T(P(:,2)); y = R(P(:,1)); % Get the x & y coords out of T & P
plot(x,y,'s','color','white');

% Find lines and plot them
lines = houghlines(BW,T,R,P,'FillGap',5,'MinLength',7);
figure, imshow(rotI), hold on
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

    % Plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

    % Determine the endpoints of the longest line segment
    len = norm(lines(k).point1 - lines(k).point2);
    if ( len > max_len)
        max_len = len;
        xy_long = xy;
    end
end

% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
```

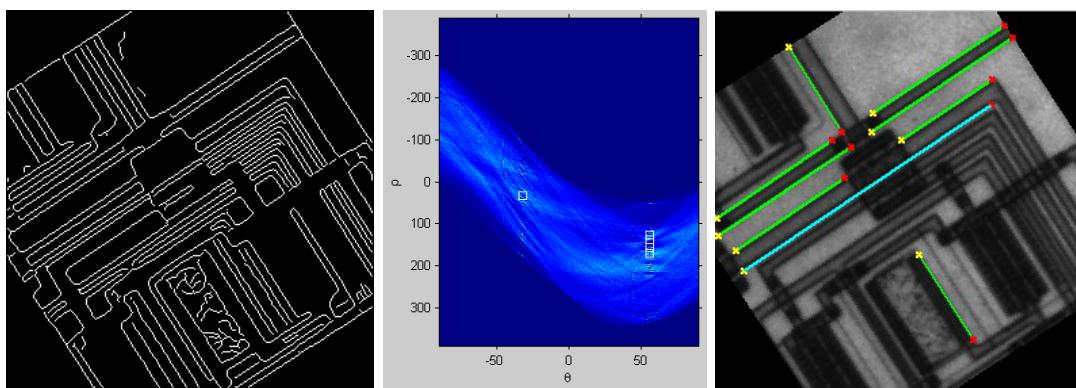


Abb. 200: Links: SW-Kantenbild als Input in die Hough-Transformation. Mitte: Hough-Raum mit den 5 Maxima. Rechts: Originalbild mit den detektierten Kanten.

7.3.5 Übungen zur Hough-Transformation

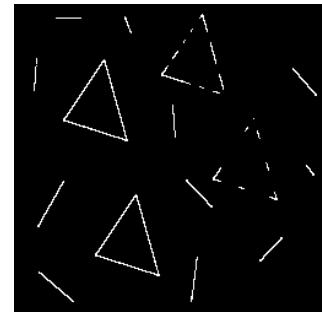
Gehen Sie auf die folgende Webseite:

<http://www.rob.cs.tu-bs.de/content/04-teaching/06-interactive/HNF.html>

1. Zeichnen Sie im kartesischen Raum parallele Linien, was fällt Ihnen auf im Hough-Raum?

2. Wählen Sie mit dem -Knopf folgendes Bild aus:

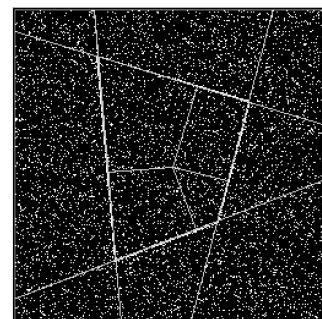
Detektieren Sie mit -Knopf 3 Maxima. Welche Geraden werden detektiert?



3. Wenn man im Hough-Raum eine horizontale Gerade zeichnen könnte, welchen Geraden für im kartesischen Raum würde dies entsprechen?

4. ImageJ und Fiji besitzen von Haus aus keine Hough-Transformation. So müssen wir Sie also selber machen. In der Plugin-Klasse [HoughTransform.java](#) finden Sie die Erstellung des Hough-Raums.

- Implementieren Sie die Non-Maximum-Suppression auf das Array *hough1* in das Array *hough2*.
- Bilden Sie das Histogramm des Arrays *hough2*.
- Berechnen Sie den Radius und Winkel zu den vier Linien des Polygons im Bild [Polygon2.png](#).
- Zeichnen Sie damit die rechtwinkligen Abstände zu den Linien des Polygons.



5. Viele von Hand aufgenommene Bilder sind nicht genau senk- oder waagerecht aufgenommen. Rotieren Sie das Bild [Shuttle2.png](#) mit Matlab automatisch so, dass der *Space Shuttle* genau horizontal liegt. Verwenden Sie das Ausgangsskript [HoughLines_Shuttle.m](#).



6. Suchen Sie die Euro-Münzen mit einem 48 Pixel Radius im Bild [euros.png](#). Verwenden Sie dazu das Matlab-Skript [HoughCircle_Coin.m](#). Zur Detektion von Kreisen können Sie die Funktion *HoughCircle* oder die Matlab-Funktion *imfindcircles* verwenden.



7. Ist es möglich beliebige Geometrien mit einer HT zu finden? Das Stichwort hierzu heisst [Generalized Hough Transform \(GHT\)](#).

7.4 Hauptkomponententransformation

Die *Hauptachsentransformation* oder *Hauptkomponentenanalyse* ist ein statistisches Verfahren um mehrdimensionale (*multivariate*) Datenräume in der Dimension zu reduzieren. Sie wird im englischen *Principal Component Analysis (PCA)* oder *Principal Component Transformation (PCT)* genannt und wird in der Bildverarbeitung manchmal auch *Karhunen-Loève-Transformation* bezeichnet. Die Hauptkomponentenanalyse wurde von [Karl Pearson](#) 1901 entwickelt.

In der Bildverarbeitung ist die PCA von grossem Nutzen, weil sie uns mehrdimensionale Bilddaten nach Kontrast sortiert zurückgeben kann. Wie wir im Kapitel über Bildstatistik gesehen haben, kann die Varianz in einem Bild als Mass für den Kontrast verwendet werden.

Mehrdimensional bedeutet hier, dass für ein Bild z.B. mehr als drei Farbkanäle vorhanden sind oder das von einem Bild mehrere Versionen jeweils als eine Dimension interpretiert werden. Grundsätzlich ist es aber ein statistisches Verfahren, dessen einzige Bedingung ist, dass die Dimensionen gleich gross sind und dass ein Wert in einer Dimension mit den Werten an derselben Position in den anderen Dimensionen korrespondiert. Alle Werte an derselben Position ergeben zusammen einen Merkmalsvektor.

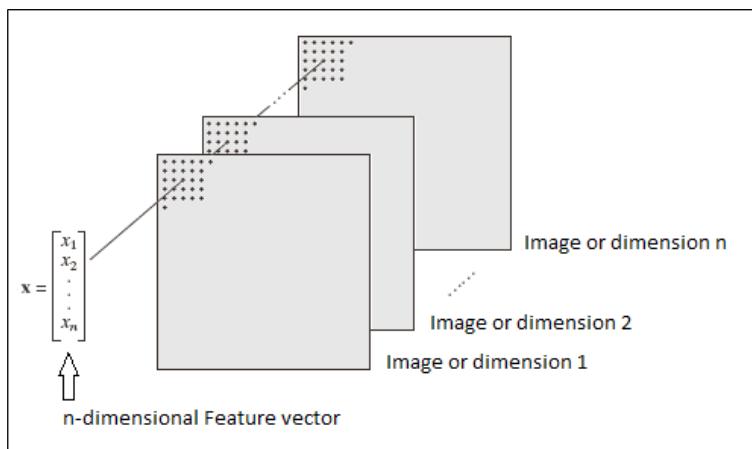


Abb. 201: n-dimensionaler Merkmalsvektor x aus allen Elementen an derselben Position in den n Dimensionen.

Die eigentliche PCA besteht schliesslich aus einer einfachen Matrixmultiplikation zwischen einer linearen Matrix E aus Eigenvektoren in den Zeilen und der Datenmatrix X mit den n-dimensionalen Merkmalsvektoren in den Spalten:

$$X_{PC} = E \cdot X = \begin{bmatrix} e_{1x}, e_{1y}, \dots, e_{1n} \\ e_{2x}, e_{2y}, \dots, e_{2n} \\ \dots, \dots, \dots \\ e_{mx}, e_{my}, \dots, e_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_{11}, x_{12}, \dots, x_{1m} \\ x_{21}, x_{22}, \dots, x_{2m} \\ \dots, \dots, \dots \\ x_{n1}, x_{n2}, \dots, x_{nm} \end{bmatrix}$$

Es gibt zwei Methoden, mit der die Transformationsmatrix bestimmt werden kann:

- Mit der Eigenwert- & Eigenvektorbestimmung über die Kovarianzmatrix
- Mit der [Singulärwertzerlegung \(Singular Value Decomposition \(SVD\)\)](#)

Wir beschränken uns in der Folge auf die erste Methode, welche anschaulicher erklärt werden kann.

Ähnlichkeit zur Fourier-Transformation

Ähnlich wie bei der Fourier-Transformation (FT), die ein Signal in einen Satz von orthogonalen Sinus- und Kosinusfrequenzen zerlegt, transformiert die PCA ein Signal in einen Satz von orthogonalen Basisvektoren. Der Hauptunterschied ist, dass die FT einen festen Satz von Basisfunktionen verwendet, während die PCA eine lineare Transformationsmatrix aus den Eigenvektoren der Kovarianzmatrix verwendet.

7.4.1 Varianz, Kovarianz und Kovarianzmatrix

Die [Varianz](#) ist ein statistisches Mass für die Verteilung einer Datenmenge. Für eine Datenmenge mit dem Mittelwert \bar{X} ist sie definiert als Summe aller quadrierten Differenzen zum Mittelwert dividiert durch die Anzahl Werte - 1:

$$\text{var}(X) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

Die Varianz kann nur für eindimensionale Datenmengen berechnet werden. Wenn wir mehrdimensionale Daten haben, so können wir mit der [Kovarianz](#) ein Zusammenhangsmass (auch Korrelationsmass) für die Kombination von zwei Dimensionen berechnen. Im Unterschied zur Varianz summieren wir nicht die quadrierte Differenz einer Dimension, sondern das Multiplikat der Differenzen zum Mittelwert beider Dimensionen:

$$\text{cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})$$

Kovarianzwerte lassen sich wie folgt interpretieren:

- Die Kovarianz ist **positiv**, wenn eine *starke positive Korrelation* zwischen den Zahlenwerten besteht. D. h., hohe oder tiefe Werte von X entsprechen hohen oder tiefen Werten von Y.
- Die Kovarianz ist **negativ**, wenn eine *starke negative Korrelation* zwischen den Zahlenwerten besteht. D. h., hohe oder tiefe Werte von X entsprechen tiefen oder hohen Werten von Y.
- Die Kovarianz ist **null**, wenn *keine Korrelation* zwischen den Zahlenwerten besteht.

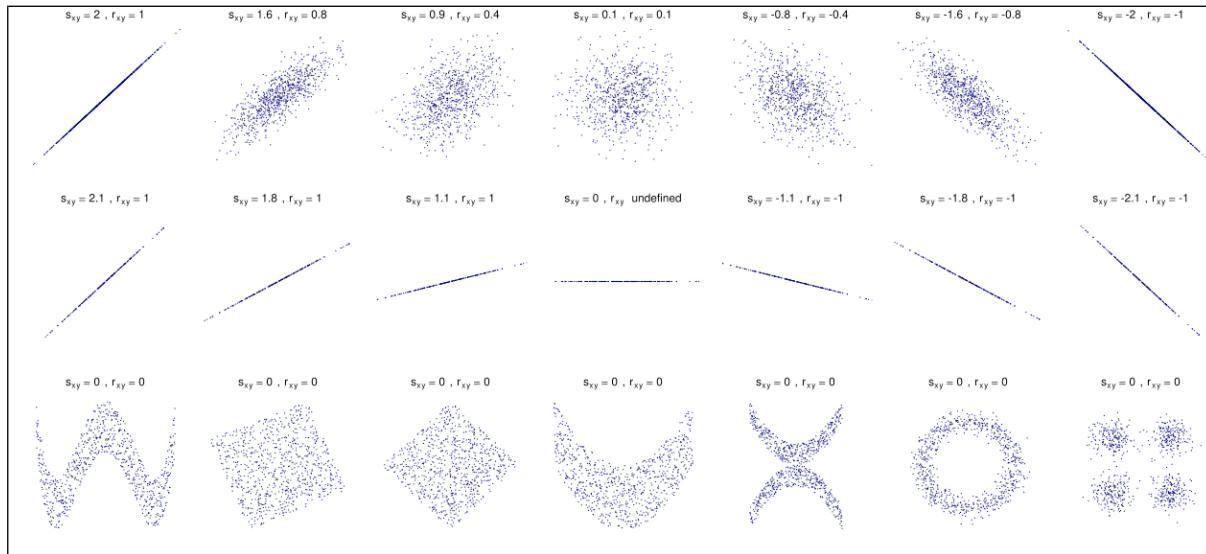


Abb. 202: Kovarianzen s_{xy} von jeweils den x- und y-Werten. Die Werte r geben die Korrelation an, die wir im Kapitel 11.3.2 kennenlernen werden.

In der **Kovarianzmatrix** sind alle Kovarianzen einer n-dimensionalen Datenmenge in einer quadratischen $n \times n$ grossen Matrix zusammengefasst. Weil wir mit der Kovarianz nur den Zusammenhang von zwei Dimensionen erhalten, gibt es z.B. für drei Dimensionen drei Kovarianzen: $cov(X, Y), cov(X, Z), cov(Y, Z)$.

Für n Dimensionen ergeben sich $\frac{n!}{2(n-2)!}$ Kovarianzen.

Für eine zwei- und eine dreidimensionale Datenmenge sähen die Kovarianzmatrizen wie folgt aus:

$$C_{XY} = \begin{bmatrix} cov(X, X), cov(X, Y) \\ cov(Y, X), cov(Y, Y) \end{bmatrix}$$

$$C_{XYZ} = \begin{bmatrix} cov(X, X), cov(X, Y), cov(X, Z) \\ cov(Y, X), cov(Y, Y), cov(Y, Z) \\ cov(Z, X), cov(Z, Y), cov(Z, Z) \end{bmatrix}$$

Wir erkennen dabei drei wichtige Eigenschaften der Kovarianzmatrix:

- Die Kovarianzmatrix ist **symmetrisch**, d.h. alle Kovarianzen kommen doppelt und gespiegelt an der Diagonalen vor ($A=A'$). Die Kovarianz $cov(X, Z)$ ist identisch mit $cov(Z, X)$. Weil die Kovarianzmatrix symmetrisch ist, ist sie diagonalisierbar und die Eigenvektoren stehen senkrecht zueinander.
- Die Kovarianzmatrix enthält auf der Diagonalen die positiven Varianzen der einzelnen Dimensionen, denn $cov(X, X)$ ist nichts anderes als die Varianz von X.

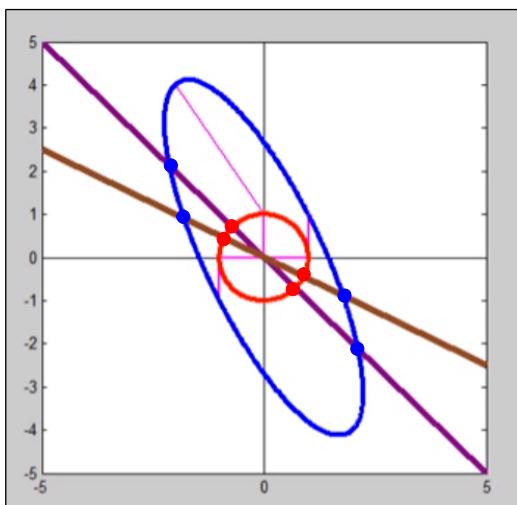
7.4.2 Eigenvektoren und Eigenwerte

Die eben erwähnten Eigenschaften der Kovarianzmatrix erlauben es, daraus die Eigenvektoren und Eigenwerte zu bestimmen. In der linearen Algebra ist ein *Eigenvektor* \vec{v} einer Matrix A ein Vektor, dessen Richtung durch die Multiplikation mit der Matrix nicht verändert wird. Verändert der Eigenvektor dabei nur die Länge, so bezeichnet man den Streckungsfaktor λ (Lambda) als *Eigenwert* der Matrix:

$$A\vec{v} = \lambda \vec{v}$$

Wie sind die Eigenvektoren und Eigenwerte der Matrix zu interpretieren? Wenn wir z.B. die Matrix $\begin{bmatrix} 1, -2 \\ 1, 4 \end{bmatrix}$ als geometrische Transformation interpretieren, dann werden alle

Punkte des roten Einheitskreises in der nachfolgenden Abbildung auf die Punkte der blauen Ellipse abgebildet. Wir sehen, dass nach der Transformation nur vier Punkte dieselbe Richtung haben wie vor der Transformation. Diese Punkte liegen in den Richtungen der beiden Eigenvektoren $[-0.71, 0.71]$ und $[-0.89, 0.45]$. Die 4 blauen Punkte entstehen durch die Skalierung der Eigenvektoren mit den Eigenwerten 3 und 2 resp. -3 und -2. Im Unterschied zu einer Kovarianzmatrix, die symmetrisch ist, stehen hier die Eigenvektoren nicht senkrecht zueinander.



$$\begin{bmatrix} 1, -2 \\ 1, 4 \end{bmatrix} \cdot \begin{bmatrix} -0.71 \\ 0.71 \end{bmatrix} = \begin{bmatrix} -2.1 \\ 2.1 \end{bmatrix} = 3 \begin{bmatrix} -0.71 \\ 0.71 \end{bmatrix}$$

$$\begin{bmatrix} 1, -2 \\ 1, 4 \end{bmatrix} \cdot \begin{bmatrix} -0.89 \\ 0.45 \end{bmatrix} = \begin{bmatrix} -1.8 \\ 0.89 \end{bmatrix} = 2 \begin{bmatrix} -0.89 \\ 0.45 \end{bmatrix}$$

Abb. 203: Transformation aller roten Punkte auf die blauen Punkte. Nur vier blaue Punkte haben dieselbe Richtung wie ihre ursprünglichen Punkte vor der Transformation. Diese Punkte liegen in Richtung der Eigenvektoren. Quelle: [Jonathan Mitchell](#)

Im nachfolgenden Beispiel wird die Transformation dargestellt, die durch die Matrix $\begin{bmatrix} 2, 1 \\ 1, 2 \end{bmatrix}$ entsteht.

Die Eigenvektoren sind $[1, 1]$ und $[1, -1]$ und die Eigenwerte sind 3 und 1:

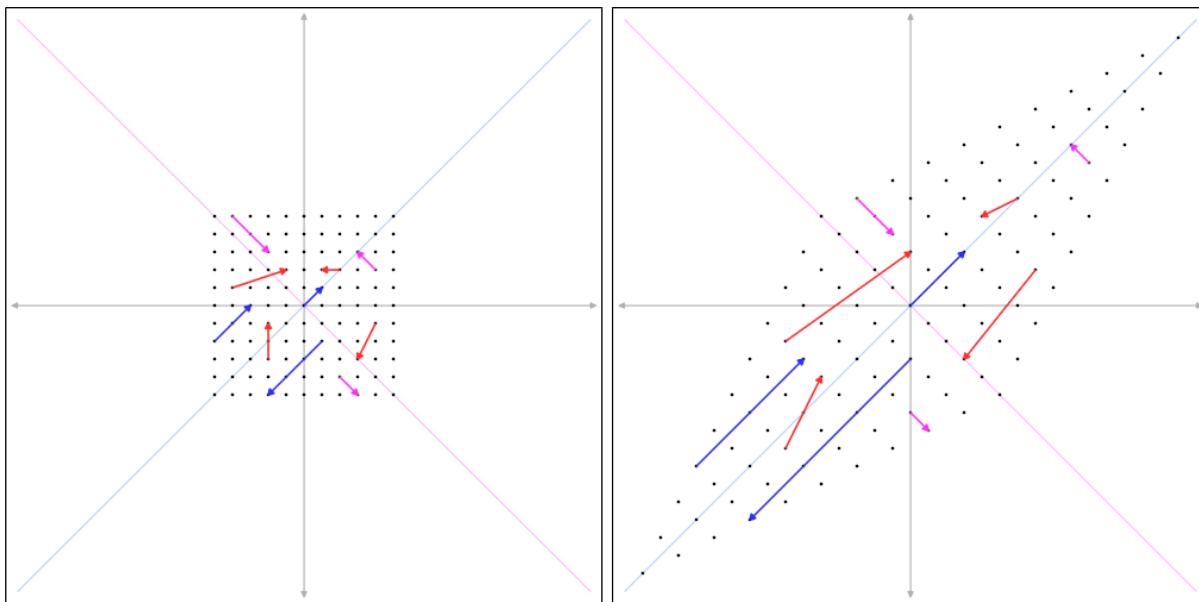


Abb. 204: Alle blauen und rosa Vektoren parallel zu den Eigenvektoren (dargestellt als hellrosa & hellblaue Diagonalen) ändern die Richtungen durch die Transformation nicht. Die blauen Vektoren werden durch den Eigenwert 3 skaliert. Die rosa Eigenvektoren bleiben gleich gross, weil deren Eigenwert 1 ist. Die roten Vektoren ändern die Richtung und sind deshalb keine Eigenvektoren. (Quelle: [Wikipedia](#))

Eine quadratische, symmetrische und damit diagonalisierbare $n \times n$ -Matrix, wie die Kovarianzmatrix, hat also immer n Eigenvektoren und Eigenwerte.

Berechnung der Eigenvektoren und Eigenwerte

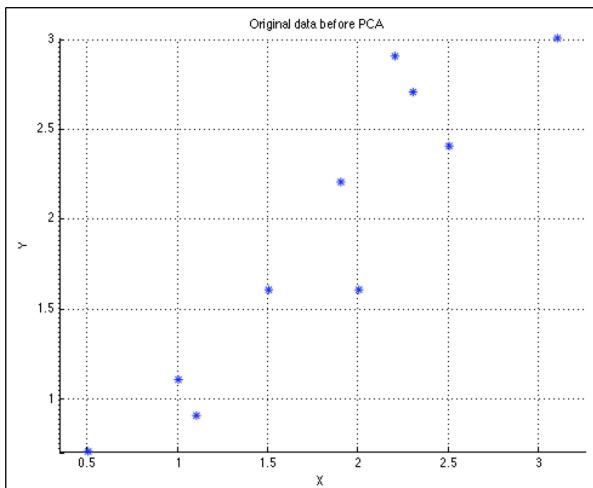
Bei kleinen Matrizen bis 4×4 können die Eigenvektoren und Eigenwerte exakt berechnet werden. Bei grösseren Matrizen ist dies nicht möglich, sodass aufwendige Verfahren der numerischen Mathematik angewendet werden müssen. Wir gehen an dieser Stelle nicht weiter darauf ein und verweisen auf Fachbücher der linearen Algebra. Eine Kurzfassung für die Berechnung von kleinen Matrizen finden Sie bei [Wikipedia](#).

7.4.3 Einfaches Beispiel mit Matlab



Im Matlab-Skript [PCA1_Simple.m](#) wird zuerst ein ganz einfaches Datenset mit X- und Y-Koordinaten als Spalten einer Matrix XY erstellt und davon die Kovarianzmatrix sowie die Eigenvektoren und Eigenwerte berechnet. Das Beispiel ist einem Tutorial von [Smith02] nachempfunden.

Im nachfolgenden Diagramm sehen Sie 10 Punkte der Variablen X und Y dargestellt. Man sagt, dass die beiden Variablen [positiv korrelieren](#), weil sie entlang einer Geraden verteilt sind. Die X-Variable könnte z.B. das Gewicht von Personen sein, während die Y-Variable die Grösse der Personen darstellt. Wenn eine Person grösser ist, dann wird sie ja wahrscheinlich auch schwerer sein.



$$XY = \begin{bmatrix} 2.5, 2.4 \\ 0.5, 0.7 \\ 2.2, 2.9 \\ 1.9, 2.2 \\ 3.1, 3.0 \\ 2.3, 2.7 \\ 2.0, 1.6 \\ 1.0, 1.1 \\ 1.5, 1.6 \\ 1.1, 0.9 \end{bmatrix}$$

Abb. 205: Korrelierender Datensatz mit den Koordinatenpaaren in den Zeilen der Matrix XY.

Schritt 1: Eigenvektoren & Eigenwerte aus der Kovarianzmatrix

Wir erhalten die Kovarianzmatrix mit `cov` und die Eigenwerte und Eigenvektoren als Einheitsvektoren mit `eig` ebenfalls als Matrizen:

$$C = \begin{bmatrix} 0.6166, 0.6154 \\ 0.6154, 0.7166 \end{bmatrix}$$

$$eigvec = \begin{bmatrix} -0.7352, 0.6779 \\ 0.6779, 0.7352 \end{bmatrix} \quad eigval = \begin{bmatrix} 0.0491, 0 \\ 0, 1.2840 \end{bmatrix}$$

```
% Create the covariance matrix C out of the data XY
C = cov(XY(:,1), XY(:,2))

% Get the eigenvectors and eigenvalues from the covariance matrix
[eigvec, eigval] = eig(C)
```

Schritt 2: Eigenvektoren & Eigenwerte sortieren

Für die weitere Verarbeitung brauchen wir die Eigenwerte und Eigenvektoren nach Grösse sortiert. Wir sortieren dazu die Eigenwerte auf der Diagonalen der Matrix `eigval` und sortieren die Eigenvektoren in derselben Reihenfolge. Der Eigenvektor mit dem grössten Eigenwert steht nun in der linken Spalte:

$$eigval = \begin{bmatrix} 1.2840 \\ 0.0491 \end{bmatrix} \quad eigvec = \begin{bmatrix} 0.6779, -0.7352 \\ 0.7352, 0.6779 \end{bmatrix}$$

```
% The eigenvalues are on the diagonal of the eigenval matrix
eigval = diag(eigval);

% We sort the eigenvalue to have the longest first
[eigval, indices] = sort(eigval, 'descend')
eigvec = eigvec(:, indices)
```

Um die Eigenvektoren zu visualisieren, berechnen wir den Mittelpunkt der Daten mit `mean` und skalieren die Eigenvektoren mit den Eigenwerten:

```
%plot the datasets
plot(XY(:,1), XY(:,2), 'b*');

% Calculate mean of x & y for plotting the major axis
mn = mean(XY)';
```

Schritt 3a: PCA mit allen Komponenten

Nun kommen wir zur eigentlichen PCA, bei der wir zuerst die Daten in XY in den Ursprung verschieben, indem wir von allen Koordinaten den Mittelwert abziehen. Zusätzlich transponieren wir die Daten, sodass die einzelnen Koordinatenpaare als Spalten erscheinen:

$$XY0^T = \begin{bmatrix} 0.69, -1.31, 0.39, 0.09, 1.29, 0.49, 0.19, -0.81, -0.31, -0.71 \\ 0.49, -1.21, 0.99, 0.29, 1.09, 0.79, -0.31, -0.81, -0.31, -1.01 \end{bmatrix}$$

Auch die sortierten Eigenvektoren transponieren wir, sodass der grösste Eigenvektor nun in der oberen Zeile erscheint:

$$eigvec^T = \begin{bmatrix} 0.6779, 0.7352 \\ -0.7352, 0.6779 \end{bmatrix}$$

Die Hauptkomponententransformation wird nun durch Multiplikation der Eigenvektormatrix mit den in den Nullpunkt verschobenen Daten erreicht. Bei einer Matrixmultiplikation muss ja die Zeilenlänge der 1. Matrix mit der Spaltenlänge der 2. Matrix übereinstimmen.

$$XY2 = eigvec^T \cdot XY0^T = \begin{bmatrix} 0.6779, 0.7352 \\ -0.7352, 0.6779 \end{bmatrix} \cdot \begin{bmatrix} 0.69, -1.31, 0.39, 0.09, 1.29, 0.49, 0.19, -0.81, -0.31, -0.71 \\ 0.49, -1.21, 0.99, 0.29, 1.09, 0.79, -0.31, -0.81, -0.31, -1.01 \end{bmatrix}$$

$$XY2 = \begin{bmatrix} 0.8280, -1.7776, 0.9922, 0.2742, 1.6758, 0.9129, -0.0991, -1.1446, -0.4380, -1.2238 \\ -0.1751, 0.1429, 0.3844, 0.1304, -0.2095, 0.1753, -0.3498, 0.0464, 0.0178, -0.1627 \end{bmatrix}$$

Zur Darstellung mit *plot* müssen wir die Daten aber wieder zurück transponieren:

```
% Shift the data to the center by subtracting the mean
XY0 = [XY(:,1)-mn(1) XY(:,2)-mn(2)]; %####

% Do the PCA with all eigenvectors
XY2 = eigvec' * XY0';
%####

% Transpose it for plotting
XY2 = XY2';

% Plot the transformed dataset
plot(XY2(:,1), XY2(:,2), 'r*');
```

Wie Sie in Abb. 206 an den roten Punkten erkennen können, haben wir die Daten durch die Transformation einfach um den Ursprung rotiert, sodass die Hauptachse zur x-Achse geworden ist. Die X-Werte korrelieren damit nicht mehr mit den Y-Werten. Man sagt deshalb auch, dass die PCA n-dimensionale Daten in n orthogonale und *dekorrelierte* Hauptkomponenten zerlegt.

Schritt 3b: PCA mit nur der grössten Komponente

In der zweiten PCA wollen wir nur die Hauptkomponente zur Transformation verwenden. Die Transformationsmatrix besteht damit nur noch aus dem Eigenvektor der Hauptkomponente. Das Resultat wird folglich auch um eine Dimension reduziert. Es fehlen einfach die y-Koordinaten, die ja aus der 2. Komponente kamen. In Abb. 206 sehen Sie, dass die grünen Punkte nun alle auf der Achse der 1. Komponente liegen. Wir haben damit eindeutig Information verloren. Je kleiner die Eigenwerte von Achsen sind, umso kleiner ist der Fehler, wenn wir diese Achsen bei der PCA weglassen. Damit kann die PCA zur verlustbehafteten Datenkompression eingesetzt werden.

$$eigvec(:1)^T = [0.6779, 0.7352]$$

$$XY1 = eigvec(:1)^T \cdot XY0^T = [0.6779, 0.7352] \cdot \begin{bmatrix} 0.69, -1.31, 0.39, 0.09, 1.29, 0.49, 0.19, -0.81, -0.31, -0.71 \\ 0.49, -1.21, 0.99, 0.29, 1.09, 0.79, -0.31, -0.81, -0.31, -1.01 \end{bmatrix}$$

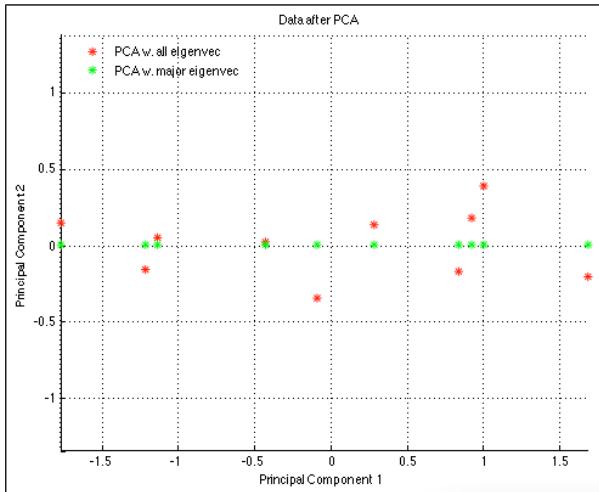
$$XY1 = [0.69, -1.31, 0.39, 0.09, 1.29, 0.49, 0.19, -0.81, -0.31, -0.71]$$

Eine andere Möglichkeit wäre den zweiten Eigenvektor auf null zu setzen. Dann bekämen wir wieder die y-Koordinaten, jedoch alle auf null gesetzt.

```
% Do the PCA with with only the strongest eigenvector
#####
XY1 = eigvec(:,1)' * XY0';
#####

% Transpose it for plotting
XY1 = XY1';

% Plot the transformed PC only with the x coordinate
plot(XY1(:,1), 0 , 'g*');
```



$$XY2 = \begin{bmatrix} 0.8280, -0.1751 \\ -1.7776, 0.1429 \\ 0.9922, 0.3844 \\ 0.2742, 0.1304 \\ 1.6758, -0.2095 \\ 0.9129, 0.1753 \\ -0.0991, -0.3498 \\ -1.1446, 0.0464 \\ -0.4380, 0.0178 \\ -1.2238, -0.1627 \end{bmatrix} \quad XY1 = \begin{bmatrix} 0.8280 \\ -1.7776 \\ 0.9922 \\ 0.2742 \\ 1.6758 \\ 0.9129 \\ -0.0991 \\ -1.1446 \\ -0.4380 \\ -1.2238 \end{bmatrix}$$

Abb. 206: Die roten Punkte nach der PCA mit allen Eigenvektoren und die grünen Punkte nach der PCA mit nur dem grössten Eigenvektor als Transformationsmatrix. Die Werte 1. und 2. Hauptkomponenten sind dekorreliert, da ihre Werte nun nicht mehr voneinander abhängen.

Schritt 4: Rekonstruktion der Daten

Wie bei den meisten Transformationen macht die PCA nur Sinn, wenn wir die Daten auch wieder zurücktransformieren können. Da die Transformation ja auch *Analyse* genannt wird, können wir die Rücktransformation auch als *Synthese* bezeichnen. Bei der *Principal Component Synthesis* können wir einfach die Inverse der PCA verwenden. Da die PCA eine lineare Rotationsmatrix ist, erhalten wir die Inverse einfach durch Transponieren.

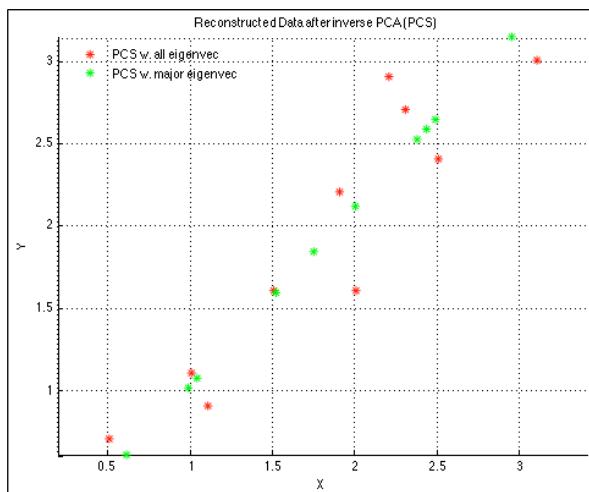
$$\text{inverse}(\text{eigvec}^T) = \text{inverse} \begin{bmatrix} 0.6779, 0.7352 \\ -0.7352, 0.6779 \end{bmatrix} = \begin{bmatrix} 0.6779, -0.7352 \\ 0.7352, 0.6779 \end{bmatrix}$$

Als letzten Schritt müssen wir die Verschiebung in den Ursprung wieder rückgängig machen, damit die Punktewolke wieder an ihrem ursprünglichen Ort zu liegen kommt.

```
% Principal Component Synthesis using all eigenvectors
XY02 = eigvec * XY2'; % eigvec = inv(eigvec')
XY02 = XY02'; % transpose for visualization
XYR2 = [XY02(:,1)+mn(1) XY02(:,2)+mn(2)]; % add back the translation

% Principal Component Synthesis using only one PC
XY01 = eigvec(:,1) * XY1'; % Achtung: you can't invert eigvec(:,1)'
XY01 = XY01'; % transpose for visualization
XYR1 = [XY01(:,1)+mn(1) XY01(:,2)+mn(2)]; % add back the translation

% Plot the reconstructed data
plot(XYR2(:,1), XYR2(:,2), 'r*');
plot(XYR1(:,1), XYR1(:,2), 'g*');
```



$$XY2 = \begin{bmatrix} 2.5, 2.4 \\ 0.5, 0.7 \\ 2.2, 2.9 \\ 1.9, 2.2 \\ 3.1, 3.0 \\ 2.3, 2.7 \\ 2.0, 1.6 \\ 1.0, 1.1 \\ 1.5, 1.6 \\ 1.1, 0.9 \end{bmatrix} \quad XYI = \begin{bmatrix} 2.37, 2.51 \\ 0.61, 0.60 \\ 2.48, 2.63 \\ 1.99, 2.11 \\ 2.94, 3.14 \\ 2.42, 2.58 \\ 1.74, 1.83 \\ 1.03, 1.06 \\ 1.51, 1.58 \\ 0.98, 1.01 \end{bmatrix}$$

Abb. 207: Die roten Punkte in XY2 wurden ohne jeglichen Informationsverlust rekonstruiert. Die grünen Punkte in XY1 wurden alle auf 1. Hauptachse projiziert.

Die Schritte 1-3 bietet Matlab auch in der Funktion `[pc,score,latent] = princomp(X)` an, worin die `pc` die Eigenvektoren, `score` die transformierten Daten und `latent` die Eigenwerte sind. Ab der Version 12b von Matlab existiert auch eine `pca` Funktion, die noch mehr Optionen bietet.

Dieses Beispiel demonstrierte an einem ganz einfachen zweidimensionalen Datensatz die Hauptkomponententransformation und die Rücktransformation einmal mit allen Komponenten und einmal um eine Komponente reduziert. Im realen Einsatz wird die PCA mit höherdimensionalen Daten verwendet, sodass die Reduktion der Dimensionen weniger ins Gewicht fällt als in diesem Beispiel.

7.4.4 Anwendungen in der Bildverarbeitung und Computergrafik

7.4.4.1 Bestimmung der Ausrichtung von 2D- und 3D-Daten

Die am einfachsten nachzuvollziehende Anwendung ist die Bestimmung der Ausrichtung und Ausdehnung einer zwei- oder dreidimensionalen Punktewolke. Dies kann sowohl in der Bildanalyse als Merkmal einer Region dienen (s. Kapitel 10.2.2.1) oder in der 3D-Computergrafik zur Berechnung eines optimal kleinen Hüllvolumens (*Oriented Bounding Box OBB*) eingesetzt werden (s. Computergrafikskript Kapitel 11.2.1).



Im nachfolgenden Matlab-Skript (`PCA2_Eigenvec_Plot.m`) wird mit der Funktion `mvnrnd` eine Punktewolke XY mit 2000 Punkten mit einer zufälligen Gauss-Verteilung erzeugt. Mit der Funktion `cov` erzeugt uns Matlab die Kovarianzmatrix der Datenmenge XY und mit der Funktion `eig` erhalten wir die Eigenvektoren und Eigenwerte. Die Eigenvektoren sind Einheitsvektoren und entsprechen den Richtungen der Hauptachsen einer normal verteilten Zufallsvariablenverteilung. Die Eigenwerte sind die quadrierten Distanzen entlang der Hauptachse bis zur Standardabweichung $\sigma=1$.

```
% create n multivariate random points with normal distribution
% from the covariance matrix sigma around the center point c
c = [0.0; 0.0];
rot = [1.0 1.5;
       1.5 3.0];
XY = mvnrnd(c, rot, 2000);

% Create the covariance matrix C out of the data XY
% This should give roughly the same as the input covariance matrices
C = cov(XY);

% Get the eigenvectors and eigenvalues from the covariance matrix
[eigvec, eigval] = eig(C)
```

```
% The squareroot of the eigenvalues on the diagonal correspond
% to length of the principal components
eigval = sqrt(diag(eigval));

% We sort the eigenvalue to have the longest first
[eigval, indices] = sort(eigval, 'descend')
eigvec = eigvec(:, indices)

%plot the datasets
plot(XY(:,1),XY(:,2), 'b.', 'MarkerSize', 1);

% Get the eigenvectors
e1 = eigvec(:, 1);
e2 = eigvec(:, 2);

% angle to x axis
phil = atan(e1(2)/e1(1)) * 180 / pi

% Build scaled principal axis from the center points
a1 = c + 3*eigval(1) * e1;
a2 = c + 3*eigval(2) * e2;

% Plot the the principal axis
plot([c(1) a1(1)], [c(2), a1(2)], 'k-', 'LineWidth', 2);
plot([c(1) a2(1)], [c(2), a2(2)], 'k-', 'LineWidth', 2);
```

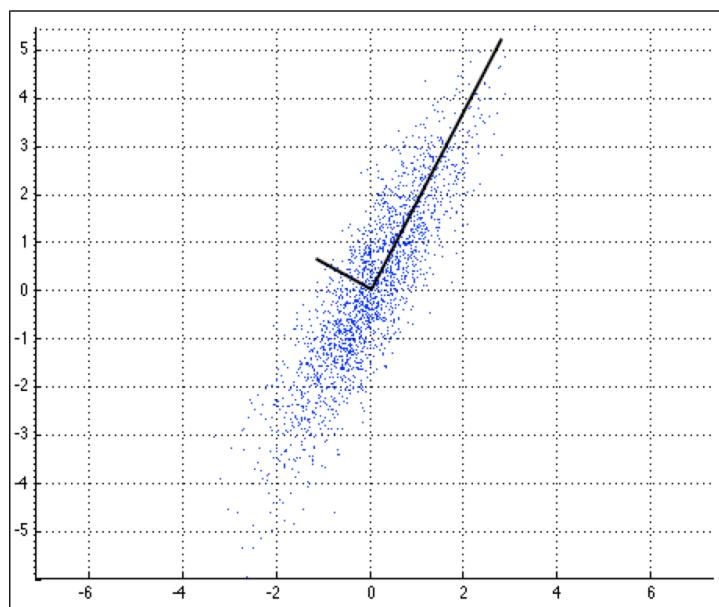


Abb. 208: Die eingezeichneten Eigenvektoren als 1. und 2. Hauptachse sind mit dem Faktor 3 skaliert und entsprechen damit einer [Standardabweichung](#) $\sigma=3$. Damit sind 99.73% aller Punkte einer Gauss-Verteilung abgedeckt.

7.4.4.2 Reduktion der Dimensionalität

Wie eingangs erwähnt, kann die PCA verwendet werden, um hochdimensionale Datensätze in ihrer Dimension zu reduzieren. Satelliten liefern z.B. oft nicht nur die sichtbaren Kanäle im Rot-, Grün- und Blaubereich, sondern auch die Bereiche im nahen, mittleren und thermalen Infrarotbereich. Damit haben wir einen sechsdimensionalen Merkmalsraum, bei dem jedes Pixel durch sechs Zahlen definiert ist. Eine hohe Dimensionalität macht es sowohl schwierig einen Datenraum auszuwerten als auch ihn darzustellen. Genau da setzt die PCA an und liefert uns die Dimensionen eines Datenraumes nach deren Varianz sortiert zurück. Wir werden im Kapitel 11.1.1.1 genauer darauf eingehen, wo es darum gehen wird, aus Satellitenbildern automatisch Gebiete von Stadt, Land und Gewässer zu erkennen. Wir beschränken uns hier auf ein minimales Beispiel, um das Prinzip zu erklären.

7.4.4.3 Statistische Klassifikation

Wenn es in der Bildanalyse darum geht, zu entscheiden, ob ein Bildausschnitt zu einer bestimmten Objektklasse gehört, so können wir dies mit Statistik beantworten, wenn wir wissen, mit welcher Wahrscheinlichkeit gewisse Objekte auftreten. Wir werden im Kapitel 11 über Klassifikation näher darauf eingehen.

7.4.4.4 Mustererkennung

Eine Mustererkennung geschieht meistens, indem man die kürzeste Distanz eines Vergleichsmusters zu mehreren Kandidaten einer bestimmten Vergleichsklasse ermittelt. Die kürzeste, meist euklidische Distanz können wir auch im Hauptkomponentenraum bestimmen. Genau das werden wir im Kapitel 11.2.1.2.2 tun, um ein Gesicht in einer Gesichterdatenbank finden.

8 Segmentierung



Segmentierung ist der erste wichtige Schritt der Bildanalyse nach der Bildvorverarbeitung und vor der Merkmalsextraktion. Ziel der Bildanalyse ist es, Informationen aus Bildern zu gewinnen. Dazu müssen wir meistens in einem Bild etwas suchen. Oftmals will man nur eine bestimmte Region im Vordergrund von der übrigen uninteressanten Region, dem Hintergrund, trennen. Segmentieren bedeutet, jedes Pixel einer bestimmten Region zuzuweisen. Wir unterteilen die verschiedenen Methoden in die Kategorien der *schwellwertbasierten*, der *regionenbasierten*, der *kantenbasierten*, der *modellbasierten* und der *bewegungsbasierten* Verfahren.

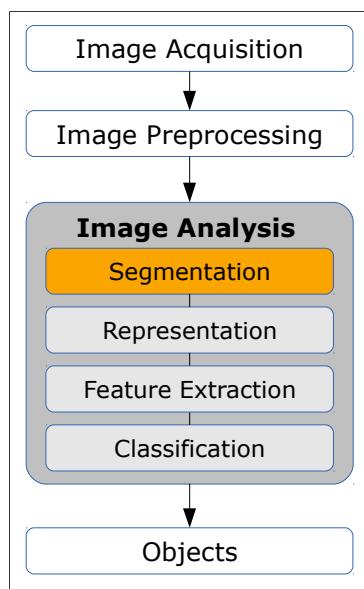


Abb. 209: Klassischer Aufbau der Bildanalyse aus Segmentierung, Merkmalsextraktion und Klassifikation

8.1 Schwellwertbasierte Segmentierung

Die schwellwertbasierte Segmentierung haben wir bereits als Binarisierung im Kapitel der Punktoperationen kennengelernt. In der Literatur wird sie deshalb auch punktbasierte Segmentierung bezeichnet. Im Unterschied zur regionenbasierten Segmentierung basiert die schwellwertbasierte Segmentierung nicht auf der *Kontinuität* einer Region, sondern auf einer Schwelle und damit auf *Diskontinuität*.

Die Operation der Binarisierung ist als solches eine Punktoperation (s. Kapitel 5.3) und damit einfach und schnell. Die automatische Berechnung des Schwellwerts kann aber aufwendig sein und die Einfachheit der Punktoperation schnell zunichtemachen. Im schlimmsten Fall gibt es keinen globalen Schwellwert, der für das ganze Bild funktioniert. In diesem Fall müssen lokale, an die Regionen des Bildes angepasste Schwellwerte bestimmt werden.

8.1.1 Globaler Schwellwert

Besteht kein Wissen über das zu segmentierende Objekt (z. B. Grösse oder Form) kann der Schwellwert durch statistische Auswertung bestimmt werden.

8.1.1.1 Mittelwert

Bei dieser einfachsten Methode wird der Schwellwert gleich dem Mittelwert aller Graustufen gesetzt. Andere iterative Methoden verwenden diesen Schwellwert als Startpunkt. In ImageJ heisst diese Methode [Mean](#).

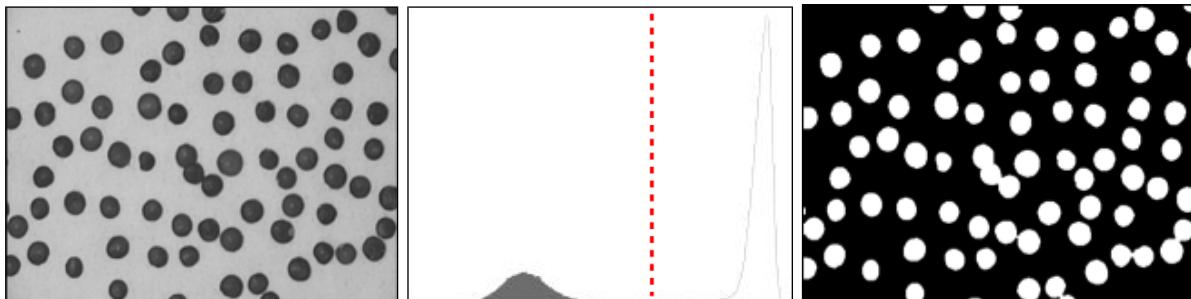


Abb. 210: Einfacher globaler Schwellwert der auf dem Mittelwert aller Graustufen basiert.

8.1.1.2 k-Means-Clustering

Durch Iteration mit dem als [k-Means-Clustering](#) bekannten Algorithmus wird der Schwellwert so bestimmt, dass die Mittelwerte beider Pixelgruppen sich maximal unterscheiden:

1. Setze einen zufälligen Schwellwert t
2. Berechne Mittelwert m_1 aller Pixel mit Grauwert $< t$ und Mittelwert m_2 aller Pixel mit Grauwert $\geq t$
3. Setze neuen Schwellwert $t = (m_1+m_2)/2$
4. Gehe zu Schritt 2, solange bis t sich nicht mehr ändert.

Diese eindimensionale Variante des k-Means-Clusterings kann ohne Weiteres für mehrdimensionale Histogramme angewendet werden und ist eine klassische Methode der unüberwachten Klassifikation (s. Kapitel 11.1.2.1).

8.1.1.3 Otsu's optimale Methode

Das als [Otsu's Methode](#) bekannte Verfahren gilt in dem Sinne als optimal, als dass es den Schwellwert t so wählt, dass die Summe der [Innerklassenvarianzen](#) (*within-class variance*) minimiert sind:

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \quad \text{worin}$$

$$\omega_i(t) = \sum_0^t p_i \quad \text{die aufsummierten Wahrscheinlichkeiten und}$$

$$\sigma_i^2 \quad \text{die Varianzen innerhalb der Klassen sind.}$$

Otsu hat gezeigt, dass die Minimierung der Varianzen innerhalb der Klassen identisch ist mit der Maximierung der [Zwischenklassenvarianz](#) (*between-class variance*):

$$\sigma_B^2(t) = \sigma^2 - \sigma_w^2 = \omega_1(t)[\mu_1(t) - \mu_G]^2 + \omega_2(t)[\mu_2(t) - \mu_G]^2$$

worin σ_i^2 die kombinierte Varianz, μ_G der globale Mittelwert und μ_i die Mittelwerte innerhalb der Klassen sind.

8.1.1.4 Prozentmethode

Wenn Sie wissen, wie gross das gesuchte Vordergrundobjekt ist, so können Sie den Schwellwert danach bestimmen. Die Prozentmethode verschiebt den Schwellwert so lange, bis die Anzahl Pixel des Vordergrunds dem gesuchten Prozentsatz entspricht.

8.1.1.5 Globale Schwellwerte in ImageJ und Matlab



Fiji bietet unter dem Menü *Image > Adjust > Auto Threshold* die Methode *Try all*, um alle automatischen Berechnungsmethoden nebeneinander anzuzeigen. Eine komplette Übersicht über alle [automatischen Methoden](#) finden Sie in der Online-Doku von Fiji.



Matlab bietet mit der Methode *graythresh* eine Berechnung für einen globalen Schwellwert an. Die Funktion wendet die Methode von [Otsu](#) an.

```
I = imread('coins.png');
level = graythresh(I);
BW = im2bw(I,level);
imshow(BW)
```

8.1.2 Lokaler Schwellwert

Ist die Helligkeit nicht gleichmäßig über das Bild verteilt, so können wir nicht mit einem globalen Schwellwert für das ganze Bild arbeiten. Der Schwellwert muss pro Pixel in einer lokalen Umgebung berechnet werden und somit ist die Operation ein relativ teurer, lokaler Operator. Lokale Schwellwerte werden deshalb auch *adaptive Schwellwerte* genannt.



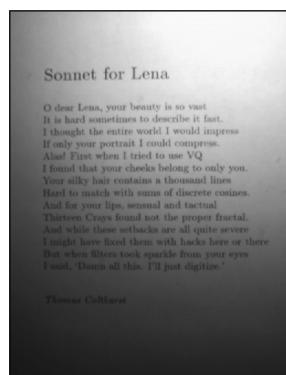
Fiji bietet einige Algorithmen zur lokalen Schwellwertberechnung an. Für alle Algorithmen muss die Grösse der lokalen Umgebung angegeben werden und optional können 2 Parameter übergeben werden:

- **Mean:** Setzt den Schwellwert als Mittelwert der lokalen Umgebung:
`pixel = (pixel > mean - c) ? object : background`
 Parameter 1: c = Optionaler Offsetwert, Default: 0
 Parameter 2: Nicht verwendet
- **Median:** Setzt den Schwellwert als Medianwert der lokalen Umgebung:
`pixel = (pixel > median - c) ? object : background`
 Parameter 1: c = Optionaler Offsetwert, Default: 0
 Parameter 2: Nicht verwendet
- **MidGray:** Setzt den Schwellwert als Mittelwert aus dem Minimum und Maximum der lokalen Umgebung:
`pixel = (pixel > ((max + min)/2) - c) ? object : background`
 Parameter 1: c = Optionaler Offsetwert, Default: 0
 Parameter 2: Nicht verwendet
- **Niblack:** Setzt den Schwellwert zusammen aus dem Mittelwert und der Standardabweichung der lokalen Umgebung:
`pixel = (pixel > mean + k * stddev - c) ? object : background`
 Parameter 1: k = Skalierungsfaktor für die Standardabweichung, Default: 0.2
 Parameter 2: c = Optionaler Offsetwert, Default: 0



Fiji bietet unter dem Menü *Image > Adjust > Auto Local Threshold* die Methode *Try all*, um alle lokalen Schwellwertmethoden nebeneinander anzuzeigen. Eine komplette Übersicht über alle [lokalen Methoden](#) finden Sie in der Online-Doku von Fiji.

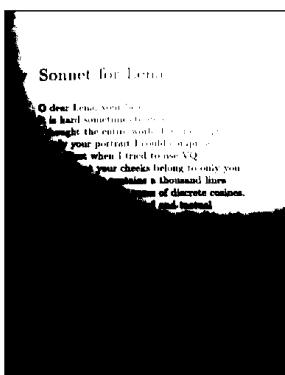
Anwendungsbeispiel:



Sonnet for Lena

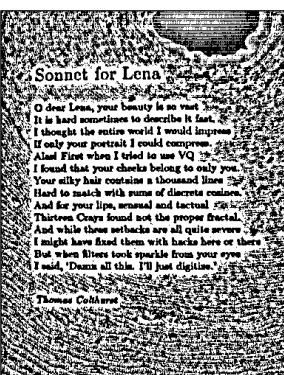
O dear Lena, your beauty is so rare.
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Also, First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactful,
The filters took sparkle from your eyes.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this, I'll just digitize.'

Thomas Goldhart



Sonnet for Lena

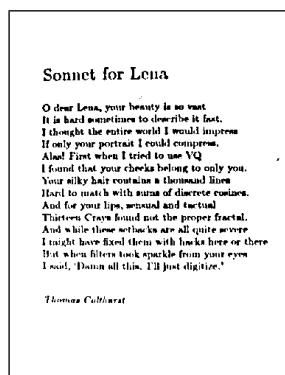
O dear Lena, your beauty is so rare.
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Also, First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactful,
The filters took sparkle from your eyes.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this, I'll just digitize.'



Sonnet for Lena

O dear Lena, your beauty is so rare.
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Also, First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactful,
The filters took sparkle from your eyes.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this, I'll just digitize.'

Thomas Goldhart



Sonnet for Lena

O dear Lena, your beauty is so rare.
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Also, First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactful,
The filters took sparkle from your eyes.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this, I'll just digitize.'

Thomas Goldhart

Abb. 211: **Bild 1:** Originalbild mit fotografiertem Text und sehr ungleichmässiger Beleuchtung.

Bild 2: Binarisierung mit globalem Schwellwert durch die Mean Methode.

Bild 3: Binarisierung mit lokalem Schwellwert durch die Mean Methode mit der Maskengrösse 7. Das Verfahren gelingt in der Umgebung des Textes, weil genug Vordergrund und Hintergrund-Pixeln in der Nachbarschaft jedes Pixels sind. D. h., der Mittelwert liegt zwischen den Intensitätswerten von Vordergrund und Hintergrund und trennt daher gut. Am Rand ist jedoch der Mittelwert der lokalen Umgebung nicht geeignet. Der Bereich der Intensitätswerte ist dort sehr klein und ihr Mittelwert ist nahe dem Wert des zentralen Pixels.

Bild 4: Gleiche Methode wie in Bild 3 aber mit zusätzlichem Offsetwert c=7.

Zu diesem Beispiel ist noch anzumerken, dass alternativ dazu auch die ungleichmässige Beleuchtung durch Bildarithmetik wegsubtrahiert oder wegdividiert werden kann (s. Kapitel 5.8). Diese Operation wäre eine schnelle Punktoperation und wäre zu favorisieren, insbesondere wenn man sie oft oder mit immer demselben Beleuchtungsausgleich verwenden könnte.



Matlab bietet keine Binarisierung mit lokalem Schwellwert an. In der Code Sammlung [MatlabCentral von Mathworks](#) finden Sie aber etliche Matlab-Skripte, z. B. mit der *Niblack* Methode, zum Thema *local thresholding*.

8.1.3 Tiefpassfilterung vor Schwellwertsegmentierung

Jegliche Segmentierung mit Schwellwerten ist schwierig oder gar unmöglich, wenn ein Bild starkes Rauschen enthält. Das Histogramm eines verrauschten Bildes gibt kaum einen Hinweis auf einen potenziellen Schwellwert. Das Bild muss deshalb vorher mit einem geeigneten Tiefpassfilter (z. B. Gaussfilter oder Medianfilter) entrauscht werden.

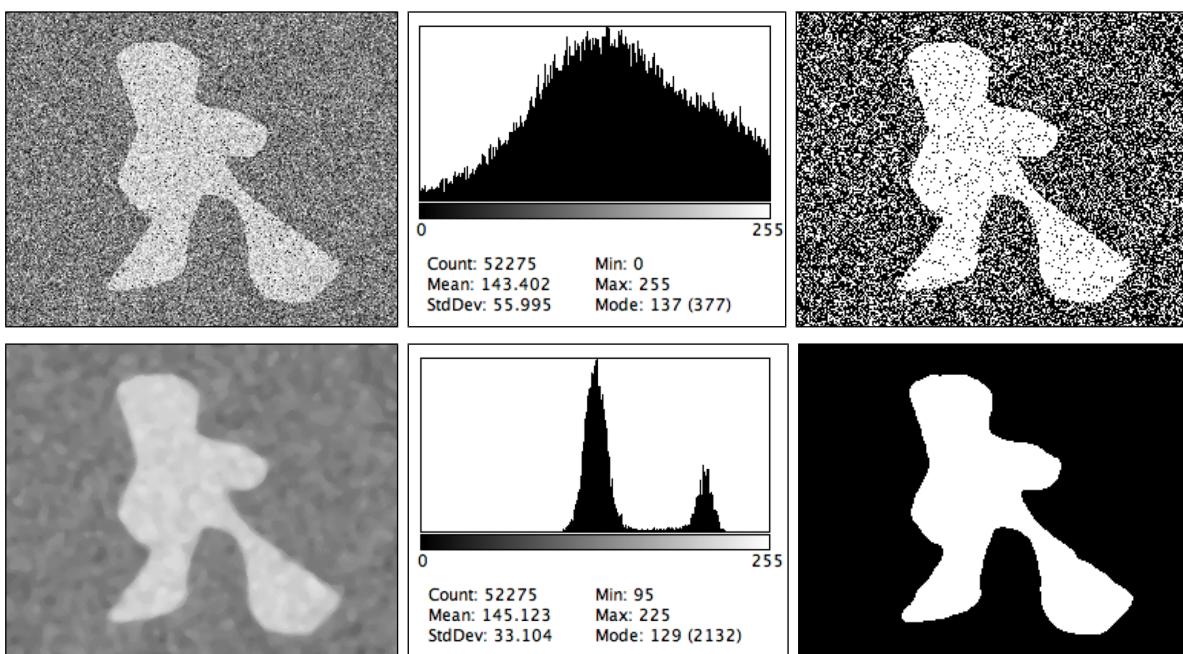


Abb. 212: Oben: Binarisierung eines verrauschten Bildes. Aus dem Histogramm lässt sich kein Schwellwert berechnen. Unten: Binarisierung nach der Medianfilterung.

8.2 Regionenbasierte Segmentierung

Im Unterschied zu den schwellwert- und kantenbasierten Verfahren beruht die regionenbasierte Segmentierung auf der Kontinuität einer Region. Die Kontinuität kann durch verschiedene Kriterien definiert werden:

- Ähnliche Grauwerte
- Ähnliche Farbkanalwerte
- Ähnliche Varianz der Pixelwerte
- Ähnliche Textur (Musterung)
- Ähnliche Bewegungsvektoren
- Ähnliche Tiefenwerte aus Stereo Bilder

Die regionenbasierte Segmentierung muss folgenden Kriterien genügen:

- **Komplettheit:** Alle Pixel müssen einer Region zugewiesen sein.
- **Verbundenheit:** Die Pixel einer Region müssen miteinander verbunden sein. Für zwei benachbarte Pixel müssen wir deshalb festlegen, ob sie nur über die Seite (d. h. über 2 Ecken) oder nur über eine Ecke miteinander verknüpft sein müssen. Für ein Pixel führt dies zu einer 4er- oder 8er-Nachbarschaft, wie wir dies bereits bei der Morphologie kennengelernt haben:

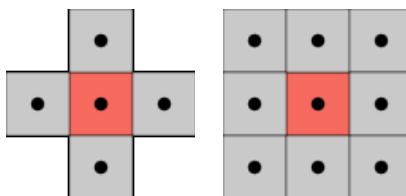


Abb. 213: 4er- oder 8er-Nachbarschaft eines Pixels

- **Disjunktheit:** Regionen müssen disjunkt sein. D. h., ein Pixel gehört zu genau einer Region.
- **Kontinuität:** Die Pixel einer Region müssen mindestens einem Ähnlichkeitskriterium genügen.
- **Separierbarkeit:** Zwei benachbarte Regionen müssen unterschiedlichen Ähnlichkeitskriterien genügen.

8.2.1 Region Growing

Bei diesem einfachsten Verfahren wird von einem per Zufall, automatisch oder von Hand gewählten Startpunkt (*seed point*) ausgegangen und die Nachbarschaftspixel anhand eines Ähnlichkeitskriteriums zur Region hinzugefügt.

Mit manueller Startpunktsetzung finden Sie diese Segmentierung in den meisten Bildverarbeitungsprogrammen als *Giesskannenwerkzeug*. Wird dabei das Ähnlichkeitskriterium zu weit angesetzt, so überläuft die gewünschte Region.

Neue Pixel werden meistens mit dem Mittelwert oder Median der Region verglichen. Kommen keine neuen Pixel zur Region mehr hinzu, kann der Prozess in der automatischen Variante mit einem neuen Startpunkt solange fortgesetzt werden, bis keine Pixel mehr zuzuweisen sind. Dieser Algorithmus kann entweder mit einem rekursiven oder iterativen *Floodfill* Algorithmus (s. Kapitel 9.1) einfach implementiert werden.

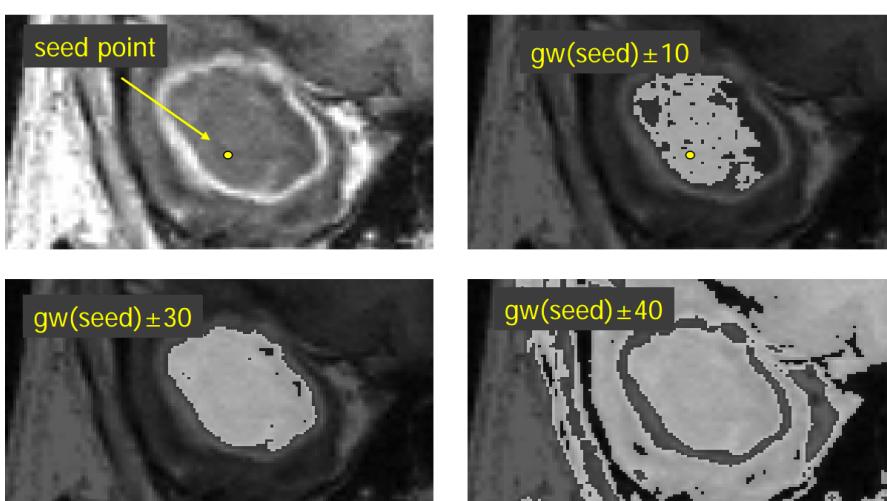


Abb. 214: Region Growing mit drei verschiedenen Schwellwerten (Quelle: Tönnies05).

Vorteile und Nachteile:

- + Allgemein schnell
- + Konzeptuell einfach
- + Findet mehr Regionen als schwellwertbasierte Verfahren
- + Ähnlichkeitskriterien können vielfältig sein
- + Gut mit manueller Startpunktsetzung
- Je nach Startpunkt können andere Regionen entstehen
- Anzahl Regionen müssen vorbestimmt sein.



Für **ImageJ** ist Region Growing im [Plugin SRG \(Seeded Region Growing\)](#) implementiert.



Für **Matlab** gibt es ein [Region Growing Skript](#) in der Matlab Central Code Sammlung.

8.2.2 Split and Merge

Wenn man das *Region Growing* als bottom-up bezeichnen könnte, so kann man das *Split and Merge* Verfahren als top-down bezeichnen. Ein Bild wird im Split-Teil solange geviertelt, bis nur noch homogene Zellen vorhanden sind. Danach werden im Merge-Teil gleich helle benachbarte Zellen zusammengefasst. Beim Aufsplitten entsteht ein Quadtree-Datenstruktur sowie ein Nachbarschaftsgraph (*Region Adjacency Graph RAG*) des Quadtrees:

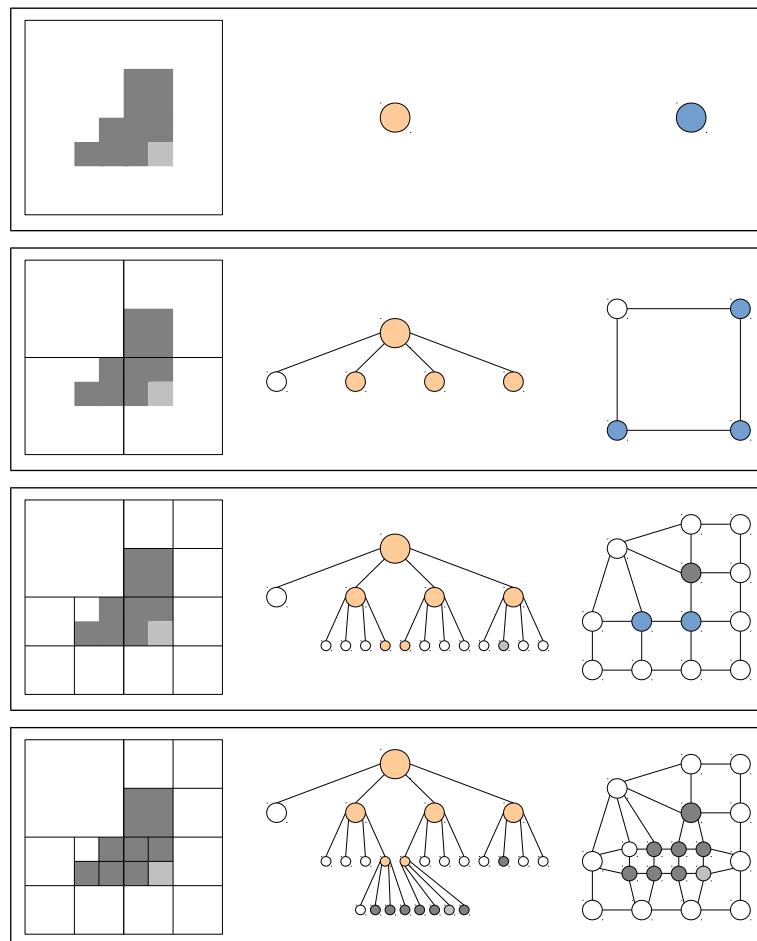


Abb. 215: Aufsplittung eines Bildes (links) in einen Quadtree (Mitte) und Zusammenfassung in einem Nachbarschaftsgraphen (rechts).

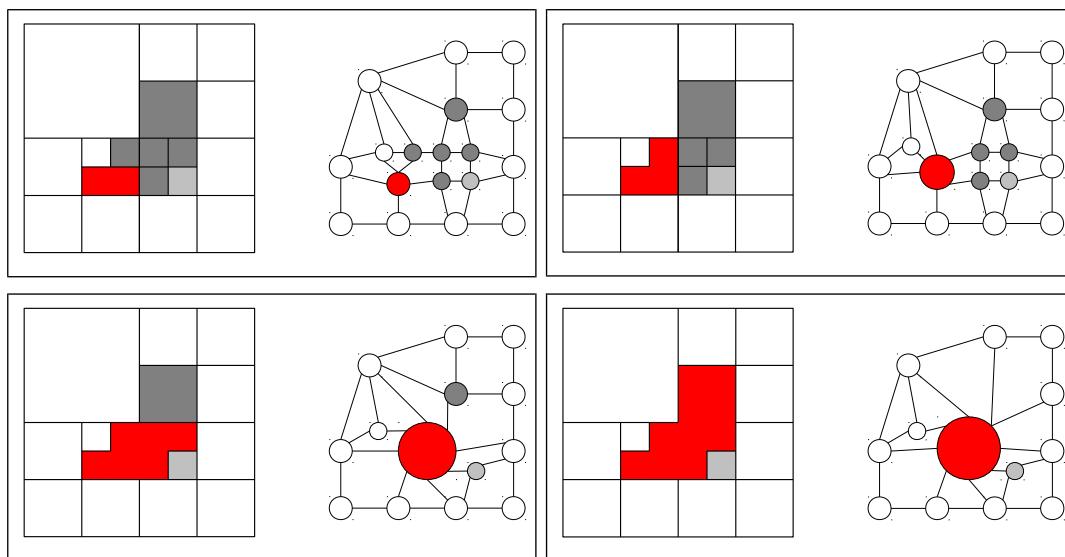


Abb. 216: Zusammenfassen der dunkelgrauen Knoten im Nachbarschaftsgraphen zu einer Region. Am Schluss enthält der Nachbarschaftsgraph noch 3 Knoten.

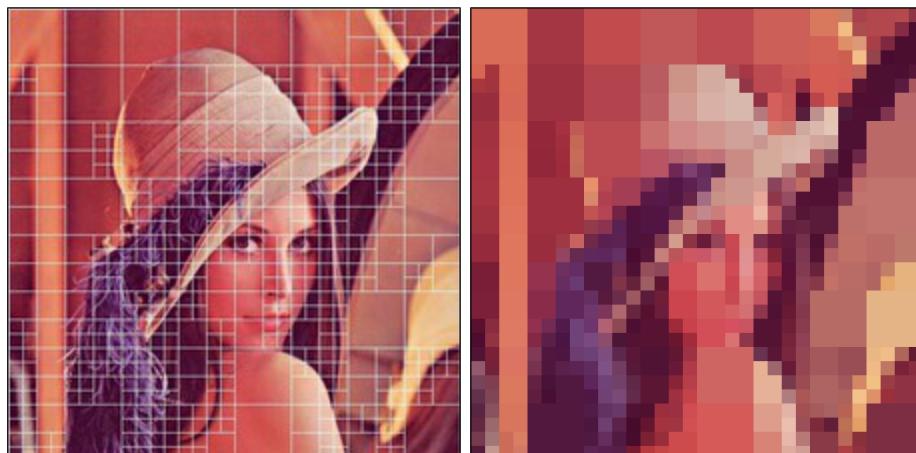


Abb. 217: Resultat einer Split & Merge Segmentation beim Lena Bild. (Quelle: Western University of Canada)

8.3 Kantenbasierte Segmentierung

Im Unterschied zur punkt- und regionenbasierten Segmentierung, wo die Kontinuität der Pixel die Grundlage ist, basiert die Segmentierung über Kanten auf der Diskontinuität der Pixelwerte. Kanten sind ja nichts anderes als starke Änderungen der Pixelwerte.



Abb. 218: Kanten müssen jedoch nicht unbedingt mit Objekt-Silhouetten übereinstimmen.

8.3.1 Gradientenbasierte Kantenfilter

Die Kantenfilter, die auf der 1. oder 2. Ableitung des Bildes basieren, haben wir bereits im Kapitel über lokale Operatoren ausgiebig hergeleitet (s. Kapitel 6.1.3 über Hochpassfilter), sodass das Kapitel hier nur der Vollständigkeit halber aufgeführt ist.

Die wichtigsten Vertreter dieser Gruppe sind:

- Differenzoperator
- Roberts-Cross Operator
- Sobel Operator
- Laplace Operator
- Laplace of a Gaussian Operator

Vorteile der gradientenbasierten Kantenfilter sind:

- Pro Pixel erhalten wir die **Stärke der Kante** durch die Gradientenlänge:

$$|G'| = \sqrt{G'_x^2 + G'_y^2}$$

- Pro Pixel erhalten wir die **Richtung der Kante** durch den Gradientenwinkel:

$$\theta = \text{atan} \left(\frac{G'_y}{G'_x} \right)$$

Nachteile der gradientenbasierten Kantenfilter sind:

- **Abhängigkeit von der Auflösung**: Kantenfilter der Grösse 3x3 können nur Intensitätswerte in dieser kleinen Nachbarschaft unterscheiden. Die detektierten Kanten hängen damit stark von der Auflösung des Bildes und der Grösse der Filtermaske ab.
- **Ungenauigkeit der Kanten**: Nach einer Filterung haben wir immer noch ein Rasterbild und keinerlei parametrische Kanteninformation. Oft haben wir auch unscharfe Kanten, die nach einer Binarisierung ganz verschwinden oder immer noch dicker als ein Pixel sein können.

8.3.2 Morphologische Kantenfilter

Auch den morphologischen Kantenfilter haben wir bereits im Kapitel 6.2.2.4 über Graustufenmorphologie kennengelernt. Er ist gegenüber den gradientenbasierten Kantenfiltern weniger anfällig auf Rauschen, weil sowohl die Dilatation als auch die Erosion glättend wirken. Nachteilig ist, dass dabei relativ dicke Kanten entstehen.

$$G_{\text{edge intern}} = G - (G \ominus S)$$

$$G_{\text{edge extern}} = (G \oplus S) - G$$

$$G_{\text{edge thick}} = (G \oplus S) - (G \ominus S)$$

$$G_{\text{edge thin}} = \min(G_{\text{edge intern}}, G_{\text{edge extern}})$$

8.3.3 Canny Kantendetektor

Dieser Kantendetektor wurde von *John Francis Canny* 1986 [[Canny86](#)] vorgestellt und gilt bis heute als beste Methode um Kanten präzise zu detektieren. Sein Paper „[A computational approach to edge detection](#)“ ist das am [drittmeisten zitierten Paper](#) der Computer Vision Wissenschaft.

Canny wollte folgende Ziele damit erreichen:

- **Geringe Fehler**: Wo eine Kante ist, soll eine detektiert werden und wo keine ist, soll auch keine detektiert werden (z. B. in Rauschen). Kanten sollen zudem möglichst durchgehend verlaufen.

- **Präzise:** Bei einer weichen Kante soll die Kante beim steilsten Anstieg und nicht beim Beginn oder Ende des Anstiegs detektiert werden.
- **Ein Pixel breite Kanten:** Um den Kantenverlauf möglichst einfach verfolgen zu können, soll eine Kante immer genau ein Pixel breit sein.

Der Algorithmus kann in vier Schritten zusammengefasst werden:

1. **Glättung mit Gaussfilter:** Ob wir nur starke oder feine Kanten detektieren wollen, hängt von dieser vorgängigen Tiefpassfilterung ab und wird durch die Grösse des Gaussfilters (s. auch Kapitel 6.1.2.2) bestimmt.



Abb. 219: Vor und nach der Tiefpassfilterung. Bildquelle: [Kalra09]

2. **Bildung der Gradienten:** Das Kantenbild wird als Betrag zweier einfacher Gradientenbilder in X- und Y-Richtung zusammengesetzt.

$$|G'| = \sqrt{G'_x^2 + G'_y^2}$$

Als Filtermasken können einfache 1D-Differenzfilter verwendet werden:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad D_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Einige Implementationen verwenden den Sobelfilter in X- und Y-Richtung. Dies ist jedoch nicht notwendig, weil unser Bild schon mit dem Gauss gefiltert ist.



Abb. 220: Vor und nach der Gradientenfilterung. Bildquelle: [Kalra09]

Die Gradientenrichtung wird in einem extra Bild gespeichert. Da wir nur die Richtungen eines Pixels zu seinen 8 Nachbarn betrachten können, ergeben sich nur die 4 Richtungen mit den Winkeln 0°, 45°, 90° oder 135°.

$$\theta = \text{atan} \left(\frac{G'_y}{G'_x} \right)$$

Abb. 221: Vier mögliche Winkel bei einer 8er-Nachbarschaft: 0°, 45°, 90° oder 135° sowie ihre gespiegelten Winkel.

3. **Non-Maximum Suppression:** Weil die Kanten meistens breiter als ein Pixel sind, werden sie in diesem Schritt durch *non-maximum suppression* auf das Pixel mit der maximalen Steigung reduziert. Mit einer 3x3 Maske wird dabei durch das Bild

iteriert und die jeweiligen Nachbarn des Pixels bei $[x,y]$ mit den Pixeln in der entsprechenden Richtung verglichen:

- Bei $\theta=0^\circ$ die Pixel $[x-1,y]$ und $[x+1,y]$
- Bei $\theta=45^\circ$ die Pixel $[x-1,y-1]$ und $[x+1,y+1]$
- Bei $\theta=90^\circ$ die Pixel $[x,y-1]$ und $[x,y+1]$
- Bei $\theta=135^\circ$ die Pixel $[x-1,y+1]$ und $[x+1,y-1]$

Hat das Pixel bei $[x,y]$ den stärksten Gradienten, so überlebt es, anderenfalls wird es auf 0 gesetzt.

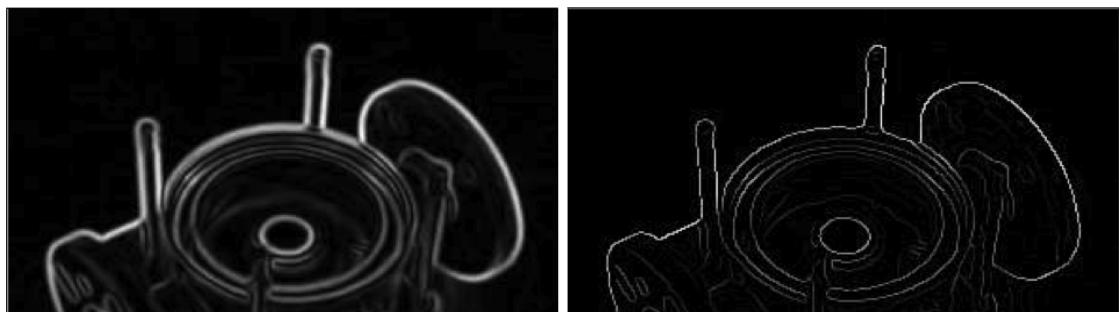


Abb. 222: Vor und nach der Non-Maximum Suppression. Bildquelle: [Kalra09]

4. **Hysterese Thresholding:** Jetzt haben wir zwar nur noch ein Pixel breite Kanten, jedoch sind diese noch unterschiedlich stark. Mit einem als *Hysterese* bekannten Verfahren unterteilen wir die Kantenpixel zunächst mittels zweier Schwellwerte t_{high} und t_{low} in starke, schwache und zu löschen Pixel. Erstere machen wir zu Weiss, schwache zu Grau und Letztere zu schwarz (= löschen). Canny empfiehlt für die beiden Schwellwerte t_{high} und t_{low} ein Verhältnis von ca. 3:1. Beim nachfolgenden Beispiel war der obere Schwellwert 80 und der Untere 20:



Abb. 223: Vor und nach der Reduktion auf starke und mittlere Kanten. Bildquelle: [Kalra09]

Zuletzt müssen wir noch entscheiden, welche der schwachen Kantenpixel noch zu den starken Kantenpixeln hinzuzuzählen sind.

- Weil wir in diesem letzten Schritt die Kanten rekursiv verfolgen, machen wir die Pixel am Bildrand zu null, sodass die Suche nach Nachbarn nicht aus dem Bild läuft.
- Als Nächstes wird das Bild von links nach rechts und von oben nach unten abgetastet. Das erste Pixel oberhalb des Schwellwerts t_{high} gehört zu einer Kante.
 - Alle seine Nachbarn werden rekursiv verfolgt und diejenigen, oberhalb der Schwelle t_{low} werden als Kante markiert.
 - In einer zusätzlichen Map wird festgehalten, welche Pixel abgearbeitet wurden, damit die Rekursion nicht endlos ist.
 - Es gibt demnach zwei Bedingungen, um die Rekursion abzubrechen:
 - Wenn ein Nachbar unter t_{low} ist.

- Wenn ein Nachbar bereits besucht wurde.

Dieser letzte Teil ist rekursiv implementiert zwar elegant aber wie alle rekursiven Algorithmen nicht besonders performant. Wie die meisten rekursiven Algorithmen kann auch dieser in einen iterativen Algorithmus mit einer Queue umgeschrieben werden:

