Inhaltsverzeichnis

Symbolverzeichnis

$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$	in der üblichen Bedeutung
A_σ	Abstraktor (nichtlinearer Vorverarbeitungsfilter, siehe Seite 19)
$G_{\sigma,\tau}$	nichtlinearer Gaußfilter (siehe Seite 18)
$\mathcal{K}(k)$	Menge aller Kantenstücke der Kanteninformation k (siehe Seite 28)
$N_x \in \mathbb{Z}$	Bildauflösung in x-Richtung
$N_y \in \mathbb{Z}$	Bildauflösung in y-Richtung
R	kontinuierlicher Ausschnitt der Bildebene
$\mathcal{R} = R \cap \mathbb{Z}^2$	Pixelmenge (siehe Seite 12)
$\mathcal{S}(p)$	Sichtbarkeitsbereich des Pixel p (siehe Seite 59)
$S^{(\mu)}$	Shepardinterpolationsformel (siehe Seite 37)
S_k	Kantenerhaltende Grauwertinterpolation (siehe Seite 60)
S_{k,U_r}	tatsächlich implementierte Variante der kantenerhaltenden Grauwertinterpolation (siehe Seite 70)
T_σ	linearer Gaußtiefpaß zur Standardabweichung σ (siehe Seite 18)
$U_r(p)$	Umgebung von $p \in \mathcal{R}$ in der Maximumsnorm (siehe Seite 17)
$I\colon R \to [0,255]$	Intensitätsfunktion (siehe Seite 12)
$f\colon \mathcal{R} \to [0,255]$	Bild (Bilddaten im Rechner)
$k\colon \mathcal{R} \to \{+,-,0\}$	Kanteninformation im Rechner (siehe Seite 28)

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Die Disziplin *Maschinelles Sehen* beschäftigt sich im Grundsatz mit der Erforschung der Möglichkeiten, Sehleistungen auf Rechnern nachzuahmen. Innerhalb dieser grundsätzlichen Übereinkunft gibt es die unterschiedlichsten Interessenlagen und Ansätze: In den ‚minimalistischen' Ansätzen wird versucht, nur fest umschriebene einzelne Sehleistungen für ganz spezielle Anwendungen zur Verfügung zu stellen. Die ‚universellen' Ansätze demgegenüber versuchen, das Sehen als Ganzes zu modellieren, und es wird gehofft, über die gelungene Modellierung des Sehens auf Rechnern auch das biologische Sehen von Lebewesen besser zu verstehen.

Neben diesen unterschiedlichen Zielen bzw. Vorstellungen was *Maschinelles Sehen* bedeuten soll, gibt es auch völlig unterschiedliche Methoden, die zum Einsatz kommen: Das Spektrum reicht von Methoden der Signalverarbeitung (Fourieranalysis, lineare Filtertheorie), über Methoden der Variationsrechnung und der Differentialgeometrie bis zu Methoden der künstlichen Intelligenz.

Trotz aller Unterschiede in den eingesetzten Methoden und in den verfolgten Zielen gibt es im Bereich *Maschinelles Sehen* ein immer wiederkehrendes Grundproblem: Selbst die einfachsten Aufgabenstellungen sind so komplex, daß sie sich nicht auf einen Schlag lösen lassen. Um diese Aufgabenstellungen trotzdem lösen zu können, müssen sie in kleine übersichtliche Probleme aufgeteilt werden. Eine mögliche Strukturierung des Vorgehens stammt von MARR [Ma82], der diese ursprünglich für das Vorgehen zur Extraktion von Gestaltinformation aus Bildern formuliert hat. Diese Strukturierung hat sich aber auch bei anderen Problemen bewährt, so daß sie im Bereich *Maschinelles Sehen* schon fast Referenzcharakter hat.

MARR unterteilt die Gewinnung eines 3D-Modells der Umwelt ausgehend von der Intensitätsverteilung des Kamerabildes in die folgenden Schritte:

Primal sketch: in diesem Schritt werden wichtige Bildprimitive (Bildmerkmale) extrahiert. Diese Extraktion stellt einen ersten Abstraktionsschritt, weg von der reinen Intensitätsverteilung, dar. MARR formuliert das so:

> “[The primal sketch] makes explicit important information about the two dimensional image, primarily the intensity changes there and their geometrical distribution and organisation.” [Ma82, S. 37]

Zu diesen Merkmalen gehören u.a.

Helligkeitskanten: Linienhaft ausgeprägte Objekte bei welchen sich die Helligkeit in Längsrichtung nur wenig ändert, in deren Querrichtung sich die Helligkeit aber stark (abrupt) ändert.

Ecken: Abrupte Änderungen in der Richtung der Helligkeitskanten.

Texturen: Musterungen von Oberflächen u.ä.

Texturgrenzen: Analog zu Helligkeitskanten die Kanten, an denen unterschiedliche Texturen aufeinandertreffen.

Texturgradienten: Entstehen z.B. durch perspektivische Verzerrung einer einheitlich gemusterten Oberfläche.

Das Ergebnis des *primal sketch* ist eine erste symbolhafte Bildbeschreibung durch die Bildmerkmale.

$2\frac{1}{2}$-D sketch: In diesem Schritt wird eine Tiefenkarte der Szene erstellt, d.h. zu jedem Bildpunkt wird die Entfernung des abgebildeten Objekts zur Bildebene angegeben. Weiter werden hier dann Sprünge in der Tiefenkarte u.ä. lokalisiert.

3D-Modell: In diesem Schritt wird ein 3D-Modell der Umwelt in kameraunabhängigen Weltkoordinaten erstellt.

Wir sehen an MARR’s Strukturierung, die häufig als Orientierung dient, daß die wesentlichen Schritte im *Maschinellen Sehen* Abstraktionen sind. Schon im *primal sketch* haben wir einen Übergang von der ursprünglichen Intensitätsverteilung zu einfachen Symbolen. Ein Problem dieses Übergangs ist, daß wir nicht in der Lage sind, mathematisch präzise zu formulieren, welche Bildprimitive wir zur Beschreibung des Bildes benutzen wollen. So ist z.B. nicht klar, wie man eine *Helligkeitskante* definieren soll. Zur Illustration der auftretenden Probleme siehe Abbildung 0.1. In der Gesamtansicht (128×128 Pixel groß) ist die Helligkeitskante noch genau zu sehen, aber je genauer (lokaler) man hinsieht, desto schwieriger wird die Lokalisation der Kante.

Die daraus entstandene Situation wird von LEVI in [Le93] so beschrieben:

> Allein zur Kantensuche wurden über 120 verschiedene Operatoren publiziert. Diese Situation ist entstanden, weil es für die Konstruktion und Anwendung eines Operators keine ‚Kochrezepte‘

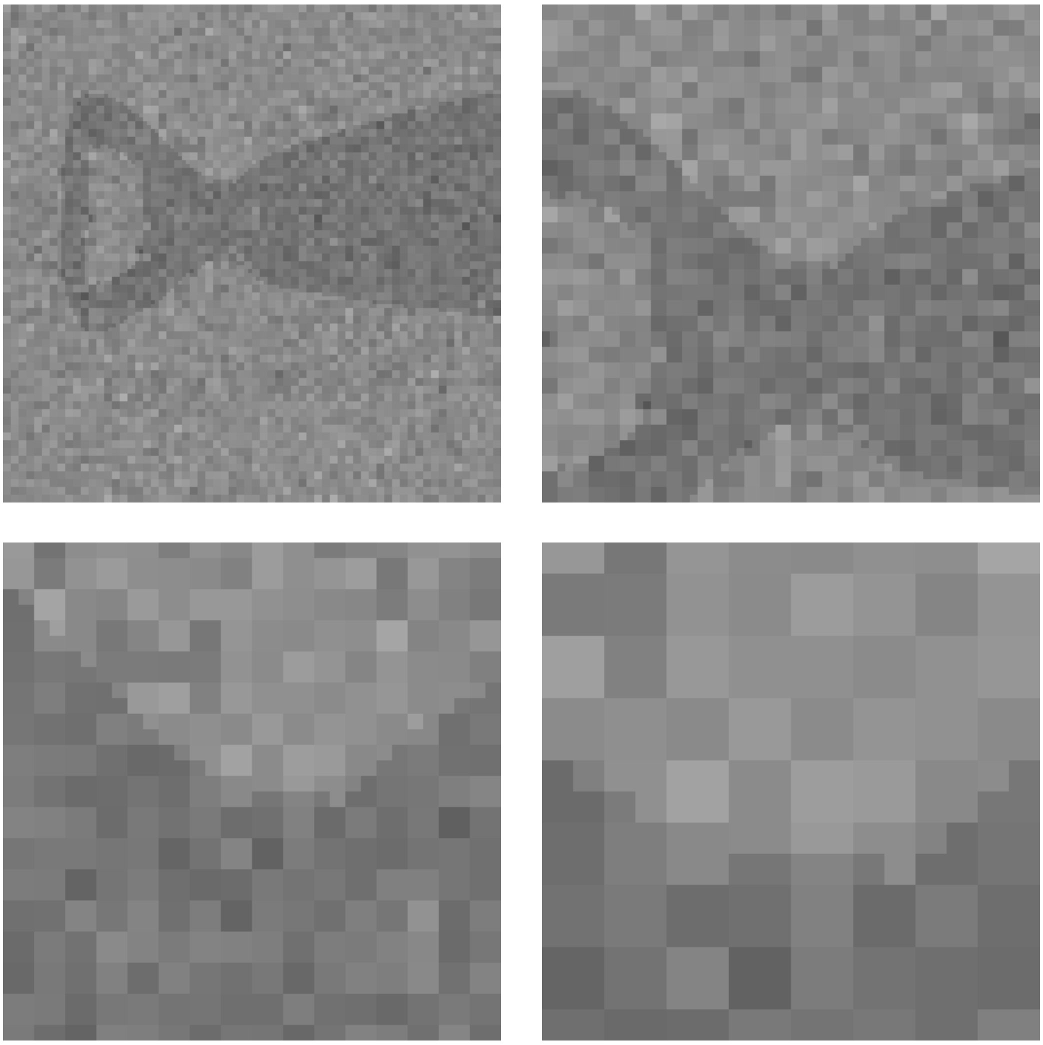


Abbildung 0.1: Was ist hier eine Helligkeitskante?

> gibt, denn die Ziele auf der Merkmalsebene, die ein solcher Operator erfüllen muß, sind nicht exakt spezifizierbar. Wäre dies möglich, so könnte man aus der Zielvorgabe einen einzelnen oder gar eine ganze Folge von Operatoren ableiten. Was wir haben sind symbolische (semantische) Zielvorgaben, die sich an menschlichen Suchvorgängen orientieren und es ist nicht bekannt, wie diese in numerische Gütevorgaben zur Merkmalssuche zu transformieren sind. [Le93, S. 107]

Trotz dieser offensichtlichen Schwierigkeiten ist es natürlich notwendig eine Merkmalsextraktion durchzuführen. Aber viele Anwendungen extrahieren Merkmale für ihre weitere Verarbeitung, ohne zu berücksichtigen, in wie weit diese Merkmale das Ausgangsbild noch ‚gut' beschreiben. Auch wir werden in Kapitel 1 zwei eng zusammengehörende Bildmerkmale, Grauwert- und Kanteninformation (etwas allgemeiner, als sonst üblich) extrahieren. Die Kapitel 2 und 3 beschäftigen sich dann mit der Frage, ob die extrahierten Merkmale das Ausgangsbild noch ‚gut' beschreiben. Dazu wird in Kapitel 2 ein Rekonstruktionsverfahren angegeben, mit dessen Hilfe wir aus den extrahierten Merkmalen wieder ein Bild konstruieren können. Auf diesem Weg ist es dann möglich, das Ausgangsbild mit dem rekonstruierten Bild qualitativ zu vergleichen. Im Kapitel 3 wird dann versucht quantitativ zu fassen, wieviel Information noch in den extrahierten Merkmalen steckt.

In MARR's Strukturierung der Vorgehensweise zur Konstruktion eines 3D-Modells der Umwelt haben wir gesehen, daß sich die Beschreibung der aufgenommenen Szene immer mehr von der ursprünglichen Intensitätsfunktion entfernt. Die Beschreibung erfolgt in zunehmend abstrakteren Begriffen, vernachlässigt notwendigerweise Bilddetails auf Pixelniveau und ergänzt die Bildbeschreibung um externes Wissen (das nicht aus dem Bild entnommen ist) und Annahmen.

Unter diesem Aspekt stellt auch unsere Extraktion der Grauwert- und Kanteninformation eine Abstraktion dar, denn dieser in Kapitel 1 vorgestellte Extraktionsprozeß eliminiert Helligkeitsschwankungen, die räumlich nur wenig ausgedehnt und die nur von geringem Kontrast sind. Wir erhalten dadurch große zusammenhängende Flächen, innerhalb derer sich die Grauwerte nur wenig ändern. Dieses wird durch einen speziellen nichtlinearen Filter, den *nichtlinearen Gaußfilter* erreicht (siehe auch [AW95]).

Die Elimination von kleinen Helligkeitsschwankungen wird natürlich auch von einem linearen Gaußtiefpaß erreicht. Da dieser aber die hohen Frequenzen im Fourierspektrum des Bildes abschneidet, werden dadurch notwendigerweise starke Helligkeitskanten verschmiert. Der *nichtlineare Gaußfilter* hat dieses Problem nicht. Er glättet kleine Helligkeitsschwankungen ohne dabei die großen Helligkeitssprünge, die unsere Kanten bilden, zu verändern. Zusätzlich zur Lokalisierung der Kante gehört zu unserer Kanteninformation auch die Information, welches die dunkelere und welches die hellere Seite der Kante war. Diese Information ist für die Rekonstruktion wichtig.

Die Grauwertinformation gewinnen wir durch Unterabtastung des mit dem *nichtlinearen Gaußfilter* vorverarbeiteten Bildes. Wäre unser vorverarbeitetes Bild bandbreitenbeschränkt (d.h. die Fouriertransformierte hat kompakten Träger), so könnten wir aus einer Unterabtastung des Bildes (mit der passenden Frequenz) das gesamte Bild rekonstruieren (SHANNON's Abtasttheorem). Unser vorverarbeitetes Bild ist aber nicht bandbreitenbeschränkt, denn wir wissen, daß es an Helligkeitskanten starke Sprünge hat. Innerhalb einer Grauwertfläche, also zwischen den Kanten, variieren die Grauwerte aber nur wenig. Vor diesem Hintergrund betrachtet, repräsentieren Grauwert- und Kanteninformation unterschiedliche Frequenzkomponenten des Fourierspektrums. Die Grauwertinformation repräsentiert den tieffrequenten Anteil im Spektrum und die Kanteninformation repräsentiert die hochfrequenten Anteile des Spektrums, die zu örtlich lokalisierten Sprüngen gehören. (Dieses ist nicht exakt richtig, gibt aber die Idee gut wieder.)

Wenn wir das Bild nur noch durch die Grauwert- und Kanteninformation repräsentieren, dann haben wir die räumlich nicht eng lokalisierten hochfrequenten Anteile des Spektrums vernachlässigt. Für solche Anteile gibt es zwei wesentliche Quellen in einem Bild. Die eine ist Rauschen und die andere ist Textur. Bei unserem Ansatz verzichten wir auf beides. Die Hoffnung für die Zukunft ist, daß es möglich sein wird, durch spezielle ‚Texturfilter' auch diese Bestandteile weiter aufzutrennen. Es wird bereits daran gearbeitet.

In Kapitel 2 werden wir sehen, daß in von uns extrahierten Merkmalen alle ‚wesentlichen' Bildinhalte erhalten geblieben sind, denn wir werden eine Rekonstruktionsmethode angeben, mit deren Hilfe es möglich ist, aus den Merkmalen wieder ein Bild zu gewinnen. Eine visuelle Inspektion der Rekonstruktion im Vergleich zum Original zeigt, daß alle größeren Strukturen des Bildes erhalten geblieben sind (siehe z.B. das Testbild TASTATUR auf Seite 146 in Anhang A).

Die verwendete Rekonstruktionsmethode nennen wir *kantenerhaltende Grauwertinterpolation*, da sie die vorgegebenen Abtastwerte der Grauwertinformation interpoliert und gleichzeitig vorgeschriebene Unstetigkeitsstellen, nämlich unsere Helligkeitskanten der Kanteninformation erzeugt. Der Interpolationsteil der *kantenerhaltenden Grauwertinterpolation* basiert auf einer Interpolationsformel, die 1968 von SHEPARD in [Sh68] vorgestellt wurde. Die gesamte Rekonstruktionsmethode mußte allerdings erheblich modifiziert werden, da die Shepard-Interpolation in ihrer Urform keine Unstetigkeitsstellen zu erzeugen vermag. Unsere Methode ist gerade aus diesem Grund auch für sich interessant, da uns kein Interpolationsverfahren bekannt ist, bei welchem man Unstetigkeitsstellen vorgeben kann.

In Kapitel 2 haben wir uns der Frage, wieviel Information noch in den extrahierten Merkmalen steckt, qualitativ genähert. Die Antwort dort ist: Es ist noch soviel Information enthalten, daß wir ein dem menschlichen Betrachter sehr ähnlich vorkommendes Bild rekonstruieren können.

In Kapitel 3 wollen wir die Frage quantitativ betrachten. Störend dabei ist, daß es keinen brauchbaren Begriff von Informationsgehalt für natürliche Bilder gibt. Deswegen versuchen wir in Kapitel 3 eine quantitative Antwort dadurch zu geben, daß wir für unsere Merkmale eine verlustfreie Datenkompressionsmethode angeben, und den benötigten Speicherplatz der ‚komprimierten Merkmale' als Maß benutzen. Dabei stellt sich heraus, daß sich der verbrauchte Speicherplatz in der Größenordnung von 1% bis 5% der Ausgangsgröße eines Bildes bewegt.

Diese Größenordnung der Kompression wird aber auch von ,normalen' verlustbehafteten Bilddatenkompressionsverfahren erreicht, so daß sich die Frage stellt, in welcher Hinsicht sich unsere Ad-hoc-Kompression der extrahierten Bildmerkmale von den Bilddatenkompressionsverfahren unterscheidet. Der letzte Teil von Kapitel 3 ist dieser Frage gewidmet. Wir sehen dort, daß die Bilddatenkompressionsverfahren (wir nennen sie Vergleichsverfahren) keine Abstraktion im Sinne einer für den Bereich *Maschinelles Sehen* brauchbaren Merkmalsextraktion liefern.

Die Vergleichsverfahren arbeiten auf der Intensitätsverteilung und erreichen die Kompressionswirkung dadurch, daß sie in einer transformierten Darstellung der Intensitätsfunktion Verluste einführen. Die eingeführten Verluste machen sich aber nach der Rücktransformation auf die Ebene der Intensitätsverteilung als zusätzliche neue Strukturen im Bild bemerkbar.

Unsere Kompression erreicht ihre Kompressionswirkung dadurch, daß sie im Bild ,unwesentliche' Strukturen wegläßt, indem sie die extrahierten Merkmale komprimiert. In diesem Sinne beruht unsere Kompression auf (den Prinzipien von maschinellem) Sehen.

Der wesentliche Unterschied zwischen unserer Kompression und den Vergleichsverfahren ist nicht die Größe der Kompressionswirkung, sondern ist die Möglichkeit mit den (de)komprimierten Bildern maschinell weiterzuarbeiten. Die Vergleichsverfahren komprimieren vor dem *primal sketch*, so daß die Einführung von Zusatzstruktur den noch auszuführenden *primal sketch* erschwert. Unsere Kompression komprimiert nach Ausführung (eines Teils) des *primal sketch*. Aus diesem Grund sollte man dieser Art von Kompression bei beabsichtigter späterer maschineller Weiterverarbeitung der Bilder den Vorzug geben.

Kapitel 1

Extraktion von Bildinformationen

In der Einleitung ist schon von den Grauwertinformationen und den Kanteninformationen eines Bildes die Rede gewesen. In diesem Kapitel wollen wir zeigen, wie diese Informationen aus einem gegebenen Grauwertbild gewonnen werden können. Ein ganz wesentlicher Bestandteil dieser ‚Informationsgewinnung' wird eine spezielle Art von Bildvorverarbeitung sein, die wir kurz mit *kantenerhaltender Glättung* beschreiben können. Doch bevor wir diese Art der Glättung und die darauf aufsetzenden Mechanismen zur Informationsextraktion genauer beschreiben, wollen wir uns erst einmal ansehen, warum ein Bild überhaupt ‚geglättet' werden muß. Wir werden uns also zunächst mit den möglichen Bildfehlern beschäftigen, die ein Bild auf dem Weg von der Außenwelt in den Rechner verändern können.

1.1 Bildaufnahme und -digitalisierung

Wir haben in der Einleitung auf MARR's Strukturierung Bezug genommen und den ersten Schritt, den *Primal Sketch*, als den untersten Bearbeitungsschritt, der direkt auf den Bilddaten arbeitet, beschrieben. Dieses ist nur aus der Sicht des *Maschinellen Sehens* richtig, denn hier betrachten wir die Bilddaten – wie sie im Rechner angekommen sind – als grundlegend gegeben und wenig beeinflußbar. Diese Sicht ist sicher nicht die, die in den Ingenieurwissenschaften oder der Photogrammetrie vertreten wird, wo man mit ‚Bilddaten' eher das direkte Kamerasignal bezeichnet.

Dieser Abschnitt handelt von der Vorgeschichte unserer Bilddaten und soll exemplarisch den Weg der Daten, beginnend beim Kamerasignal, in den Rechner beschreiben. Insbesondere wollen wir uns die möglichen Fehlerquellen ansehen, die unsere Bilddaten verändern, verfremden oder verfälschen können. Man spricht dann meist davon, daß das Bild verrauscht sei und modelliert den Bildfehler (der Begriff unterstellt, daß es das ‚wahre' Bild wirklich gibt!) über einen Rauschterm, da man in aller Regel kein besseres Modell für die Fehler hat. Eine genauere Kenntnis der Fehler bzw. der möglichen Fehlerursachen ist aber wichtig, um beurteilen zu können, welche Merkmale in den digitalisierten Daten (unserem Bild) mit Merkmalen der aufgenommenen Szene korreliert sind und welche als Artefakte aufgefaßt werden müssen.

Es gibt viele Möglichkeiten der Bildgewinnung. Das nachfolgend aufgezeigte Verfahren der Aufnahme mit einer Videokamera ist das Verfahren, mit welchem alle unsere ‚lokalen' Testbilder erzeugt wurden (siehe Tabelle 3.1 auf Seite 81). Darüber hinaus gibt es andere Möglichkeiten, z.B. Scannen von gedruckten Papiervorlagen oder Scannen von Filmvorlagen, wie es bei der Produktion von Photo-CDs praktiziert wird. Jedes Verfahren birgt andere mögliche Fehlerquellen.

Wir wollen uns im folgenden einen typischen Weg ansehen, um zu einem Bild zu gelangen. Wir wollen dabei unter ‚Bild' bzw. ‚Bilddaten' die in einem Rechner gespeicherten Grauwerte verstehen. Wir betrachten ein konventionelles Szenario: Eine Videokamera steht irgendwo in der Landschaft (meist ist diese Landschaft eine eher langweile Laborumgebung) und ist auf eine Szene gerichtet. Diese Szene wird durch die Linse der Kamera auf die Bildebene innerhalb der Kamera projiziert. Bei dieser Projektion treten alle Probleme und Fehlerquellen auf, die auch aus der Optik bekannt sind. Die Fehler, die hier entstehen, sind aber für Anwendungen im Bereich *Maschinelles Sehen* meist vernachlässigbar klein.

Bei den üblichen Videokameras befindet sich in der Bildebene ein lichtempfindlicher CCD-Chip (CCD steht für Charge Coupled Device). Ein typischer CCD-Chip besteht an der Oberfläche aus vielen kleinen lichtempfindlichen Flächen, die in einem Rechteckraster auf der Oberfläche angeordnet sind. (Wir beschränken uns hier auf sog. Arraysensoren.) In diesen Flächen werden durch eintretende Photonen Ladungen erzeugt. Diese Ladungen werden dann durch ebenfalls an der Chipoberfläche liegende Leiterbahnen einem Verstärker zugeführt. Insgesamt macht die lichtempfindliche Fläche bei typischen CCD-Chips weniger als 50 % der gesamten Chipoberfläche aus. (Diese Angabe hängt von der Bauweise des jeweiligen CCD-Chips ab.)

Am Ausgang der Kamera wird ein mit den nötigen Synchronisierungssignalen versehenes analoges Videosignal geliefert. Dieses analoge Videosignal wird von einem Videodigitalisierer (Framegrabber) weiter verstärkt, zeilenweise zeitlich diskretisiert und diese zeitlich diskreten Spannungswerte werden dann über einen Analog-Digital-Wandler (ADU) quantisiert.

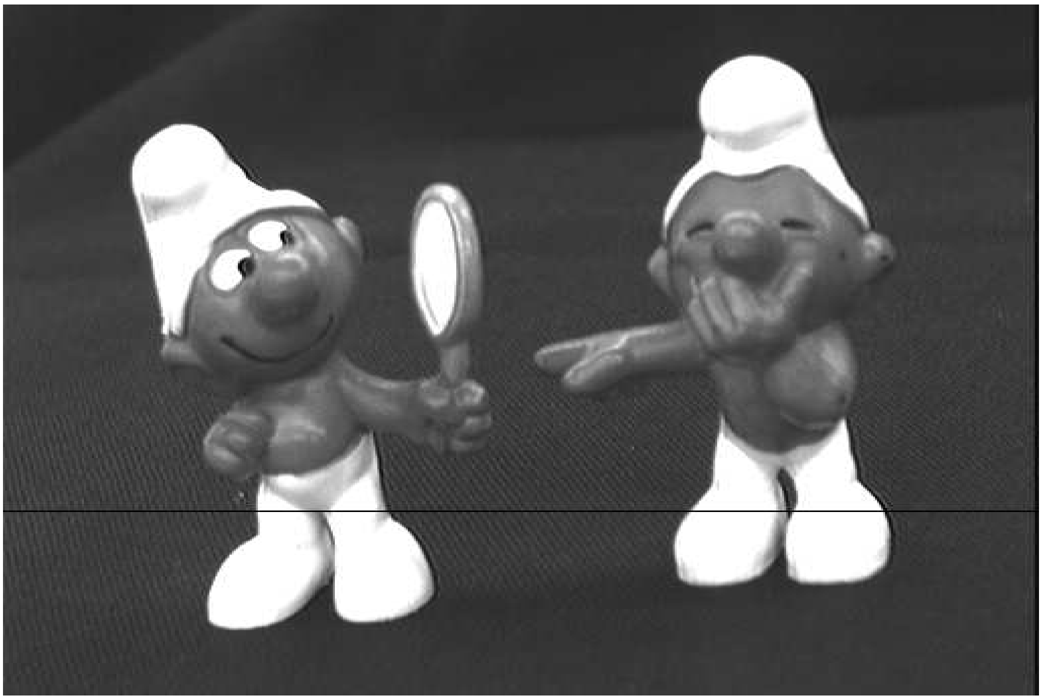


Abbildung 1.1: Testbild Schlumpf mit eingezeichneter Zeile 375.

Was wir auf diese Weise erhalten, sind Bilddaten, die ihre Zeilenstruktur durch die Zeilenstruktur des analogen Videosignals erhalten, welches in der Kamera produziert wird. Die Spaltenstruktur der Bilddaten wird durch das Timing des Videodigitalisierers bestimmt. Dieser legt fest, wieviele Abtastpunkte pro Videozeile erzeugt werden. Bei dem für unsere Testbilder verwendeten Videodigitalisierer kann man zwischen 512 und 768 Pixel pro Bildzeile wählen. Unsere selbst aufgenommenen Testbilder sind mit 512 Pixel pro Zeile digitalisiert, was zu einem Seitenverhältnis von 3:2 führt. In die Werte der Bilddaten gehen dann noch z.B. Verstärkerrauschen und Nichtlinearitäten der ADU-Kennlinie ein.

Unterschiedliche Bildfehler (optisch oder eleltronisch bedingt) können auf komplexe Weise miteinander in Beziehung stehen. Es nicht in jedem Fall klar, ob in bestimmten Teilen eines Bildes das ‚Rauschen' durch echtes Verstärkerrauschen oder möglicherweise durch Aliaseffekte entstanden ist.

Betrachten wir beispielsweise unser Testbild Schlumpf (Abbildung 1.1), welches ein auf oben geschilderte Weise digitalisiertes Kamerabild ist. Wir haben bei diesem Bild die Zeile 375 gesondert eingezeichnet, denn die nächsten Abbildungen zeigen das Grauwertprofil entlang dieser Zeile. In den Profildar-

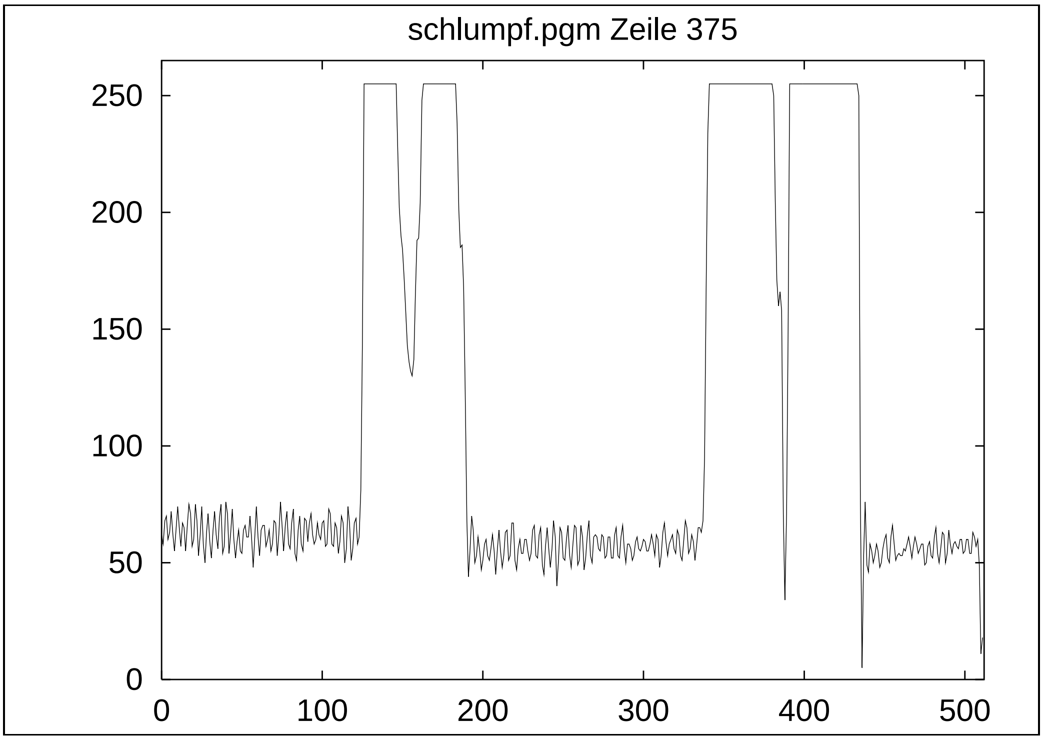


Abbildung 1.2: Rauscheffekte (?) in Abbildung 1.1 (Zeile 375)

stellungen sind auf der Abszisse die Pixelpositionen aufgetragen, wobei der Wert 0 der ganz linken und der Wert 511 der ganz rechten Position entspricht. Auf der Ordinate sind die entsprechenden Grauwerte aufgetragen.

Der Darstellung des Profils in Abbildung 1.2 ist zu entnehmen, daß die ersten ca. 100 Grauwerte um einen mittleren Grauwert von ca. 60 verteilt sind. Man könnte auf die Idee kommen, daß ein einfarbiger Hintergrund etwas verrauscht ist. Betrachten wir dagegen dasselbe Grauwertprofil in der Darstellung von Abbildung 1.3 auf der nächsten Seite, so erkennen wir, daß es sich bei dem vermeintlichen Rauschen möglicherweise um Aliaseffekte handelt, denn in der Punktwolke können wir deutlich Spuren von tieffrequenten Schwingungen wahrnehmen. Diese Spuren entstehen auch, wenn eine Schwingung (z.B. Sinusschwingung) einer bestimmten Frequenz mit einer wesentlich kleineren Frequenz abgetastet wird. Ein Blick in das Bild zeigt uns, daß der Hintergrund (ein Stück Stoff) texturiert ist, also die Vermutung über Aliaseffekte nicht unbegründet ist.

In dieser sehr übersichtlichen Situation können wir Menschen die Hinweise auf die Textur im Bild gut erkennen. Die Erkennung der Textur – oder zumindest von einigermaßen regelmässigen Schwingungsmustern – im Grauwertprofil ist auch für uns Menschen stark an die gewählte Darstellung

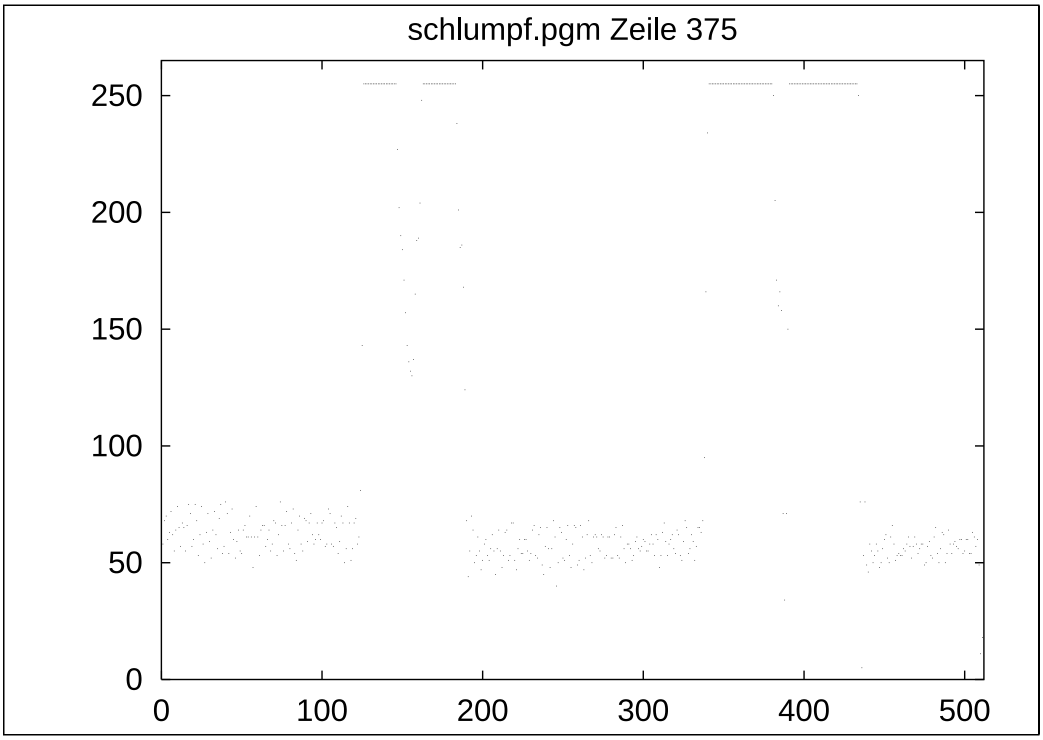


Abbildung 1.3: Aliaseffekte (?) in Abbildung 1.1 (Zeile 375)

gebunden. In der Darstellung von Abbildung 1.2 wurden die Grauwerte benachbarter Pixelpositionen durch Geradenstücke verbunden. Dadurch wird unsere Aufmerksamkeit stark auf direkte Nachbarn fixiert und wir können die möglichen Aliaseffekte nicht mehr wahrnehmen. Erst wenn die Grauwerte unverbunden dargestellt werden, können wir Beziehungen zwischen Grauwerten über größere räumliche Distanz wahrnehmen (tiefe Schwingungsmuster). Diese Unterschiede im menschlichen Wahrnehmungsvermögen stehen in starker Analogie zu den Möglichkeiten maschineller Verfahren zur Texturerkennung.

Es ist in Abbildung 1.1 durch lokale Verfahren nicht leicht entscheidbar, wodurch die leichten Helligkeitsvariationen verursacht sind. Dies ist ein Grund, warum wir im folgenden bewußt auf kleine Strukturen verzichten, sei es nur ‚Rauschen‘ oder sei es ‚wirkliche Textur‘ aus der realen Szene.

Die verfolgte Absicht ist, von einem vorliegenden Bild erst einmal die größeren Strukturen zu betrachten, in der Hoffnung, dadurch maschinell Hypothesen über den Bildinhalt generieren zu können. Dazu soll unsere Arbeit die notwendige Voraussetzung schaffen: Die Abstraktion von kleinen Bilddetails. Im Bereich *Maschinelles Sehen* existieren zur Zeit aber noch keinen gut einsetzbaren ‚Hypothesengenerierer‘, so daß das Ergebnis der Abstraktion zur Zeit nur der visuellen Kontrolle von Menschen unterzogen werden kann, um zu testen, ob die ‚wesentlichen Bildinhalte' erhalten geblieben sind.

Die so maschinell gewonnenen Hypothesen können dann in weiteren Verarbeitungsschritten als Arbeitsgrundlage dienen, die im ersten Schritt nicht berücksichtigte Feinstruktur zu interpretieren. Aus der Beschreibung ist klar, daß dies ein Arbeitsprogramm für Jahre (und Jahrzehnte) ist und unsere Arbeit nur einen kleinen Beitrag leisten kann.

1.2 Modellierung der Bilddaten

Bevor wir uns der genauen Beschreibung der Gewinnung der relevanten Bildinformationen widmen, wollen wir in diesem Abschnitt ein wenig Terminologie einführen.

Das durch die Kameralinse auf die Bildebene fallende Licht wird modelliert durch eine Intensitätsfunktion $I\colon R \to [0, 255]$. Dabei bezeichnet $R \subset \mathbb{R}^2$ den endlichen Ausschnitt der Bildebene, der von dem lichtempfindlichen Sensor abgedeckt wird. Wir wählen die Koordinaten in der Bildebene so, daß $R = [-\frac{1}{2}, N_x - \frac{1}{2}] \times [-\frac{1}{2}, N_y - \frac{1}{2}]$ gilt. Dabei sind N_x bzw. N_y die ganzzahligen Auflösungen in x- bzw. y-Richtung des Bildsensors.

Unsere Bilddaten (wir sprechen auch einfach von ‚Bild') werden modelliert durch eine Funktion $f\colon \mathcal{R} \to [0, 255] \cap \mathbb{Z}$. Der Definitionsbereich, die Pixelmenge $\mathcal{R}$, ist gegeben durch $\mathcal{R} = R \cap \mathbb{Z}^2 \subset \mathbb{Z}^2$.

Es gibt im wesentlichen zwei unterschiedliche Sichtweisen der ‚wahren Natur' eines Pixels $p \in \mathcal{R}$. In der einen Sichtweise ist p ein Punkt in $\mathbb{Z}^2$ und der Wert $f(p)$ ist (modulo Fehlerterm $e(p)$) durch Abtastung der Intensitätsfunktion entstanden, d.h. es gilt

$$f(p) = \lfloor I(p) + e(p) \rfloor.$$

Diese Interpretation eines Pixels p wollen wir die *Punktinterpretation* nennen.

In der anderen Sicht ist ein Pixel eine (kleine) Fläche S_p

$$S_p = [x - \tfrac{1}{2}, x + \tfrac{1}{2}] \times [y - \tfrac{1}{2}, y + \tfrac{1}{2}]$$

und $p = (x, y) \in \mathcal{R}$ repräsentiert ihren Mittelpunkt. Wir nennen dieses die *Flächeninterpretation* eines Pixels. In dieser Interpretation stellen wir uns den Wert $f(p)$ als durch ein Flächenintegral

$$f(p) = \lfloor \int_{S_p} I(q)\, dF(q) + e(p) \rfloor$$

gegeben vor und die Pixelmenge $\mathcal{R}$ ist eine Menge von ausgezeichneten Punkten im $\mathbb{R}^2$.