```
trace(x,y)
    create a queue Q of positions
    enqueue (x,y) in Q
    while 'que' is not empty
        pop position (xq,yq) from queue
        if (xq,yq) is inside gradient image
            AND pixel(xq,yq) is strong or candidate
            AND the result(xq,yq) is not yet marked as edge
        then
            mark result(xq,yq) as edge
            enqueue all 8-connected neighbors of (xq,yq) in Q
```

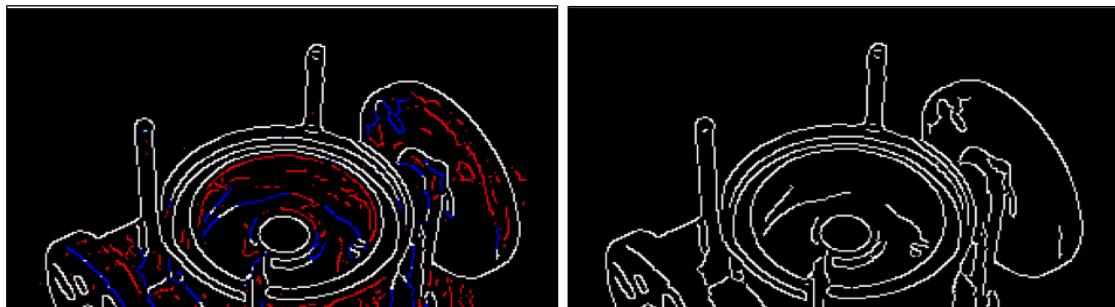


Abb. 224: Nach der Kantenvervollständigung. Im linken Bild sind die schwachen Kanten, die zu starken Kanten verknüpft werden konnten blau markiert und die unverknüpfbaren rot. Bildquelle: [Kalra09]

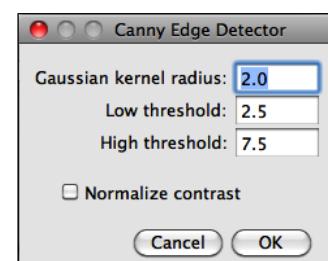
Der Canny Kantendetektor hat also insgesamt 3 Parameter:

1. **Breite der Gauss-Filtermaske:** Bei einigen Implementierungen wird die Filterstärke durch einen Parameter Sigma σ der Gauss-Verteilung definiert.
 - Eine 3x3 Gauss-Filtermaske hat ein σ von 1.5
 - Eine 5x5 Gauss-Filtermaske hat ein σ von 2.5
 - Eine 7x7 Gauss-Filtermaske hat ein σ von 3.5
 - Eine 9x9 Gauss-Filtermaske hat ein σ von 4.5
 - Eine 11x11 Gauss-Filtermaske hat ein σ von 5.5
2. **Schwellwerte t_{high}** der Kantenvervollständigung.
3. **Schwellwerte t_{low}** der Kantenvervollständigung. Der Wertebereich ist bei beiden abhängig von der Implementation.



Abb. 225: Einfluss der Gauss-Filterung auf die Anzahl der detektierten Kanten. V.l.n.r.: Original (512x512), Canny Filterungen mit 3x3 ($\sigma=1.5$), 5x5 ($\sigma=2.5$) und 7x7 ($\sigma=3.5$) Gaussmaske.

In **ImageJ** ist der *Canny Edge Detector* interessanterweise nicht standardmäßig eingebaut. Es gibt aber ein gutes Plugin von [Tom Gibara](#), welches die 3 Parameter in einem Dialog zur Verfügung stellt.

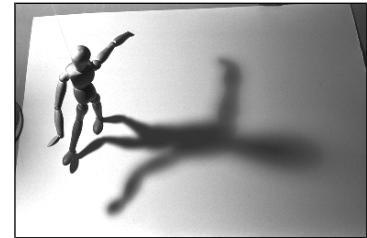




In **Matlab** kann der *Canny Edge Detector* mit der Funktion `BW=edge(I, 'canny', thresh, sigma)` angewendet werden. Der Parameter `thresh` ist optional. Sie können beide Schwellwerte als Vektor oder nur einen Wert als oberen Schwellwert angeben. Der Untere wird dann aus dem Oberen erzeugt mit $t_{high}=0.4 \times t_{low}$. Der Parameter `sigma` wird standardmäßig auf $\sqrt{2}$ gesetzt.

8.3.4 Übung zu Kantendetektoren

1. Neben Sie das Bild *ShadowSmall.jpg* und erstellen Sie zwei Versionen davon, einmal mit 1% und einmal mit 5% zusätzlichem Gauss-Rauschen.
2. Erstellen Sie je ein Kantenbild von allen 3 Bildern mit folgenden Kantendetektoren:
 1. Sobelfilter
 2. LoG-Filter
 3. Morphologischer Filter
 4. Canny Edge Filter
3. Vergleichen Sie die Resultate. Welcher Filter hat welche Vor- und Nachteile?



8.4 Modellbasierte Segmentierung

Die vorgängigen Methoden der Segmentierung basierten entweder auf der Kontinuität innerhalb einer Region oder auf der Diskontinuität zwischen den Regionen. Daraus konnten sich beliebige Formen der Regionen ergeben. In der modellbasierten Segmentation gehen wir bereits mit einer mehr oder weniger bestimmten Vorstellung einer Segmentgrenze auf die Suche.

8.4.1 Hough Transformation

Die Hough-Transformation (HT) haben wir bereits im Kapitel 7.3 bei den globalen Operatoren kennengelernt. Manche Autoren ordnen die HT im Kapitel der kantenbasierten Segmentation ein. Im Unterschied zu den dort behandelten Kantenfiltern, ermöglicht die HT jedoch nach ganz bestimmten Geraden und Kreisen zu suchen. Dem Vorteil, dass wir daraus direkt eine logische Information gewinnen, z. B. wo eine bestimmte Münze liegt, steht der Nachteil gegenüber, dass wir nur nach bestimmten Geradenlängen oder Kreisdurchmessern suchen können.

8.4.2 Aktive Konturen

Bei aktiven Konturen wird eine Objektkontur durch eine parametrisierte Kurve beschrieben. Der Algorithmus wurde von den Herren *Kass, Witkin* und *Terzopoulos* 1988 entwickelt. Ihr Paper „*Snakes: Active contour models*“ [Kass88] ist auf Rang 4 der am meisten zitierten Paper der Computer Vision Wissenschaften.

Active Contours sind auch unter den synonymen Begriffen *Snakes*, *Deformable Countours* oder *Minimal Energy Countours* bekannt.

Aktive Konturen eignen sich gut, um in verrauschten Bildern Konturen von Regionen zu bestimmen, die mit anderen Segmentationsmethoden nur bedingt als zusammenhängend erkannt werden. Sie werden deshalb häufig in medizinischen Bildern eingesetzt, wo ein Objekt auch über eine ganze Sequenz von Schichtbildern verfolgt werden kann.

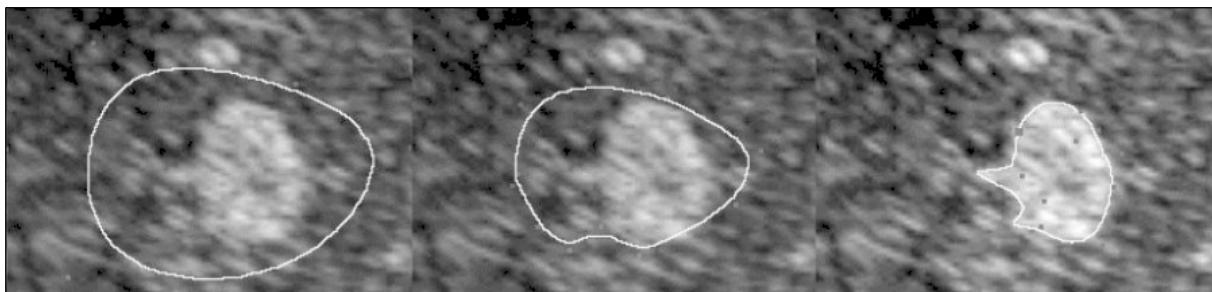


Abb. 226: Annäherung einer aktiven Kontur an seine Region (Quelle: Achim Gegler, Uni Ulm)

8.4.2.1 Parameterkurve

Um die Kontur zu beschreiben, wird eine parametrisierte, geschlossene Kurve verwendet:

$$r(s) = (x(s), y(s)), s \in [0, 1]$$

worin $x(s)$ und $y(s)$ die Koordinaten entlang der Kurve sind. Für einen einfachen Kreis würde die Parameterdarstellung wie folgt aussehen:

$$r(s) = (r_{\text{radius}} \cos(2\pi s), r_{\text{radius}} \sin(2\pi s)), s \in [0, 1]$$

B-Spline Kurven

Um flexibel zu bleiben, werden für aktive Konturen meistens *B-Spline Kurven* verwendet. Der Begriff Spline bezeichnet dabei eine Straklatte, wie sie z. B. im Schiffsbau eingesetzt wurden, um runde Kurven zu zeichnen.

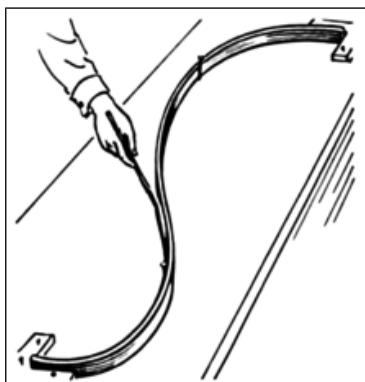


Abb. 227: Links: Eine Straklatte zum zeichnen einer Kurve (Quelle: Wikipedia).

Eine Spline-Kurve n -ten Grades setzt sich stückweise aus Polynomkurven zusammen die höchstens den n -ten Grad haben dürfen. Ein quadratischer Spline 3. Grades setzt sich also aus Polynomen 3. Grades ($f(x) = a + bx + cx^2$) zusammen.

Eine B-Spline-Kurve ist eine gewichtete Summe von N Basispolynomen:

$$x(s) = \sum_{n=0}^{N_B-1} x_n \cdot B_n(s)$$

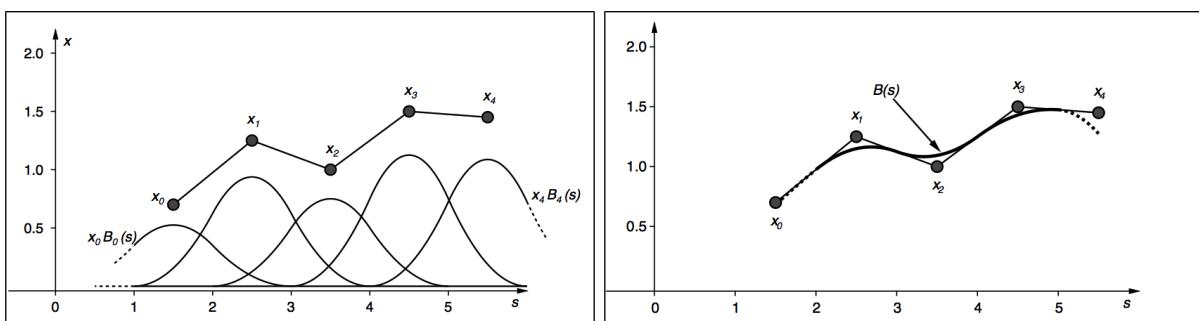


Abb. 228: Links: N Basis Polynomkurven. Rechts: B-Spline-Kurve als Summe der Basispolynome. Verbindet man die Kontrollpunkte, so erhält man das Kontrollpolygon, an das sich die Spline-Kurve angleicht (Quelle: [Blake00])

In der Parameterdarstellung für eine geschlossene Kurve werden n Kontrollpunkte q_n verwendet.

$$r(s) = \sum_{n=0}^{N_B-1} B_n(s) \cdot q_n \quad \text{mit } 0 \leq s \leq L \text{ und } q_n = (qx_n, qy_n)$$

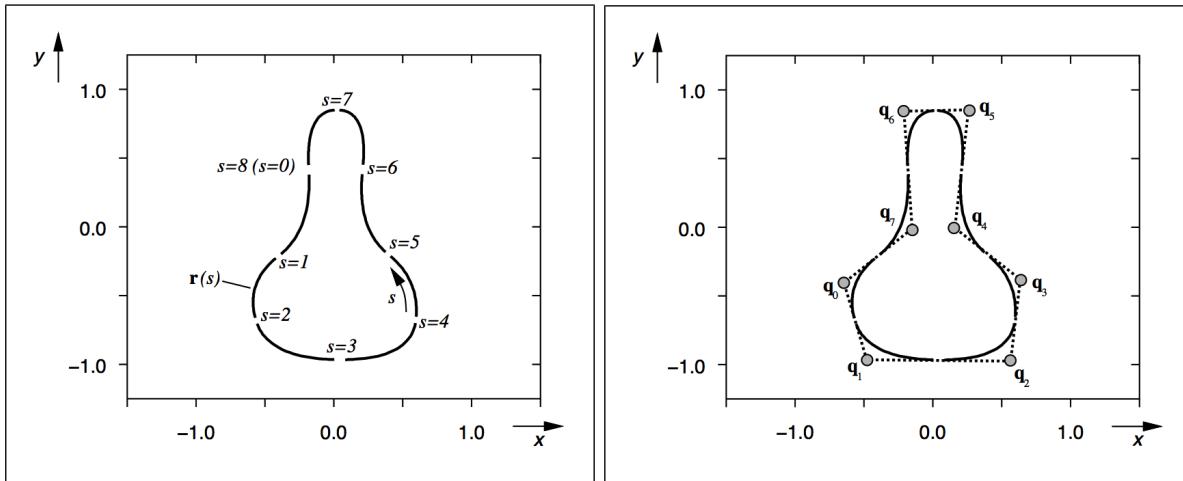


Abb. 229: Eine quadratische Spline-Parameterkurve $r(s)$ mit $0 \leq s \leq 8$ und 8 Kontrollpunkten q_0 bis q_7 . (Quelle: [Blake00])

8.4.2.2 Energiefunktion

Damit die Kurve eine aktive Kontur sein kann, wird eine Energiefunktion definiert, damit die Energie in der Kontur minimiert werden kann. Wenn man eine Straklatte zwischen 3 Nägel spannt, wird genau so die Spannungsenergie im Holz bei einer bestimmten Lage minimiert. Für aktive Konturen setzt sich die Energiefunktion aus internen und externen Kräften zusammen:

$$E_{\text{snake}} = \int_0^1 E_{\text{int}}(r(s)) + E_{\text{ext}}(r(s)) ds$$

Interne Energie

Die interne Energie besteht aus zwei Komponenten:

$$E_{\text{int}} = \frac{1}{2} (\alpha(s) \cdot E_{\text{elas}} + \beta(s) \cdot E_{\text{curv}})$$

- Die **interne Energie E_{elas}** entspricht der 1. Ableitung und sagt etwas aus über die Elastizität der Kurve. Die 1. Ableitung an einem Punkt kann mit dem Tangentialvektor visualisiert werden. Die Länge des Tangentialvektors entspricht der Elastizität an diesem Punkt.

$$E_{\text{elas}} = \left| \frac{dr}{ds} \right|^2$$

Mit dem Parameter $\alpha(s)$ wird die Elastizität der Kurve eingestellt:

$\alpha(s) > 0$ bewirkt ein Schrumpfen der Kurve

$\alpha(s) < 0$ bewirkt ein Dehnen der Kurve

- Die **interne Energie E_{curv}** entspricht der 2. Ableitung der Kurve und damit der Krümmung:

$$E_{curv} = \left| \frac{dr^2}{ds^2} \right|^2$$

Mit dem Parameter $\beta(s)$ wird die Krümmung der Kurve eingestellt:

$\beta(s) > 0$ bewirkt eine gleichmässige Krümmung der Kurve

$\beta(s) = 0$ ermöglicht Knicke in der Kurve

Die interne Energie entspricht den Kräften, die in einem Gummiband oder einer Kette aus Federn wirken. Bei gleich starken Federkräften gleichen sich die Federlängen aus. Alle Federkräfte gleichen sich aus und bewirken aber ein Schrumpfen. Biegekräfte in den Gelenken gleichen den inneren Winkel aus.

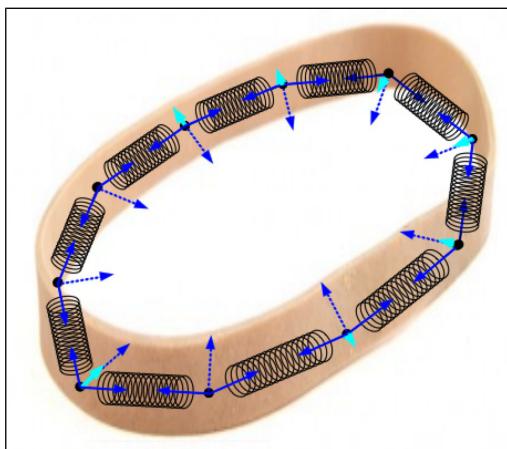


Abb. 230: Federkräfte (dunkelblau) bewirken ein Schrumpfen und die Biegekräfte (hellblau) gleichen die Krümmung aus.

Externe Energie

Die externe Energie an der Stelle $r(s)$ kommt aus der darunter liegenden Stelle des Bildes $I(x,y)$. Sie bewirkt, dass die Kurve zu hellen Stellen im Bild gezogen wird.

$$E_{ext}(r(s)) = -\gamma(s) \cdot I(x, y)$$

Wenn die Kontur zu Kanten hingezogen werden soll, so müssen wir das Bild mit Gradientenfilter ableiten. Der Faktor Gamma $\gamma(s)$ erlaubt eine Skalierung der externen Energie. Der Faktor ist negativ, weil er den internen Energien entgegengesetzt ist.

$$E_{ext}(r(s)) = -\gamma(s) \cdot |\nabla I(x, y)|$$

Zusammengefasst sieht die Energiefunktion einer aktiven Kontur ziemlich furchterregend aus:

$$E_{snake} = \int_0^1 \frac{1}{2} \left(\alpha(s) \cdot \left| \frac{dv}{ds} \right|^2 + \beta(s) \cdot \left| \frac{dv^2}{ds^2} \right|^2 \right) - \gamma(s) \cdot |\nabla I(x, y)| ds$$

Um diese Funktion zu minimieren, schlagen Kass et al. eine sogenannte [Variationsrechnung](#) vor. Dies entspricht dem Lösen einer Differenzialgleich von dem wir hier mal absehen wollen, weil es eine einfachere diskrete Lösung gibt.

8.4.2.3 Algorithmus von Williams und Shah

Williams und Shah entwickelten 1992 [[Williams92](#)] einen relativ einfachen [gierigen \(Engl.: greedy\) Algorithmus](#), der schrittweise sein Resultat zu einem Maximum oder Minimum optimiert. Numerisch gesehen ist es ein [Gradientenabstieg](#), bei dem man von einem Startwert in Richtung des negativen Gradienten schreitet, bis man keine numerische Verbesserung mehr erzielt.

Als Input dient das Graustufenbild $I(x,y)$ und eine initiale Kontur mit n Punkten p_1, p_2, \dots, p_n . In jeder Iteration werden diese Positionen der Konturpunkte verschoben, wenn die Energie dadurch minimiert wird. Die Energiefunktion setzt sich aus den Vektoren zu den benachbarten Punkten wie folgt zusammen:

$$E_{\text{snake}} = \sum_{i=1}^n (\alpha E_{\text{elas}}(p_i) + \beta E_{\text{curv}}(p_i) + \gamma E_{\text{img}}(p_i))$$

$$E_{\text{elas}} = |p_i - p_{i-1}|^2 = (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2$$

$$E_{\text{curv}} = |p_{i-1} - 2p_i + p_{i+1}|^2 = (x_{i-1} - x_i + x_{i+1})^2 + (y_{i-1} - y_i + y_{i+1})^2$$

$$E_{\text{img}} = -|\nabla I(x, y)|^2$$

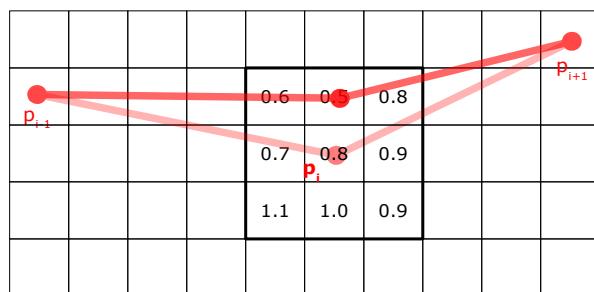
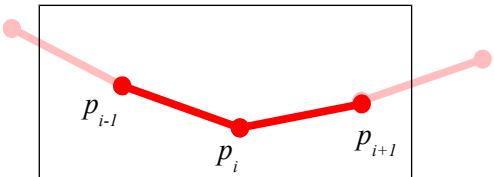


Abb. 231: In einer begrenzten Umgebung von z. B. 3x3 Pixel wird für jedes Pixel die Energie berechnet und die Position von Punkt p_i zum Pixel mit der geringsten Energie verschoben.

8.4.2.4 Snake Test Applikation in C#

Basierend auf der Idee von [David Young](#) wurde eine Testapplikation entwickelt, mit der die verschiedenen Kräfte in der Energiefunktion interaktiv verändert werden können. Sie finden die DotNet-Applikation in C# für VisualStudio 2010 im Ordner [SnakesDemo](#). Für die Filteroperationen verwendet die Applikation das [AForge.NET](#) Framework.

Sie können per Mausklick ein rotes Startpolygon setzen, das Sie durch [\[Double Points\]](#) verfeinern oder mit [\[Half Points\]](#) größer machen können. Das Polygon wird durch eine Instanz der Klasse [Snake](#) repräsentiert und besteht aus einer Liste von [SnakePoints](#), die wiederum aus einer Position und vier Kraftvektoren bestehen:

```
private class SnakePoint
{
    public Vec2F pos;          // position
    public Vec2F F_elas;        // internal elastic force vector (blue)
    public Vec2F F_curv;        // internal curvature equalization force vector (cyan)
    public Vec2F F_ballon;      // internal balloon expansion force vector (magenta)
    public Vec2F F_img;         // external image force vector (green)
}
private List<SnakePoint> m_points = new List<SnakePoint>();
```

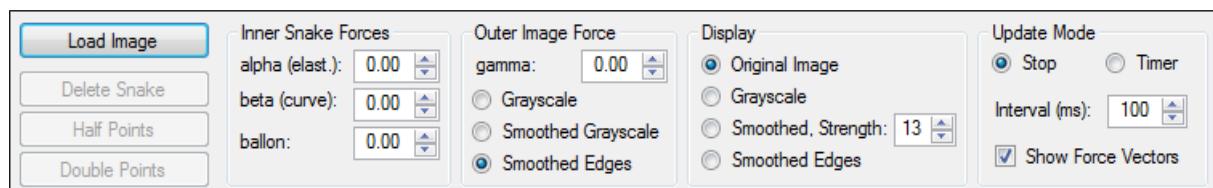


Abb. 232: Kontrollpanel mit den Parametern der inneren und äusseren Snake-Kräfte. In der Gruppe der äusseren Bildkräfte können Sie auswählen, aus welcher Bildversion die Kräfte berechnet werden sollen.

In der Routine [update](#) werden alle Punkte mit einer Vektoraddition aller Kräfte an den jeweiligen Punkten aktualisiert:

```
private void update()
```

```
{
    for (int i=0; i<m_points.Count; ++i)
    {   SnakePoint p = (SnakePoint)m_points[i];
        p.pos += p.F_elas + p.F_curv + p.F_ball + p.F_img;
    }
    Invalidate();
}
```

Die Berechnung der elastischen Kraft (*F_elas*) und der Bildkraft (*F_img*) geschieht in **calculateForces**. In einer Schleife werden für alle Punkte die Nachbarn *p_next* und *p_prev* bestimmt.

```
private void calculateForces()
{
    ...

    for (int i=0; i<m_snakePoints.Count; ++i)
    {
        SnakePoint p = m_points[i];

        // Get the neighbors
        SnakePoint p_next, p_prev;
        if (i == 0)
        {   p_next = m_points[i+1];
            p_prev = m_points[m_points.Count-1];
        } else if (i == m_points .Count-1)
        {   p_next = m_points[0];
            p_prev = m_points[i-1];
        } else
        {   p_next = m_points[i+1];
            p_prev = m_points[i-1];
        }

        /////////////////
        // Elasticity Force //
        /////////////////

        // Build the elasticity force vector
        Vec2F vNext = p_next.pos - p.pos;
        Vec2F vPrev = p_prev.pos - p.pos;

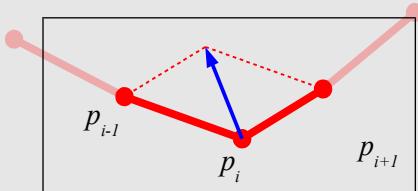
        p.F_elas = vNext + vPrev;

        // Scale the force components
        p.F_elas *= (float)numAlpha.Value * C_ELAS;

        /////////////////
        // Image Force //
        /////////////////
        ...

        // Build the image force vector
        // bmp.GetPixel(x, y).R retrieves the gray value (0-255) at pos. x,y
        p.F_img.x = bmp.GetPixel(x+1, y).R - bmp.GetPixel(x-1, y).R;
        p.F_img.y = bmp.GetPixel(x, y+1).R - bmp.GetPixel(x, y-1).R;

        // Scale the force components
        p.F_img *= (float)numGamma.Value * C_IMG;
    }
}
```



In der Routine *drawSnake* wird das Polygon mit allen Kraftvektoren gezeichnet. Solange der Update Modus auf Stop steht, wird *updateSnake* nur bei der Edition des Polygons und der Skalierungsfaktoren der Kräfte aufgerufen. Wenn Sie den Update Modus auf *Timer* stellen, dann wird das Polygon mit all seinen Kräften alle 100 Millisekunden aktualisiert.

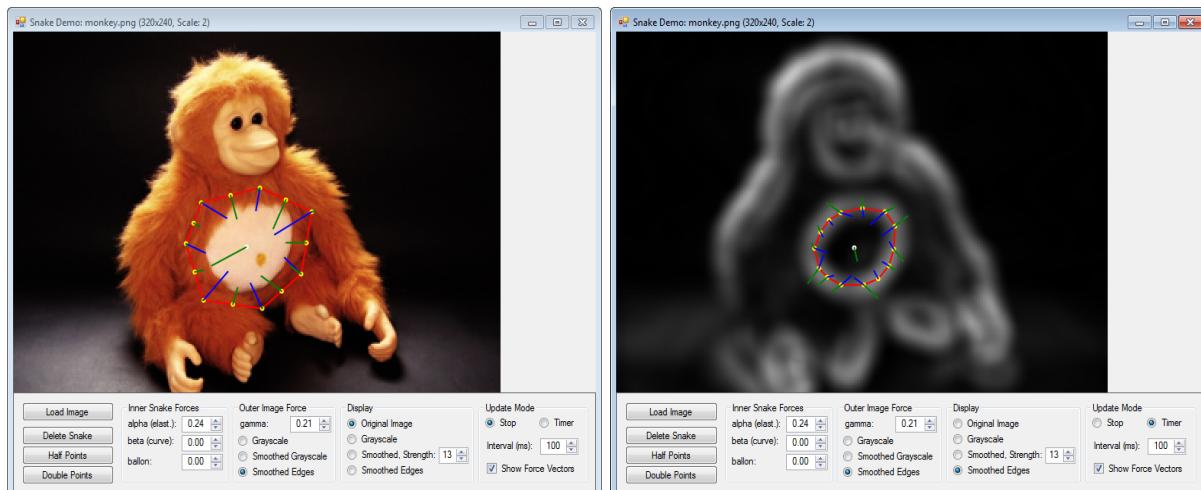


Abb. 233: Links: Startkontur im inaktiven Zustand. Rechts: Die aktive Kontur, welche sich auf dem Gipfel der Gradienten des Bauchflecks einpendelt.

8.4.2.5 Übung Snakes

Spielen Sie mit der *SnakesDemo*-Applikation.

- Justieren Sie die verschiedenen Parameter. Insbesondere die Faktoren *alpha* und *gamma* verändern die interne und externe Energie die auf die Kontur wirken.
- Probieren Sie andere Bilder aus.

Studieren Sie das Projekt im VisualStudio 2010 oder 2012.

- Studieren Sie die Methoden *update*, *calculateForce* und *draw* der Klasse Snake.
- Berechnen Sie in der Methode *calculateForce* zusätzlich folgende Kraftvektoren:
- *F_curv* als Krümmungsausgleichskraft und eine Ballonexpansionskraft. Sie soll bewirken, dass alle Knicke möglichst ausgeglichen sind.
- *F_ball* als eine nach aussen wirkende Ballonexpansionskraft.

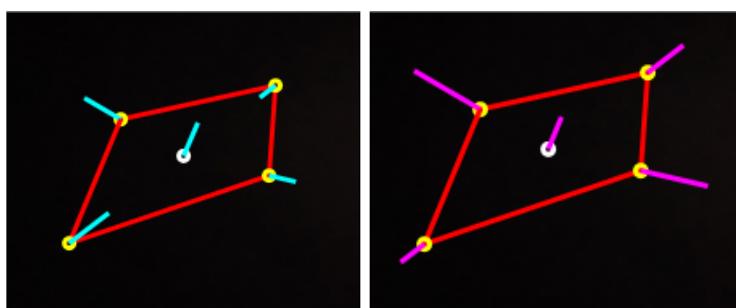


Abb. 234: Links: Krümmungsausgleichskräfte (*F_curv*) in Cyan.
Rechts: Ballonexpansionskräfte (*F_ball*) in Magenta.

8.4.2.6 Snakes in ImageJ und Matlab

Für **ImageJ** gibt es zwei Plugins für Snakes:



- [SplineSnake](#) von der EPFL Lausanne
- [Active Contour \(Snake\)](#) von Philippe Andrey und Thomas Boudier

Für **Matlab** gibt es:



- eine Funktion [activecontour](#) für Snakes
- eine [Testapplikation für Snakes](#)

8.5 Bewegungsbasierte Segmentation

8.5.1 Bewegung vor statischem Hintergrund

Zeichnet sich eine Region in einer Bildsequenz dadurch aus, dass sie sich von einem Bild zum nächsten bewegt, so können wir sie durch den Absolutwert der Differenz bestimmen. Dieses Bedürfnis hat man meistens in Überwachungsvideos, die von einer fest installierten Kamera aufgenommen werden. Solche Videos erfüllen die Bedingung, dass sich nur bewegte Objekte im Bild verändern und der Hintergrund mit Ausnahme von Rauschen und relativ langsamem Beleuchtungsänderung unverändert bleibt.

Wir können bewegte Objekte durch zwei verschiedene Differenzen bestimmen:

- Absolute Differenz zweier benachbarter Bilder in der Sequenz. Alle Pixel, deren absolute Differenz grösser als ein bestimmter Schwellwert T ist, gelten als bewegt:

$$|G_{i-1}(x, y) - G_i(x, y)| > T$$
- Absolute Differenz zwischen einem Bild aus der Sequenz und einem Hintergrundsbild G_{BG} (*background subtraction*):

$$|G_{BG}(x, y) - G_i(x, y)| > T$$



Beispiel mit Matlab

Das nachfolgende Matlab-Skript *Motion1.m* lädt eine AVI-Videosequenz und zeigt in einer Schlaufe das absolute Differenzbild zum Vorgängerbild an:

```
% create video object and read number of frames
video = VideoReader('..../Videos/street.avi');
numF = video.NumberOfFrames;

% get first frame in sequence and convert to gray-scale
lastFrame = rgb2gray(read(video, 1));

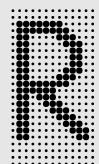
% Open window and hold on
figure, hold on;

% loop through the video sequence and display absolute difference
for fNr=2 : numF
    thisFrame = rgb2gray(read(video, fNr));
    absDiff = abs(im2double(lastFrame) - im2double(thisFrame));
    imshow(absDiff);
    pause(0.2);
    lastFrame = thisFrame;
end
```

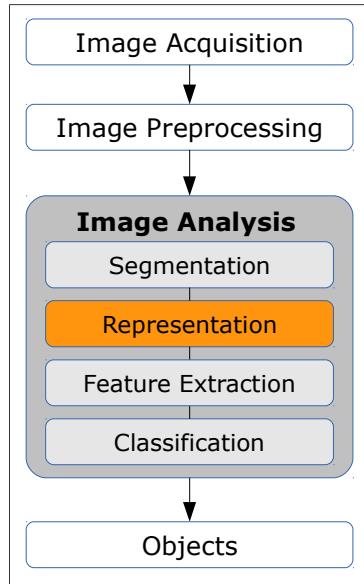


Abb. 235: Bild 32 und 33 aus der Videodatei *street.avi* und rechts das absolute Differenzbild davon.

9 Repräsentation



Regionen, die wir durch die Segmentierung gewonnen haben, werden mit Ausnahme der modellbasierten Segmentierung weiterhin in Rasterform dargestellt. Für die nachfolgenden Schritte der Merkmalsextraktion und Klassifikation müssen wir eine geeignete Repräsentation finden, die eine einfachere und effizientere Verarbeitung erlauben. Als Erstes werden wir die Regionen mit *Region Labeling* zählen und markieren, um sie danach durch *Masken*, *Lauflängen*, *Konturen*, *Graphen* oder *Skelette* zu repräsentieren.



9.1 Region Labeling

Das Ziel des *Region Labeling* ist es, zusammenhängende Vordergrundobjekte (Engl. *connected components* oder *blobs* (von *binary large object*)) zu markieren und für spätere Verarbeitungsschritte unterscheidbar zu machen. Region Labeling ist noch keine eigentliche Repräsentation der Region, sondern nur ein Vorbereitungsschritt.

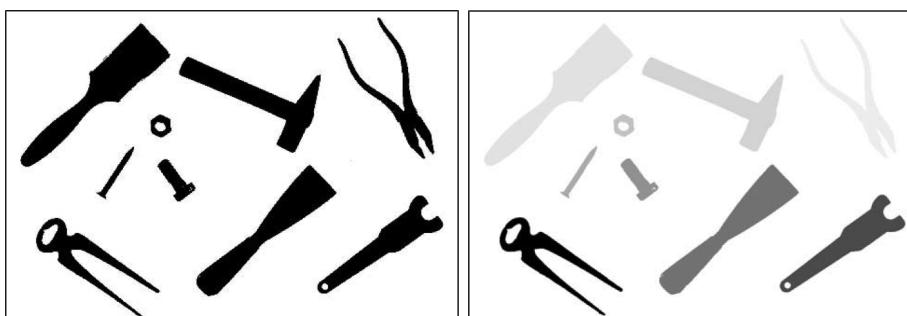


Abb. 236: Vordergrundsobjekte durch unterscheidbare Graustufen markiert (Quelle: Burger06).

Das *Region Labeling* iteriert dabei über alle Pixel eines binären Bildes (0=Hintergrund, 1=Vordergrund) und flutet mit der *FloodFill* Routine alle Vordergrundpixel einer Region mit einem Label.

```

RegionLabeling(I)
label=2
Iterate over all image coordinates u,v
if I(u,v) = 1 then
  FloodFill(I, u, v, label)
  
```

```

    label = label + 1
    return

```

Es ist naheliegend die [FloodFill](#) Routine als rekursive Prozedur zu implementieren, bei der für jedes Nachbarschaftspixel wiederum [FloodFill](#) aufgerufen wird:

```

FloodFill(I,u,v,label)
  if coordinate u,v is within image boundaries and I(u,v) = 1 then
    Set 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)
  return

```

Rekursive Algorithmen sind zwar elegant aber nicht sehr performant, insbesondere, wenn die Rekursionstiefe gross wird. Bei jedem Funktionsaufruf müssen die lokalen Variablen auf den Stack-Speicher gelegt werden. Dieser ist nicht beliebig gross und bei gewissen Sprachen wie. z. B. Java, sogar begrenzt auf ca. 10'000 rekursive Funktionsaufrufe.

Deshalb sollten wir rekursive Algorithmen mit grosser Tiefe in eine iterative Variante umschreiben. Für unseren Fall verwenden wir eine Queue, um die Nachbarn eines Vordergrundpixels aufzusammeln:

```

FloodFill(I,u,v,label)
  Create an empty queue Q
  Enqueue(Q, u,v)
  while Q is not empty do
    Dequeue(Q, x,y)
    if coordinate x,y is within image boundaries and I(x,y) = 1 then
      Set I(x,y) = label
      Enqueue(Q, x+1, y )
      Enqueue(Q, x , y+1)
      Enqueue(Q, x , y-1)
      Enqueue(Q, x-1, y )
  return

```



In **ImageJ** finden Sie das Region Labeling innerhalb der [Plugin-Sammlung](#) zum Buch von [Burger06]. Im selben Buch finden Sie zudem mehr Details zur genauen Umsetzung in Java. Alternativ dazu gibt es das Plugin [ijblob](#), welches noch viele zusätzliche Funktionen zur Merkmalsextraktion zur Verfügung stellt (s. auch Beispiel im Kapitel 10.6).



In **Matlab** ist das Region Labeling in der Funktion [bwlabel](#) implementiert. Alternativ dazu, und gemäss Matlab-Dokumentation weniger speicherintensiv, soll die Kombination der Funktionen [bwconncomp](#) und [labelmatrix](#) sein.

9.2 Masken



Eine Maske ist eine binäre Matrix exklusiv für eine Region. Algorithmisch haben wir dadurch nicht viel gewonnen. In den meisten Bildverarbeitungsprogrammen ist die Maske jedoch ein wichtiges Werkzeug, um Operationen nur auf eine Region anzuwenden. In ImageJ können Selektionen ([Edit > Selections > Convert to mask](#)) oder Resultate von binären Operationen ([Process > Binary > Convert to mask](#)) zu Masken konvertiert werden. Der Vorteil der Maske gegenüber anderen Repräsentationen liegt in der Einfachheit der Datenstruktur. Ein weiterer Vorteil ist, dass ein 8-Bit-Pixelwert für Teiltransparenz ([Alpha Blending](#)) verwendet werden kann.

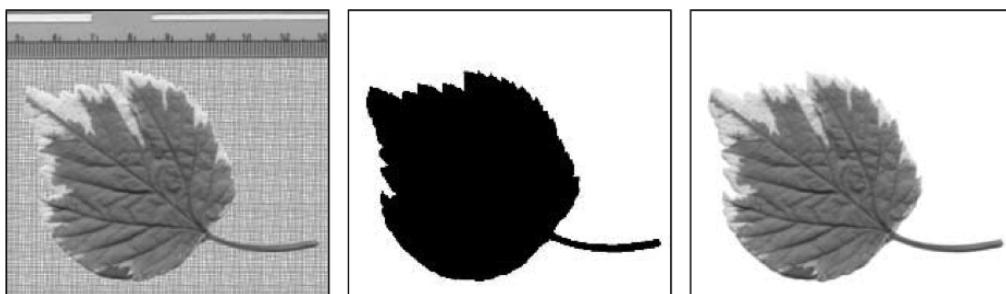


Abb. 237: Originalbild links, Binärmaske in der Mitte und die freigestellte Region rechts. (Quelle: Burger06)

9.3 Lauflängen

RLE oder *Run Length Encoding* ist ein einfacher Kompressionsalgorithmus und wird genauer im Kapitel 12.2.1 behandelt. Dabei wird z. B. eine Zeile mit schwarzen und weissen Pixeln über deren Lauflängen zusammengefasst:

WWWWWWWWSSSSWWWWSSSSSSSSSSSSWWSSS: RLE: 8,3,5,15,3,3

RLE eignet sich gut für 1D-Signale und wurde deshalb auch für die Kompression im Fax-Standard eingesetzt. Für mehr Information und Implementationsdetails zur Lauflängencodierung empfehle ich [[Nischwitz11](#)].

9.4 Baum- und Graphstrukturen

Bei der Split and Merge Segmentierung (s. 8.2.2) haben wir die *Quad-Tree*-Struktur kennengelernt, mit deren ein Bild aufgesplittet wird und zusammen mit dem *Nachbarschaftsgraphen* ähnliche Knoten zu Regionen zusammengefasst werden. Kombiniert repräsentieren diese beiden Strukturen alle Regionen eines Bildes.

9.5 Konturen



Burger stellt in seinem Buch für **ImageJ** einen Algorithmus *ContourTracing* vor (Kapitel 11), mit dem man gleichzeitig die Regionen markieren und die Konturen zurückerhält. Der Algorithmus gibt die Konturen getrennt nach externen und internen Konturen zurück und kann auch mit 1-Pixel breiten Regionen umgehen.

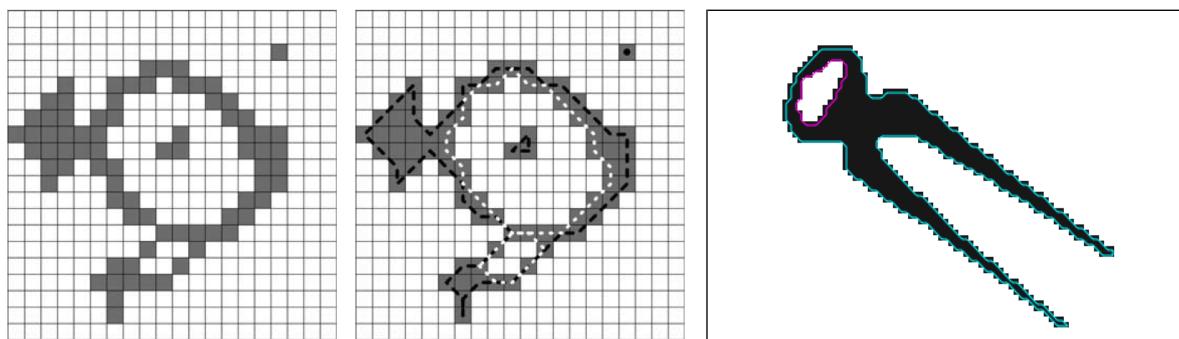


Abb. 238: Externe und interne Konturen aus dem Imaging Book Plugin von [Burger06]



In **Matlab** steht der *Contour Tracing* Algorithmus ohne Löcher in der Funktion [bwtraceboundary](#) zur Verfügung. Die Funktion [bwboundaries](#) liefert zusätzlich die Konturen der Löcher zurück.

9.5.1 Chain Code

Chain Code oder *Freeman Code* eignen sich im Vergleich zu RLE gut die 2D-Struktur einer Region zu repräsentieren. Ausgehend von einem Startpixel wird die innere Kontur einer

Region im Gegenuhrzeigersinn codiert. Man kann dazu entweder einen 2-Bit-Code für die 4er-Nachbarschaft oder einen 3-Bit-Code für die 8er-Nachbarschaft verwenden. In [Burger06] finden Sie weitere Varianten des Chain Codes.

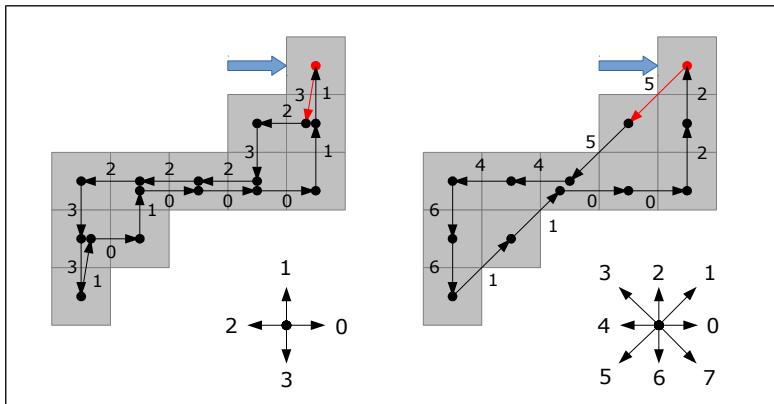


Abb. 239: 4er-Chain Code links: $C_{h4}=3232223310100011 = 16 \text{ Zeichen à } 2 \text{ Bit} = 32 \text{ Bit}$
 8er-Chain Code rechts: $C_{h8}=554466110022 = 12 \text{ Zeichen à } 3 \text{ Bit} = 36 \text{ Bit}$

9.5.2 Crack Code

Der Crack Code ist eine alternative Repräsentation einer Kontur und hat den Vorteil, dass auch Linien und ein einzelnes Pixel eine korrekte Fläche erhalten. Im Unterschied zum Chain Code verläuft die Konturlinie nicht durch das Zentrum der Randpixel, sondern im Spalt (= Crack) zwischen den Vordergrundpixeln und den Hintergrundpixeln hindurch. Es gibt wiederum eine Variante für die 4er- und die 8er-Nachbarschaft.

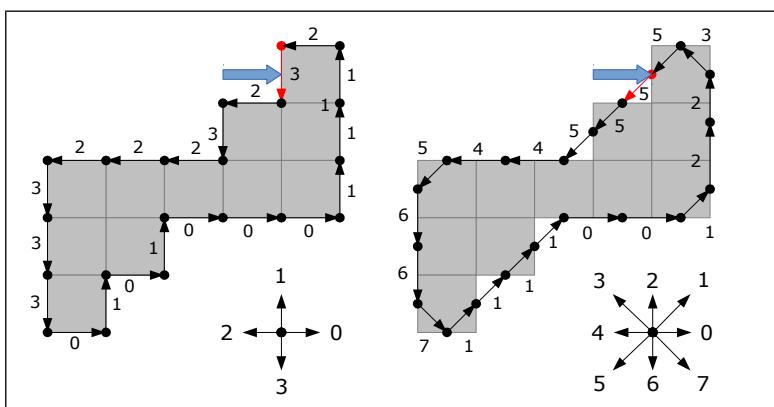


Abb. 240: 4er-Crack Code links: $C_{r4}=32322233301010001112 = 19 \text{ Zeichen à } 2 \text{ Bit} = 38 \text{ Bit}$
 8er-Crack Code rechts: $C_{r8}=55544566711110012235 = 19 \text{ Zeichen à } 3 \text{ Bit} = 57 \text{ Bit}$

Nachfolgend der Code, mit dem der 4er Crack Code für eine Region beginnend bei $[x \text{ start}, y \text{ start}]$ ermittelt werden kann. Vom aktuellen Pixel A ausgehend entscheiden wir in einer Schlaufe, ob wir zum linken Pixel L, geradeaus zum Pixel R oder nach rechts abbiegen müssen.

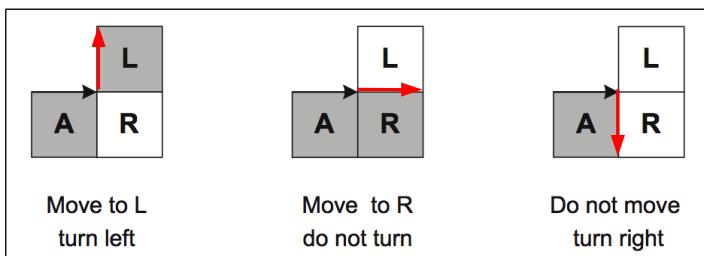


Abb. 241: 3 Mögliche Richtungen beim aktuellen Pixel A. Quelle: [Cattin12]

```
// define the directions of L and R, depending on the last
```

```

// crack code 0 1 2 3
int xR[4] = { 1, 0, -1, 0 };
int yR[4] = { 0, -1, 0, 1 };
int xL[4] = { 1, -1, -1, 1 };
int yL[4] = { -1, -1, 1, 1 };

unsigned char code = 1;
int x = x_start, y = y_start;

do
{
    if(pixel(x+xL[code], y+yL[code]))
    {
        x += xL[code];           // move to L
        y += yL[code];
        code = (code + 1)%4; // turn left
        c[i++] = code;
    }
    else if(pixel(x+xR[code], y+yR[code]))
    {
        x += xR[code];           // move to R
        y += yR[code];
        c[i++] = code;
    }
    else
    {
        code = (code + 3)%4; // turn right
        c[i++] = code;
    }
}
while(!(x==x_start && y==y_start && code==1));

```

9.5.3 Signaturen

Die Konturen sind bis jetzt gar nicht invariant und die Chain Codes nur sind nur invariant gegenüber Translation. Wenn wir Regionen aber miteinander vergleichen wollen, so müssen sie auch invariant gegenüber Rotation und Skalierung sein. Dazu verschieben wir die Konturpixel um den Ursprung, indem wir den Schwerpunkt abziehen. Danach wandeln wir die kartesischen Koordinaten der Konturpixel in Polarkoordinaten um. Der resultierende Vektor in Polarkoordinaten mit der Länge der Anzahl Konturpixel wird *Signatur* genannt.



Im nachfolgenden Beispiel mit **Matlab** (*SignatureTest.m*) werden zwei Bilder (s. Abb. 242) mit derselben Region aber mit unterschiedlichen Transformationen eingelesen. Die Signatur erhalten wir über die Funktion *signature* (von Gonzales08). Sie liefert zwei Vektoren mit den Radien und den Winkeln sowie die Koordinaten des Schwerpunkts zurück. Wir können nun die Signaturen gemeinsam als Funktion über den Winkel 0-360° in einem Plot darstellen (s. Abb. 243 links).

Die Radien haben die Einheit Pixel und sind damit nicht vergleichbar. Wir dividieren sie deshalb durch den jeweils max. Radius und erhalten so Radien zwischen 0 und 1 (s. Abb. 243 rechts).

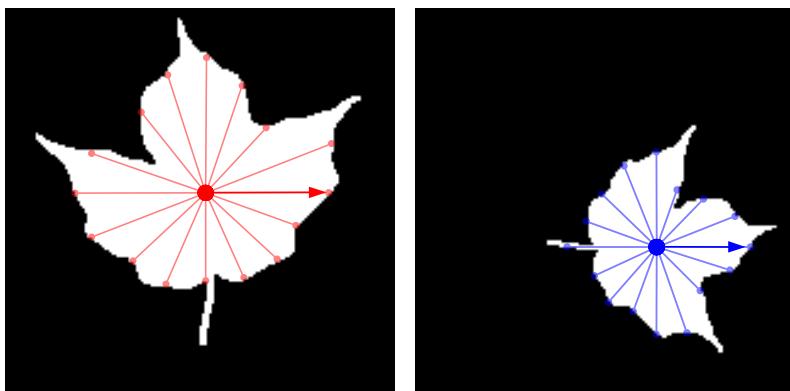


Abb. 242: Zwei gleiche Regionen mit unterschiedlichen Transformationen. Die beiden Pfeile stellen jeweils den kürzesten Radius vom Schwerpunkt zur Kontur dar.

```
%reads images into I2 & I2
```

```

I1 = imread('..../Images/leaf_bw1.tif');
I2 = imread('..../Images/leaf_bw2.tif');

% Get the boundary pixels & keep only the main outer bound
b1 = bwboundaries(I1, 'noholes'); b1 = b1{1};
b2 = bwboundaries(I2, 'noholes'); b2 = b2{1};

% Get the signature vector & the center of mass
[r1, phi1, x01, y01] = signature(b1);
[r2, phi2, x02, y02] = signature(b2);

% Divide by the max. radius to make it scale invariant
r1 = r1 / max(r1);
r2 = r2 / max(r2);

```

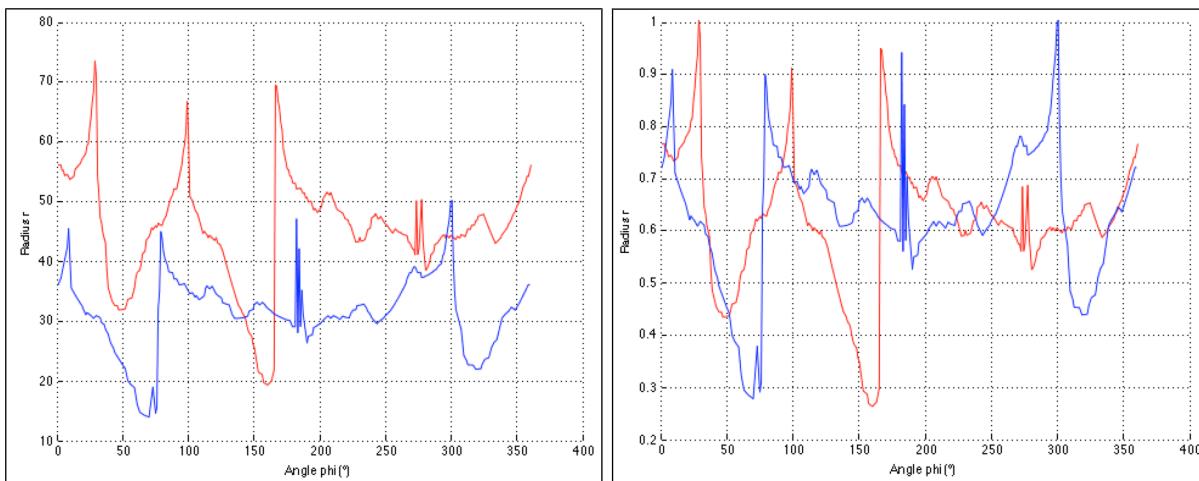
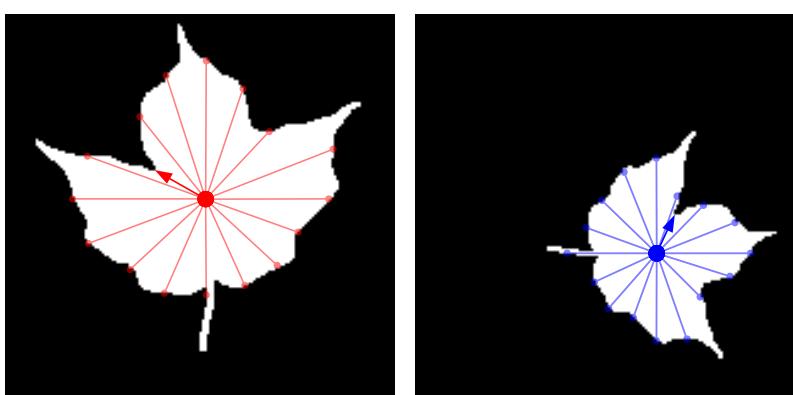


Abb. 243: Links: Signaturen der beiden Regionen in Polarkoordinaten vor der Skalierung. Rechts dieselben nach der Skalierung.

Wir können die Signaturen noch nicht miteinander vergleichen, weil die Polarkoordinaten jeweils beim Winkel 0 beginnen und so die unterschiedliche Rotation nicht berücksichtigen. Um dieselbe Ausrichtung zu erhalten, gibt es verschiedene Möglichkeiten:

- Wir suchen den jeweils kleinsten oder grössten Radius und verschieben die Radien im Vektor so, dass diese am Anfang des Vektors stehen (s. Abb. 244 links).
- Wir suchen der Winkel der Hauptachse der Region und verschieben die Radien dementsprechend. Wir können dies mit der Hauptkomponentenanalyse machen, wie dies im Kapitel 10.2.2.1 werden wird.

Beide Varianten sind nur bei asymmetrischen Regionen eindeutig. Für symmetrische Regionen ist eine Ausrichtung nicht möglich aber auch nicht notwendig. Schwierig sind Regionen mit fast symmetrischen Ausrichtungen und Regionen mit Löchern, welche durch die äussere Kontur nicht berücksichtigt werden.



Als letzten Schritt zur kompletten Invarianz müssen wir die beiden Vektoren gleich lang machen, weil die beiden Signaturen ja noch unterschiedlich lang sind. Wir interpolieren dazu die beiden Signaturen und tasten die 360° gleichmäßig alle 6° ab (s. Abb. 244 rechts).

```
% Circular shift the radii so that the smallest is in front
% This makes the signature invariant to rotation
min_r1 = find(r1 == min(r1));
min_r2 = find(r2 == min(r2));
r1 = circshift(r1, -min_r1);
r2 = circshift(r2, -min_r2);

% Interpolate the signature so that both have the same length
phi6 = 0:6:360;
r1 = interp1(phi1, r1, phi6, 'cubic');
r2 = interp1(phi2, r2, phi6, 'cubic');
phi1 = phi6;
phi2 = phi6;

% Plot both signatures
plot(phi1, r1, 'r');
plot(phi2, r2, 'b');
```

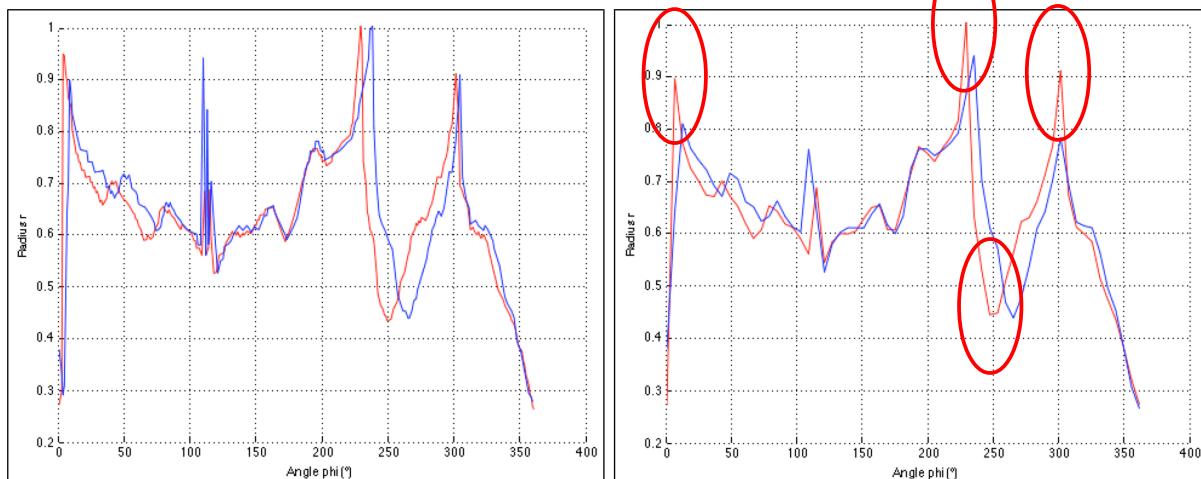


Abb. 244: Links: Signaturen nach der Ausrichtung zum kleinsten Radius. Rechts: Nach der gleichmässigen Interpolation haben die beiden Signaturen die gleiche Länge.

Wir haben nun beide Signaturen gleich lang und invariant gegenüber Translation, Rotation und Skalierung. Wir erkennen die Ähnlichkeit gut und sehen aber auch die Differenzen, welche durch die Quantisierung bei der Rotation und Skalierung entstanden sind. Für eine spätere Klassifikation (Kapitel 11) reicht meist eine Überprüfung einiger Extrempunkte (Maxima, Minima oder starke Knicke) als Merkmale aus.

9.5.4 Fourier Deskriptoren

Im Kapitel 7.1 haben wir die Fourier Transformation kennengelernt und gesehen, wie sie ein periodisches Signal in eine Summe von Sinus- und Kosinusfrequenzanteilen zerlegen kann. Eine Kontur einer Region ist ebenfalls ein sich wiederholendes Signal und es liegt deshalb nahe, sie durch Frequenzanteile zu beschreiben.

In der nachfolgenden Darstellung wird die Kontur von einem Startpunkt aus in einen Vektor aus komplexen Zahlen geschrieben. Die komplexen Zahlen des Vektors k werden durch die x-Koordinate im Realteil und durch die y-Koordinate im Imaginärteil gebildet.

$$k(l) = x(l) + i \cdot y(l)$$

Damit haben wir ein zweidimensionales Signal in ein eindimensionales Signal umgewandelt. Dieses kann somit direkt mit der diskreten 1D-FT transformiert werden. Weil die FT invertierbar ist, können wir die Konturlinie daraus wieder fehlerfrei rekonstruieren.

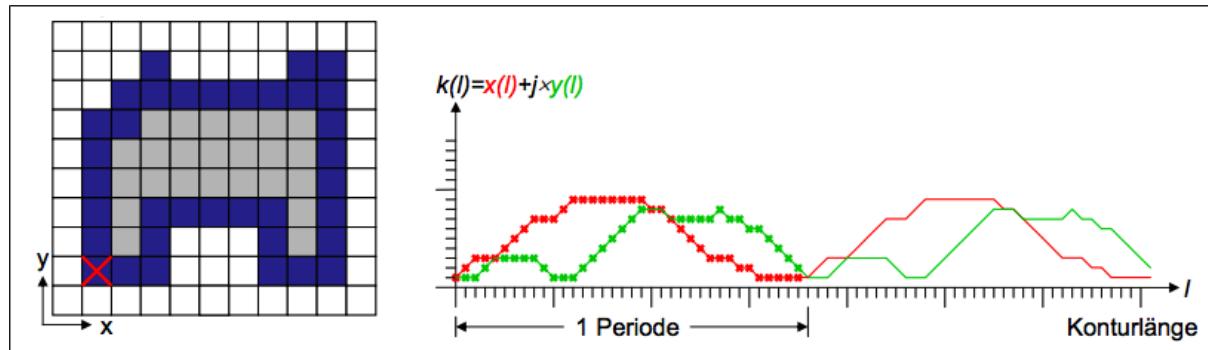


Abb. 245: x- und y-Koordinaten der Konturlinie (Quelle: B. Lang, HS Osnabrück)

Da mit der Konturlinie oft eine Klassifikation durchgeführt wird, um ähnliche Objekte in einem Bild zu suchen, müsste die Kontur zweier identischer Objekte, die aber ungleich gross sind, gleich häufig abgetastet werden.

Man tastet die Kontur deshalb nicht bei jedem Pixel ab, sondern in regelmässigen Abständen und erhält so K Konturpunkte. Zwischenpixelpositionen werden bilinear interpoliert:

$$s(k) = x(k) + i \cdot y(k) \text{ mit } k=0, 1, 2 \dots K$$

Diese Reihe $s(k)$ aus komplexen Konturpunkten übergibt man der diskreten FT und erhält eine Reihe $a(u)$ mit K komplexwertigen Fourier-Koeffizienten (= Fourier-Deskriptoren):

$$a(u) = \frac{1}{K} \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K}$$

Wir können die Konturpunkte mit der inversen diskreten FT (IDFT) wieder rekonstruieren:

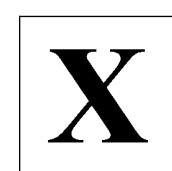
$$s(k) = \sum_{u=0}^{K-1} a(u) e^{j2\pi uk/K}$$

Je mehr Fourier-Koeffizienten wir für die Rekonstruktion verwenden, umso genauer kann eine Kontur mit scharfen Ecken reproduziert werden.

Beispiel mit Matlab



Das nachfolgende Matlab-Skript wird ein binäres Bild des Buchstabens X eingelesen und die Kontur mit verschiedenen Anzahlen an Koeffizienten rekonstruiert. Es verwendet die Hilfsfunktionen `bound2im.m`, `frdescp.m` und `ifrdescp.m` aus der [DIPUM-Toolbox](#) von [Gonzales08]. Die Funktion `bwboundaries` liefert eine Kontur mit 858 Pixeln zurück. Für das Beispiel werden der Einfachheit halber alle Konturpixel transformiert. Die Hilfsfunktion `bound2im` kann aus einer Kontur wieder ein Bild machen.