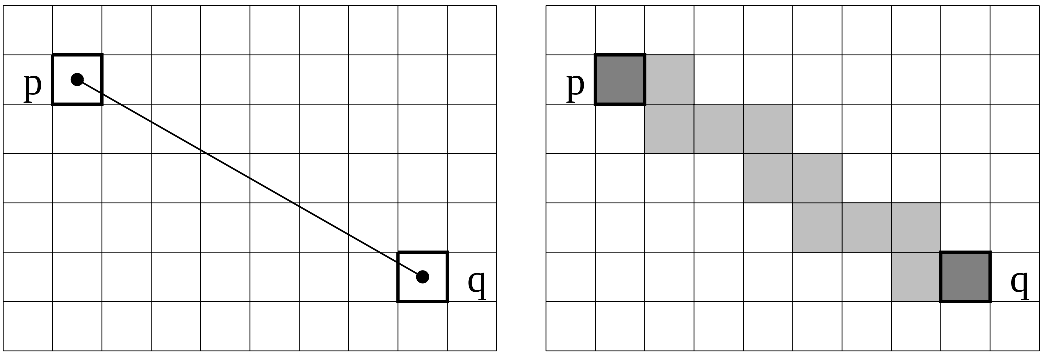


Abbildung 1.4: Zwei Möglichkeiten zur Definition der Verbindungsstrecke zwischen zwei Pixeln.

Der genaue mathematische Zusammenhang zwischen I und f ist für uns in dieser Arbeit nicht weiter interessant, da wir nicht auf die Intensitätsfunktion zurückschließen wollen. Die Formeln dienen hier nur als Modell der verschiedenen Sichtweisen und wir haben schon weiter oben gesehen, daß keine der beiden Formeln die Realität in einer CCD-Kamera richtig abbildet.

Wir werden innerhalb dieser Arbeit zwischen den beiden Interpretationen der Natur eines Pixels wechseln, je nachdem welche Sicht gerade bei einem vorliegenden Problem den methodisch günstigeren Ansatz liefert.

So gibt es beispielsweise zwei Möglichkeiten die Verbindungsstrecke zweier Pixel $p, q \in \mathcal{R}$ zu definieren. Bei der einen Definition ist die Verbindungsstrecke eine Teilmenge von $\mathcal{R}$. So wird es in der diskreten Geometrie gemacht. Für unsere spätere Anwendung ist aber die folgende Definition die angemessene.(Siehe Abbildung 1.4.)

Definition: *Gegeben seien die Pixelmenge $\mathcal{R}$ und zwei Pixel $p, q \in \mathcal{R}$ (in der Flächeninterpretation). Die Verbindungsstrecke $\overline{pq}$ ist gegeben durch die Verbindungsstrecke der Mittelpunkte der Pixel p und q im $\mathbb{R}^2$.*

$$\overline{pq} := \{r \in \mathbb{R}^2 \mid r = \lambda p + (1 - \lambda)q,\ \lambda \in [0, 1]\} \subset \mathbb{R}^2$$

1.3 Glättung von Bildern

In der Geschichte der Bildverarbeitung sind die Aufgaben ‚Bildglättung‘ und ‚Kantenfinden‘ stark miteinander verwoben. In den Bereichen der Bildverarbeitung, in denen es nicht um Bildbesserung für menschliche Betrachter geht, ist die Glättung eines Bildes in aller Regel eine direkte Vorstufe zur Kantendetektion. Dies ist dadurch begründet, daß die meisten Verfahren versuchen Helligkeitskanten über diskrete Ableitungen zu finden. Dieses Vorgehen ist aber sehr rauschempfindlich, so daß das Bild mit dem einen oder anderen Glättungsverfahren vorher geglättet wird. Die dazu verwendeten linearen Glättungsfilter haben aber das Problem, daß sie im wesentlichen die hohen Frequenzen des Fourierspektrums eines Bildes dämpfen (oder abschneiden). Dadurch wird eine ideale Kante verschmiert, so daß die Kantenlokalisation zusätzlich erschwert wird.

Das Problem bei der Glättung von Bildern zur späteren besseren Detektion von Kanten ist, daß die für diesen Glättungsprozeß häufig benutzten linearen Filter die auftretenden hohen Frequenzanteile nicht differenziert verarbeiten können. Sie unterscheiden nicht, ob die hohen Frequenzanteile durch wenig lokalisierte schnelle kleine Schwingungen oder durch gut lokalisierte große Sprünge in den Grauwerten verursacht sind. Genau diese Unterscheidung muß aber getroffen werden, wenn wir große Sprünge (Kanten) erhalten und kleines Rauschen einebnen wollen. Diese Unterscheidung kann aber im Frequenzbereich nicht getroffen werden, d.h. also daß diese kantenerhaltende Glättung nicht durch lineare Filter erfolgen kann.

Das Problem ist also nicht, daß geglättet wird, sondern wie geglättet wird.

Wie wir in Abbildung 1.5 und 1.6 sehen können, führt eine Glättung des ‚Rauschanteils' eines Bildes mit linearen Filtern zwangsläufig zu einer Delokalisierung der Kanten (hier demonstriert mit einem linearen Gaußtiefpaß, siehe auch Seite 18). Diese Delokalisierung macht sich dadurch bemerkbar, daß der Anstieg der Helligkeitskante nicht mehr so steil ist, wie im Original, und das, obwohl die Kanten sehr deutlich ausgeprägt sind. Abbildung 1.6 zeigt den Effekt, daß der lineare Gaußtiefpaß an einer Kante mit unsymmetrischem Helligkeitsprofil nicht nur den Sprung verschmiert, sondern zusätzlich noch einen unsymmetrischen Verlauf erzeugt, was die Lokalisierung der ursprünglichen Sprungstelle weiter erschwert.

Um diesem prinzipiellen Dilemma zu entgehen, müssen wir die Glättung mit einem nichtlinearen Verfahren durchführen. Der von uns favorisierte Ansatz ist der sog. *nichtlineare Gaußfilter* wie er von WEULE in [Weu94] beschrieben wurde. Dieser *nichtlineare Gaußfilter* liefert ähnliche Ergebnisse in der Glättung wie das Verfahren von BLAKE und ZISSERMAN (siehe [BZ87]) ist aber wesentlich besser parallelisierbar. Wir wollen im folgenden die Wirkungsweise dieses *nichtlinearen Gaußfilters* kurz beschreiben, da auf diesem die Extraktion unserer Bildinformation aufsetzt.

Wir sehen uns zunächst einige (einfache) lineare Glättungsfilter an, um zu verstehen, wie sich der *nichtlineare Gaußfilter* von ihnen unterscheidet. Dazu werden wir die Glättungsfilter

- Spalttiefpaß,
- Binomialtiefpaß und

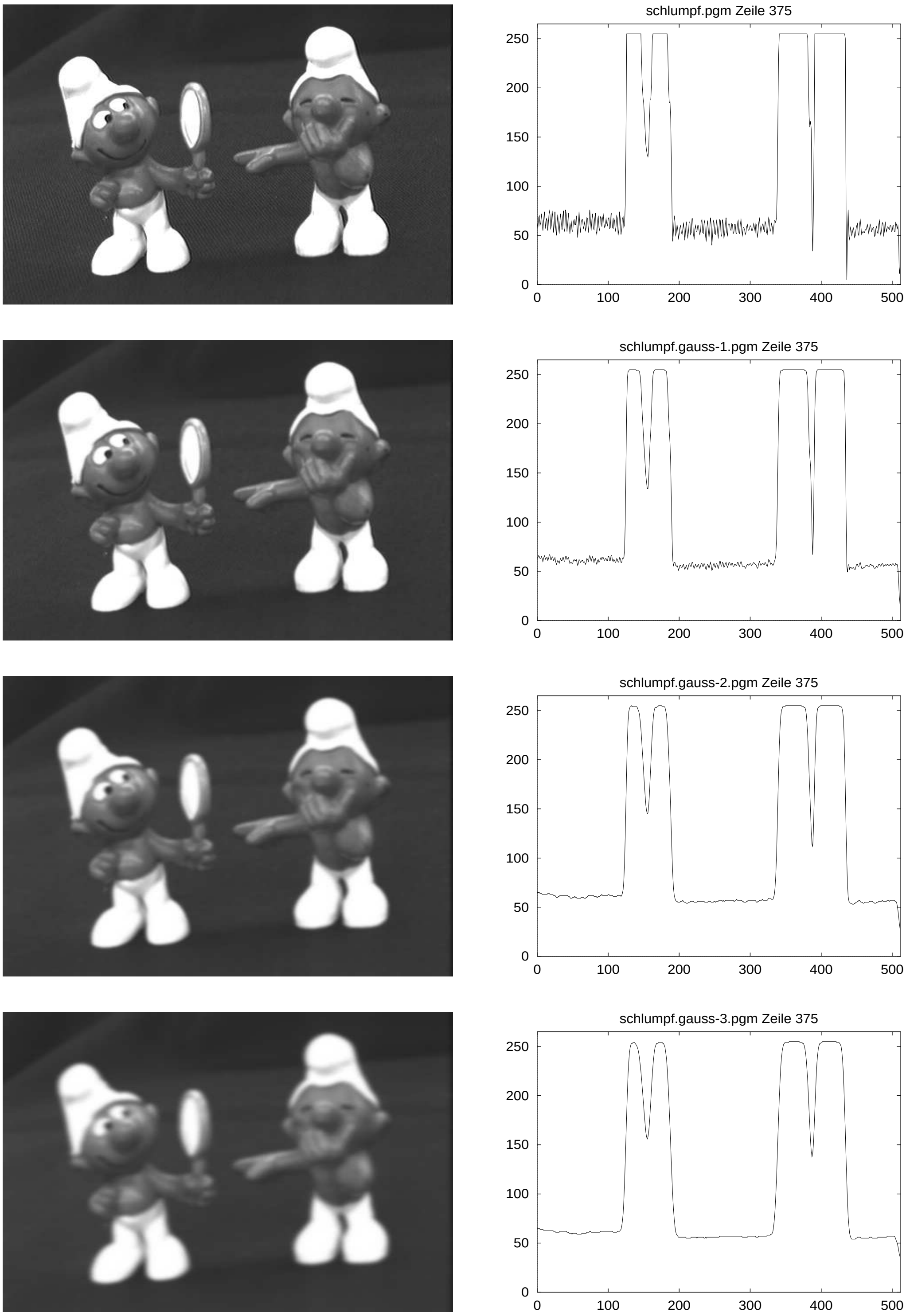


Abbildung 1.5: Testbild SCHLUMPF (Original ganz oben) gefiltert mit Gaußtiefpaß zu den Werten $\sigma = 1$, $\sigma = 2$ und $\sigma = 3$ (von oben nach unten).

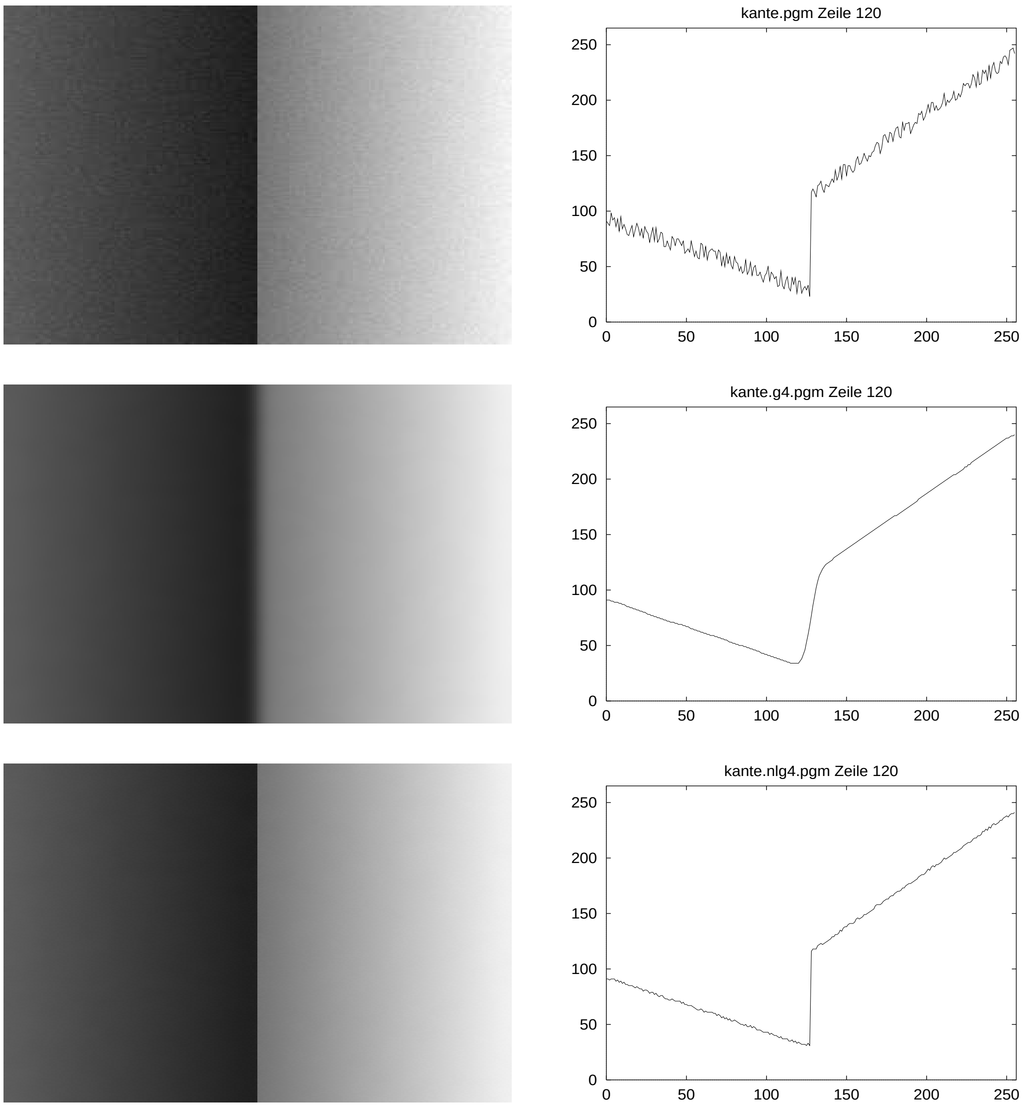


Abbildung 1.6: Eine Helligkeitskante mit nicht symmetrischem Profil (oben). Der Gaußtiefpaß glättet das Rauschen und verschmiert die Kante unsymmetrisch (mitte). *Nichtlinearer Gaußfilter* (unten).

- Gaußtiefpaß

betrachten und diese dann mit der des *nichtlinearen Gaußfilters* vergleichen. Bei der Angabe der Filter geben wir nur die Formel für den inneren Bereich des Bildes an, d.h. wir gehen davon aus, daß alle auftretenden Indizes in unserer Pixelmenge $\mathcal{R}$ liegen. An den Rändern sind alle Formeln geeignet zu modifizieren.

Ein allgemeiner (linearer) Glättungsfilter AGF für unsere Bilddaten f läßt sich schreiben als

$$\mathrm{AGF}(p) = \frac{1}{K} \sum_{q \in \mathcal{R}} m(q-p) \cdot f(q)$$

mit einer sog. Filtermaske m (auch ‚Gewichtsfunktion' genannt) und einer Normierungskonstanten K, die dafür sorgt, daß sich die nicht negativen Koeffizienten der Filtermaske zu Eins summieren.

Die wohl einfachste Operation ist die Bildung arithmetischer Mittelwerte. Der dazu gehörende Filter heißt *Spalttiefpaß*. Die folgende Formel beschreibt die Berechnung des Filterwertes eines linearen Spalttiefpasses mit einer Fenstergröße von $2n+1$ Pixeln an einer Stelle $p \in \mathcal{R}$:

$$\mathrm{STP}_n(p) = \frac{1}{K} \sum_{q \in U_n(p)} f(q) = \frac{1}{K} \sum_{q \in \mathcal{R}} \chi_{n,p}(q) \cdot f(q).$$

Dabei ist die Normierungskonstante $K = \sharp U_n(p) = (2n+1)^2$ und $U_n(p)$ ist die n-Umgebung des Pixels p in der Maximumnorm

$$U_n(p) = \{q \in \mathcal{R} \mid ||p-q||_\infty \leq n\}.$$

Die Funktion $\chi_{n,p}$ ist die zu dieser Umgebung $U_n(p)$ gehörende charakteristische Funktion.

Ein weiterer Glättungsfilter ist der *Binomialtiefpaß*, der sich von dem Spalttiefpaß dadurch unterscheidet, daß nicht alle Grauwerte gleich gewichtet werden. Die Filtermaske besteht hier (bei einer Fenstergröße von $2n+1$) aus einer Binomialverteilung, d.h. aus der Gewichtsfunktion

$$b_n(x,y) = \chi_{n,0}(x,y) \cdot \binom{2n}{n+x} \cdot \binom{2n}{n+y},$$

dabei ist die Normierungskonstante $K = 2^{4n}$.

Der wohl bekannteste, am häufigsten benutzte und in gewisser Weise ausgezeichnete lineare Glättungsfilter (siehe [MH80]) ist der *Gaußtiefpaß*. Bei ihm ist die Filtermaske durch eine Gaußglocke mit Standardabweichung σ

gegeben. Durch die Standardabweichung der Gaußglocke ist implizit die Fenstergröße des Gaußtiefpasses gegeben, da die meisten Implementierungen die kleinen Werte der Gaußfunktion außerhalb einer Umgebung $U_r(0)$ mit $2\sigma \leq r \leq 3\sigma$ durch den Wert 0 ersetzen. Unser Gaußtiefpaß T_σ zur Standardabweichung σ ist durch folgende Formel gegeben:

$$T_\sigma(p) = \frac{1}{K} \sum_{q \in \mathcal{R}} \chi_{3\sigma,p}(q)\, e^{-\frac{||p-q||^2}{2\sigma^2}} \cdot f(q).$$

Dabei ist K wieder eine Normierungskonstante, die dafür sorgt, daß sich die verwendeteten Koeffizienten zu Eins summieren, d.h. die Konstante K ist in aller Regel von dem Wert $2\pi\sigma^2$ des Flächenintegrals über die Gaußfunktion verschieden.

Wir sehen also, daß die vorgestellten linearen Glättungsfilter die Berechnung des Wertes der Gewichtsfunktion zur Gewichtung des Grauwerts $f(q)$ an einer Stelle q im Bild nur von den Stellen p und q abhängig machen. Das führt dazu, daß lineare Filter keinen Unterschied zwischen Stellen innerhalb einer ‚verrauschten' Grauwertfläche und Stellen an Kanten machen. Betrachten wir noch einmal die Abbildung 1.2 auf Seite 10, also das Grauwertprofil der Zeile 375 des Testbilds SCHLUMPF. Wenn wir die Grauwertflächen glätten wollen, ohne die Kanten zu verschmieren, dann ist es nicht von Vorteil, wenn an den Stellen nahe einer Kante (also z.B. für $q = (375, 120)$) die Werte der einen Grauwertfläche (zwischen 50 und 80) mit denen der anderen (255) gemittelt werden. Denn genau diese Mittelung von Werten verschiedener Flächen verschmiert die Kante. Um diese Mittelung von Grauwerten, die zu verschiedenen Grauwertflächen gehören, zu vermeiden, wurde in dem *nichtlinearen Gaußfilter* eine zusätzliche Gewichtung im Wertebereich eingeführt. Das führt dazu, daß die Gewichtsfunktion in der Glättungsformel datenabhängig wird. Die Darstellung des Glättungsfilters $G_{\sigma,\tau}$ (wir verwenden das Wort ‚Filter' auch für nichtlineare Verfahrensvorschriften), angewendet auf ein Bild$f$, ist die folgende:

$$G_{\sigma,\tau} f(p) = \frac{1}{K(f,p)} \sum_{q \in \mathcal{R}} e^{-\frac{||p-q||^2}{2\sigma^2}} \cdot e^{-\frac{|f(p)-f(q)|^2}{2\tau^2}} \cdot f(q).$$

Wir haben in der Formel die Abschneidefunktionen weggelassen, da Sie die Formel nur unübersichtlicher machen. Die Formel ist aber trotzdem so zu verstehen, daß die Gaußfunktionen nicht beliebig weit von der Null weg ausgewertet werden.

Die Normierungskonstante hängt jetzt von den Daten ab, und ist nur noch zur Laufzeit berechenbar. Sie ist gegeben durch

$$K(f,p) = \sum_{q \in \mathcal{R}} e^{-\frac{||p-q||^2}{2\sigma^2}} \cdot e^{-\frac{|f(p)-f(q)|^2}{2\tau^2}}.$$

Der *nichtlineare Gaußfilter* $G_{\sigma,\tau}$ besitzt zwei Parameter σ und τ. Über den Parameter σ stellt man ein, in welcher Umgebung im Ortsbereich die Glättung stattfinden soll und mit dem Parameter τ wird die Umgebung im Wertebereich gewählt. Durch geeignete Wahl der Parameter ist es möglich zu bestimmen, welche Bildstrukturen erhalten bleiben. Durch den Parameter σ gibt man die Größe der Struktur vor und durch den Parameter τ den Kontrast zur Umgebung. Nur die Strukturen, die kleiner und kontrastärmer sind als die Parameterwahl vorschreibt werden geglättet.