```
I1 = imread('.../Images/X.png'); % reads image into I
w = size(I1,1);
h = size(I1,2);

% Get boundary pixel coordinates without holes
b = bwboundaries(I1, 'noholes');

b = b{1}; % There is only one boundary

% Recreate image from boundary
I2 = bound2im(b, w, h); figure; imshow(I2);

% Create fourier descriptors
a = frdescp(b);
```

```
% Reconstruct boundary from fourier descriptors
% with increasing No. of coefficients
s002=ifrdescp(a, 2); I002=bound2im(s002, w, h); figure; imshow(I002);
s004=ifrdescp(a, 4); I004=bound2im(s004, w, h); figure; imshow(I004);
s008=ifrdescp(a, 8); I008=bound2im(s008, w, h); figure; imshow(I008);
s016=ifrdescp(a, 16); I016=bound2im(s016, w, h); figure; imshow(I016);
s032=ifrdescp(a, 32); I032=bound2im(s032, w, h); figure; imshow(I032);
s064=ifrdescp(a, 64); I064=bound2im(s064, w, h); figure; imshow(I064);
s128=ifrdescp(a,128); I128=bound2im(s128, w, h); figure; imshow(I128);
```

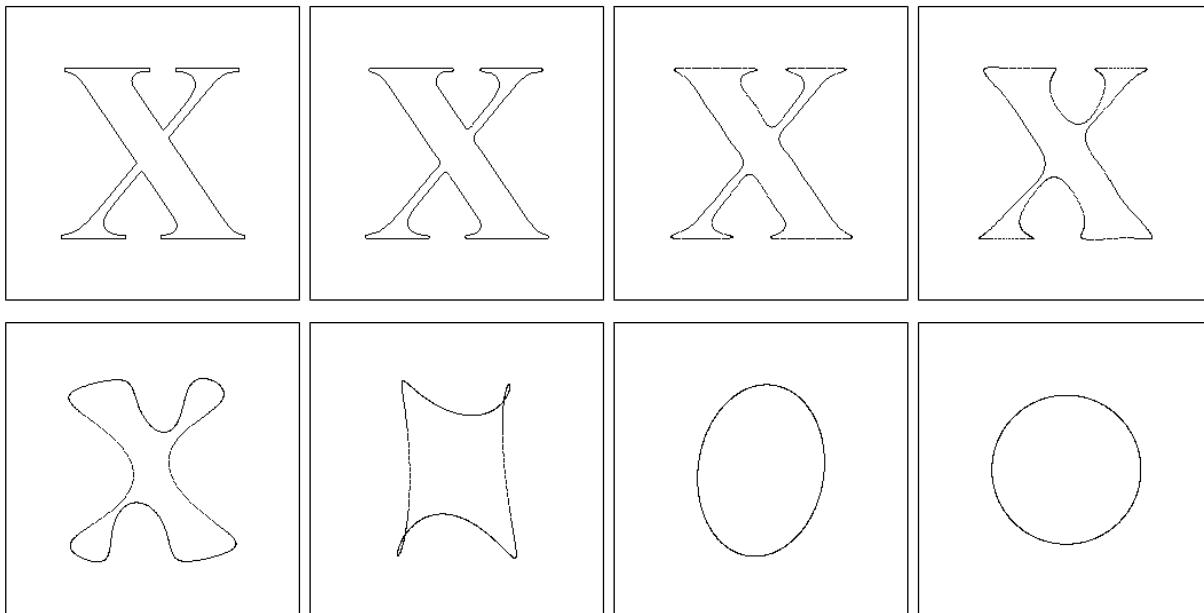


Abb. 246: Von oben links nach unten rechts: Vollständige Kontur mit 858 Pixeln. Danach Rekonstruktionen mit 128, 64, 32, 16, 8, 4 und 2 Fourier-Koeffizienten. Die rekonstruierten Konturen können wegen Fliesskomma-Ungenauigkeiten Lücken von einem Pixel aufweisen. Diese könnten mit einem morphologischen Filter geschlossen werden.

Eine schöne Animation einer Summe aus Ellipsenbewegungen finden Sie [hier](#).

Invarianz

Bis jetzt sind diese Fourier Deskriptoren nicht invariant gegenüber Translation, Rotation und Skalierung. Wir können alternativ zu den kartesischen Koordinaten die Signatur der Kontur mit Polarkoordinaten verwenden und diese zusätzlich invariant machen, so wie wir es im Kapitel der Signaturen gemacht haben.

Fourier-Deskriptoren lassen sich aber auch selbst durch Korrekturen invariant machen:

- **Translationsinvarianz:** Die Position betrifft nur den ersten Koeffizienten $a(0)$. Durch eine Verschiebung in den Ursprung erhalten wir Translationsinvarianz.
- **Skalierungsinvarianz:** Um die Skalierungsinvarianz zu erreichen, müssen wir nur alle Koeffizienten durch den Betrag des 1. Koeffizienten $a(1)$ dividieren.
- **Rotationsinvarianz:** Wird eine Kontur um den Winkel φ gedreht, so werden alle Fourier-Koeffizienten um den Faktor $e^{i\varphi}$ multipliziert. Um die Rotationsinvarianz zu erreichen, können wir von allen Koeffizienten den Phasenwinkel des 1. Koeffizienten $a(1)$ subtrahieren.

Mehr Informationen zu Fourier-Deskriptoren finden Sie bei den Standardwerken [[Gonzales08](#)] und [[Jähne05](#)]. Die in diesem Abschnitt verwendeten Formeln stammen aus [[Gonzales08](#)] und unterscheiden sich von denen in [[Jähne05](#)] durch eine unterschiedliche Indexierung der Koeffizienten.

9.6 Skelett

Bei manchen Regionen wie z. B. bei Buchstaben, kann das Skelett eines Buchstabens sogar besser sein für die Identifikation (*OCR = Optical Character Recognition*) als die Region oder die Kontur des Buchstabens. Es gibt jedoch keine eindeutige Definition, was das Skelett einer Region ist. Es scheint etwas Wesentliches zu sein, eine Art Grundgerüst einer Region. Das Skelett ist überall dort hilfreich, wo wir an der Struktur des Grundgerüsts mehr interessiert sind als an der ursprünglichen Erscheinung.

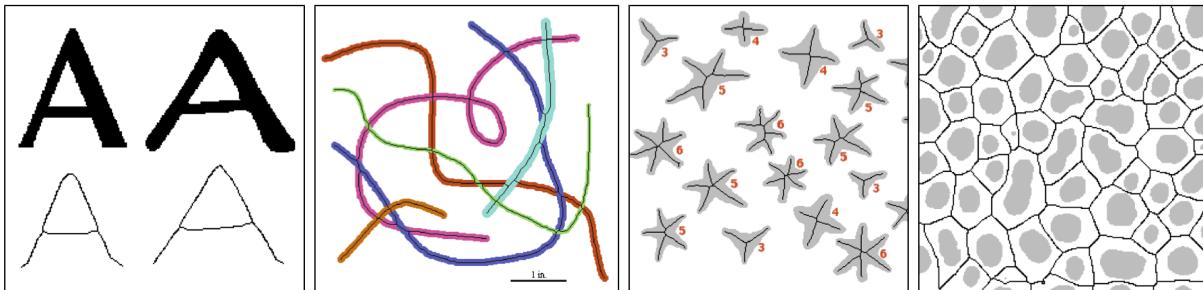


Abb. 247: v.l.n.r.: Gleiche Struktur des Buchstabens A unabhängig von der Schriftart. Bei Linien ist man oftmals nur an der Zentrumslinie interessiert. Bei den Sternen wollte man die Anzahl Zacken wissen. Manchmal ist es auch sinnvoll den Bildhintergrund zu skelettieren, um z. B. die Zellwände zu bestimmen.

Einen guten Vergleich von verschiedenen Algorithmen zur Skelettierung finden Sie in der Bachelorarbeit von Christoph Bullmann [[Bullmann08](#)]. Er unterscheidet 3 Kategorien von Algorithmen, um ein Skelett zu erhalten:

- Auf **Distanztransformation** basierend:
 - Mediale Achse
 - Distanzskelett
- Auf **kritischen Punkten** basierend:
 - Einfacher kritischer Punkteansatz
 - Triangulation
- Auf **Ausdünnung** basierend:
 - Morphologisches Ausdünnen
 - Sequenzielles Ausdünnen
 - Paralleles Ausdünnen

9.6.1 Distanztransformation

Bei der Distanztransformation wird jedem Pixel einer Region im Vordergrund seine kürzeste Distanz zum Hintergrund zugewiesen. Die Skelettlinie ist demnach die Menge aller Punkte, die nicht von einem minimalen Weg durchlaufen werden. Im Vergleich zu den Ausdünnungsalgorithmen liefern die auf der Distanztransformation basierenden Algorithmen jedoch schlechtere Ergebnisse, weshalb nicht weiter darauf eingegangen wird. Wichtig für das Gesamtverständnis sind jedoch die Begriffe der *medialen Achse* und der *minimalen Distanz*.

Mediale Achse

Die *mediale Achse* ist die Linie der Mittelpunkte aus Kreisen mit maximalem Radius, die an mindestens zwei Punkten den Rand der Region berühren. Diese Definition wird häufig als die perfekte Definition für ein Skelett angegeben. Im diskreten Rasterbild kann diese Definition jedoch nicht perfekt umgesetzt werden, da eine Region mit einer geraden Anzahl Pixel kein Pixel in der Mitte hat.

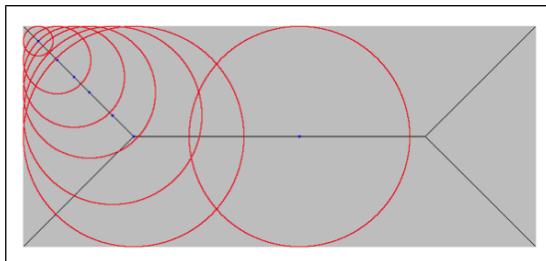


Abb. 248: Rechteck mit dem Skelett aus medialen Achsen.

Minimale Distanz

Die *minimale Distanz* wird als kürzester Weg von einem Punkt in der Region zum Rand der Region interpretiert.

Diese Distanz zwischen zwei Punkten p und q kann auf 3 Arten gemessen werden:

- **Euklidische Distanz:** Diese Distanz bemisst die kürzest mögliche direkte Distanz im euklidischen Raum: $\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$
- **Manhattan Distanz:** Diese Distanz erhält ein Taxifahrer in Manhattan, wenn er um die rechtwinkligen Blöcke Manhattans fahren muss. Sie entspricht der Summe der absoluten Differenzen der Einzelkoordinaten: $|p_x - q_x| + |p_y - q_y|$
Diese Distanz wird auch *Taxidistanz* oder *Minkowski-Distanz* genannt.
- **Schachbrett Distanz:** Diese Distanz entspricht der grösseren der beiden absoluten Differenzen der Einzelkoordinaten: $\max(|p_x - q_x|, |p_y - q_y|)$
Die Bezeichnung kommt aus dem Schachspiel, wo diese Distanz die min. Anzahl Züge für den König angibt, um zu einem anderen Feld zu kommen.

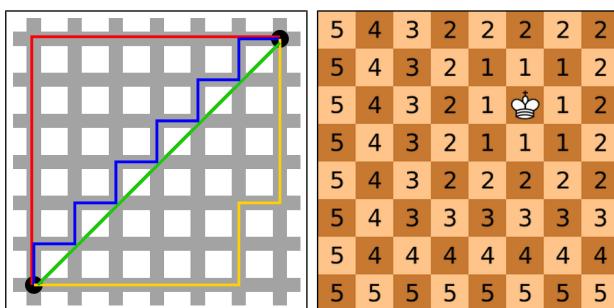


Abb. 249: Links verschiedene, gleich lange Manhattan-Distanzen im Vergleich zur euklidischen Distanz (grün). Rechts: In allen Felder steht die Schachbrettdistanz zum König. (Bildquelle: Wikipedia)

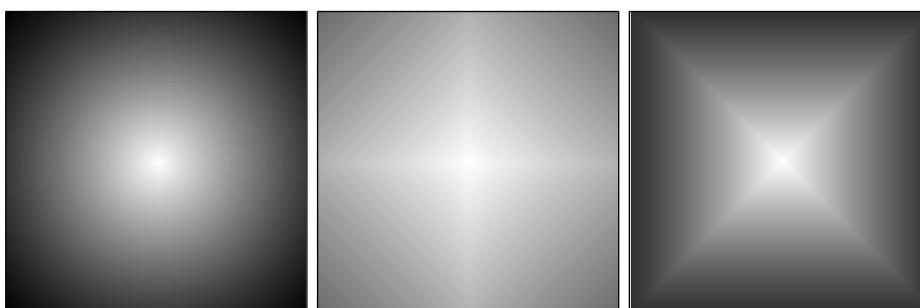


Abb. 250: Ein schwarzes Quadrat nach der Distanztransformation mit der euklidischen, der Manhattan-, und der Schachbrettdistanz.

9.6.2 Ausdünnung (Thinning)

Das am meisten verwendete Verfahren zur Reduktion einer Region auf sein Skelett ist als *Thinning* bekannt und ein Algorithmus aus der Morphologie. Der Thinning Operator \otimes zwischen einer binären Region A und einem Set von Strukturelementen B ist wie folgt definiert:

$$A \otimes B = A - (A \otimes B)$$

$A \otimes B$ ist darin eine *Hit-Or-Miss*-Operation (s. Kapitel 6.2.1.8). Diese wird allerdings ohne den Miss-Teil gemacht. Das scheint soweit einfach. Kompliziert wird es, weil B aus einem Set von 8 Strukturelementen besteht (B^1 - B^8):

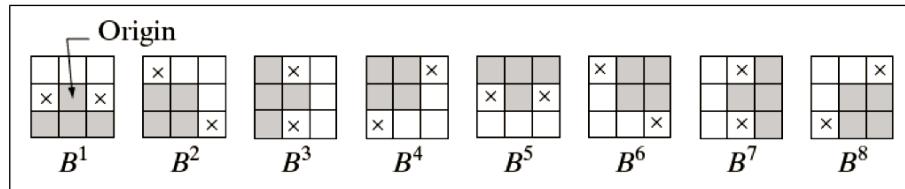


Abb. 251: Die 8 SE enthalten *Don't-care-Elemente* (x) bei denen es egal ist, was sich im darunterliegenden Bild befindet. Die 8 SE entsprechen 8 Himmelsrichtungen N, NO, O, SO, S, SW, W und NW, bei welchen jeweils das Pixel unter dem Zentrum gelöscht wird.

Das Bild wird solange mit dem jeweils nächsten Strukturelement verknüpft, bis sich das Resultat nicht mehr ändert.

$$A \otimes B^n = (((((((((A \otimes B^1) \otimes B^2) \otimes B^3) \otimes B^4) \otimes B^5) \otimes B^6) \otimes B^7) \otimes B^8) \otimes B^1) \dots) \otimes B^n$$

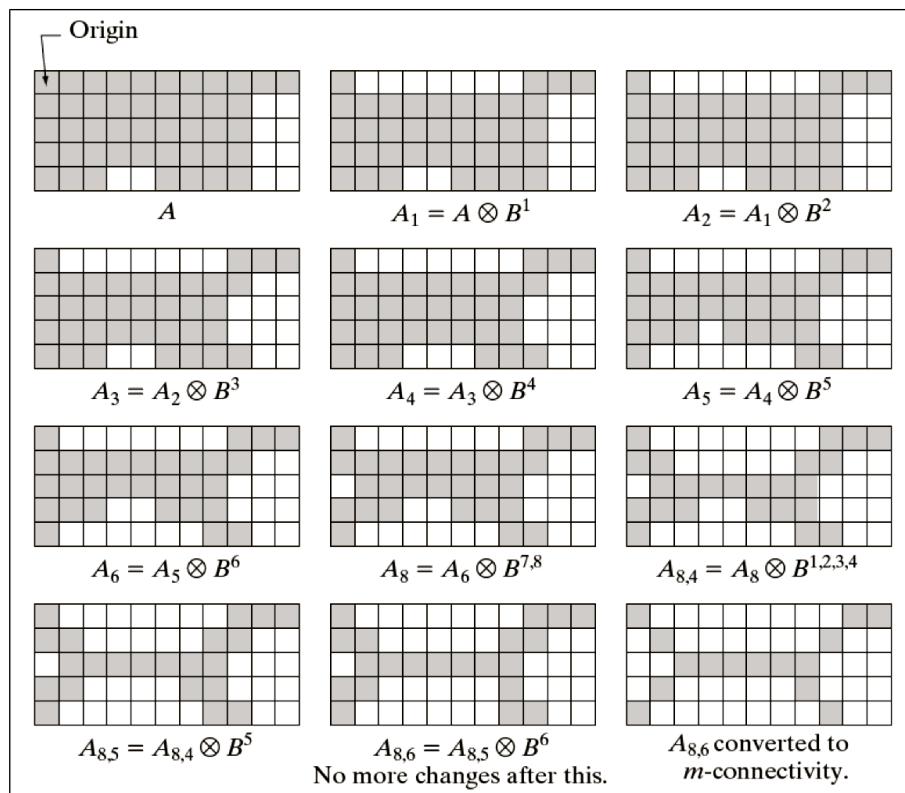


Abb. 252: Oben-Mitte bis 3. Zeile Mitte sind die Resultate nach den ersten 8 Verknüpfungen. Die Verknüpfung mit dem SE B7 hat keine Änderung gebracht. Der Algorithmus führt standardmäßig zu einem Skelett mit einer 8er-Nachbarschaft. Ein Skelett mit einer 4er-Nachbarschaft bedingt einen zusätzlichen Arbeitsschritt. (Quelle: Gonzales08)

Da die Thinning Operation sehr viele Verschiebungen und logische Operationen verwendet, ist es unabdingbar, dass die morphologischen Grundoperationen sehr schnell realisiert sind (s. Kapitel 6.2.1.9).

Um von einem Skelett zu einem logischen Graphen mit Löchern, Verbindungen sowie Knoten- und Endpunkten (s. auch Kapitel 10.3) zu kommen, braucht es meistens noch einiges an Bereinigung. Die störenden kleinen Endstücke können z. B. mit der morphologischen Operation namens *Pruning* (Beschneiden) entfernt werden.

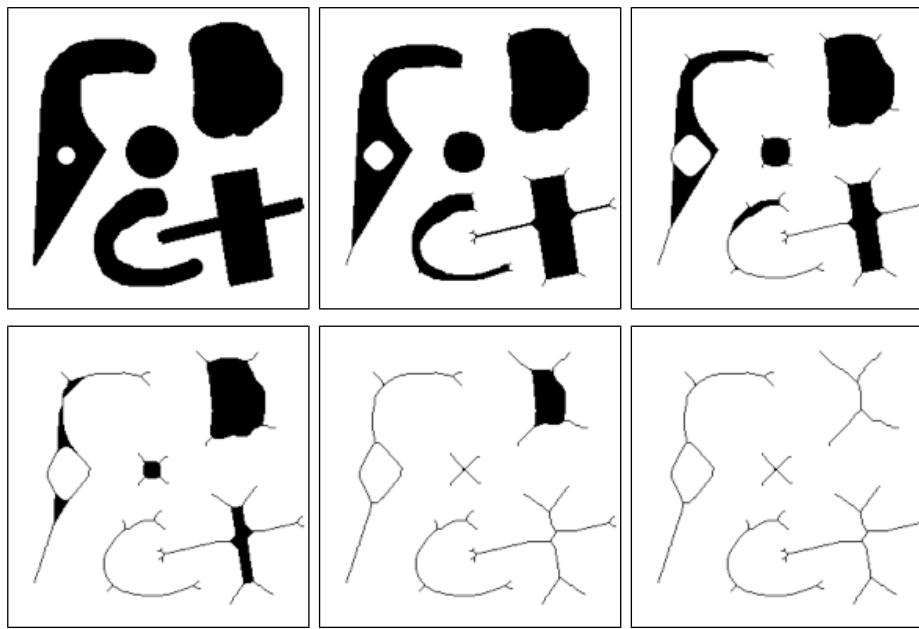


Abb. 253: Oben-Links: Original vor dem Thinning, nach 5, 10, 15, 25 Iteration sowie nach der Konvergenz.



In **ImageJ** ist unter dem Menü *Process > Binary > Skeletonize* der Thinning Algorithmus von [Zhang und Suen](#) integriert. Im selben Menü finden Sie auch die Distanztransformation mit euklidischer Distanzmessung (*Process > Binary > Distance Map*).



In **Matlab** finden Sie die den Thinning Algorithmus in der Funktion *bwmorph* und die Distanztransformation in der Funktion *bwdist* mit verschiedenen Distanzmessarten. Die *Pruning* Operation, um störende Schwänzchen zu entfernen, wird ebenfalls in der *bwmorph* angeboten.

```
I1 = imread('.../Images/X.png'); % reads X image into I1
% skeletonize the region by removing pixels from the border
S1 = bwmorph(I1, 'skel', Inf);

% remove trailing pixels 5 times (spur pixels)
S2 = bwmorph(S1, 'spur', 5);

% remove trailing pixels another 40 times (spur pixels)
S3 = bwmorph(S1, 'spur', 50);

figure, imshow(I1);
figure, imshow(S1);
figure, imshow(S2);
figure, imshow(S3);
```

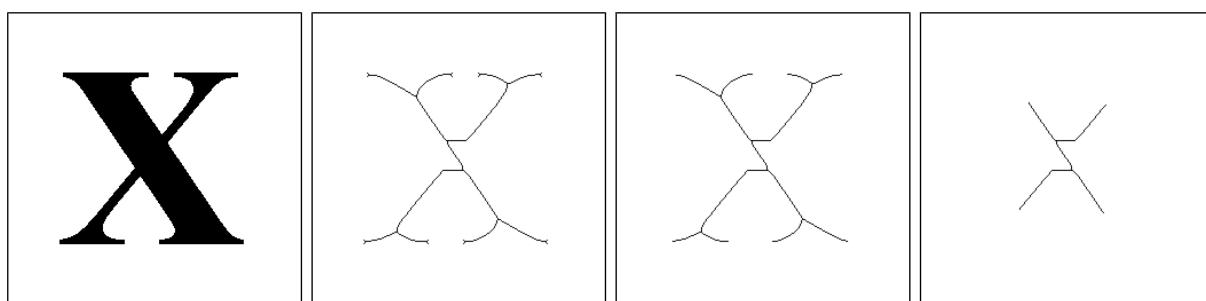
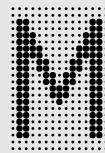


Abb. 254: Skelettierung des Buchstabens X einer Schriftart mit Serifen. V.l.n.r.: Original, nach der Skelettierung, nach 5 Endpixelentfernungen und nach weiteren 50.

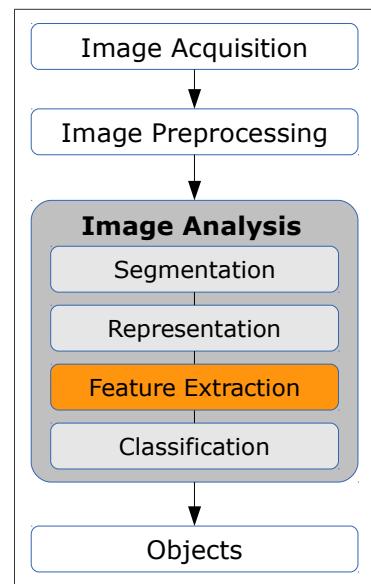
10 Merkmale



Merkmale beziffern möglichst sinnvolle Eigenschaften von Regionen, um damit in der darauf folgenden Klassifikation die Zuweisung zu einer Klasse vornehmen zu können. Beziffern bedeutet hier, dass wir ausschliesslich numerische Attribute von Regionen bestimmen, da der Computer keine andersartigen Merkmale versteht. Eine wichtige Eigenschaft eines guten Merkmals ist zudem, dass es eine möglichst hohe Invarianz gegenüber Translation, Rotation und Skalierung aufweist, damit wir Regionen unabhängig von deren Position, Orientierung und Grösse miteinander vergleichen können.

Wir haben nun alle Regionen in einem Bild gefunden und können sie auf verschiedene Arten repräsentieren. Es bleibt nun die Frage zu lösen, wie wir anhand dieser Repräsentationen z. B. ein bestimmtes Objekt suchen können. Wo im Bild 236 ist den die Schraube? Für den Menschen mit seinem gespeicherten Wissen ist die Frage einfach zu beantworten. Wir haben Bilder von all diesen Werkzeugen gespeichert und wir erkennen diese Objekte unabhängig von der Grösse und der Orientierung. Wir könnten das Problem ähnlich lösen, und in der Tat tut dies die Methode des *Template Matchings*, welche wir im Kapitel 11.3 kennenlernen werden.

Da der Computer viel besser mit Zahlen als mit Bildern umgehen kann, liegt es nahe, gefundene Regionen durch spezifische Merkmale zu beschreiben. Diese Merkmale sollten möglichst invariant sein gegenüber der Position, der Rotation und der Skalierung eines Objekts. Mit diesen Merkmalen sollen Regionen möglichst eindeutig beschrieben werden können, damit wir sie damit unterscheiden können. Oft reicht dazu ein einziges Merkmal nicht aus, weswegen man mehrere Merkmale in einen *Merkmalsvektor (feature vector)* zusammenfasst.



Beispiel von Merkmalen:

- Geometrische Merkmale
 - Breite, Höhe, Fläche, Umfang (Perimeter)
 - Kompaktheit (Kreisförmigkeit)
 - Konvexität, Dichte, Durchmesser
- Statistische Merkmale
 - Histogrammbasierte Merkmale
 - Momente
 - Orientierung
 - Exzentrizität (Länglichkeit)
- Projektionen auf Achsen
- Topologische Merkmale
 - Eulerzahl
- Texturmerkmale
 - Statistische Texturmerkmale
 - Spektrale Texturmerkmale
- Punktmerkmale
 - Harris Corners
 - SIFT-Deskriptoren

10.1 Geometrische Merkmale

10.1.1 Breite und Höhe

Vom *Chain Code* lassen sich einfach die Breite und die Höhe einer konvexen Region (= ohne Einbuchtungen) berechnen. Für den 4er-Chain Code können wir einfach die Anzahl der 1er (oder 3er) Richtungen für die Höhe und die Anzahl der 2er (oder 4er) für die Breite aufsummieren.

$$w_{Ch4} = \sum_{i=0}^N w_i \quad \text{with } \begin{cases} 1 \text{ if } c_i = 0 \\ 0 \text{ else } \end{cases} \quad h_{Ch4} = \sum_{i=0}^N h_i \quad \text{with } \begin{cases} 1 \text{ if } c_i = 1 \\ 0 \text{ else } \end{cases}$$

Analog für den 8er Chain Code:

$$w_{Ch8} = \sum_{i=0}^N w_i \quad \text{with } \begin{cases} 1 \text{ if } c_i = 0, 1, 7 \\ 0 \text{ else } \end{cases} \quad h_{Ch8} = \sum_{i=0}^N h_i \quad \text{with } \begin{cases} 1 \text{ if } c_i = 1, 2, 3 \\ 0 \text{ else } \end{cases}$$

Für konkave Regionen muss für beide Richtungen das Minimum und Maximum berechnet werden. Dies ist allerdings immer noch schneller, als wenn wir durch alle Pixel der Region iterieren müssen.

```
x = 0;
xmax = 0; xmin = 0;
for( int i=0; i< codeLength; i++ )
{
    if( c[i]==0 || c[i]==1 || c[i]==7 ) ++x;
    if( c[i]==3 || c[i]==4 || c[i]==5 ) --x;
    if( x>xmax ) xmax = x;
    if( x<xmin ) xmin = x;
}
width = xmax - xmin;
```

Die Breite und Höhe entsprechen der Grösse der *Bounding Box* um eine Region als minimales, achsparalleles Rechteck.

10.1.2 Perimeter

Die Länge der Kontur in Pixeln wird Perimeter genannt, da der Umfang z.T. mit der Länge der konvexen Hülle gleichgesetzt wird. Beim 4er-Chain Code entspricht der Perimeter der Anzahl Segmente des Chain Codes da alle die Länge 1 haben. Beim 8er-Chain Code haben alle ungeraden Segmente die Länge $\sqrt{2}$.

$$P_{Ch4} = \sum_{i=0}^N n_i \quad P_{Ch8} = \sum_{i=0}^N n_i \quad \text{with } \begin{cases} 1 \text{ if } c_i \bmod 2 = 0 \\ \sqrt{2} \text{ if } c_i \bmod 2 = 1 \end{cases}$$

Bei beiden Berechnungen wird die tatsächliche Anzahl Pixel im Perimeter überschätzt. [Burger06] empfiehlt deshalb eine Korrektur um den Faktor 0.95.

Dazu ist allerdings anzumerken, dass wir in diesen Formeln die Anzahl Pixel im Perimeter aufsummieren und nicht die effektive Länge an der Aussenseite der Randpixel.

Diese lässt sich nur mit dem 4er-Crack Code exakt berechnen:

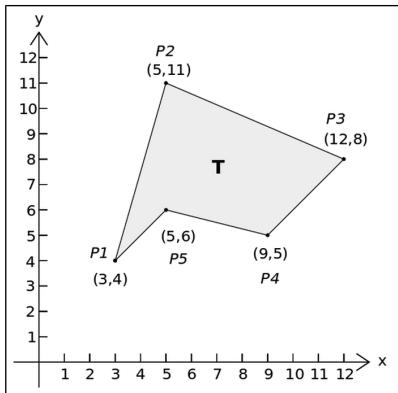
$$P_{Cr4} = \sum_{i=0}^N n_i \quad P_{Cr8} = \sum_{i=0}^N n_i \quad \text{with } \begin{cases} 1 \text{ if } c_i \bmod 2 = 0 \\ \sqrt{2}/2 \text{ if } c_i \bmod 2 = 1 \end{cases}$$

10.1.3 Fläche

Die Fläche einer Region R entspricht der Anzahl ihrer Pixel.

$$\text{area} = N = |R|$$

Wenn wir die Koordinaten der M Konturpixel einer Region ohne Löcher haben, so kann die Fläche über die [Gaußsche Trapezformel](#) angenähert werden. Dabei entsteht eine negative Fläche, wenn wir im Uhrzeigersinn und eine positive Fläche, wenn wir im Gegenuhrzeigersinn durchiterieren. Indizes grösser n müssen jeweils modulo n genommen werden ($n+1=1$):



$$A = \frac{1}{2} \left| \sum_{i=1}^{n-1} (y_i + y_{i+1})(x_i - x_{i+1}) \right|$$

$$A = \frac{1}{2} |(4+11)(3-5)+(11+8)(5-12)+(8+5)(12-9)+(5+6)(9-5)+(6+4)(5-3)|$$

$$A = \frac{1}{2} |15 \cdot -2 + 19 \cdot -7 + 13 \cdot 3 + 11 \cdot 4 + 10 \cdot 2| = \frac{1}{2} |-30 - 133 + 39 + 44 + 20| = \frac{1}{2} |-60| = 30$$

Abb. 255: Berechnung der Polygonfläche als Summe der Trapezflächen unterhalb der Teilstrecken.

Wenn wir den Chain Code einer Kontur haben, so lässt sie die Fläche mit folgendem Algorithmus berechnen (für den 8er-Nachbarschafts Chain Code):

```
double area = 0.0;
int y=0; int x=0;
for( int i=0; i<codeLength; i++ )
{
    switch( c[i] )
    {
        case 0: area -= y;           x++;      break;
        case 1: area -= (y+0.5);   x++; y--; break;
        case 2:                  y--;      break;
        case 3: area += (y+0.5);   x--; y--; break;
        case 4: area += y;          x--;      break;
        case 5: area += (y-0.5);   x--; y++; break;
        case 6:                  y++;      break;
        case 7: area -= (y-0.5);   x++; y++; break;
    }
}
```

10.1.4 Kompaktheit oder Kreisförmigkeit

Als Kompaktheit einer Region kann das Verhältnis von der Fläche zum Perimeter verwendet werden. Die kompakteste Region ist demnach eine Kreisfläche, weshalb die Kompaktheit auch Kreisförmigkeit ([circularity](#)) genannt wird. Da der Perimeter linear mit der Vergrösserung zunimmt, die Fläche aber quadratisch, dividiert man ebenfalls durch den Perimeter im Quadrat. Für eine kreisförmige Region ergibt sich so eine Kreisförmigkeit von $1/4\pi$. Mit einer Skalierung auf den Kreis erhalten wir folgende Formel für die Kreisförmigkeit zwischen 0-1:

$$\text{circularity}(R) = 4\pi \cdot \frac{\text{area}}{\text{perimeter}^2(R)}$$

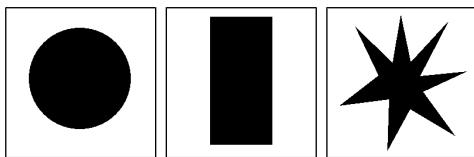


Abb. 256: Kreisförmigkeit v.l.n.r.: 1.0, 0.6, 0.12

Invarianz der Merkmale

Die Merkmale Breite, Höhe, Perimeter und Fläche sind zwar (abgesehen von Quantisierungsfehlern) unabhängig von der Translation und Rotation einer Region, sie verändern sich jedoch bei einer Skalierung. Wie am Beispiel der Kreisförmigkeit ersichtlich, können Merkmale durch Kombination mit anderen Merkmalen auch invariant gegenüber der Skalierung gemacht werden.

10.1.5 Konvexität, Dichte und Durchmesser

Als konvexe Hülle H bezeichnet man das kleinste Polygon um eine Region. Ein schneller Algorithmus, um diese zu berechnen, ist der [QuickHull-Algorithmus](#):

1. Finde Punkt mit max. & min. x-Koordinate.
2. Teile alle Punkte in die Gruppe links und rechts von der Linie. Beide Gruppen werden rekursiv weiterverarbeitet.
3. Finde den Punkt mit der grössten Distanz zur Linie.
4. Punkte innerhalb des Dreiecks können ignoriert werden, denn sie sind nicht Teil der konvexen Hülle.
5. Repetiere Schritt 3 & 4 bis keine Punkte mehr vorhanden sind.

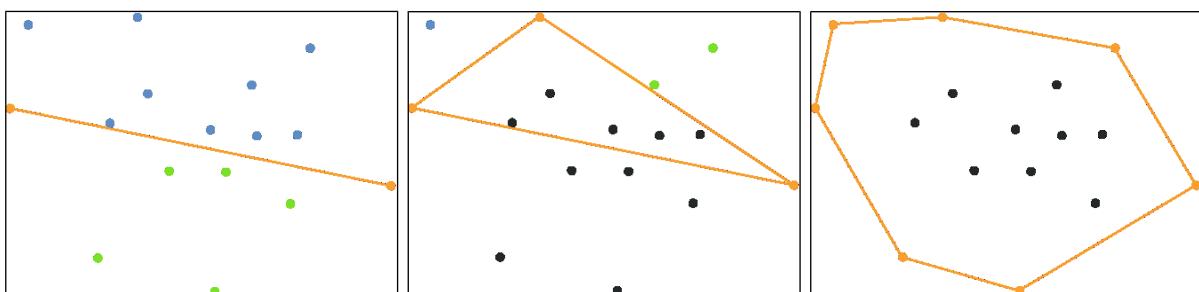


Abb. 257: Links: Schritt 1 & 2, Mitte: Schritt 3 & 4, Rechts: Konvexe Hülle

Von der konvexen Hülle können wir folgende Merkmale ableiten:

Die **Konvexität** der Region R ist definiert als Verhältnis des Perimeters (= Umfang) der konvexen Hülle zum Perimeter der Region:

$$\text{convexity}(R) = \frac{\text{perimeter}(H)}{\text{perimeter}(R)}$$

Die **Dichte** oder **Solidität** der Region R ist definiert als Verhältnis der Fläche der Region zur Fläche der konvexen Hülle:

$$\text{density}(R) = \frac{\text{area}(R)}{\text{area}(H)}$$

Der **Durchmesser** der Region R ist die maximale Distanz zwischen zwei Punkten der konvexen Hülle. Dieser Durchmesser wird manchmal auch *Feret Durchmesser* genannt.

$$\text{diameter}(R) = \max(\text{diameter}(H))$$

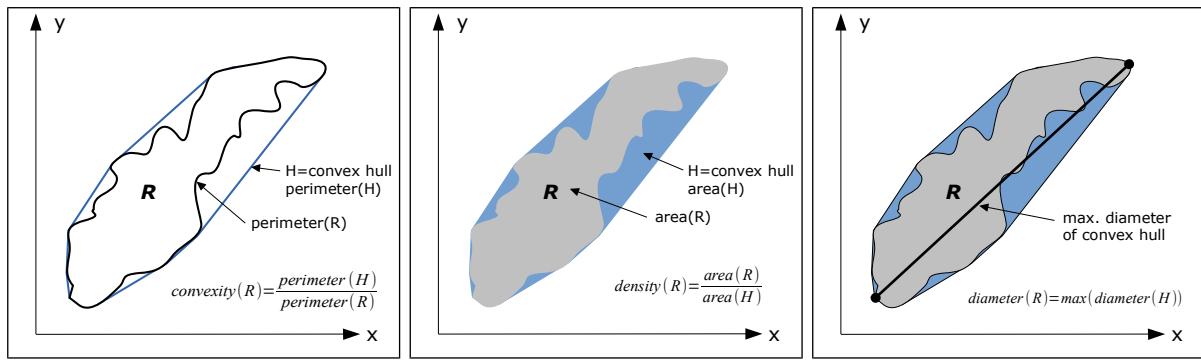


Abb. 258: Konvexität, Dichte und Durchmesser

10.2 Statistische Merkmale

10.2.1 Histogrammbasierte Merkmale

Alle statistischen Kennzahlen eines Bildes, die wir im Kapitel 4.2 kennengelernt haben, können wir natürlich auch als Merkmale für eine Region verwenden. Insbesondere eignen sich Kontrast, Mittelwert und Standardabweichung gut als Merkmale, um Regionen zu beschreiben.

Eine Histogrammauswertung kann genauso gut auf den Kanälen eines bestimmten Farbmodells (s. Kapitel 2.2) gemacht werden. Deshalb zählen *Merkmale aus Farben* ebenfalls zu den histogrammbasierten Merkmalen. Insbesondere eignen sich alle Farbmodelle gut, die den Farbton in einem Kanal konzentrieren, da der Farbton damit ja invariant von der Helligkeit und der Sättigung wird.

10.2.2 Momentbasierte Merkmale

Aus der statistischen Positionsverteilung der Pixel einer Region können wir wichtige Merkmale wie die Orientierung und die Exzentrizität herleiten. Wir haben bis jetzt statistische Kennzahlen ausschliesslich aus den Pixelwerten abgeleitet (s. Kapitel 4). Für die Merkmale in diesem Kapitel verwenden wir ausschliesslich die Koordinaten der Pixel einer Region.

Das bekannteste, auf Koordinaten basierende Merkmal ist der Schwerpunkt als Mittelwert aller Pixelkoordinaten einer Region \$R\$:

$$s = (\bar{x}, \bar{y}) \text{ mit } \bar{x} = \frac{1}{|R|} \sum_{(x,y) \in R} x \text{ und } \bar{y} = \frac{1}{|R|} \sum_{(x,y) \in R} y$$

Der Schwerpunkt ist nur ein Beispiel von statistischen Kennzahlen, die in der Statistik als *Momente* bekannt sind. Wir können [Momente in der Bildverarbeitung](#) gut verwenden, um für binäre Regionen Merkmale wie Fläche, Schwerpunkt und Ausrichtung zu bestimmen.

Ein Moment der Ordnung \$p,q\$ für eine Region \$R\$ ist wie folgt definiert:

$$m_{pq} = \sum_{x \in R} \sum_{y \in R} g(x, y) x^p y^q \text{ mit } p, q = 0, 1, 2, \dots$$

\$g(x,y)\$ steht darin für den Grauwert, der, wenn wir ihn nicht ignorieren, als Dichte interpretiert werden kann. Für binäre Regionen kann er weggelassen werden, sodass sich die Definition eines Moments einer binären Region vereinfacht zu:

$$m_{pq} = \sum_{x \in R} \sum_{y \in R} x^p y^q \text{ mit } p, q = 0, 1, 2, \dots$$

Die **Fläche** einer Region \$R\$ kann damit auch als Moment der 0. Ordnung (\$p+q=0\$) formuliert werden:

$$\text{area}(R) = |R| = m_{00}(R) = \sum_{(x,y) \in R} 1 = \sum_{(x,y) \in R} x^0 y^0$$

Die Fläche der Region wird damit nur korrekt berechnet, wenn wir die Summe über alle Pixel der Region bilden. Ist nur eine Region im Bild, so können wir auch die Summe über alle Pixel des Bildes I erstellen. Wir brauchen dann allerdings die Pixelwerte als Gewicht:

$$\text{area}(R) = |R| = m_{00}(R) = \sum_{(x,y) \in I} g(x,y) x^0 y^0$$

Der **Schwerpunkt** einer Region R kann berechnet werden, indem man die Momente 1. Ordnung ($p+q=1$) durch die Fläche dividiert. Statistisch interpretiert entspricht der Schwerpunkt dem Erwartungswert:

$$\bar{x} = \frac{1}{|R|} \cdot \sum_{(x,y) \in R} x^1 y^0 = \frac{m_{10}(R)}{m_{00}(R)} \quad \text{und} \quad \bar{y} = \frac{1}{|R|} \cdot \sum_{(x,y) \in R} x^0 y^1 = \frac{m_{01}(R)}{m_{00}(R)}$$

10.2.2.1 Zentrale Momente (translationsinvariant)

Um ein Moment einer Region R invariant gegenüber Translation zu machen, verschiebt man die Region in den Schwerpunkt und erhält so die zentralen Momente der Ordnung p,q :

$$\mu_{pq} = \sum_{x \in R} \sum_{y \in R} (x - \bar{x})^p (y - \bar{y})^q \quad \text{mit } p, q = 0, 1, 2, \dots$$

Die zentralen Momente bis zum 3. Grad sind:

$$\mu_{00} = M_{00}$$

$$\mu_{01} = 0$$

$$\mu_{10} = 0$$

$$\mu_{11} = M_{11} - \bar{x} M_{01} = M_{11} - \bar{y} M_{10}$$

$$\mu_{20} = M_{20} - \bar{x} M_{10}$$

$$\mu_{02} = M_{02} - \bar{y} M_{01}$$

$$\mu_{21} = M_{21} - 2\bar{x} M_{11} - \bar{y} M_{20} + 2\bar{x}^2 M_{01}$$

$$\mu_{12} = M_{12} - 2\bar{y} M_{11} - \bar{x} M_{02} + 2\bar{y}^2 M_{10}$$

$$\mu_{30} = M_{30} - 3\bar{x} M_{20} + 2\bar{x}^2 M_{10}$$

$$\mu_{03} = M_{03} - 3\bar{y} M_{02} + 2\bar{y}^2 M_{01}$$

Informationen über die Ausrichtung eines Objektes können wir gewinnen, wenn wir die zentralen Momente der 2. Ordnung ($p+q=2$: $\mu_{11}, \mu_{20}, \mu_{02}$) auswerten. Statistisch interpretiert entsprechen die Momente 2. Grades der Varianz.

Daraus können wir die *Orientierung* der Region, deren *Exzentrizität* sowie die Längen der beiden *Hauptachsen* berechnen. Die Mathematik dahinter entstammt der Hauptkomponentenanalyse, die im Kapitel 7.4 kennengelernt haben. Darin wird ja die Kovarianzmatrix für die Region R aus den Varianzen einer Datenverteilung aufgebaut:

$$\text{Cov}(R) = \begin{bmatrix} \mu_{20}, \mu_{11} \\ \mu_{11}, \mu_{02} \end{bmatrix}$$

Die beiden Eigenwerte der Kovarianzmatrix sind proportional zu den quadrierten Längen der Eigenvektoren und sind wie folgt definiert:

$$\lambda_i = \frac{\mu_{20} + \mu_{02}}{2} \pm \frac{\sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}}{2}$$

Die **Orientierung** entspricht dem Winkel θ von der x-Achse zur Hauptachse der Region.

Die Eigenvektoren der Kovarianzmatrix entsprechen den grossen und kleinen Halbachsen der Region. Somit kann die Ausrichtung des Bildes aus dem Winkel des Eigenvektors mit dem grössten Eigenwert bestimmt werden. Die Hauptachse einer Region ist die Achse, um die eine Drehung das geringste Trägheitsmoment entwickelt. Sie entspricht ebenfalls der langen Achse des umgebenden Rechtecks um die Region. Dieses umgebende Rechteck muss nicht unbedingt das minimalste umgebende Rechteck sein.

$$\text{Orientierung: } \theta = -\frac{1}{2} \tan^{-1} \left(\frac{2 \cdot \mu_{11}}{\mu_{20} - \mu_{02}} \right)$$

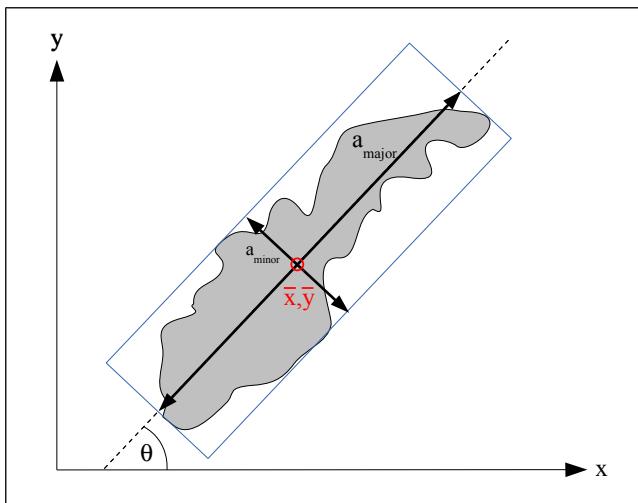


Abb. 259: Orientierung einer Region mit den beiden Hauptachsen und dem minimalsten Rechteck.

Die **Exzentrizität** entspricht der Länglichkeit einer Region und liegt im Wertebereich von 0-1. Eine runde Region hat eine Länglichkeit von 0 und eine lange Region eine Länglichkeit von 1.

$$\text{Exzentrizität: } \epsilon = \sqrt{1 - \frac{\lambda_2}{\lambda_1}}$$

Die **Längen der beiden Hauptachsen** (*major axis, minor axis*) sind wie folgt definiert:

$$a_{\text{major}} = 2 \sqrt{2(\varphi_1 + \sqrt{\varphi_2})}$$

$$a_{\text{minor}} = 2 \sqrt{2(\varphi_1 - \sqrt{\varphi_2})}$$

$$\text{mit } \varphi_1 = \mu_{20} + \mu_0 \text{ und } \varphi_2 = (\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2$$

10.2.2.2 Normalisierte zentrale Momente (skalierungsinvariant)

Meistens möchte man Merkmale invariant von der Grösse einer Region haben. Skaliert man eine Region um den Faktor s , so verändern sich die zentralen Momente um den Faktor $s^{(p+q+2)}$. Mit einer zusätzlichen Division durch die Grösse der Region erhalten wir die normalisierten zentralen Momente der Ordnung $p,q=2,3,\dots,n$:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(1+\frac{p+q}{2}\right)}} \text{ mit } p, q = 2, 3, 4, \dots, n$$

10.2.2.3 Invariante Momente (rotationsinvariant)

Zu guter Letzt gibt es noch die zusätzlich rotationsinvarianten Momente, die damit komplett invariant sind gegenüber Translation, Skalierung und Rotation. Diese Momente werden z.T. auch nach deren Entwickler [Ming Kuei Hu](#) als [Hu's 7 Momente](#) bezeichnet.

Die sieben invarianten Momente nach Hu sind wie folgt definiert:

$$\varphi_1 = \eta_{20} - \eta_{02}$$

$$\varphi_2 = (\eta_{20} - \eta_{02})^2 + 4 \cdot \eta_{11}^2$$

$$\varphi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} + \eta_{03})^2$$

$$\varphi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$\varphi_5 = (\eta_{30} - 3\eta_{12}) \cdot (\eta_{30} + \eta_{12}) \cdot [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03}) \cdot (\eta_{21} + \eta_{03}) \cdot [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$\varphi_6 = (\eta_{20} - \eta_{02}) \cdot [(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11} \cdot (\eta_{30} + \eta_{12}) \cdot (\eta_{21} + \eta_{03})$$

$$\varphi_7 = (3\eta_{21} - \eta_{03}) \cdot (\eta_{30} + \eta_{12}) \cdot [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{12} - \eta_{30}) \cdot (\eta_{21} + \eta_{03}) \cdot [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Diese Momente werden in der Praxis meist logarithmiert, um den grossen Wertebereich zu reduzieren. Alle Momente ausser dem 7. sind auch invariant gegenüber Spiegelung. Das 7. Moment wird negativ, ist aber im Absolutwert invariant.



Vergleich der invarianten Momente mit verschiedenen Varianten zweier Bilder

Im nachfolgenden Matlab-Beispiel werden jeweils 8 Versionen von [Lena](#) und vom [Cameraman](#) Bild erzeugt. Von allen Versionen berechnen wir mit der Funktion [invvmoments.m](#) (Quelle: [\[Gonzales08\]](#)) den Vektor der 7 invarianten Momente von Hu. Wir berechnen die Momente von einem ganzen Graustufenbild und nicht nur von einer binären Region. Wir logarithmieren die Werte und nehmen davon den Absolutwert. Beim Logarithmieren kann übrigens ein ungültiger Wert entstehen, falls das Moment nahe oder gleich null ist.

```
I1      = imread('..../Images/Lena128.png');
I1_half = imresize(I1, 0.5);
I1_rot180 = imrotate(I1, 180);
I1_rot90 = imrotate(I1, 90);
I1_rot03 = imrotate(I1, 3);
I1_mirror = flipdim(I1, 1);
I1_noise = imnoise(I1, 'gaussian', 0, 0.01);
I1_affine = imtransform(I1, maketform('affine',[1 0 0; .5 1 0; 0 0 1]));

figure(1), imshow(I1), title('Original Image');
figure(2), imshow(I1_half), title('Halfsize Image');
figure(3), imshow(I1_rot180), title('180° rotated Image');
figure(4), imshow(I1_rot90), title('90° rotated Image');
figure(5), imshow(I1_rot03), title('3° rotated Image');
figure(6), imshow(I1_mirror), title('Mirrored Image');
figure(7), imshow(I1_noise), title('Noisy Image');
figure(8), imshow(I1_affine), title('Affine transformed Image');

im1      = abs(log10(invvmoments(I1)));
im1_half = abs(log10(invvmoments(I1_half)));
im1_rot180 = abs(log10(invvmoments(I1_rot180)));
im1_rot90 = abs(log10(invvmoments(I1_rot90)));
im1_rot03 = abs(log10(invvmoments(I1_rot03)));
im1_mirror = abs(log10(invvmoments(I1_mirror)));
im1_noise = abs(log10(invvmoments(I1_noise)));
im1_affine = abs(log10(invvmoments(I1_affine))');
```

Zusätzlich zu den skalierten und rotierten Versionen testen wir auch eine gespiegelte, eine verrauschte und eine affin transformierte Version der Bilder.



Abb. 260: 8 Versionen vom Lena-Bild sowie das Cameraman-Bild, von dem dieselben Transformationen erzeugt wurden. Beide Originalbilder haben die gleiche Auflösung von 128 x 128 Pixel.

In der nachfolgenden Tabelle sehen wir die Resultate aller invarianten Momente von allen Bildversionen. Die blauen Werte in der Spalte rechts von den Momenten entsprechen den prozentualen Abweichungen vom Medianwert aller Bildversionen. Wir können folgende Schlüsse aus den Messungen ziehen:

- Hu's Momente sind erstaunlich invariant. Mit Ausnahme des affinen Spielverderbers liegen alle Werte eines Moments innerhalb 0.001% vom Medianwert entfernt.
- Hu's Momente sind nicht invariant gegenüber affinen Transformationen.
- Alle 7 Momente unterscheiden sich zwischen den Bildern, aber gleich ausgeprägt. Die Momente 1 und 2 differieren nur um 1.7% während die Momente 3 und 5 sich um 13% unterscheiden.

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	
Lena128.png	2.8709	0.000	8.1508	0.000	11.9001	0.000	10.9209	0.000
halbiert	2.8710	0.000	8.1508	0.000	11.8997	0.000	10.9202	0.000
rotiert um 180°	2.8709	0.000	8.1508	0.000	11.9001	0.000	10.9209	0.000
rotiert um 90°	2.8709	0.000	8.1508	0.000	11.9001	0.000	10.9209	0.000
rotiert um 3°	2.8709	0.000	8.1450	0.001	11.8912	0.001	10.9236	0.000
gespiegelt	2.8709	0.000	8.1508	0.000	11.9001	0.000	10.9209	0.000
verrauscht	2.8715	0.000	8.1641	0.002	11.8456	0.005	10.9390	0.002
affine transformiert	2.8081	0.022	6.2093	0.313	11.1650	0.066	11.2343	0.028
Median	2.8709	8.1508	11.8999	10.9209	23.6598	15.0916	22.3320	
Cameraman128.png	2.8209	0.000	8.0104	0.000	10.3546	0.000	10.1459	0.000
halbiert	2.8210	0.000	8.0106	0.000	10.3541	0.000	10.1464	0.000
rotiert um 180°	2.8209	0.000	8.0104	0.000	10.3546	0.000	10.1459	0.000
rotiert um 90°	2.8209	0.000	8.0104	0.000	10.3546	0.000	10.1459	0.000
rotiert um 3°	2.8210	0.000	8.0077	0.000	10.3537	0.000	10.1470	0.000
gespiegelt	2.8209	0.000	8.0104	0.000	10.3546	0.000	10.1459	0.000
verrauscht	2.8267	0.002	8.0780	0.008	10.3915	0.004	10.1914	0.004
affine transformiert	2.7578	0.023	6.1056	0.312	9.8251	0.054	10.3051	0.015
Median	2.8209	8.0104	10.3546	10.1462	20.4444	14.2309	21.3569	
Differenz Median in %	1.7%		1.7%	13.0%	7.1%	13.6%	5.7%	4.4%

Abb. 261: Hu's 7 invarianten Momente von den Versionen des Lena- und des Cameraman-Bildes.

Affine-transformations-invariante Momente

Die Herren Flusser und Suk [Flusser93] haben 1993 4 Momente vorgestellt, die zusätzlich invariant sind gegenüber affinen Transformationen. 2003 haben sie 7 Momente präsentiert [Flusser03], die zu sogar unempfindlich sind gegenüber verunschärften Bilderversionen.

10.3 Topologische Merkmale

Die *Topologie* ist ein fundamentales Teilgebiet der Mathematik und beschäftigt sich mit den Eigenschaften geometrischer Strukturen, die unter stetigen Verformungen erhalten

bleiben, wobei der Begriff der Stetigkeit durch die Topologie in sehr allgemeiner Form definiert wird. Man definiert dadurch sogenannte *homöomorphe Klassen* mit gleicher Topologie. Anhand dieser Klassen können 2D- aber auch 3D-Objekte kategorisiert werden.

10.3.1 Anzahl Löcher

Ein einfaches topologisches Merkmal ist die Anzahl Löcher einer Region. Das Alphabet der grossen Buchstaben können wir anhand der Löcher in 3 Kategorien unterteilen:

- 2 Löcher: **B**
- 1 Loch: **A,R,D,O,P,Q**
- 0 Löcher: **C,E,F,G,H,I,J,K,L,M,N,S,T,U,V,W,X,Y,Z**

Fortgeschrittene Algorithmen zur Bestimmung der Konturen liefern auch die inneren Konturen von Löchern zurück (s. Kapitel 9.5). Fehlen die inneren Konturen, so kann die Anzahl Löcher mit der Euler-Charakteristik bestimmt werden.

Euler-Charakteristik

Der Schweizer Mathematiker Leonhard Euler, der als Begründer der Topologie gilt, hat mit der [Euler Charakteristik](#) eine Kennzahl für topologisch gleichartige Geometrien entwickelt. Sie ist definiert als die Anzahl Ecken e minus die Anzahl Kanten k plus die Anzahl Flächen f :

$$E = e - k + f$$

Euler hat bewiesen, dass für bestimmte Topologien die Euler-Charakteristik konstant ist:

- Konvexe Polyeder: $E = e - k + f = 2$
- Planare Polygonnetze: $E = e - k + f = 1$ (ohne Löcher)

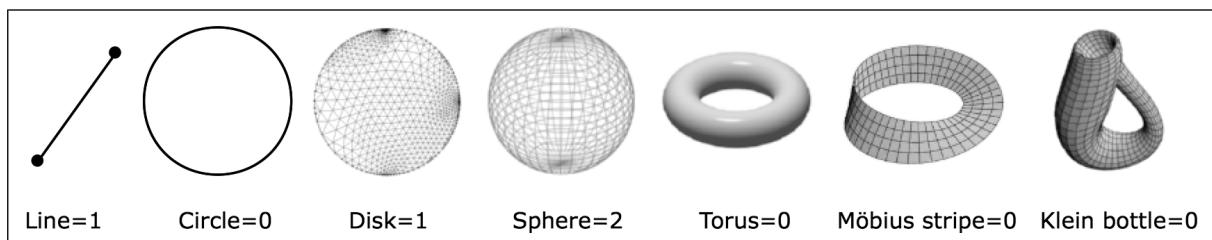


Abb. 262: Topologisch verschiedene Geometrien und ihre Euler-Charakteristik.

Wir können die konstante Euler-Charakteristik für planare Polygonnetze auf Pixelregionen anwenden, indem wir die Pixel als Flächen, die Pixelkanten als Kanten und die Pixelecken als Ecken interpretieren. Kanten und Ecken werden jeweils nur einmal gezählt. Ergibt die Euler-Charakteristik nicht eins, so muss das Polygonnetz Löcher enthalten. Die Anzahl Löcher L ist dann gleich 1 minus die Euler-Charakteristik E :

$$L = 1 - E$$

$e = 24$ $k = 38$ $f = 15$ $E = 1$ $L = 0$	$e = 24$ $k = 38$ $f = 14$ $E = 0$ $L = 1$	$e = 24$ $k = 38$ $f = 13$ $E = -1$ $L = 2$	$e = 24$ $k = 36$ $f = 12$ $E = 0$ $L = 1$	$e = 23$ $k = 34$ $f = 11$ $E = 0$ $L = 1$	$e = 22$ $k = 31$ $f = 10$ $E = 1$ $L = 0$

Abb. 263: Bestimmung der Anzahl Löcher L durch Zählen den Ecken e , Kanten k und Pixel f bei verschiedenen Regionen. Mit der Euler-Charakteristik erhalten wir $L = 1-E = 1-e-k+f$. Am zweiten Beispiel erkennen wir, dass die Euler'sche Charakteristik die 4er-Nachbarschaft eines Pixels für die Zusammengehörigkeit verwendet.

10.4 Texturmerkmale

Der Begriff der *Textur* ist je nach Anwendungsgebiet unterschiedlich definiert. Im allgemeinen Sprachumgang bezeichnet der Begriff meistens eine Oberflächenbeschaffenheit, die man manuell ertasten kann. In der 3D-Computergrafik ist eine Textur ein Rasterbild, das man auf Dreiecksnetze projizieren kann (*Texture Mapping*).

In der Bildverarbeitung verwendet man den Begriff der Textur für eine *bestimmte Musterung* einer Region. Es gibt keine allgemeine und klare Definition für so eine Musterung, weil es kein numerisches Mass gibt für Adjektive, wie fein, grob, weich, körnig oder verkratzt. Eine Textur zeichnet sich meistens durch eine mehr oder weniger starke Regelmässigkeit aus, die auch ein grosses Mass an Zufälligkeit enthalten kann. Eine zusätzliche Schwierigkeit ist die starke Massstabsabhängigkeit.

Dieser Mangel an klarer Definition macht die texturbasierte Segmentation zu einem der schwierigsten Gebiete in der Bildverarbeitung. Um ein Bild überhaupt in Regionen unterteilen zu können, muss man diese Texturmerkmale für jedes Pixel in einer relativ grossen lokalen Umgebung bestimmen. Das Thema wird zudem in den verschiedenen Standardwerken ([*Gonzale08*], [*Jähne05*], [*Nischwitz11*]) sehr unterschiedlich behandelt.

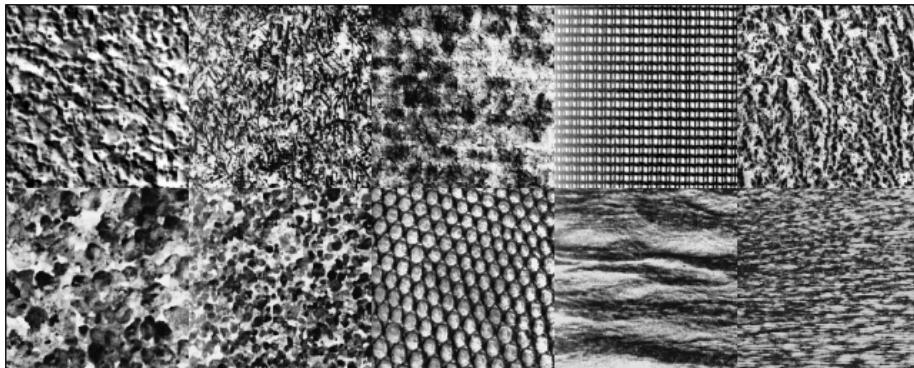


Abb. 264: 10 Quadrate mit Texturen, die z.T. sogar von Auge schwierig zu unterscheiden sind.

Man unterscheidet drei Methoden, um Texturmerkmale zu bestimmen:

- Statistische Texturmerkmale
- Strukturelle Texturmerkmale
- Spektrale Texturmerkmale

10.4.1 Statistische Texturmerkmale

Die einfachsten Texturmerkmale basieren auf statistischen Kennzahlen aus der Umgebung eines Pixels. Wir können dazu einige Masse, die wir im Kapitel 4.2 kennengelernt haben verwenden. Am ehesten eignen sich die Masse **Varianz**, **Schiefe**, **Wölbung** und **Entropie**.

Der Nachteil dieser auf dem Histogramm der lokalen Umgebung basierenden Kennzahlen ist, dass sie keinerlei Struktur oder Grauwertnachbarschaften berücksichtigen. Sie sind invariant gegenüber Pixelpermutationen. So liefert ein schwarzweiss-gestreiftes Muster dieselben Masse wie ein schwarz-weisses Schachbrett-Muster.

Auf der anderen Seite haben diese Masse den Vorteil, dass sie invariant gegenüber der Orientierung sind.

10.4.1.1 Grauwertmatrix (Co-occurrence Matrix)

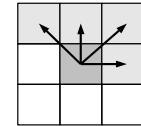
Die *Grauwertmatrix* oder *Co-occurrence Matrix* (meist abgekürzt mit *GLCM*) gibt in ihren Zellen die Häufigkeit einer bestimmten Grauwertkombination von zwei Pixeln in einem Bild I an. Die beiden Pixel liegen durch eine bestimmte fixe Distanz ($\Delta x, \Delta y$) auseinander.

$$C_{\Delta x, \Delta y}(i, j) = \sum_{x=1}^n \sum_{y=1}^m \begin{cases} 1, & \text{if } I(x, y) = i \text{ and } I(x+\Delta x, y+\Delta y) = j \\ 0 & \text{otherwise} \end{cases}$$

Für die Erstellung gehen wir also durch alle Pixel eines $n \times m$ grossen Bildes I und vergleichen z. B. das aktuelle Pixel $I(x, y)$ mit seinem rechten Nachbarn ($\Delta x=1, \Delta y=0$). Wenn der aktuelle Wert z. B. 1 ist und der rechte Nachbar den Wert 2 hat, dann erhöhen wir in der Grauwertmatrix die Zelle $C(1,2)$ um 1. Im nachfolgenden Beispiel tritt dieser Fall 4-mal auf:

$$I = \begin{bmatrix} 0 & 1 & 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ 0 & 1 & 2 & 2 & 3 & 3 \\ 0 & 2 & 2 & 3 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix} \quad \begin{array}{|c|c|c|c|c|c|} \hline & & & & & \\ \hline \end{array} \quad C_{1,0} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 5 & 2 & 0 & 0 \\ 1 & 4 & 6 & 0 \\ 0 & 0 & 4 & 5 \end{bmatrix}$$

Für die direkte Nachbarschaft mit dem Offset von einem Pixel in beide Richtungen gibt es demnach vier mögliche Richtungskombinationen $0^\circ, 45^\circ, 90^\circ, 135^\circ$. Oftmals, wie z. B. für die Haralick'schen Texturmasse im nächsten Kapitel, werden vier Grauwertmatrizen für diese Richtungen berechnet.



Für ein 8-Bit-Graustufenbild ergibt sich eine 256×256 Pixel grosse Grauwertmatrix. Damit deren Erzeugung und die weitere Verarbeitung (im nächsten Kapitel) nicht zu aufwendig wird, reduziert man die Anzahl Graustufen meist auf z. B. 8, 16 oder 32 Stufen.

Für eine *normalisierte Grauwertmatrix* werden die Summen noch durch die Gesamt-pixelzahl dividiert, womit wir in den Zellen die Wahrscheinlichkeit erhalten für die entsprechende Grauwertkombination zweier Pixel in einer bestimmten Richtung. Die Grauwertmatrix ist somit ein 2D-Histogramm, ähnlich wie wir es im Kapitel 4.1.3 kennengelernt haben.



In **Matlab** kann die Grauwertmatrix mit dem Befehl [graycomatrix](#) erzeugt.

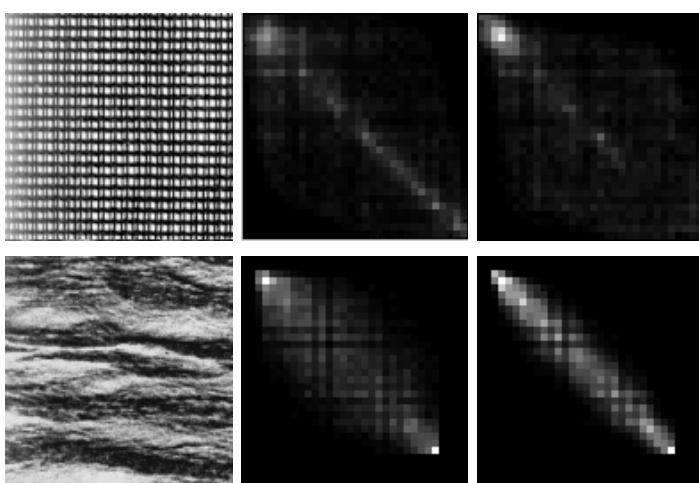


Abb. 265: 3 Texturbilder und ihre Grauwertmatrizen (32×32) mit $\Delta x=1, \Delta y=0$ und $\Delta x=0, \Delta y=1$

10.4.1.1.1 Haralick'sche Texturmasse

Robert Haralick [Haralick79] entwickelte 1979 aus der Grauwertmatrix die nach ihm benannten Haralick'schen Texturmasse. Für eine Grauwertmatrix $C_{\Delta,\alpha}$ mit Delta Δ , Winkel α und der Matrixgrösse K sind die wichtigsten Masse wie folgt definiert:

- **Energie** (manchmal auch Uniformität genannt): Gibt die Summe der quadrierten Elemente (Indizes i und j) zurück. Bereich: $0..K^2$
- **Kontrast**: Gibt den Kontrast von einem Pixel zum Nachbar über die ganze Region zurück. Bereich: $0..1$, der Kontrast einer homogenen Region ist 0.
- **Homogenität**: Gibt ein Mass für die Nähe zur Hauptdiagonalen zurück. Bereich: $0..1$, die Homogenität einer homogenen Region ist 1.
- **Entropie**: Gibt ein Mass für die Zufälligkeit der Pixel in der Region zurück. Die Zufälligkeit ist 0, wenn alle Elemente von C 0 sind und maximal, wenn alle Elemente gleich sind. Bereich: $0..2 \cdot \log_2(K)$
- **Korrelation**: Gibt ein Mass zurück wie korreliert ein Pixel zu seinem Nachbarn ist. Bereich: $-1..1$, die max. positive Korrelation ist 1 und die max. negative Korrelation ist -1. Keine Korrelation ist 0. Die Werte $\mu_i, \mu_j, \sigma_i, \sigma_j$ sind die horizontalen vertikalen Mittelwerte und Standardabweichungen aus der Grauwertmatrix:

$$\begin{aligned}\mu_i &= \frac{1}{K} \sum_{i=1}^K i \sum_{j=1}^K C_{ij} & \mu_j &= \frac{1}{K} \sum_{j=1}^K j \sum_{i=1}^K C_{ij} \\ \sigma_i^2 &= \sum_{i=1}^K (i - \mu_i)^2 \sum_{j=1}^K C_{ij} & \sigma_j^2 &= \sum_{j=1}^K (j - \mu_j)^2 \sum_{i=1}^K C_{ij}\end{aligned}$$

Diese Masse können in einem Merkmalsvektor zusammengefasst und für die Segmentation verwendet werden.



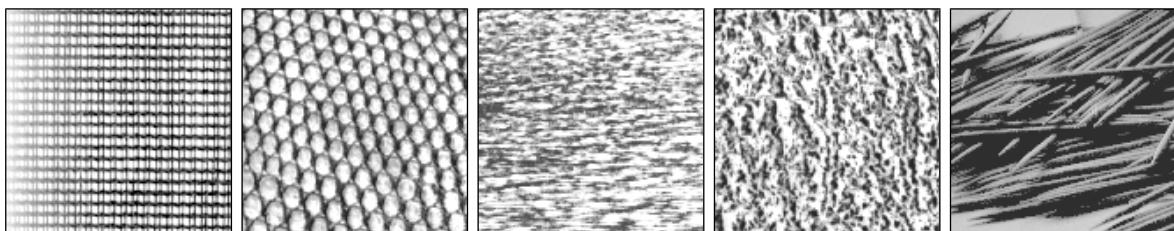
In **Matlab** bekommen sie die Haralick'schen Texturmasse für Energie, Kontrast, Homogenität und Korrelation mit der Funktion [graycoprops](#).



Für **ImageJ** gibt es ein [Texture Analyzer](#) Plugin von Julio Cabrera, das die Haralick'schen Texturmasse für eine rechteckige Region (ROI) berechnen kann.

10.4.2 Spektrale Texturmerkmale

Mit der Fourier-Analyse (Kapitel 7.1) können wir die spektrale Zusammensetzung einer Region bestimmen. Aus dem Amplitudenspektrum können wir Merkmale für dominante Richtungen und Frequenzen bestimmen.



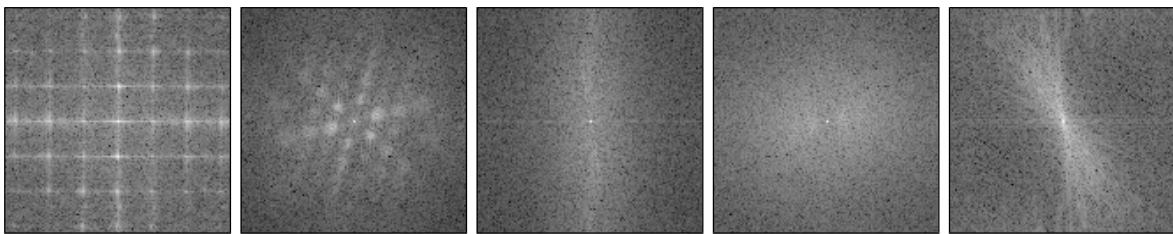


Abb. 266: Texturen oben und ihre Amplitudenspektren unten.

10.5 Punktmerkmale

Punktmerkmale, im Englischen oft *featurepoints*, *keypoints* oder auch nur *features* genannt, sind kontrastreiche Eckpunkte, die im Unterschied zu den vorangegangenen Merkmalen für ein gesamtes Bild bestimmt werden, ohne es vorgängig in mehrere Regionen zu segmentieren.

Solche Merkmale sind in einer Reihe von Anwendungen von Nutzen, wie z.B. bei der Objektdetektion, Objektverfolgung, bei der Zuordnung von Bildstrukturen in Stereoaufnahmen, als Referenzpunkte beim Zusammensetzen von Panoramen oder beim Kalibrieren von Kamerasystemen.

Obwohl Merkmalspunkte auffällig sind, so ist deren automatische Detektion nicht so einfach, wie man meinen könnte. Ein guter Detektor muss wichtige von unwichtigen Eckpunkten unterscheiden und diese zuverlässig und genau lokalisieren können. Er muss zudem möglichst dieselben Punkte in verschiedenen Bildern einer Szenerie finden, auch wenn die Bilder eine unterschiedliche Grösse, Perspektive oder Beleuchtung aufweisen. Zu diesem Zweck kann ein Punktmerkmal auch durch einen Deskriptor definiert werden, der Information über die nähere Umgebung des Punktes enthält.

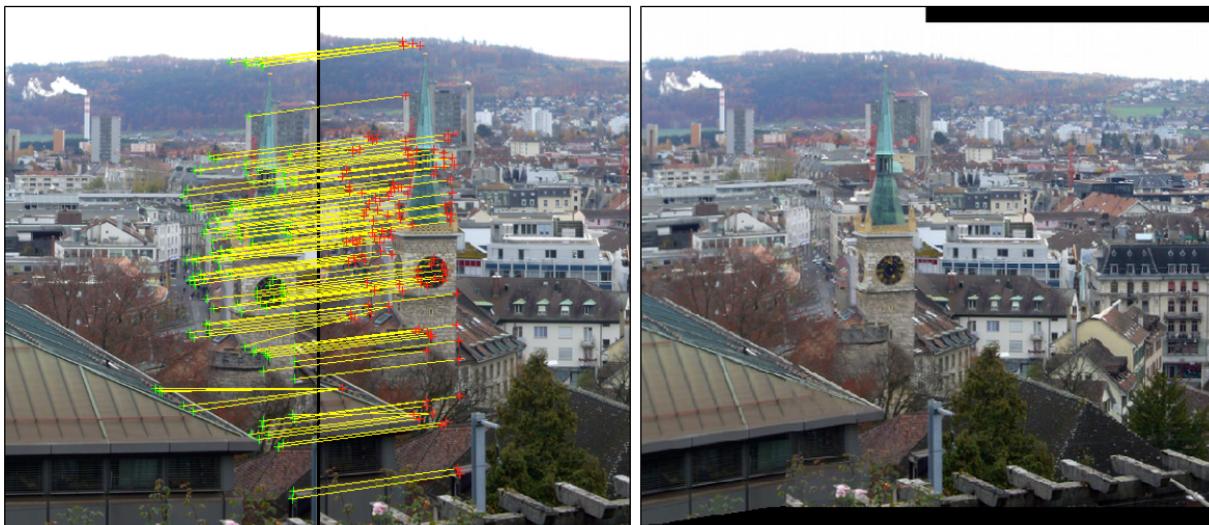


Abb. 267: Panorama Stitching: Korrespondierende Punktmerkmale in zwei Bildern können verwendet werden, um mehrere Bilder zu einem Panorama zusammenzusetzen. Quelle: [Cattin12]

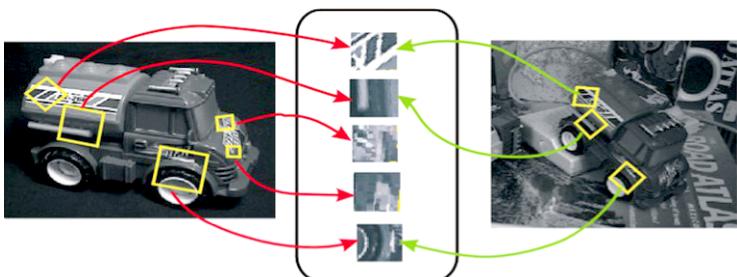


Abb. 268: Objektdetection: Objekte können in einer Datenbank durch sogenannte markante Punktmerkmale definiert werden. Damit kann ein Objekt in Bildern mit anderer Perspektive wieder gefunden werden. [Lowe99]

10.5.1 Harris Corners

Der *Harris Corner Detector* wurde von Harris und Stephens 1988 [Harris88] entwickelt und ist eine Weiterentwicklung von ähnlichen Algorithmen, die auf der gleichen Idee basieren: Ein Eckpunkt ist ein Ort, wo der Gradient gleichzeitig in mehr als einer Richtung einen hohen Wert aufweist. Kanten, wo der Gradient zwar hoch ist, aber nur in einer Richtung ausgeprägt ist, sollen nicht als Eckpunkte erkannt werden. Die nachfolgende Beschreibung des Algorithmus folgt einer ausführlichen Herleitung in [Burger06].

10.5.1.1 Lokale Strukturmatrix

Basis des Harris-Detektors sind die 1. Ableitungen des Bildes $I(x,y)$ in horizontaler und vertikaler Richtung mit einem einfachen Hochpassfilter, wie wir ihn in Kapitel 6.1.3 kennengelernt haben:

$$I_x(x,y) = \frac{dI}{dx}(x,y) \text{ und } I_y(x,y) = \frac{dI}{dy}(x,y)$$

An allen Positionen $[x,y]$ werden daraus nun drei Werte $A(x,y)$, $B(x,y)$ und $C(x,y)$ berechnet:

$$A(x,y) = I_x^2(x,y) \quad B(x,y) = I_y^2(x,y) \quad C(x,y) = I_x(x,y) \cdot I_y(x,y)$$

[Burger06] empfiehlt an dieser Stelle zusätzlich einen Gaussfilter mit einer Filtergrösse von 7×7 anzuwenden, um die Detektion unempfindlicher gegenüber Rauschen und feinsten Kanten zu machen.

$$\bar{A}(x,y) = A(x,y) * G_\sigma \quad \bar{B}(x,y) = B(x,y) * G_\sigma \quad \bar{C}(x,y) = C(x,y) * G_\sigma$$

Danach wird die sogenannte *lokale Strukturmatrix* gebildet:

$$M = \begin{bmatrix} \bar{A}(x,y), \bar{C}(x,y) \\ \bar{C}(x,y), \bar{B}(x,y) \end{bmatrix}$$

Die Matrix M charakterisiert die Struktur in der Umgebung des Punktes $[x,y]$:

keine Struktur	Kante (blauer Pfeil = Kantenvektor)	Ecke (Doppelecke "Schachbrett")	rechtwinklige Ecke
$M = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$M = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}$	$M = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$	$M = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$

Abb. 269: V.l.n.r.: Strukturmatrix M bei einer homogenen Umgebung, bei einer vertikalen Kante, bei einer Schachbrettecke und bei einer einfachen Ecke.

- Wenn der Punkt in einer uniformen Region des Bildes liegt, dann sind sämtliche Ableitungen Null und M ist somit eine Nullmatrix.
- Wenn der Punkt auf einer Kante parallel zur x-Richtung ist, dann ist $A > 0$ aber B und C sind null.
- Wenn der Punkt auf einer Ecke liegt (z.B. eine Ecke eines Schachbrettmusters), dann messen wir an einigen Stellen in der Umgebung ein $A > 0$, an anderen Stellen aber ein $B > 0$. Evtl. ist die Summe über alle C Werte ($= I_x I_y$) im Fensterbereich null (Schachbrettecke), evtl. aber auch nicht (einseitige Ecke).

10.5.1.2 Corner Response Function

Da wir aber nicht nur Ecken von horizontalen und vertikalen Strukturen detektieren wollen, müssen wir einmal mehr die Eigenwerte zu Hilfe nehmen. Ähnlich wie die Kovarianzmatrix im Kapitel 7.4.1 lässt sich die Strukturmatrix M diagonalisieren, um daraus die Eigenwerte zu gewinnen:

$$\lambda_{1,2} = \frac{\text{trace}(M)}{2} \pm \sqrt{\left(\frac{\text{trace}(M)}{2}\right)^2 - \det(M)} \quad (\text{trace}(M) = \text{Spur von } M, \det(M) = \text{Determinante von } M)$$

$$\lambda_{1,2} = \frac{1}{2} (\bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2})$$

Die Eigenwerte λ_1 und λ_2 sind immer positiv und enthalten die rotationsinvariante Information über die lokale Kantenstruktur (s. auch Kapitel 7.4.4.1). Innerhalb einer homogenen Umgebung sind auch die Eigenwerte null. Umgekehrt gilt auf einer perfekten Kante $\lambda_1 > 0$ und $\lambda_2 = 0$, unabhängig von der Orientierung der Kante. Die Eigenwerte kodieren ja die Kantenstärke, die zugehörigen Eigenvektoren die Kantenrichtung.

An einer Ecke muss also eine starke Kante sowohl in der Hauptrichtung (gemäß dem grösseren der beiden Eigenwerte) wie auch normal dazu (gemäß dem kleineren Eigenwert) vorhanden sein. Die Differenz der beiden Eigenwerte sollte demnach an einer Ecke sehr klein sein:

$$\lambda_1 - \lambda_2 = 2 \cdot \sqrt{\left(\frac{\text{trace}(M)}{2}\right)^2 - \det(M)}$$

Davon ausgehend schlägt Harris nun folgende *Corner Response Function (CRF)* vor:

$$Q(x, y) = \det(M) - \alpha \cdot (\text{trace}(M))^2$$

$$Q(x, y) = (\bar{A}\bar{B} - \bar{C}^2) - \alpha \cdot (\bar{A} + \bar{B})^2$$

Der Parameter α steuert die Empfindlichkeit des Eckendetektors und hat meist einen Wert von 0.04 - 0.06. Je grösser der α -Wert, umso weniger Punkte werden gefunden.

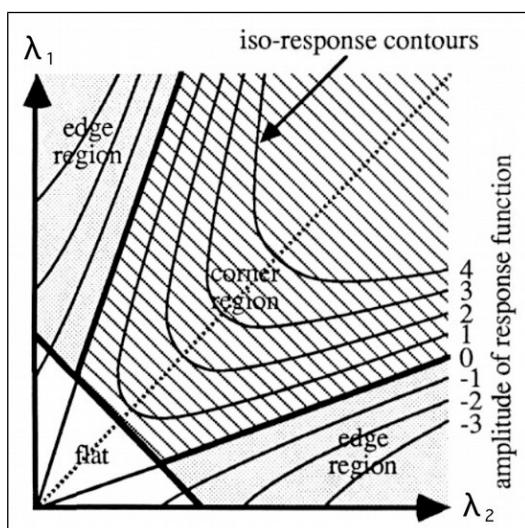


Abb. 270: Kurven gleicher Stärke der Corner Response Function in Abhängigkeit der beiden Eigenwerte. Quelle: [Harris88]

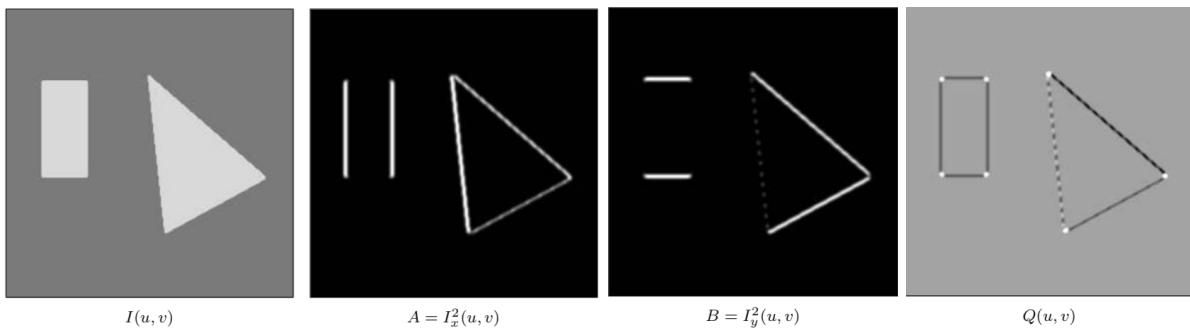


Abb. 271: V.l.n.r: Ausgangsbild, vertikale Kanten quadriert, horizontale Kanten quadriert und die Ausgabe der Corner Response Funktion als Bild dargestellt. Quelle: [Burger06]

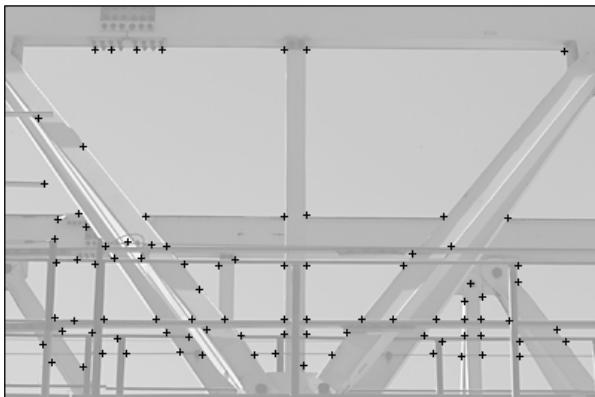


Abb. 272: Eckpunktendetektion in ImageJ mit einem α -Wert von 0.05



Auf die genaue Erklärung der Implementation wird hier verzichtet. Sie finden eine ausführliche Beschreibung in Java und ImageJ in [Burger06].



In Matlab finden Sie den *Harris Corner Detector* in der Funktion `corner`:

```
I = imread('..../images/Chessboard.jpg');
C = corner(I);
imshow(I);
hold on
plot(C(:,1), C(:,2), 'rO');
```

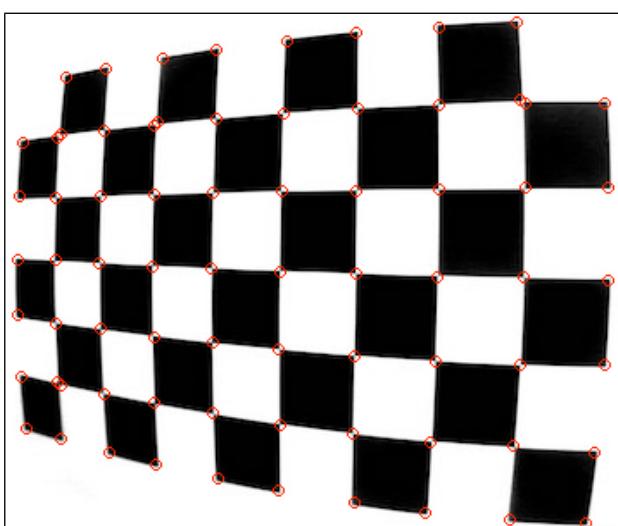


Abb. 273: Klassische Anwendung des *Harris Corner* Detektors: Eckpunkte finden in einem Kamera-Kalibrierungsschachbrett, um die Verzerrung der Optik zu berechnen. Gewisse Ecken sind doppelt vorhanden. Sie müssten in einem nachfolgenden Schritt noch bereinigt werden.

10.5.2 SIFT-Deskriptoren

Der *Scale-Invariant Feature Transform (SIFT)* Algorithmus findet Merkmalspunkte ebenfalls unabhängig von deren Lage und Orientierung. Er ist aber zusätzlich invariant gegenüber der Skalierung, Beleuchtung und bis zu einem gewissen Grad auch gegenüber affinen Transformationen und teilweisen Verdeckungen. Der Algorithmus gilt als Meilenstein der Computer Vision und wurde von David Lowe im Jahr 1999 [Lowe99] veröffentlicht und in den Vereinigten Staaten patentiert. Seine Publikation „*Distinctive Image Features from Scale-Invariant Keypoints*“ von 2004 [Lowe04] ist die mit Abstand am meisten zitierte Publikation der Computer Vision Wissenschaften.

Das Verfahren zu Erstellung der SIFT-Merkmale ist in vier Schritte gegliedert:

1. Detektion der Extremwerte im Multiskalenraum
2. Reduktion der Extrempunkte zu Keypoints
3. Bestimmung der Orientierung der Keypoints
4. Bestimmung eines Deskriptors für jeden Keypoint

10.5.2.1 Detektion der Extremwerte im Multiskalenraum

Das Hauptproblem bei den Eckpunkten aus dem Harris-Detektor ist, dass je nach Skalierung eines Bildes unterschiedliche Eckpunkte gefunden werden. Aus einer Kante in einem grösseren Bild kann schnell eine Ecke in einem verkleinerten Bild werden.

Um eine Skalierungsinvarianz zu erreichen, wird bei SIFT ein sogenannter Multiskalenraum erstellt. Dieser besteht aus mehreren Grössen eines Bildes, genannt *Oktaven* und in jeder Oktave aus mehreren Gauss-gefilterten Versionen eines Bildes, genannt *Skalen*. Davon werden wiederum durch Differenzbildung Kantenbilder erstellt.

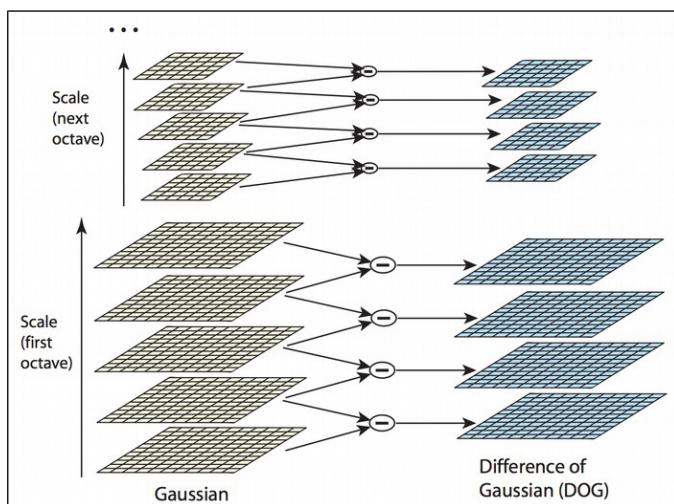
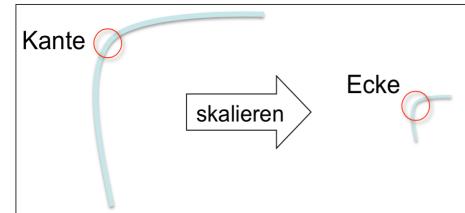


Abb. 274: Multiskalenraum aus mehreren Oktaven mit mehreren Skalen. Quelle: [Lowe99]

10.5.2.1.1 Skalen

In einem ersten Schritt werden von einem Ausgangsbild $I(x,y)$ mit dem Gaussfilter G (s. 6.1.2.2) und der Faltungsoperation k verunschärzte Skalen $S_k(x,y)$ erstellt. Lowe schlägt vor, 5 Skalen mit einem um den Faktor $\sqrt{2}$ vergrösserten Sigma zu erstellen. Als initiales Sigma verwendet er einen Wert von 1.6:

$$S_k(x,y) = G(k \cdot \sqrt{2} \cdot \sigma) * I(x,y)$$

Da die Merkmalsdetektion wie beim Harris Corner Detektor im Kantenbild erfolgt, müssten wir von den verunschärften Bildern nochmals eine Faltung mit dem Sobel- (1.

Ableitung s. 6.1.3.1) oder dem Laplace-Filter (2. Ableitung s. 6.1.3.2) erstellen. Da die Faltungsoperation ja relativ teuer ist und wir bereits mehrere verunschärzte Skalen des selben Bildes haben, können wir ein Kantenbild $D_k(x,y)$ auch durch Subtraktion zweier benachbarter Skalen gewinnen. So ein Differenzbild heisst auch *Difference of Gaussian (DoG)* und ist eine Annäherung an ein Kantenbild, das mit dem *Laplace of Gaussian (LoG)* Filter erstellt worden wäre. Wir haben solche DoG-Kantenbilder schon einmal beim Schärfen von Bildern mit der Unscharfmaskierung (s. 6.1.5) kennengelernt.

$$D_k(x,y) = S_{k+1}(x,y) - S_k(x,y)$$

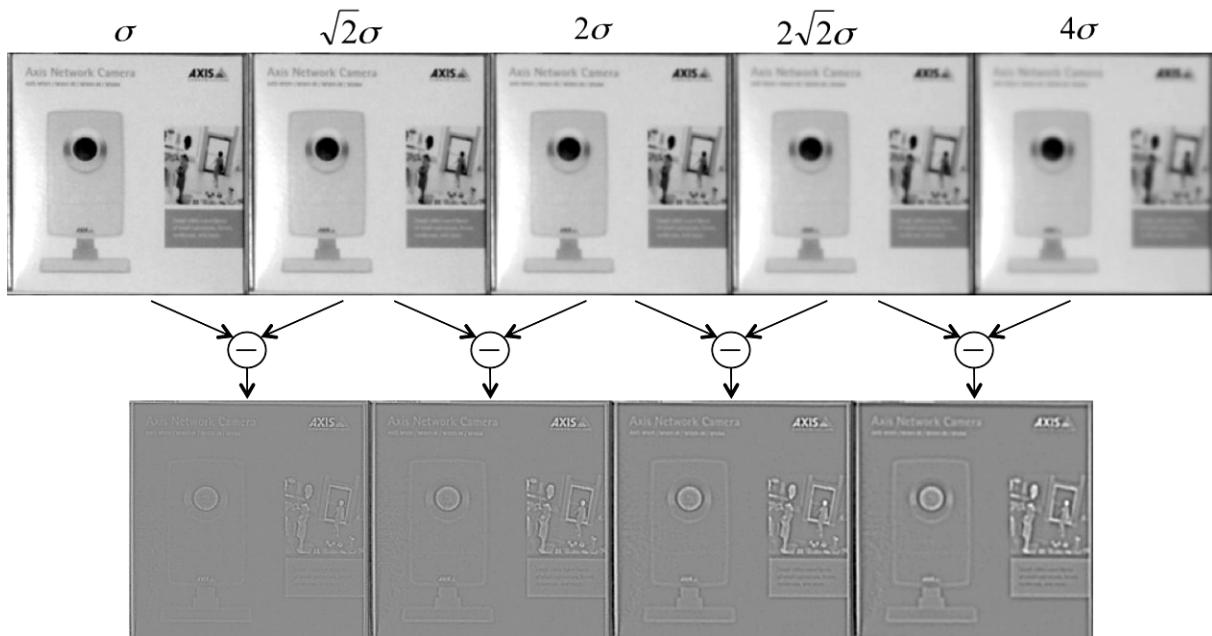


Abb. 275: 5 mit Gaussfilter erstellte Skalen und darunter die 4 durch einfache Differenz erzeugten Kantenbilder. Die Kantenbilder sind zwecks Visualisierung aufgehellt, da Differenzbilder sonst praktisch schwarz sind. Quelle: [Cattin12].

10.5.2.1.2 Oktaven

Im zweiten Schritt werden die Oktaven erstellt. Lowe schlägt 4 Oktaven vor, die durch Weglassen jeder zweiten Zeile und Spalte des mit 2σ gefilterten Bildes entstehen.

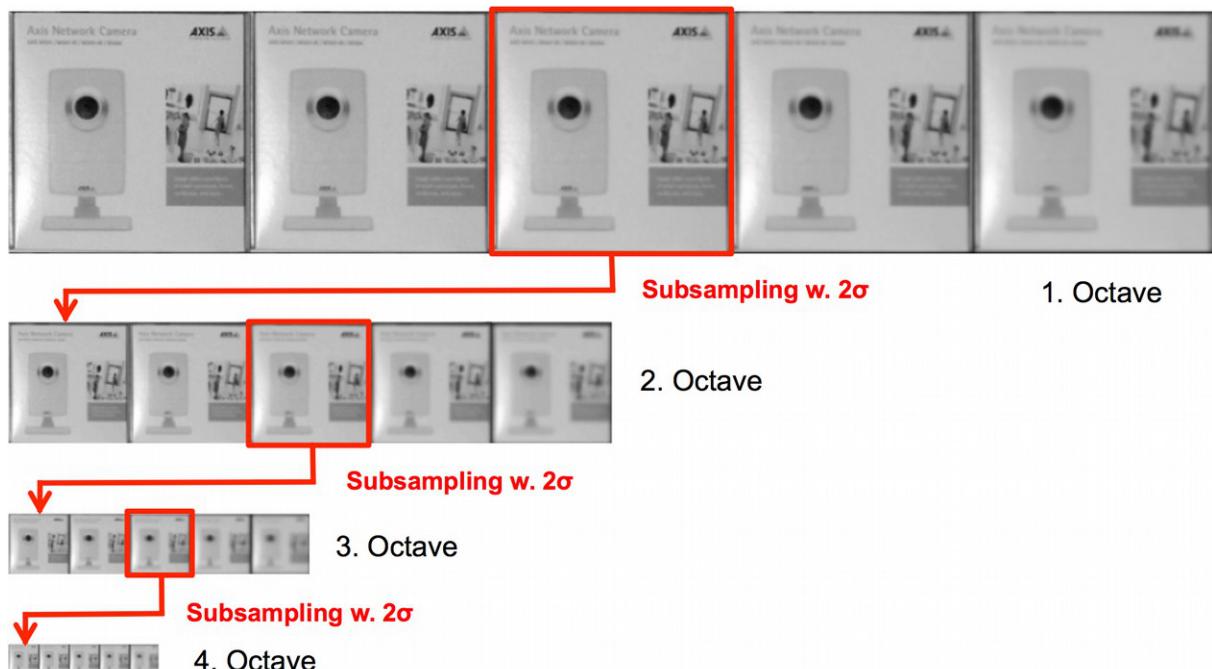
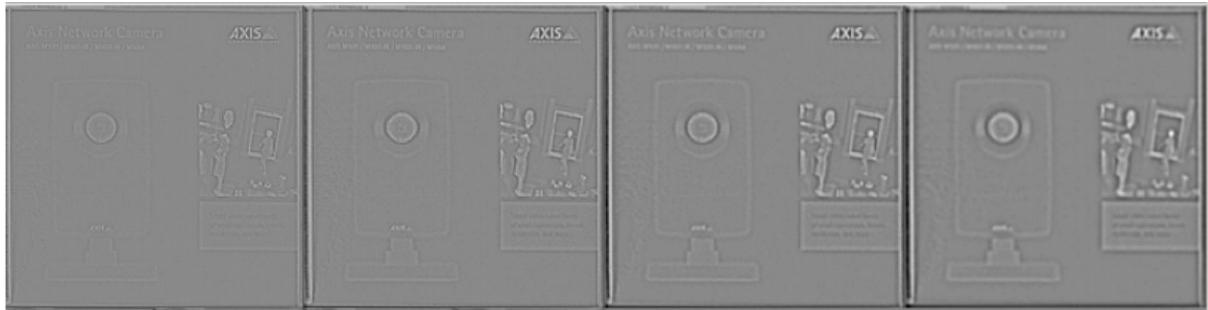


Abb. 276: 4 Oktaven von Gaussbildern mit je 5 Skalen. Quelle: [Cattin12]



1. Octave



2. Octave



3. Octave

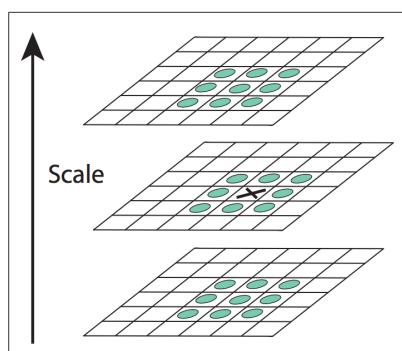


4. Octave

Abb. 277: Multiskalenraum von DoG-Kantenbilder aus 4 Oktaven mit jeweils 4 Skalen. Die Bilder sind wiederum zwecks Visualisierung aufgehellt. Quelle: [Cattin12]

10.5.2.1.3 Extremwerte bestimmen

In den Kantenbildern aller Oktaven werden nun die Kandidaten für die *Keypoints* bestimmt.



Dazu wird in einer $9 \times 9 \times 9$ Pixel grossen Umgebung das Maximum und Minimum bestimmt. Um ein Pixel (in der nachfolgenden Abb. mit X markiert) werden die 8 Nachbarn sowie die 9 Nachbarn in der darüberliegenden und die 9 Nachbarn in der darunterliegenden Skala verglichen. In der ersten und letzten Skala werden keine Pixel verglichen, da sie ja nicht 2 Nachbarbilder haben. Aus 4 Kantenbilder einer Oktave entstehen so 2 Ergebnisbilder und aus allen 4 Oktaven 8 Ergebnisbilder mit Extremwertpixeln.

Abb. 278: Links: $9 \times 9 \times 9$ Nachbarn zur Bestimmung der Extremwerte. Quelle: [Lowe99]

Es entstehen so vorerst sehr viele Extremwertpixel (s. Abb. 279), die in der Folge auf die wirklich markanten Keypoints reduziert werden müssen. Zuerst muss aber für alle Kandidaten die exakte Position im Ausgangsbild in Subpixelgenauigkeit bestimmt werden.

To Do: Erklärung der Subpixelgenauigkeit mit Taylorreihe

10.5.2.2 Reduktion der Extrempunkte zu Keypoints

Als Erstes werden alle Extremwertpunkte verworfen, welche in der Umgebung einen geringen Kontrast aufweisen. Lowe schlägt dazu einen Schwellwert von 0.03 im normierten Grauwertbereich von 0-1 vor.

Als Zweites werden alle Extrempunkte verworfen, welche nur auf einer Kante liegen. Dazu wird wie beim *Harris Corner Detektor* eine Strukturmatrix verwendet, die nun aber aus den Werten der DoG-Kantenbilder (= 2. Ableitung) gebildet wird. Diese Matrix wird auch *Hesse-Matrix* genannt:

$$H = \begin{bmatrix} D_{xx}, D_{xy} \\ D_{xy}, D_{yy} \end{bmatrix}$$

Über die Spur und die Determinante werden die Summe der Eigenwerte sowie das Produkt der Eigenwerte berechnet:

$$\text{trace}(H) = D_{xx} + D_{yy} = \lambda_1 + \lambda_2$$

$$\det(H) = D_{xx} \cdot D_{yy} - D_{xy}^2 = \lambda_1 \cdot \lambda_2$$

Wenn wir r als das Verhältnis der beiden Eigenwerte definieren $r = \lambda_1 / \lambda_2$ dann ergibt sich daraus $\lambda_1 = r \lambda_2$.

Daraus können wir folgenden Term bilden:

$$\frac{\text{trace}(H)^2}{\det(H)} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2^2} = \frac{(r+1)^2}{r}$$

Wie beim [Harris Corner Detektor](#) haben wir Ecken, wenn die Eigenwerte gleich sind. In diesem Fall wird der Ausdruck $(r+1)^2/r$ minimal. Der Ausdruck wird umso grösser, je grösser die Differenz der Eigenwerte ist. Lowe schlägt als Schwellwert für Eckpunkte einen Wert von $r=10$ vor und testet mit folgender Gleichung:

$$\frac{\text{trace}(H)^2}{\det(H)} < \frac{(r+1)^2}{r}$$

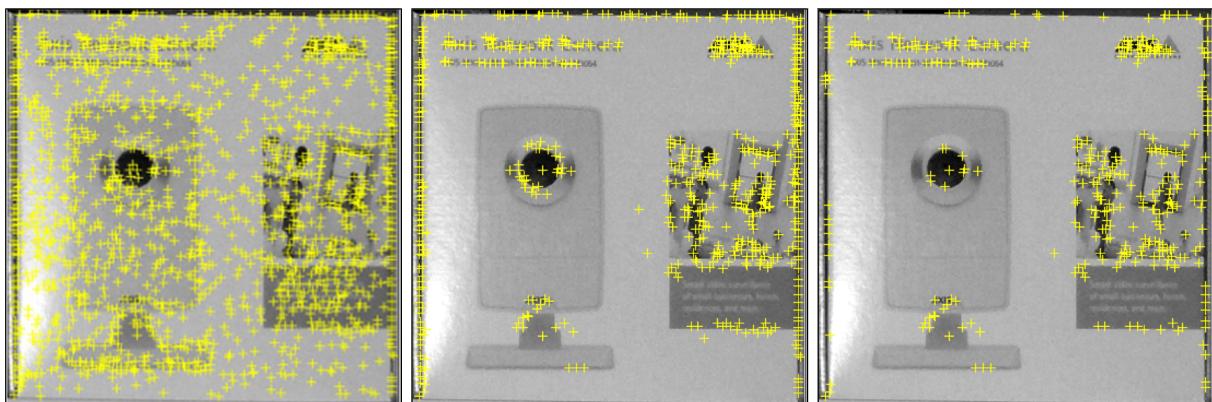
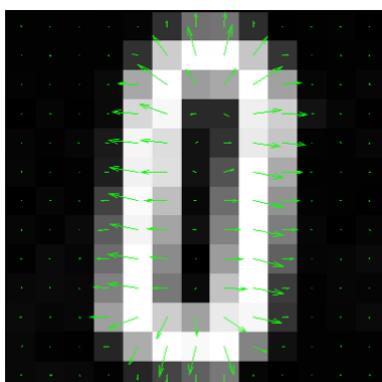


Abb. 279: Links: Alle Extrempunkte, Mitte: nach der Eliminierung der kontrastlosen Extrempunkte. Rechts: nach der weiteren Reduktion auf die Eckpunkte. Quelle: [Cattin12]

10.5.2.3 Bestimmung der Orientierung der Keypoints

Für jeden Keypoint wird nun eine Orientierung berechnet. Dazu wird in einer Kreisumgebung mit einem Radius des 1.5-fachen σ um den Keypoint in der Oktave und Skala, wo der Keypoint gefunden wurde, die dominierende Gradientenrichtung bestimmt. Damit wird am Schluss die Rotationsinvarianz erreicht. Die Gradienten werden durch einfache Differenz der Nachbarwerte bestimmt:



$$\begin{aligned} G_x &= G(x+1, y) - G(x-1, y) \\ G_y &= G(x, y+1) - G(x, y-1) \\ |G| &= \sqrt{G_x^2 + G_y^2} \\ \varphi &= \text{atan}(G_y / G_x) \end{aligned}$$

Die Gradientenrichtungen werden in einem Histogramm mit 36 Bereichen à je 10° Breite eingetragen. Der Histogrammeintrag wird zusätzlich gewichtet durch die Magnitude des Gradienten und durch die Distanz des Pixels zum Keypoint (je näher je höher). Der höchste Winkel-

bereich wird am Schluss zur Richtung des Keypoints. Gibt es noch weitere Winkelbereiche, die min. 80% der max. Richtung erreichen, so werden daraus neue Keypoints an derselben Stelle mit der entsprechenden Richtung und Magnitude generiert.

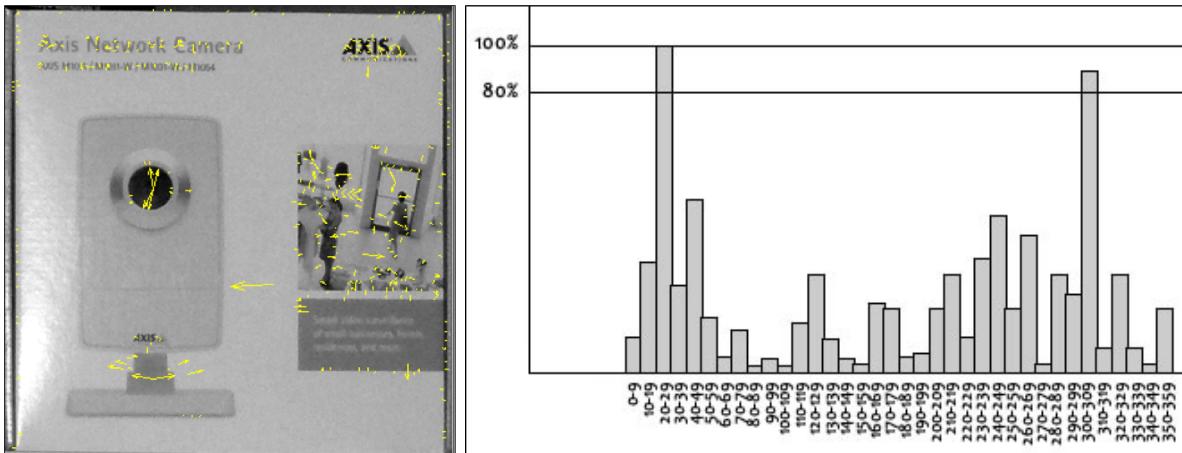


Abb. 280: Links: Finale Keypoints mit Richtungsvektoren, Quelle: [Cattin12]. Rechts: Gradienten Histogramm. Die max. Richtung wird die Richtung des Keypoints. Die 2. welche noch grösser ist als 80% der Grössten wird die Richtung eines neuen Keypoints. Quelle: [Nischwitz11]

10.5.2.4 Bestimmung eines Deskriptors für jeden Keypoint

Aus den Gradienten des vorangegangenen Schrittes wird zum Abschluss ein sogenannter Deskriptor für jeden Keypoint berechnet. Dieser fasst die charakteristischen Merkmale der näheren Umgebung eines Keypoints zusammen. Sie bilden eine Art Fingerabdruck des Keypoints.

Um den Keypoint werden 16×16 Gradienten in 4×4 Bereichen jeweils 8 Richtungsvektoren der darunterliegenden Gradienten zusammengefasst. Die zugrundeliegenden Gradienten werden zuerst in die Hauptrichtung des Keypoints gedreht, womit sie invariant gegenüber der Rotation werden.

Die Zusammenfassung der Gradienten geschieht wiederum mit Gauss-Gewichtung und in 8 Richtungen alle 45° . Daraus entsteht ein Deskriptor als $4 \times 4 \times 8 = 128$ -facher Merkmalsvektor.

Um den Deskriptor abschliessend noch invariant gegenüber Helligkeitsunterschieden zu machen, wird er auf Einheitslänge normiert.

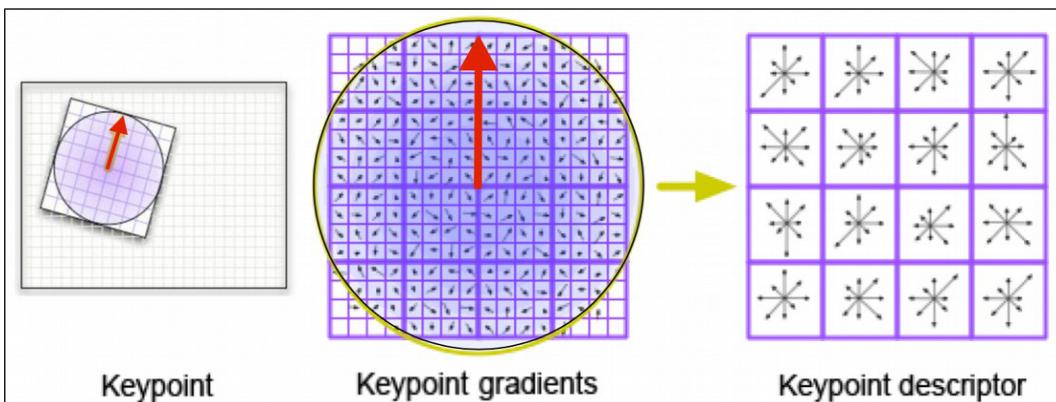


Abb. 281: Links: Ein Keypoint mit seiner Hauptrichtung. Mitte: Ein Keypoint mit den 16×16 darunterliegenden, normierten und Gauss gewichteten Gradienten. Rechts: Keypoint-Deskriptor als $4 \times 4 \times 8$ Gradienten = 128-facher Merkmalsvektor. Quelle: Jonas Hurrelmann

Die eigentliche Anwendung von SIFT-Deskriptoren ist die Suche nach ähnlichen Deskriptoren in einem anderen Bild zu suchen. Wir werden dies im Kapitel 11.2.1.2.3 tun.

10.5.3 SIFT-Nachfahren

Die Zeit bleibt nicht stehen und so wurden schon einige verbesserte Deskriptoren auf der Basis von SIFT entwickelt:

- **GLOH**: Das *Gradient Location & Orientation Histogram* Verfahren von [Miko05] unterscheidet sich v.a. bei der Berechnung des Deskriptors welcher aus einem kreisförmigen, log-polaren Gitter mit 16 Richtungen besteht. Weil daraus ein 272 dimensionaler Merkmalsvektor entsteht wird dieser mit der PCA auf 64 Dimensionen reduziert.
- **SURF**: Das *Speeded Up Robust Features* Verfahren wurde 2008 an der ETH entwickelt mit dem Ziel robuster und viel effizienter zu sein. Wie einige [Videos auf Youtube](#) belegen läuft die SURF-Implementation in [OpenCV](#) sogar in Echtzeit.



10.5.3.1 Deskriptoren in Matlab

SIFT: Wahrscheinlich aus patentrechtlichen Gründen gibt es von Haus keine SIFT-Funktionen in Matlab. In einem Beispiel in Kapitel 11.2.1.2.3, welches von [Lowe's Homepage](#) stammt, werden die Deskriptoren mit der Funktion `sift` berechnet. Diese Funktion ruft aber ein ausführbares Programm auf, welches (für Windows und Linux kompiliert) im gleichen Ordner liegt.

Eine andere SIFT-Implementation für Matlab finden Sie in der [VLFeat](#) Bibliothek.

SURF, GLOH und noch andere Deskriptoren finden Sie in Matlab in der [Computer Vision Toolbox](#).



10.5.3.2 Deskriptoren in Fiji

SIFT: Die ImageJ Erweiterung Fiji bietet über das [Feature Extraction Plugin](#) eine SIFT-Implementation von Stefan Saalfeld an.

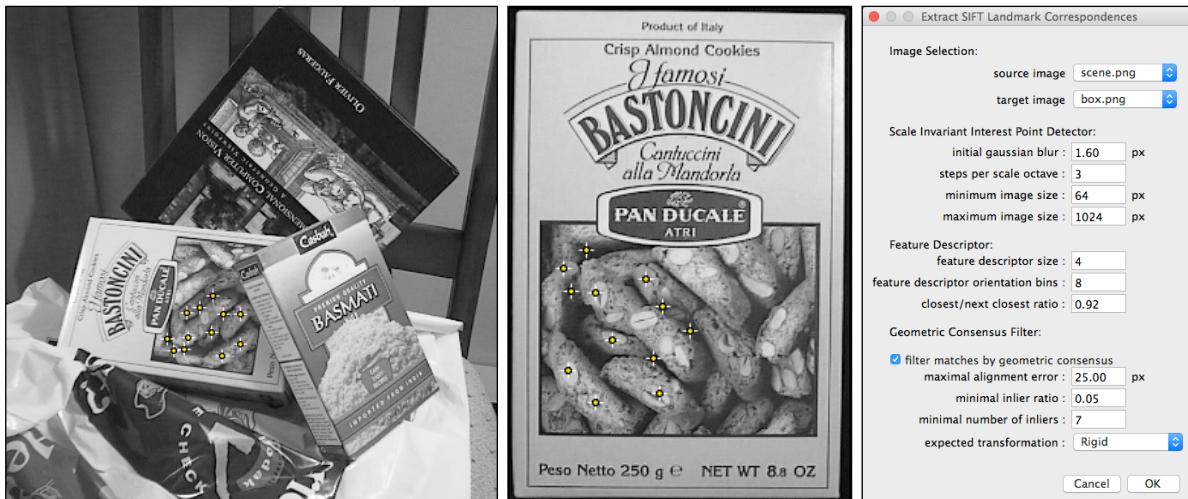


Abb. 282: SIFT-Feature Detection in Fiji mit dem Feature Extraction Plugin.

10.6 Beispiel & Übung

10.6.1 In ImageJ mit ijblob



Für dieses Beispiel verwenden wir das [ijblob](#) Plugin von Torsten Wagner für ImageJ. Es besteht aus der [ijblob](#) Bibliothek und dem GUI-Tool Shape Filter.

Das Bild [screws.png](#) wurde automatisch binarisiert und 6-fach dilatiert und wieder erodiert, um die Löcher zu schliessen. Danach wurde der Shape Filter Dialog aufgerufen und das Resultat aller Merkmale wird in einer Tabelle ausgegeben. Die Bedeutung einiger zusätzlicher Merkmale finden sie [hier](#).

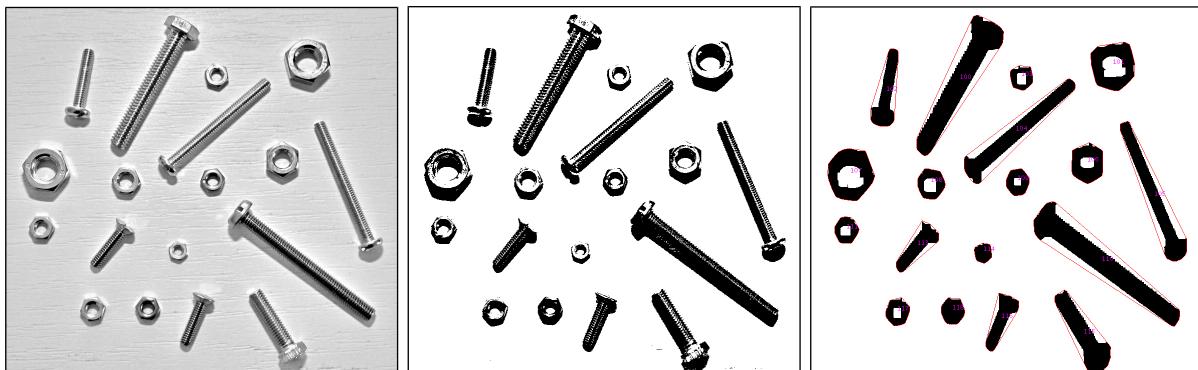
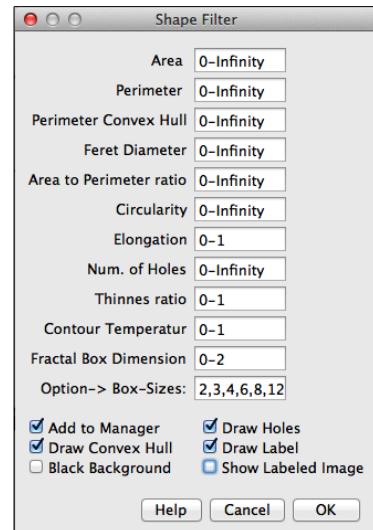


Abb. 283: Links: Original, Mitte nach Binarisierung und rechts mit konvexen Hüllen nach der Merkmalsextraktion von ijblob.

Label	X	Y	A	P	H	D	A/P	C	E	#H	TR	CT	FD	FDG
109	412	349	1888	158	160	55.7	11.9	13.3	0.39	1	0.95	0.12	1.75	1.00
107	78	334	7296	316	314	108.2	23.1	13.7	0.37	1	0.92	0.12	1.70	1.00
108	239	353	2545	187	185	64.3	13.6	13.8	0.34	1	0.91	0.13	1.80	1.00
118	282	608	1888	162	161	55.5	11.6	13.9	0.34	0	0.90	0.13	1.77	1.00
111	70	446	1896	164	161	55.2	11.5	14.3	0.36	1	0.88	0.15	1.71	1.00
106	552	312	3933	238	235	82.1	16.5	14.4	0.38	1	0.87	0.14	1.78	1.00
101	603	117	6582	309	303	103.4	21.3	14.5	0.27	1	0.87	0.15	1.70	1.00
114	344	491	1034	124	121	41.2	8.3	14.8	0.31	0	0.85	0.16	1.63	1.00
117	172	610	1899	169	164	58.0	11.2	15.1	0.35	1	0.83	0.17	1.73	1.00
103	420	141	1657	160	154	53.3	10.3	15.5	0.13	1	0.81	0.17	1.67	1.00
116	380	625	3063	297	286	117.4	10.3	28.8	0.86	0	0.44	0.17	1.61	1.00
113	212	479	3028	298	278	113.7	10.2	29.3	0.85	0	0.43	0.20	1.62	1.00
115	545	656	6381	460	420	179.8	13.9	33.1	0.87	0	0.38	0.22	1.68	1.00
102	150	171	4015	375	361	156.9	10.7	35.0	0.90	0	0.36	0.18	1.69	1.00
100	297	147	12418	751	681	300.6	16.5	45.4	0.92	0	0.28	0.23	1.71	1.00
110	581	511	13654	868	788	355.1	15.7	55.2	0.94	0	0.23	0.23	1.69	1.00
104	409	250	8096	682	626	284.3	11.9	57.5	0.94	0	0.22	0.22	1.62	1.00

Abb. 284: Merkmale extrahiert mit ijblob, sortiert nach der Exzentrizität E. Die Schrauben lassen sich klar durch die Exzentrizität unterscheiden.

Legende:

- X: Schwerpunkt in X-Richtung
- Y: Schwerpunkt in Y-Richtung
- A: Fläche in Pixel
- P: Perimeter in Pixel
- H: Perimeter der konvexen Hülle
- D: Durchmesser (Feret Durchmesser)
- C: Circularity nach ijblob: P^2/A
- E: Elongation (entspricht unserer Exzentrizität)
- #H: Anzahl Löcher
- TR: Thinness Ratio (entspricht unserer Circularity)
- CT: Contour Temperatur
- FD: Fractal Dimension
- FDG: Fractal Dimension Goodness

**10.6.2 Übung in Matlab: Merkmale mit regionprops**

In Matlab erhalten Sie die Merkmale einer Region über die Funktion [regionprops](#). Sie erhalten entweder einzelne Merkmale oder alle zusammen in einem Vektor.

Aufgaben:

- Detektieren und markieren Sie alle grossen und kleinen Schrauben im Bild [screws.png](#).
- Zeichnen Sie ein konvexes Polygon um die Schrauben und ein Kreuz beim Schwerpunkt.
- Wie gut können die invarianten Momente (mit [invmoments.m](#)) die Schrauben von den Muttern unterscheiden?

**10.6.3 Übung in ImageJ: Hu's invariante Momente**

Wie bereits erwähnt gibt es für **ImageJ** das Plugin [Moment Calculator](#) welches die Momente bis zum 4. Grad berechnen kann. Die zentralen, die normalisierten zentralen und Hu's Momente müssten daraus berechnet werden.

Aufgaben:

- Implementieren Sie eine Klasse [Moments](#) mit einer Methode [invmoments](#), welche die 7 invarianten Momente von Hu zurückgibt. Wenn Sie von Grund auf starten, ohne das Plugin [Moment Calculator](#), dann finden Sie in [Burger06] Code für eine erste Implementierung. Dieser wir allerdings nicht sehr performant sein, da er für jedes einfache und zentrale Moment das ganze Bild durch iteriert. Einen geschickteren Aufbau finden Sie sicher im Skript [invmoments.m](#) von [Gonzales08].
- Vergleichen Sie Ihre Resultate mit den Werten vom Skript [invmoments.m](#).
- Testen Sie die Invarianz mit verschiedenen Versionen eines Bildes wie in Abb. 260.

**10.6.4 Übung: Rotationsinvarianz von Harris Corners**

Aufgabe: Testen Sie die Rotationsinvarianz des Harris-Corner-Detektors.

**10.6.5 Übung: Skalierungsinvarianz von SIFT-Punktmerkmalen**

Aufgabe: Testen Sie die Skalierungsinvarianz der SIFT-Punktmerkmale.

11 Klassifikation



Klassifikation ist der letzte wichtige Schritt in der Bildanalyse, indem wir die Merkmale von Pixeln oder Regionen den *Objektklassen* zuweisen und damit Objekte identifizieren. Erst nach diesem Schritt können wir von erkannten *Objekten* sprechen. Im vorangegangenen Kapitel haben wir gelernt, wie wir Pixel oder Regionen repräsentieren und durch Merkmale beschreiben können. Aus diesen Merkmalen müssen zuerst diejenigen ausgewählt werden, die eine möglichst hohe Trennkraft haben. Die n ausgewählten Merkmale bilden einen n -dimensionalen *Merkmalsraum* in dem wir die *Objektklassen* bestimmen müssen zu denen wir die Merkmale zuordnen können. Diese Bestimmung der Merkmale und Klassen geschieht in einer vorgängigen *Trainingsphase* und kann unüberwacht automatisch oder überwacht durch Vorwissen geschehen. Erst jetzt können wir die eigentliche *Klassifikation* vornehmen. Diese Zuweisung ist der eigentliche Kern der so genannten *Mustererkennung* (*Pattern Recognition* oder *Pattern Matching*), die oftmals gleichgesetzt wird mit der Klassifikation.

11.1 Merkmale und Klassen

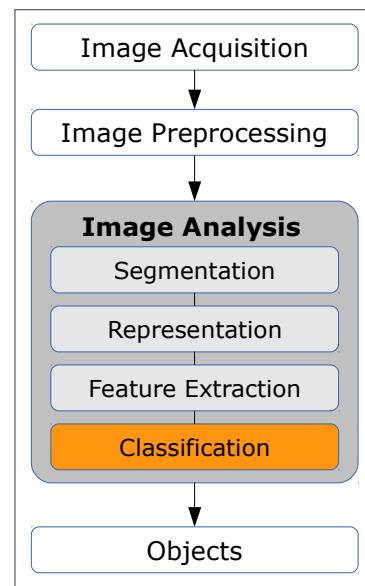
Die Auswahl der Merkmale und Klassen definiert die *Trainingsphase* des Klassifikationsprozesses.

11.1.1 Merkmale und Merkmalsraum

Im vorangegangenen Kapitel über Merkmale haben wir bereits viele verschiedene Arten von Merkmalen kennengelernt. Merkmale (*features*) sind nach der Segmentation einer Region oder einzelnen Pixeln zugewiesen. Allen Merkmalen gemeinsam ist, dass sie numerisch sind.

Jetzt soll es nur noch darum gehen, aus den extrahierten Merkmalen möglichst wenige gute Merkmale auszuwählen, damit wir alle Pixel oder Regionen möglichst fehlerfrei einer Objektklasse zuweisen können. Genügt ein Merkmal nicht für die Zuteilung, so kombinieren wir es mit einem weiteren Merkmal, bis die Klassenzuteilung möglichst fehlerfrei ist.

Aus den n ausgewählten Merkmalen wird ein *n-dimensionaler Merkmalsraum* (*feature space*) aufgespannt. Eine Region oder ein Pixel wird damit durch einen *n-dimensionalen Merkmalsvektor* (*feature vector*) beschrieben. Je mehr Dimensionen ein Merkmalsraum hat, umso schwieriger wird es auch mit den schnellsten Computern, damit eine Klassifikation durchzuführen.



Beispiel: Mehrkanalige Bilder

Für Farbbilder liegt es nahe, die Farbkanäle des jeweiligen Farbmodells als Merkmal zu verwenden. Im nachfolgenden Bild sollen die blauen Beeren bestimmt werden. Die blauen und grünen Bildinhalte lassen einem vermuten, dass das RGB-Farbmodell ein guter Kandidat für Merkmale sein könnte:



Abb. 285: Das Farbbild der Beeren ([blueberries.png](#))

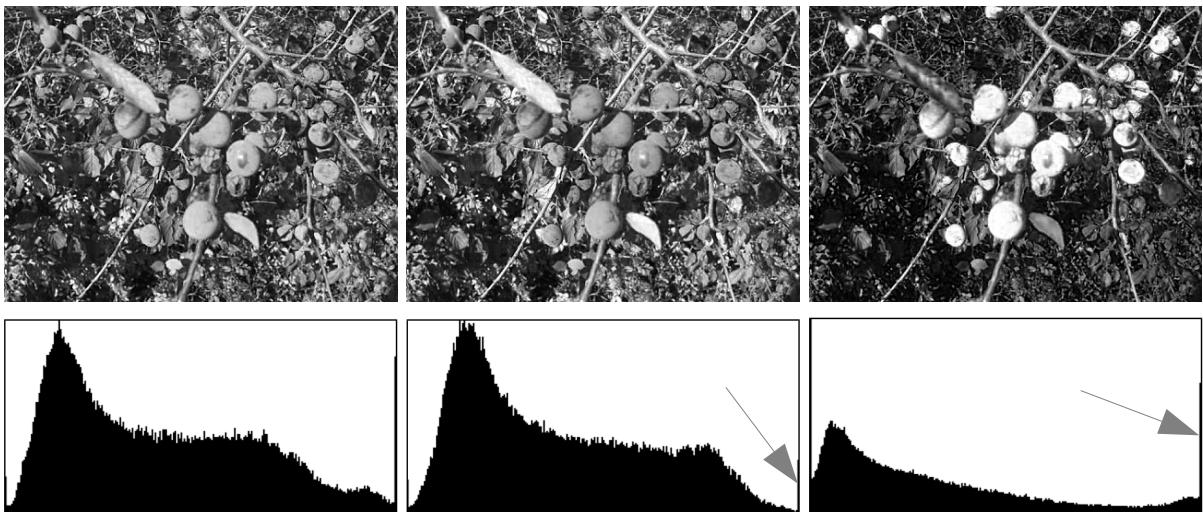


Abb. 286: Rotkanal links, Grünkanal in der Mitte und der Blaukanal rechts im RGB-Farbraum. Die Umwandlung wurde in ImageJ gemacht mit Image > Type > HSB-Stack und dann mit Image > Stacks > Stacks to Image in einzelne Graustufenbilder.

 Die Histogramme zeigen, dass die RGB-Kanäle ungeeignet sind für die Auftrennung. Der Blaukanal weist zwar einen starken Peak ganz rechts auf, aber dieser korreliert mit den Peaks in denselben Bereichen der anderen Kanäle.

Im nächsten Schritt schauen wir uns die Kombinationen von jeweils zwei Kanälen an:

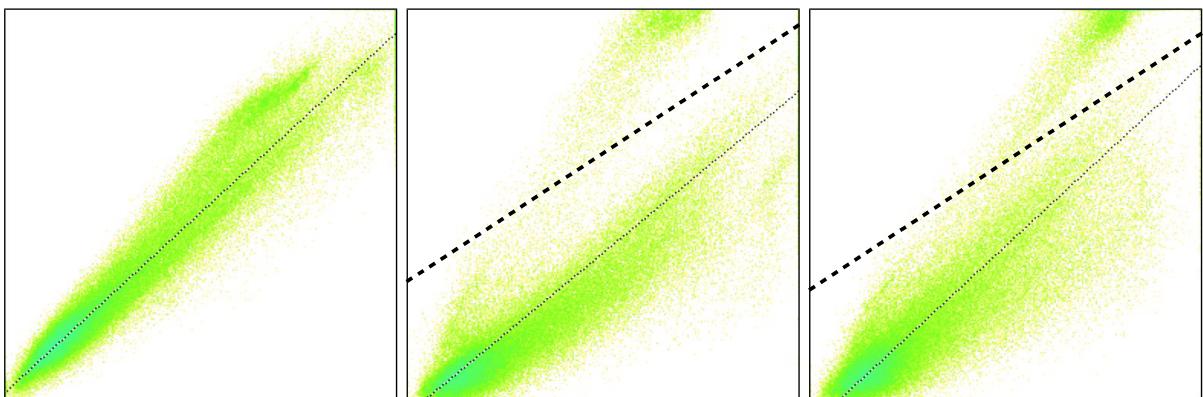


Abb. 287: V.l.n.r.: 2D-Histogramme für die Rot-Grün-, Rot-Blau- und Grün-Blau-Kanäle jeweils logarithmisch skaliert und invertiert dargestellt. Die 2D-Histogramme wurden mit Fiji aus den jeweiligen Farbkanälen mit Plugins > Analyze > 2D-Histogram erstellt. Die fein gestrichelte Linie ist die Achse, entlang derer die grösste Varianz besteht.

Die Trennbarkeit wird dabei v.a. in den beiden Kombinationen mit dem Blaukanal etwas besser (fett gestrichelte Linie). Erstaunlicherweise wäre die Trennbarkeit im Rot-Blau-Histogramm besser als im Grün-Blau-Histogramm.

Weitere Kandidaten für Merkmale ergäben sich im HSB-Farbraum (*hue*, *saturation*, *brightness*) (s. Kapitel 2.2.4, Seite 29):

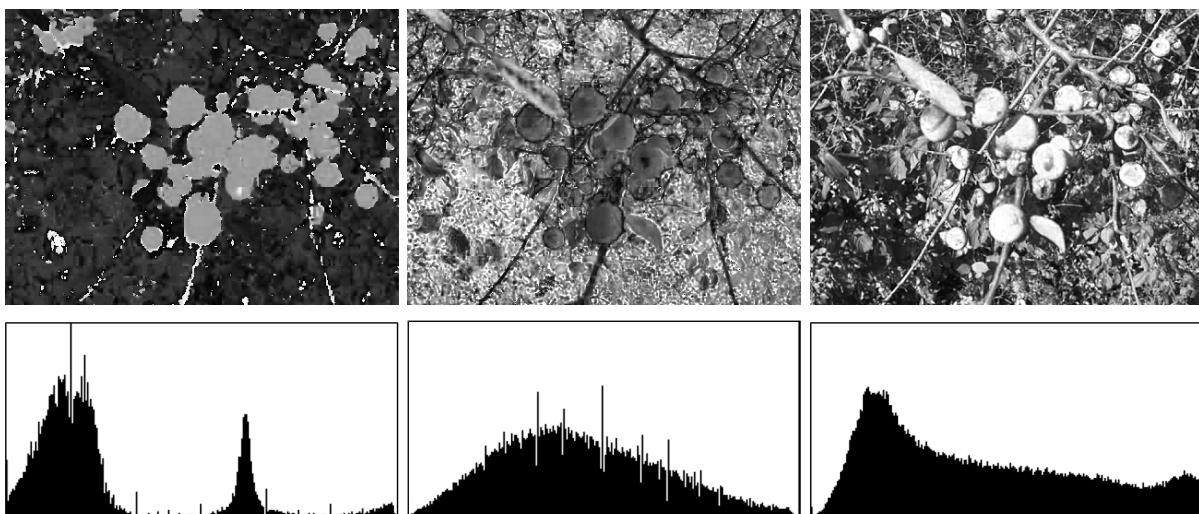


Abb. 288: Farbtonkanal (*Hue*) links, Sättigungskanal (*Saturation*) in der Mitte und der Helligkeitskanal (*Brightness*) rechts nach der Konvertierung in den HSB-Farbraum. An den 1D-Histogrammen der jeweiligen Kanäle erkennen wir nur im Farbtonkanal eine gute Separierbarkeit in 2 oder 3 Klassen.

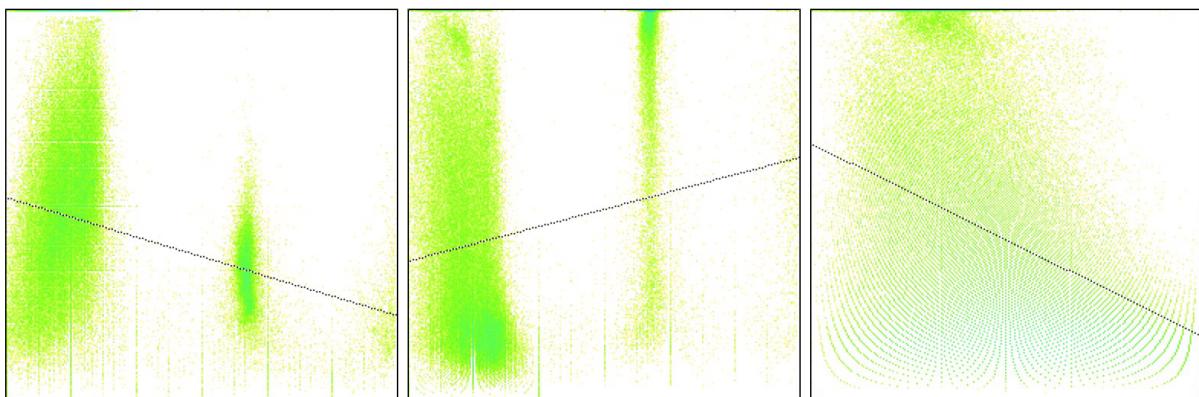


Abb. 289: V.l.n.r.: 2D-Histogramm für die *Hue-Saturation*-, die *Hue-Brightness*- und den *Saturation-Brightness*-Kanal. Beide Histogramme mit dem Farbtonmerkmal zeigen eine klare Trennbarkeit auf. Jedoch bringt die zusätzliche Dimension keine Verbesserung der Separierbarkeit gegenüber dem 1D-Hue-Histogramm.

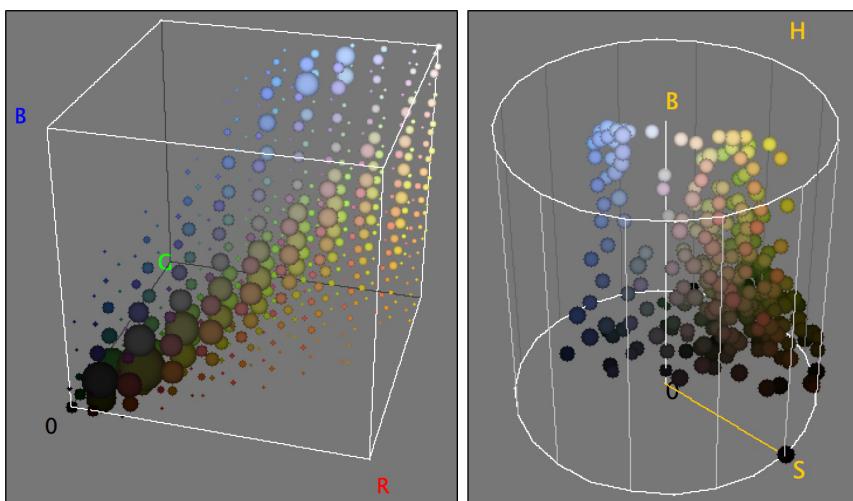


Abb. 290: Würde man zusätzlich ein 3. Merkmal verwenden, so ergäbe das einen 3D-Merkmalesraum, wobei die Häufigkeit eines 3D-Merkals (= 4. Dimension) in Form der Kugelgrösse dargestellt wird (links: RGB, rechts HSB). Man sieht gut, dass dieses zusätzliche Merkmal die Separierung in 2 oder 3 Klassen nicht verbessert.

Beispiel 2: Fischer's Iris Datenset

Die zu klassifizierenden Merkmale müssen nicht unbedingt direkt aus Pixelwerten abgeleitet sein. Ein sehr bekannter Datensatz der Klassifikationslehre ist das [Iris Datenset](#) vom britischen Statistiker [Sir Ronald Fisher](#) aus dem Jahre 1937. Er besteht aus 50

Messungen von 4 Merkmalen von 3 verschiedenen Schwertlilienarten (*Iris Setosa*, *Iris Virginica* und *Iris Versicolor*). Die Merkmale sind die Breite und Länge der Kelchblätter (*Sepal*) sowie der Kronblätter (*Petalum*).

Der Datensatz wird gerne als Testdatensatz für Lernverfahren wie [Support Vector Machinen](#) verwendet, da er linear trennbar ist. Für die Cluster-Analyse hingegen ist er ein Paradebeispiel, dass Cluster und Klassen nicht übereinstimmen müssen: Ohne Verwendung des Vorwissens über die realen Spezies werden zuverlässig nur zwei Cluster identifiziert, die Trennung des zweiten Clusters in zwei Spezies ist nur mit Vorwissen möglich.

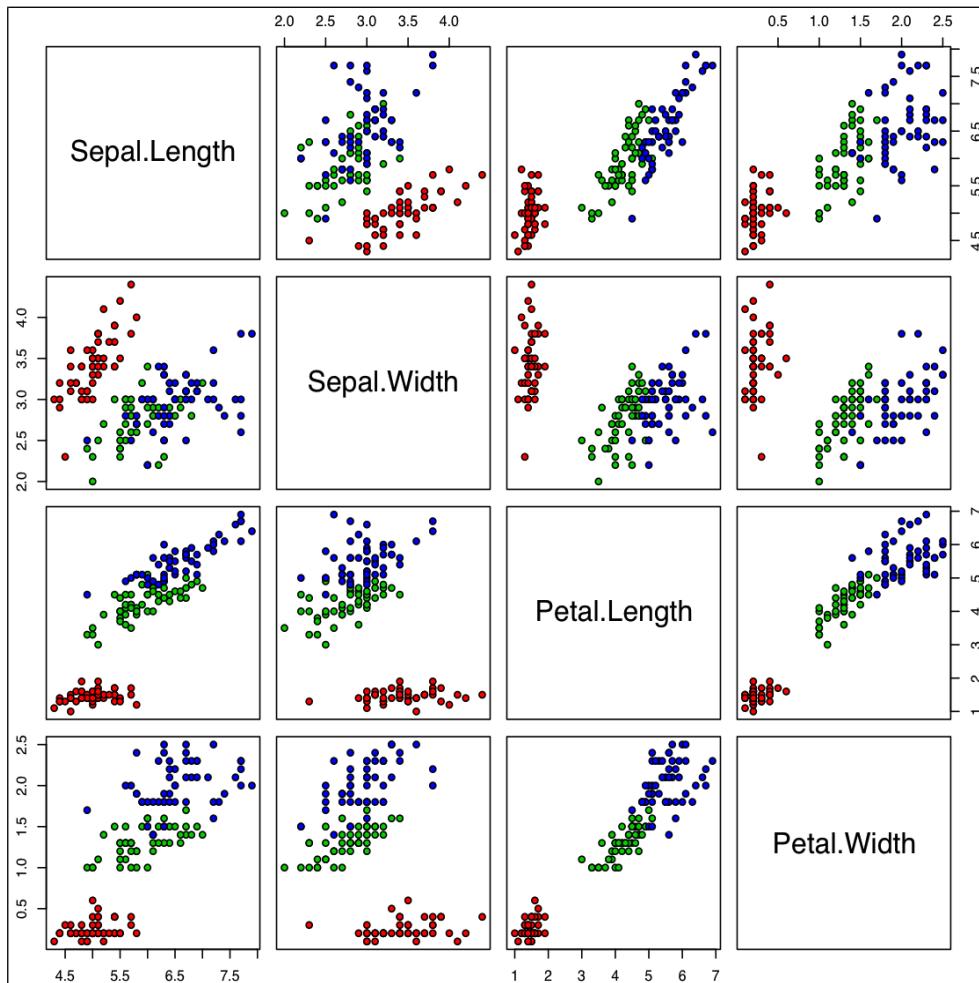


Abb. 291: Alle 6 2D-Histogramme aus den 4 Merkmalen *Sepal Length*, *Sepal Width*, *Pentalum Length* und *Pentalum Width*. Alle Kombinationen gegenüber der Hauptdiagonale sind gespiegelt. Aus n Merkmalen gibt es $(n^2-n)/2$ Merkmalskombinationen. Die roten Punkte sind Messungen der *Iris Setosa*, die grünen Punkte der *Iris Versicolor* und die blauen Punkte der *Iris Virginica* (Quelle: Wikipedia).

11.1.1.1 Reduktion der Merkmale mit der Hauptkomponentenanalyse

Je mehr Merkmale wir haben, umso schwieriger wird deren Beurteilung und Visualisierung. Um die Dimensionalität des Merkmalraums zu reduzieren, können wir die Hauptachsentransformation einsetzen. In Kapitel 7.4 haben wir an einem einfachen Datenset gesehen, wie wir dessen zwei Dimensionen auf eine reduzieren konnten. Im nachfolgenden Beispiel aus [Gonzales08] wollen wir die 6 Merkmale, die wir aus Landsat-Satellitenaufnahmen erhalten, auf die wichtigsten 3 Merkmale reduzieren. Der Landsat4-Satellit von der NASA liefert u.a. 6 Aufnahmen in den Spektralbereichen 450-520nm (Blau), 520-600nm (Grün), 630-690nm (Rot), 760-900nm (nahes Infrarot), 1550-1750nm (mittleres Infrarot) und 10'400-12'500nm (thermales Infrarot):

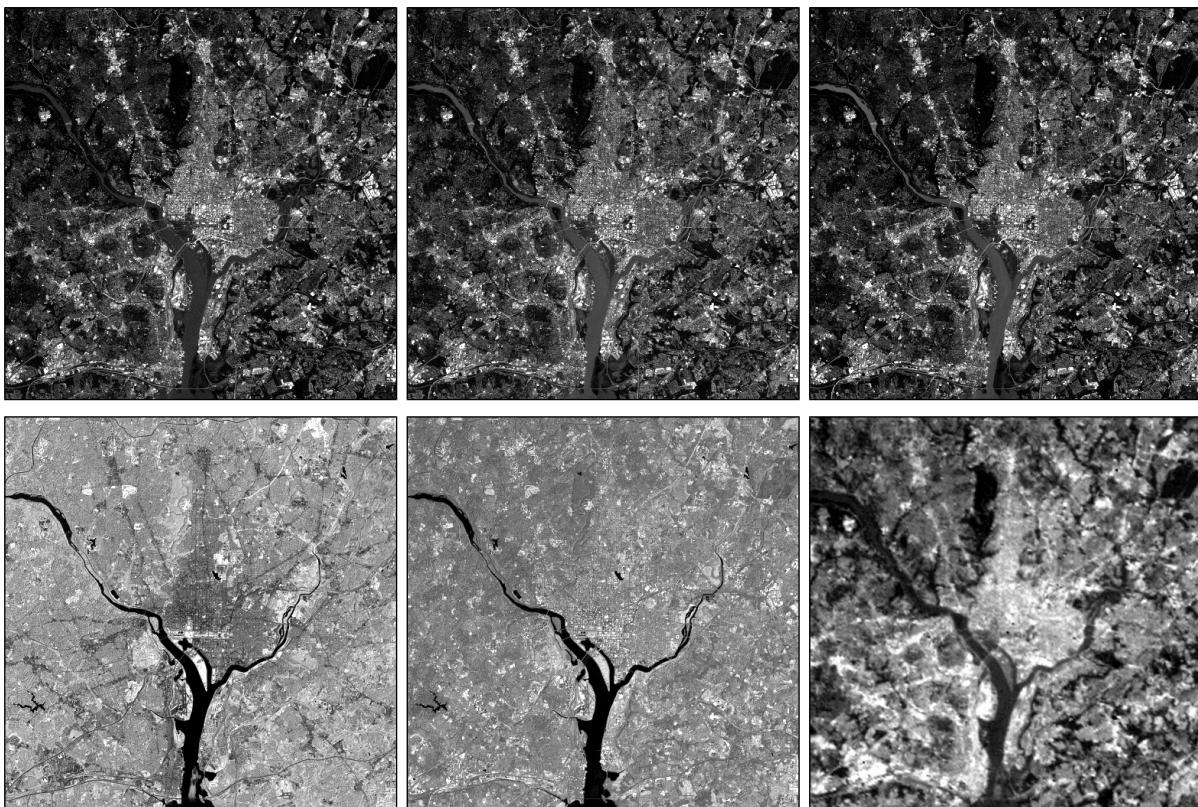


Abb. 292: LandSat-Satellitenaufnahmen: Oben: Blau-, Grün- & Rotkanal. Unten: Naher, mittlerer & thermaler Infrarotbereich. (Quelle: [Gonzales08])



Daraus berechnen wir im nachfolgenden Matlab-Skript die Hauptkomponenten. Nachdem wir die 6 Bilder der Grösse 512×512 geladen haben, konkatenieren wir sie in eine 3D-Matrix der Grösse $512 \times 512 \times 6$. Alle 6 Merkmale sind so übereinander in einem Stack. Als Nächstes werden alle Bilder mit der Funktion `imstack2vectors` (Quelle: [Gonzales08]) in einen Spaltenvektor der Länge 262'144 umgewandelt. Damit gehen wir in die Funktion `principalcomps` von [Gonzales08], die uns die Hauptkomponenten nach Grösse sortiert zurückgibt. Nach der PCA werden die Komponenten in den Spaltenvektoren wieder in Bilder umgewandelt und angezeigt.