Der Abbildung 1.7 auf der nächsten Seite können wir entnehmen, daß ein *nichtlinearer Gaußfilter* die Kanten des Testbilds SCHLUMPF wenig verschmiert, während der lineare *Gaußtiefpaß* beim selben Parameter σ die Kanten stark verschmiert. Hier macht sich also die Einführung einer zusätzlichen Gewichtung im Wertebereich erheblich bemerkbar.

Wir sehen aber auch, daß beim *nichtlinearen Gaußfilter* die Grauwertflächen nicht so glatt sind wie beim linearen *Gaußtiefpaß*. Aus diesen und anderen Gründen ist es sinnvoll den *nichtlinearen Gaußfilter* iteriert anzuwenden, mit jeweils anderer Wahl der Parameter σ und τ. J. WEULE konnte in [Weu94] zeigen, daß eine Iteration des *nichtlinearen Gaußfilters* auch zur Versteilerung von im Original nicht so steilen Kanten beitragen kann.

Bei unserer Vorverarbeitung benutzen wir eine Iteration des *nichtlinearen Gaußfilters*, der ein linearer Gaußtiefpaß vorgeschaltet ist.

$$A_\sigma = G_{8\sigma,4} \circ G_{4\sigma,8} \circ G_{2\sigma,16} \circ G_{\sigma,32} \circ T_\sigma.$$

Wir benötigen diesen ersten Gaußtiefpaß, da der *nichtlineare Gaußfilter* so konstruiert wurde, daß er auch kleine Strukturen erhält, wenn sie nur genügend kontraststark sind. In dieser Filterkette fällt dem ersten Gaußtiefpaß die Aufgabe zu, kontrastreiche kleine Strukturen im Kontrast zu reduzieren. Dabei werden allerdings die Kanten von größeren Strukturen, die wir erhalten wollen, etwas verschmiert. Sie werden nicht allzustark verschmiert, da der erste Gaußtiefpaß nur mit kleinem σ (d.h. $\sigma \leq 1.5$) angewendet wird. Die nachfolgende Iteration des *nichtlinearen Gaußfilters* ist dann in der Lage diese leichte Kantenverschmierung wieder auszugleichen. Im Resultat sind die Kanten von größeren Strukturen erhalten geblieben, aber die im Kontrast reduzierten kleineren Strukturen werden vollständig ‚eingeebnet'. Experimente mit Testbildern haben gezeigt, daß es ausreicht, den Parameter τ fest zu wählen.

Wie sich die Wahl des Parameters σ in den Bildern wiederspiegelt, entnimmt man am besten den Serien von Bildern im Anhang B. Dort wurden die drei folgenden Testbilder mit dem Filter A_σ für verschiedene σ ($0.5 \leq \sigma \leq 3.0$) gefiltert. Die folgenden Abbildungen 1.8 bis 1.13 (Seite 22–24) zeigen die drei Testbilder BABOON (512×512 Pixel), RADIO (256×256

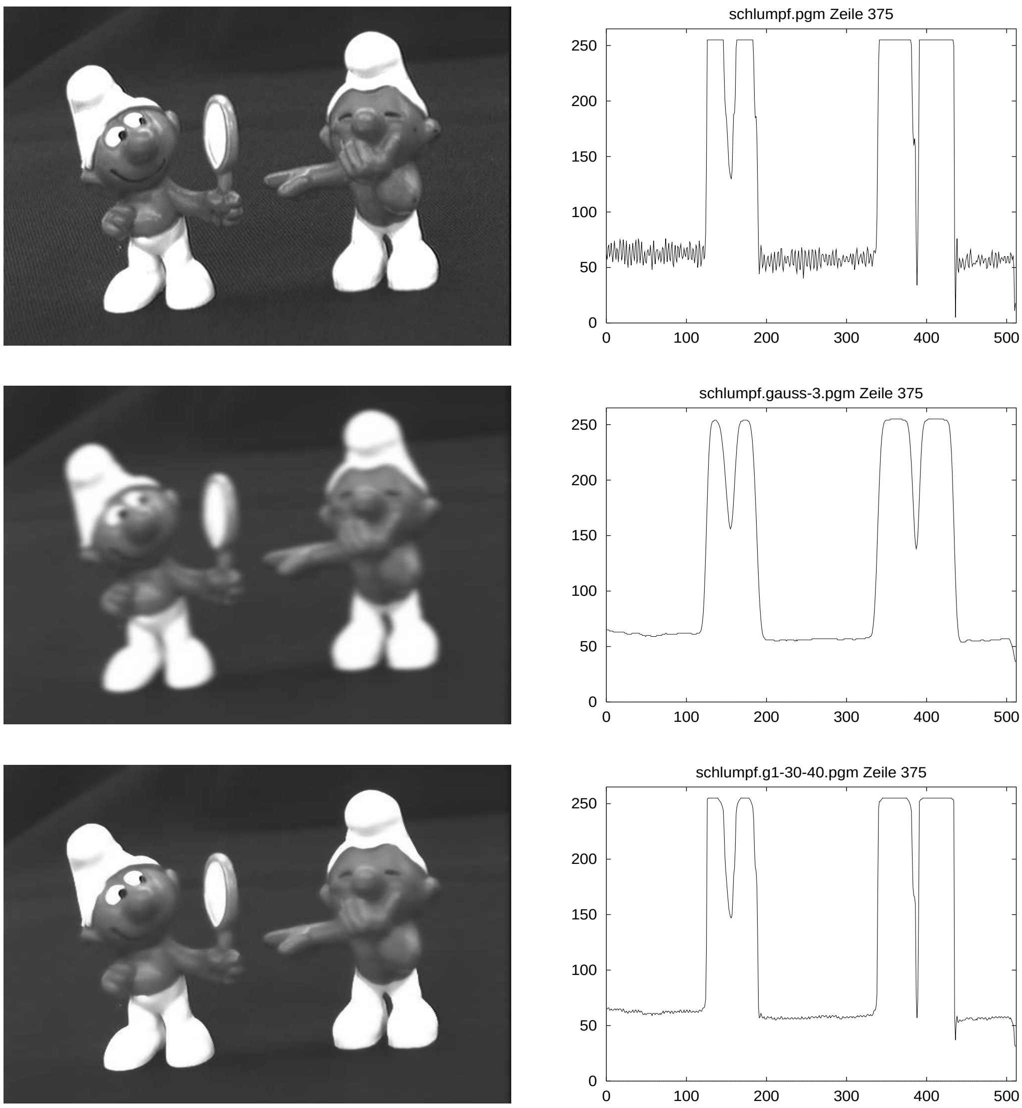


Abbildung 1.7: Testbild Schlumpf (Original oben) gefiltert mit Gaußtiefpaß (mitte, $\sigma = 3$) und mit *nichtlinearem Gaußfilter* (unten, $\sigma = 3$, $\tau = 40$).

Pixel im Seitenverhältnis 3:2) und CASINO (512×512 Pixel) mit den entsprechenden Filterergebnissen des Filters $A_{1.0}$.

Wir haben diese drei Testbilder ausgewählt, da sich an ihnen gut die Wirkung der Filterung demonstrieren läßt. Obwohl alle Bilder mit dem selben Filterparameter $\sigma = 1$ behandelt wurden, würden wir – als menschliche Betrachter – die Qualität des Resultats unterschiedlich bewerten. Die unterschiedliche Bewertung liegt darin begründet, daß wir für die durch die Filterung zerstörte Struktur unterschiedliche Interpretationen besitzen.

Bei dem Testbild BABOON wird durch die Filterung die Feinstruktur im Fell und auf dem Nasenrücken vernichtet. Durch diese Beschreibung ist sofort klar, daß diese Struktur zum eigentlichen Bildinhalt gehört und also Bildinhalt verloren gegangen ist. Beim Testbild RADIO ist dagegen von der Filterung nur das Rauschen betroffen. Ein Bestandteil des Bildes, von dem wir eher glauben, daß er nicht zum eigentlichen Bildinhalt gehört.

Dementsprechend sind wir geneigt in dem Ergebnis der Filterung des Testbilds BABOON eine Bildverschlechterung und in dem Ergebnis der Filterung des Testbilds RADIO eine Bildverbesserung zu sehen. Bei dieser Betrachtung übersehen wir aber, daß wir diese Unterscheidung erst treffen können, wenn wir den Bildinhalt bereits erkannt haben! Um diese Erkennung zu erleichtern ist es aber sinnvoll erst einmal von Feinstruktur zu abstrahieren.

Um einen Eindruck von der Feinstruktur – wie sie ein Algorithmus ‚sieht' – zu bekommen, betrachten wir Abbildung 1.14. Diese zeigt Ausschnitte aus den Testbildern BABOON und RADIO. Auf dieser Vergrößerung ist die Unterscheidung zwischen ‚Fell' und ‚Rauschen' nicht mehr so offensichtlich.

Das dritte Testbild CASINO zeigt noch einen anderen Aspekt der Abstraktion von Feinstruktur. Wir haben im Original (Abbildung 1.12) keine Probleme den Schriftzug ASINO zu lesen (das ‚C' ist von einem Pfeiler verdeckt). Und das, obwohl im Original gar keine Buchstaben vorhanden sind. Es sind nur einzelne Glühbirnen vorhanden, die *von uns* zu Buchstaben gruppiert werden. Etwas analoges leistet unser Filter. In der Abbildung 1.13 sind die Glühbirnen als Einzelobjekte verschwunden. Das was übriggeblieben ist, sind flächige Objekte (Buchstaben), die in dieser Form einer Buchstabenerkennung zugänglich sind.

Als Resultat der Filterung bekommen wir Bilder, deren Grauwerte innerhalb einer Grauwertfläche nur wenig variieren. Dabei ist mit ‚wenig' gemeint, daß sich die Grauwerte wertemäßig nur wenig unterscheiden und die Variation zusätzlich langsam, also mit geringer Frequenz, erfolgt. Solche Grauwertflächen stoßen dann an Kanten zusammen. Dies ist also unsere durch unsere Glättung bestimmte ‚Definition' von *Kante*.

Mit dieser Interpretation der Resultatbilder der Abstraktion im Hinterkopf werden wir in den nächsten Abschnitten dann die Kanten extrahieren

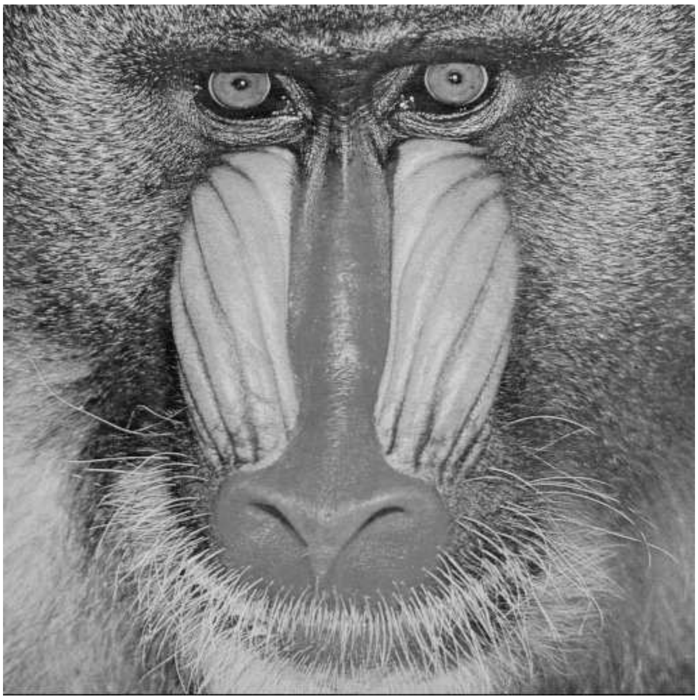


Abbildung 1.8: Original des Testbilds BABOON.

Abbildung 1.9: BABOON gefiltert mit A_σ für $\sigma = 1.0$

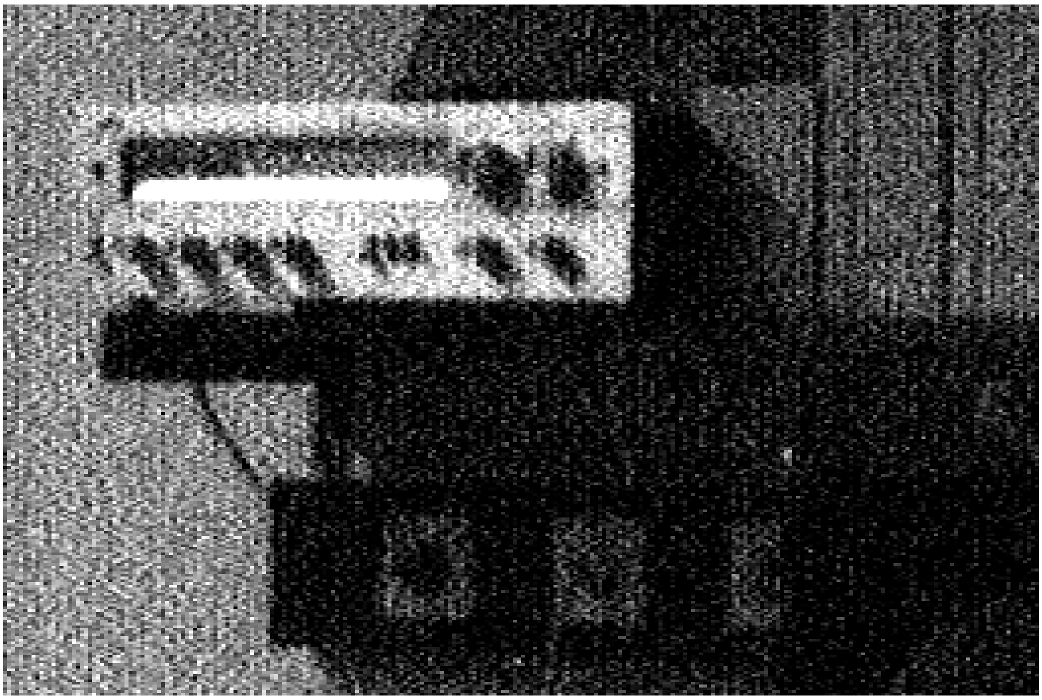


Abbildung 1.10: Original des Testbilds RADIO.

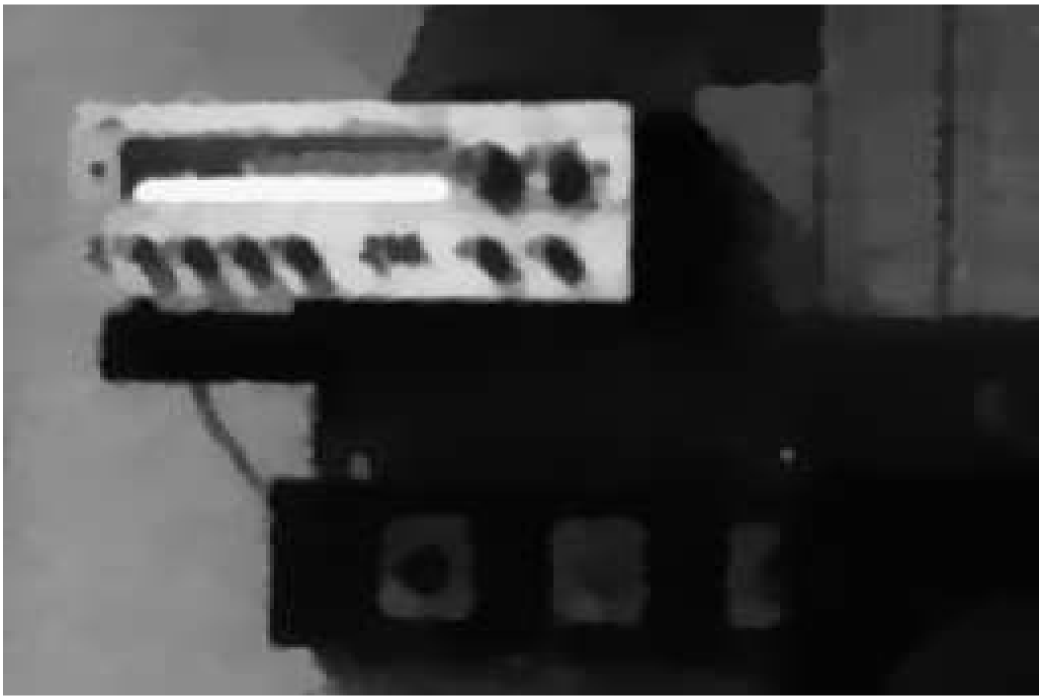


Abbildung 1.11: RADIO gefiltert mit A_σ für $\sigma = 1.0$

Abbildung 1.12: Original des Testbilds CASINO.

Abbildung 1.13: CASINO gefiltert mit A_σ für $\sigma = 1.0$

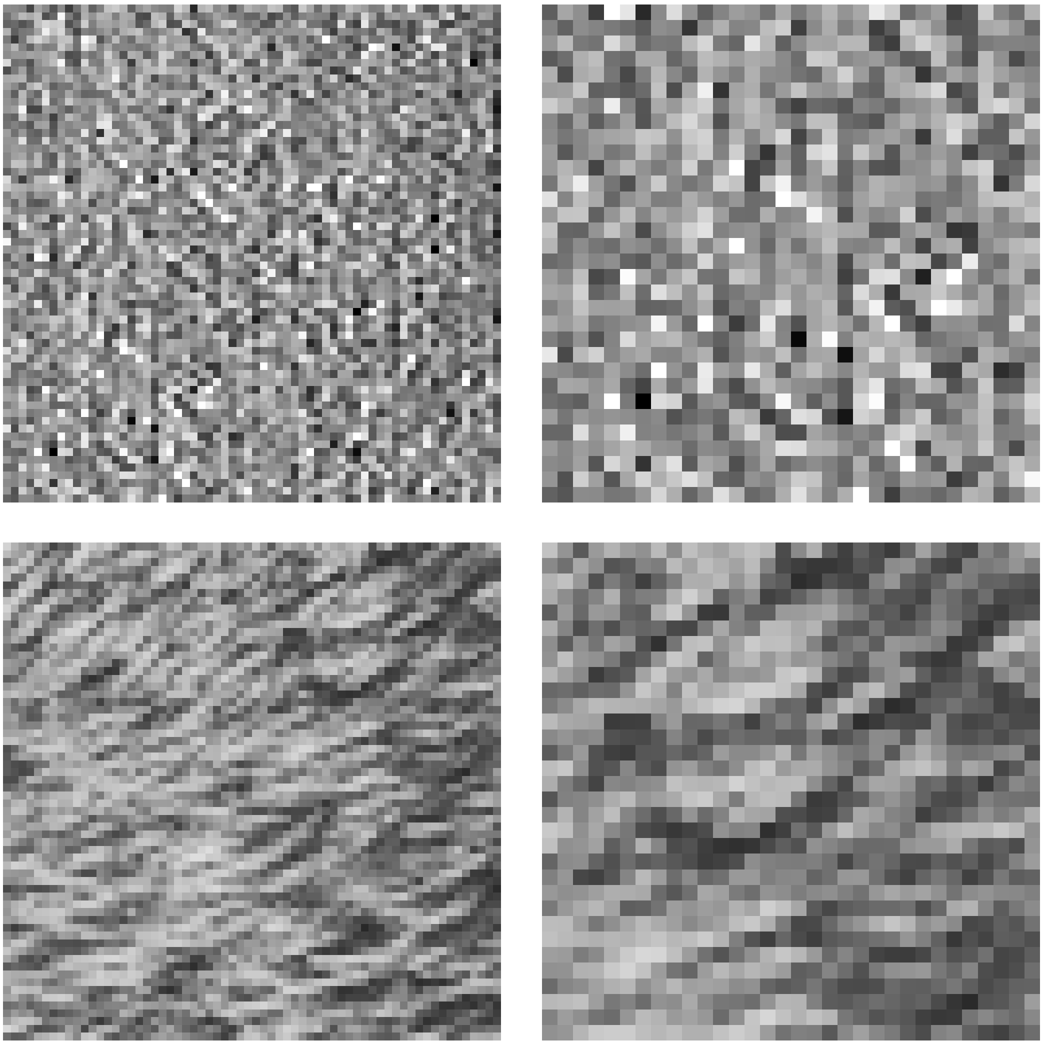


Abbildung 1.14: Zwei 64×64 bzw. 32×32 Pixel große Ausschnitte aus den Testbildern BABOON und RADIO.

und als Kanteninformation abspeichern. Diese repräsentiert die örtlich lokalisierbaren hohen Frequenzanteile. Zwischen diesen Kanten befinden sich dann nur noch Bereiche mit wenig Grauwertfluktuation (nach ‚Definition'). Also können wir diese Grauwerte gut durch wenige Samples (Abtastwerte) repräsentieren. Das wird unsere Grauwertinformation.