```
% Read registered satellite images all of size 512 x 512
f1 = imread('..../Images/Sat1_00450-00520_b.tif');
f2 = imread('..../Images/Sat2_00520-00600_g.tif');
f3 = imread('..../Images/Sat3_00630-00690_r.tif');
f4 = imread('..../Images/Sat4_00760-00900_nir.tif');
f5 = imread('..../Images/Sat5_01550-01750_mir.tif');
f6 = imread('..../Images/Sat6_10400-12500_tir.tif');

% Concatenate all images int an 3D array of 512 x 512 x 6
S = cat(3, f1,f2,f3,f4,f5,f6);

% Convert to a matrix with 6 columns where each image is a column of lenght 262144
X = imstack2vectors(S);

% Do the pca with 6 components
P = principalcomps(X, 6);

d = diag(P.Cy)

% Get the single components and reshape the columns to an image of 512x512
PC1 = reshape(P.Y(:,1), 512, 512);
PC2 = reshape(P.Y(:,2), 512, 512);
PC3 = reshape(P.Y(:,3), 512, 512);
PC4 = reshape(P.Y(:,4), 512, 512);
PC5 = reshape(P.Y(:,5), 512, 512);
```

```

PC6 = reshape(P.Y(:,6), 512, 512);

% Show all components
figure; imshow(PC1, []);
figure; imshow(PC2, []);
figure; imshow(PC3, []);
figure; imshow(PC4, []);
figure; imshow(PC5, []);
figure; imshow(PC6, []);

```

Wir erkennen, dass die ersten drei Komponenten am meisten Kontrast (=Varianz) enthalten und das die letzten drei Komponenten kaum noch Information beitragen. Der Kontrast in den einzelnen Komponenten ist proportional zu den Eigenwerten der Komponenten. Die drei ersten Komponenten machen zusammen rund 98% des Informationsgehalts aus, sodass wir ohne merklichen Verlust auf die drei schwächsten Komponenten verzichten können.

	Eigenwert	Anteil
PC1	1.0352	68.82%
PC2	0.2959	19.67%
PC3	0.1403	9.33%
PC4	0.0203	1.35%
PC5	0.0094	0.62%
PC6	0.0031	0.21%
Total	1.5042	100.00%

Abb. 293: Eigenwerte nach Grösse sortiert.

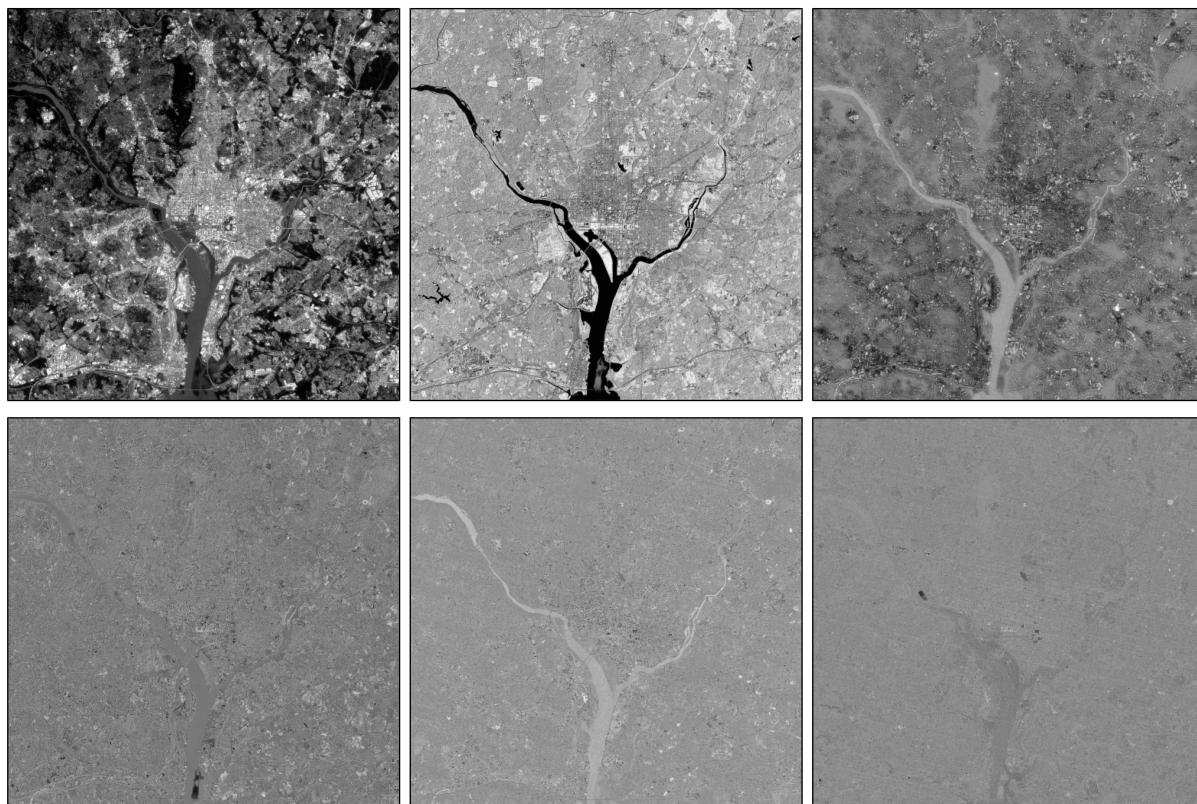


Abb. 294: V.o.l.n.u.r.: Alle Komponenten der Grösse (Eigenwert) nach sortiert. Quelle: [Gonzales08]

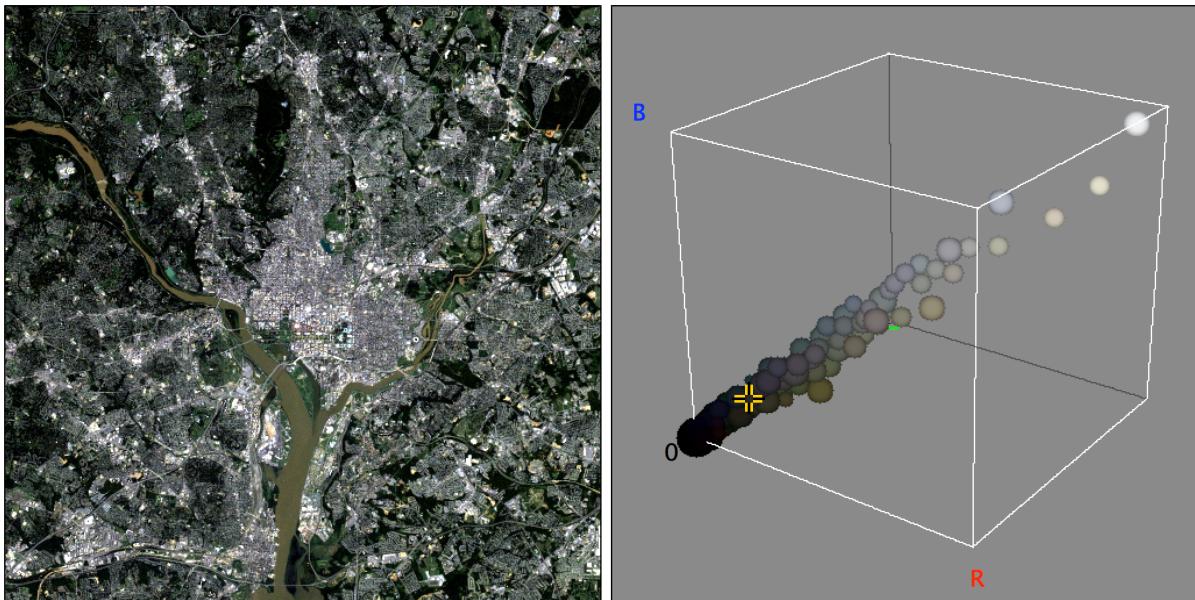


Abb. 295: Die ursprünglichen RGB-Kanäle waren mehr oder weniger korreliert, sodass die Kombination fast ein Graustufenbild ergibt.

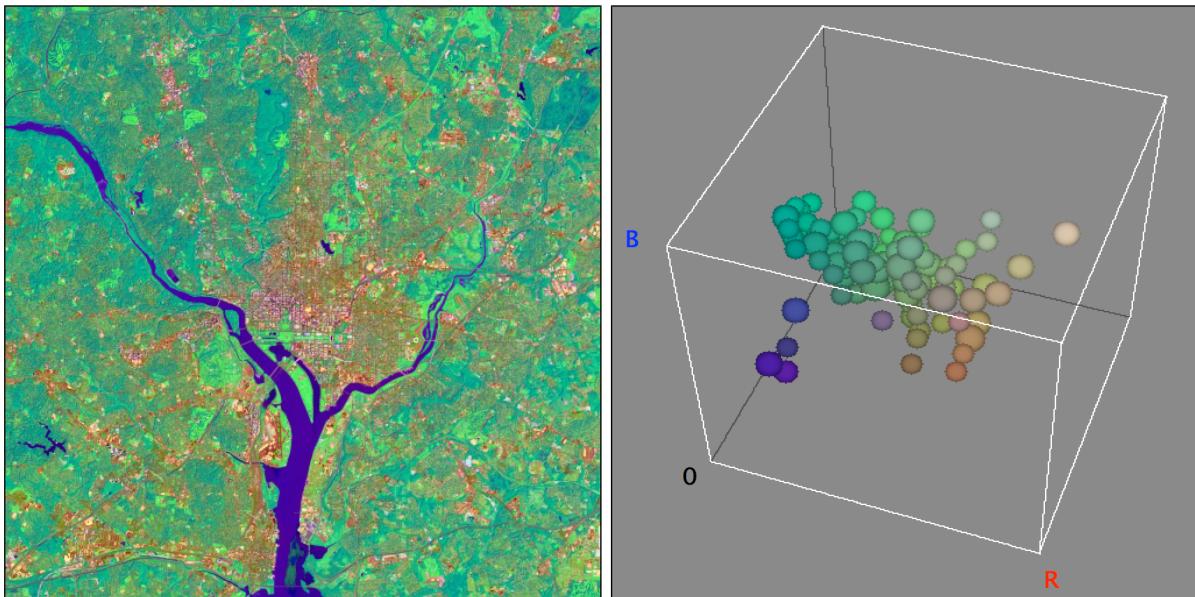


Abb. 296: Verwendet man die 3 Hauptkomponenten als RGB-Kanäle, so ergibt sich eine viel grössere Varianz im Farbraum. Vorsicht: Die 3 Hauptkomponenten haben nichts mehr mit den ursprünglichen RGB-Farben zu tun. Dass hier der Fluss bläulich, die Stadt bräunlich und das Land grünlich erscheint, ist reiner Zufall.

11.1.1.2 Gütekriterium für ein Merkmal

Sind die Stichproben zweier Klassen k_1 und k_2 von einem Merkmal m normal verteilt, d. h., ihre Häufigkeitsverteilung gleicht einer Gauss-Glockenkurve, so können wir mit dem Mittelwert μ und der Varianz σ^2 ein Gütemass q für die Trennbarkeit wie folgt berechnen:

$$q = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \quad \text{mit} \quad \mu_i = \frac{1}{N} \sum_i x_i \quad \text{und} \quad \sigma_i^2 = \frac{1}{N} \sum_i (x_i - \mu)^2$$

Je grösser dieser Wert q , umso besser sind die beiden Klassen getrennt und umso eindeutiger lässt sich ein Merkmal einer Klasse zuweisen. Dieser Wert ist umso grösser, je weiter die Mittelwerte beider Klassen entfernt sind und je kleiner die Summe beider Varianzen sind. Der Abstand der beiden Mittelwerte wird auch *Zwischenklassenvarianz* (*between-class variance*) und die Summe beider Varianzen die *Innerklassenvarianz* (*within-class variance*) genannt.

$$q = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} = \frac{\text{Zwischenklassenvarianz}}{\text{Innerklassenvarianz}} = \frac{S_B}{S_W} = \frac{\text{between-class Scatter}}{\text{within-class scatter}}$$

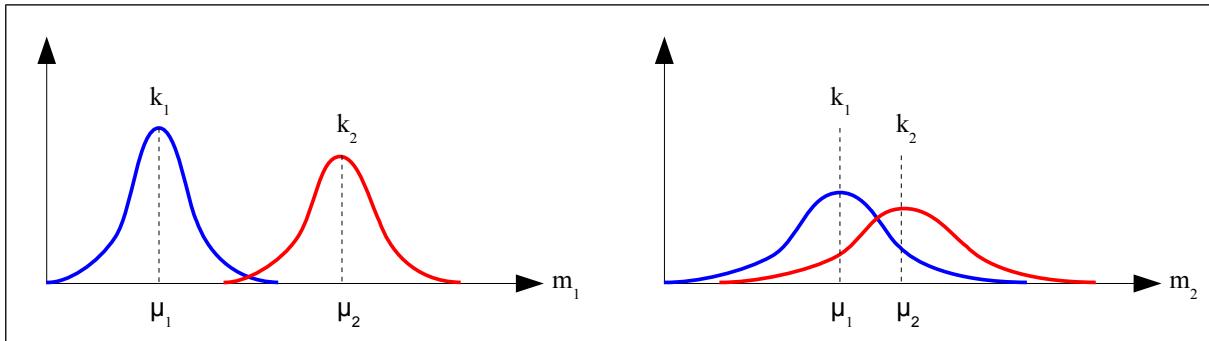


Abb. 297: Je weiter auseinander die Mittelwerte und je kleiner die Streuung zweier Verteilungen von Klassen, umso besser ist Trennbarkeit.

11.1.2 Bestimmung der Klassen

Man unterscheidet dabei folgende Herangehensweisen bei der Bestimmung der Klassen:

- **Überwachtes Lernen (supervised learning)**: Dabei werden durch a-priori Wissen eines Lehrers die Häufungen von Merkmalen (*Cluster*) im n-dimensionalen Merkmalsraum als gewünschte Klassen identifiziert. Für die Beispiele aus dem vorangegangenen Kapitel würde man also in der Trainingsphase von Auge die Anzahl und die Schwerpunkte der Cluster bestimmen.
- **Unüberwachtes Lernen (unsupervised learning, clustering)**: Dabei werden die Klassen in einem n-dimensionalen Merkmalsraum automatisch bestimmt. Die Anzahl der Klassen kann dabei vorgegeben oder unbestimmt sein. Diese Verfahren werden deshalb auch unter dem Begriff der *Cluster Analyse* zusammengefasst.
- **Bestärkendes Lernen (reinforcement learning)**: Dabei wird wiederum durch einen Lehrer ein sogenanntes *künstliches neuronales Netzwerk (KNN)* aufgebaut. Der Lehrer sagt jedoch nur, ob eine zufällige Zuweisung richtig oder falsch ist. Bei einer korrekten Zuweisung wird deren Wichtigkeit erhöht, wodurch sich Entscheidungswege durch ein anfänglich wirres Netzwerk vereinfachen und damit schnell zum Ziel führen.

11.1.2.1 k-Means Clustering

Der k-Means Clustering Algorithmus ist eine der bekanntesten Methoden des unüberwachten Lernens von Cluster-Zentren. Wir haben diese Methode bereits einmal als globale Schwellwertmethode kennengelernt (s. 8.1.1.2).

Dabei wird die Anzahl Klassen (k) vorausgesetzt, für die das Cluster-Zentrum iterativ bestimmt wird. Der Algorithmus geht wie folgt:

1. Bestimme k zufällige Cluster-Zentren $c_1 - c_k$.
2. Weise alle Merkmale dem nächsten Cluster-Zentrum zu. Für die Distanzmessung können verschiedene Metriken zum Einsatz kommen (s. Kapitel 9.6.1):
 - Manhattan Distanz: $|c_x - m_x| + |c_y - m_y|$
 - Euklidische Distanz: $\sqrt{(c_x - m_x)^2 + (c_y - m_y)^2}$
 - Quadratische euklidische Distanz: $(c_x - m_x)^2 + (c_y - m_y)^2$
 - Schachbrett Distanz: $\max(|c_x - m_x|, |c_y - m_y|)$

- Kosinus Distanz: $1 - \frac{\vec{c} \cdot \vec{m}}{|\vec{c}| |\vec{m}|}$ (entspricht 1 - Kosinus vom Winkel zw. den normierten Vektoren zu c und m).
- Berechne alle Cluster-Zentren neu aus dem Mittelwert aller im Zugewiesenen Merkmale.
 - Wiederhole Schritt 2 und 3 solange bis sich keine Zuweisung mehr ändert.

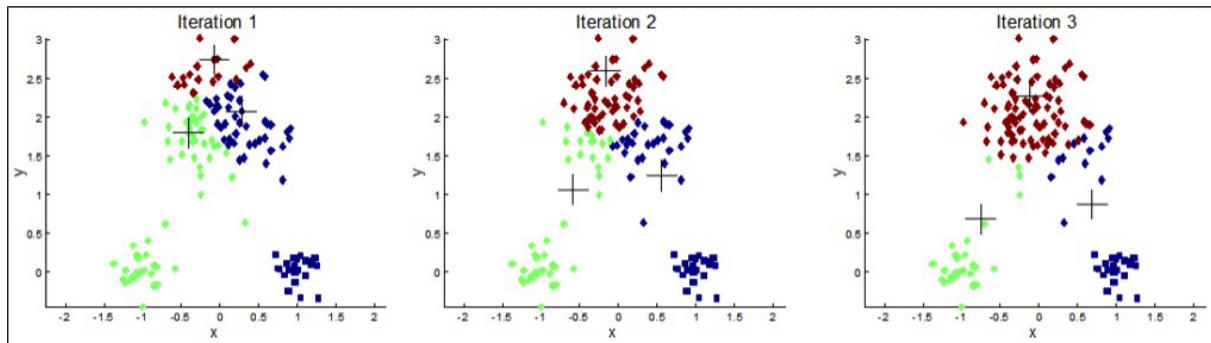


Abb. 298: Iteration 1 - 6 des k-Means-Algorithmus.

k-Means optimiert die quadratischen Abweichungen von einem Mittelwert. Der Algorithmus muss nicht die beste mögliche Lösung finden. Die gefundenen Zentren hängen von den gewählten Startpunkten ab. Der einfachste Ansatz ist, den Algorithmus mehrmals hintereinander mit verschiedenen Startzentren zu starten und dann die beste Lösung mit den kleinsten Varianzen in den Clustern auszuwählen.

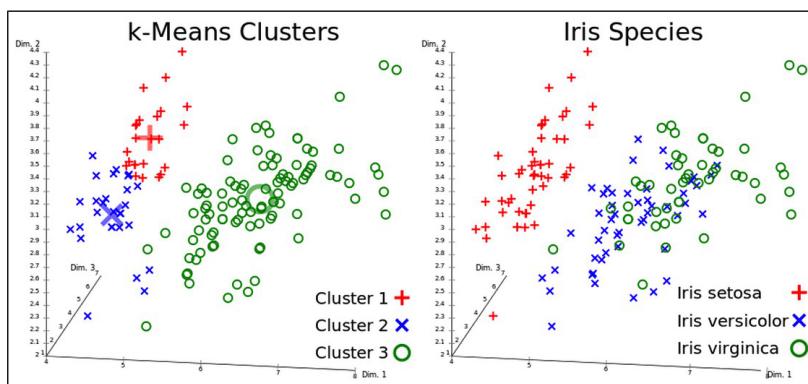


Abb. 299: Unüberwachtes Lernen der Cluster-Zentren mit k-Means muss nicht zur richtigen Klassenbestimmung führen, wie das Beispiel des Iris Datensets zeigt. Eine falsche Merkmalsauswahl wie hier mit den schlecht separierbaren Sepal-Längen und Breiten führt zu einer falschen Cluster-Bildung. (Quelle: Wikipedia)

Anstatt des Mittelwerts (*mean*) kann für die Zentrumsberechnung auch der *Median* verwendet werden. In diesem Fall heisst der Algorithmus *k-Mediods* und ist damit weniger anfällig auf Ausreisser, wie z. B. die beiden blauen Punkte im Bild 298 unten rechts. Wegen der Sortierung zur Medianbestimmung ist diese Variante aber auch langsamer.

Eine Erweiterung des k-Means-Algorithmus ist der *ISODATA-Algorithmus*. ISODATA steht für *Iterative Self Organizing Data Analysis*. Nach dem Schritt 3 des k-Means-Algorithmus werden die Cluster überprüft. Zwei Cluster mit einem zu geringen Abstand werden zusammengelegt oder ein Cluster mit einer zu grossen Standardabweichung wird in zwei Cluster aufgeteilt. Danach wiederholt sich wieder alles bis keine neue Merkmalszuweisung oder Cluster-Aufteilung oder Zusammenlegung mehr stattfindet.

In **ImageJ** steht der k-Means-Algorithmus in der Plugin-Sammlung [ij-Plugin-Toolkit](#) zur Verfügung.





Beispiel mit Matlab

In **Matlab** steht der k-Means-Algorithmus mit der Funktion [kmeans](#) zur Verfügung. Das nachfolgende Skript [kmeansTest.m](#) erstellt 200 Zufallspunkte und unterteilt sie mit k-Means Clustering in 3 Cluster.

```
%Seed random generator to have allways the same data
rng(0);

%Generate random data
X = [randn(200,2) + ones(200,2);
      randn(200,2) - ones(200,2)];

%Do the k-means clustering for 3 clusters
[idx,ctrs] = kmeans(X, 3, ...
    'Distance', 'sqEuclidean', ...
    'Replicates', 5);

%Plot the clusters in RGB
figure('Color',[1 1 1]);
plot(X(idx==1,1),X(idx==1,2), 'r.', 'MarkerSize', 12), hold on
plot(X(idx==2,1),X(idx==2,2), 'g.', 'MarkerSize', 12)
plot(X(idx==3,1),X(idx==3,2), 'b.', 'MarkerSize', 12)

%Plot the cluster centers
plot(ctrs(:,1), ctrs(:,2), 'kx', 'MarkerSize',12)
plot(ctrs(:,1), ctrs(:,2), 'ko', 'MarkerSize',12)
```

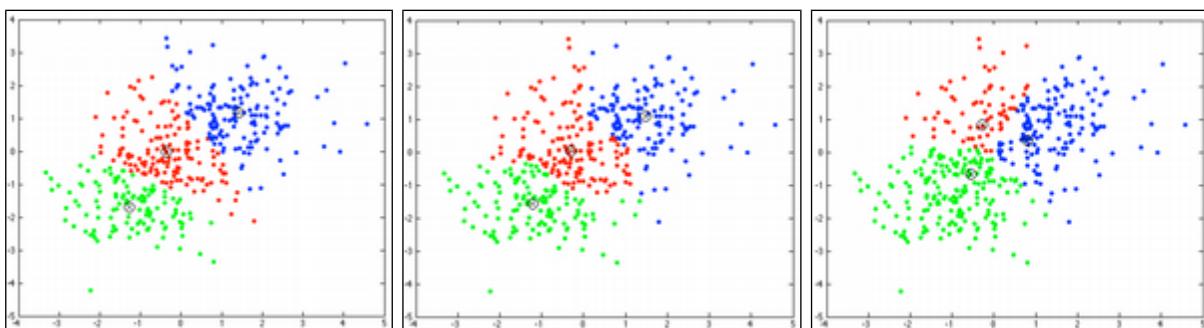


Abb. 300: Links: k-Means Clustering mit quadratisch-euklidischer Distanzmetrik
 Mitte: k-Means Clustering mit Cityblock Distanzmetrik
 Rechts: k-Means Clustering mit 'Cosine'-Metrik bei der Winkel des Punktes zum Gesamtschwerpunkt berücksichtigt wird. Es entsteht dabei eine gleichmässige radiale Kuchenunterteilung.

11.2 Klassifikatoren

Nachdem die Klassen durch überwachtes oder unüberwachtes Training bestimmt worden sind, müssen wir die Merkmale noch den verschiedenen Klassen zuweisen, also die eigentliche Klassifizierung vornehmen. Die verschiedenen Klassifikatoren können in die Gruppen der *Distanzklassifikatoren* oder *statistischen Klassifikatoren* eingeteilt werden.

11.2.1 Distanzklassifikatoren

11.2.1.1 Minimale Distanz zum Cluster-Zentrum

Einer der einfachsten und intuitivsten Zuweisungsalgorithmen besteht aus der minimalsten Distanz zum jeweiligen Cluster-Zentrum (*Minimal Distance Classifier*). Diese Zuordnung haben wir ebenfalls im k-Means-Algorithmus verwendet. Für die Distanzmessung wird meist die euklidische oder quadratisch-euklidische Distanz verwendet.

Für den Fall eines zweidimensionalen Merkmalsraums, wie im nachfolgenden Beispiel, ist leicht zu erkennen, dass sich die Entscheidung, ob ein Kandidat zur einen oder anderen Klasse gehört, entlang der Mittelsenkrechten der Verbindungsgeraden zwischen den Cluster-

Zentren abspielt. Diese Trennlinie ist im 2D-Raum eine Geradengleichung und wird *Diskriminantenfunktion* genannt. Im 3D-Raum beschreibt diese Funktion eine Ebene. Für noch höher dimensionierte Merkmalsräume wird diese Trennebene *Hyper-Ebene* genannt.

Eine Ergänzung des *Minimal Distance Classifier* ist die Erweiterung um die *Zurückweisungsklasse*. Dieser Klasse werden alle Kandidaten zugewiesen, die ausserhalb der maximalen Radien der Cluster liegen.

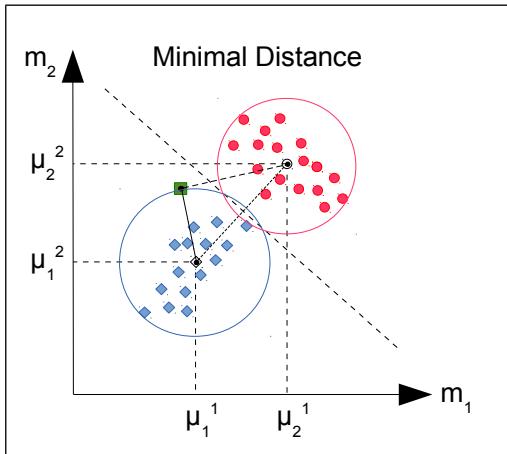


Abb. 301: Zuweisung zum nächsten Cluster-Zentrum

Voronoi-Diagramm und Delaunay-Triangulation

Wenn man für mehrere Cluster-Zentren die Mittelsenkrechten zu den jeweils nächsten Nachbarn zeichnet, so entsteht dabei das sogenannte *Voronoi-Diagramm*. Die Region um ein Cluster-Zentrum, begrenzt durch die Mittelsenkrechten, wird Voronoi-Region genannt. Die Verbindungslinien zwischen den jeweils nächsten Cluster-Zentren ergeben ein Dreiecksnetz, das Delaunay-Triangulation genannt wird. Man sagt auch, dass das Voronoi-Diagramm und die Delaunay-Triangulation *dual* zueinanderstehen.

In der Delaunay-Triangulation erfüllen alle Dreiecke des Netzes die sogenannte Umkreisbedingung: Der Umkreis eines Dreiecks des Netzes darf keine weiteren Punkte des Dreiecksnetzes enthalten. Dadurch weisen die Dreiecke des Netzes möglichst grosse Innenwinkel auf. Die Delaunay-Triangulation ist dann nicht eindeutig, wenn auf einem Umkreis mehr als drei Punkte liegen, d. h., man kann sich beliebig aussuchen, welche drei Punkte man zu einem Dreieck verbindet.

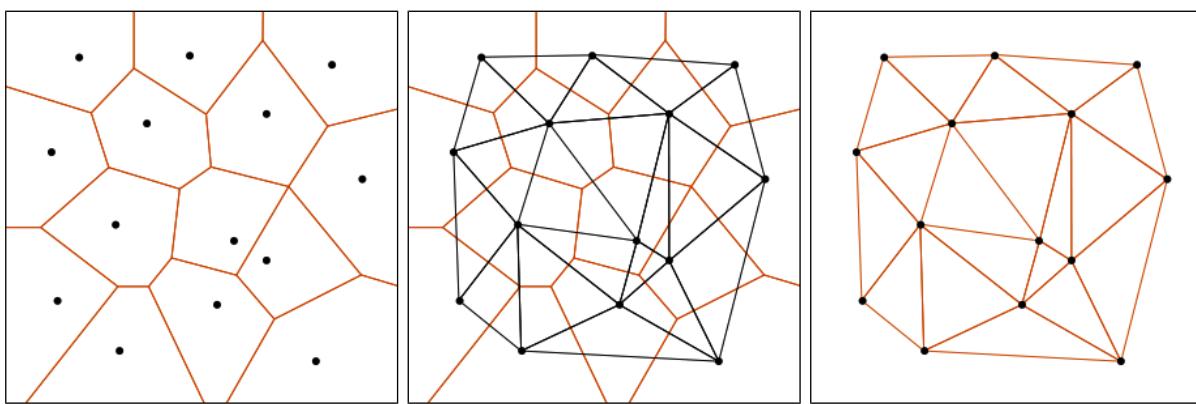


Abb. 302: Links: Voronoi-Diagramm aus den braunen Mittelsenkrechten, Mitte: Delaunay-Triangulation über Voronoi-Diagramm und rechts: Delaunay-Triangulation als Dreiecksnetz. (Quelle: Wikipedia)



Beispiel mit Matlab

Das nachfolgende Skript [kmeansImageTest.m](#) entstammt ursprünglich der [Matlab-Dokumentation](#) und segmentiert ein eingefärbtes Zellbild in 3 Farbklassen, die mit k-Means Clustering aus dem ab-Teil des L*a*b-Farbraum berechnet werden. Der L*a*b-Farbraum, den wir bis jetzt nicht verwendet haben, teilt den Farbraum ähnlich wie YUV oder YCbCr in einen Helligkeitskanal L und zwei Farbkanäle a und b auf.

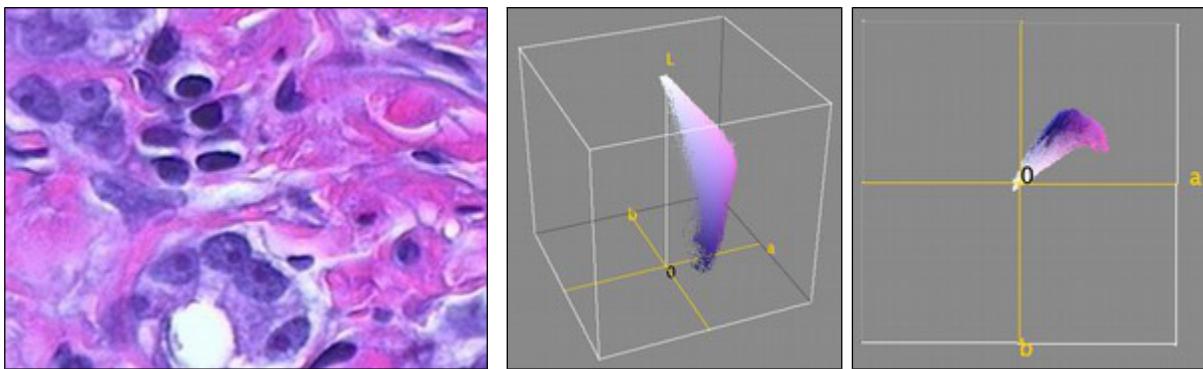


Abb. 303: Das eingefärbte Zellbild links und das 3D-Farbhistogramm davon im L*a*b*-Farbraum in der Mitte. Rechts ist die ab-Ebene (von unten) dargestellt in der mit k-Means Clustering 3 Farbklassen berechnet werden. Die Histogramme wurden mit den *3D-Color Inspector* von Fiji erstellt.

```
%Read & display image
I = imread('../Images/hestain.png');
imshow(I), title('hestain image');

%Convert Image from RGB Color Space to Lab Color Space
cform = makecform('srgb2lab');
labI = applycform(I,cform);

%Get the ab channels into a 2 columns matrix to have a as x & b as y-coord.
ab = double(labI(:,:,2:3));
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,2);

%Dose k-means clustering for 3 classes. Repeat it 3 times
nColors = 3;
[iClust cClust] = kmeans(ab, nColors, 'distance', 'sqEuclidean', 'Replicates', 3);

%Reshape the cluster indexes back to image size & display them as an image
pixel_labels = reshape(iClust, nrows, ncols);
imshow(pixel_labels,[]), title('image labeled by cluster index');

%Create a cell array of empty images
segmented_images = cell(1,3);

%Create a 3 channel image with by replicating the pixel labels
rgb_label = repmat(pixel_labels,[1 1 3]);

for k = 1:nColors
    segment = I;                      %copy image I into segment
    segment(rgb_label ~= k) = 0;        %set other segments to black
    segmented_images{k} = segment;      %copy segment to cell array
end

%Display segmentend images
figure, imshow(segmented_images{1}), title('objects in cluster 1');
figure, imshow(segmented_images{2}), title('objects in cluster 2');
figure, imshow(segmented_images{3}), title('objects in cluster 3');
```

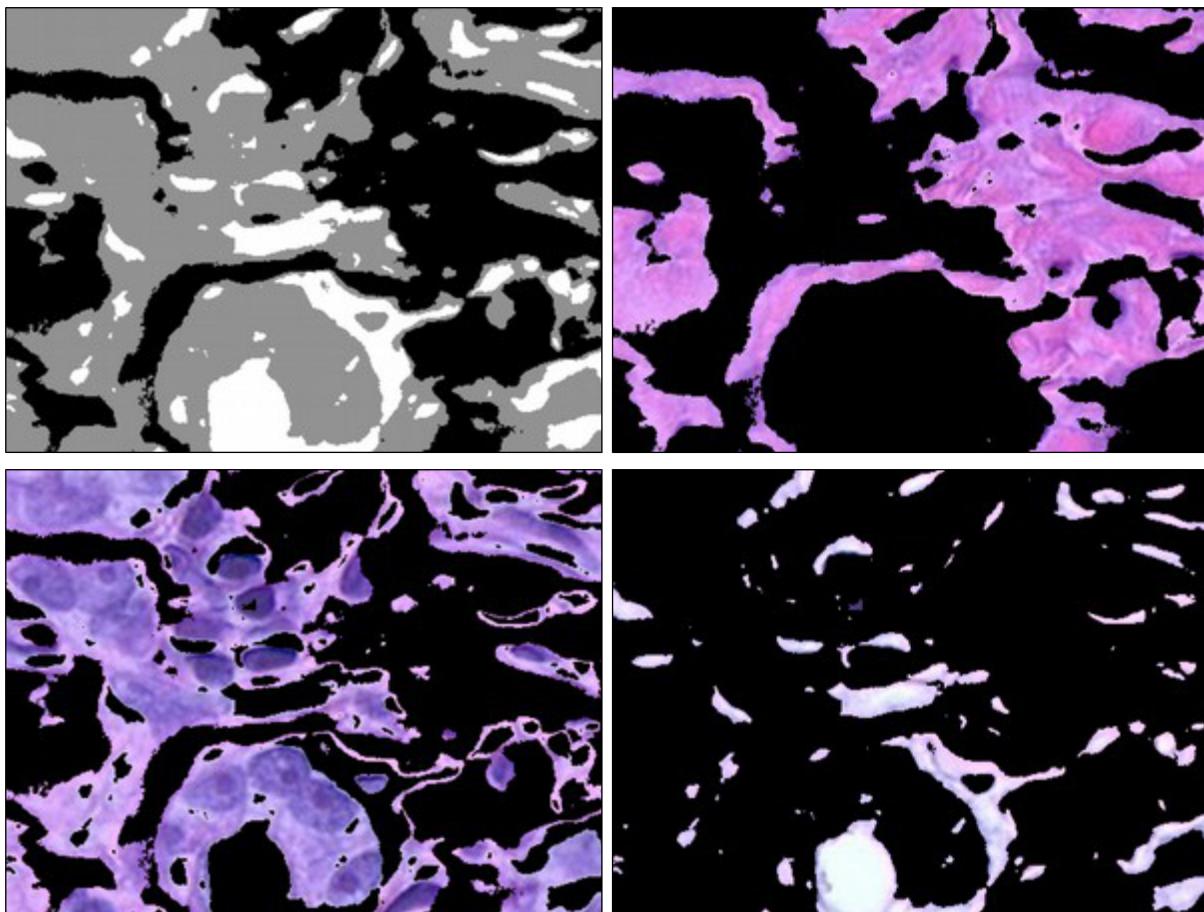


Abb. 304: Oben links: 3 Klassen nach k-Means Clustering als Graustufendarstellung. Danach die 3 Farbklassen einzeln dargestellt.

11.2.1.2 Minimale Distanz zu k Nachbarn

Eine Variante des *Minimal Distance Classifier* ist die Zuweisung zum nächsten Nachbarn (*Nearest Neighbor Classifier (NN-Classifier)*) oder zum Mittelwert der k nächsten Nachbarn (*k -Nearest Neighbors (k NN-Classifier)*). Sobald k gleich der Anzahl der Klassen-Elemente ist, haben wir wieder den Klassifikator zum Schwerpunkt einer Klasse. Der *k NN-Algorithmus* ist zwar einfach aber rechnerisch aufwendig und speicherintensiv. Um nicht die Distanz zu allen Nachbarn berechnen zu müssen, werden in der Regel Raumunterteilungsstrukturen wie z.B. *kd-Trees* aufgebaut.

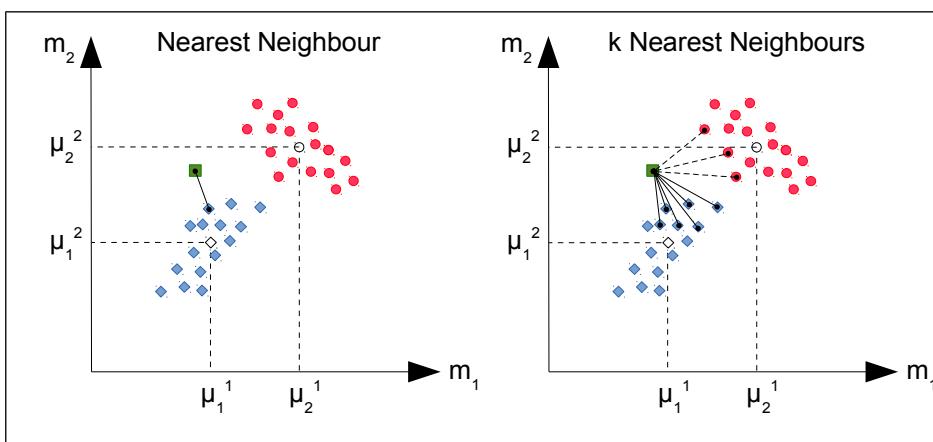


Abb. 305: Links: Zuweisung zum nächsten Nachbarn und rechts: Zuweisung zu den k -nächsten Nachbarn.

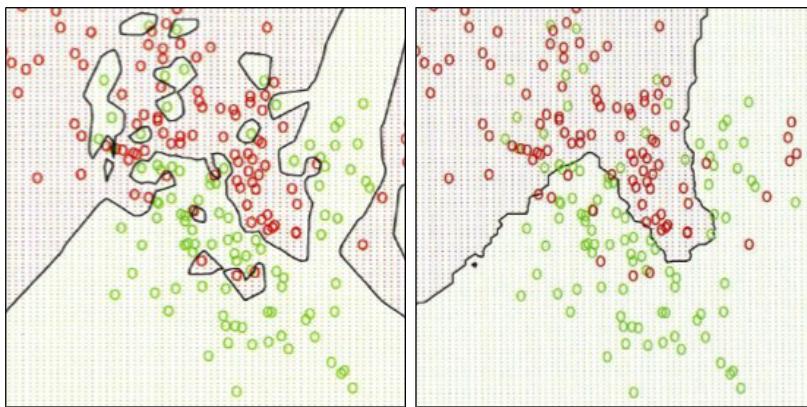


Abb. 306: Der kNN-Klassifikator erlaubt nicht lineare Klassengrenzen. Links: $k=1$, rechts: $k=15$



11.2.1.2.1 Beispiel: knnsearch in Matlab

In Matlab steht der kNN-Klassifikator mit der Funktion [knnsearch](#) zur Verfügung. Im nachfolgenden Beispiel ([knnsearchTest1.m](#)) wird zuerst das *FisherIrist Dataset* geladen und angezeigt. Für einen neu zu klassifizierenden Punkt mit den Koordinaten [5.0, 1.45] werden mit [knnsearch](#) die 10 nächsten Nachbarn gesucht und markiert.

```
%loads the fisheriris dataset into matrix meas and vector species
load fisheriris

%get the 3rd & 4th column petalum length & width
x = meas(:,3:4);

%Plots the data by species
figure('Color',[1 1 1]);
gscatter(x(:,1),x(:,2), species);
set(legend,'location','best')

%Set & draw a new point to be classified
newpoint = [5.0 1.45];
line(newpoint(1),newpoint(2), ...
    'marker','x', 'color','k', 'markersize',10, 'linewidth',2);

%Search & draw the 10 nearest neighbors
[n,d] = knnsearch(x,newpoint,'k',10);
line(x(n,1),x(n,2), ...
    'color',[.5 .5 .5], 'marker','o', 'linestyle','none', 'markersize',10);

%Print the frequency table
tabulate(species(n))
```

Das Resultat der Suche ergibt, dass 8 von 10 Nachbarn und damit auch der neue Punkt zur Spezies der *Versicolor Iris* gehören.

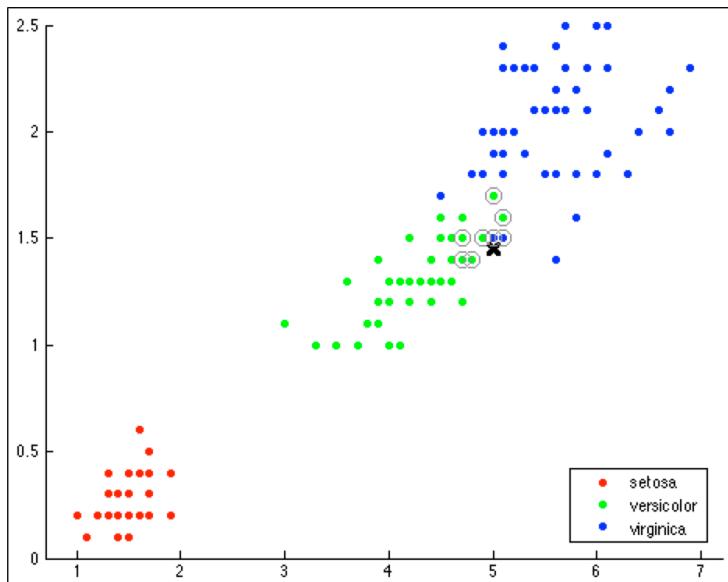


Abb. 307: Zuweisung des neuen Punktes (Kreuz) zur Klasse *Versicolor*, weil 8 der 10 nächsten Nachbaren ebenfalls dazugehören.



11.2.1.2.2 Beispiel: Gesichtserkennung im Hauptkomponentenraum

Wir konnten die Hauptkomponentenanalyse, die wir im Kapitel 7.4 kennengelernt haben, schon an einigen Stellen in der Bildanalyse verwenden. Wir haben damit die Ausrichtung und Ausdehnung einer Region bestimmt (s. Kapitel 10.2.2.1) oder haben die Dimensionalität von Merkmalen reduziert (s. Kapitel 11.1.1.1). Wir können die PCA jedoch auch für die Klassifikation einsetzen, indem wir eine minimale Distanz im Hauptkomponentenraum bestimmen. Wir erinnern uns, dass die PCA eine Transformation in einen Raum ist, bei welchem die Koordinatenachsen den nach Varianz sortierten Hauptkomponenten entsprechen. Bis jetzt haben wir max. 6 Dimensionen mit der PCA verarbeitet.

In diesem Beispiel werden wir 400 Bilder von 40 Gesichtern in den Hauptkomponentenraum transformieren. Jedes Gesicht stellt dabei eine Dimension dar, sodass unser Hauptkomponentenraum ebenfalls 400 Dimensionen aufweist.

Das Ziel danach ist es, ein Gesicht im Hauptkomponentenraum zu suchen. Natürlich finden wir damit das identische Gesicht. Für das bräuchten wir aber nicht die aufwendige Transformation. Dies könnten wir ja auch mit einem arithmetischen Bildvergleich erreichen. Wenn das Verfahren etwas taugt, dann müsste man damit aber auch zuerst alle anderen 9 Bilder von derselben Person zuerst finden, bevor wir ein Bild einer anderen Person detektieren. Die Gesichtsbilder entstammen einer [Bildersammlung von ATT](#), die für die Entwicklung von Gesichtserkennungsalgorithmen zur Verfügung gestellt wird. Eine Liste von weiteren Gesichterdatenbanken finden Sie unter [face-rec.org](#).

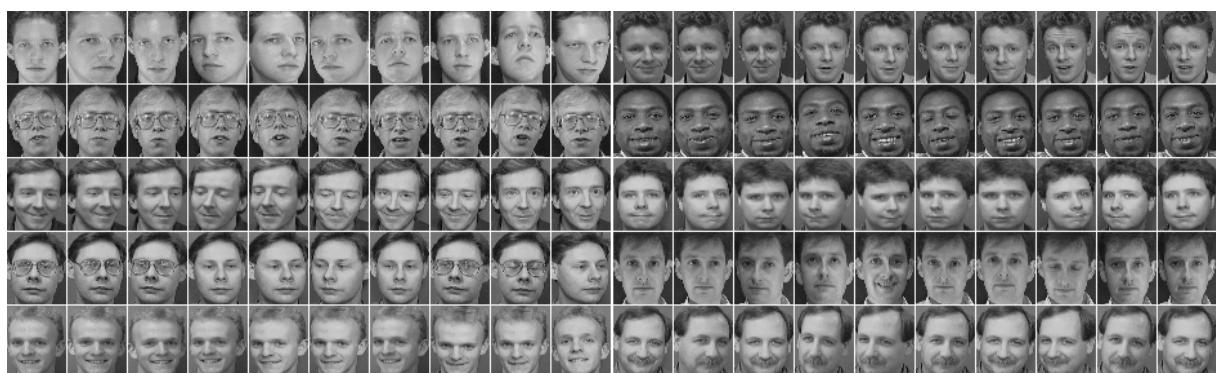


Abb. 308: Die ersten 100 Bilder von 10 Personen aus der ATT-Gesichtersammlung.

Schritt 1: Bilder laden und jedes Bild in eine Spalte einer Matrix umwandeln:

Sie finden alle nachfolgenden Teilschritte im Matlab-Skript [PCA4_Eigenfaces.m](#). Zuerst werden alle Bilder, die sich in Unterordnern befinden, geladen und als Spaltenvektor in eine Große Matrix `face` geschrieben. Die Matrix hat am Schluss die Größe von 400 Spalten und 10'304 Zeilen (da ein Bild die Größe 92 x 112 = 10'304 Pixel hat).

```

k = 0;
for i=1:1:40
    for j=1:1:10
        filename = sprintf('../images/att_faces/s%d/%d.pgm',i,j);
        image_data = imread(filename);
        k = k + 1;
        faces(:,k) = image_data(:);
    end;
end;
nImages = k;                                %total number of images
imsize = size(image_data);                   %size of image (they all should have the same
size)
nPixels = imsiz(1)*imsiz(2);                %number of pixels in image
faces = double(faces)/255;                  %convert to double and normalize

```

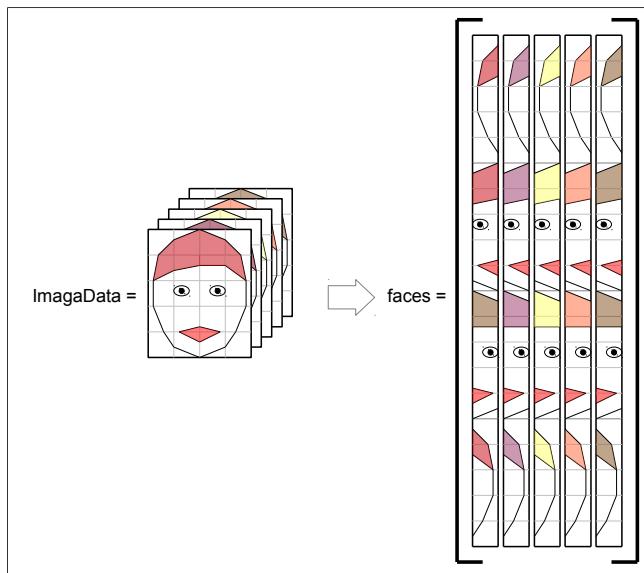


Abb. 309: Eine Serie von gleichgrossen Bildern wird in eine Matrix umgewandelt, wo ein Bild eine Spalte ist:

Schritt 2: Mittelwert aller Gesichter berechnen und subtrahieren:

Mit der Funktion `mean` erhalten wir den Mittelwert der Matrix `face` als Spaltenvektor `mn`, den wir danach von dieser subtrahieren, um alle Grauwerte um den Nullpunkt zu verschieben. Als Bild visualisiert, ergibt sich daraus ein stark verschwommenes Gesicht. Dies ist klar, den die einzelnen Gesichter sind kaum aufeinander registriert.

```

mn = mean(faces, 2);
for i=1:nImages
    faces(:,i) = faces(:,i)-mn;           % subtract the mean
end;
figure('Color',[1 1 1]);
imshow(reshape(mn, imsiz)); title('mean face');

```



Schritt 3: Eigenvektoren und Eigenwerte berechnen

Für die Berechnung der Eigenvektoren und Eigenwerte finden Sie zwei Methoden:

- Die auskommentierte Variante verwendet die Matlab-Funktion `princomp` und produziert zwei Matrizen der Größe 10304 x 10304.
- Die aktive Variante erzeugt eine kleinere Kovarianzmatrix mit `faces'*faces` anstatt `faces*faces'` und ist rund 20-mal schneller. Diese Version entstammt aus dem ur-

sprünglichen Paper der Eigenface-Erfinder [Turk und Pentland 1991](#) [[Turk91](#)]. Dieses Paper ist auf Rang 6 der am [meisten zitierten Paper](#) der Computer Vision.

Visualisiert man die ersten 30 Spalten der Eigenvektor-Matrix, so sehen wir die nach Varianz sortierten Hauptkomponenten aller Gesichter.

```
%% Step 3: Method 1 using princomp from Matlab
% tic;
% [eigvec, score, eigval] = princomp(faces'); %eigvec: <10304x10304>
% eigvec = eigvec(:,1:nImages); %eigvec: Cut down to <10304x400>
% toc;

%% Step 3: Method 2: Create covariance matrix faster by using
% Turk and Pentland's trick to get the eigenvectors of faces'*faces' from
% the eigenvectors of faces'*faces
tic;
C = faces'*faces;
[eigvec,eigval] = eig(C);
eigvec = faces * eigvec; % Convert eigenvecs back as if they came
from A'*A; % Normalize eigenvectors
eigvec = eigvec / (sqrt(abs(eigval))); % Normalize eigenvectors

% eigvec & eigval are in fact sorted but in the wrong order
eigval = diag(eigval); % Get the eigenvalue from the diagonal
eigval = eigval / nImages; % Normalize eigenvalues
[eigval, indices] = sort(eigval, 'descend'); % Sort the eigenvalues
eigvec = eigvec(:, indices); % Sort the eigenvectors accordingly
toc;

% Display the 30 first eigenfaces
figure('Color',[1 1 1]);
for n = 1:30
    subplot(3, 10, n);
    eigvecImg = reshape(eigvec(:,n), imsize); % Reshape vector to image
    imshow(eigvecImg, []); % Show eigenface image with max. contrast
end

% Display the eigenvalues
normEigval = eigval / sum(eigval);
figure('Color',[1 1 1]);
plot(cumsum(normEigval));
xlabel('No. of eigenvectors'), ylabel('Variance accounted for');
xlim([1 200]), ylim([0 1]), grid on;
```



Abb. 310: Die ersten 30 nach Varianz sortierten Hauptkomponenten. Diese Gesichter werden auch *Eigengesichter* oder *Eigenfaces* genannt.

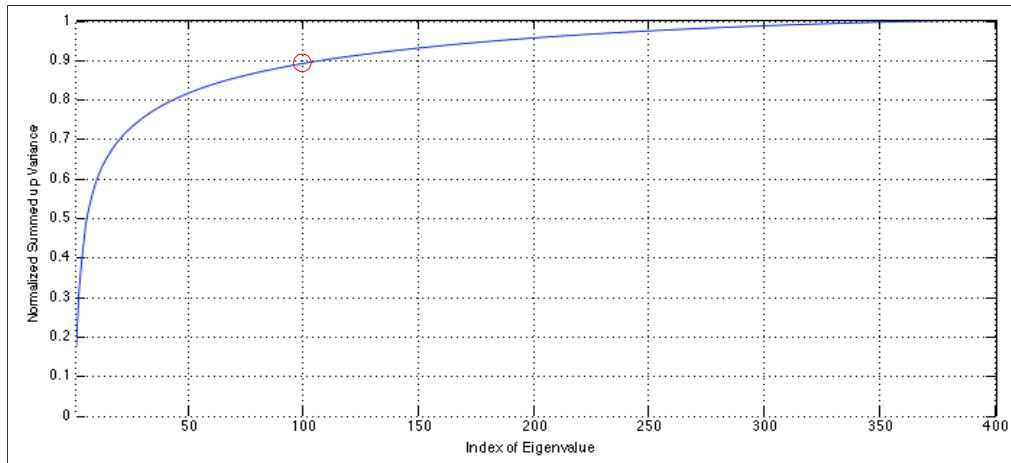


Abb. 311: Die aufsummierten, normierten Eigenwerte zeigen, dass wir mit den ersten 100 Hauptkomponenten rund 90% der Gesamtvarianz abdecken.

Schritt 4: Transformation in den Hauptkomponentenraum

Die Transformation in den Hauptkomponentenraum ist wie in allen PCA-Beispielen nichts anderes als eine lineare Transformation, bestehend aus einer Multiplikation der Eigenvektor-Matrix mit der Gesichtermatrix. Damit es nicht vergessen geht, wiederhole ich nochmals, dass `faces` ein vierhundertdimensionaler Datenraum ist, wo jede Dimension ein Gesicht ist. Die neue Matrix `faces2` hat nun ebenfalls 400 Dimensionen mit den Anteilen an den 400 Hauptkomponenten.

$$\text{faces2}_{400 \times 400} = \text{eigvec}' \cdot \text{faces} = \begin{bmatrix} e_{1,1}, e_{1,2}, \dots, e_{1,10304} \\ e_{2,1}, e_{2,2}, \dots, e_{2,10304} \\ \dots, \dots, \dots \\ e_{400,1}, e_{400,2}, \dots, e_{400,10304} \end{bmatrix} \cdot \begin{bmatrix} f_{1,1}, f_{1,2}, \dots, f_{1,400} \\ f_{2,1}, f_{2,2}, \dots, f_{2,400} \\ \dots, \dots, \dots \\ f_{10304,1}, f_{10304,2}, \dots, f_{10304,400} \end{bmatrix}$$

```
% ######
faces2 = eigvec' * faces;
######
```

Schritt 5: Rücktransformation aus den Hauptkomponentenraum

Um zu beweisen, dass sich die Transformation umkehren lässt, wollen wir ein Gesicht mit dem Index 78 wieder rekonstruieren. Wir tun dies schlicht mit der Inversen der Vorwärtmatrix. Um den Effekt der Anzahl Hauptkomponenten zu sehen, invertieren wir einmal mit nur einem Eigenvektor, einmal mit 10, 50, 100, 200 und einmal mit allen 400 Eigenvektoren. Für die vollständige Rücktransformation sieht die Matrixmultiplikation wie folgt aus:

$$\text{faces}_{10304 \times 400} = \text{eigvec}_{400} \cdot \text{faces2} = \begin{bmatrix} e_{1,1}, e_{1,2}, \dots, e_{1,400} \\ e_{2,1}, e_{2,2}, \dots, e_{2,400} \\ \dots, \dots, \dots \\ e_{10304,1}, e_{10304,2}, \dots, e_{10304,400} \end{bmatrix} \cdot \begin{bmatrix} f_{1,78} \\ f_{2,78} \\ \dots \\ f_{10304,78} \end{bmatrix}$$

Um den Fehler der Rekonstruktion zu bestimmen, berechnen wir die absolute Differenz pro Pixel. Für die Visualisierung dürfen wir nicht vergessen, den Mittelwert `mn` wieder hinzuzuaddieren.

```
i = 78; %index of face to be reconstructed
eigvec001 = eigvec; eigvec001(:, 2:end) = 0; % keep the biggest PC
eigvec010 = eigvec; eigvec010(:, 11:end) = 0; % keep the 10 biggest PC
eigvec050 = eigvec; eigvec050(:, 51:end) = 0; % keep the 50 biggest PC
eigvec100 = eigvec; eigvec100(:, 101:end) = 0; % keep the 80 biggest PC
eigvec200 = eigvec; eigvec200(:, 201:end) = 0; % keep the 80 biggest PC

faces001 = eigvec001 * faces2(:,i);
faces010 = eigvec010 * faces2(:,i);
faces040 = eigvec050 * faces2(:,i);
```

```

faces100 = eigvec100 * faces2(:,i);
faces200 = eigvec200 * faces2(:,i);
faces400 = eigvec    * faces2(:,i);

diff001 = abs(faces001 - faces(:,i));
...
diff400 = abs(faces400 - faces(:,i));

diffSum001 = sprintf('delta per px: %3.2e',sum(sum(diff001))/nPixels);
...
diffSum400 = sprintf('delta per px: %3.2e',sum(sum(diff400))/nPixels);

figure('Color',[1 1 1]);
subplot(2,5, 1); imshow(reshape(mn+faces001, imsize)); title('Reconstr. w. 1 PC');
subplot(2,5, 6); imshow(reshape(diff001, imsize), []); title(diffSum001);
...
subplot(2,5, 5); imshow(reshape(mn+faces400, imsize)); title('Reconstr. w. 400 PC');
subplot(2,5,10); imshow(reshape(diff400, imsize), []); title(diffSum400);

```

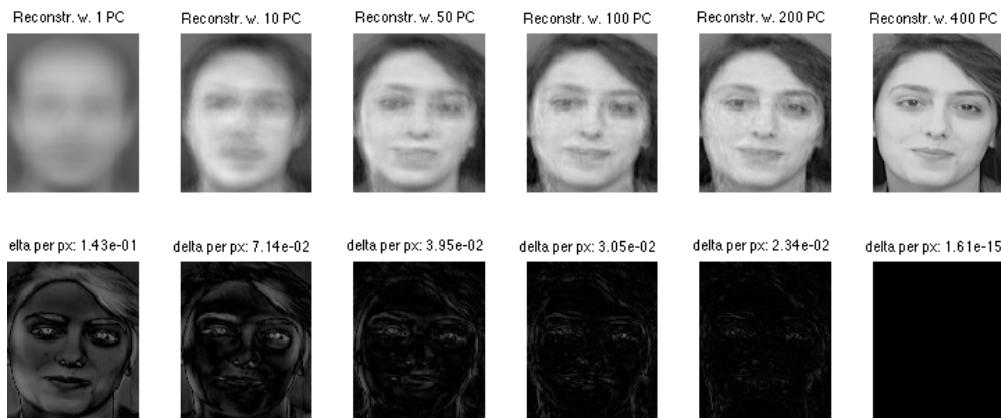


Abb. 312: Oben: Rekonstruktionen mit 1, 10, 40, 80 und 400 Hauptkomponenten. Unten: Differenzbild zum Original. Es ist nicht erstaunlich, dass die Rekonstruktion mit einer Hauptkomponente zum Mittelwertsgesicht führt. Mehr erstaunlich ist jedoch, dass doch sehr viele Hauptkomponenten notwendig sind, um ein relativ störungsfreies Gesicht zu rekonstruieren. Dies liegt daran, dass wir eine sehr sensible Gesichterwahrnehmung besitzen.

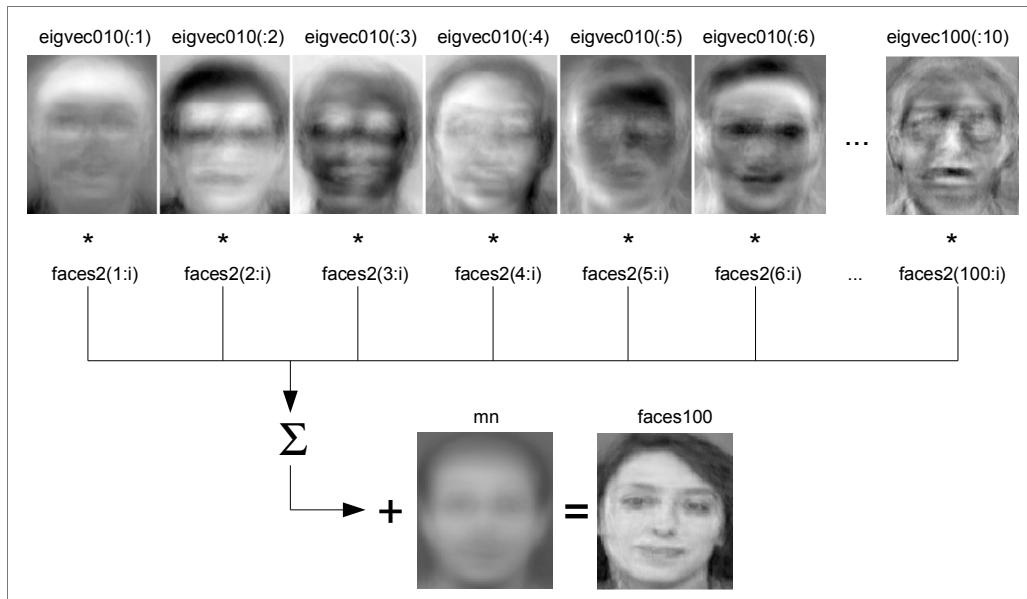


Abb. 313: Rekonstruktion eines Gesichts als Summe von 100 gewichteten Eigenfaces (=Eigenvektoren) und dem abschliessend addierten Mittelwertsgesicht.

Schritt 6: Suche nach k nächsten Gesichtern im Hauptkomponentenraum

Im letzten Schritt, der eigentlichen Klassifikation, wollen wir mit einem Suchbild nach einer Person in der Gesichterdatenbank suchen. Im Beispiel suchen wir das Bild mit dem

Index 78. Das Ziel bei der Personenidentifikation ist es, mit einem Bild alle anderen Bilder derselben Person zu finden.

Wir transformieren dazu das Suchbild in den Hauptkomponentenraum und erhalten dadurch 400 Gewichte der Hauptkomponenten. In diesem 400 Dimensionen umfassenden Raum berechnen wir die euklidische Distanz zu allen anderen Gesichtern im Hauptkomponentenraum. Diese 400 Distanzen sortieren wir und zeigen die 14 Gesichter mit den kürzesten Distanzen an.

```
i = 78; %index of face to be searched
searchFace = reshape(mn+faces(:,i), imsize); %reshape from vector image
search = eigvec' * (searchFace(:) - mn); %transform into PC space

% Calculate the squared euclidean distances to all faces in the PC space
% We use the dot product to square the vector difference.
for i=1:nImages
    distPC(i) = dot(faces2(:,i)-search, faces2(:,i)-search);
end;

%% Sort the distances and show the nearest 16 faces
[sortedDistPC, sortIndex] = sort(distPC); % sort distances
figure('Color',[1 1 1]);
for i=1:14
    subplot(2,7,i);
    imshow(reshape(mn+faces(:,sortIndex(i)), imsize));
    title(sprintf('Dist=%2.2f', sortedDistPC(i)));
end;
```



Abb. 314: Zum Suchbild berechnen wir logischerweise die Distanz 0. Alle neun anderen Bilder derselben Person sind jedoch auf den nächsten Rängen, noch bevor eine andere Person erscheint. Die nächste andere Person hat aber nur eine geringfügig höhere Distanz als das letzte Bild der Suchperson.

Nachteile von Eigenfaces

Die Gesichtserkennung im PCA-Raum mit Eigenfaces funktioniert mit der ATT-Gesichtersammlung sehr gut, weil die Gesichter aller Personen gleich und auch alle Gesichter einer Person (= einer Klasse) sehr gleichmäßig beleuchtet sind. Die Gesichtsmerkmale der Personen stimmen so gut mit den kontrastreichen Stellen überein.

Wären die Gesichter unterschiedlich beleuchtet, wie z.B. in Abb. 315, so würde die Detektion im PCA-Raum nur noch schlechte Resultate liefern, weil die Kontrastunterschiede nicht mehr mit den Klassenmerkmalen übereinstimmen.



Abb. 315: Zehn Bilder einer Person aus der Yale-Gesichterdatenbank mit unterschiedlichen Beleuchtungen.

Die PCA sortiert den Gesichter-Raum unabhängig von Klasseninformationen ausschliesslich anhand der Kontraste (Varianzen). Stimmen diese Kontraste nicht mit den Gesichtsmerkmalen überein, so verlieren wir durch die Sortierung die Unterscheidungsgüte zwischen den Gesichtsklassen.

Wie an einem einfachen Beispiel ersichtlich wird, kann durch die Projektion auf die Achse der grössten Varianz auch entscheidende Information für die Klassendifferenz verloren gehen:

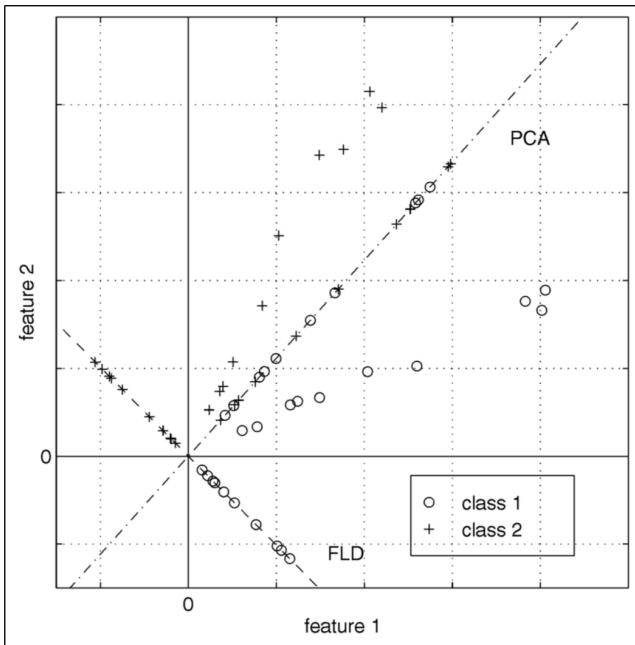


Abb. 316: 2 Klassen mit 2 Merkmalen. Durch Projektion beider Klassen auf die Achse der grössten Varianz verlieren wir Klassenunterscheidbarkeit. Durch Projektion auf die 2. Hauptachse würde die Unterscheidbarkeit erhalten bleiben.

Genau hier setzt die Weiterentwicklung mit der *Fisherface* Gesichtsdetektion an, welche durch die Herren *Belhumeur, Hespanha & Kriegman* 1997 [Belhumeur97] vorgestellt wurde. Das Verfahren ist nach dem britischen Statistiker *R.A. Fisher* benannt, weil es im Kern anstatt die PCA, die *lineare Diskriminanzanalyse* (LDA oder FLD) von Fisher verwendet. Die LDA sortiert den Merkmalsraum nicht nach Varianz, sondern nach maximaler Trennbarkeit von Merkmalen. Man verwendet dazu das Gütemass für Trennbarkeit, welches wir in Kapitel 11.1.1.2 kennengelernt haben:

$$q = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} = \frac{\text{Zwischenklassenvarianz}}{\text{Innerklassenvarianz}} = \frac{S_B}{S_W} = \frac{\text{between-class Scatter}}{\text{within-class scatter}}$$

11.2.1.2.3 Beispiel: SIFT-Deskriptoren Matching

Sinn und Zweck der SIFT-Deskriptoren ist natürlich die Suche von ähnlichen Deskriptoren, um damit ein Matching zweier Bilder vornehmen zu können. SIFT-Deskriptoren (s. Kapitel 10.5.2) sind von Grund auf invariant gegenüber Skalierung, Rotation, Helligkeitsunterschieden und in einem gewissen Mass auch gegenüber Verdeckungen und unterschiedlichen Blickwinkeln.

Der Vergleich zweier SIFT-Deskriptoren ist eine einfache Nächste-Nachbar-Suche im 128 dimensionalen, euklidischen Raum. Als Vergleichsmass reicht die quadrierte euklidische Distanz, die dem Punktprodukt zweier Deskriptoren $d1$ und $d2$ entspricht:

$$dist^2 = d1[0]*d2[0]+d1[1]*d2[1]+\dots+d1[127]*d2[127] = d1 \cdot d2$$

Da die Deskriptoren auf Einheitslänge normiert sind, entspricht das Punktprodukt dem Kosinus des Zwischenwinkels und wir können das Verhältnis der Zwischenwinkel als Annäherung an das Verhältnis der euklidischen Distanzen nehmen.



Im folgenden Matlab Beispiel, welches von [Lowe's Homepage](#) stammt, wird die Korrespondenz zwischen zwei Bildern mit dem Matlab Skript [*m/SIFT_Lowe/match.m*](#) berechnet. Zuerst werden für beide Bilder die Deskriptoren mit der Funktion `sift` berechnet. Diese ruft ein ausführbares Programm auf, welches (für Windows und Linux kompiliert) im gleichen Ordner liegt.

In einer Schlaufe werden dann für jeden Deskriptor des ersten Bildes die Winkel zu allen Deskriptoren des 2. Bildes berechnet und nach Grösse sortiert. Als identisch gilt ein Deskriptorenpaar, wenn deren Zwischenwinkel nur halb so gross oder kleiner ist, (*distRatio=0.5*) wie der Winkel zum nächsten Nachbar. Damit wird also nicht mit einem absoluten Mass verglichen, sondern mit einem relativen.

```
% Example call: match('scene.pgm','book.pgm');

function num = match(image1, image2)

% Find SIFT keypoints for each image
[im1, d1, loc1] = sift(image1);
[im2, d2, loc2] = sift(image2);

% distRatio: Only keep matches in which the ratio of vector angles from the
% nearest to second nearest neighbor is less than distRatio.
distRatio = 0.5;

% For each descriptor in the first image, select its match to second image.
d2t = d2';
for i = 1 : size(des1,1)
    dot = d1(i,:) * d2t; % Precompute matrix transpose
    [angle,indx] = sort(acos(dot)); % Computes vector of dot products
                                    % Take inverse cosine and sort results

    % Check if nearest neighbor has angle less than distRatio times the 2nd.
    if (angle(1) < distRatio * angle(2))
        match(i) = indx(1);
    else
        match(i) = 0;
    end
end

% Create a new image showing the two images side by side.
im3 = appendimages(im1,im2);

% Show a figure with lines joining the accepted matches.
figure('Position', [100 100 size(im3,2) size(im3,1)]);
colormap('gray');
imagesc(im3);
hold on;
cols1 = size(im1,2);
for i = 1: size(des1,1)
    if (match(i) > 0)
        line([loc1(i,2) loc2(match(i),2)+cols1, ...
               [loc1(i,1) loc2(match(i),1)], 'Color', 'c');
    end
end
```

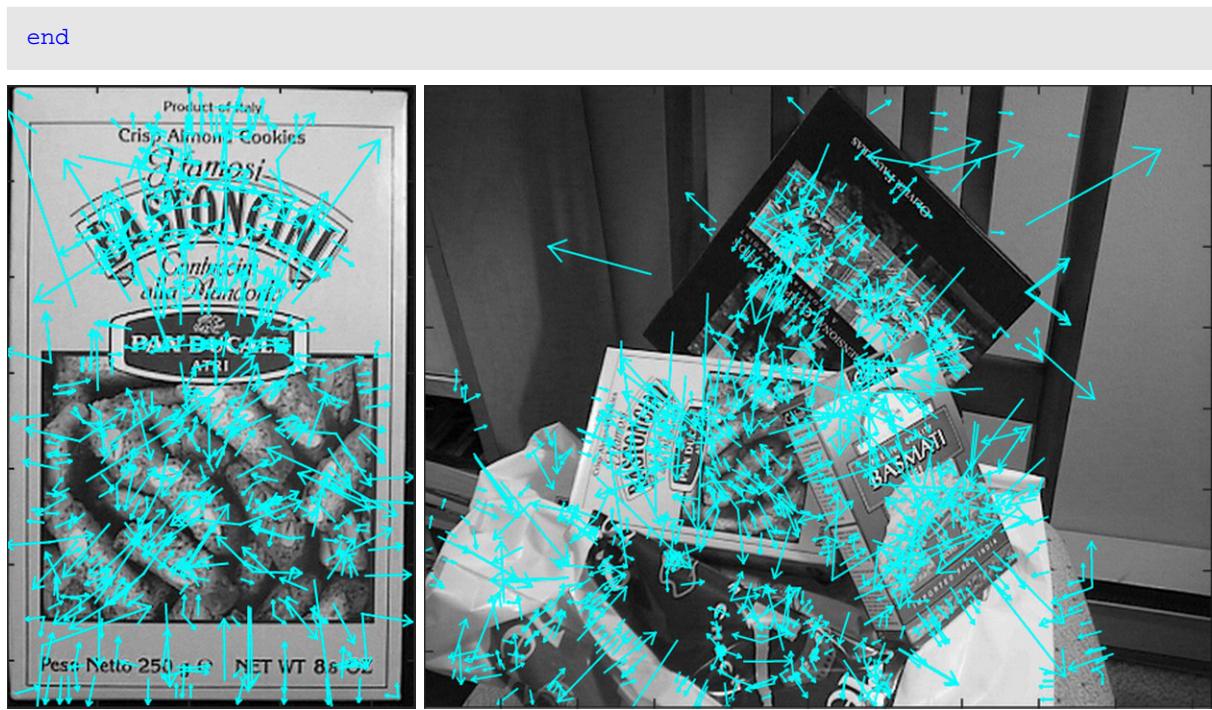


Abb. 317: Vergleichsbilder `box2.pgm` mit 640 Keypoints und `scene.pgm` mit 1021 Keypoints.

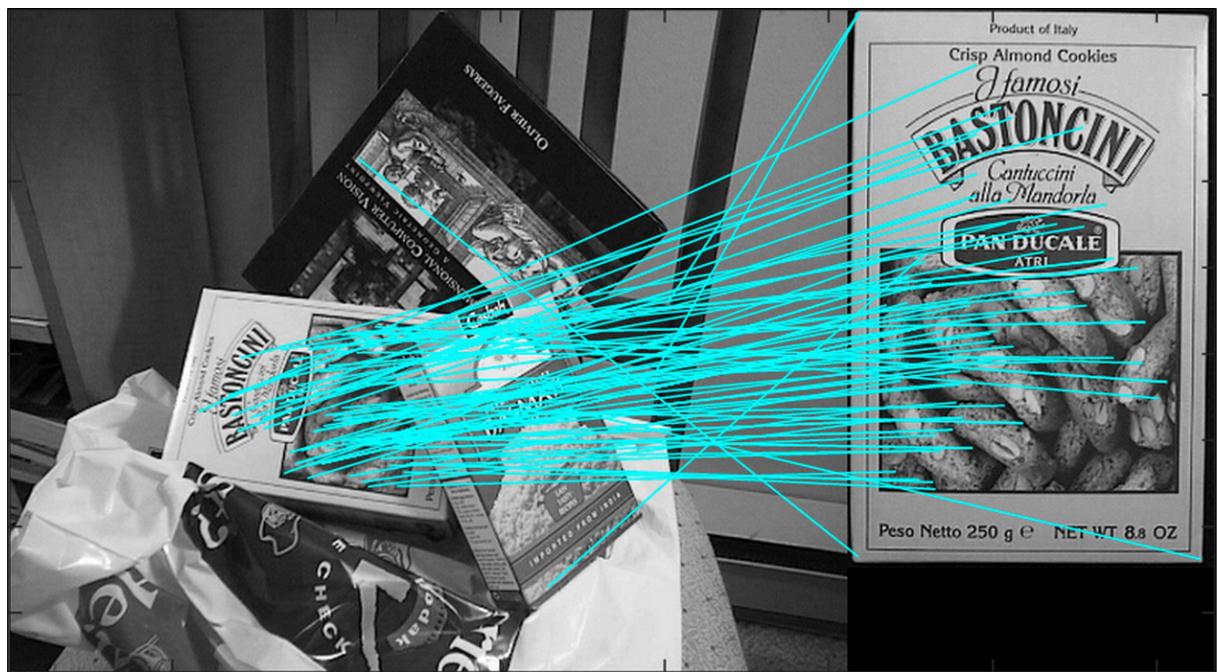


Abb. 318: Das Resultat des Matches mit allen Deskriptoren-Paaren, deren Distanz min. 50% kleiner war als die Distanz zum nächst weiter entfernten Deskriptor. Insgesamt wurden 62 der 640 Keypoints des rechten `box2` Bildes im Szenenbild erkannt.

Die Distanzen aller Deskriptoren des 1. Bildes zu allen Deskriptoren des 2. Bildes zu berechnen, ist natürlich genau aber auch sehr aufwendig. Es läge deshalb nahe, eine Beschleunigungsstruktur, wie z.B. einen kd-Baum dafür aufzubauen, um die Nächste-Nachbar-Suche zu beschleunigen. Ein kd-Baum über 128 Dimensionen verbraucht aber sehr viel Speicher und ist daher nicht unbedingt schneller. Lowe schlägt in [Lowe04] deshalb den Algorithmus *Best-Bin-First* vor, welcher den nächsten Nachbarn nach Wahrscheinlichkeit angibt.

11.2.1.2.4 Beispiel: SURF-Deskriptoren Matching

Wie bereits erwähnt verfügt Matlab selber über keinen SIFT-Merkmalendetektor. In der [Computer Vision System Toolbox](#) (ab R2014) finden sich aber einige [Detektoren](#) von SIFT-Nachfahren. Im Beispiel finden Sie ein ähnliches Matching, wie im vorangegangenen Beispiel basierend auf SURF-Deskriptoren [Object Detection In A Cluttered Scene Using Point Feature Matching](#). Sie können das Beispiel öffnen, in dem Sie im Matlab-Editor folgendes tippen: `edit FeatureBasedObjectDetectionExample`

11.2.2 Statistische Klassifikatoren

11.2.2.1 Bayes-Klassifikator

Der [Bayes-Klassifikator](#) ist ein aus dem [Bayes-Theorem](#) hergeleiteter Klassifikator. Er ordnet jedes Objekt der Klasse zu, zu der es mit der grössten Wahrscheinlichkeit gehört. Er wird deshalb auch [Maximum Likelihood Classifier](#) genannt.

Der Bayes-Klassifikator setzt voraus, dass die Wahrscheinlichkeit, mit der ein Punkt des Merkmalsraums zu einer bestimmten Klasse gehört, bekannt ist. D. h., für jede Klasse muss die Wahrscheinlichkeitsdichte definiert sein. Diese Dichtefunktionen sind aber meistens nicht bekannt. Deshalb nimmt man für jede Klasse einen Typ von Wahrscheinlichkeitsverteilung an, in der Regel eine Normalverteilung.

Auf die weitere Ausführung zum Bayes-Klassifikator soll hier verzichtet werden, da vertiefte Kenntnisse in Wahrscheinlichkeitstheorie zum Verständnis notwendig sind. Im nächsten Kapitel wird dafür eine verständlichere, geometrisch nachvollziehbare Version eines statistischen Klassifikators vorgestellt.

Mehr zum Bayes-Klassifikator erfahren Sie im Standardwerk [Pattern Classification](#) von Duda und Hart [[DudaHart00](#)]. In [[Gonzales08](#)] finden Sie ein komplettes Beispiel für Matlab.

11.2.2.2 Mahalanobis-Distanz zum Cluster-Zentrum

Die euklidische Distanz in mehrdimensionalen Räumen ist irreführend, wenn die Achsen der Verteilungsdichten einer Klasse nicht parallel zu den Koordinatenachsen verlaufen. Im nachfolgenden Beispiel sehen Sie zwei Cluster mit den Zentren bei $[-1.5, 0]$ und bei $[1.4, 0]$. Für unseren Kandidaten bei $[0, 0]$ ergibt sich nach euklidischer Distanz eine Zuordnung zur blauen Klasse. Wir sehen jedoch, dass die beiden Cluster nicht gleichmässig verteilt sind, und müssen uns daher fragen, ob die blaue Klasse auch wirklich die wahrscheinlichere ist.

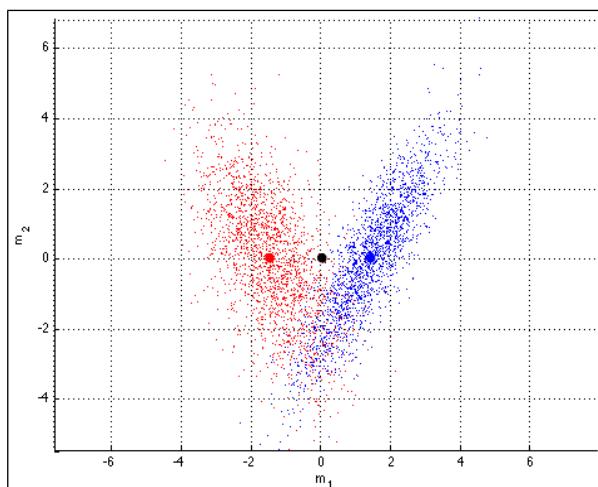


Abb. 319: Zwei Cluster mit normalverteilten zweidimensionalen Merkmalen mit den Cluster-Zentren bei $[-1.5, 0]$ und $[1.4, 0]$. Nach euklidischer Distanz müsste der schwarze Kandidat bei $[0, 0]$ zur blauen Klasse gehören.

Hauptkomponentenanalyse

Wenn wir von einer normalverteilten Wahrscheinlichkeitsdichte ausgehen, können wir mit der Hauptkomponentenanalyse die Dichteverteilung und Ausrichtung bestimmen (s. Kapitel 7.4). Aus der Kovarianzmatrix können wir die Eigenvektoren und Eigenwerte berechnen. Die Eigenvektoren sind Einheitsvektoren und entsprechen den Richtungen der Hauptachsen einer normal verteilten Zufallsvariablenverteilung. Die Eigenwerte sind die quadrierten Distanzen entlang der Hauptachse bis zur Standardabweichung $\sigma=1$. Weil die Skalierung der Eigenwerte der Kovarianzmatrix einhergeht mit der Standardabweichung, können wir sie als Mass für die Wahrscheinlichkeit verwenden.



Beispiel mit Matlab

Der nachfolgende Ausschnitt aus dem Matlab-Programm *MahalanobisDemo.m* erzeugt den blauen Cluster mit 1000 Punkten und berechnet davon die Kovarianzmatrix und daraus wiederum die Eigenvektoren und Eigenwerte. Zur Verdeutlichung der Dichte und der Ausrichtung werden die Hauptachsen bis zu $\sigma=1$ eingezeichnet. Die Ellipsen verdeutlichen die Dichten bei $\sigma=1,2,3,4$.