Bei der Rekonstruktion des Bildes müssen wir dann nur noch dafür sorgen, daß das Rekonstruktionsverfahren unsere oben gemachte Unterscheidung bezüglich der Grauwerte berücksichtigt. Es darf also nicht Grauwerte aus unterschiedlichen Flächen miteinander verrechnen.

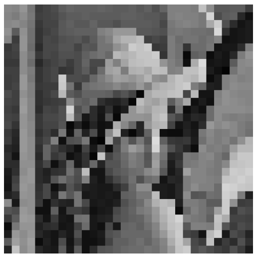


Abbildung 1.15: Markierung der Abtastpunkte im (aufgehellten) Testbild LENA für den Abtastparameter $s = 16$ (links) und die Grauwertinformation (vergrößert, rechts).

1.4 Grauwertinformation

Die Grauwertinformation, die wir aus den abstrahierten Bildern extrahieren, soll nur die ungefähren Helligkeiten der einzelnen Grauwertflächen wiedergeben. Da wir unsere späteren Verarbeitungsschritte (Interpolation aus den Grauwerten, siehe Kapitel 2) so auslegen werden, daß es nicht erforderlich ist, für alle Grauwertflächen auch wirklich einen Grauwert zu kennen, können wir uns hier auf ein sehr einfaches Verfahren, die *Unterabtastung im regulären Raster*, beschränken.

Definition: *Die* Unterabtastung im regulären Raster *zum Parameter* $s \in \mathbb{N}$ *mit* $s \leq \min\{N_x, N_y\}$ *ist eine Abbildung* sub_s*, die einem Bild* $f : \mathcal{R} \to [0, 255]$ *das Bild* $\mathrm{sub}_s f : \mathcal{R}_s \to [0, 255]$ *zuordnet. Dabei ist die Pixelmenge* $\mathcal{R}_s$ *durch*

$$\mathcal{R}_s = [0, \frac{N_x}{s} - 1] \times [0, \frac{N_y}{s} - 1] \cap \mathbb{Z}^2,$$

gegeben und die Grauwerte $\mathrm{sub}_s f(x, y)$ *berechnen sich aus den Werten von* f *durch*

$$\mathrm{sub}_s f(x, y) = f(\left\lfloor \frac{s}{2} \right\rfloor + s \cdot x, \left\lfloor \frac{s}{2} \right\rfloor + s \cdot y)$$

für $0 \leq x < \left\lfloor \frac{N_x}{s} \right\rfloor$ *und* $0 \leq y < \left\lfloor \frac{N_y}{s} \right\rfloor$.

Der linke Teil der Abbildung 1.15 zeigt zu Demonstrationszwecken die Verteilung der Abtastwerte im Original des Testbildes LENA (anstelle des ab-

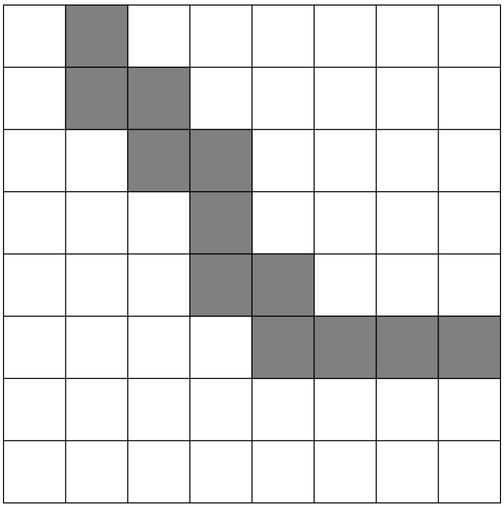


Abbildung 1.16: Kante auf den Pixeln markiert.

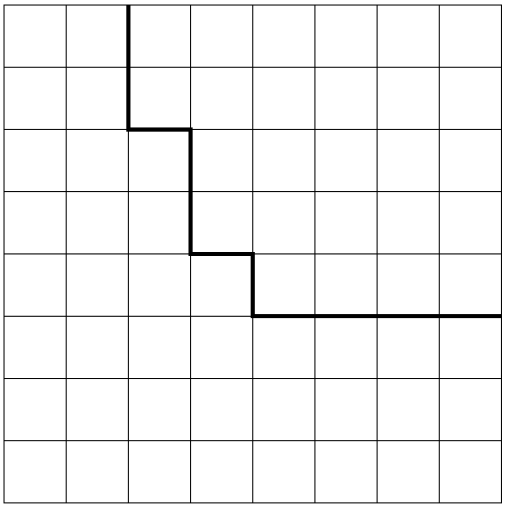


Abbildung 1.17: Kante zwischen den Pixeln markiert.

strahierten Bildes, aus welchem die Grauwertinformation zur weiteren Verarbeitung gewonnen wird) für den Abtastparameter $s = 8$. Zur besseren Sichtbarkeit der Markierungen haben wir das Testbild etwas aufgehellt. Durch die Abtastung haben wir aus dem ursprünglichen 512×512-Bild ein 32×32 Pixel großes Abtastbild bekommen. Dieses ist im rechten Teil der Abbildung 1.15 vergrößert dargestellt.

Dieses einfache Verfahren zur Gewinnung der Grauwertinformation aus den abstrahierten Bildern berücksichtigt keine Eigenschaften des abgetasteten Bildes. Unsere spätere Rekonstruktion, die *kantenerhaltende Grauwertinterpolation* wird aber so beschaffen sein, daß sie nicht auf die spezielle Wahl der Grauwerte als Raster angewiesen ist. So bleibt der Weg offen, durch eine dem Bild angepaßtere Wahl der Grauwertinformation das Rekonstruktionsergebnis weiter zu verbessern.

1.5 Kanteninformation

In diesem Abschnitt beschäftigen wir uns mit der Extraktion der Kanteninformation. Diese Kanteninformation besteht im wesentlichen aus einem speziell kodierten Output eines Kantenfinders. Anders als bei anderen Kantenfindern üblich, wird bei uns ein Pixel p nicht zu einer Kante gehörig, sondern als neben einer Kante liegend markiert. Der Unterschied besteht darin, daß hier also eine Kante zwischen den Pixeln markiert wird und nicht auf den Pixeln. Dadurch ist bei uns eine Kante immer ein ‚linienhaftes' Objekt.

Doch bevor wir uns mit der Gewinnung der Information über die Helligkeitskanten im Bild genauer beschäftigen, wollen wir erst einmal definieren, was wir unter *Kanteninformation* verstehen.

Definition: *Gegeben sei die Pixelmenge $\mathcal{R}$ und gegeben seinen die drei Symbole* `+`, `-` *und* `0`. *Eine Kanteninformation ist eine Abbildung*

$$k\colon \mathcal{R} \to \{\texttt{+}, \texttt{-}, \texttt{0}\}.$$

Um jetzt sagen zu können in welcher Weise eine solche Kanteninformation k Informationen über die Helligkeitskanten im Bild beinhaltet, brauchen wir eine Interpretation dieser Abbildung. Das Symbol `+` bedeutet, daß dieses Pixel auf der helleren Seite einer Kante liegt. Das Symbol `-` bedeutet, daß das entsprechende Pixel auf der dunkleren Seite einer Kante liegt. Alle anderen Pixel tragen das Symbol `0`. Entsprechend dieser Interpretation liegen Kanten dann genau dort, wo beim Übergang von einem Pixel zu einem seiner vier direkten Nachbarn ein Symbolwechsel von `+` nach `-` oder umgekehrt stattfindet.

Definition: *Gegeben sei eine Kanteninformation* $k\colon \mathcal{R} \to \{\texttt{+}, \texttt{-}, \texttt{0}\}$.
Die Menge

$$\{p_x + \frac{1}{2}\} \times [p_y - \frac{1}{2}, p_y + \frac{1}{2}]$$

heißt senkrechtes Kantenstück $\mathcal{K}_s(p)$ *zu einem Pixel* $p = (p_x, p_y) \in \mathcal{R}$ *falls* $0 \leq p_x < N_x - 1$ *und falls die Aussage*

$$\Big(\{k(p_x, p_y) = \texttt{-}\} \wedge \{k(p_x{+}1, p_y) = \texttt{+}\}\Big) \vee \Big(\{k(p_x, p_y) = \texttt{+}\} \wedge \{k(p_x{+}1, p_y) = \texttt{-}\}\Big)$$

wahr ist.
Die Menge

$$[p_x - \frac{1}{2}, p_x + \frac{1}{2}] \times \{p_y + \frac{1}{2}\}$$

heißt waagerechtes Kantenstück $\mathcal{K}_w(p)$ *zu einem Pixel* $p = (p_x, p_y) \in \mathcal{R}$ *falls* $0 \leq p_y < N_y - 1$ *und falls die Aussage*

$$\Big(k(p_x, p_y) = \texttt{-} \;\wedge\; k(p_x, p_y + 1) = \texttt{+}\Big) \vee \Big(k(p_x, p_y) = \texttt{+} \;\wedge\; k(p_x, p_y + 1) = \texttt{-}\Big)$$

wahr ist.
Die Menge aller Kantenstücke $\mathcal{K}(k)$ *einer Kanteninformation* k *ist die Menge*

$$\mathcal{K}(k) = \{\mathcal{K}_s(p), \mathcal{K}_w(p) \mid p \in \mathcal{R}\}.$$

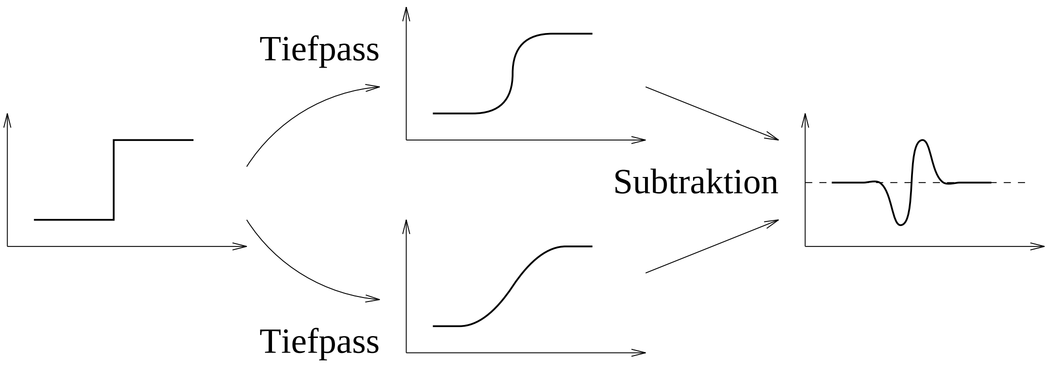


Abbildung 1.18: Prinzipielle Arbeitsweise eines DOG-Filters.

Nachdem nun erklärt ist, wie wir zu einer Kanteninformation k alle Kanten finden, nämlich als Menge aller Kantenstücke $\mathcal{K}(k)$ bleibt nur noch die Aufgabe zu lösen, zu einem gegebenen Bild eine Kanteninformation k so zu bestimmen, daß die Helligkeitskanten im Bild gut durch die Menge aller Kantenstücke repräsentiert wird.

Wir lösen dieses Problem, indem wir einen speziellen Kantenfinder auf unser Bild anwenden und aus dessen Output die Kanteninformation k erzeugen. Wir benutzen dazu eine nichtlineare Modifikation eines einfachen DOG-Filters (Difference of Gaussian), dessen Ausgang dann nach Vorzeichen klassifiziert unsere Symbole `+`, `-` und `0` ergibt.

Bevor wir uns die Modifikation ansehen, betrachten wir erst einmal die prinzipielle Arbeitsweise eines unmodifizierten (linearen) DOG-Filters. Ein DOG-Filter nimmt unser Bild f, filtert es mit zwei verschiedenen Gaußtiefpässen T_{σ_1} und T_{σ_2} und liefert als Ausgang die Differenz: $\mathrm{DOG}(f) = T_{\sigma_1}(f) - T_{\sigma_2}(f)$. Abbildung 1.18 zeigt das Ergebnis der Anwendung eines DOG-Filters auf eine ideale Sprungkante. Wie wir sehen wird die Kante durch einen Nulldurchgang markiert und rechts bzw. links von der Kante hat der DOG-Filterausgang positives bzw. negatives Vorzeichen. Das ist genau der Output, aus welchem wir unsere Kanteninformation erzeugen wollen.

Wenn wir nur mit solchen idealen Kanten arbeiten würden, dann reichte der DOG-Filter zur Erzeugung unserer Kanteninformation vollkommen aus. Ist aber das zu bearbeitende Bild ein wenig verrauscht, so liefert der DOG-Filter nicht nur an Helligkeitskanten solche Nulldurchgänge, sondern auch an anderen Stellen. Abbildung 1.19 zeigt das markierte Ergebnis einer Filterung mit einem DOG-Filter. Die hellen Stellen sind die mit positivem Vorzeichen, die dunklen Stellen die mit negativem und der Wert 0 wird durch die grauen Stellen repräsentiert.

Abbildung 1.19: Markierter Ausgang eines DOG-Filters für das Testbild LENA.

Abbildung 1.20: Markierter Ausgang eines nichtlinear-modifizierten DOG-Filters für das Testbild LENA.

Die Idee des *nichtlinearen DOG-Filters* (NLD-Filter) läßt sich folgendermaßen beschreiben: Wir bezeichnen mit g_σ den zweidimensionalen Gaußkern, dann läßt sich der Gaußtiefpaß T_σ (siehe Seite 18) als diskrete Faltung $T_\sigma(f) = g_\sigma * f$ schreiben. Der DOG-Filter schreibt sich dann

$$\mathrm{DOG}(f) = (g_{\sigma_1} - g_{\sigma_2}) * f = (\delta_0 - g_\tau) * (g_{\sigma_1} * f)$$

mit τ geeignet (d.h. $\tau^{-2} = \sigma_1^{-2} - \sigma_2^{-2}$) und δ_0 die Punktauswertung in 0. Wir sehen also, daß der DOG-Filter aufgefaßt werden kann als Hintereinanderausführung einer Glättung $f \mapsto g_{\sigma_1} * f$ und eines Differenzenfilters $f \mapsto f - g_\tau * f$.

Die nichtlineare Modifikation betrifft den Differenzfilter. Dieser ist der rauschempfindliche Anteil. Der Differenzfilter D_τ ist im linearen Fall durch

$$\begin{aligned} D_\tau f(p) &= f(p) - \frac{1}{K} \sum_{q \in \mathcal{R}} e^{-\frac{||p-q||^2}{2\tau^2}} \cdot f(q) \\ &= \frac{1}{K} \sum_{q \in \mathcal{R}} e^{-\frac{||p-q||^2}{2\tau^2}} \cdot (f(p) - f(q)) \end{aligned}$$

gegeben und wird beim NLD-Filter durch

$$\mathrm{NLD}_{\tau,\lambda} = \frac{1}{K} \sum_{q \in \mathcal{R}} e^{-\frac{||p-q||^2}{2\tau^2}} \cdot \varphi_\lambda(f(p) - f(q))$$

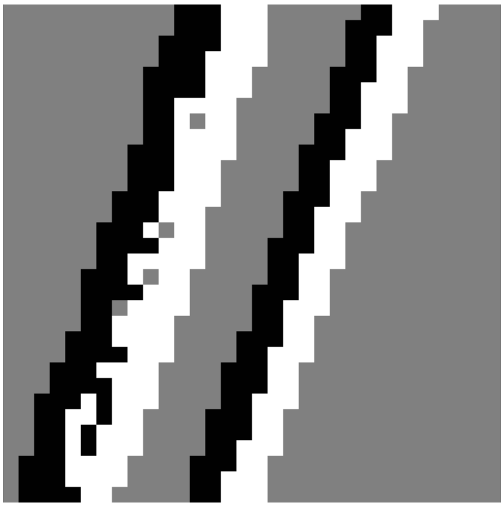


Abbildung 1.21: Markierter Ausgang eines NLD-Filters für das Testbild Lena (Ausschnitt).

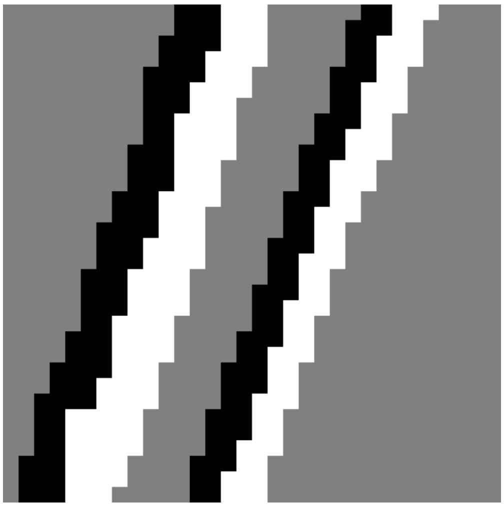


Abbildung 1.22: Ausschnitt aus Abbildung 1.21 nach der ‚Verbesserung' der Kante.

ersetzt. Die Funktion $\varphi_\lambda\colon \mathbb{R} \to \mathbb{R}$ ist eine Abschneidefunktion, die Differenzen betragsmäßig kleiner als λ im Mittelungsprozeß nicht mehr berücksichtigt:

$$\varphi_\lambda(x) = \begin{cases} 0 & \text{für } x \in [-\lambda, \lambda] \\ x & \text{sonst} \end{cases} .$$

Die Kanteninformation K wird aus dem abstrahierten Bild insgesamt auf folgende Weise gewonnen: Das abstrahierte Bild wird mit dem NLD-Filter gefiltert. Der auf diese Weise gewonnene vorzeichenbehaftete Output wird nach Vorzeichen klassifiziert. Den Stellen mit positiven Vorzeichen wird das Symbol `+`, den Stellen mit negativem Vorzeichen das Symbol `-` und allen anderen Stellen das Symbol `0` zugeordnet. Da wir mit natürlichen Bildern arbeiten, in denen die Kanten nicht immer deutlich ausgeprägt sind, liefert diese Zuordnung nicht immer ganz ‚fehlerfreie' Kantenverläufe (siehe Abbildung 1.21). Deswegen wird diese erste Zuordnung der Symbole noch einmal einer Nachbearbeitung unterzogen, in deren Folge der Kantenverlauf auf Pixelebene verbessert wird (Abbildung 1.22).

Kapitel 2

Bildrekonstruktion durch Interpolation

Nachdem wir im letzten Kapitel ein Verfahren vorgestellt haben, mit welchem in einem Bild zwei Komponenten unterschiedlichen Informationscharakters isoliert werden, wollen wir in diesem Kapitel ein Verfahren vorstellen, mit welchem es möglich ist, aus diesen Komponenten wieder ein Bild zurückzugewinnen. Dieses Verfahren ist u.a. aus folgenden Gründen interessant:

Zum ersten belegen die Ergebnisse der Rekonstruktion mit diesem Verfahren, daß die isolierten Komponenten eines Grauwertbildes (wie im letzten Kapitel beschrieben), einen wesentlichen Teil der Bildinformation ausmachen. Die Vorgehensweise, Bilder auf einer gewissen Größenskala zu betrachten und alle Bilddetails unterhalb einer gewissen Größenordnung zu ignorieren, wird erst dadurch gerechtfertigt, daß es mit der nachfolgend zu schildernden Interpolationsmethode möglich ist, diese ‚abstrahierten Bilder' wieder zu visualisieren.