```
% create n multivariate random points with normal distribution
% from the covariance matrix sigma around the center point c1
c1 = [1.4; 0.0];
cov1 = [1.0 1.5;
         1.5 3.0];
r1 = mvnrnd(c1, cov1, 2000);

% Create the covariance matrix out of the data r
% This should give roughly the same as the input covariance matrices
cv1 = cov(r1(:,1),r1(:,2));

% Get the eigenvectors and eigenvalues from the covariance matrix
[eigvec1, eigval1] = eig(cv1);

% The squareroot of the eigenvalues on the diagonal correspond
% to length of the principal components
eigval1 = sqrt(diag(eigval1));

% We sort the eigenvalue to have the longest first
[eigval1, indices1] = sort(eigval1, 'descend')
eigvec1 = eigvec1(:, indices1)

%plot the datasets
plot(r1(:,1),r1(:,2), 'b.', 'MarkerSize', 1);

% Get the eigenvectors
e11 = eigvec1(:, 1);
e12 = eigvec1(:, 2);

% Build scaled principal axis from the center points
a11 = c1 + eigval1(1) * e11;
a12 = c1 + eigval1(2) * e12;

% Plot the ellipses every half sigma
phil = atan(e11(2)/e11(1))
for s = 1:1:4
    ellipse(s*eigval1(1), s*eigval1(2), phil, c1(1), c1(2), 'b');
end

% Plot the the principal axis
plot([c1(1) a11(1)], [c1(2), a11(2)], 'k-', 'LineWidth', 2);
plot([c1(1) a12(1)], [c1(2), a12(2)], 'k-', 'LineWidth', 2);
```

Die Ausgabe des Programms ist im nachfolgenden Bild zu sehen. Wir erkennen darin, dass der schwarze Punkt in der Mitte vom roten Cluster-Zentrum ca. 1.8 σ und vom blauen Zentrum ca. 2.7 σ entfernt ist.

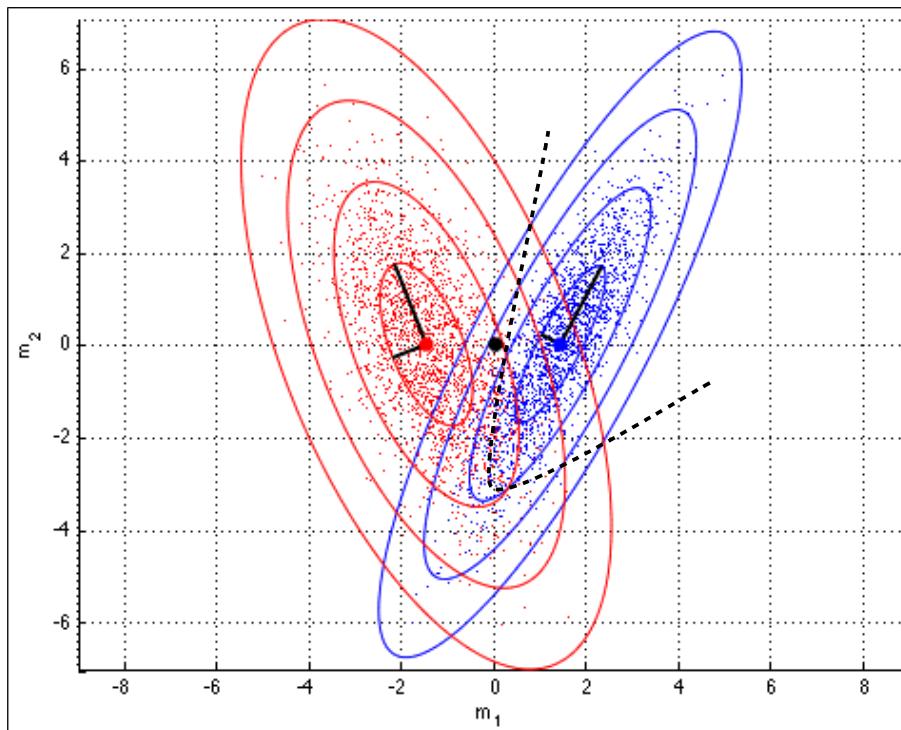


Abb. 320: Zwei Cluster mit ihren Dichtelinien mit jeweils 1σ Abstand. Verbindet man alle Kreuzungspunkte korrespondierender Dichtelinien, so erhält man den Grenzverlauf (gestrichelt) zwischen den beiden Clustern. Dieser entspricht dem Talboden zwischen den beiden Bergen.

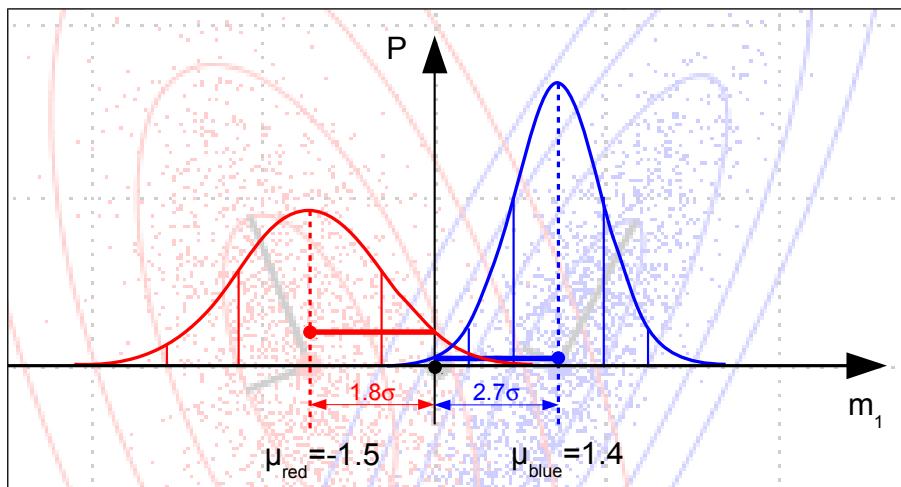


Abb. 321: Querschnitt durch die obere Grafik bei $y=0$. Die vertikale Achse entspricht der absoluten Häufigkeit der beiden Verteilungen. Wir sehen die beiden Gauss-Glockenkurven mit jeweils der gleichen Fläche 1. In den Gauss-Glocken sind die unterschiedlichen Skalen der Standardabweichungen eingezeichnet, die mit den darunterliegenden Ellipsen korrespondieren.

Der schwarze Punkt in der Mitte kommt auf der roten Gaussglocke höher zu liegen und gehört damit **wahrscheinlicher** zum roten Cluster als zum Blauen.

Diese Distanzangabe im Dichteraum einer multivariaten (= multidimensional für Statistiker) Zufallsvariablen wird **Mahalanobis-Distanz** genannt. Sie ist definiert als Distanz von einem Punkt p_n zum Schwerpunkt μ_n eines n -dimensionalen, normal verteilten Zufallszahlenraums mit der Kovarianzmatrix C_n :

$$D_M(p_n) = \sqrt{(p_n - \mu_n)^T \cdot C_n^{-1} \cdot (p_n - \mu_n)}$$

Die Distanzeinheit ist in Sigma des normalverteilten Zufallszahlenraums. Ist die Kovarianzmatrix eine Identitätsmatrix, so resultiert wieder die euklidische Distanz. In Matlab erhalten wir diese Distanz mit der Funktion [mahal](#):

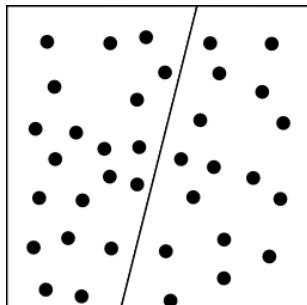
```
% Mahalanobis Distance  
plot(c1(1),c1(2), 'b*', 'linewidth', 3);  
plot(c2(1),c2(2), 'r*', 'linewidth', 3);  
plot(0,0, 'k*', 'linewidth', 3);  
mhd1 = sqrt(mahal([0 0], r1))  
mhd2 = sqrt(mahal([0 0], r2))
```

Wir lernen an diesem Beispiel Folgendes:

- Der Grenzverlauf zwischen zwei Klassen verläuft **nicht linear**, wie es die meisten auf euklidischer Distanz basierenden Klassifikatoren erzeugen. Insbesondere bei sich überlappenden oder sogar durchdringenden Clustern ist ein auf Dichteverteilungen basierender Grenzverlauf entscheidend für eine korrekte Klassifikation.
- Mit der Mahalanobis-Distanz kennen wir einen auf statistischen Verteilungsdichten basierenden Klassifikator.
- Die Mahalanobis-Distanz ist jedoch nur ein korrektes Mass, wenn die Daten auch wirklich normalverteilt sind.

11.2.3 Beurteilung von Klassifikatoren

Bei einer Klassifizierung werden Objekte aufgrund der Merkmale durch einen Klassifikator in verschiedene Klassen eingeordnet. Der Klassifikator kann dabei Fehler machen, indem er ein Objekt einer falschen Klasse zuweist. Aus der relativen Häufigkeit dieser Fehler lassen sich quantitative Masse zur Beurteilung eines Klassifikators ableiten.



Um einen Klassifikator zu beurteilen, muss man ihn in einer Reihe von Fällen anwenden, bei denen man Kenntnis über die wahre Klasse der jeweiligen Objekte hat.

Stellen wir uns als Beispiel eine Menge an Personen, die entweder gesund (Punkte rechts) oder krank (Punkte links) sind. Da ein Test meist nicht perfekt ist, entsteht daraus nicht nur die Gruppe der kranken (richtig positiv) und der gesunden (richtig negativ) Personen, sondern auch die beiden Gruppen der falsch zugewiesenen Personen.

Nach einer Klassifizierung können vier mögliche Fälle auftreten:

- **Richtig positiv:** Der Patient ist krank, und der Test hat dies richtig angezeigt.
- **Richtig negativ:** Der Patient ist gesund, und der Test hat dies richtig angezeigt.
- **Falsch positiv:** Der Patient ist gesund, aber der Test hat ihn fälschlicherweise als krank eingestuft.
- **Falsch negativ:** Der Patient ist krank, aber der Test hat ihn fälschlicherweise als gesund eingestuft.

Diese vier Möglichkeiten werden in der sogenannten *Wahrheitsmatrix* zusammengefasst:

	Person ist krank ($r_p + f_n$)	Person ist gesund ($f_p + r_n$)
Test positiv ($r_p + f_p$)	richtig positiv (r_p)	falsch positiv (f_p)
Test negativ ($f_n + r_n$)	falsch negativ (f_n)	richtig negativ (r_n)

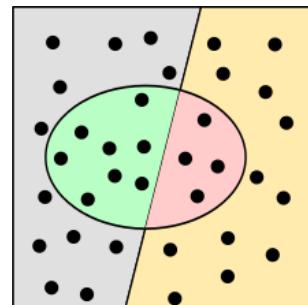


Abb. 322: Wahrheitsmatrix und ihre Visualisierung. Jeder Mensch wird durch einen Punkt dargestellt, der links (krank) bzw. rechts (gesund) der schwarzen Linie liegt. Die Punkte im Oval sind die von dem Test als krank klassifizierten Menschen. Richtig bewertete Fälle sind grün, falsch bewertete rot unterlegt. Quelle: Wikipedia.

Durch Berechnung von relativen Häufigkeiten können aus den Werten der Wahrheitsmatrix nun verschiedene Masse zur Beurteilung des Klassifikators berechnet werden.

Die **Sensitivität** gibt den Anteil der richtig positiv klassifizierten Objekte an der Gesamtheit der tatsächlich positiven Objekte an. Bei einer medizinischen Diagnose entspricht die Sensitivität dem Anteil an tatsächlich Kranken, bei denen die Krankheit auch erkannt wurde.

$$\text{Sensitivität} = \frac{r_p}{r_p + f_n}$$

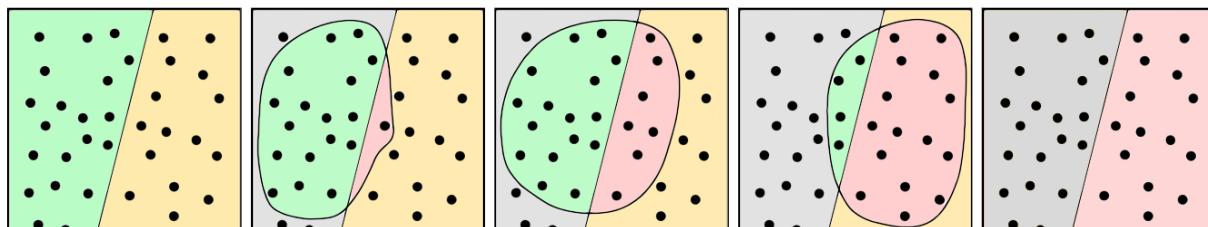
Die **Genauigkeit** ist der Anteil der richtig positiv klassifizierten Objekte an der Gesamtheit der positiven Objekte an (erste Zeile der Wahrheitsmatrix). Bei einer medizinischen Diagnose an gibt die Genauigkeit an, wie viele Personen, bei denen die Krankheit festgestellt wurde, auch tatsächlich krank sind.

$$\text{Genauigkeit} = \frac{r_p}{r_p + f_p}$$

Das **F-Mass** ist ein kombiniertes Mass aus der Sensitivität und der Genauigkeit für die Güte eines Klassifikators ableiten. Das F-Maß kombiniert Genauigkeit und Sensitivität mittels des gewichteten [harmonischen Mittels](#):

$$F = 2 \cdot \frac{\text{Genauigkeit} \cdot \text{Sensitivität}}{\text{Genauigkeit} + \text{Sensitivität}}$$

Das F-Mass gibt eine Wert grösser 0 und kleiner gleich 1 und bedingt mindestens einen richtig positiven Test:



richtig positiv = 20	richtig positiv = 17	richtig positiv = 6	richtig positiv = 4	richtig positiv = 0
falsch positiv = 0	falsch positiv = 1	falsch positiv = 7	falsch positiv = 15	falsch positiv = 16
falsch negativ = 0	falsch negativ = 3	falsch negativ = 4	falsch negativ = 16	falsch negativ = 20
richtig negativ = 16	richtig negativ = 15	richtig negativ = 9	richtig negativ = 1	richtig negativ = 1
Sensitivity = 1.00	Sensitivity = 0.85	Sensitivity = 0.60	Sensitivity = 0.20	Sensitivity = 0.00
Precision = 1	Precision = 0.94	Precision = 0.46	Precision = 0.21	Precision = 0.00
F = 1	F = 0.89	F = 0.52	F = 0.21	F = #DIV/0!

11.3 Template Matching

Suchvorgänge können als Klassifikation interpretiert werden, wenn wir bestimmte Pixel oder Regionen einer einzigen Klasse zuordnen möchten. Insofern können wir den Prozess *Segmentierung-Merkalsextraktion-Klassifikation* als eine Methode betrachten, um ein Objekt mit bestimmten Merkmalen zu suchen.

Mit *Template Matching* (= Mustersuche) gibt es eine alternative Methode, mit der man beliebige nicht durch Merkmale parametrisierbare Muster in einem Bild suchen kann. In seiner einfachen Version setzt Template Matching ein Suchmuster voraus, dessen Grösse und Orientierung gegeben sind und nur die Position gesucht werden muss. In allen Varianten von Template Matching wird für alle möglichen Positionen des Templates im Suchbild ein Mass für die Musterähnlichkeit bestimmt.

11.3.1 Hohe Musterähnlichkeit durch kleine Differenz

Das am einfachsten zu berechnende Mass für Ähnlichkeit ist die absolute Differenz zwischen zwei Signalen:

Durchschnittlicher absoluter Abstand (*MAD = Mean absolute distance*):

$$MAD(x, y) = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N |g(x, y) - t(x, y)|$$

Alternativ dazu:

Durchschnittlicher quadratischer Abstand (*MSD = Mean squared distance*):

$$MSD(x, y) = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N (g(x, y) - t(x, y))^2$$

Je kleiner diese Differenzmasse sind, umso grösser ist die Übereinstimmung. Diese Masse sind sehr robust gegenüber Rauschen dafür aber anfällig gegenüber durchschnittlichen Helligkeitsunterschieden:

Original	Brighter	Darker	Low Contrast	Inverse	Noisy
MAD=0.00 MSD=0.00	0.18 0.07	0.38 0.22	0.30 0.10	0.89 0.84	0.15 0.06

Abb. 323: Differenzwerte (MAD und MSD) zwischen dem Originalbild links und 5 Varianten rechts davon. Man beachte, dass die verrauschte Version eine kleinere Differenz ergibt als die in Helligkeit und Kontrast variierten Versionen.

11.3.1.1 Block Matching Algorithmen (BMA)

Bei der Bewegungskompensation in MPEG-Videostandards (s. Kapitel 12.4.5) werden identische Bildblöcke, die sich gar nicht oder nur durch eine Verschiebung unterscheiden, durch einen Bewegungsvektor ersetzt. Das bedeutet, dass bei der Videokompression die Hauptaufgabe darin besteht, für jeden Bildblock in vorgängigen oder z.T. auch nachfolgenden Videobildern nach dem entsprechenden Bildblock zu suchen.

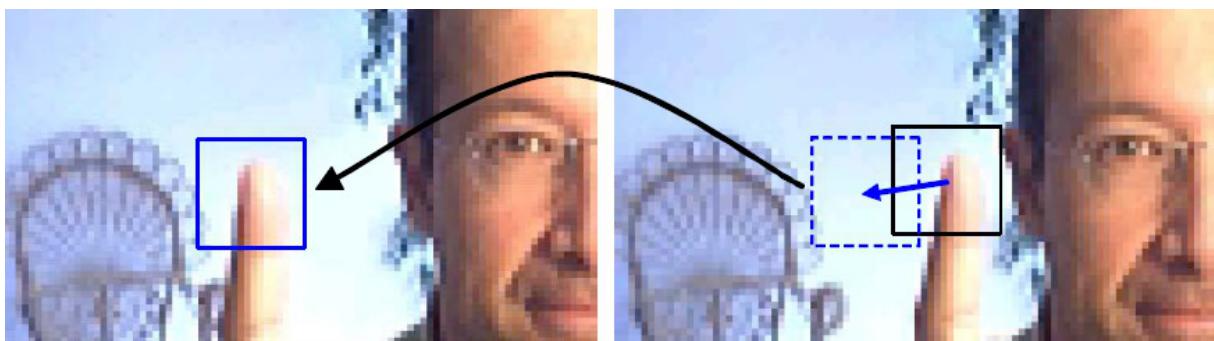


Abb. 324: Welcher Bildblock im Vorgängerbild links hat die geringste Differenz zum Bildblock Bild rechts?

Als Mass für die Ähnlichkeit wird der MAD oder MSD der Vergleichsblöcke verwendet. Ist diese Differenz null, so sind die Blöcke identisch. Dies wird jedoch kaum vorkommen, da sich bei statischen Blöcken alleine durch das Bildrauschen eine Differenz ergibt. Deshalb wird neben dem gefundenen Bewegungsvektor auch ein Fehlerbild (= Differenzbild) mitkodiert.

In MPEG-Standards hat ein Bildblock z. B. die Grösse von 16×16 Pixeln. Bei einem MPEG-2 Video (Frame-Grösse im PAL-Standard: 720×576) müssen also 1620 korrespondierende Bildblöcke gesucht werden. Bei einem Video mit 25 Frames pro Sekunde (FPS) bewegen sich Bildinhalte relativ langsam, sodass wir den Suchbereich stark einschränken können. Algorithmen, welche dieses Template Matching effizient durchführen, werden in der Videocodierung *Block Matching Algorithmen (BMA)* genannt.

Full Search

Der einfachste, aufwendigste und zugleich genauste BMA ist der *Full Search* Algorithmus. Er bestimmt für jede mögliche Position die Differenz. Bei einer Suchbreite w sind dies $(2w+1)^2$ MAD- oder MSD-Berechnungen. Mit $w=7$ Pixeln auf allen Seiten des Blocks ergeben sich so $(2*7+1)^2 \cdot 16 \cdot 16 = 57'600$ Pixelvergleiche für jeden der Bildblöcke. Damit kann ein Video aber nicht mehr in Echtzeit in Software komprimiert werden.

Three Step Search

Dieser BMA ist einer der am meisten verwendeten Algorithmen im MPEG-1-Standard. Er reduziert die Anzahl der Tests drastisch und kann trotzdem eine Verschiebung fast so genau bestimmen, wie der *Full Search* Algorithmus. Er heisst *Three Step Search*, weil er in drei Schritten einen Verschiebungsvektor bestimmen kann. In der nachfolgenden Abbildung sehen Sie einen möglichen Verlauf der Suche.

1. Im ersten Schritt werden 9 Vergleichsblöcke mit dem Ausgangsblock verglichen. Die Zentren sind jeweils 4 Pixel vom Zentrum des Ausgangsblockes entfernt. Es werden also nicht 9 Pixel verglichen, sondern es wird die Differenz (MSE oder MAE) der 9 16×16 er Blöcke zum Ausgangsblock berechnet.
2. Im zweiten Schritt werden 8 neue Blöcke verglichen, die um das Zentrum mit der geringsten Differenz des ersten Schrittes angeordnet sind. Die Distanz verkleinert sich auf 2 Pixel.
3. Im dritten Schritt wiederholt sich das Ganze mit den letzten 8 Zentren, die nur noch ein Pixel Abstand haben vom letzten Besten.

Im nachfolgenden Beispiel ergibt sich so ein Verschiebungsvektor von Pixel A zum Pixel B. Im Vergleich zur *Full Search* Methode mit $(2x7+1)^2=225$ Blockvergleichen, waren bei der TSS-Methode insgesamt nur 25 Vergleiche notwendig.

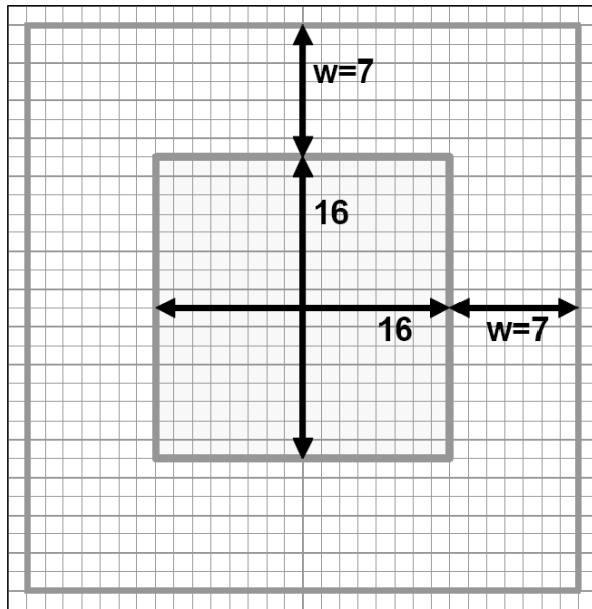


Abb. 325: 16x16 Pixel Ausgangsblock mit einem 7 Pixel breiten Suchbereich.

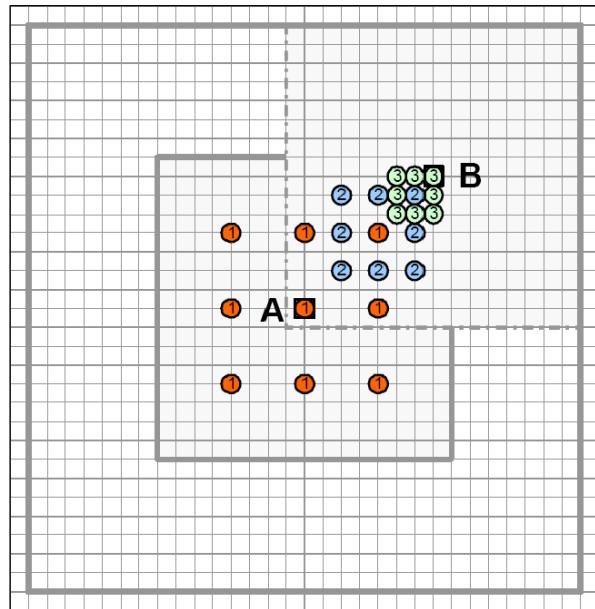


Abb. 326: Three Step Search (TSS). Jeder Kreis stellt das Zentrum eines 16x16 Pixel grossen Vergleichsblocks dar.

11.3.2 Hohe Musterähnlichkeit durch grosse Korrelation

Bis jetzt haben wir mit Template Matching praktisch identische Bildbereiche verglichen. Oftmals weicht ein Template aber stärker vom entsprechenden Bereich im Suchbild ab. Eine höhere Robustheit gegenüber Helligkeits- und Kontrastunterschieden bietet die *normierte Kreuzkorrelationsfunktion (NCCF)*, die für jede Position eines Templates in einem Bild einen *Korrelationskoeffizienten* zw. -1 und 1 zurückgibt. Die NCCF wird im Englischen auch oft *Pearson Korrelation* genannt.

Für ein Template t und ein gleich grosser Ausschnitt g aus einem Suchbild ist er wie folgt definiert:

$$NCCF(x, y) = \frac{cov_{gt}}{\sqrt{var_g \cdot var_t}} = \frac{\frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N (g(x, y) - \bar{g})(t(x, y) - \bar{t})}{\sqrt{\frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N (g(x, y) - \bar{g})^2 \sum_{x=1}^M \sum_{y=1}^N (t(x, y) - \bar{t})^2}}$$

Worin \bar{g} bzw. \bar{t} der Mittelwert im entsprechenden Bildausschnitt bzw. des Templates ist.

Die Kovarianz (s. auch 7.4.1) und damit auch der Korrelationskoeffizient ist besonders hoch, wenn $g(x, y)$ weit weg von \bar{g} und $t(x, y)$ weit weg von \bar{t} ist. Umgekehrt ist die Kovarianz klein, wenn diese Distanzen gegenläufig, also die Varianzen sehr unterschiedlich sind.

Der Betrag der resultierenden Korrelationskoeffizienten ist unabhängig von Kontrast- und Helligkeitsunterschieden, wie Sie gut in Abb. 327 erkennen können. Die Werte für MAD, MAS und CC wurden mit dem Matlab-Skript *Correlation_e.m* berechnet.

Der Korrelationskoeffizient geht gegen 1, wenn es einen Kontrastfaktor $s > 0$ und eine Helligkeitsverschiebung d gibt, sodass $g(x, y) = s \cdot t(x, y) + d$ ist.

Der Korrelationskoeffizient geht gegen -1, wenn es darin einen Kontrastfaktor $s < 0$ gibt.

Der Korrelationskoeffizient ist 0, wenn keine lineare Abhängigkeit zwischen g und t besteht.



Abb. 327: Korrelationskoeffizienten CC zwischen dem Originalbild links und 5 Varianten rechts davon. Man beachte, dass Sie kaum abhängig sind von der Helligkeit und vom Kontrast, jedoch sensibler auf Rauschen sind.

11.3.2.1 Beispiel mit normierter Kreuzkorrelation



Im **Matlab-Skript** *CorrelationSearch1.m* wird aus einem Bild der Suchbereich des Buchstabens *e* ausgeschnitten als Template *t*. Mit der Funktion *normxcorr2* wird der Korrelationskoeffizient *C* für alle Positionen vom Template *t* im Bild *g* berechnet. Das Resultat ist demnach wiederum ein Bild und kann angezeigt werden. Pixel mit hoher Korrelation sind hell. Danach werden alle Positionen gesucht, die einen Korrelationskoeffizienten von grösser 0.75, grösser 0.85, grösser 0.95 und grösser 0.99 haben.

```

X = imread('..../images/TemplateMatchingText.png'); %Load image into matrix X
g = im2double(X); %Convert to doubles [0-1]
t = g(13:25, 61:72); %Extract template for letter e
imshow(t); title('Template');

tic %Start watch
CC = normxcorr2(t, g); %Normalized cross corelation
tic %Stop watch

figure; % Open new window
subplot(1,2,1), imshow(CC,[]); title('Correlation coefficients w. normxcorr2');

pos_e99 = CC > 0.99; %positions w. CC > 0.99
pos_e95 = CC > 0.95; %positions w. CC > 0.95
pos_e85 = CC > 0.85; %positions w. CC > 0.85
pos_e75 = CC > 0.75; %positions w. CC > 0.85

% Mark positions with high correlation
subplot(1,2,2), imshow(g), title('red>99%, green>95%, blue>85%, yellow>75%');
hold on
ps = size(t,1)/2;
for y=1:size(pos_e99,2)
    for x=1:size(pos_e99,1)
        if pos_e99(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',3,'Color','red');
        elseif pos_e95(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',3,'Color','green');
        elseif pos_e85(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',3,'Color','blue');
        elseif pos_e75(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',3,'Color','yellow');
        end
    end
end

```



Abb. 328: Links: Gesucht wurde der Buchstaben e, für den ein Template aus dem ersten e der ersten Zeile erzeugt worden ist. In der Mitte sehen Sie die Korrelationskoeffizienten. Im Bild rechts sehen Sie die Positionen markiert, wo die Korrelationskoeffizienten gross sind. Die Farben haben folgende Bedeutung: Rot CC>0.99, Grün CC>0.95, Blau CC> 0.85 und Gelb CC>0.75. Man erkennt leicht, dass das Matching eine maximale Korrelation liefert zwischen der dritten und der ersten Zeile. Dies, obwohl sich der Kontrast der dritten Zeile im Vergleich zur ersten Zeile stark unterscheidet. Gut ersichtlich ist auch die hohe Sensibilität auf Grössenskalierungen und Rotation.

11.3.2.2 Beispiel mit nicht normierter Korrelation

Der Aufwand der normierten Kreuzkorrelation ist hoch, da für jede Position der Mittelwert \bar{g} des Bildbereichs unter dem Template berechnet werden muss. Mit gewissen Abstrichen kann durch Vereinfachungen aber eine starke Beschleunigung erzielt werden. Da der Korrelationskoeffizient unabhängig von Kontrast- und Helligkeitsunterschieden ist, können der Kontrast und die Helligkeit so angenommen werden, dass \bar{g} und \bar{t} 0 werden und wegfallen. Unter der weiteren Annahme, dass das Template auf dieselbe Grösse wie das Suchbild vergrössert wird (mit Nullen ergänzt), kann die Varianz für g und t einmal berechnet werden und vor das Summenzeichen ausgelagert werden:

$$CC(x, y) = \frac{1}{\sqrt{\text{var}_g \cdot \text{var}_t}} \cdot \frac{1}{MN} \cdot \sum_{x=1}^M \sum_{y=1}^N g(x, y) \cdot t(x, y)$$

Lassen wir die Skalierung (=Normierung) vor dem Summenzeichen ganz weg, so unterscheidet sich die als *Kreuzkorrelation* oder einfach *Korrelation* bekannte Funktion nur durch die fehlende Spiegelung des Templates von der Konvolution (Faltung) zweier Funktionen:

Korrelation zweier Funktionen: $C(x, y) = \sum_{x=1}^M \sum_{y=1}^N g(x, y) \cdot t(x, y)$

Konvolution zweier Funktionen: $f(x, y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} g(x, y) \cdot h(u-x, v-y)$

Demnach können wir die Korrelation einfach mit einer Faltungsfunktion berechnen, wobei wir vorher das Template um 180 Grad drehen müssen. Weil Korrelation und Konvolution sehr ähnlich sind, erstaunt es nicht, dass wir die Korrelation auch im Frequenzraum durch Multiplikation durchführen können.



Im **Matlab-Skript** *CorrelationSearch2.m* wird dasselbe e gesucht, wie im vorangegangenen Beispiel. Es kommt vorerst nichts Schlaues dabei heraus, weil durch die fehlende Normierung auch überall dort eine maximale Korrelation berechnet wird, wo das Template mit dem weissen Hintergrund multipliziert wird. D.h., dass Korrelation zur Suche eines Musters nur funktioniert, wenn der Hintergrund schwarz ist. Für unser Beispiel genügt es, Bild und Template zu invertieren.

```
x = imread('../images/TemplateMatchingText.png'); %Load image into matrix x
g = im2double(x); %Convert to doubles [0-1]
imshow(g); title('Original');
```

```

t = g(13:25, 61:72);                                %Extract template for letter e
figure;                                               %Open new image
imshow(t); title('Template');

tic;                                                 %Start watch
g = ones(size(g,1),size(g,2)) - g;                 %Invert image for correlation
t = g(13:25, 61:72);                               %Template e

C = conv2(g, rot90(t,2));                         %correlation by convolution
% Correlation in frequency domain
% C = real(ifft2(fft2(g) .* fft2(rot90(t,2),size(g,1),size(g,2)))); 
toc;                                                 %Stop watch

figure;
subplot(1,2,1), imshow(C,[]); title('Correlation coefficients w. correlation');

pos_e99 = C > max(C(:))*0.99; %positions of e 99% of max.
pos_e95 = C > max(C(:))*0.95; %positions of e 95% of max.
pos_e85 = C > max(C(:))*0.85; %positions of e 85% of max.
pos_e75 = C > max(C(:))*0.75; %positions of e 85% of max.

% Mark positions with high correlation
ps = size(t,1)/2;
subplot(1,2,2), imshow(g), title('red>99%, green>95%, blue>85%, yellow>75%');
hold on
for y=1:size(pos_e99,2)
    for x=1:size(pos_e99,1)
        if pos_e99(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',2,'Color','red');
        elseif pos_e95(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',2,'Color','green');
        elseif pos_e85(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',2,'Color','blue');
        elseif pos_e75(x,y)==1
            plot(y-ps,x-ps,'x','LineWidth',2,'Color','yellow');
        end
    end
end

```

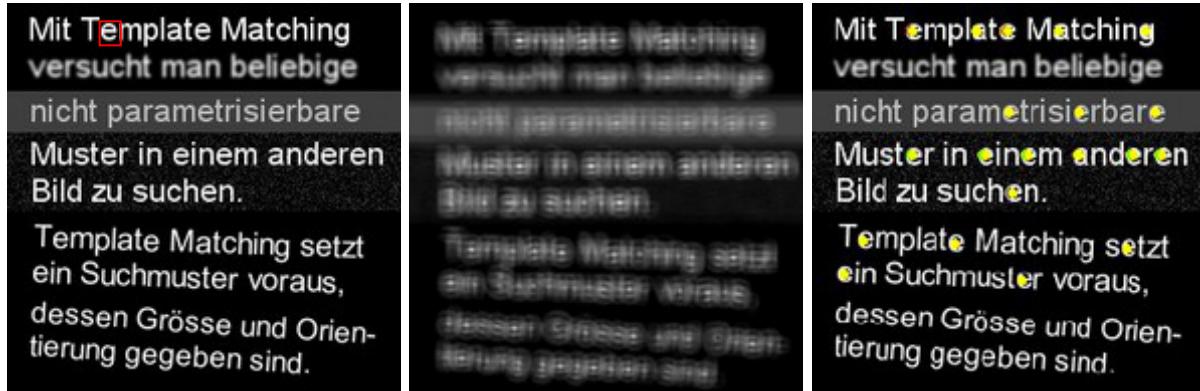


Abb. 329: Links: Das invertierte Bild mit Template e. In der Mitte die Korrelationswerte als Bild und rechts die Punkte der höchsten Korrelation. Die roten und grünen Punkte in der ersten und vierten Zeile werden durch gelbe Punkte überdeckt, die für eine benachbarte Position berechnet wurden. Im letzten Abschnitt, mit der, um zwei Punkte verkleinerten und rotierten Schrift, werden keine e's mehr gefunden und in der ersten und vierten Zeile bekommt auch ein a eine Korrelation, die grösser als 75% vom Maximum ist.

Es ist klar ersichtlich, dass die nichtnormierte Korrelation mit Faltung im Ortsraum oder mit Multiplikation im Frequenzraum nicht mehr dieselbe Präzision liefert wie die normierten Korrelationskoeffizienten. Dafür ist sie um einiges schneller:

Methode:	Zeit(Sek.)	Anteil
Normierte Kreuzkorrelation	0.0320	100%
Nichtnormierte Korrr. im Frequenzraum:	0.0100	31%
Nichtnormierte Korrr. im Ortsraum:	0.0042	13%

11.4 Übungen

11.4.1 Übung: k-Means Clustering



Das Beispiel im Kapitel 11.2.1.1 verwendet den L*a*b Farbraum, um das Zellbild mit k-Means-Clustering zu segmentieren. Probieren Sie andere Farbräume aus, um festzustellen, ob man damit die blauen Zellkerne besser segmentieren kann. Visualisieren Sie die Farbhistogramme zuerst in Fiji mit dem *3D-Color Inspector*.

11.4.2 Übung: Eigenfaces



Bauen Sie das Skript *PCA4_Eigenfaces.m* für mit der Gesichtserkennung aus. Suchen Sie in einer Schlaufe nach allen 400 Gesichtern:

- Für wie viele Gesichter finden wir unter den 9 nächsten Gesichtern eine andere Person?
- Für wie viele Gesichter findet das Verfahren an zweiter Stelle eine andere Person? Für dieses Suchbild wäre die Identifikation fehlgeschlagen.

11.4.3 Übung: Face Recognition Contest

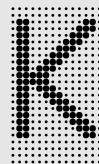
In Zweier-Teams soll eine komplette Gesichtserkennung und Personenidentifikation in beliebigen Bildern implementiert werden. Ziel ist es, dass ihr Programm möglichst zuverlässig alle Personen in einem Gruppenfoto erkennen kann.

1. Wir erstellen am **Freitag 6. November 2015** von allen Studierenden je 10 Graustufenaufnahmen des Gesichts mit möglichst gleichmässiger Beleuchtung. Sie finden die Fotos etwas später im Ordner *Exercises/Images/cpvr_faces_160* und *cpvr_faces_320*. Die Gruppenfotos sind im Ordner *cpvr_classes*.
2. Implementieren Sie mit Template Matching die Gesichterlokalisierung in einem beliebigen farbigen Gruppenbild:
 1. Bilden Sie dazu ein Farbtonmodell für Haut. Analysieren Sie dazu verschiedene 2D- oder 3D-Farbbild-Histogramme.
 2. Segmentieren Sie das Farbbild nach Hautfarbton. Binarisieren Sie das Hautbild, um mit weissen Flecken die Hautregionen zu markieren. Bereinigen Sie kleine Hautregionen, die zu klein für ein Gesicht sind, und löschen Sie kleine Löcher in grösseren Hautregionen.
 3. Suchen Sie in den verbleibenden Gesichtsregionen im Graustufengruppenbild mit Template Matching die Gesichter. Suchen Sie mit einem auf die Grösse des Gesichtsflecks angepassten Graustufendurchschnittsgesicht.
3. Anstatt die Gesichter mit Template Matching zu suchen, können Gesichter auch mit dem sogenannten *Viola-Jones-Algorithmus* (Nr. 9 auf der List der *Top10-CV-Paper*) gefunden werden.
4. Identifizieren Sie die gefundenen Gesichter mit der Eigenface-Gesichtsdatenbank. Alternativ ist auch eine Gesichtsdatenbank mit *Fisherfaces* möglich. Sie finden im Internet einige Beispiele mit Matlab.

Abgabe und Wettbewerb: Freitag, 18. März 2016

Am Tag des Wettbewerbs werden wir mehrere neue Gruppenfotos unter ähnlichen Beleuchtungsverhältnissen von uns machen und damit unsere Programme testen. Wer am meisten Gesichter erkennen und identifizieren kann, hat gewonnen!

12 Bild- und Videokompression



kompression: Wohl kaum ein anderes Gebiet aus der Bildverarbeitung ist unbewusst so omnipräsent wie die Bild- und Videokompression. Durch die massenhafte Übertragung von digitalen Daten über das Internet ist der Bedarf an Datenreduktion in gleichem Masse gestiegen. Durch die Bilder in den Webseiten und durch die Dienste wie Youtube machen Bild- und Videodaten den Grossteil des übertragenen Datenvolumens aus. Wir tun also gut daran, wenn wir die Redundanz in der Information in diesen Daten auf ein Minimum reduzieren können. Bevor wir damit beginnen, müssen wir einige Grundlagen aus der Informationstheorie verstehen. Erst damit können wir Bilder entweder mit oder ohne Verlust an Information komprimieren.

12.1 Signalttheorie

12.1.1 Digitalisierung des Bildsignals

Das Signal, das z. B. einen CCD-Chip verlässt, ist ein analoges Signal und muss zuerst digitalisiert werden, bevor es im Computer verarbeitet werden kann.

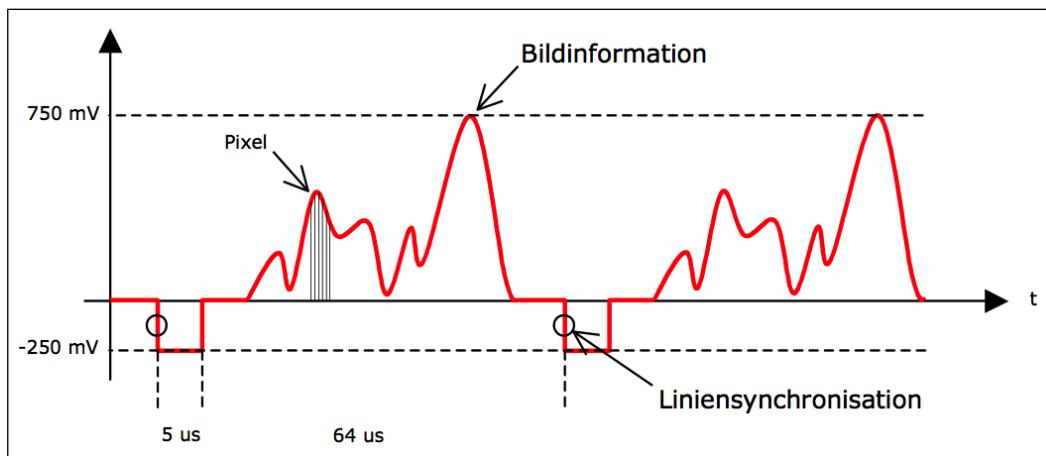


Abb. 330: Bildsignal beim Verlassen des CCD

Dabei stellt sich die Frage, wie viel Information wir benötigen oder wie fein ein analoges Signal abgetastet (*Sampling Rate*) werden soll. Das menschliche Auge hat ein Auflösungsvermögen von etwa 0.075 mm auf einer Distanz von 25 cm. D. h., ein Bild von 7.5 x 7.5 cm in einer Distanz von 25 cm hat eine Auflösung von 1000 x 1000 Pixeln.

12.1.1.1 Nyquist-Frequenz

Die *Nyquist-Frequenz*, auch als *Nyquist-Grenze* bezeichnet, ist definiert als die halbe Abtastfrequenz:

$$f_{nyquist} = \frac{1}{2} f_{abtast}$$

Der Begriff wurde durch *Claude Elwood Shannon* geprägt und nach *Harry Nyquist* benannt. Nach dem zugrundeliegenden *Shannon-Abtasttheorem* müssen alle Anteile in einem Signal kleinere Frequenzen als die Nyquist-Frequenz haben, damit das abgetastete Signal genau rekonstruiert werden kann.

12.1.1.2 Shannon Theorem

Das Abtasttheorem von *Shannon* besagt also, dass die Abtastfrequenz mindestens doppelt so gross sein muss wie die höchste Frequenz im abzutastenden Signal.

$$f_{\text{abtast}} > 2 f_{\text{signal}}$$

Ist dies nicht gewährleistet, so spricht man von Unterabtastung. Die dabei entstehenden, falschen Frequenzen (*Aliasing, Moiréeffekte*), die mit dem Original nichts mehr zu tun haben. Die sogenannte Aliasing-Frequenz berechnete sich aus der Differenz der Signalfrequenz und der Abtastfrequenz:

$$f_{\text{aliasing}} = |f_{\text{signal}} - f_{\text{abtast}}|$$

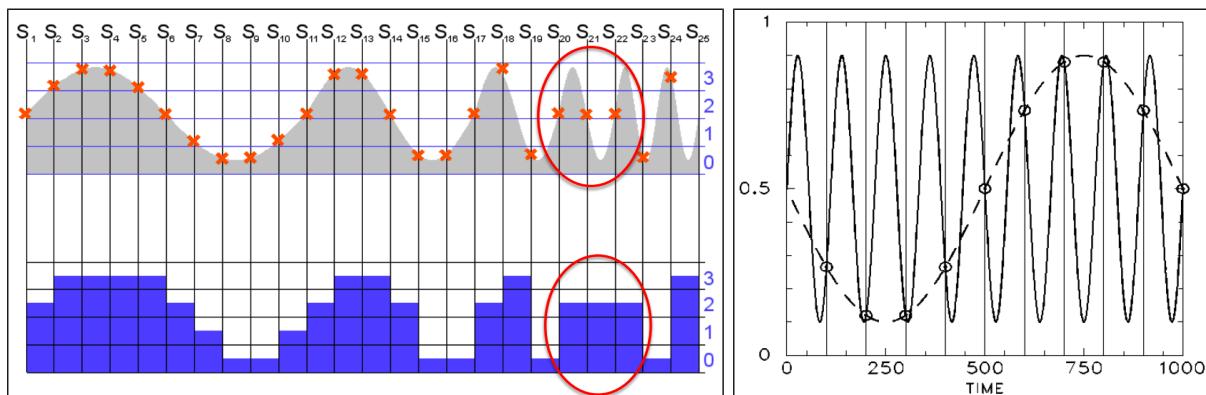


Abb. 331: Wird ein Signal weniger als die doppelte Signalfrequenz abgetastet, so entstehen Aliasing-Frequenzen.

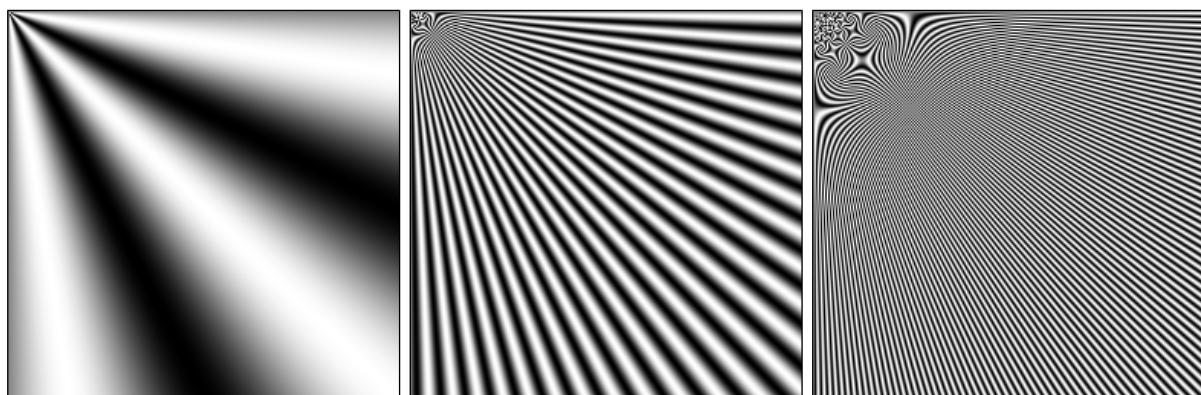


Abb. 332: Aliasing-Effekte in der oberen linken Ecke, sobald die Bildauflösung zu klein wird.

12.1.2 Informationstheorie

Wir haben die statistische Kennzahl der Entropie bereits im Kapitel 4.2.9 kennengelernt. Die Entropie ist eine wichtige Kennzahl aus der Informationstheorie, die 1949 vom amerikanischen Mathematiker *Claude Elwood Shannon (1916-2001)* begründet wurde. Sie untersucht die Darstellung, die Speicherung und die Übertragung von Information. Die Informationstheorie versteht unter der Information (Abk. *I*) ein rein technisches Mass für die formal darstellbaren Aspekte der Information ohne Aussagen zum Sinngehalt. Zur Untersuchung von Nachrichten und deren Übertragung werden vorwiegend Methoden aus der Wahrscheinlichkeitsrechnung und der mathematischen Statistik verwendet.

- Der **statistische Informationsgehalt I_z** eines Zeichens z in einer Nachricht:

$$I_z = -ld(p_z) \quad [\text{bit}]$$

Darin ist p_z die Wahrscheinlichkeit (= Anzahl der Zeichen z / tot. Anzahl Zeichen), mit der z in der Nachricht auftritt. Der Logarithmus Dualis berechnet sich wie folgt:

$$ld(a) = \log_2(a) = \frac{\log_{10}(a)}{\log_{10}(2)} = \frac{\log_{10}(a)}{0.30103}$$

- Der **mittlere Informationsgehalt H** pro Zeichen (*Entropie*) einer Nachricht:

$$H = \sum_{z=1}^n p_z \cdot I_z = \sum_{z=1}^n p_z \cdot -\ln(p_z) \quad [\text{bit}/\text{Zeichen}]$$

- Der **gesamte Informationsgehalt I_{tot}** einer Nachricht:

$$I_{tot} = n \cdot H \quad [\text{bit}] \quad \text{mit } n = \text{total Anzahl Zeichen}$$

Die Einheit des Informationsgehaltes heisst **bit** (*basic indissoluble information unit*). Das ist leicht zu verwechseln mit der Einheit für die Darstellung von Daten mit Hilfe binärer Zeichen **Bit** (*binary digit*). Es gibt im Gegensatz zu bit nur ganzzahlige Bit. Es besteht folgender Zusammenhang: Zur Darstellung von n bit werden mindestens n Bit benötigt.

Demnach ist der Informationsgehalt einer Nachricht umso höher, je unwahrscheinlicher deren einzelne Zeichen vorkommen. Damit ist auch wissenschaftlich bewiesen, dass Wiederholungen in einem Text keine neue Information bringen. Die Formeln zeigen ebenfalls, dass der Informationsgehalt einer Nachricht nicht proportional, sondern lediglich logarithmisch steigt.

Beispiel: Informationsgehalt der deutschen Schriftsprache:

Nimmt man an, dass alle 30 Zeichen (29 Buchstaben + Zwischenraum) gleich verteilt sind, so gilt $p=1/30$. Der Informationsgehalt $I = -\ln(1/30) = \ln 30 = 4.9$ bit. Zur binären Darstellung eines Zeichens benötigt man daher mindestens 5 Bit. Diese Betrachtung ist jedoch stark vereinfacht, da die Buchstaben nicht mit gleicher Wahrscheinlichkeit auftreten (Zeichenwahrscheinlichkeit in Prozent):

Zeichen:	a	b	c	d	e	f	g	h	i	j	k	l	m
Deutsch	6.47	1.93	2.68	4.83	17.48	1.65	3.06	4.23	7.73	0.27	1.46	3.49	2.58
Englisch	8.04	1.54	3.06	3.99	12.51	2.30	1.96	5.49	7.26	0.16	0.67	4.14	2.53
Zeichen	n	o	p	q	r	s	t	u	v	w	x	y	z
Deutsch	9.84	2.98	0.96	0.02	7.54	6.83	6.13	4.17	0.94	1.48	0.04	0.08	1.14
Englisch	7.09	7.60	2.00	0.11	6.12	6.54	9.25	2.71	0.99	1.92	0.19	1.73	0.09

Beispiel: Informationsgehalt des Wortes WINTERSEMESTER:

Zeichen	p	I_z	H_z [bit]
W	1/14	3.8	0.27
I	1/14	3.8	0.27
N	1/14	3.8	0.27
T	2/14	2.8	0.40
E	4/14	1.8	0.51
R	2/14	2.8	0.40
S	2/14	2.8	0.40
M	1/14	3.8	0.27
Summe (H)	1.00		2.80

$$I_{tot} = 14 * 2.80 = 39.2 \text{ bit}$$

$$\text{ASCII codiert} = 14 * 8 = 112 \text{ Bit}$$

$$\text{Redundanz} = 112 - 39.2 = 72.8 \text{ bit}$$

Beispiel: Informationsgehalt analoger Signale

Bei analogen Signalen handelt es sich um kontinuierliche Signale, deren Informationsgehalt sich nach der Umwandlung von analog nach digital berechnen lässt. Dazu wird das

Signal in m diskrete und gleich grosse Werte zerlegt. Der Informationsgehalt für das analoge Signal ist dann $I_d(m)$.

Beispiel: Ein Telefongespräch wird digitalisiert, indem die Schallamplituden in 128 Stufen unterteilt werden. Jeder Amplitudenwert kann also durch eine Folge von 7 binären Zeichen dargestellt werden. Dies geschieht 5000-mal pro Sekunde. Eine Nachricht von 1s Dauer enthält somit $I=35000$ bit Information.

12.1.3 Bildkompression = Datenreduktion

Wenn wir einen reinen Text als ASCII-Datei speichern, so belegt jedes gespeicherte Zeichen 8 Bit. Wir könnten auch sagen, es wird mit 8 Bit codiert. Ob dieses Zeichen nun 1000 Mal oder nur ein einziges Mal im Text vorkommt, spielt dabei keine Rolle. Das ist im Hinblick auf den Speicherbedarf einer Datei sicherlich keine effektive Codierungsmethode! Dieser überflüssige Teil der Daten, der keine zusätzliche Information mehr enthält, wird *Redundanz* genannt. Die Verminderung dieser Redundanz, möglichst auf 0, nennt man *Daten-Kompression*. Die maximale Kompression ist dann erreicht, wenn die mittlere Anzahl Bits pro Zeichen **H** (bit) erreicht.

Jede weitergehende Verringerung der Datenmenge, die zu Verlust von Information führt, wobei aber die wesentlichen Teile davon erhalten bleiben, d. h., der visuelle Eindruck des Bildes nicht leidet, bezeichnet man als *Daten-Reduktion*. Diese Unterscheidung wird aber oft nicht gemacht, sondern man spricht einfach von Datenkompression. Wichtig ist der Unterschied, ob bei der Manipulation (Codierung) der Urdaten Informationen verloren gehen oder nicht.

12.2 Verlustfreie Kompression

12.2.1 Run Length Encoding (RLE)

Als Lauflänge (*Run-Length*) einer Farbwertfolge bezeichnet man die Länge, während der die Bildpunkte (*Pixel*) gleiche Farbwerte besitzen. Bei der Lauflängen-Codierung werden nicht nur die Farbwerte der Pixel einer Linie abgespeichert, sondern zusätzlich die Anzahl, wie oft aufeinander folgende Pixel denselben Grauwert aufweisen. Es werden also Zahlenpaare der Form **(n, g)** erzeugt, mit n = Anzahl sich wiederholender Farbwerte g .

Dieses Verfahren lohnt sich für Bilder mit wenigen Farb- oder Graustufen, vor allem für Binärbilder oder Nachrichten in binärer Form. Es lohnt sich nicht für 8-Bit-Bilder oder etwa Textfiles. Bei binären Bildern gibt es ja nur 0 oder 1 Werte, also könnte der Farbwert sogar weggelassen werden, wenn der Start z. B. mit der Anzahl 1er definiert wird. Run Length Encoding war eines der ersten Kompressionsverfahren und wurde unter anderem in den ersten Fax-Standards (1980) eingesetzt.

Beispiel: Dezimalzahl komprimiert

Die Zahl **55'351**₍₁₀₎ = 111000000000111₍₂₎ benötigt 16 Bit zur Darstellung. Lauflängen codiert sieht die Zahl wie folgt aus: **3 10 3** oder 0011 1010 0011. Mit Lauflängen-Codierung reduziert sich der Speicherplatz auf 12 Bit, also um 25%.

Übung: Wie viel Bits braucht folgendes 1-Bild mit Lauflängencodierung?

```
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
```

12.2.2 Huffman-Code

Gemäss Informationstheorie wäre diejenige Codierung die effizienteste, welche die Häufigkeiten der Zeichen im Text berücksichtigt. Wenn ein Zeichen häufig im Text auftritt, sollte ein möglichst kurzer Code dafür gewählt werden. Wenn ein Zeichen hingegen nur ganz selten vorkommt, macht es nichts, wenn sein Code etwas länger ist.

A	.	-
B	-	...
C	-	..-
D	-	..
E	.	
F	..	-.
G	--	.
H	
I	..	
J	.---	
K	-.	
L	.-.	
M	--	
N	-.	
O	---	
P	---	.
Q	---	-
R	.-	
S	...	
T	-	
U	..-	
V-	
W	---	
X	-...-	
Y	-.--	
Z	---.	

Ein Beispiel für eine solche Codierung ist das Morse-Alphabet (1838 von Samuel Morse erfunden, 1791-1872, amerikanischer Maler und Erfinder):

Die Zuordnung mit unterschiedlichen Codelängen hat aber den Nachteil, dass die Eindeutigkeit bei der Decodierung nicht gewährleistet ist. Ein spezielles Verfahren, das die Eindeutigkeit der Zuordnung und eine Minimallänge der codierten Nachricht liefert, ist der Algorithmus von Huffman (David A. Huffman (1925-1999), amerikanischer Computerpionier), welcher zu einer gegebenen Nachricht einen Huffman-Code konstruiert.

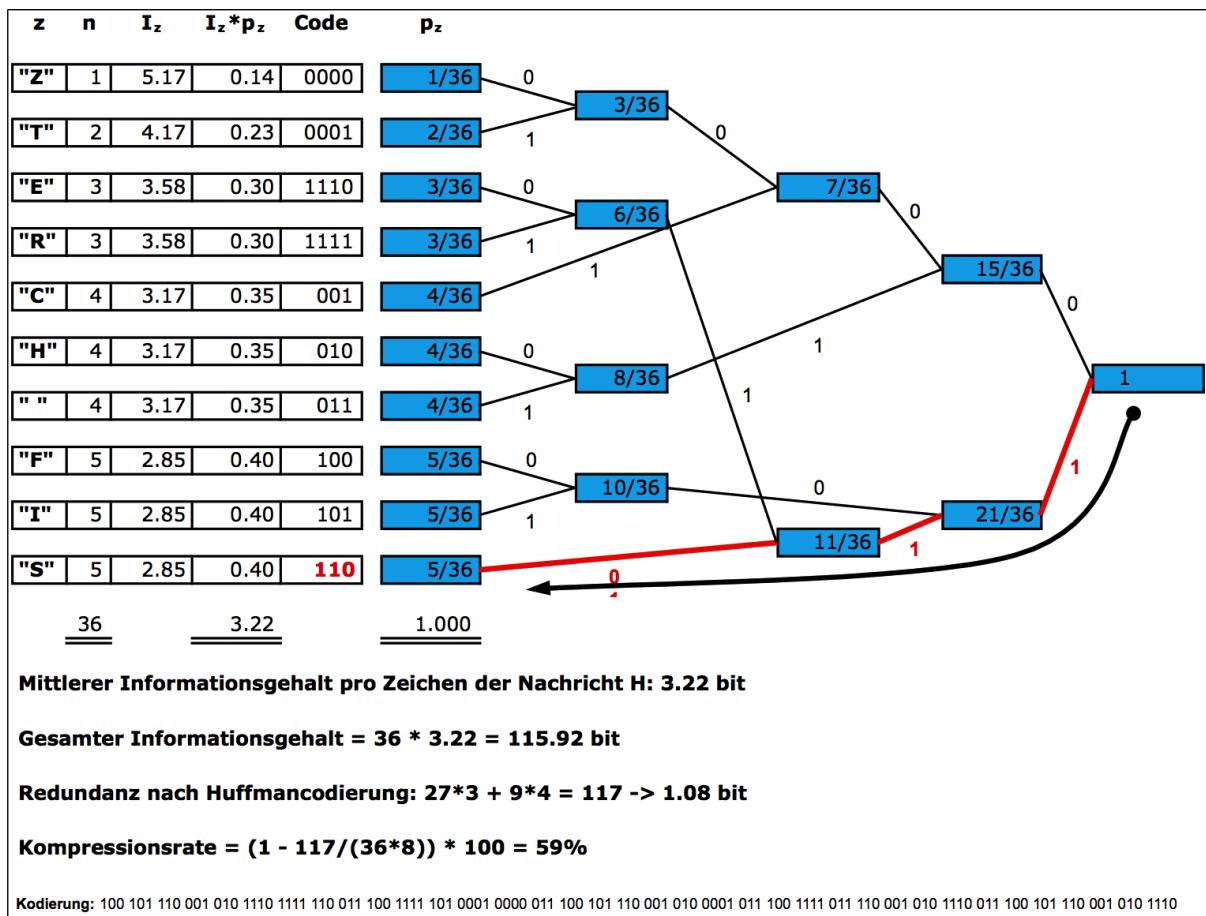
Beispiel:

Berechnen wir den Huffman-Code für folgende Nachricht:

FISCHERS FRITZ FISCHT FRISCHE FISCHE

Vorgehen:

1. Sortieren Sie alle vorkommenden Zeichen z (inkl. Leerschlag) nach ihrer Häufigkeit p_z .
2. Verknüpfen Sie, in sich wiederholenden Schritten, die jeweils kleinsten beiden unverknüpften Häufigkeiten zu einem neuen Knoten und notieren Sie dort die Summe beider Häufigkeiten. Ist ein neuer Knoten gleich wahrscheinlich wie eine noch nicht verknüpfte Wahrscheinlichkeit, so verknüpfen Sie zuerst die noch nicht verknüpfte Wahrscheinlichkeit. Wiederholen Sie dies so lange, bis am Schluss der Wurzelknoten mit der Häufigkeit 1.0 resultiert.
3. Bestimmen Sie nun den Code jedes Zeichens, indem Sie, vom Wurzelknoten ausgehend, bei jedem Ast zur grösseren Häufigkeit eine 1 und zu kleineren Häufigkeit eine 0 notieren. Der Code eines Zeichens ist die Aneinanderreichung aller Nullen und Einsen seiner Äste.
4. Berechnen Sie den Informationsgehalt jedes Zeichens I_z .
5. Berechnen Sie den mittleren Informationsgehalt pro Zeichen der ganzen Nachricht.
6. Berechnen Sie den gesamten Informationsgehalt der Nachricht.
7. Berechnen Sie die Redundanz nach der Huffman-Codierung.
8. Berechnen Sie die Kompressionsrate.



Decodierung - einfach dank Code-Baum

Warum verwendet der Huffman-Algorithmus einen Code-Baum? Der Grund dafür ist die effiziente Decodierung. Will man einen Binärstring anhand eines Code-Baumes decodieren, geht man analog zur oben beschriebenen Codewort-Erzeugung vor.

Etwas technischer ausgedrückt garantiert der Code-Baum, dass der daraus resultierende Code präfixfrei ist. Das heisst, kein Codewort ist der Anfang eines anderen Codewortes. Wäre dem nicht so, könnte man nicht so einfach wie oben beschrieben decodieren: man wüsste nicht, ob man in einem Knoten oder schon in einem Blatt ist!

Nachteile vom Huffman Encoding

- Der Huffman-Baum muss mit der Datei abgespeichert werden.
- Die Quelldatei muss zweimal eingelesen werden: Einmal um die Häufigkeiten der Zeichen auszuzählen und einen Baum zu bilden und ein zweites Mal für die eigentliche Codierung. Das schliesst Huffman zur Echtzeitcodierung Datenströmen aus. Dieses Problem kann aber gelöst werden, indem man Baum und Codes erst während der Kompression erstellt: Man startet zuerst mit einem neutralen Baum und passt ihn mit nach und nach besser an die Gegebenheiten der Eingangsdaten an. Dieses Verfahren heisst *dynamischer Huffman-Code*. Allerdings geht das zulasten der Kompressionsrate, da sich der Encoder erst nach und nach an die Daten gewöhnen muss und am Anfang daher noch gar nicht komprimieren kann.
- Der Wirkungsgrad ist bei Huffman nur dann 100%, wenn die Zeichenhäufigkeiten Zweierpotenzen sind. Das wäre ein Text, in dem ein Buchstabe einmal vorkommt, ein anderer zweimal, ein Dritter viermal, der nächste achtmal, usw. Das ist natürlich völlige Utopie, weswegen Huffman praktisch nie die Maximalleistung erreichen wird. Ein Beispiel für nicht optimale Huffman-Codes wären die Huffman-Morse-Codes. Generieren wir entsprechend der Buchstabenhäufigkeit ein Morse-Alphabet aus Punkten und Strichen, so bekommen seltene Buchstaben Codelängen bis 11 Bit:

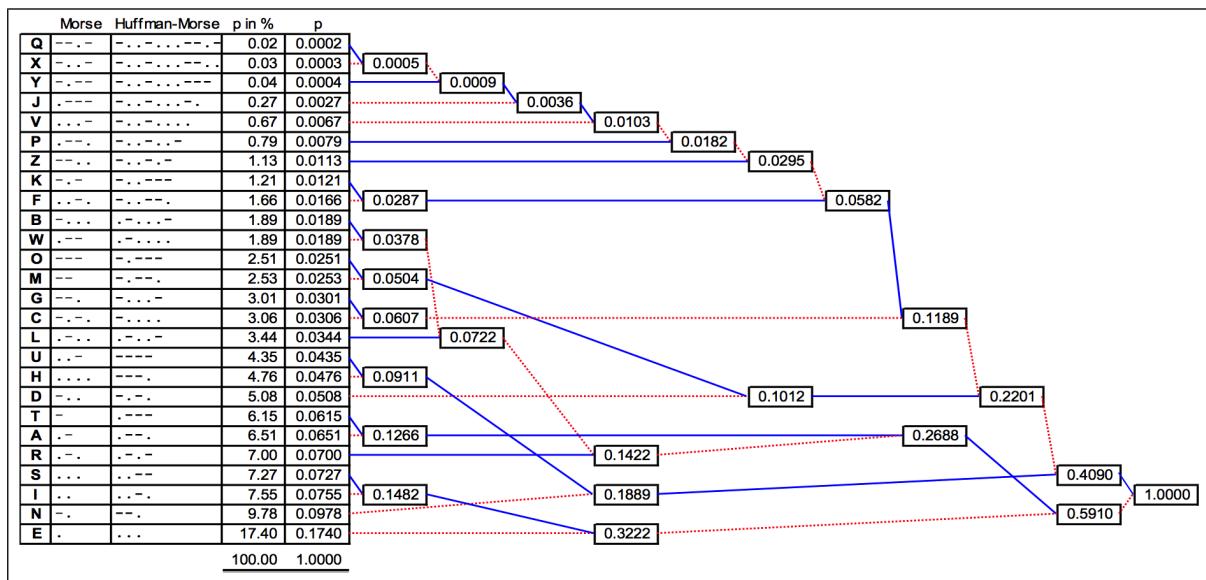


Abb. 333: Huffman-Code des Morse-Alphabets

Diese Nachteile haben dazu geführt, dass sich die Huffman-Codierung als primärer Kompressionsalgorithmus nicht durchgesetzt hat. Es wird aber an manchen Orten als sekundäres Verfahren eingesetzt, wie z. B. im JPEG-Format oder in neuen FAX-Standards.

12.2.3 LZ-Kodierung

Einer der am häufigsten verwendeten Algorithmen zur verlustfreien Kompression ist der LZ-Algorithmus. Die bekanntesten Anwendungen sind die Formate [GIF](#), [TIFF](#) und [PostScript](#) sowie der Kompressionsstandard [V.42bis](#). Die LZ-Kodierung wurde 1977 von [Abraham Lempel](#) und [Jakob Ziv](#) entwickelt und ist als LZ77- bzw. LZ78 Algorithmus bekannt. 1983 modifizierte [Terry Welch](#) den LZ78 Algorithmus für den Einsatz in High-Performance-Disk-Controllern. Das Ergebnis war der [LZW](#)-Algorithmus, den heute bekanntesten LZ-Kompressor. Terry Welch lässt den Algorithmus für [UNISYS](#) patentieren.

Compuserve, die das freie [GIF](#)-Format erfunden hat, merkt anfänglich gar nicht, dass es einen geschützten Algorithmus verwendet. Erst 1993 merkt [UNISYS](#) das und fordert seither Lizenzgebühren für den [LZW](#)-Algorithmus. Das Patent lief bis 2004 weltweit aus und wurde zum Paradebeispiel für die Problematik von Software- und Algorithmenpatente. Obwohl der nachfolgend erläuterte Algorithmus eine enorme erfinderische Leistung darstellt, bleibt deren Patentierung fragwürdig, wenn das Patentamt nicht in der Lage ist, die wahre Leistung den wahren Erfindern zuzuordnen. Diese liegt im [LZW](#) Fall eindeutig bei L. und Z. und nicht bei W.

Der Algorithmus erstellt ein [Dictionary](#) (Wörterbuch) aus den Daten, die im unkomprimierten Datenstrom vorkommen. Muster (zusammenhängende Bytes) in diesem Strom werden in ein Dictionary verfrachtet und durch Verweise auf den jeweiligen Eintrag ersetzt. Dieses Wörterbuch wird on-the-fly aufgebaut, und zwar sowohl vom Encoder als auch vom Decoder. Es braucht deshalb nicht mit abgespeichert zu werden!

Der erste Dictionary Eintrag erhält den Index 256. Die Indizes 0–255 sind reserviert für die Zeichen, aus denen die Muster aufgebaut sein können. Ist die zu komprimierende Datei keinem bestimmten Format zugeordnet, so sind dies einfach alle möglichen Zahlen oder ASCII Zeichen eines Bytes.

Beispiel: LZ-Kodierung folgender Information:**BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH (39 Bytes)**

1. Zu Beginn enthält das Wörterbuch für jedes im Eingabedatenstrom vorkommende Zeichen einen Eintrag, das Präfix ist das erste Zeichen.
2. Zeichen = nächstes Zeichen aus dem Eingabedatenstrom
3. Ist Präfix + Zeichen im Wörterbuch?
 - JA: Präfix = Präfix + Zeichen
 - NEIN: 1. Gib den Code für Präfix aus.
2. Trage Präfix + Zeichen in das Wörterbuch ein.
3. Präfix = Zeichen
4. Ist das Ende des Eingabedatenstromes erreicht?
NEIN: Gehe zu Schritt 2
JA: Wenn Präfix nicht leer ist, gib seinen korrespondierenden Code aus.

Dictionary	Input	Output	(ASCII)
256 BL	BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	66	B
257 LA	LAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	76	L
258 AH	AH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	65	A
259 H_	H_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	72	H
260 _B	_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	95	_
261 BLA	BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	256	BL
262 AH_	AH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	258	AH
263 _BL	_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	260	_B
264 LAH	LAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	257	LA
265 H_B	H_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH	259	H_
266 BLAH	BLAH_BLAH_BLAH_BLAH_BLAH	261	BLA
267 H_BL	H_BLAH_BLAH_BLAH_BLAH	265	H_B
268 LAH_	LAH_BLAH_BLAH_BLAH	264	LAH
269 _BLA	_BLAH_BLAH_BLAH	263	_BL
270 AH_B	AH_BLAH_BLAH	262	AH_
271 BLAH_	BLAH_BLAH	266	BLAH
272 _BLAH	_BLAH	269	_BLA
		72	H

Unkomprimiert: 39 x 8 Bit = 312 Bits

Komprimiert: 18 x 9 Bit = 162 Bits -> Reduktion = 48%

Wenn man den ursprünglichen Text (39 Bytes) mit ZIP komprimiert, wird das File sogar 132 Bytes gross, was auf den Overhead des ZIP Formats zurückzuführen ist. Speichert man hingegen ein 4000 Byte grosses BLAH_BLAH, schafft die ZIP-Kompression eine Reduktion um 96% auf 155 Bytes!

Die Grösse des Wörterbuchs hängt von der Bitbreite eines einzelnen Eintrags ab. Im obigen Beispiel könnten so $2^9 = 512$ Einträge gespeichert werden. In klassischen LZ-Kompressoren beträgt die Bitbreite zwischen 10 und 13 Bit. Natürlich kann auch dann das Wörterbuch noch überlaufen. Bei GIF (LZW) wird in diesem Fall das Wörterbuch geleert und mit einem neuen weitergemacht.

LZ-Dekodierung

Analog zur Komprimierung können bei der Dekomprimierung fortlaufend die Dictionary Einträge gebildet werden. Bei der Dekodierung wird der jeweils nächste Wörterbucheintrag aus dem zuletzt übersetzten Code und dem angehängten Zeichen gebildet.

1. Zu Beginn enthält das Wörterbuch für jedes im Eingabedatenstrom vorkommende Zeichen einen Eintrag.
2. Code = erster Code aus dem Eingabedatenstrom (immer ein Zeichen)
3. Gib Code aus.

4. Merke Code in alterCode
5. Code = nächster Code aus dem Eingabedatenstrom
6. Ist Code im Wörterbuch?
 - JA: Gib Code aus.
Präfix = alterCode
Zeichen = erstes Zeichen von Code
Trage Präfix + Zeichen in das Wörterbuch ein.
 - NEIN: Präfix = alterCode
Zeichen = erstes Zeichen von alterCode
Trage Präfix + Zeichen (= Code) in das Wörterbuch ein UND gib es aus.
7. Wenn das Ende des Eingabedatenstromes noch nicht erreicht ist, gehe zu Schritt 4.

Beispiel 1: BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH_BLAH

Input	Output	Dictionary
66	B	
76	L	256 BL
65	A	257 LA
72	H	258 AH
95	_	259 H_
256	BL	260 _B
258	AH	261 BLA
260	_B	262 AH_
257	LA	263 _BL
259	H_	264 LAH
261	BLA	265 H_B
265	H_B	266 BLAH
264	LAH	267 H_BL
263	_BL	268 LAH_
262	AH_	269 _BLA
266	BLAH	270 AH_B
269	_BLA	271 BLAH_
72	H	272 _BLAH

Beispiel 2: OLEOLEOLEOLEOLEOLEFCB

Codes	Output	Dictionary
79	O	
76	L	256 OL
69	E	257 LE
256	OL	258 EO
258	EO	259 OLE
257	LE	260 EOL
259	OLE	261 LEO
262	OLEO	262 OLEO <<< Schritt 6.NEIN !
257	LE	263 OLEOL
70	F	264 LEF
76	C	265 FC
66	B	266 CB

12.2.4 Übungen verlustfreie Kompression

1. Was ist der Vorteil dieses Huffman-Morse Codes gegenüber dem ursprünglichen Morse Code und erklären Sie warum?
2. Wie hoch ist der gesamte Informationsgehalt der nachfolgenden Nachricht auf 3 Nachkommastellen gerundet: FRISCHE_FISCHE_ZISCHEN
3. Wie hoch ist die Redundanz in dieser Nachricht von 2.?
4. Folgende Nachricht ist LZ komprimiert (Zeichen der ASCII-Werte in Klammern): 77(M),73(I),78(N),85(U),83(S),83,257,259,256,258,260,262,83,264,259,261,265
Wie lautet die Nachricht dekomprimiert?

12.3 Verlustbehaftete Kompression

Unter verlustbehafteter Kompression versteht man nicht zwangsläufig, dass das decodierte Bild Qualität verloren hat (was natürlich bei sehr hohen Kompressionsraten nicht ausbleibt), sondern vielmehr, dass Informationsgehalt bei der Kompression verloren geht.

Informationsverlust kann akzeptabel sein wenn:

- Die verlorene Information für die Wahrnehmung irrelevant ist, d.h., wir merken den Verlust gar nicht.
- Die Vorteile der geringeren Dateigröße bei der Speicherung oder Übermittlung, die Nachteile einer schlechteren Qualität überwiegt.

Bei beiden Punkten muss angemerkt werden, dass sie sehr subjektiv beurteilt werden.

12.3.1 JPEG-Kompression

Das **JPEG** ist das am meisten verbreitete Bildformat mit verlustbehafteter (*lossy*) Kompression. Es wurde von der *Joint Photographic Expert Group* Ende der Achtzigerjahre entwickelt und erlaubt sehr hohe Kompressionsraten bei Bildern mit hoher Farbtiefe. Im JPEG-Standard wurden vier verschiedene Modi definiert:

- **Sequentielle Kodierung:** In diesem Modus wird das Bild in einem einzigen Durchgang von links oben nach rechts unten kodiert. Von allen Modi liefert diese Kodierung die beste Kompressionsrate und wird am meisten eingesetzt.
- **Progressive Kodierung:** Dieser Modus erlaubt einen Bildaufbau mit sukzessiv wachsender Qualität. Zuerst werden die Gleichanteile aller Blöcke, dann die niederfrequenten Koeffizienten und zuletzt die hochfrequenten Werte übertragen. Dem Empfänger ist es daher möglich, das Bild schrittweise genauer zu rekonstruieren.
- **Hierarchisch-progressive Kodierung:** Am Anfang des Bildaufbaus sind starke Blockeffekte bei der Verwendung des progressiven Modus zu erkennen. Dies soll in diesem Modus verhindert werden, indem in einem ersten Schritt das Bild um den Faktor 16 verkleinert wird. Danach wird es im progressiven Verfahren kodiert und übertragen. Die Übertragungsrate steigt gegenüber dem progressiven Modus um bis zu 30 Prozent.
- **Verlustfreie Kodierung:** In diesem Modus wird eine verlustfreie Entropiecodierung vorgenommen und liefert deshalb nur eine Kompressionsrate von etwa 50%.

Nachfolgend beschränken wir uns auf die Erklärung der sequenziellen Codierung, die aus folgenden fünf Schritten besteht:

Schritt 1: Konvertierung ins YCbCr-Farbmodell

Schritt 2: Diskrete Kosinus Transformation

Schritt 3: Quantisierung der DCT-Koeffizienten

Schritt 4: Zig-Zag Kodierung der Koeffizienten

Schritt 5: Kompression der Koeffizienten mit Lauflängen- und Entropiecodierung

12.3.1.1 Schritt 1: Konvertierung ins YCrCb-Format

Da das menschliche Auge Helligkeitsunterschiede besser erkennen kann als Farbveränderungen, konvertiert man die Bilddaten zuerst in das YCbCr-Farbmodell (s. Kapitel 2.2.5).

Mit der YCbCr-Transformation hat noch keine Kompression stattgefunden. Abgesehen von den allgegenwärtigen Rundungsfehlern sind noch keine Verluste aufgetreten. Ihren eigentlichen Sinn erfüllt die Transformation erst, wenn man die drei Kanäle unterschiedlich stark abtastet. Der Grauwertkanal Y wird in voller Auflösung durch die nachfolgenden Schritte geschickt. Bei den Farbkanälen Cr und Cb werden aber jeweils 2 oder 4 Pixel gemittelt, bevor sie durch die gleichen Schritte weiter verarbeitet werden. Dadurch ent-

steht bereits eine grosse Datenreduktion, die allerdings für das Auge kaum wahrnehmbar ist. Wie genau die Pixel abgetastet werden, wird mit der sogenannten A:B:C-Notation beschrieben:

- 4:4:4 Chrominanz identisch zur Luminanz (D1)
- 4:2:2 Chrominanz horizontal halbiert, vertikal unverändert (Digital Beta)
- 4:2:0 Chrominanz horizontal und vertikal halbiert (JPEG, DV(PAL), MPEG-1/2)
- 4:1:1 Chrominanz horizontal geviertelt, vertikal unverändert (DVCPRO, DV(NTSC))

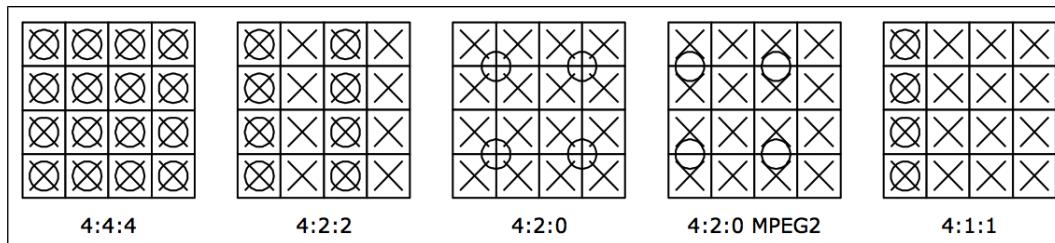


Abb. 334: Abtastvarianten bei YCrCb-Umwandlung

12.3.1.2 Schritt 2: Diskrete Kosinustransformation

Die *Diskrete Kosinustransformation (DCT)* der JPEG-Kompression ist verwandt mit der Fourier-Transformation und zerlegt ein Signal in 64 Kosinuskoefizienten (Frequenzanteile). Die Umkehrung dieser Transformation wird *Invers diskrete Kosinustransformation (IDCT)* genannt und erzeugt aus den Koeffizienten wieder das ursprüngliche Signal im Ortsraum. Man entschied sich für diese, auf Kosinusfunktionen beschränkte Version, aus Performancegründen um die Berechnung des Terms $e^{ik\pi n/N}$ der DFT zu vermeiden. Eine sehr gute Beschreibung der mathematischen Hintergründe der DCT finden sie auf der [Website](#) einer Studentenarbeit.

Diskrete Kosinustransformation (DCT):

$$F(u, v) = \frac{1}{4} \cdot c_u c_v \cdot \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cdot \cos \frac{(2x+1)u\pi}{16} \cdot \cos \frac{(2y+1)v\pi}{16}$$

mit: $c_u, c_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } u, v = 0 \\ 1 & \text{sonst} \end{cases}$

Inverse diskrete Kosinustransformation (IDCT) mit gleichen Regeln für c_v und c_u :

$$f(x, y) = \frac{1}{4} \cdot \sum_{u=0}^7 \sum_{v=0}^7 c_u c_v \cdot F(u, v) \cdot \cos \frac{(2x+1)u\pi}{16} \cdot \cos \frac{(2y+1)v\pi}{16}$$

Die DCT ist leider aber wenig rechnerfreundlich, da ihr Aufwand überproportional ($n \cdot \log(n)$) mit der Bildgrösse wächst. JPEG behandelt das Bild deswegen nicht als Ganzes, sondern in Blöcken aus 8x8 Pixeln, damit bei grossen Bildern die Rechenzeiten nicht zu stark anwachsen.

Ein anderer Grund sich auf ein "Fenster" innerhalb des Bildes zu beschränken wurde bereits im Kapitel über die Wavelet-Transformation erläutert. Ein Nachteil der DFT und somit auch der DCT ist deren Unvermögen ein Signal in einem breiten Zeitabschnitt (= breite Bildzeile) gleichzeitig in tiefe wie hohe Frequenzen zu zerlegen. Die DCT beschränkt sich deshalb auf ein relativ kleines Fenster, um hohe Frequenzen mit möglichst wenigen Kosinustermen beschreiben zu können.

Beispiel: 8x8 Grauwerte:

$$f(x,y) = \begin{bmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 162 & 162 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{bmatrix}$$

Wir erhalten dadurch folgende 8x8-Integer-Matrix:

$$F(u,v) = \begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -17 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 0 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -1 & 0 \end{bmatrix}$$

Das Element $F(0,0)$ ist proportional zum mittleren Grauwert des Blockes, es wird auch DC-Term (*Direct Current*) genannt. Alle übrigen Koeffizienten repräsentieren die Frequenzanteile in aufsteigender Reihenfolge von oben links nach unten rechts. Sie werden AC-Koeffizienten (*Alternate Current*) genannt. Bis hierhin ist das Verfahren noch verlustfrei, wenn man von den Rundungsfehlern absieht.

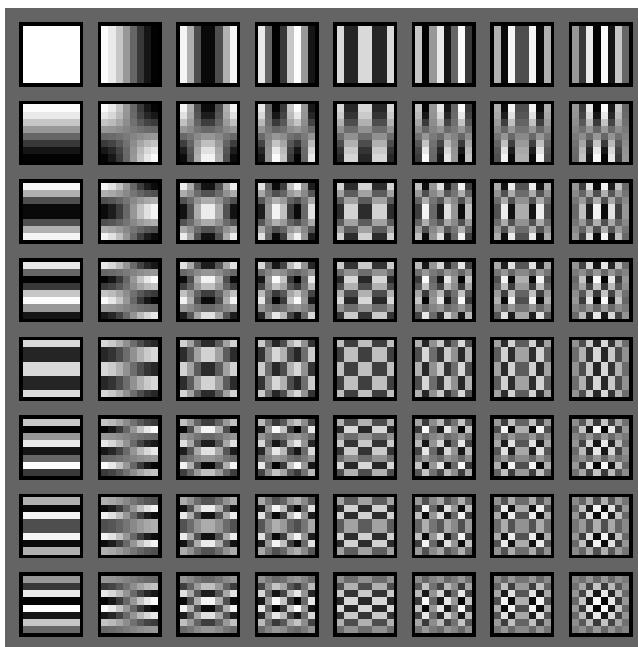
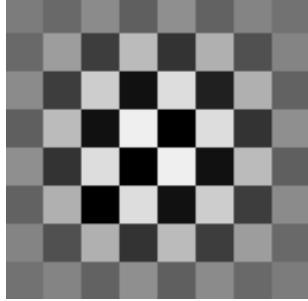


Abb. 335: Darstellung der 64 Basisfunktionen (Kosinusfrequenzanteile) der DCT

In Abb. 153 sehen Sie die Visualisierung der 64 Kosinusfrequenzanteile. Die Darstellung einer einzelnen Frequenz erhält man, indem man nur dessen Frequenzanteil mit der IDCT zurücktransformiert und als Graustufenbild darstellt. Würde man z. B. folgendes Spektrum im Frequenzraum:

$$F(u, v) = \begin{bmatrix} 950 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 500 \end{bmatrix}$$

durch die IDCT lassen, so erhielt man folgendes Ausgangsbild im Ortsraum:

$$f(x, y) = \begin{bmatrix} 124 & 105 & 139 & 95 & 143 & 98 & 132 & 114 \\ 105 & 157 & 61 & 187 & 51 & 176 & 80 & 132 \\ 139 & 61 & 205 & 17 & 221 & 32 & 176 & 98 \\ 95 & 187 & 17 & 239 & -1 & 221 & 51 & 143 \\ 143 & 51 & 221 & -1 & 239 & 17 & 187 & 95 \\ 98 & 176 & 32 & 221 & 17 & 205 & 61 & 139 \\ 132 & 80 & 176 & 51 & 187 & 61 & 157 & 105 \\ 114 & 132 & 98 & 143 & 95 & 139 & 105 & 124 \end{bmatrix}$$


12.3.1.3 Schritt 3: Quantisierung der DCT-Koeffizienten

Im eigentlich verlustbehafteten Prozess der Quantisierung werden die 64 DCT-Koeffizienten nun durch eine Quantisierungstabelle $Q(u,v)$ dividiert und das Ergebnis auf eine ganze Zahl gerundet. Das menschliche Auge reagiert auf niedrige Frequenzen viel empfindlicher als auf hohe. Die Koeffizienten höherer Frequenzen werden deshalb durch höhere Zahlen geteilt als die der niedrigeren. Hiermit erreicht man, dass sich der Wertebereich verkleinert. Viele Koeffizienten werden dadurch zu Null. Ebenfalls berücksichtigt wird, dass das Auge vertikale Strukturen (= horizontale Frequenzen) besser sieht als horizontale. In der Quantisierungstabelle wird dies berücksichtigt durch asymmetrische Divisoren. Die Symmetriearchse verläuft von oben links nach unten rechts. Die Standard-Quantisierungstabellen für die Luminanz- und Chromianzkanäle sind in der opensource JPEG-Implementation [JPEG](#) wie folgt definiert:

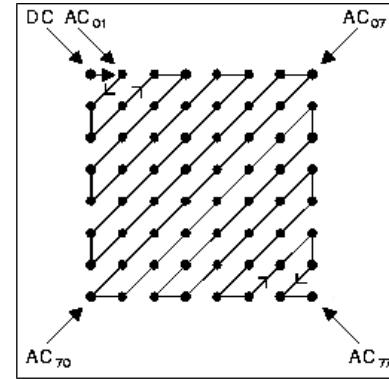
$$Q_{lum}(u, v) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad Q_{chrom}(u, v) = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

$$F(u, v) = \text{round}\left(\frac{F(u, v)}{Q(u, v) \cdot QScale}\right) = \begin{bmatrix} 79 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Da die Art der Quantisierung nicht fest vorgeschrieben ist, kann an dieser Stelle die Qualität des Bildes, durch die Wahl der Quantisierungstabelle mitbestimmt werden. Diese Tatsache bewirkt aber auch, dass die verwendete Quantisierungstabelle mit abgespeichert und dem Decoder mitgeschickt werden muss. Die meisten Encoder verwenden die obige oder eine andere von JPEG vorgeschlagene Tabelle als Basis für 50% Qualität und skalieren die Werte linear (*QScale*). Dass diese Vorgehensweise nicht optimal ist, beweisen Programme wie Adobe Photoshop, die eigene Tabellen verwenden und dazu noch auf das konkrete Bild einmessen, wodurch sie eine viel höhere Qualität bei nur wenig steigender Dateigröße ermöglichen.

12.3.1.4 Schritt 4. Zick-Zack-Codierung der Koeffizienten

Die Codierung der quantisierten Koeffizienten erfolgt getrennt für DC- und AC-Koeffizienten. Aus den 8x8 Blöcken wird eine sequenzielle Zahlenfolge erzeugt. Dabei ist der erste Wert der DC-Koeffizient, allerdings wird nicht der originale Wert, sondern die Differenz zum DC-Koeffizienten im vorhergehenden Block codiert, wodurch auch dieser zu einem kleinen und damit besser komprimierbaren Wert wird. Die nachfolgenden 63 AC-Koeffizienten werden anhand einer Zick-Zack-Kurve in eine sequentielle Reihenfolge gebracht, wodurch eine Sortierung hin zu höheren Frequenzen entsteht. Da aber gerade die hohen Frequenzanteile oft 0 sind, entsteht eine für die weitere Kompression der Bilddaten günstige Reihenfolge. Für unser obiges Beispiel ergibt sich somit folgende Zig-Zag-Reihenfolge:



79, 0, -2, -1, -1, 0, 0, -1 und 55 mal die 0.

12.3.1.5 Schritt 5. Kompression mit Lauflängen- und Entropiecodierung

Abschliessend werden Koeffizienten mit einer Mischung aus RLE und Huffman-Codierung gespeichert.

Die **DC-Terme** werden dabei gesondert behandelt. Weil diese nur relativ gering von Block zu Block ändern, speichert man nur die Differenz zum vorangegangenen Block.

Für die **AC-Terme** wendet man eine Lauflängencodierung an, dass sich aber nur auf die Längen der Nullen bezieht. Kommen am Schluss nur noch Nullen vor, so brauchen wir diese gar nicht mehr zu speichern und beenden den Block mit (0/0). Für unser obiges Beispiel ergibt sich mit einem vorangegangenen DC-Wert von 82 folgende Sequenz:

3,(1,-2),(0,-1),(0,-1),(0,-1),(2,-1),(0,0)
zulesen als: DC(82-97),(1*0,-2),(0*0,-1),(0*0,-1),(2*0,-1),(Ende)

Diese Zahlenfolge wird nun mit einem speziellen Huffman-Code gespeichert. Die eigentlichen Werte ohne die Nullen (3,-2 und -1), werden mit folgender Tabelle, genannt *VLI* (*Variable Length Integer*) kodiert:

VLI Tabelle	
Grösse	Amplitude
0	0
1	-1,1
2	-3,-2,2,3
3	-7...-4,4..7
4	-15...-8,8..15
5	-31...-16,16..31
6	-63...-32,32..63
7	-127...-64,64..127
...	
13	-8181...-4096,4096..8181
14	-16383...-8182,8182..16383
15	-32767...-16384,16384..32767

Die Zahl 50 wird also mit 6 Bits ganz normal als Binärzahl gespeichert: 110010. Ist eine 6-Bit-Zahl kleiner als 32 ($= 2^{(6-1)}$), so wird sie als die negative Zahl seiner Invertierten betrachtet. Die Zahl -50 ist also das invertierte Bitmuster der Zahl 50: 001101. Diese Codierungsart wird auch Zweierkomplement genannt.

Weil die Länge eines *VLI* nicht konstant ist, braucht es vorne dran einen Code für die Länge. Dieser wird *VLC* (*Variable Length Code*) genannt und kommt getrennt für DC- und AC-Terme, aus zwei verschiedenen Tabellen. Für einen DC-Term ist der VLC einfach ein Längencode. Für einen AC-Term ist ein VLC die Kombination aus der Anzahl Nullen (*Run*) und der Längencode für den nachfolgenden VLI:

VLC Tabelle für DC-Terme:

Grösse	Code
0	010
1	011
2	100
3	00
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

VLC Tabelle für AC-Terme:

(RLE0,Grösse)	Code
(0/0)	1010 (EOB = End of Block)
(0/1)	00
(0/2)	01
(0/3)	100
(0/4)	1011
...	
(1/1)	1100
(1/2)	11011
(1/3)	11110001
(1/4)	111110110
...	
(2/1)	11100
(2/2)	11111001
(2/3)	111110111
(2/4)	111111110100
...	

Alles zusammen ergibt sich so eine Huffman-Codeabfolge von (VLC/VLI)-Paaren:

```
(      3),(     1,-2),(0,-1), (0,-1), (0,-1), (   2,-1),( 0,0)
(0111,11),(11011,01),(00,0),(00,0),(00,0),(11100,0),(1010)
```

Ohne die Klammern sind es noch 31 Bits:

```
01111 1101101 000 000 000 111000 1010
```

Von den 64 Bytes oder 512 Bits des 8x8 Graustufenblocks sind also nur noch 31 Bits übrig geblieben. Vernachlässigt man den Overhead der mit abgespeicherten Tabellen, so entspricht dies einer Kompressionsrate von 94%.

12.3.1.6 JPEG Dekompression

Bei der Dekompression werden die Schritte 1 – 5 rückwärts durchgeführt, wobei für die Rücktransformation in den Ortsraum die inverse DCT verwendet wird.

Progressive Dekompression: Der JPEG-Standard definiert einen *sequenziellen* und einen *progressiven* Modus. Das bis jetzt beschriebene Verfahren umschrieb den sequenziellen Modus. Im progressiven Modus werden zuerst die ersten Koeffizienten

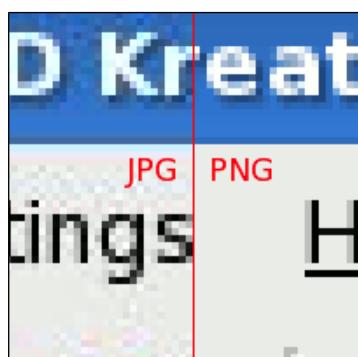
jedes Bildblocks, dann die Zweiten usw. der Reihe nach abgespeichert, sodass die Annäherung an das Originalbild immer besser wird. Wie beim Interlacing, das bei GIF angewendet wird, liegt der Sinn darin, dem Benutzer, noch bevor die gesamte Datei geladen ist, ein grobes Vorschaubild zu geben. Dies ist besonders dann sinnvoll, wenn das Laden eines Bildes länger als eine halbe bis ganze Sekunde dauert. Gewisse Bibliotheken, wie z. B. die Intel JPEG-Library können zudem progressiv gespeicherte Dateien umgekehrt proportional schneller laden in halber, Viertel- oder Achtelauflösung.

Verlustfreie Verarbeitung: Nicht alle Verarbeitungsschritte benötigen eine Dekompression mit der inversen DCT und damit eine wiederholte Kompression mit der DCT. Folgende Operationen können verlustfrei, ohne Dekompression durchgeführt werden:

- Bilddrehung um 90°, 180° und 270°
- Horizontale und vertikale Spiegelung
- Beschneiden um ein Vielfaches von 16 Pixeln

12.3.1.7 Schwächen von JPEG

Die wesentliche Schwäche der JPEG-Kompression ist die Unterteilung in 8x8 Pixel Blöcke.



Die einzelnen Blöcke haben nichts miteinander zu tun, obwohl sie Teil des gesamten Bildes sind. Im Frequenzspektrum werden Bereiche verworfen, die im nächsten Block von Bedeutung sein können. Außerdem wird die Helligkeit pro Block gemittelt, sodass bei höheren Kompressionsraten keine kontinuierlichen Übergänge zwischen den Blöcken vorhanden sind. Die JPEG-Kompression eignet sich damit nur für „natürliche“ Bilder mit vielen verschiedenen Farben. Für synthetische (unnatürliche) Bilder mit wenigen verschiedenen Farben (<=256) ist das PNG oder GIF-Format mit dem LZW-Algorithmus besser geeignet.

Abb. 336: Kompressionsartefakte in einem synthetischen Bild treten v.a. bei starken Kontrasten und feinen Strukturen auf.

Bildvergleich bei unterschiedlichen Kompressionsgraden

Nachfolgend sehen Sie die Kompressionsraten eines 73 x 68 Pixel grossen Bildes mit 24 Bit Farbtiefe bei verschiedenen Kompressionsgraden. Bei Kompressionsgrad 80 erreicht man bereits eine Reduktion um 90%, wobei die Blockartefakte von Auge kaum zu erkennen sind.

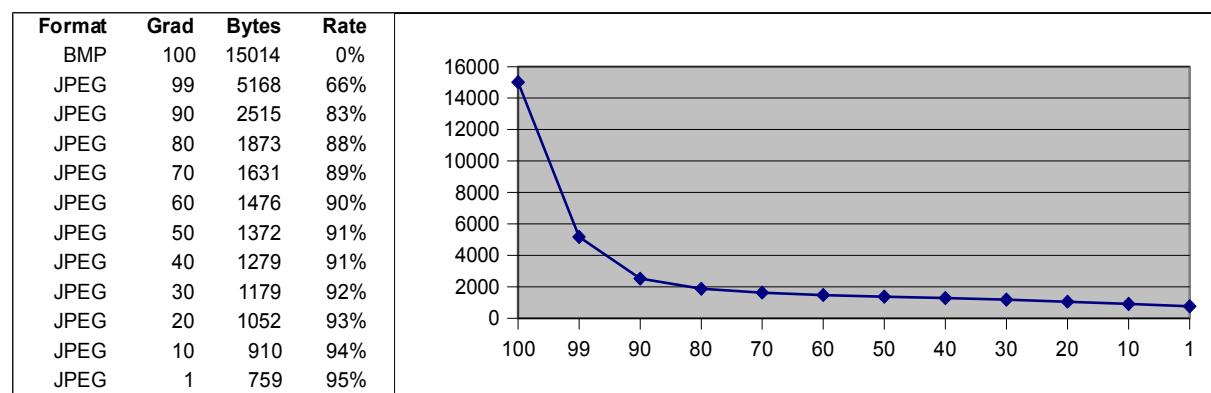


Abb. 337: Kompressionsraten bei verschiedenen JPEG-Kompressionsgraden



Abb. 338: Das Auge der Lena: Links mit JPEG-Kompressionsgrad 99, rechts mit Kompressionsgrad 60

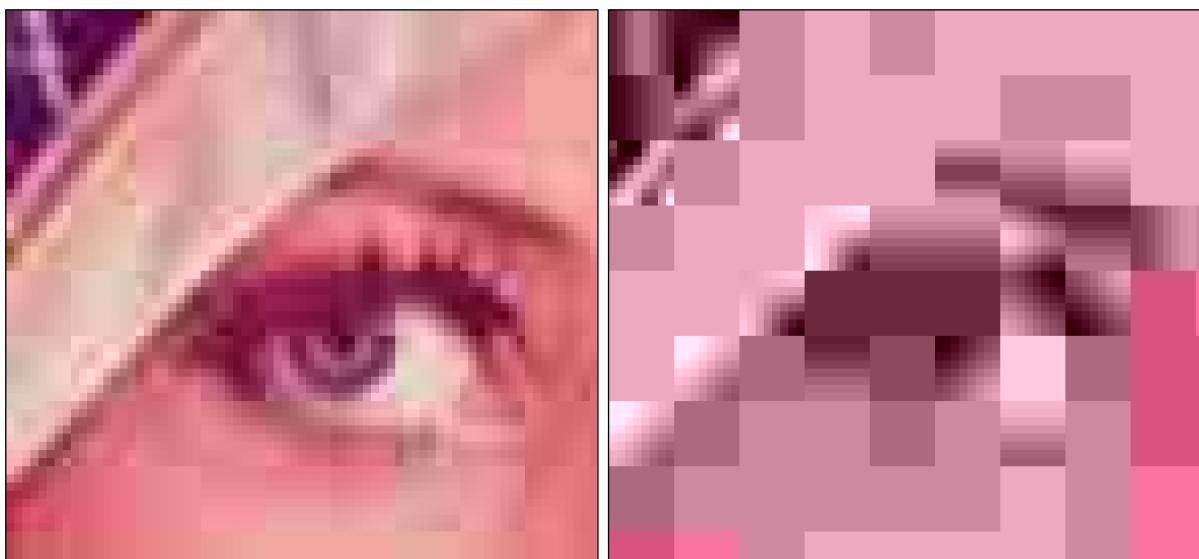


Abb. 339: Das Auge der Lena: Links mit JPEG-Kompressionsgrad 10, rechts mit Kompressionsgrad 1

12.3.2 JPEG2000-Kompression

Obwohl JPEG zweifellos ein gut durchdachtes und effizientes Verfahren ist, stören doch die harten Übergänge an den Blockgrenzen, die vor allem bei höheren Kompressionsraten entstehen. Außerdem verlangt die Industrie nach Methoden, die Kompressionsrate nicht über abstrakte Parameter wie die Quantisierungstabelle, sondern über konkrete Werte wie die Zieldateigröße einzustellen. Als Grundlage für das zukünftige JPEG2000-Format wurde eine Technologie gesucht, die das Bild nicht in kleine Blöcke aufteilt, sondern in seiner Gesamtheit beschreibt. Das vom JPEG-Komitee favorisierte Verfahren auf der Basis von [Wavelet-Transformationen](#) bietet diesen Vorteil. Wir haben diese Transformation und deren Vorteile im Kapitel über globale Operatoren kennengelernt und beschränken uns hier auf die Formataspekte.

JPEG2000 könnte bereits längst zum Standard avanciert sein, wäre die Entwicklung bisher nicht ungünstiger verlaufen als bei JPEG. Statt sich gemeinsam an einen runden Tisch zu setzen, kochte bei den Wavelets zunächst jeder sein eigenes Süppchen. Und somit gibt es auch mehrere kommerzielle auf Wavelets basierende Bildkompressionsverfahren. Zumaldest ist der Standard seit 2001 verabschiedet aber eine frei verfügbare, allgemeingültige Implementation lässt weiterhin auf sich warten.

Kompression

Wie wir gesehen haben, verkleinert die schnelle Wavelet Transformation ein Bild rekursiv durch einen Tiefpassfilter in seine halbe Grösse und speichert die dabei verloren gegangene Information (Hochpassfilter) separat ab. Dies wird solange wiederholt, bis eine max. Rekursionstiefe erreicht ist.

Würden wir denselben 8x8 Datenblock wie beim DCT-Beispiel in JPEG mit einer 2D-HaarTransformation umwandeln, so erhielten wir folgendes Resultat:

$F(u,v)=$	157.5	0.0	-1.7	0.8	-1.3	-0.7	1.6	0.0
	-2.2	-1.6	-1.6	-0.4	-1.1	-0.3	-0.9	0.0
	-3.0	-2.0	-0.8	0.0	-0.6	-0.8	0.0	0.0
	-0.6	0.1	-0.4	0.0	-0.1	0.1	0.0	0.0
	-1.6	-0.8	-0.6	-0.4	0.5	-0.3	-0.8	0.0
	-1.6	-0.2	-2.0	0.1	-0.8	-1.3	0.3	0.0
	-0.4	0.1	-0.5	0.5	-0.3	-0.3	1.0	0.0
	-0.1	0.4	-0.3	0.0	0.0	-0.5	0.0	0.0

Wie nach der DCT erhalten wir oben links den DC. Die anderen Koeffizienten sind nun aber nicht mehr Kosinus-Frequenzanteile, sondern die Differenzen zum Durchschnitt. Bis jetzt wurde nichts komprimiert (abgesehen von Rundungsfehlern), sondern lediglich transformiert. Bei der Detailbildung sind Zahlen mit Nachkommastellen entstanden, die wie bei JPEG ebenfalls durch Ganzzahlenrundung entfernt werden, wodurch ein erster Verlust entsteht.

Analog zum JPEG-Verfahren findet nach der Wavelet-Transformation eine verlustbehaftete Quantisierung und eine anschliessende Entropiecodierung zur Reduktion der Datenmenge statt. Aus lizenzrechtlichen Gründen nutzt JPEG bei der Entropiecodierung die Huffman- und die RLE-Kodierung. Bei JPEG2000 wird eine noch effizientere Implementierung der arithmetischen Codierung verwendet.

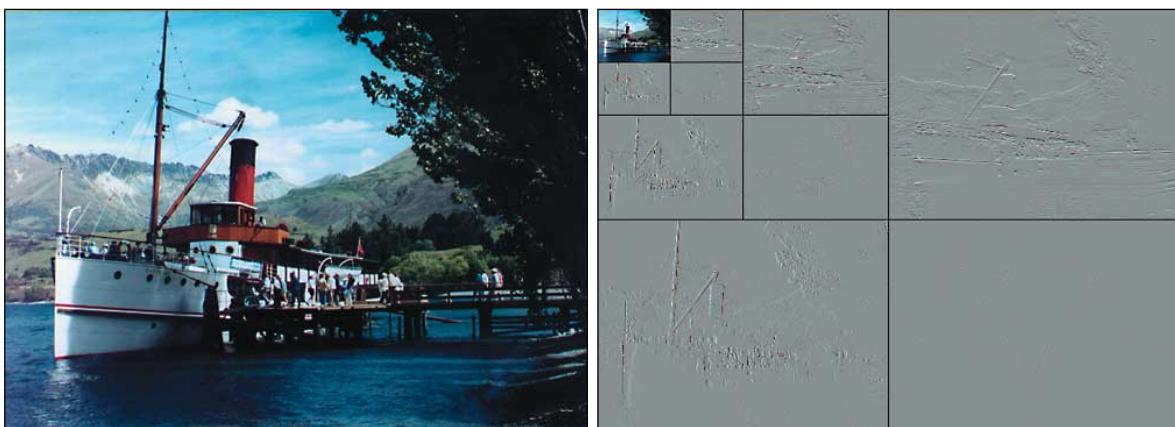


Abb. 340: Originalbild und seine hoch- und tiefpassgefilterten Verkleinerungen [Bertuch01]

Bei JPEG2000 kommt nicht die einfache Haar-Transformation zum Einsatz, sondern ein weicherer Tiefpassfilter, der mehr umliegende Pixel, verschieden stark in die Mittelwertbildung einbezieht. Die am häufigsten verwendete Funktion ist das sogenannte 9/7-tap-Wavelet:

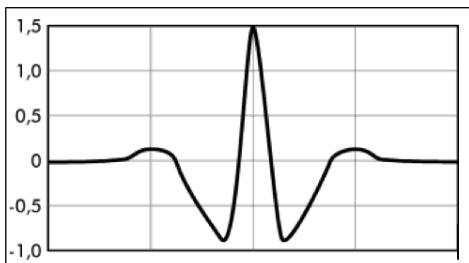


Abb. 341: 9/7-tap-Wavelet [Bertuch01]

Dekompression

Auch die Dekompression der Bilddaten ist bei JPEG2000 stark verbessert: Die inkrementelle progressive Dekompression wird fester Bestandteil von JPEG2000. Das Bild wird von Anfang an komplett angezeigt, lediglich die Anzahl der Details erhöht sich mit jedem Ladeschritt für das gesamte Bild.

Weniger Kompression in Regions of Interest

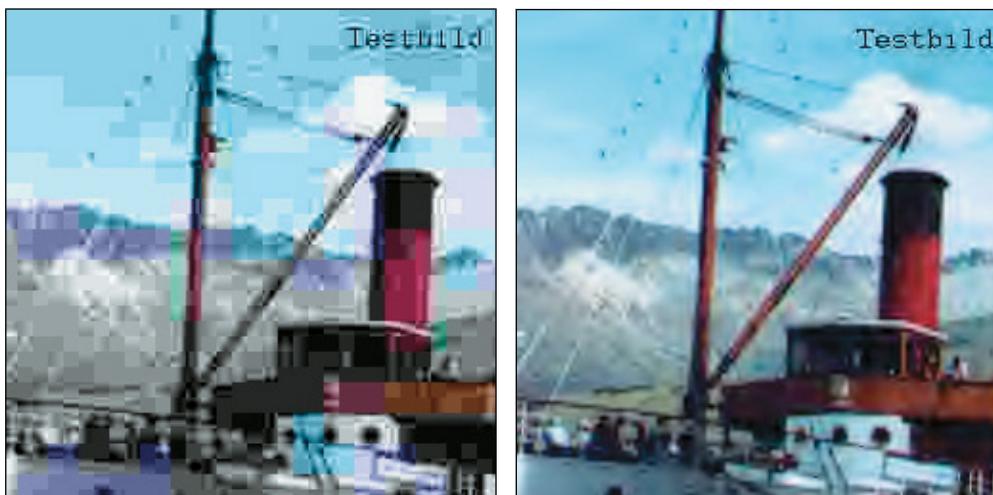
Um das Verhältnis zwischen Dateigröße und Bildqualität weiter zu optimieren, kennt JPEG2000 sogenannte *Regions of Interest (ROI)*. Der Anwender definiert mit der ROI einen Bildteil, der besonders schonend komprimiert wird, und erlaubt gleichzeitig eine stärkere Kompression des restlichen Bildes.



Abb. 342: JPEG2000: Regions of Interest

Vergleich JPEG und JPEG2000

Die qualitativen Vorteile von JPEG2000 kommen vor allem bei hohen Kompressionsraten (> 50) zum Tragen. Bei niedriger Kompression (< 20) ist JPEG ebenbürtig, ja im verlustfreien Modus sogar schneller.



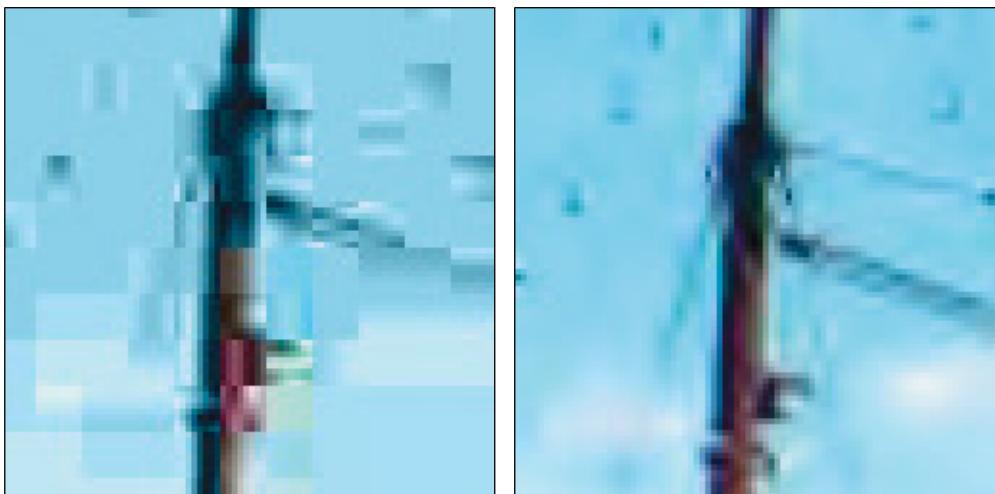


Abb. 343 Kompression mit Faktor 90 bei JPEG links und mit JPEG2000 rechts [Bertuch01]

12.3.3 Neuere Formate

Die Forschung bleibt nicht stehen, auch wenn wahrscheinlich in Zukunft keine Wunder mehr zu erwarten sind. Die zwei aktuellsten Formate [WebP](#) und [BPG](#) sind beide aus den aktuellsten Video-Kompressions-Formaten entstanden.

12.3.3.1 WebP

Das [WebP Format](#) entstand aus dem [Video-Codec WebM von Google](#). Dieses entstand wiederum aus dem VP8-Video-Codec der von Google aufgekauften Firma [On2 Technologies](#).

12.3.3.2 BPG

Das aktuellste opensource Projekt [BPG \(Better Portable Graphics\)](#) ist ein neues Bildformat, das ebenfalls das alte JPEG-Format ersetzen will, wenn Qualität oder Dateigröße eine Rolle spielt. Die Hauptvorteile sind:

- Hohe Kompressionsrate. Dateien sind viel kleiner als JPEG für ähnliche Qualität.
- Von den meisten Web-Browsern unterstützt mit einem kleinen Javascript Decoder.
- Basiert auf einer Teilmenge der HEVC-Video-Kompressions-Standard.
- Gleiche Farbformate wie JPEG
- Alpha-Kanal
- 8 bis 14 Bit pro Kanal für einen höheren Dynamikumfang
- Unterstützt verlustfreie Komprimierung.
- Verschiedene Metadaten (EXIF, ICC-Profil, XMP)
- Animation Unterstützung

In einer [Vergleichs-Webapplikation](#) können die Formate [JPEG](#), [JPEG2000](#), [JPEG XR](#), [WebP](#) und [BPG](#) miteinander verglichen werden.

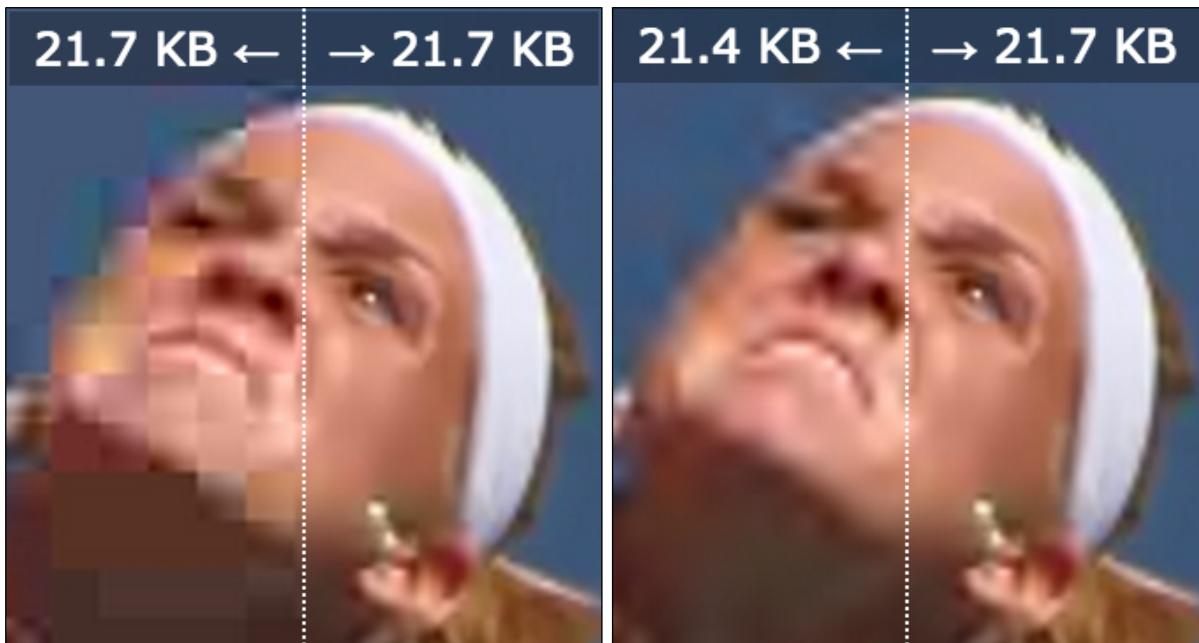


Abb. 344 Links: JPEG/BPG, Rechts JPEG2000/BPG: Klare Qualitätsunterschiede bei hohen Kompressionsraten und gleicher Dateigröße.

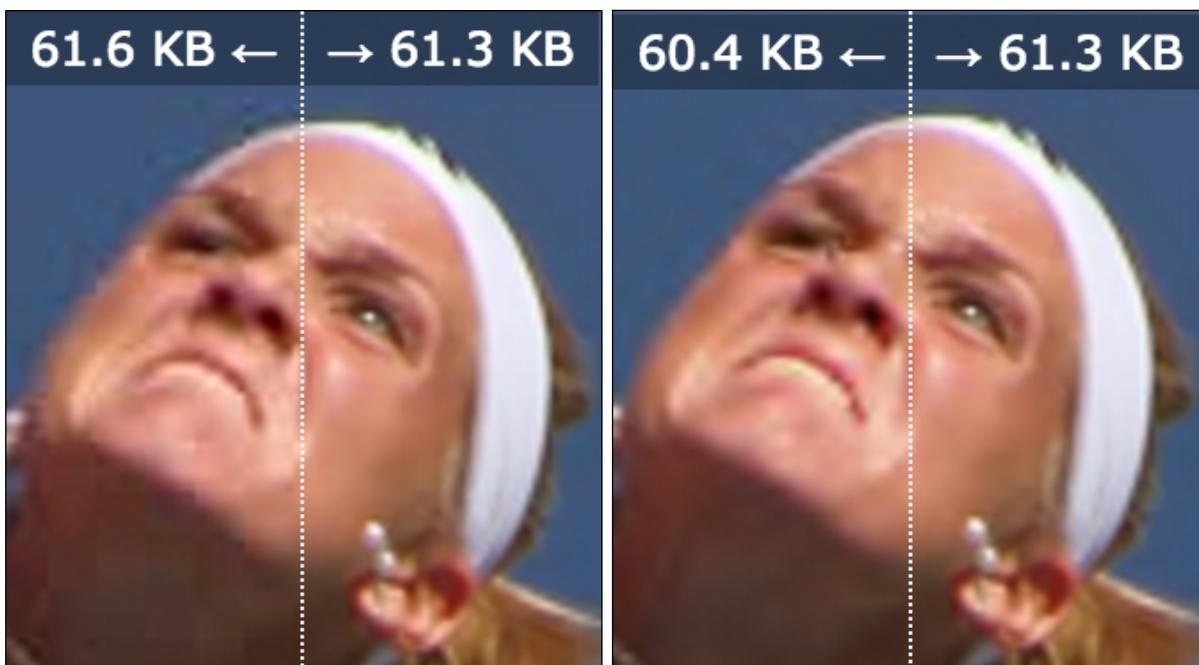


Abb. 345 Links: JPEG/BPG, Rechts JPEG2000/BPG: Geringere Qualitätsunterschiede bei tiefen Kompressionsraten und gleicher Dateigröße.



12.3.4 Übung DCT mit Excel

Nachfolgend sehen sie das Excel-Sheet [dct.xls](#), das als Ausgangslage für diese Übung zur Verfügung steht. Die beiden Transformationen, DCT und IDCT sind in VBA-Code ausprogrammiert und können über die beiden Knöpfe ausgelöst werden.

Abb. 346: dct.xlsx Excel-Sheet mit diskreter Kosinus Transformation

Zu den beiden Routinen kommen Sie, indem Sie den Visual Basic-Editor öffnen mit ALT-F11 oder über das Menü *Extras>Makro>Visual Basic-Editor*:

Mit folgender VBA-Routine wird die DCT berechnet:

```
Private Sub cmdDCT_Click()
    Dim u As Integer, v As Integer
    Dim x As Integer, y As Integer
    Dim invSqr2 As Double
    Dim pi16 As Double
    Dim cu As Double, cv As Double
    Dim cos1 As Double, cos2 As Double
    Dim sum As Double
    Dim f(7, 7) As Integer 'source array
    Dim D(7, 7) As Integer 'destination array
    Dim Q(7, 7) As Integer 'quantisation matrix

    'fill source array and quantisation matrix
    For x = 0 To 7: For y = 0 To 7: f(x,y) = Cells(y+ 3, x+1): Next y: Next x
    For x = 0 To 7: For y = 0 To 7: Q(x,y) = Cells(y+13, x+1): Next y: Next x

    'pre calcs
    invSqr2 = 1 / Sqr(2)
    piDiv16 = 3.14159265358979 / 16

    For u = 0 To 7
        cu = If(u = 0, invSqr2, 1)
        For v = 0 To 7
            cv = If(v = 0, invSqr2, 1)
            sum = 0
            For x = 0 To 7
                cos1 = Cos(piDiv16 * u * (2 * x + 1))
                For y = 0 To 7
                    cos2 = Cos(piDiv16 * v * (2 * y + 1))
                    sum = sum + f(x, y) * cos1 * cos2
                Next y
            Next x
            sum = 0.25 * cu * cv * sum / Q(u, v)
            D(u, v) = CInt(sum)
        Next v
    Next u