Zum zweiten ist die Interpolationsmethode als solche interessant, da sie einen Weg aufzeigt, wie neben reinen Abtastdaten eines Signals, hier den Grauwerten des Bildsignals, weitere ‚Strukturdaten' Eingang in die Interpolation finden. In unserem Fall sind diese ‚Strukturdaten' die Kanteninformation, welche aus dem Bild extrahiert wurde. Diese Kanteninformation ist unsere Repräsentation von Helligkeitskanten im Bild, also eine Repräsentation von Unstetigkeitsstellen in der als kontinuierlich gedachten Helligkeitsfunktion. Diese Unstetigkeitsstellen gilt es bei der Interpolation zu berücksichtigen. Der hier eingeschlagene Weg könnte über diese spezielle Anwendung im Bereich *Maschinelles Sehen* hinaus von Bedeutung sein. Er zeigt, daß es in bestimmten Situationen nötig und möglich ist, Signale mit bekannten Unstetigkeitsstellen zu produzieren. Es ist sicher auch in anderen Bereichen des

Maschinellen Sehens und darüber hinaus sinnvoll, Vorwissen über die Struktur der behandelten Signale in die (Re-) Konstruktion einzubeziehen.

Das hier vorgestellte Rekonstruktionsverfahren ist in einem übertragenen Sinn eine ‚Verallgemeinerung' des Abtasttheorems von SHANNON. Das Abtasttheorem erlaubt die Rekonstruktion (exakt) eines bandbreitenbeschränkten Signal aus seinen Abtastwerten, wenn diese dicht genug liegen. Unser Verfahren erlaubt die Rekonstruktion eines nicht bandbreitenbeschränkten Bildes (nicht exakt, aber im ‚wesentlichen') aus Abtastwerten und Kanteninformationen. Dabei repräsentiert die Kanteninformation die örtlich lokalisierten Anteile der hohen Frequenzen im Fourierspektrum, während die Abtastwerte für die Rekonstruktion der tieffrequenten Regionen zwischen den Kanten des Bildes zuständig sind. Wir sind leider noch nicht in der Lage, in diesem Bereich genauere mathematische Aussagen zu machen, da noch nicht einmal klar ist, wie die entsprechenden Funktionenräume zur Modellierung der Bilder gewählt werden sollen.

2.1 Globale Shepard-Interpolation

Bevor wie uns dem Problem zuwenden, die Kanteninformation bei der Interpolation zu berücksichtigen, möchten wir uns allgemeiner mit der Interpolation von spärlichen (wohlgemerkt nicht nur Bild-) Daten beschäftigen.

Die allgemeine Aufgabe der Interpolation bzw. Approximation von spärlichen oder verstreuten Daten (Interpolation of scattered data) wird z.B. von L.L. SCHUMAKER in seiner Einleitung wie folgt beschrieben:

> Problem 1.1. Let D be a domain in the (x, y)-plane, and suppose F is a real-valued function defined on D. Suppose we are given the values $F_i = F(x_i, y_i)$ of F at some set of points (x_i, y_i) located in D, $i = 1, 2, \ldots, N$. Find a function f defined on D which reasonably approximates F. [...]
>
> There are basically two approaches to handling Problem 1.1. First, we may try to construct a function f which interpolates the data exactly; i.e., such that $f(x_i, y_i) = F_i$, $i = 1, 2, \ldots, N$. This approach may be desirable when the function values at the data points are known to high precision and where it is highly desirable that these values be preserved by the approximating function.
>
> The second approach involves constructing f which only approximately fits the data. This may be regarded as data smoothing and will be desirable when (as is often the case) the data are subject to inaccurate measurement or even errors. [Schu76]

Bei den meisten Methoden, die benutzt werden, um solche Funktionen F zu finden, ist die Funktion F konstruktionsbedingt stetig. R. Franke hat sich in seinem Methodenvergleich zur Konstruktion solcher Funktionen F auf C^1-Funktionen beschränkt.

> The basic problem which is being addressed here is evaluation of methods for obtaining a smooth (at least continuous first partial derivatives) bivariate function, $F(x, y)$, which takes on certain prescribed values, $F(x_k, y_k) = f_k$, $k = 1, \ldots, N$. The points (x_k, y_k) are not assumed to satisfy any particular conditions as to spacing or density, hence the term "scattered." It is usually convenient to think of the values f_k as arising from some underlying (not necessarily known) function $f(x, y)$, so that $f_k = f(x_k, y_k)$, $k = 1, \ldots, N$. [Fr82]

Die Methoden, die Franke vergleicht, lassen sich grob in zwei Gruppen klassifizieren. Die eine Gruppe von Methoden parzelliert den Definitionsbereich D und die andere nicht. Dabei ist mit ‚Parzellierung' jede einer Vorschrift gehorchende Aufteilung des Definitionsbereiches D in Teile D_i $i = 1, 2, \ldots, l$ gemeint, deren Vereinigung D überdeckt. Beispiele sind Parkettierungen, überlappende und nicht überlappende Rechteckzerlegungen, Triangulierungen u.ä.

Parzellierungsmethoden haben für unsere spätere Anwendung die folgenden Nachteile:

1. Es gibt meist keine eindeutige Wahl einer Parzellierung und damit wird die Interpolationsfunktion F abhängig von der Wahl einer Parzellierung.

2. Es ist häufig schwierig eine ‚gute', d.h. auf die Anwendung passende Parzellierung zu finden.

Für eine tiefere Erörterung dieser Problematik siehe z.B. Barnhill [Ba77].

An dem folgenden Beispiel (XOR-Problem) wollen wir die Problematik der Parzellierungsmethoden demonstrieren und die von uns im weiteren benutzten distanzgewichteten Interpolationsmethoden motivieren. Dieses Beispiel wird aus historischen Gründen gerne zur Untersuchung von Klassifikationsaufgaben bei neuronalen Netzen benutzt [BrLo88, MoDa88, We91]. Das XOR-Problem ist für den Bereich *Neuronale Netze* deswegen interessant, da es die einfachste Abbildung darstellt, welches ein einstufiges Perzeptron-Netzwerk nicht mehr ‚lernen' kann (vergleiche A. Zell [Ze94, Seite 99f]). Wir werden an diesem Beispiel sehen, welchen Einfluß auf das Ergebnis der Interpolation die unterschiedliche Wahl einer Parzellierung haben kann.

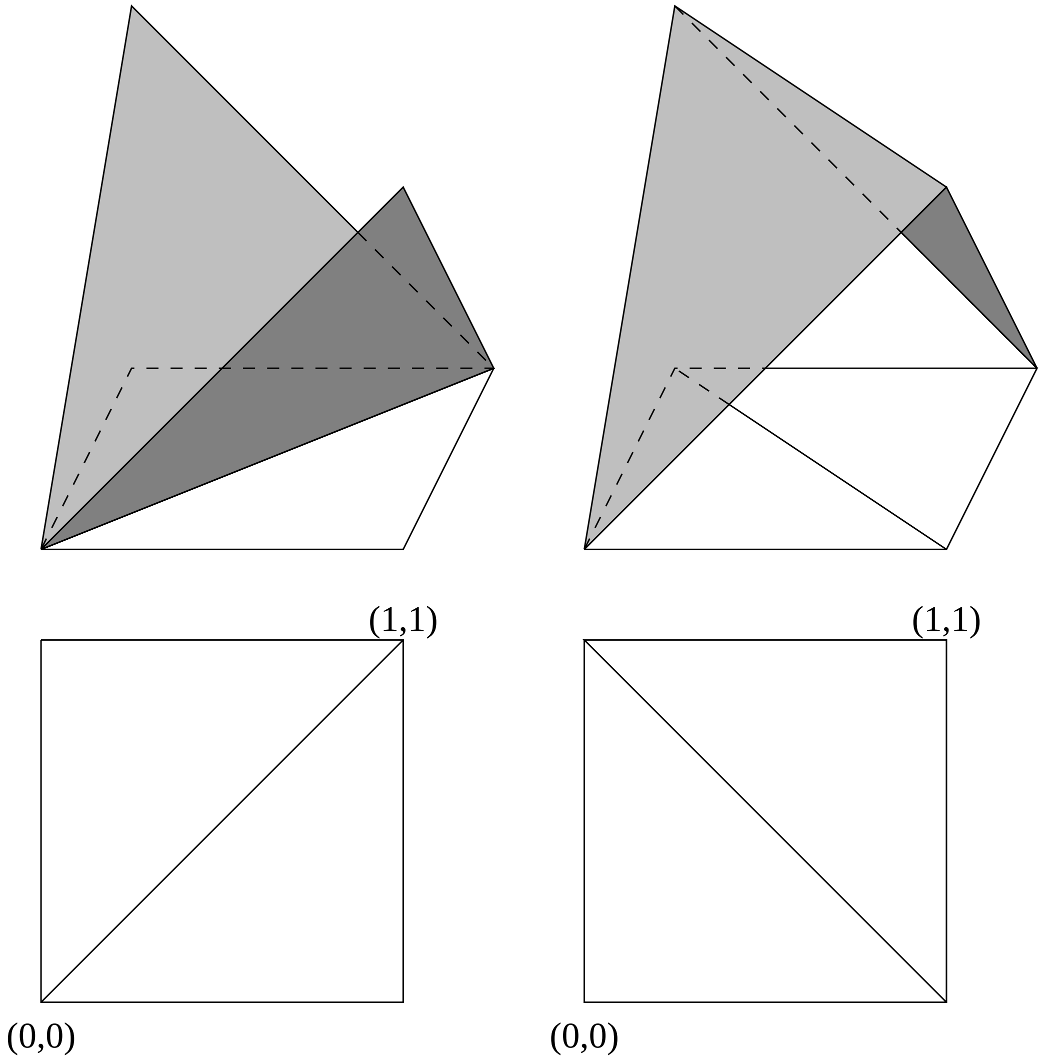


Abbildung 2.1: Unterschiedliche Triangulierungen führen zu unterschiedlichen Funktionen beim XOR-Problem.

Beispiel (XOR-Problem): *Gegeben ist das Einheitsquadrat* $D = [0,1]^2$ *und gesucht wird eine Funktion* $F\colon D \to \mathbb{R}$, *welche die vorgegebenen Werte* $F(0,0) = F(1,1) = 0$ *und* $F(1,0) = F(0,1) = 1$ *annimmt.*

Wir konstruieren diese Funktion, indem wir folgendes Verfahren vorgeben: Wir wählen eine Triangulierung von D und bekommen die Funktion durch baryzentrische Interpolation über jedem einzelnen Dreieck der Triangulierung. (Dieses Verfahren wird häufig in der Computergraphik angewendet.)

Wir wollen kurz erläutern, wie die baryzentrische Interpolation funktioniert. Diese Interpolationsmethode beruht auf der Tatsache, daß drei nicht kolineare Punkte p_0, p_1, p_2 im $\mathbb{R}^3$ genau eine affine Ebene definieren. Für einen Punkt $p \in \Delta(p_0p_1p_2)$ des Dreiecks $\Delta(p_0p_1p_2) \subset \mathbb{R}^2$ ist dann $F(p) := \lambda_0 f_0 + \lambda_1 f_1 + \lambda_2 f_2$. Dabei sind f_i die vorgegebenen Werte der Punkte p_i und λ_i sind die Koordinaten der Konvexkombination $p = \lambda_0 p_0 + \lambda_1 p_1 + \lambda_2 p_2$. Auf diese Weise erhält man zu einer Triangulierung des Definitionsbereichs D eine stetige Funktion F, die aus lauter Ebenenstücken zusammengesetzt ist.

Je nachdem für welche Triangulierung wir uns entscheiden, erhalten wir unterschiedliche Funktionen. Abbildung 2.1 auf der vorherigen Seite zeigt in der unteren Hälfte zwei verschiedene Triangulierungen, die zu den vorgegebenen Punkten aus D korrespondieren. In der oberen Hälfte zeigt sie die zu den Triangulierungen gehörenden Funktionen, wenn sie nach dem Verfahren der baryzentrischen Interpolation konstruiert werden.

Wir haben also gesehen, daß bei vorgegebener Konstruktionsmethode unterschiedliche Funktionen konstruiert werden können, abhängig davon, welche Triangulierung gewählt wird. Wir haben auch gesehen, daß bei der Berechnung eines Interpolationswertes $F(p)$ an einer Stelle $p \in [0,1]^2$ immer nur drei der vier vorgegebenen Werte eingehen. Das bringt uns auf die Idee bei der Interpolation an einer Stelle $p \in [0,1]^2$ immer alle vier vorgegebenen Werte zu berücksichtigen und den ‚Einfluß' eines Wertes von der Entfernung zwischen dem Punkt p und der entsprechenden Ecke des Einheitsquadrats abhängig zu machen.

Wir wollen also den Ansatz

$$F(p) = \sum_{i=1}^{N} \frac{1}{||p - p_i||^\mu} \cdot f_i$$

versuchen. Dabei sind die Punkte $p_i \in \mathbb{R}^2$ mit ihren Werten $f_i \in \mathbb{R}$ $(i = 1, 2, \ldots, N)$ vorgegeben. Die auftretende Norm ist die euklidische und μ eine positive reelle Zahl.

Diese Formel hat noch den Nachteil, daß sie anstatt die vorgegebenen Werte zu interpolieren, an diesen Stellen Pole produziert. Dieses Problem wird durch ‚Normierung' gelöst. Insgesamt erhalten wir die folgende Interpolationformel, die von D. Shepard in [Sh68] vorgestellt wurde.

Definition: *Gegeben seien paarweise verschiedene Punkte p_i und reelle Werte f_i; $(p_i, f_i) \in \mathbb{R}^2 \times \mathbb{R}$ für $i = 1, 2, \ldots, N$. Die Shepard-Interpolation zum Parameter $\mu > 0$ ist gegeben durch $S^{(\mu)} \colon \mathbb{R}^2 \to \mathbb{R}$*

$$S^{(\mu)}(p) = \begin{cases} f_i & \textit{für } p = p_i \\ \left(\sum_{i=1}^{N} \frac{f_i}{||p - p_i||^\mu} \right) \Big/ \left(\sum_{i=1}^{N} \frac{1}{||p - p_i||^\mu} \right) & \textit{sonst.} \end{cases} \tag{2.1}$$

W. GORDON und J. WIXOM schreiben über diese Formel:

> In spite of our failure to uncover any references other than [4] (bei uns [Sh68], U.D.), the basic simplicity and applicability of these metric interpolation schemes leads one to suspect more antique origins. [GoWi78]

Auch uns ist es nicht gelungen frühere Referenzen als [Sh68] zu finden. Es gibt aber Arbeiten auf anderen, weiter entfernten Gebieten, in welchen die Autoren aus gänzlich anderen Überlegungen heraus, auf die Shepard-Interpolationsformel geführt werden. Siehe dazu [MoDa89] und [Wo90].

Wir wollen diese Interpolationsformel zunächst für verschieden Parameter $\mu > 0$ auf unser XOR-Problem anwenden. Die folgenden Abbildungen (Abb. 2.2 bis Abb. 2.6 auf Seite 39–43) zeigen Graphen der durch verschiedene Parameterwahl erzeugten Interpolationsfunktionen. Alle Abbildungen sind zweigeteilt. Der obere Teil zeigt den Graphen in einer leichten Aufsicht und der Betrachter schaut genau auf den Ursprung des Koordinatensystems. Der untere Teil einer jeden Abbildung zeigt den jeweils gleichen Graphen nochmals in einer stärkeren Aufsicht und zugleich so, daß der Betrachter den Schnitt des Graphen mit der x-z-Achse gut erkennen kann. Außerdem zeigt der zweite Teil eine Konturkarte des Graphen. Eingezeichnet sind die Höhenlinien zu $z = j/11$ für $j = 1, \ldots, 10$. (In den Abbildungsbeschriftungen wurde der Parameter μ als dritte Variable von S notiert.)

Wie zu erwarten war, ist das qualitative Verhalten von $S^{(\mu)}$ für verschiedene Wahlen von $\mu > 0$ unterschiedlich. Die ersten drei Abbildungen (Abb. 2.2 bis 2.4) zeigen die Graphen zu ‚gemäßigten‘ Wahlen von μ. Betrachten wir die Abbildung 2.2 im Vergleich zu den beiden anderen Abbildungen 2.3 und 2.4, so stellt der Parameter $\mu = 2$ eine ‚ausgewogene‘ Wahl dar. Dabei ist ‚ausgewogen‘ in folgendem Sinn gemeint: Die Graphen zu den Werten $\mu \leq 1$ und $\mu \geq 3$ haben die Tendenz zur Plateaubildung. Für $\mu \leq 1$ bildet sich ein Plateau über dem Mittelpunkt des Einheitsquadrats und für $\mu \geq 3$ bilden sich vier Plateaus über den Ecken. Dieses läßt sich auch gut an den Konturkarten der Abbildungen erkennen. Die Abbildungen 2.5 auf Seite 42 und 2.6 auf Seite 43 dienen nur dazu, diese Tendenz deutlich vor Augen zu führen.

Verschiedene Experimente haben gezeigt, daß für unsere spätere Anwendung ein Wert von $\mu = 2$ eine gute Wahl ist. Dieser Wert ist gut in bezug auf die optische Qualität der interpolierten Bilder, und gut in bezug auf das Laufzeitverhalten der Anwendung (denn aus Quadraten läßt sich gut die Wurzel ziehen).

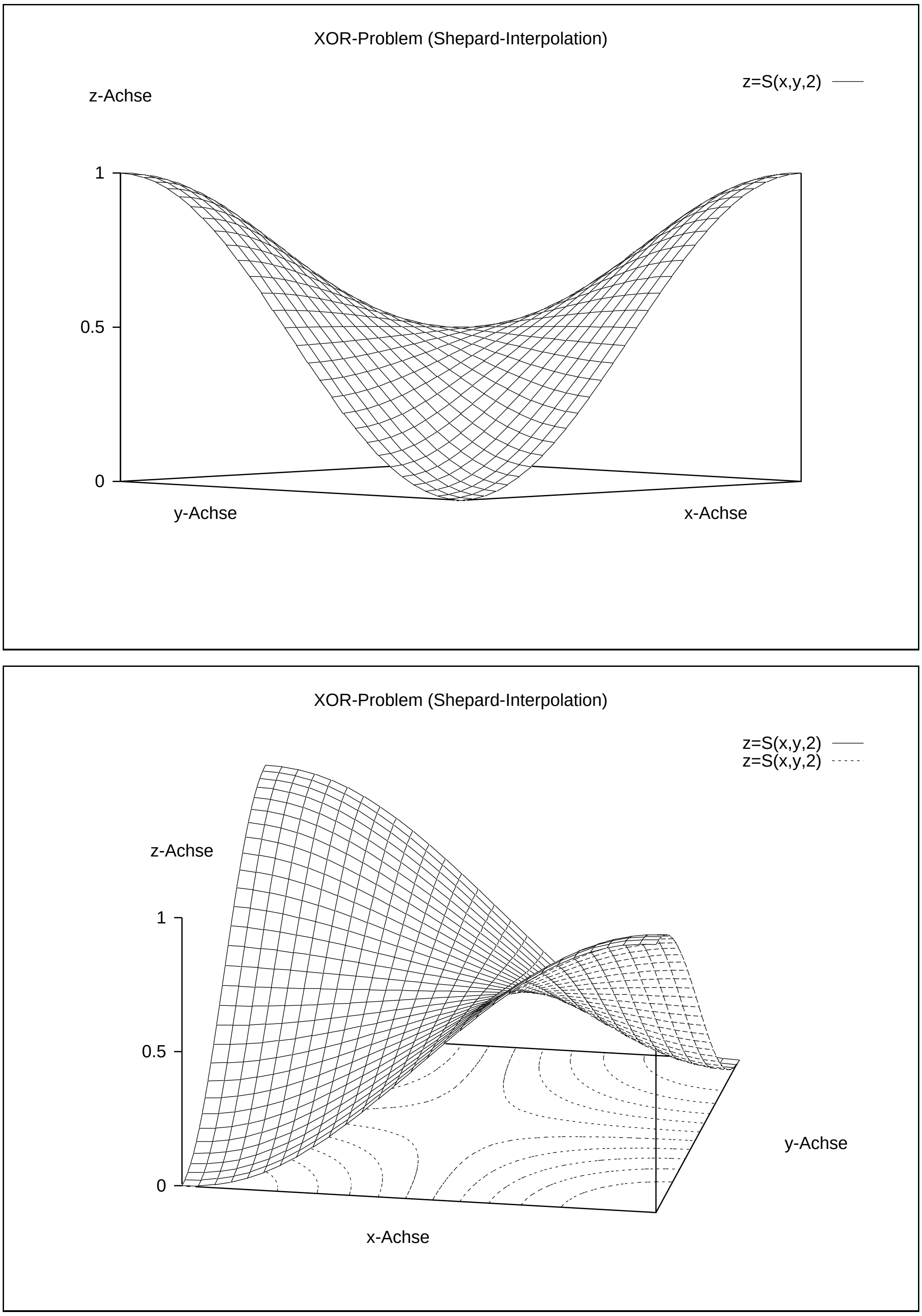
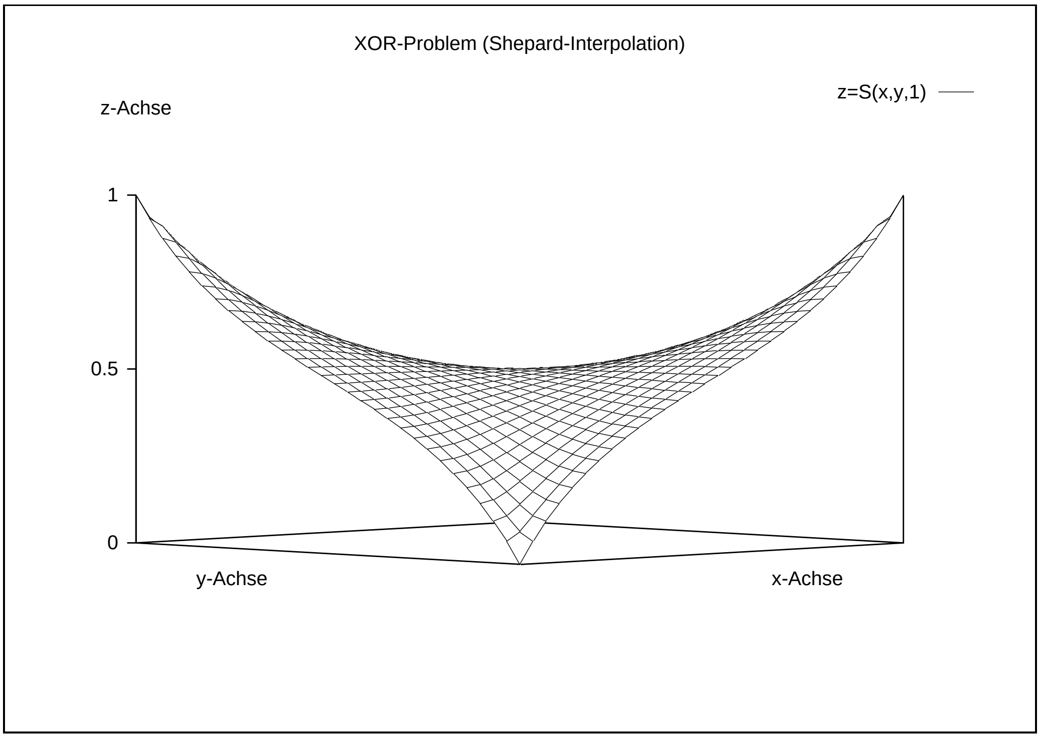


Abbildung 2.2: Shepard-Interpolation zum Parameter $\mu = 2$



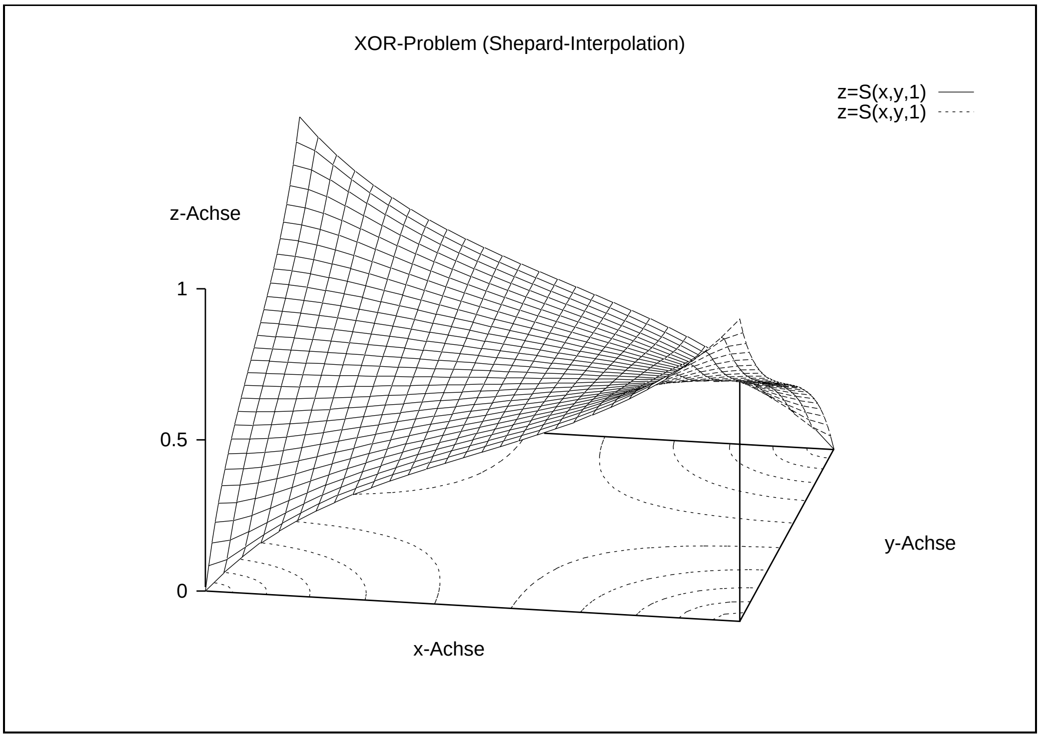
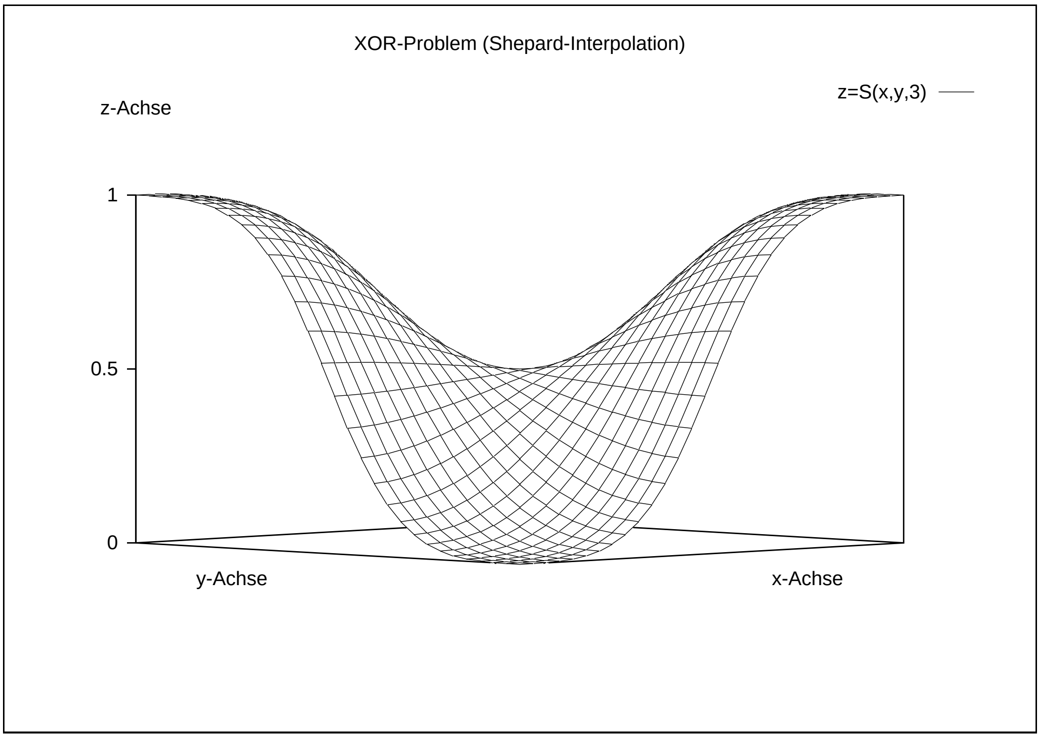


Abbildung 2.3: Shepard-Interpolation zum Parameter $\mu = 1$



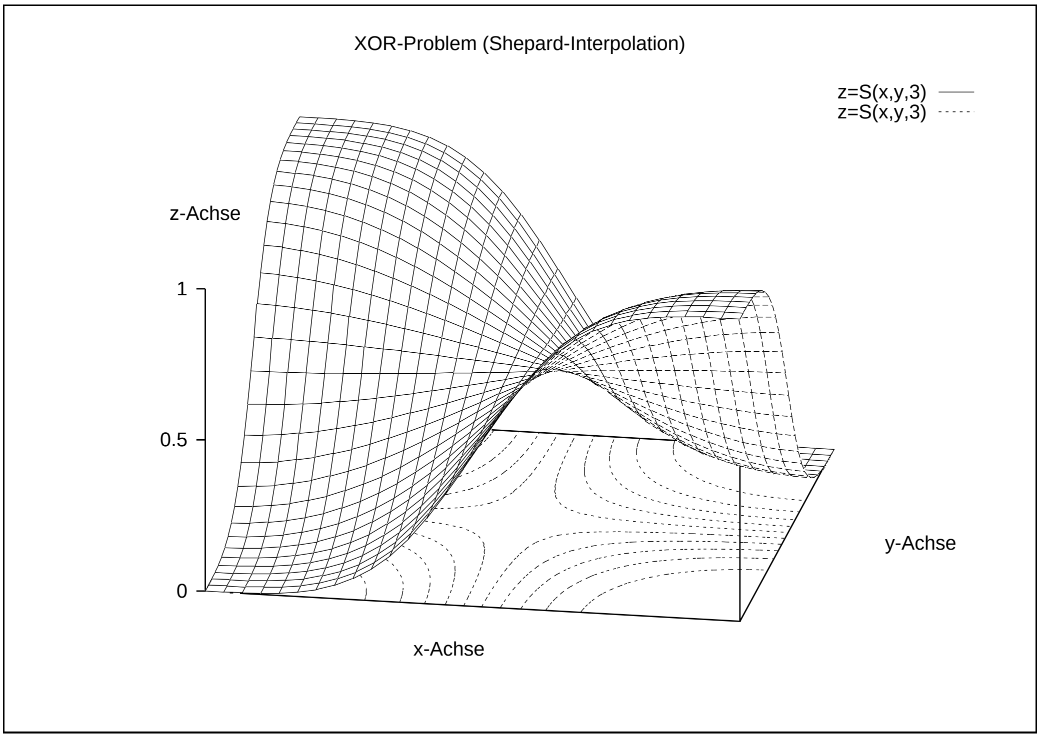
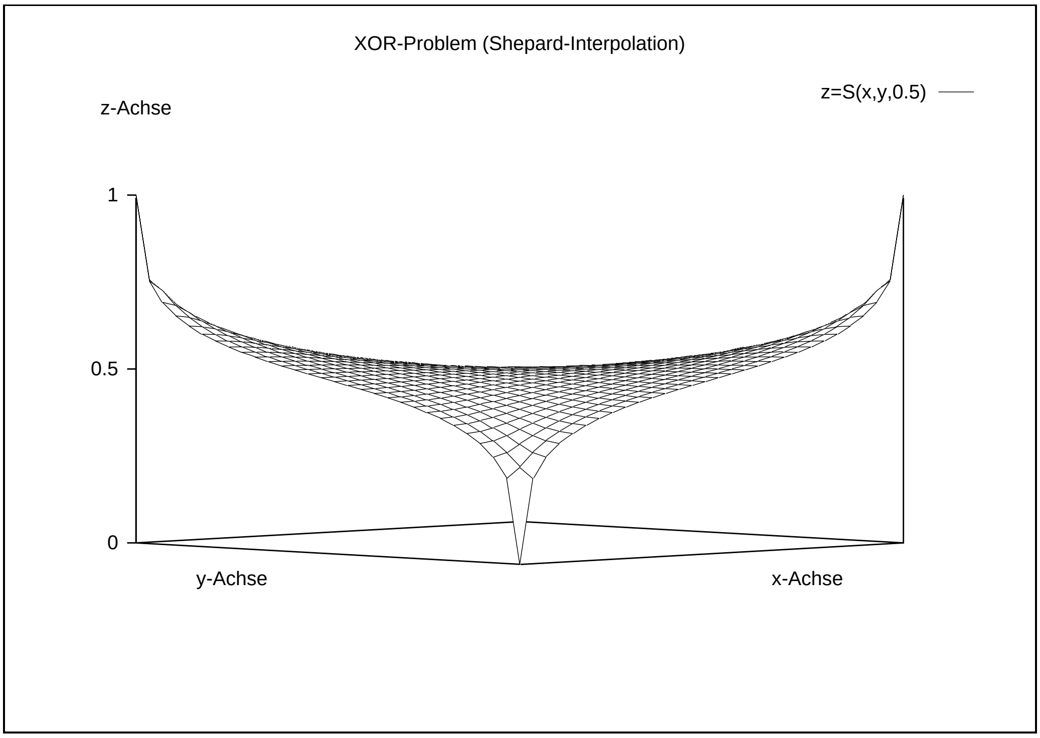


Abbildung 2.4: Shepard-Interpolation zum Parameter $\mu = 3$



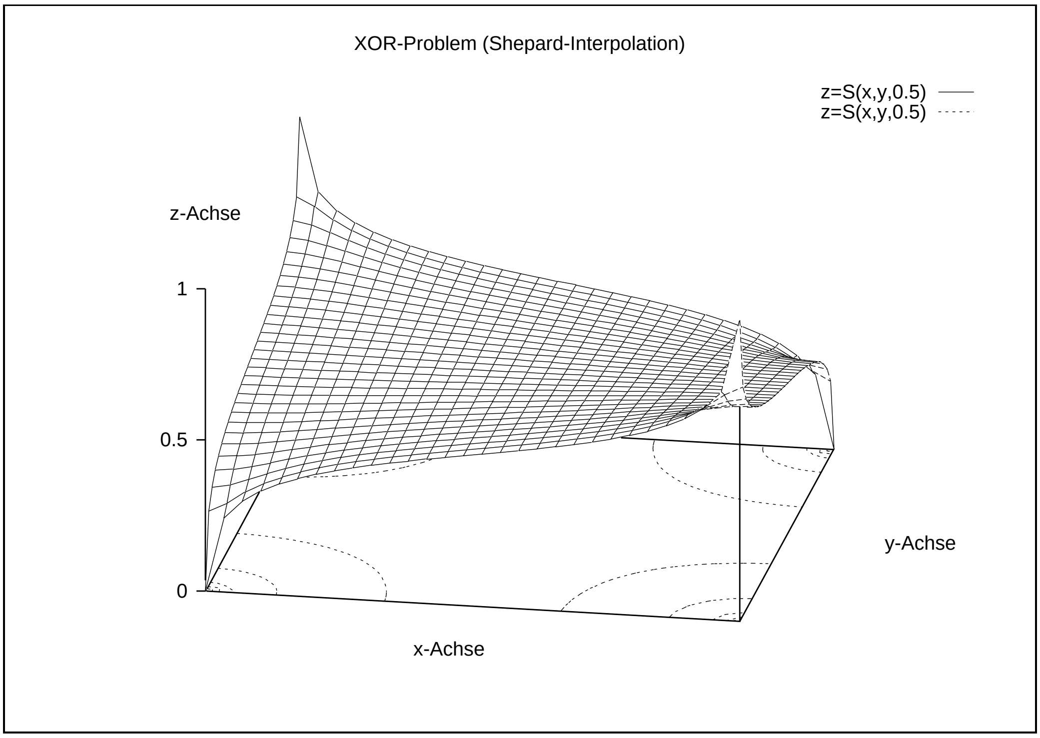
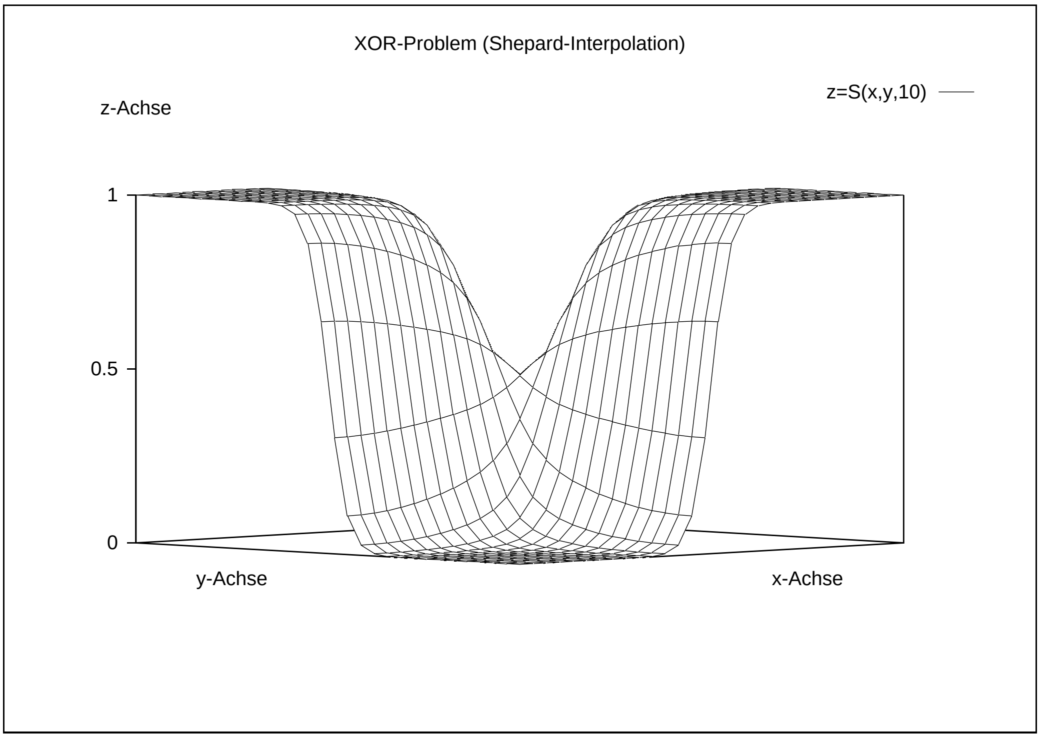


Abbildung 2.5: Shepard-Interpolation zum Parameter $\mu = 0.5$



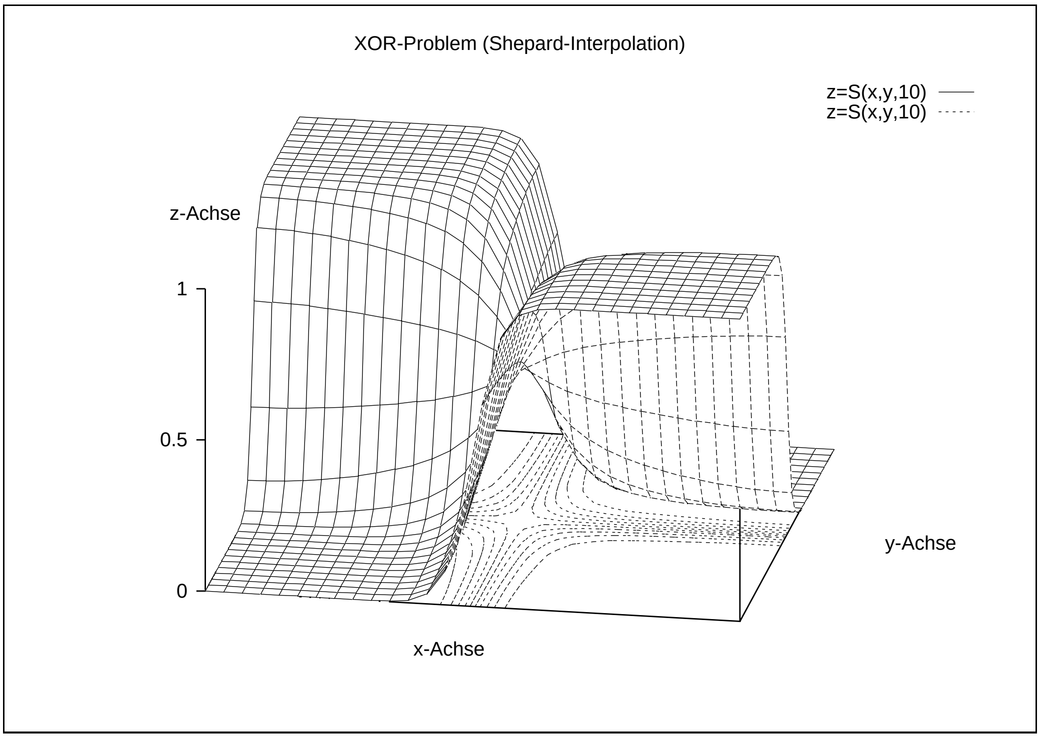


Abbildung 2.6: Shepard-Interpolation zum Parameter $\mu = 10$

Doch bevor wir uns unserer speziellen Anwendung zuwenden, wollen wir einige der über die Shepard-Interpolation bekannten Tatsachen zusammenstellen.