    'fill target cells
    For u = 0 To 7
        For v = 0 To 7
            Cells(v+3, u+13) = D(u,v)
        Next v
    Next u
End Sub
```

Zuerst werden die Bereiche der Quellmatrix und der Quantisierungsmatrix in die Arrays f(7,7) und Q(7,7) abgefüllt. Um das Ziel-Array D zu berechnen, wird die innerste Schleife $u \cdot v \cdot x \cdot y = 8^4 = 4096$ Mal durchlaufen. Am Schluss wird das Resultat mit der Funktion *Cells(Row, Column)* in die Excel-Zellen kopiert.

Die Einfärbung der Zellen wird über die bedingte Formatierung mit Farbskalen erreicht und ist erst ab Microsoft Office 2011 verfügbar.

Aufgaben

1. Setzen Sie nachfolgende Zahlen in die Ausgangsmatrix.
 - a) Welche Frequenzen sind in diesem Signal?
 - b) Was lässt sich über die Rekonstruktion nach der IDCT sagen?

2. Berechnen Sie in einem separaten 8x8-Bereich die absolute Differenz (=Fehler) pro Zelle nach der IDCT.
3. Berechnen Sie daraus in einer Zelle, den durchschnittlichen absoluten Fehler.
4. Setzen Sie in eine Zelle den Skalierungsfaktor *QScale*. Verwenden Sie diesen in den Transformationen um die Quantisierung zu skalieren. Welche Auswirkungen hat er?
5. Erzeugen Sie mit der IDCT ungefähr folgende Signale:

200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0
200	200	200	200	0	0	0	0

363	268	132	37	37	132	268	363	489	445	363	257	143	37	-45	-89	214	214	214	214	214	214	214	214	105	87	120	76	124	80	113	95
363	268	132	37	37	132	268	363	445	407	339	249	151	61	-7	-45	147	147	147	147	147	147	147	147	87	138	43	167	33	157	62	113
363	268	132	37	37	132	268	363	363	339	293	233	167	107	61	37	53	53	53	53	53	53	53	53	120	43	186	-1	201	14	157	80
363	268	132	37	37	132	268	363	257	249	233	211	189	167	151	143	-14	-14	-14	-14	-14	-14	-14	-14	76	167	-1	219	-19	201	33	124
363	268	132	37	37	132	268	363	143	151	167	189	211	233	249	257	-14	-14	-14	-14	-14	-14	-14	-14	124	33	201	-19	219	-1	167	76
363	268	132	37	37	132	268	363	37	61	107	167	233	293	339	363	53	53	53	53	53	53	53	53	80	157	14	201	-1	186	43	120
363	268	132	37	37	132	268	363	-45	-7	61	151	249	339	407	445	147	147	147	147	147	147	147	147	113	62	157	33	167	43	138	87
363	268	132	37	37	132	268	363	-89	-45	37	143	257	363	445	489	214	214	214	214	214	214	214	214	95	113	80	124	76	120	87	105

Welche Frequenzen müssen Sie für diese Signale eingeben?

12.4 Videokompression

12.4.1 Grundbegriffe

12.4.1.1 Kompressionsstandard

Ein Kompressionsstandard ist eine Spezifikation, gemäss der eine Bild-, Video- oder Audiodatei aufgebaut ist. Mit dieser Spezifikation sollte es möglich sein, eine Software zu entwickeln, die solche Dateien lesen kann und die Daten zu einem abspielbaren Film verarbeiten kann. Wie man die Daten zu verarbeiten hat, muss nicht bis ins letzte Detail spezifiziert sein. Ein Kompressionsstandard ist also weder eine Software noch ein Algorithmus.

Ein offizieller Standard bedeutet im Übrigen nicht, dass eine standardkonforme Software, Opensource oder sonst irgendwie gratis ist. Kompressionsstandards werden von internationalen Organisationen erarbeitet:

- *ITU (International Telecommunication Union)* mit Sitz in Genf ist die einzige Organisation, die sich offiziell und weltweit mit technischen Aspekten der Telekommunikation beschäftigt, und ist heute eine Teilorganisation der UNO. Zu den bekanntesten ITU-Standards gehören: H.261 - H.264
- *MPEG (Moving Pictures Experts Group)* ist ein Gremium der *International Standard Organization (ISO)* ebenfalls mit Sitz in Genf. Die ISO ist die internationale Vereinigung der Standardisierungsgremien von 148 Ländern. Zu den bekanntesten MPEG-Standards gehören: MPEG-1, MPEG-2, MPEG-4

12.4.1.2 Video-Codec

Die Abkürzung *Codec* steht für *Codieren-Decodieren* und bezeichnet eine Software, die auf Windows Betriebssystemen in Form einer DLL (*Dynamic Link Library*) installiert ist. Neben den Codecs, die einen offiziellen Kompressionsstandard umsetzen, gab und gibt es einige Codecs von Firmen wie z. B. DivX oder Sörensen aber auch frei opensource Codecs, deren Bedeutung aber immer mehr abnimmt.

12.4.1.3 Video-Formate

Dieser Begriff verursacht am meisten Konfusion, da er mit allen anderen Begriffen verwechselt werden kann. Dies hängt z.T. Auch mit der Dateiendung (*mpeg, mpg, avi, divx, mov* etc.) unter Windows zusammen, die eigentlich nur vom System verwendet werden dürfte, um die bevorzugte Applikation zu starten. Läuft die Applikation, so sollte diese innerhalb der Datei nachschauen, um welches Videoformat es sich handelt. Weitere Gründe für die Konfusion sind sogenannte *Containerformate*. Diese legen nochmals eine Dateistruktur um das eigentliche Videoformat. Die bekanntesten Containerformate sind *Microsofts AVI* und *Apples Quicktime*.

12.4.1.4 Video-Player

Ein *Video-Player* ist eine Software zum Anzeigen eines digitalen Videoformats. Welche Video-Codecs ein *Player* abspielen kann, ist ganz unterschiedlich. Die meisten (Software)-DVD- oder Blue-Ray-Player können nur die dafür vorgesehenen Formate MPEG-2 und H.264 abspielen. Generische Player, wie der Microsoft Media-Player, Apples QuicktimePlayer oder der VideoLAN-Player können fast alles abspielen, solange die entsprechende Codec-Komponente zur Verfügung steht.

12.4.2 Ziele der Videokompression

Die Standbild-Kompressionsverfahren JPEG und sein Wavelet Nachfolger werden auch für die Videokompression eingesetzt. In DV-Videokameras war die JPEG-Kompression direkt in die Hardware eingebaut und ermöglicht so eine Kompression der Videodaten in Echtzeit. Ein *DV-Stream* mit einem JPEG-Bild pro Frame hatte eine Datenrate von 25 MBit/s oder rund 10 GB/h. Auf eine 4.7 GB grosse DVD bringen wir damit also nur eine halbe Stunde DV-Video. Um Kinofilme auf DVDs verkaufen zu können, war also eine weitere Kompression notwendig, die über die Standbildkompression mit JPEG oder JPEG2000 hinausging.

Grundsätzlich werden zwei Ziele angestrebt:

- 1. Sehr gute Qualität bei moderaten (5-10 MBit/s) oder grossen Datenraten (50-150 MBit/s):** In den moderaten Bereich gehört die DVD, mit der es seit 1995 möglich ist, Filme in guter Qualität zu geniessen. Mit *High Definition TV (HDTV)* mit einer Auflösung von 1920x1080 Pixeln hat sich die Datenrate rund verzehnfacht. Solche Filme passen nur noch auf eine Blue-Ray Disc (mit 25 oder 50 GB). Im Jahr 2012 wurden die Nachfolgestandards UHD oder UHDTV definiert mit den Auflösungen 4K (3840×2160) und 8K (7680×4320).
- 2. Akzeptable Qualität bei sehr kleinen Datenraten (0.5-1 MBit/s):** In diesem Zielgebiet spielt die Auflösung und die Bildqualität eine sekundäre Rolle, weil primär neue Anwendungsfelder erschlossen werden sollen. Im Vordergrund steht hier die Videotelefonie am PC oder per Mobiltelefon.



Abb. 347: Links: Standardauflösungen von SD bis 8K UHD, rechts: Video Streaming auf Mobiltelefon.

Wir konzentrieren uns in den nachfolgenden Unterkapiteln nur auf die Bild- bzw. auf die Videokompression und lassen Audio und die dazugehörige Synchronisation ausser Acht. Für die Kompression des Tonsignals wird ein ähnlicher aber nicht minder komplexer Aufwand betrieben.

12.4.3 Kompression als Reduktion der Redundanz

Grundsätzlich kann die Kompression als Reduktion der Redundanz innerhalb der Information betrachtet werden. Ziel ist es dabei, die Information in einem Signal auf dessen Essenz zu reduzieren, d. h. überflüssige Information, wie mehrfach vorhandene Informationen und unwichtige Information herauszufiltern und zu löschen.

Geht bei der Kompression Information verloren, wenn auch unwichtige, so spricht man von *verlustbehafteter Kompression*. Dies ist bei Bildern und Videos durchaus zulässig und auch notwendig. Bei Texten hingegen darf aber keine einiger Buchstaben verloren gehen. Hier kann nur *verlustfreie Kompression* angewendet werden.

Bei der Videokompression wird die Redundanz in zwei Bereiche unterteilt: räumliche Redundanz

Obwohl ein Bild als 2D-Signal interpretiert werden kann, bezeichnet man die Einzelbild-kompression innerhalb der Videokompression als Reduktion der räumlichen Redundanz. Die bekanntesten Standards sind:

- **GIF** und **PNG** als Vertreter der verlustfreien Kompressionsformate
- **JPEG** und **JPEG2000** als Vertreter der verlustbehafteten Kompressionsformate

12.4.3.1 Zeitliche Redundanz

Video und Film können als Bildinformation über die Zeit betrachtet werden. Zusätzlich zur Redundanz in der Bildebene können wir auch die Redundanz auf der Zeitachse reduzieren:

- **3D-Transformkodierung:** Die Algorithmen dieser Gruppe fassen eine Bildsequenz als 3D-Signal auf und komprimieren es mit klassischer Signaltransformation über drei Dimensionen (zwei für das Bild und eine für die Zeit).
- **Bewegungskompensation:** Die Lösungsansätze dieser Kategorie konzentrieren sich auf den Umstand, dass sich Videobilder in kurzer Zeitabfolge oft kaum unterscheiden oder dass ein bestimmter Bildbereich im nächsten Bild zwar gleich ist, sich aber an einem anderen Ort befindet. Im Kern bestehen diese Algorithmen deshalb aus einer Abschätzung und Kompensation der Bildbewegung.

12.4.4 3D-Transformkodierung

Wie erwähnt kann eine Videosequenz als 3D-Signal aufgefasst werden, wenn man sich die aufeinanderfolgenden Bilder als dritte Dimension vorstellt. In den nachfolgenden Abbildungen sehen Sie oben links ein Bild aus der *Salesman* MPEG-Testsequenz. Wenn wir aus den nachfolgenden 100 Bildern jeweils die 100. Spalte bzw. Die 80. Zeile zusammenfügen, so erhalten wir zwei Bilder als Veränderung der Spalte bzw. Der Zeile über die Zeit.

Eine Filmsequenz kann man so als Datenvolumen interpretieren und die Reduktion der zeitlichen Redundanz kann auf eine Kompression dieses Datenvolumens vereinfacht werden. Dazu eignen sich sowohl die Diskrete Kosinus Transformation (DCT) als auch die Wavelet Transformation (WT).

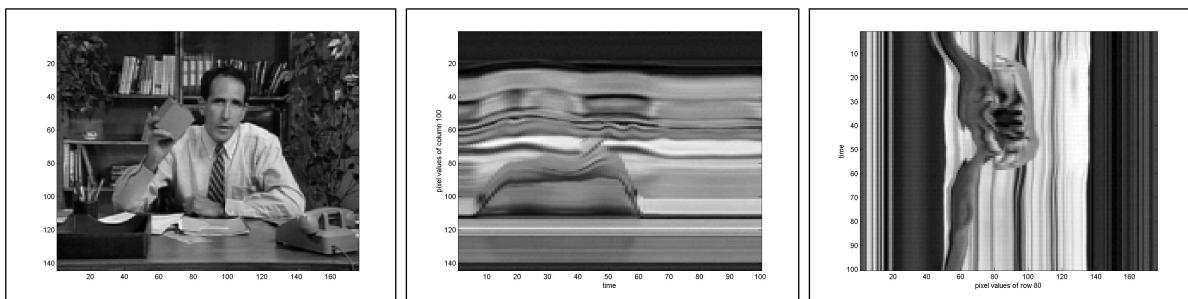


Abb. 348: Salesmen Testsequenz (links) mit einem Spaltenquerschnitt (Mitte) und einem Zeilenquerschnitt (rechts) über die nächsten 100 Bilder (Quelle: S. B. Gokturk).

Die Schritte der 3D-DCT sind im Prinzip dieselben, wie bei der DCT in JPEG mit dem Unterschied, dass nicht 8×8 Pixelblöcke, sondern $8 \times 8 \times 8$ Pixelvolumen gebildet werden müssen:

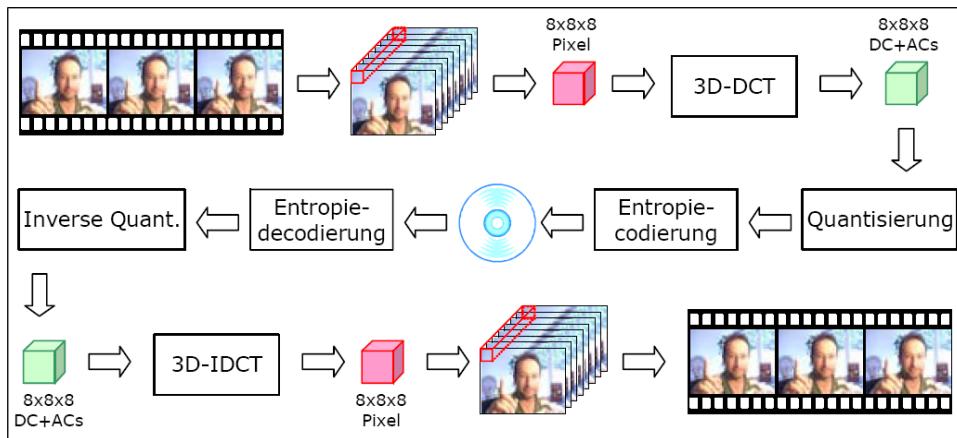


Abb. 349: Ablaufschema der Codierung und Decodierung mit der 3D-DCT

Die 2D-DCT, die wir bei der JPEG-Kompression kennengelernt haben, können wir ohne Weiteres um eine Dimension erweitern:

$$F(u, v, w) = \frac{1}{8} \cdot c_u c_v c_w \cdot \sum_{x=0}^7 \sum_{y=0}^7 \sum_{z=0}^7 f(x, y, z) \cdot \cos \frac{(2x+1)u\pi}{16} \cdot \cos \frac{(2y+1)v\pi}{16} \cdot \cos \frac{(2z+1)w\pi}{16}$$

mit: $c_u, c_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } u, v = 0 \\ 1 & \text{sonst} \end{cases}$

Nach der 3D-DCT erhalten wir einen DC und 511 (8x8x8-1) Kosinuskoeffizienten, die man mit einer 3D-Quantisierungsmatrix zu Nullen dividieren kann. Durch eine 3D-Zickackserialisierung, vom DC zur diagonal gegenüberliegenden Würfecke, ergibt sich ein enormes Kompressionspotenzial für die Lauflängencodierung. Bei dieser 3D-DCT ist ein Zeitabschnitt 8 Bilder lang, was bei 25 FPS (*Frames per Second*) etwa einer 1/3-Sekunde entspricht. Wie bei den JPEG-Standbildern führt dies bei einem hohen Kompressionsgrad zu einer Art Blockbildung, die sich in einem sichtbaren Pulsieren des Filmes äussert. Der Hauptgrund, weshalb sich die 3D-DCT Kompression aber nicht etablieren konnte, ist der hohe Rechenaufwand, der ebenfalls bei der Dekodierung anfällt.

12.4.5 Bewegungskompensation: MPEG

Die Reduktion der zeitlichen Redundanz durch Bewegungskompensation ist die am meisten erforschte Methode und hat durch die MPEG-Standards eine weite Verbreitung gefunden. Das Gremium besteht aus bis zu 300 Fachleuten aus rund 200 Firmen, die sich zum Ziel gesetzt haben, die Standards zur Videokompression flexibel und effizient zu gestalten.

Um die grösstmögliche Anwendungsfreiheit zu gewährleisten, definiert ein MPEG-Standard nur ein Datenmodell zur Dekompression von Bewegtbildern und Tonspuren. Auf diese Weise bleibt MPEG als generischer Standard anwendungsunabhängig, und eine Implementierung muss nicht alle Funktionen offen legen.

Eine hervorragende Informationsquelle über MPEG ist die Website von [Leonardo Chiariglione \(www.chiariglione.org\)](http://www.chiariglione.org), auf der auch die offizielle MPEG-Site geführt wird. Der Italiener Chiariglione ist einer der treibenden Kräfte des MPEG-Gremiums.

12.4.6 MPEG-1

[MPEG-1](#) wurde im November 1992 mit dem Ziel verabschiedet, für Medien mit geringer Bandbreite (1 MBit/s bis 1,5 MBit/s) die Übertragung von farbigen Bewegtbildern mit zugehörigem synchronem, digitalem Audio in akzeptabler Bildfrequenz und möglichst guter Qualität zu erreichen. Dabei hatte man vor allem die zu dieser Zeit aktuellen CD-

ROM-Laufwerke mit einer Bandbreite von 1,2 MBit/s im Blick. Ein MPEG-1-Dekodierer kann Bilder mit einer Auflösung von 352x240 bei 30 Hz (NTSC) oder 352 x 288 bei 25 Hz (PAL) decodieren.

Wie bei allen Kompressionsverfahren ist hauptsächlich die Codierung interessant, die bei MPEG-1 in vier Schritten passiert:

4. Transformation ins YCbCr-Farbmodell mit 4:2:0-Reduktion der Auflösung
5. Reduktion der zeitlichen Redundanz durch Bewegungskompensation
6. Reduktion der räumlichen Redundanz mit DCT und Quantisierung
7. RLE- und Entropiecodierung nach Huffman

Auf den 1., 3. und 4. Schritt wollen wir nicht mehr eingehen, da sie konzeptionell oder sogar ganz mit den entsprechenden Schritten der JPEG-Kompression identisch sind.

MPEG-Vorgänger H.261

Viele der nachfolgenden Eigenschaften sind keine Neuheiten des MPEG-1-Standards, sondern wurden eins zu eins übernommen von einem Vorgänger mit der Bezeichnung **H.261**. Dieser *ITU (International Telecommunication Union)* Standard aus dem Jahre 1990 war auf die ISDN-Bandbreite zugeschnitten und war für die Bildtelefonie vorgesehen. Seine Bildformate waren beschränkt auf die Größen 352 x 288 (*CIF = Common Intermediate Formate*) und 176 x 144 Pixel (*QCIF = Quater CIF*) und damit nicht geeignet für die Spezifikation von MPEG-1.

12.4.6.1 Schichtenmodell

Wie bei anderen digitalen Datenströmen wird auch bei Video ein Schichtenmodell verwendet, um die Daten hierarchisch zu strukturieren. Dieses Schichtenmodell wird auch **MPEG-Syntax** genannt:

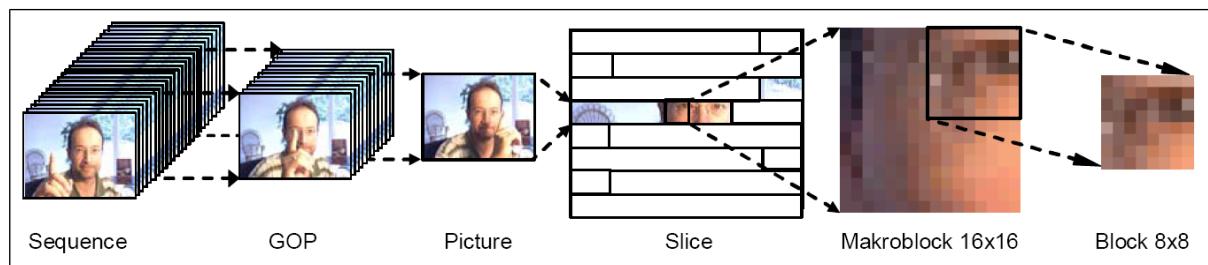


Abb. 350: Visualisierung der einzelnen Schichten des MPEG-Schichtenmodells.

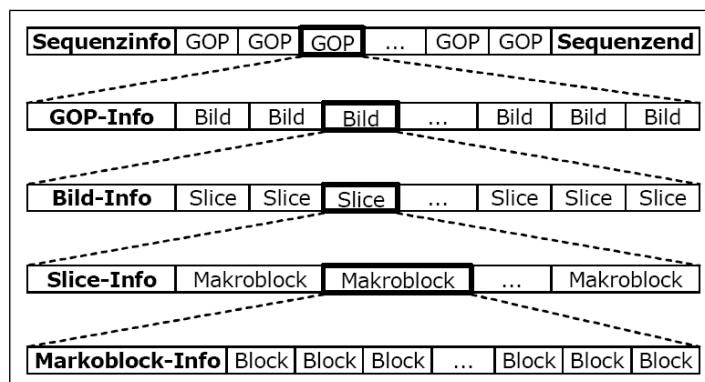


Abb. 351: MPEG-1 Schichtenmodell

- Zuoberst ist die **Sequenz**, sie stellt dem Decoder Informationen über das Format der Bilder, benötigte Bildwiederholfrequenz und Datenrate, sowie optional eine an die Daten besser angepasste Quantisierungsmatrix zur Verfügung.

- Die Sequenz ist in Gruppen von Bildern unterteilt, sogenannte *Group Of Picture (GOP)*. Eine GOP enthält verschiedene Typen von Bildern in einer bestimmten Reihenfolge. Ein Videoschnitt ohne zusätzlichen Aufwand ist nur an GOP-Grenzen möglich.
- Die Bildschicht enthält alle Informationen, die benötigt werden, um ein *Bild (Frame)* zu decodieren. Der Header enthält die Information, als wievieltes Bild der entsprechenden GOP das Bild dargestellt werden soll.
- Ein Bild besteht aus einer Folge von Scheiben (*Slices*), die 16 Pixel hoch sind. Die Slices folgen der Abtastrichtung üblicher Bildschirme und enthalten eine frei wählbare Anzahl von Makroblöcken. Durch die Unterteilung in Slices erreicht man eine bessere Kontrolle über Datenrate und Übertragungssicherheit.
- Ein *Makroblock* ist 16x16 Pixel gross und besteht aus 8x8er Pixelblöcken mit Helligkeits- (4 Blöcke für den Y-Kanal) und Farbinformationen (2 Blöcke für U- und V-Kanal). Ein Makroblock enthält zudem Informationen zur Bewegungskompensation.
- Die *Blockschicht* bildet die unterste Schicht aus 8x8 Pixelblöcken mit den Helligkeits- oder Farbinformationen, so wie wir es von JPEG her kennen, mit RLE und Huffman-Codierung.

MPEG Video arbeitet bitweise, d. h., nur wenige Symbole am Start der Schichten müssen an Byte-Grenzen ausgerichtet sein. Der Rest der Informationen wird, wie in JPEG, mit variablen Codelängen dargestellt, um eine möglichst optimale Kompression zu erreichen.

12.4.6.2 Bildtypen

Um die zeitliche Redundanz zu verringern, bedient sich MPEG-1 des sogenannten *Interframe Codings* (bildübergreifende Codierung) und führt dazu drei verschiedene Bildtypen ein:

- Intra-Frames (I-Bilder)** stellen in einer GOP die Fixpunkte dar und werden deshalb auch *Key-Frames* genannt. Sie werden vollständig codiert, d. h., ohne Referenzierung eines anderen Bildes und verwenden dazu eine JPEG-ähnliche Kompression mit DCT und abschliessender Huffman- und Lauflängencodierung.
- Predicted Frames (P-Bilder)** werden mit Bezug auf das letzte vorhergehende I- oder P-Bild kodiert. Dazu verwenden sie eine Bewegungsestimation und Kompensation, die wir im nächsten Kapitel genauer betrachten werden.
- Bidirectional Frames (B-Bilder)** bieten die höchste Kompressionsrate, da sie sich auf das vorangegangene I- bzw. P-Bild und/oder auf das nachfolgende I- bzw. P-Bild beziehen können.

Die Verwendung der jeweiligen Bildtypen bestimmt wesentlich sowohl Qualität als auch Kompressionsgrad des Videos. So verbessert man durch I-Bilder die Qualität, während durch B-Bilder der Kompressionsgrad erhöht wird. Insbesondere ist der Abstand zweier I-Bilder im Datenstrom ein Mass für die Qualität des Videos. Akzeptable Qualität und Kompression erreicht man, wenn etwa 2 I-Bilder pro Sekunde und zwischen zwei P- bzw. Zwischen einem I- und P-Bild 2 B-Bilder vorkommen.

Da sich B-Bilder auch auf das nachfolgende I- oder P-Bild beziehen können, ist die Übertragungsreihenfolge nicht gleich der Abspielreihenfolge:

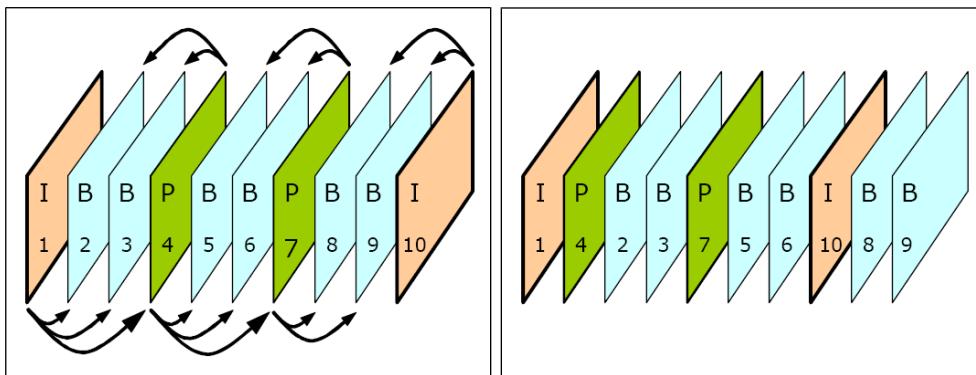


Abb. 352: Abspielreihenfolge der I-, P- und B-Bilder links und rechts der Übertragungsreihenfolge Ein weiterer Bildtyp ist der D-Typ (*DC Coded Picture*). Er hat mit der GOP und der Bildkompression nichts zu tun und wird mit reduzierter Auflösung zur Schnellsuche eingesetzt.

12.4.6.3 Bewegungskompensation

Bei der Bewegungskompensation werden identische Bildblöcke, die sich gar nicht oder nur durch eine Verschiebung unterscheiden, durch einen Bewegungsvektor ersetzt. Die Berechnung dieses Vektors geschieht auf der Ebene der Makroblöcke und ist im Kern eine Suche nach einem ähnlichen Block im referenzierten Bild.

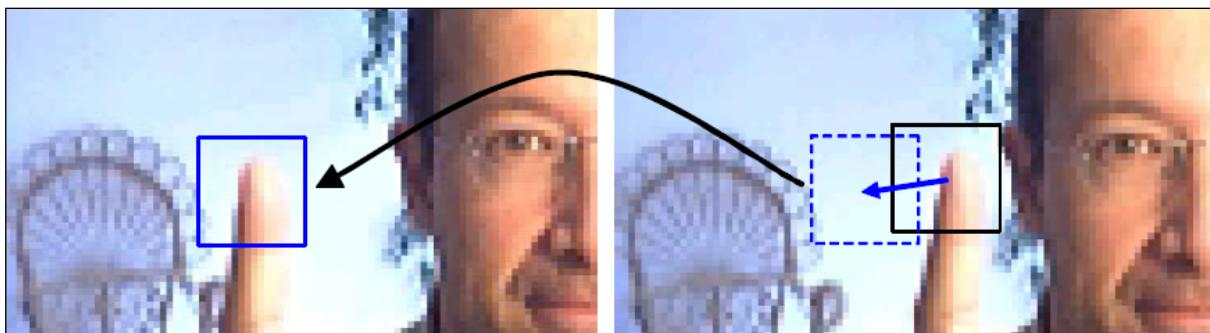


Abb. 353: Welcher 16x16 Pixelbereich im I-Bild links hat die geringste Differenz zum Makroblock im P-Bild rechts?

Ein Mass für die Ähnlichkeit ist z. B. Die Differenz der quadrierten Summen beider Vergleichsblöcke. Ist diese Differenz Null, so sind die Blöcke identisch. Dies wird jedoch kaum vorkommen, da sich bei statischen Blöcken alleine durch das Bildrauschen eine Differenz ergibt. Bei sich bewegenden Objekten wird zudem die veränderte Perspektive zu einer merklichen Differenz führen. Diese Differenz wird zusätzlich zum Bewegungsvektor in Form eines *Fehlerbildes* mit-kodiert. Da sich ein Fehlerbild aus der Differenz zweier Bilder ergibt, ist es für ähnliche Bilder mehrheitlich Schwarz und ermöglicht daher eine hohe Kompression.

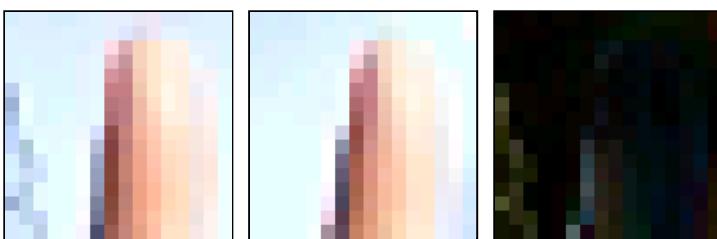


Abb. 354: Links: Vergleichsbild mit der geringsten Differenz zum Makroblock (Mitte). Rechts: RGB Differenzbild = $\text{abs}(\text{links} - \text{rechts})$. In Wirklichkeit wird nicht im RGB-, sondern kanalgetrennt im YCbCr-Raum gearbeitet.

B-Bilder mit bidirektionaler Bewegungskompensation können am stärksten komprimiert werden, da sie sich sowohl auf ein Vorangegangenes wie auch auf ein nachfolgendes I- oder P-Bild beziehen können. Sind Vorwärts- und Rückwärts-Prädiktion etwa gleich gut, so werden die Bewegungsvektoren und die Fehlerbilder gemittelt. Unterscheiden sich die beiden Prädiktionen, so wird die Bessere verwendet.

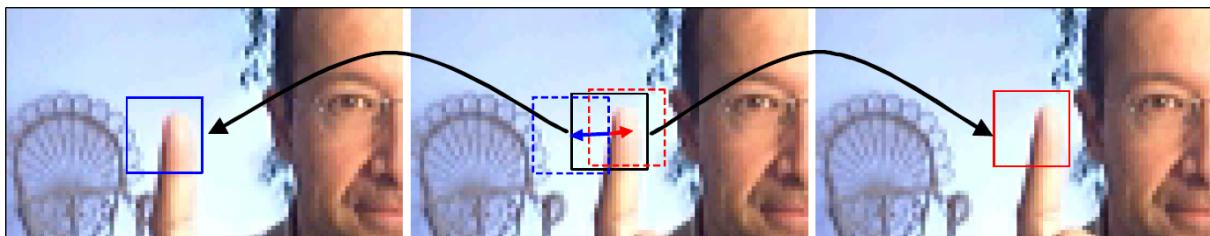


Abb. 355: bidirektionale Bewegungsestimation

B-Bilder können aber nicht nur eine genauere Bewegung berechnen, sie können auch Bewegungen detektieren, die mit einer Rückwärts-Prädiktion gar nicht gefunden werden können. Wie in der nachfolgenden Grafik ersichtlich ist, kann bei Objekten, die neu in einem Bild auftauchen oder bei Blöcken, die im vergangenen Bild verdeckt waren, nur eine Rückwärtskompensation aus dem nächsten Bild berechnet werden. Auch wenn diese Beispiele durchaus überzeugend sind, so sei hier angemerkt, dass unter Experten die B-Bilder immer noch umstritten sind.

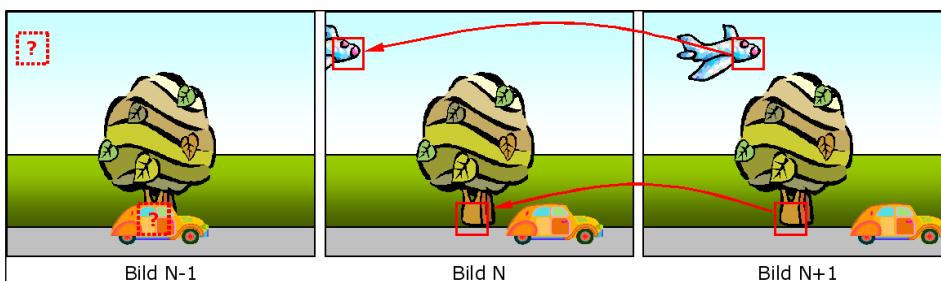


Abb. 356: Sequenz aus drei Bildern mit zwei Beispielblöcken mit Rückwärtskompensation BMA (Block Matching Algorithm). Bleibt noch die Frage zu klären, wie nun genau ein Bewegungsvektor berechnet wird. Die Suche nach einem ähnlichen Block geschieht auf Makroblockebene. Auf diesem wichtigsten Teil von MPEG-1 (wie auch MPEG-2) wurde schon viel Forschungsarbeit investiert, um eine möglichst exakte Übereinstimmung in möglichst kurzer Zeit zu erreichen.

Die effiziente Suche wird mit *Block Matching Algorithmen* realisiert, die wir im Kapitel 11.3.1.1 kennengelernt haben.

12.4.6.4 Sequenzanalyse

Nachfolgend sehen Sie die ersten 9 Bilder einer *Group of Pictures (GOP)* aus einem MPEG-1 Video, das mit dem freien Encoder *TMPGEnc* (www.tmpgenc.net) erstellt wurde. Das Originalvideo umfasst 240 Bilder der Grösse 160 x 120 Pixel und ist unkomprimiert 13'838'336 Bytes gross. Die dargestellte Sequenz wurde mit dem *MPEGRepair* Tool der Firma *PixelTools* erstellt:

- **Weisse Rahmen** markieren interkodierte Makroblöcke. Das I-Bild besteht nur aus solchen Blöcken und dient als Referenz für die P-Bilder und die nachfolgenden B-Bilder (Bild 2 und 3).
- **Blaue Rahmen** markieren vorwärts codierte Makroblöcke aus einem vorangegangenen I- oder P-Bild. Blaue Rahmen kommen in P- oder B-Bildern vor.
- **Rote Rahmen** markieren rückwärts codierte Makroblöcke aus einem zukünftigen P- oder I-Bild. Rote Rahmen kommen nur in B-Bildern vor.
- **Rosarote Rahmen** markieren interpolierte Makroblöcke, die zur Hälfte aus einem Zukünftigen und zur Hälfte aus einem zurückliegenden I- oder P-Bild berechnet wurden.
- **Kein Rahmen** in den P-Bildern bedeutet, dass der Makroblock sich praktisch nicht von seinem Vorgängerblock im I- oder P-Bild unterscheidet und deshalb unverändert übernommen wird.
- **Blaue Bewegungsvektoren** kennzeichnen eine Verschiebung aus einem vorangegangenen I- oder P-Bild und können in P- oder B-Bildern vorkommen.

- **Rote Bewegungsvektoren** kennzeichnen eine Verschiebung in ein nachfolgendes I- oder P-Bild und können nur in B-Bildern vorkommen. Ist sowohl ein blauer wie auch ein roter Vektor eingezeichnet, so konnten beide Verschiebungen berechnet werden. Aus welchem Bild die Information schlussendlich bezogen wurde, ist durch die Rahmenfarbe gekennzeichnet.

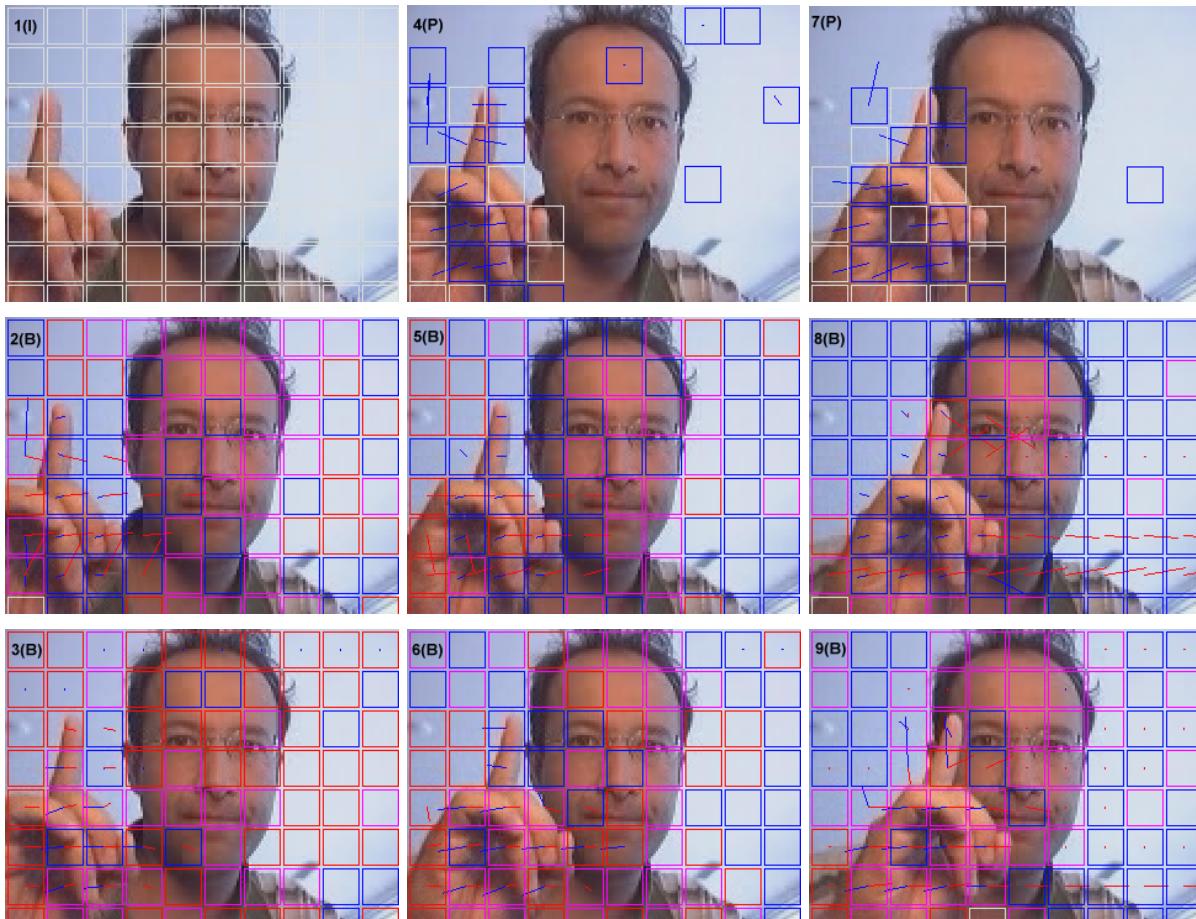
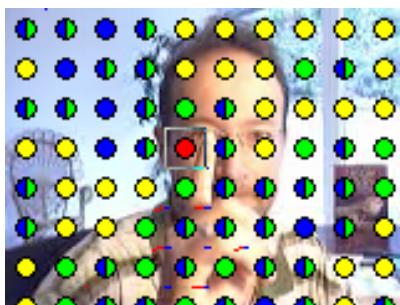


Abb. 357: IBBPBBPBB Bildsequenz von links oben, spaltenweise nach rechts unten.

12.4.6.5 Übung MPEG-1

1. Installieren Sie die Software *StreamEye Studio* der Firma Elecard (Trial: 21 Tage).
2. Installieren Sie den MPEG-1/2 Encoder *TMPGEnc*.
3. codieren Sie das Video *Small_di.avi* mit *TMPGEnc* im MPEG-1 Standard mit gleicher Ausgabegrösse 160x120 bei konstanter Bitrate von 200 kbit/s mit 30 FPS. Belassen Sie die vorgegebene GOP-Struktur.
4. Wie gross ist das komprimierte Video?
5. Stimmt die Dateigrösse mit der eingestellten Bitrate überein?
6. Analysieren Sie den codierten MPEG-1 Stream mit dem Programm *ESEyeApp_lite* aus dem *StreamEye Studio*. Mit den Cursor-Links/Rechts-Tasten können Sie Bild um Bild vor- oder rückwärts gehen.
7. Mit dem Schalter *Step-by* auf *STRM*. Verstehen Sie die Reihenfolge?
8. Was bedeuten die Farben der runden Punkte wenn man die Makroblock Typen (*MB-Types*) visualisiert?



9. Wie gross sind die I-Frames ungefähr in % vom unkomprimierten Originalbild?
10. Wie ist das Größenverhältnis zwischen I-, P- und B-Frames?
11. *Bringen es die B-Frames überhaupt? Machen Sie einen Versuch ohne B-Frames?*

12.4.7 MPEG-2

Noch bevor MPEG-1 verabschiedet war, begann 1991 die Planung für den Nachfolgestandard **MPEG-2**, der zusammen mit der ITU entwickelt wurde und dort **H.262** genannt wird.

Wenn MPEG-1 die Qualität von VHS-Video erreichen soll, so ist MPEG-2 für eine viel höhere Qualität vorgesehen. Als Zielanwendungen waren *Video-On-Demand* Applikationen und die Ausstrahlung von digitalem Video über Satelliten vorgesehen. Der grosse Durchbruch gelang dem Format durch das Aufkommen der *DVD (Digital Versatile Disc)*. MPEG-2 ist vollständig abwärts kompatibel, weshalb jeder MPEG-2 Decoder auch MPEG-1 Dateien decodieren kann.

Flexible Datenrate: Das wenig flexible Konzept der konstanten Datenrate wurde aufgegeben. Neu erhält der Encoder eine Referenz-Datenrate (z. B. 4 MBit/s), die er grob einzuhalten hat. In schwierigen Szenen mit viel Bewegung darf er aber bis auf 10 oder mehr MBit/s hochfahren, um die Qualität aufrechtzuerhalten.

Interlacing: Ein weiterer Unterschied ist, dass man die vollen Bilder leider wieder durch die Untugend der Fernsehhalbbilder (*Interlacing*) ersetzt hat. Dieser Rückschritt, ein Zugeständnis an die Fernsehindustrie, verursacht einen riesigen Aufwand, sowohl in der Codierung wie auch in der Decodierung.

Lizenzkosten: Zum ersten Mal ist der Verkauf von MPEG-2-konformer Hard- und Software nicht mehr gratis. Pro verkauften Encoder/Decoder sind 2.5 US-Dollar an die MPEG abzuliefern.

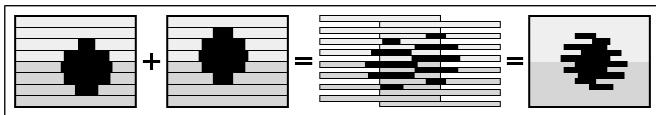


Abb. 358: Wenn die beiden Halbbilder zeitlich verschoben aufgenommen wurden, kommt es zu auffälligen Kammartefakten, die nur mit aufwendigen Filtern korrigiert werden können (Bildquelle: M. Fiedler, TU Chemnitz).

12.4.7.1 Profiles & Levels

Um die Implementierung eines MPEG-2-Encoders zu vereinfachen, wurden die vielen Features in sogenannten *Profilen* gruppiert, die nochmals in Bitraten abhängige *Levels* unterteilt sind. Fast alle existierenden MPEG-2 Daten sind aber im *Main Level* des *Main Profiles* codiert (Abkürzung: *MP@ML*).

MPEG-3: Ein dritter Standard *MPEG-3* war vorgesehen für eine weitere Qualitätssteigerung, den *HDTV (High Definition Television)* Fernsehstandard. Die erzielte Qualität mit MPEG-2 war allerdings so überzeugend, dass die Anforderungen von HDTV in MPEG-2 integriert wurden. *MP3* hat übrigens nichts mit MPEG-3 zu tun, sondern bezeichnet den *MPEG-1 Audio Layer 3*.

MPEG-2-Profiles

<i>Simple</i>	(nur I- und P-Bilder, kein Interlacing)
<i>Main</i>	(I-, P- und B-Bilder, mit Interlacing)
<i>SNR</i>	(Main + <i>SNR Scalability</i> (progressive Übertragung der Daten))
<i>Spatial</i>	(Main + <i>Spatial Scalability</i> (Video wird in mehreren Auflösungen codiert))
<i>High</i>	(höhere Präzision, besseres Chroma-Subsampling)

MPEG-2-Levels (Angaben für PAL)

<i>Simple</i>	(max. 352x288/25 fps, 4 MBit/s)
<i>Main</i>	(max. 720x576/25 fps, 15 MBit/s)
<i>High1440</i>	(max. 1440x1152/30 fps, 60 MBit/s)
<i>High</i>	(max. 1920x1152/30 fps, 80 MBit/s)

12.4.8 MPEG-4

Wer schon einmal ein MPEG-2 Video des HDTV Standards in voller Auflösung gesehen hat, kann sich kaum noch eine Steigerung an Qualität vorstellen. Die Motivation für die Entwicklung von MPEG-4 war dann auch viel mehr die weitere Reduktion des Datenstroms sowie eine Öffnung für einen breiten Multimedia-Einsatzbereich. Die erste Version von MPEG-4 wurde als ISO Standard 14496 1998 verabschiedet. Als Zielanwendungen waren Applikationen aus folgenden Bereichen vorgesehen:

- Digitales Fernsehen über Internet
- Interaktives Fernsehen
- Interaktive Multimediaanwendungen

Bis heute ist nur ein kleiner Teil der angestrebten Ziele realisiert worden. Die bekanntesten Vertreter des Standards sind Video Codecs, wie *DivX* oder die Open Source Version *XviD*, die allerdings nur einen kleinsten Ausschnitt der Spezifikation erfüllen. Weil die Kompressionsrate noch einmal stark verbessert werden konnte, werden diese Codecs vor allem für das illegale Kopieren von DVD-Videos auf CD-ROMs eingesetzt.

12.4.8.1 MPEG-4 Szenengraph

Der grösste Unterschied zu seinen Vorgängerstandards ist, dass MPEG-4 sich nicht auf die Speicherung von natürlichen Bewegtbildern beschränkt, sondern jeglichen multi-medialen Inhalt aufnehmen kann. Grob wird dieser zwar immer noch in Bild und Ton unterschieden, einzelne Bestandteile (Objekte) können aber natürlichen oder

synthetischen Ursprungs sein. Um das Ganze in Ordnung zu halten, verwendet man einen hierarchischen Szenengraphen, ähnlich, wie wir ihn im Kapitel über Szenengraphen kennengelernt haben.

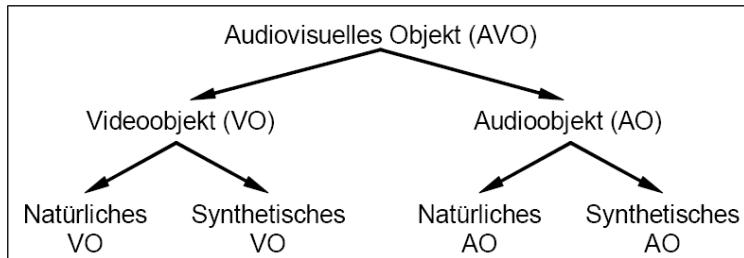


Abb. 359: MPEG-4 Unterteilung in natürliche und synthetische Objekte

Beispieldiagramm

Das nachfolgende Beispiel ist aus der MPEG-4 Dokumentation zeigt eine breite Palette an audiovisuellen Objekten, die zu einer Szene zusammengesetzt werden.

Man kann sich eine Präsentationsszene vorstellen, wo eine Instruktorin zu einem Thema spricht, das auf einer Leinwand im Hintergrund visualisiert wird. Zusätzlich werden 3D-Objekte verwendet, um gewisse Aspekte dreidimensional erklären zu können.

Angenommen sie wären ein Hersteller von Trainingsvideos, die solche Szenen enthalten. Die Produktion für mehrere Sprachen mit klassischem Video wäre eine aufwendige und teure Angelegenheit, da für jede Sprache, ein Synchronsprecher engagiert werden müsste. In einer MPEG-4 Szene muss nur ein Audioobjekt ausgetauscht werden und schon ist eine andere Sprachversion realisiert. Bei einem synthetischen Audioobjekt müsste man sogar nur eine Textdatei austauschen, damit durch Sprachsynthese eine andere Sprachversion erzeugt wird.

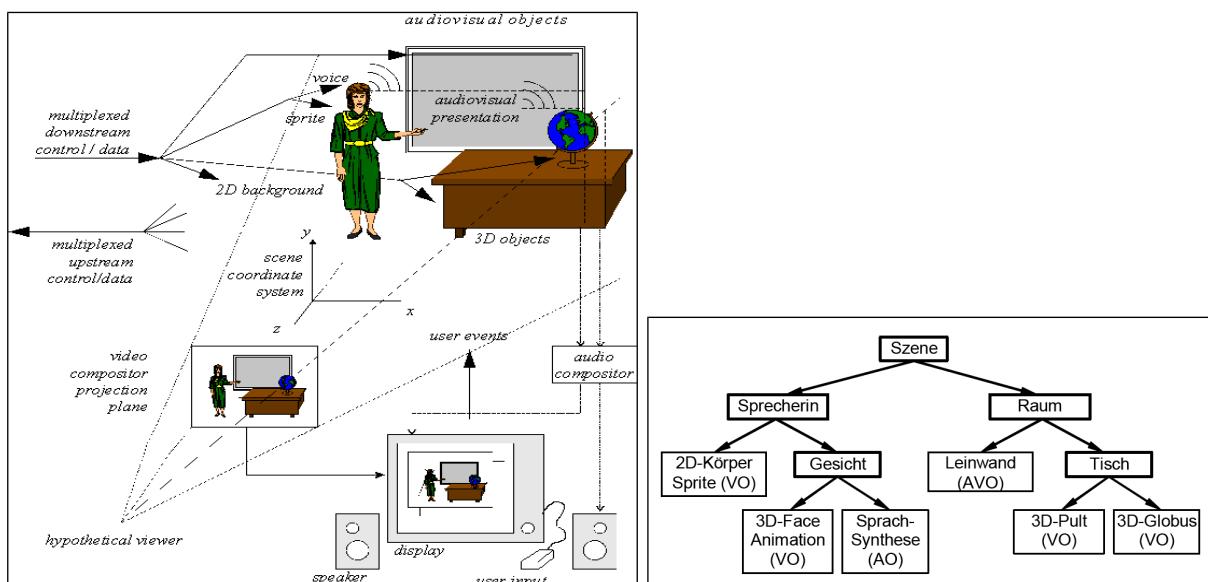


Abb. 360: Beispieldiagramm-Szene mit ihrem Szenengraphen (Quelle: Phil Chou, Xerox PARC)

Der Szenengraph beschreibt nicht nur wo, welche Objekte platziert werden, sondern auch wann ein Objekt erscheinen oder entfernt werden soll. Zu guter Letzt ist das Ganze noch interaktiv. Mit dem Schnittstellen API [MPEG-J](#) (J für Java) kann auf den Szenengraphen per JAVA zugegriffen werden.

Alle Objekte und der Szenengraph selbst werden getrennt in sogenannten [Elementary Streams](#) übertragen und unabhängig voneinander dekomprimiert. Ist alles bereit, so kann die Szene anhand des Szenengraphen zusammengesetzt ([Composition](#)) und gezeichnet werden ([Rendering](#)).

12.4.8.2 MPEG-4 Visual Standard

Der ISO Standard 14496-2 beschreibt den visuellen Teil des MPEG-4 Standards, der folgende Arten der Codierung für visuelle Objekte vorsieht:

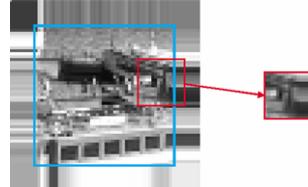
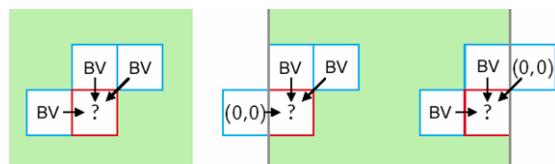
- *Video Object Coding*
 1. *Motion Vector Coding*
 - H.262 (MPEG-2)
 - H.263 (MPEG-4)
 2. *Shape Coding*
 3. *Sprite Coding*
- *Mesh Objekt Coding*
- *Model-Based Coding*
 4. *Facial Animation*
 5. *Body Animation*

Während dem *Video Object Coding* für natürliches Video vorgesehen ist, so sind *Mesh Object* und *Model-based Coding* für synthetische Objekte vorgesehen.

12.4.8.2.1 Video Object Coding

Motion Vector Coding

In dieser Codierungsart wird die klassische Videokompression untergebracht. Ein Video Objekt wird *Video Object Plane (VOP)* genannt und ist in dieser Codierungsart rechteckig. Es kann entweder mit dem alten H.262 Standard (MPEG-2) oder mit dem neuen H.263 Standard (MPEG-4) komprimiert werden. Der Nachfolger erreicht eine bessere Kompressionsrate durch folgende Erweiterungen:

- **Bewegungsvektoren können über den Bildrand hinaus zeigen** und ermöglichen so eine bessere Bewegungskompensation bei Kamerataschen. Die Randpixel werden für das Block Matching einfach kopiert:
 
- **Voraussage der Bewegungsvektoren:** Die Bewegungsvektoren werden nicht mehr direkt codiert, sondern als Differenz zum Median der bereits codierten Nachbarblöcke. Die Differenz (= Voraussage) ist meistens sehr klein und kann daher mit wenigen Bits codiert werden.
 
- **1/4-Pixel-Bewegungskompensation (QPel):** MPEG-2 berechnet die Bewegungskompensation standardmäßig auf ein 1/2 Pixel genau, indem die Blöcke vor dem Vergleich bilinear vergrößert werden. Für schnelle Bewegungen ist dies ausreichend genau. Bei langsamen Bewegungen kann dies aber zu einer Bewegungsunschärfe führen, obwohl sich ein Objekt eben langsam bewegt. Für eine genauere Berechnung des Bewegungsvektors kann MPEG-4 eine präzisere 1/4-Pixel Auflösung verwenden, die durch eine bilineare Vergrößerung erreicht wird.
- **Globale Bewegungskompensation:** Bei einem Kamerataschen oder bei einer Aufnahme aus einem fahrenden Auto bewegen sich viele Blöcke gleichmäßig in dieselbe Richtung. Mittels Fluchtpunkten
 

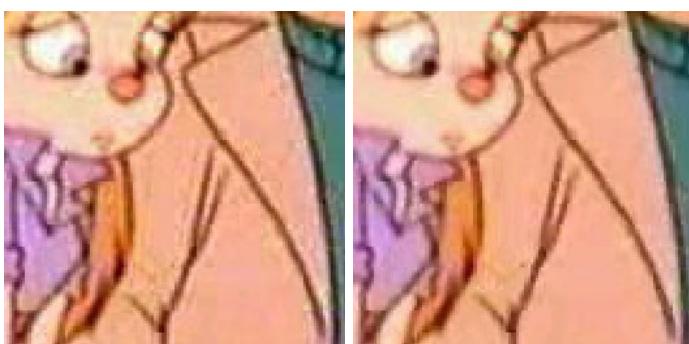
kann eine globale Bewegungstransformation berechnet werden, mit der jedes Pixel einzeln (!!!) transformiert wird.

- **Nachbearbeitung:** Eine starke Kompression führt auch bei MPEG-4 zu den klassischen Blockartefakten, da ja immer noch die DCT angewendet wird. Die meisten MPEG-4 Codecs führen eine Nachbearbeitung (*Postprocessing*) durch, obwohl diese erst in einem MPEG-4-Nachtrag beschrieben wird und zu keinem offiziellen Profil gehört. Bei DivX wird das Nachbearbeiten z. B. nicht beim Codec, sondern in einem eigenen Decoderdialog eingestellt. Man unterscheidet zwei Typen von Nachbearbeitung:

6. *Deblocking*: Dabei werden die Blockartefakte durch Filter verringert:



7. *Deringing*: Dabei werden die Kompressionsartefakte der DCT, die als Rauschen an Kanten (= hohe Frequenzen) in Erscheinung treten reduziert:



Shape Coding

Ist die VOP nicht rechteckig, so wird die Form des Video Objekts anhand eines Transparenzkanals (auch Alphakanal genannt) bestimmt. Zum Luminanz- und Chrominanzkanal wird zusätzlich ein Byte pro Pixel übertragen, das den Transparenzgrad des Pixels enthält. Da ein MPEG-4 Stream mehrere VOPs enthalten kann, ist es nun möglich eine Person im Vordergrund, getrennt vom Hintergrund zu codieren und zu übertragen. Da wir uns hauptsächlich auf die Akteure im Vordergrund konzentrieren, können wir den Hintergrund stärker komprimieren.

Im April 2003 berichtete das Magazin *c't* über eine Software namens VOGUE (*Video Object Generation Tool*) mit der ein klassisches Video eine Szenensegmentierung in Vorder- und Hintergrund durchgeführt werden kann. Eine russische Hacker-Gruppe namens *@F7RM@RCh* soll daraus ein Tool entwickelt haben, mit dem das Schauspielergesicht in ein sogenanntes *Facelet* gepackt wird, das dann getrennt von einer DivX Datei übermittelt wird. Durch eine solche getrennte Codierung der VOPs liesse sich eine DivX Datei nochmals um 50% komprimieren. Noch erstaunlicher war die Möglichkeit, dass durch einfaches Austauschen der Facelet Datei, der Schauspieler ausgewechselt werden könnte. Der Artikel versetzte die Fachwelt in Staunen und die Schauspielergewerkschaft in Schrecken. Nach etwas Recherche im Internet fand man dann aber heraus, dass es sich

um einen visionären Aprilscherz handelte (s. Name der Hacker-Gruppe). Das VOGUE Tool existiert allerdings wirklich und wurde im Rahmen des [MoMuSys](#) Projekts entwickelt.

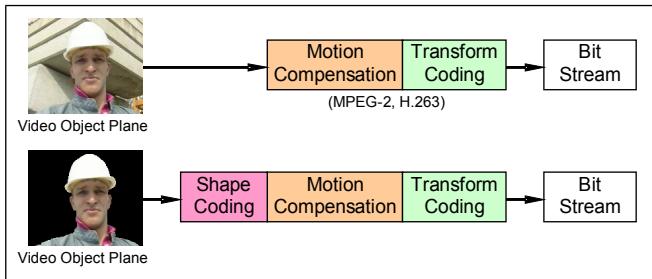


Abb. 361: Shape Coding bei nicht rechteckigen VOPs

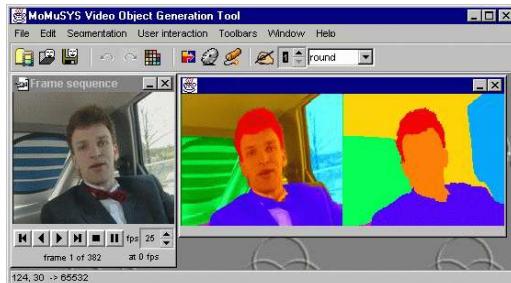


Abb. 362: Video Objekt Generation Tool

Sprite Coding

Eine noch höhere Kompression als mit [Shape Coding](#) erreicht man durch das sogenannte [Sprite Coding](#), bei dem ein statisches Hintergrundsbild verwendet wird. Ein vielleicht sinnvolleres Beispiel als die Tennisszene aus der MPEG-Dokumentation, wäre ein Nachrichtensprecher vor statischem Hintergrund (s. Abb. 19).

12.4.8.2.2 Mesh Object Coding

Eine Mischung aus natürlichem und synthetischem Inhalt stellt das sogenannte [Mesh Objekt](#) dar. Dabei wird ein zweidimensionales Dreiecksnetz (= [Mesh](#)) über ein Rasterbild gelegt (s. Abb. 20). Ein Objekt kann dann in begrenztem Mass animiert werden, indem anhand eines veränderten Dreiecksnetzes das darunter liegende Rasterbild nachskaliert wird. Dieses Vorgehen wird auch [Morphing](#) genannt und spart natürlich viel Platz, da für ein Bild nur das Dreiecksnetz abgespeichert werden muss.

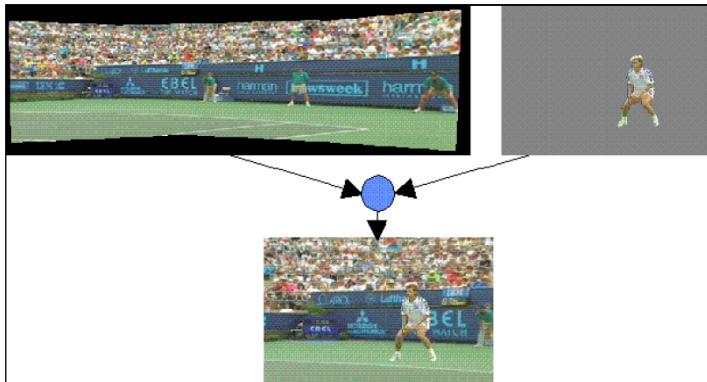


Abb. 363: Sprite Coding (Quelle: MPEG)



Abb. 364: 2D-Mesh Objekt (Quelle: MPEG)

12.4.8.2.3 Model-Based Coding

Noch einen Schritt weiter geht die modellbasierte Codierung ([Model-Based Coding](#)), indem die Objekte in 3D animiert und gespeichert werden. Für die Struktur und das Rendering wird dabei der 3D-Web-Standard [VRML97](#) eingesetzt.

Für menschliche Gesichter und Körper definiert MPEG-4 sogar eigene Standards ([Facial Animation](#), [Body Animation](#)), um die Handhabung und die Komprimierbarkeit weiter zu optimieren. Das Gesicht z. B. wird darin mit 84 [Feature Points](#) (FP) beschrieben, auf welche von aussen mit der MPEG-J Schnittstelle zugegriffen werden kann.

Um die Mimik nicht neu erfinden zu müssen, sind 68 low-level Animationen definiert, mit denen z. B. der Mund geöffnet oder geblinzelt werden kann. Für klassische Gesichtsausdrücke, wie Freude, Traurigkeit etc., aber auch für die Mundbewegungen der Viseme, sind high-level Animationen vordefiniert.

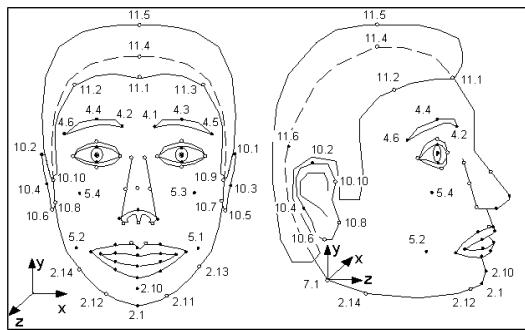


Abb. 365: MPEG-4 Gesicht mit 84 Feature Points



Abb. 366: synthetischer Sprecher von AT&T

12.4.8.3 MPEG-4 Profile

Wie MPEG-2 wurde der MPEG-4 Standard in *Profile* und *Levels* unterteilt. Ein Profil definiert die vom Decoder zu implementierenden Fähigkeiten und ein Level begrenzt die Bandbreite der Übertragungsrate. Die Einteilung in Profile und Level beschränkt nicht nur die Komplexität der Implementierung sondern sichert, neben der Performance auch die Kompatibilität zwischen Encoder und Decoder. Wird eine MPEG-4 Datei mit einem bestimmten *Profil@Level* erzeugt, so kann diese durch jeden Decoder, der das *Profil@Level* beherrscht, diesen auch decodieren. Das *Simple* Profil z. B. ist für einfache Videokompression gedacht, die bis zu 4 VOPs enthalten können, die mit Bewegungskompensation codiert sind. Die maximale Auflösung entspricht im niedrigsten Level dem QCIF (176×144 Pixel).

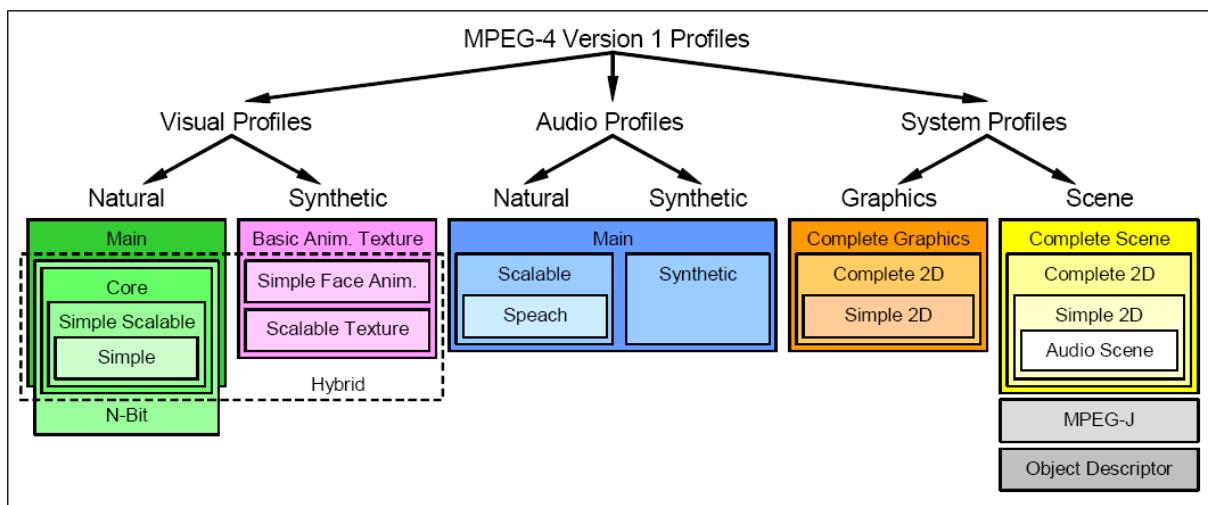


Abb. 367: Struktur der MPEG-4 Profile der Version 1

Die Struktur der MPEG-4 Profile ist einigermaßen hierarchisch, jedoch kamen mit der Version 2 von MPEG-4 einige neue Profile hinzu, sodass man durchaus von einem Wildwuchs an Profilen sprechen kann. In der Hoffnung, dass mehr MPEG-4-konforme Software entsteht, wurden die Profile immer kleiner und auf spezifische Applikationen zugeschnitten. Die bekannten MPEG-4 Encoder *DivX* und *XviD* beherrschen gerade mal das *Advanced Simple* Profil, welches das *Simple* Profil erweitert um die 1/4-Pixel-Bewegungskompensation, um szenenabhängige Quantisierungstabellen, sowie um die globale Bewegungskompensation.

MPEG-4 Lizenzierung: Wie schon MPEG-2 ist auch MPEG-4 alles andere als gratis. Die MPEG (www.mpeglala.com) bietet mit der *MPEG-4 Visual Patent Portfolio License* eine Sammellizenz an, die die Nutzung von MPEG-4 im Bereich *Video-On-Demand* und in Hardware integriert regelt:

- Pro verkauften MPEG-4-Encoder/Decoder sind 0.25 US-Dollar zu bezahlen. Die ersten 50'000 Einheiten pro Jahr sind gratis.

- Nutzungsgebühren für *Video-on-Demand* oder Download: Für jeden Titel, der länger als zwölf Minuten ist, 2 Prozent der Leihgebühren, oder aber 2 US-Cent.

12.4.9 MPEG-4/AVC, H.264

Der aktuellste Videokompressionsstandard wurde gemeinsam von der MPEG und der ITU entwickelt. Bei der MPEG heisst er *MPEG-4/AVC* und bei der ITU heisst er *H.264*.

Der H.264 Standard ist etwa dreimal so effizient wie der H.262 (MPEG-2). Das heisst, dass bei vergleichbarer Qualität etwa ein Drittel der MPEG-2-Datenmenge entsteht. Allerdings ist der Rechenaufwand auch um den Faktor 2 bis 3 höher.

Er wird für beide klassischen Zielgebiete der Videokompression eingesetzt: Zum einen für sehr kleine Bitraten in mobilen Geräten (*iPod*, *Sony PSP*) oder im *Flash Video Player* und zum anderen für sehr gute Bildqualität mit hohen Bitraten wie bei der *Blue-ray Disc*. Auch der *HDTV-Stream* der über Satelliten ausgestrahlt ist mit dem H.264 komprimiert. Konkurrenzverfahren, wie Microsofts *VC1* oder die MPEG-2-Erweiterung für die *HD-DVD* konnten sich nicht durchsetzen.

12.4.9.1 Verbesserungen zum MPEG-4 Part-2 Standard

25. **Integer Transformation:** Der H.264-Standard verabschiedet sich von der klassischen 8x8-DCT und verwendet neu eine von der DCT abgeleitete Integer Transformation auf 4x4 oder 8x8 Blöcken im High-Profil. Die kleineren 4x4-Blöcke erlauben in einem Bild, wo viel passiert eine genauere Codierung mit weniger Blockartefakten (*Ringing*).
26. **Neue Quantisierung:** Die Redundanzreduktion wird neu nicht mehr durch Division mit der Quantisierungsmatrix, sondern durch Addition und Subtraktion mit derselben erreicht. Dadurch treten keine Rundungsfehler auf.
27. **Arithmetische Entropiecodierung:** Neben einer angepassten Huffman-Codierung (CAVLC) wird neu auch die noch effizientere arithmetische Codierung (CABAC) verwendet.
28. **Flexible Interframe Codierung:** Bei der Interframe Codierung gibt es keine starren GOP-Strukturen und Grössen mehr. B-Frames können sich neu auch auf andere B-Frames beziehen (*B-Pyramiden*). Ganz allgemein können Bewegungsvektoren sich auch mehrere Makroblöcke anderer Frames beziehen (*Multiframe-Prediction*).
29. **IDR-Frames:** IDR-Frames sind I-Frames, bei denen die nachfolgenden B- und P-Frames sich nur auf zeitlich nachfolgende Frames beziehen dürfen. Sie sollen bei Szenenwechseln eingesetzt werden, um einen klaren und sauberen Übergang zu gewährleisten.
30. **RDO: Laufende Qualitätskontrolle:** Die *Rate Distortion Optimization* (RDO) übernimmt eine laufende Qualitätskontrolle während des Encoding-Prozesses. Die Komprimierungsqualität wird laufend mit dem *PSNR*-Mass (s. nächstes Kapitel) ausgewertet um die Stärke der Kompression anzupassen, um die gewünschte Zielbitrate zu erreichen.

12.4.9.2 H.264-Profile

Nachfolgend die Hauptprofile des H.264 Standards:

Eigenschaft / Profil	Baseline	Main	High	High10
I-/P-/B-Frames	X/X/-	X/X/X	X/X/X	X/X/X
Multiframe-Prediction	X	X	X	X
Deblocking-Filter	X	X	X	X
Interlaced Encoding	-	X	X	X
Codierung CAVLC / CABAC	-/X	X/X	X/X	X/X
Farbräume: 4:0:0/4:2:0/4:2:2	-/X/-	X/-/-	X/X/-	X/X/-
Sample-Tiefe: 8/9/10 Bit	X/-/-	X/-/-	X/X/-	X/X/X
Transformblockgrösse: 4x4 / 8x8	X/-	X/-	X/X	X/X
Adaptive Quantisierungsmatrix	-	X	X	X
Beispiele	iPod, PSP	YouTube ²	Blue-ray	Blue-ray

2) Highquality Videos bei YouTube mit 720 Zeilen.

12.4.9.3 H.264-Level

Nachfolgend die Haupt-Levels des H.264 Standards, die nochmals feiner in Unterlevels unterteilt sind:

Level	Max. Makroblöcke / Sek.	Max. Bildgrösse in Makroblöcken	Max. Bitrate Baseline und Main Profile	Max. Bitrate High Profile	Max. Bitrate für High10 Profile	Beispiele für Auflösung / FPS
1	1'485	99	0.06 Mbit/s	0.08 Mbit/s	0.19 Mbit/s	128x96 / 30.9 176x144 / 15.0
2	11'880	396	2 Mbit/s	2.5 Mbit/s	6 Mbit/s	352x288 / 30.0
3	40'500	1'620	10 Mbit/s	12.5 Mbit/s	30 Mbit/s	720x480 / 30.0 720x576 / 25.0
4	245'760	8'192	20 Mbit/s	25.0 Mbit/s	60 Mbit/s	1920x1080 / 30.1 2048x1024 / 30.0
5	589'824	22'080	135 Mbit/s	168.6 Mbit/s	405 Mbit/s	1920x1080 / 72.3 2560x1920 / 30.7

12.4.10 Codec-Vergleich

Nachdem wir die Theorie hinter den Kompressionsstandards von MPEG-1 bis MPEG-4 kennengelernt haben, wollen wir verschiedene Codecs in der Praxis vergleichen. Als Testvideo wurde ein kleines Video der Grösse 160 x 120 erstellt, das nur 8 Sekunden dauert. Es enthält zum einen eine relativ statische Szene, wo sich nur eine Hand bewegt und zum anderen eine Szene mit einem Kamerabewegung. Mit 30 Frames pro Sekunde belegt die unkomprimierte Datei 13.6 MBytes auf der Festplatte. Alle Codecs lesen diese unkomprimierte Datei und müssen ihr Können bei den Zieldatenraten (*Target Rate*) 50, 150, 250, 350, 450 und 550 KBit/s unter Beweis stellen.



Abb. 368: 2 Testsequenzen mit einer lokalen und einer globalen Bewegung

Qualitätsmaß: PSNR

Bleibt die Frage übrig, nach welchen Kriterien die komprimierten Videos beurteilt werden sollen. Da die Zieldatenrate nur eine Wunschvorgabe ist, bleibt die effektive Dateigrösse ein wichtiges Kriterium. Diese müssen wir der durchschnittlichen Bildqualität gegenüberstellen. Wir verwenden dazu das PSNR-Mass (*Peak-Signal-to-Noise Ratio*), welches wir im Kapitel 3.2.6 über Sensorrauschen kennengelernt haben.

Verglichene Codecs

Die meisten getesteten Codecs entsprechen mehr oder weniger einem MPEG-1, MPEG-2 oder MPEG-4 Standard:

- **WMV9:** Dieser Codec ist die 3. Version von Microsofts MPEG-4 Kompressors aus dem Windows Media Framework. Der Codec ist aber nicht MPEG-4 ISO standardkonform und basiert zudem auf einem frühen Draft des H.264 Standards.
- **DivX 5.2.1:** Eine frühere Version des Codecs entstand, indem Microsofts MPEG-4 Codec gehackt wurde. Dieser war durch einen französischen Hacker namens Jerome Rota aus einer Beta-Version des Windows Media Player extrahiert worden. Die von Rota gegründete Firma *DivXNetworks*, Inc. entwickelte später eine komplett neue Version, um Patentverletzungen zu vermeiden. Der ursprüngliche Name lautete übrigens *DivX ;-)*, was ohne *Smily* die Bezeichnung war für eine kommerzielle selbst zerstörbare DVD. DivX unterstützt ab der Version 5 das *Advance Simple Profile* von MPEG-4. Eine kostenpflichtige Version kann unter www.divx.com heruntergeladen werden.
Der Codec ist mittlerweile so weit verbreitet, dass viele DVD-Player einen DivX-Decoder eingebaut haben, obwohl die meisten DivX-Filme illegale Kopien sind!
- **Xvid 1.0.2:** Wie der Name vermuten lässt, ist dieser Codec ein Verwandter von DivX. Der opensource Codec ist aus dem *OpenDivX* Projekt entstanden und unterstützt ebenfalls das *Advance Simple Profile* von MPEG-4. Wegen Patentrechtsangelegenheiten kann XviD nicht in den USA oder Japan lizenziert werden.
- **3ivx 4.5.1:** Dieser MPEG-4 Codec mit AS Profil stammt von der australischen Firma *3ivx Technologies* und kann kostenlos für Windows und für Apple QuickTime bei www.3ivx.com heruntergeladen werden.
- **mpeable 0.10:** Dieser Codec stammt von der Berliner Firma *dicas* und unterstützt als Einziger die neuste Version von *MPEG-4 H.264*. Der Codec kann von www.mpeable.com kostenlos heruntergeladen werden.
- **TMPGEnc 2.521:** Dies ist kein Codec sondern ein bekannter MPEG-1 und MPEG-2 Encoder aus Japan. Die MPEG-2 Codierung ist kostenpflichtig. TMPGEnc kann unter www.tmgenc.net heruntergeladen werden.

- **Indeo 5.11:** Dieser Codec wurde ursprünglich von Intel entwickelt und wird heute von der Firma *Indeo* betreut. Der Codec wird standardmäßig mit MS Betriebssystemen ausgeliefert und benutzt eine Wavelet-Kompression.
- **Huffyuv:** Dieser Codec von *Ben Rudiak-Gould* ist einer der am meisten verwendeten Codecs, wenn man ein Video verlustfrei oder nahezu verlustfrei abspeichern will. Er funktioniert ähnlich wie der verlustfreie Modus von JPEG mit Huffman-Codierung und kann für Videos im YUV oder RGB Format angewendet werden. Ein kleiner Verlust entsteht dabei, wenn man RGB Daten ins YUY2 Format (YUV 4:2:2) konvertiert. Der Codec ist opensource und kann unter <http://neuron2.net/www.math.berkeley.edu/benrg/huffyuv.html> heruntergeladen werden.
- **VP6:** VP6 ist der neuste Codec der amerikanischen Firma *On2-Technologies*. Diese Firma hält sich bewusst nicht an MPEG Standards, weil sie die damit verbundenen hohen Lizenziierungskosten für kommerzielle Videoanbieter massiv unterbietet will. Der Codec wird z. B. von der BBC für das Streaming von Reporterbeiträgen über Satelliten verwendet. Der grösste Erfolg dürfte aber die Ernennung zum Codec der chinesischen DVD-Weiterentwicklung *EVD (Enhanced Versatile Disc)* sein. Für nicht-kommerzielle Anwendung ist der Codec bis jetzt gratis.

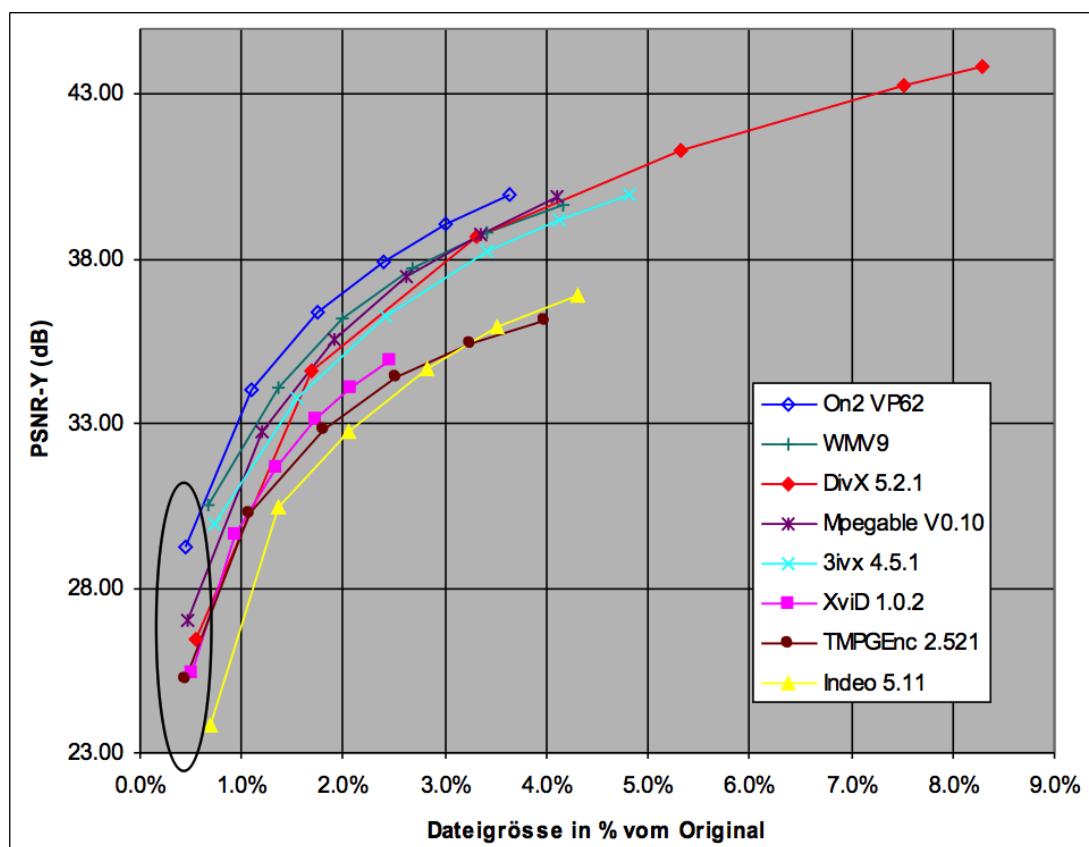


Abb. 369: Codec-Vergleich im Jahre 2010 bei 50, 150, 250, 350, 450 und 550 KBits/s.

13 Anhänge

13.1 Masseinheiten

p (Pico-) = 10^{-12} = 0,000000000001 (Billionstel)
 n (Nano-) = 10^{-9} = 0,000000001 (Milliardstel)
 μ (Mikro-) = 10^{-6} = 0,000001 (Millionstel)
 m (Milli-) = 10^{-3} = 0,001 (Tausendstel)
 c (Zenti-) = 10^{-2} = 0,01 (Hundertstel)
 d (Dezi-) = 10^{-1} = 0,1 (Zehntel)
 10^0 = 1 (Eins)
 da (Deka-) = 10^1 = 10 (Zehn)
 h (Hekto-) = 10^2 = 100 (Hundert)
 k (kilo-) = 10^3 = 1'000 (Tausend)
 M (mega-) = 10^6 = 1'000'000 (Million)
 G (giga-) = 10^9 = 1'000'000'000 (Milliarden)
 T (tera-) = 10^{12} = 1'000'000'000 (Billion)

13.2 Griechisches Alphabet

α	A	alpha	v	N	ny
β	B	beta	ξ	Ξ	xi
γ	Γ	gamma	\circ	O	omikron
δ	Δ	delta	π	Π	pi
ϵ	E	epsilon	ρ	P	rho
ζ	Z	zeta	ς	Σ	sigma
η	H	eta	τ	T	tau
θ	Θ	theta	υ	Y	psilon
ι	I	iota	ϕ	Φ	phi
κ	K	kappa	χ	X	chi
λ	Λ	lamda	ψ	Ψ	psi
μ	M	my	ω	Ω	omega

13.3 Einheiten des internationalen Einheitssystems

Nachfolgend sehen sie die Liste der 7 Basiseinheiten (auch SI-Einheiten genannt) des internationalen Einheitssystems (*Système international d'unités*) [Quelle: Wikipedia].

Grösse	Formel zeichen	Einheit	Einheits-zeichen	Definition
Länge	l	Meter	m	Länge der Strecke, die das Licht im Vakuum während der Dauer von $1/299.792.458$ Sekunden durchläuft.
Masse	m	Kilogramm	kg	Das Kilogramm ist die Einheit der Masse; es ist gleich der Masse des Internationalen Kilogrammprototyps. ¹⁾ Ein original nicht präfixierter Name für das Kilogramm war das Grave.
Zeit	t	Sekunde	s	das $9.192.631.770$ -fache der Periodendauer der dem Übergang zwischen den beiden Hyperfeinstruktur niveaus des Grundzustandes von Atomen des Nuklids ^{133}Cs entsprechenden Strahlung
Stromstärke	I	Ampere	A	Stärke eines konstanten elektrischen Stromes, der, durch zwei parallele, geradlinige, unendlich lange und im Vakuum im Abstand von 1 Meter voneinander angeordnete Leiter von vernachlässigbar kleinem, kreisförmigem Querschnitt fliessend, zwischen diesen Leitern je 1 Meter Leiterlänge die Kraft $2 \cdot 10^{-7}$ Newton hervorrufen würde
Temperatur	T	Kelvin	K	der 273.16 Teil der Differenz zwischen dem absoluten Nullpunkt und dem Tripelpunkt von Wasser
Lichtstärke	I_V	Candela	cd	die Lichtstärke in einer bestimmten Richtung einer Strahlungsquelle, die monochromatische Strahlung der Frequenz $540 \cdot 10^{12}$ Hz (= 555 nm Wellenlänge) aussendet und deren Strahlstärke in dieser Richtung $1/683$ Watt durch Steradian beträgt.
Stoffmenge	n	Mol	mol	die Stoffmenge eines Systems, das aus eben soviel Einzelteilchen besteht, wie Atome in 0.012 Kilogramm des Kohlenstoffnuklids ^{12}C enthalten sind.

14 Stichwortverzeichnis

1/4-Pixel-Bewegungskompensation.....	277	Debayering.....	52
3D-Transformkodierung.....	267	Deblocking.....	278
A/D-Wandler.....	43	Deconvolution.....	119
Abbildung.....	107	Defekte Pixel.....	44
Abbildungsfehler.....	39	Deformable Countours.....	160
Abbildungsmassstab	38	Delaunay-Triangulation.....	216
Aberration.....	39	Demosaicing.....	52
Absorption.....	15	Deringing.....	278
adaptive Schwellwerte.....	151	Dichte.....	184
Aktive Konturen.....	160	Dilatation = Maximumfilter.....	101
Aliasing.....	243	Diskrete Kosinustransformation.....	252
Amplitude.....	108	Diskrete Wavelet-Transformation.....	128
Anwendung Bildschärfung.....	90	Diskriminantenfunktion.....	216
Auflösungsgrenze.....	41	Distanzklassifikatoren.....	215
Auge.....	20	Distanztransformation.....	177
Ausdünnung.....	178	DivX.....	283
B-Bilder.....	270f.	Dunkelbildkorrektur.....	45
B-Spline Kurven.....	161	Durchmesser.....	184
Bayermusterfilter.....	51	Dynamic Range.....	43
between-class variance.....	150, 212	Eigenfaces.....	222
Bewegungsbasierte Segmentation.....	167	Eigengesichter.....	222
Bewegungskompensation.....	268, 271	Eigenvektoren.....	141, 221
Bidirectional Frames.....	270	Eigenwerte.....	141
Bild- und Videokompression.....	242	Emission.....	15
Bildanalyse.....	11	Entfaltung.....	119
Bildaufnahme.....	37	Entropie.....	244
Bildkompression.....	245	Erosion = Minimumfilter.....	101
Bildsensor.....	42	Euklidische Distanz.....	178
Bildsynthese.....	11	Euler-Charakteristik.....	190
Bildverarbeitung.....	11	Exzentrizität.....	186
Binäre Dilatation.....	94	F-Mass.....	234
Binäre Erosion.....	95	Facial Animation.....	279
Binarisierung.....	64	Facial Animation, Body Animation.....	279
Bipolar.....	21	Faltungssatz.....	118
bit.....	244	Farbmodelle.....	26
Blende.....	38	Farbspektrum.....	17
Blendenzahl.....	38	Fast Fourier Transformation.....	112
blinden Fleck.....	22	feature space.....	206
Block Matching Algorithm.....	272	feature vector.....	181, 206
Block Matching Algorithmen.....	235	Feret Durchmesser.....	184
Blockschicht.....	270	Festbrennweitenobjektive.....	41
Blooming.....	45	FFT.....	112
BMA.....	272	Fiji.....	35
Bot-Hat.....	104	Filter.....	78
Bounding Box.....	182	Filterbank.....	132
Brechung im Wellenmodell.....	18	Filterkernel.....	78, 118
Brechungsindex.....	14	Filtermaske.....	78
Brennpunkt.....	37	Firewire.....	53
Brennweite.....	37	Fisherface.....	226
Camera Link	53	Fixed Pattern Noise.....	44
CCD-Bildsensor.....	47	Fläche.....	182
Chain Code.....	170	Flat Field Correction.....	45
Choroid.....	21	FLD.....	226
Chromatische Aberration.....	39	FloodFill.....	168
Circular Hough Transform.....	136	Fourier Deskriptoren.....	174
circularity.....	183	Fourier-Transformation.....	107
Claude Elwood Shannon.....	243	Fovea.....	22
Closing.....	96, 103	Freeman Code.....	170
CMOS-Bildsensor.....	49	Frequency Domain.....	107
CMY(K)-Farbmodell.....	26	Frequenzraum.....	107
Co-occurrence Matrix.....	192	Full Search.....	236
CoaXPress.....	53	Full Well Capacity.....	43
Cornea.....	21	Gain.....	43
Corner Response Function.....	196	Gain Noise.....	44
Cos4-Gesetz.....	40	Ganglionzellen.....	21
Cos4-Vignettierung.....	40	Gaussfilter.....	81
Crack Code.....	171	Gaussische Trapezformel.....	183
Cumulative Density Function.....	57	Gegenfarben-Theorie.....	24
Dark Current Noise.....	44	gelber Fleck.....	22
Dark Field Correction.....	45	Genauigkeit.....	233

Geometrische Merkmale.....	182	KNN.....	213
Gesichtserkennung.....	220	kNN-Classifier.....	218
Giesskannenwerkzeug.....	153	Kompaktheit.....	183
GigE Vision.....	53	Kompressionsstandard.....	265
Glaskörper.....	21	Kontrastempfindlichkeit.....	25
Globale Bewegungskompensation.....	277	Kontrasterhöhung durch Farben.....	71
Globale Operatoren.....	107	Konturen.....	170
GOP.....	270	Konvexität.....	184
Gradienten.....	83	Konvolution.....	79, 239
Grauwertgradienten.....	83	Korrelation.....	79
Grauwertmatrix.....	192	Korrelationskoeffizienten.....	237
Grauwertreduktion.....	65	Kovarianz.....	140
gray level slicing.....	65	Kovarianzmatrix.....	141
Group Of Picture.....	270	Kreisförmigkeit.....	183
Gruppengeschwindigkeit.....	18	Kreisfrequenz.....	108
H.261.....	269	Kreuzkorrelationsfunktion.....	237
H.262.....	274	kumulativen Wahrscheinlichkeitsdichtefunktion.....	57
H.264.....	281	künstliches neuronales Netzwerk.....	213
Haar-Transformation.....	129	Laplace.....	87
Haralick'sche Texturmasse.....	192	Laplace of Gaussian.....	89
Harris Corner Detector.....	195	Laplace-Operator.....	87
Hauptachsen.....	187	Laser.....	16
Hauptkomponentenanalyse.....	139, 230	Lauflänge.....	245
Hauptkomponententransformation.....	139	Lauflängen.....	170
HDTV.....	266, 275	LDA.....	226
High Definition Television.....	275	Licht.....	13
Histogramm.....	56	Lichtenergie.....	16
Hit-or-Miss-Operator.....	98	Lichtquellen.....	16
Hochpassfilter.....	83	lineare Diskriminanzanalyse.....	226
Horizontalzellen.....	21	Lineare Faltung.....	79
Hornhaut.....	21	Linsengleichung.....	38
Hough Transformation.....	160	Lokale Operatoren.....	78
Hough-Transformation.....	134	Lokale Strukturmatrix.....	195
Hough-Transformation für Geraden.....	134	Lookup Tables.....	71
Hough-Transformation für Kreise.....	136	Lookup-Table.....	75
HSB-.....	29	Lumineszenz.....	15
HSI-Farbmodelle.....	29	Lumineszenzstrahler.....	16
HSL-.....	29	LZ-Kodierung.....	248
HSV-Farbmodell.....	28	Makroblock.....	270
Hu's 7 Momente.....	188	Manhattan Distanz.....	178
Huffman-Code.....	246	Masken.....	169
Huffyuv.....	284	Matlab.....	33
Hyper-Ebene.....	216	Maximum-Filter.....	101
I-Bilder.....	270	Mean absolute distance.....	235
IEEE 1394a.....	53	Mean squared distance.....	235
IEEE 1394b.....	53	Mechanische Vignettierung.....	40
Image Analysis.....	11	Mediale Achse.....	177
Image Processing.....	11	Medianfilter.....	102
ImageJ.....	33	Merkmale.....	206
ImagePlay.....	32	Merkmalsraum.....	206
Informationsgehalt.....	243	Merkmalsvektor.....	181
Informationstheorie.....	243	Mesh Object Coding.....	279
Innerklassenvarianz.....	212	Metamerie.....	24
Innerklassenvarianzen.....	150	Mexican-Hat.....	89
Interlaced.....	48	Minimal Distance Classifier.....	215
Intra-Frames	270	Minimal Energy Counters.....	160
Invariante Momente.....	188	Minimale Distanz.....	178
Invarianz.....	176	Minimumfilter.....	101
Inverse Filterung.....	119	Model-Based Coding.....	279
Invertierung.....	64	Modellbasierte Segmentierung.....	160
ISODATA-Algorithmus.....	214	Modulationsübertragungsfunktion.....	41
ITU.....	265	Morphologische Operatoren.....	94
JPEG-Kompression.....	251	Morphologische Operatoren auf Binärbildern.....	94
JPEG2000-Kompression.....	258	Morphologische Operatoren auf Graustufenbildern.....	100
k-Means Clustering.....	213	Motion Vector Coding.....	277
k-Means-Clustering.....	150	MP3.....	275
k-Mediods.....	214	MPEG.....	265, 268
k-Nearest Neighbors.....	218	MPEG-1.....	268
Kameraschnittstellen.....	53	MPEG-2.....	274
Kantenextraktion auf Graustufenbilder.....	104	MPEG-2-Levels.....	275
Karhunen-Loëve-Transformation.....	139	MPEG-3.....	275
Key-Frames.....	270	MPEG-4.....	275
Klassifikation.....	206	MPEG-4 Profile.....	280
Klassifikatoren.....	215	MPEG-4 Visual Standard.....	277

MPEG-4/AVC.....	281	Sensorrauschen.....	44
MPEG-J.....	276	Shannon Theorem.....	243
Multi-Resolution-Analyse.....	130	Shannon-Abtasttheorem.....	242
Multiskalenraum.....	198	Shape Coding.....	278
Mustererkennung.....	206	SIFT-Deskriptoren.....	198
Nachbearbeitung.....	278	Signal to Noise Ratio.....	46
Nearest Neighbor Classifier.....	218	Signal-Rausch-Abstand.....	46
Netzhaut.....	21	Signaltheorie.....	242
Noise.....	44	Signaturen.....	172
Non-Maximum Suppression.....	157	Singular Value Decomposition.....	140
normierte Kreuzkorrelationsfunktion.....	237	Singulärwertzerlegung.....	140
Nyquist-Frequenz.....	113, 242	Slices.....	270
Objektivtypen.....	41	Snakes.....	160
Opening.....	96, 103	SNR.....	46
Optik.....	37	Sobelfilter.....	86
Orientierung.....	186	Solidität.....	184
Ortsraum.....	107	Space Domain.....	107
Otsu's optimale Methode.....	150	Sphärische Aberration.....	39
P-Bilder.....	270	Split and Merge.....	154
PCA.....	139	Sprite Coding.....	279
pdf.....	57	Stäbchen.....	21
Peak Signal to Noise Ratio.....	46	Statistische Texturmerkmale.....	191
Perimeter.....	182	Strahlenmodell.....	13
Periode.....	108	SVD.....	140
Phasengeschwindigkeit.....	18	Szenengraph.....	275
Photon Noise.....	44	Telezentrische Objektive.....	41
Photonenmodell.....	15	Template Matching.....	235
Point Spread Function (PSF).....	119	Textur.....	191
Polarisation.....	20	Texturmerkmale.....	191
Posterizing.....	65	Thinning.....	178
Predicted Frames.....	270	Three Step Search.....	236
Prewitt Operator.....	84	Threshold.....	64
Principal Component Analysis.....	139	Tiefpassfilter.....	81
probability density function.....	57	Time Domain.....	107
Profiles & Levels.....	275	TIR.....	14
Progressive Scan.....	48	Top10-CV-Paper.....	241
PSF.....	89	Topologie.....	189
PSNR.....	283	Total interne Reflexion.....	14
Punktmerkmale.....	194	Trainingsphase.....	206
Punktspreizfunktion.....	89	Transformation.....	107
Pupille.....	21	Tristimulustheorie.....	23
QPel.....	277	Undercolor Removal.....	28
Quanteneffizienz.....	43	Unscharfmaskierung.....	90
Quantization Noise.....	44	USB 2.0.....	53
Quantum Efficiency.....	43	USB 3.0.....	53
QuickHull-Algorithmus.....	184	Varianz.....	140
Rangordnungsoperation.....	101	Verlustfreie Kompression.....	245
Rauschquellen.....	44	Video Object Coding.....	277
Readout Noise.....	44	Video-Codec.....	265
Rechteckfilter.....	81	Video-Formate.....	265
Redundanz.....	266	Video-Player.....	265
Reflexion.....	13	Videokompression.....	242
Refraktion.....	14	Vignettierung.....	40
Regenbogen.....	19	Viola-Jones-Algorithmus.....	241
Region Growing.....	153	VOP.....	277
Region Labeling.....	168	Voronoi-Diagramm.....	216
Retina.....	21	VP6.....	284
Rezeptoren.....	21	VRML97.....	279
RGB-Farbmodell.....	26	Wahrscheinlichkeitsdichtefunktion.....	57
Rhodopsin.....	21	Wärmestrahler.....	16
RLE.....	245	Wavelets.....	126
RMSE.....	47	Weissbildkorrektur.....	45
Roberts Cross Operator.....	84	Wellenlänge.....	108
Roberts Operator.....	84	Wellenmodell.....	17
Root Mean Squared Error.....	47	within-class variance.....	150, 212
Run Length Encoding.....	245	Xvid.....	283
Scale-Invariant Feature Transform (SIFT).....	198	YCbCr-Farbmodell.....	30
Schachbrett Distanz.....	178	Zapfen.....	22
Schärfentiefe.....	38	Zeitliche Redundanz.....	267
Schichtenmodell.....	269	Zentrale Momente.....	186
Schwellwertbasierte Segmentierung.....	149	Zoomobjektive.....	41
seed point.....	153	Zwischenklassenvarianz.....	150, 212
Sensitivität.....	233	Redundanz.....	266
Sensorabschattung.....	40	-Hat-Operation.....	104

15 Literaturverzeichnis

- Altman96: Altman, Joshua: Surfing the Wavelets
<http://www.wavelet.org/wavelet/tutorial/wavelet.htm>
- Badouel90: An Efficient Ray-Polygon Intersection
 Graphics Gems, Academic Press
- Bay08: Herbert Bay, Tinne Tuytelaars, Luc Van Gool: [SURF: Speeded Up Robust Features](#), 2008, ETH Zürich
- Bertuch01: Manfred Bertuch: Bilder schrumpfen, JPEG2000: Grundlagen und erste Anwendungen ix 8/2001: <http://www.heise.de/ix/artikel/2001/08/108/>
- Belhumeur97: P. Belhumeur, J. Hespanha, D. Kriegman: Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, July 1997
- BlinnCorner: Jim Blinn: Jim Blinn's Corner. Optimizing C++ vector Expressions
 IEEE Computer Graphics Applications
www.research.microsoft.com/~blinn/
- Blake00: A. Blake and M. Isard. Active Contours, Springer Verlag, 1st edition, 2000
- Blinn77: Jim Blinn: Models for light reflections for computer synthesized pictures
 Computer Graphics 11
- Bourke98: Paul Bourke: Image Filtering in the Freq. Domain
<http://astronomy.swin.edu.au/~pbourke/analysis/imagefilter>
- Bullman08: C. Bullmann: [Vergleich und Anwendungen von Skelettierungsalgorithmen in der digitalen Bildverarbeitung](#), 2008
- Burke97: Barbara Burke Hubbard: Wavelets – Die Mathematik der kleinen Wellen
 Birkhäuser Verlag
- Burger06: W. Burger und M. J. Burge: Digitale Bildverarbeitung, eine Einführung mit Java und ImageJ, 2. Auflage, Springer Verlag, <http://www.imagingbook.com/>
- Canny86: John F. Canny: A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-8(6):679–698, Nov. 1986.
- Cattin12: Roger Cattin: Skripts zu den Vorlesungen CPVR1-3.
- DudaHart00: R. Duda & P. Hart: Pattern Classification 2nd Ed.
- Fiedler00: Projektarbeit Datenkompression
- Flusser93: Flusser J. & Suk T.: Pattern recognition by affine moment invariants, Pattern Recognition Vol. 26, 1993
- Flusser03: Flusser J. & Suk T.: Combined Blur and Affine Moment Invariants and their use in Pattern Recognition, Pattern Recognition, Vol. 36, 2003
- Foley97: Foley, van Dam, Feiner, Hughes: Computer Graphics, Principals and Practice
 Addison Wesley
- Gonzales08: Raphael C. Gonzales und Richard E. Woods: Digital Image Processing, Pearson International Verlag.
- Haines94: Eric Haines: Point in Polygon Strategies
 Graphics Gems IV, Academic Press
- Haralick79: Robert M. Haralick: Statistical and structural approaches to texture, Proceedings IEEE, Vol. 67, 1979
- Harris88: C. Harris und M. Stephens: [A combined corner and edge detector](#). Proceedings of the 4th Alvey Vision Conference, 1988
- Heckbert90: Paul S. Heckbert: Adaptive radiosity textures for bi-directional ray tracing
 SIGGRAPH 90
- Isensee00: Pete Isensee: Fast Math Using Template Metaprogramming
 Game Programming Gems
- Jähne05: Bernd Jähne: Digital Image Processing, Concepts, Algorithms and Scientific Applications
 Springer, 6. Auflage
- Jensen99: Henrik Wann Jensen: Realistic Image Synthesis Using Photon Mapping
 A. K. Peters Ltd., <http://graphics.stanford.edu/~henrik/>
- Kalra09: Prem Kalra: Canny Edge Detection, 2009:
<http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf>
- Kass88: M. Kass, A. Witkin, D. Terzopoulos: Snakes : Active Contour Models, International Journal for Computer Vision, 1988
- Lowe99: David G. Lowe, 1999: [Object Recognition from Local Scale-Invariant Features](#),
 International Journal of Computer Vision

- Lowe04: David G. Lowe, 2004: [Distinctive Image Features from Scale-Invariant Keypoints](#), International Journal of Computer Vision
- Miko05: Krystian Mikolajczyk, Cordelia Schmid: [A performance evaluation of local descriptors](#), 2005
- MRI [The Basics of MRI](#)
- Möller99: Tomas Möller & Ben Trumbore: Fast, Minimum Storage Ray/Triangle Intersection
<http://www.acm.org/jgt/papers/MollerTrumbore97>
- Nischwitz11: Nischwitz, Fischer, Haberäcker & Socher: Computergraphik und Bildverarbeitung, Band II: Bildverarbeitung, 3. Auflage, Verlag: VIEWEG+TEUBNER
- OpenGL99: M. Woo, J. Neider, T. Davis, D. Shreiner: OpenGL Programming Guide Addison Wesley, 3rd Edition, Online Version unter:
- Paeth90: Alan W. Paeth: Median Finding on 3x3 Grid, Graphic Gems.
<http://tog.acm.org/resources/GraphicsGems/gems/Median.c>
- Peters07: Richard Allan Peters: Lectures on Image Processing
http://archive.org/details/Lectures_on_Image_Processing
- Phong75: Bui-Thong Phong: Illumination for computer generated pictures
Communication of the ACM
- Rauch99: Christian Rauch: Bildkompression mit Wavelet-Transformation
<http://home.arcor-online.de/chr.rauch/wavelet/wavelet.htm>
- Serr82: Jean Serra: Image Analysis and Mathematical Morphology, Vol. 1, Academic Press
- Serr88: Jean Serra: Image Analysis and Mathematical Morphology, Vol. 2, Academic Press
- Smith02: Lindsay I. Smith: A tutorial on Principal Components Analysis, 2002
http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
- Stollnitz95: Eric J. Stollnitz et al: Wavelets for computer graphics – A primer, Part 1
<http://www.cs.washington.edu/research/graphics/projects/wavelets/article/index.html>
- Sunday01: Dan Sunday: Intersection of Rays and Segments with Triangles
http://www.softsurfer.com/Archive/algorithm_0105/algorithm_0105.htm
- Tillikainen07: N. Petteri Tiilikainen: A Comparative Study of Active Contour Snakes, 2007
- Tönnies05: Klaus D. Tönnies: Grundlagen der Bildverarbeitung. Verlag Pearson Studium.
- Turk91: Matthew Turk & Ales Pentland: Eigenfaces for Recognition, Journal of Cognitive Neuroscience, Vol. 3, 1991. Link: <http://www.face-rec.org/algorithms/PCA/jcn.pdf>
- Whitted80: Turner Whitted: An Improved Illumination Model for Shaded Display
Communications of the ACM
- Williams92: D. J. Williams and M. Shah: [A fast algorithm for active contours and curvature estimation](#)
In: Computer Vision, Graphics, and Image Processing. Image Understanding, 1992
- Young95: David Young: Active Contour Models (Snakes),
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/YOUNG/vision7.html