Satz: *Gegeben seien paarweise verschiedene Punkte $p_i \in \mathbb{R}^2$ mit skalaren Werten $f_i \in \mathbb{R}$ für $i = 1, 2, \ldots, N$ und ein Skalar $\mu > 0$. Dann gilt für die Shepard-Interpolation $S^{(\mu)} \colon \mathbb{R}^2 \to \mathbb{R}$ zu diesen Daten:*

1. *$S^{(\mu)}$ ist stetig und besitzt die ‚cardinal form'*

$$S^{(\mu)}(p) = \sum_{i=1}^{N} w_i^{(\mu)}(p) \cdot f_i$$

   mit den Gewichtsfunktionen

$$w_i^{(\mu)}(p) = \Big(\prod_{\substack{j=1 \\ j \neq i}}^{N} ||p - p_j||^{\mu} \Big) \Big/ \Big(\sum_{k=1}^{N} \prod_{\substack{j=1 \\ j \neq k}}^{N} ||p - p_j||^{\mu} \Big).$$

2. *Es gelten die folgenden Differenzierbarkeitsaussagen in Abhängigkeit von μ:*
   (a) $S^{(\mu)} \in C^{\infty}$ *falls* $\mu \in 2\mathbb{Z}$
   (b) $S^{(\mu)} \in C^{\mu-1}$ *falls* $\mu \in 2\mathbb{Z} + 1$
   (c) $S^{(\mu)} \in C^{\lfloor \mu \rfloor}$ *falls* $\mu \notin \mathbb{Z}$

3. *$S^{(\mu)}$ interpoliert die vorgegebenen Werte, $S^{(\mu)}(p_i) = f_i$, $i = 1, 2, \ldots, N$ und weiter gilt für die Werte von $S^{(\mu)}$:*
   (a) $\min_{i=1,\ldots,N} f_i \leq S^{(\mu)}(p) \leq \max_{i=1,\ldots,N} f_i \quad \forall p \in \mathbb{R}^2$
   (b) *Sei $(q_j)_{j \in \mathbb{N}} \subset \mathbb{R}^2$ eine Folge mit $\lim_{j \to \infty} ||q_j|| = \infty$, dann gilt*

$$\lim_{j \to \infty} S^{(\mu)}(q_j) = \frac{1}{N} \sum_{i=1}^{N} f_i$$

4. *Für $\mu > 1$ gilt: $\dfrac{\partial}{\partial x} S^{(\mu)}(p_i) = \dfrac{\partial}{\partial y} S^{(\mu)}(p_i) = 0$.*

Beweise dieser Aussagen sind zu finden in [GoWi78], [BDL83] und [Sh68].

Ausgehend von der Distanzgewichtung bei der ursprünglichen Shepard-Interpolation wurden in der Vergangenheit unterschiedliche Modifikationen und Erweiterungen der Shepard-Interpolation untersucht. Viele der Modifikationen waren motiviert durch offensichtliche Nachteile dieser Interpolationsmethode.

Einer dieser Nachteile ist die Nichtlokalität der Interpolation. In die Berechnung der Gewichtsfunktionen $w_i^{(\mu)}$ gehen bei jeder Gewichtsfunktion immer alle Stützstellen p_i ein. Um dem dadurch bedingten hohen Rechenaufwand – insbesondere bei großen Datenmengen – zu begegnen, wurden verschiedene Ansätze zur Lokalisierung der Shepard-Interpolation untersucht, siehe z.B. [Ba77, S. 112], [Fr82] und [Re88a].

Die von R. Renka in [Re88a] vorgestellte *Modified Quadratic Shepard*-Methode, sowie deren Fortran77 Implementierung [Re88b], werden noch Gegenstand genauerer Untersuchung sein.

Der ‚hohe Rechenaufwand', der früher als gravierender Nachteil der globalen Shepard-Interpolation hervorgehoben wurde, sollte heutzutage weniger kritisch – aber nicht unkritisch – gesehen werden, denn bei der Rechenleistung heutiger Digitalrechner kann man auch Interpolationen mit größeren Datenmengen in akzeptabler Zeit berechnen.

Dazu eine kurze Überschlagsrechnung: Wollte man die (numerisch nicht besonders günstige) Formel

$$S^{(2)}(x,y) = \left(\sum_{i=1}^{N} \frac{1}{(x-x_i)^2+(y-y_i)^2} \cdot f_i\right) \Big/ \left(\sum_{i=1}^{N} \frac{1}{(x-x_i)^2+(y-y_i)^2}\right)$$

an einer Stelle p auswerten ($p_i = (x_i, y_i)$ sind die Stützstellen), so benötigt man dazu, ohne sich besonders geschickt anstellen zu müssen $(3N-2)$ Additionen, $2N$ Subtraktionen, $3N$ Multiplikationen und $(N+1)$ Divisionen. Gehen wir davon aus, daß wir die Berechnung in der Floatingpoint-Einheit eines Intel 486DX Prozessors vornehmen, dann sollten wir realistischerweise noch annehmen, daß jede der Zahlen x_i, y_i und f_i (mindestens) einmal in die Floatingpoint-Einheit geladen werden muß. Also zählen wir noch $3N$ Ladeoperationen dazu.

Wir betrachten nun die Kosten der einzelnen Operationen: Addition, Subtraktion und Division benötigen jeweils 8–20 Taktzyklen (datenabhängig), die Multiplikation benötigt 16 und die Ladeoperation 4 Taktzyklen. Insgesamt können wir eine Auswertung durch $180N$ Taktzyklen nach oben abschätzen. Für einen mit 50 MHz getakteten i486DX ergibt sich bei einer Größenordnung von $N = 1000$ Eingabewerten eine Zeit für die Auswertung von $S^{(2)}(p)$ von ca. 3,6 ms. Gehen wir davon aus, daß das Programm nicht in Assembler, sondern in einer Hochsprache (mit möglicherweise schlecht optimierendem Compiler) geschrieben ist, so kommen wir etwa auf eine Zeit von 7 ms pro Auswertung.

Eine Zeit von 7 ms pro Auswertung voraussetzend, brauchen wir also ca. 460 Sekunden oder ca. 8 Minuten um ein Bild der Größe 256×256 aus den 32×32 Abtastwerten zu interpolieren. Diese Zeitspanne übersteigt keinesfalls in der Bildverarbeitung übliche Zeiten, zumal diese Zeitangaben relativ zu betrachten sind. Sie hängen u.a. von der Architektur des Rechners, dem Compiler und insbesondere von der Optimierung des Compilers ab. Hinzu kommt, daß die globale Shepard-Interpolation trivial parallelisierbar ist, denn während der Berechnung ist keine Kommunikation zwischen den beteiligten Prozessoren notwendig.

Es ist also nicht in erster Linie die Rechenzeit, die es verbietet, die globale Shepard-Interpolation zu benutzen, um aus Abtastwerten von Grauwertbildern wieder Grauwertbilder zu interpolieren. Es ist die mangelnde optische Qualität der Interpolation, die verlangt, sich nach anderen Methoden umzusehen.

Die Abbildungen 2.8 bis 2.11 auf Seite 48 zeigen vier 256×256 Pixel große Bilder, die aus 32×32 Abtastwerten mittels der globalen Shepard-Interpolation berechnet wurden. Die Abtastwerte stammen von dem Testbild Lena (siehe Abbildung 2.7 auf der nächsten Seite). Für den Parameter μ haben wir die Werte 1, 2, 3 und 5 gewählt. Über diesen Parameter stellt man die Größe der ‚Einflußzone' jedes einzelnen Abtastwertes auf seine direkte Umgebung ein. Für $\mu = 1$ ist diese ‚Einflußzone' ziemlich klein, so daß als Folge davon der Grauwert $S^{(1)}(p)$ relativ rasch bis (fast) auf den arithmetischen Mittelwert aller Abtastwerte abfällt, wenn sich der Punkt p auch nur ein wenig von einer Stelle p_i wegbewegt. Der visuelle Eindruck ist der einer grauen Fläche, aus welcher nur die Abtastwerte markant hervorspringen. Das Verhalten der Grauwertfläche entspricht genau dem Verhalten des Graphen im XOR-Problem (siehe Abbildung 2.3 auf Seite 40).

Mit zunehmenden Wert für μ wird die ‚Einflußzone' eines jeden Abtastwertes größer, bis schließlich für $\mu = 5$ ein Wert erreicht ist, wo die Größe der ‚Einflußzone' durch die Zonen der nächsten Abtastwerte begrenzt wird. Deswegen sieht das Bild für $\mu = 5$ (Abbildung 2.11) aus, als sei es mit einfarbigen Rechtecken gepflastert.

Wie wir später sehen werden, lassen sich diese störenden optischen Effekte der Shepard-Interpolation vermeiden, wenn wir einen ‚passenden' Parameterwert μ wählen und die Interpolationsformel nicht global benutzen.

Die Wahl von $\mu = 2$ wird bzgl. des Laufzeitverhaltens durch die Tabelle 2.1 auf Seite 49 nahegelegt. Daß der Parameter $\mu = 2$ in unserem noch zu beschreibenden Verfahren, der *kantenerhaltenden Interpolation*, auch die optisch besten Bilder liefern wird, ist beim Betrachten der Abbildungen 2.8 bis 2.11 jetzt noch nicht zu vermuten.

Abbildung 2.7: Original des Testbildes LENA.

Die in Tabelle 2.1 aufgeführten Rechenzeiten für die Abbildungen 2.8 bis 2.11 wurden auf einem PC-kompatiblen Rechner mit i486DX50-Prozessor unter dem Betriebssystem LINUX (in der Version 1.2.8) gemessen. Als Compiler kam der GNU-Compiler gcc (in der Version 2.5.8) zum Einsatz. Das Programm wurde auf der höchsten Optimierungsstufe ‚-O2' übersetzt. (Die Zeitmessung wurde auf einem völlig unbelasteten System durchgeführt, d.h. das Berechnungsprogramm ist als einziges Programm gelaufen und hatte über die gesamte Laufzeit mehr als 99.9% der gesamten Rechnerkapazität zur Verfügung.)

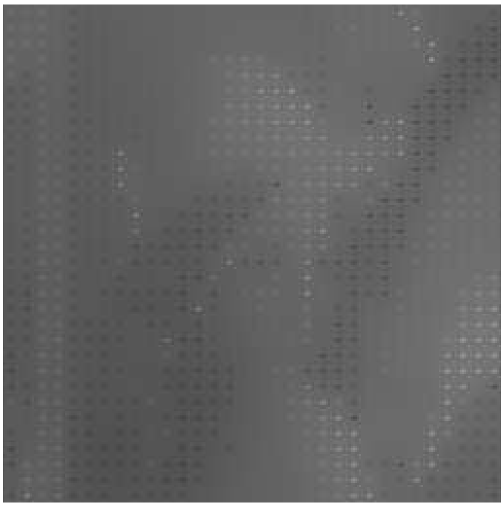


Abbildung 2.8: Globale Shepard-Interpolation zum Parameter $\mu = 1$.

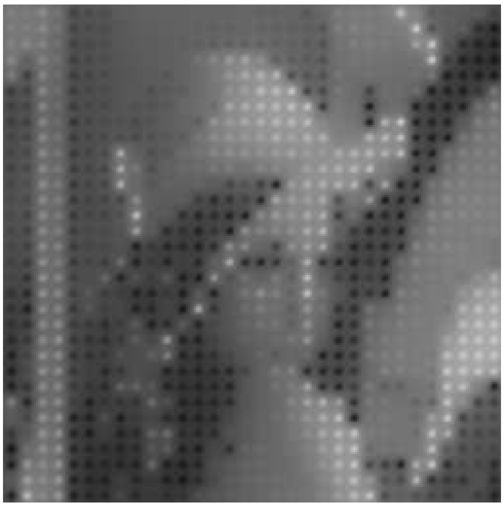


Abbildung 2.9: Globale Shepard-Interpolation zum Parameter $\mu = 2$.

Abbildung 2.10: Globale Shepard-Interpolation zum Parameter $\mu = 3$.

Abbildung 2.11: Globale Shepard-Interpolation zum Parameter $\mu = 5$.

Der Einbruch in der Rechenzeit für $\mu = 2$ erklärt sich durch die einfachere Gestalt der Gewichtsfunktion. Während für $\mu \neq 2$ der Ausdruck

$$\left((x-u)^2 + (y-v)^2\right)^{-\frac{\mu}{2}}$$

ausgewertet werden muß, vereinfacht sich dieser für $\mu = 2$ zu

$$\left((x-u)^2 + (y-v)^2\right)^{-1}.$$

Optimierung	$\mu = 1$	$\mu = 2$	$\mu = 3$	$\mu = 4$	$\mu = 5$
gcc -O2	1535	497	1533	1532	1535
gcc -O1	1507	472	1505	1508	1505
gcc -O0	1778	743	1776	1777	1776

Tabelle 2.1: Rechenzeiten in Sekunden für verschiedene Parameter μ.

Optimierungsstufe	-O2	-O1	-O0
absolute Laufzeit (in sec)	251	307	504
mittlere Auswertungszeit (in msec)	3.8	4.9	7.7

Tabelle 2.2: Mittlere Auswertungszeiten in msec in Abhängigkeit von der Optimierungsstufe des Compilers (für $\mu = 2$).

Wir möchten an dieser Stelle anmerken, daß zur Berechnung der Tabelle 2.1 für alle Parameter μ derselbe Quellcode benutzt wurde. Die Laufzeitersparnis kommt durch eine interne Optimierung der Mathematikbibliothek zustande. Wenn wir die o.g. Vereinfachung explizit im Quellcode berücksichtigen, so sind die Laufzeiten noch einmal um ca. 40% geringer (siehe Tabelle 2.2).

Um die Abhängigkeit der Programmlaufzeiten von der Compileroptimierung zu verdeutlichen, haben wir das Programm zur Berechnung der globalen Shepard-Interpolation zusätzlich mit den Compileroptionen ‚-O0' (keine Optimierung) und ‚-O1' (mäßige Optimierung) übersetzt. Die Ergebnisse der Zeitmessungen sind in Tabelle 2.1 mit aufgenommen. (Eine verwunderliche Feststellung ist, daß in diesem Fall eine höhere Compileroptimierung eine längere Laufzeit ergibt!)

Betrachten wir für $\mu = 2$ die mittlere Dauer einer Auswertung von $S^{(2)}(p)$, so ergibt sich die Tabelle 2.2. In diesem Fall wurde die sich für den Parameter $\mu = 2$ ergebende Vereinfachung des Quellcodes berücksichtigt. Die Überschlagsrechnung hatte demgegenüber eine Zeit von 3.6 msec pro Auswertung ergeben (für $N = 1000$ anstelle von $N = 32^2 = 1024$).

Nachdem wir uns jetzt ausführlicher mit der globalen Shepard-Interpolation in ihrer Urform beschäftigt haben, wollen wir noch kurz einige Modifikationen und Aspekte dieser Interpolationsformel betrachten.

Eine interessante Modifikation der Shepardformel sorgt dafür, daß die Ableitung der modifizierten Formel – im Gegensatz zur ursprünglichen Shepardformel – in den Stützstellen (der Abtastwerte) beliebige wählbare Werte annimmt. Um an den Stützstellen p_i neben den Werten f_i auch Werte von Ableitungen interpolieren zu können, wurde die Shepard-Interpolationsformel dahingehend modifiziert, daß die Werte f_i in Gleichung 2.1 auf Seite 37 durch das Taylorpolynom $T^r f$ der Ordnung r (mit $r < \mu$) einer zu interpolierenden Funktion $f \in C^r$ ersetzt werden. Siehe dazu [BDL83] und [CoTa88].

Die Shepard-Interpolation und einige ihrer Modifikationen wurden auch unter weniger anwendungsorientierten und mehr theorieorientierten Aspekten untersucht. Bei dieser theoretischen Betrachtungsweise unterstellt man, daß die Werte f_i an den Stützstellen p_i von einer Funktion f einer speziellen Funktionenklasse kommen, $f_i = f(p_i)$, und untersucht bei festen (oder variablen) Stützstellen den dadurch induzierten Operator (bzw. die induzierte Familie von Operatoren) zwischen Funktionenräumen. Siehe dazu die Arbeiten [GoWi78], [Fa86], [Co87], [DMT90] und [Sz91]. Zur Illustration diene der folgende Satz.

Satz: *Gegeben seien paarweise verschiedene Punkte $p_i \in \mathbb{R}^2$ $(i = 1, \ldots, N)$. Für den durch die Shepardformel 2.1 induzierten Operator*

$$S^{(2)} \colon (C[0,1]^2, ||.||_\infty) \to (C[0,1]^2, ||.||_\infty)$$
$$S^{(2)} f(p) := \sum_{i=1}^{N} w_i^{(2)}(p) \cdot f(p_i)$$

gelten die folgenden Aussagen:

1. *$S^{(2)}$ ist linear und stetig mit $||S^{(2)}|| = 1$.*
2. *$S^{(2)}$ ist Projektor auf span$\{w_i^{(2)} : i = 1, 2, \ldots N\} \subset C^\infty[0,1]^2$.*
3. *$S^{(2)}$ ist positiv.*

Der Beweis erfolgt durch direkte Rechnung.

2.2 Lokalisierung der Shepard-Interpolation

Wie wir im letzten Abschnitt schon angesprochen haben, ist die Shepard-Interpolation ein globales Verfahren, d.h. bei der Berechnung des Interpolationswertes an einer Stelle $p \in \mathbb{R}^2$ gehen alle, auch sehr weit entfernte Werte f_i mit in die Berechnung ein. Der globale Charakter der Shepard-Interpolation bedingt einen hohen Rechenaufwand.

Wie wir sahen, kann der Rechenaufwand nicht das einzige Argument gegen den Einsatz einer globalen Interpolationsmethode sein. Ein aus unserer Sicht wesentlicherer Aspekt ist, daß wir mit einer globalen – aber auch mit einer lokal uniformen – Interpolation nicht die visuelle Qualität bei den interpolierten Bildern erreichen, die uns von einem ‚Bild' vorschwebt.

Einwände gegen die *globale Shepard-Interpolation* (wie wir das einfache Interpolationsverfahren nennen wollen) sind nicht neu. SHEPARD selbst hat die meisten der späteren Einwände, die gegen eine Anwendung der globalen Shepard-Interpolation sprechen, vorweggenommen und hat versucht durch geeignete Modifikationen zu einer vielseitig einsetzbaren Interpolationsmethode zu kommen. Er schreibt dazu in [Sh68, S. 518f]:

> Though the above method was sufficiently simple and general to be quite attractive, it did have several shortcomings.
>
> 1. When the number of data points is large, the calculation of $z = f_1(P)$ (in unserer Notation $z = S^{(\mu)}(P)$, U.D.) becomes proportionately longer. Eventually the method will become inefficient or impractical.
> 2. Only the distances to P from the data points D_i, and not the direction, are considered. [...]
> 3. The zero directional derivatives obtained at every data point D_i represent an arbitrary and undesirable constraint on the interpolated surface.
> 4. Computational error becomes significant in the neighborhood of points D_i, as the predominant term results from the difference of two almost equal numbers. [Sh68, S. 518f]

Zu allen aufgeführten Mängeln schlug SHEPARD Modifikationen der *globalen Shepard-Interpolation* vor. Während die Modifikationen, die zu den Punkten 1, 3 und 4 korrespondieren, von anderen Autoren aufgegriffen wurden, blieb die Modifikation, die die Interpolation ‚richtungssensitiv' macht (Punkt 2), von späteren Autoren unberücksichtig. Und dies' obwohl die Modifikation auf einer leicht verständlichen Idee beruht.

> In order to improve the interpolation it was clear that a direction factor, in addition to a distance factor, was needed in defining weightings. Intuitively this represented the "shadowing" of the influence of a data point from P by a nearer one in the same direction. [Sh68, S. 520]

Die explizite Einführung einer ‚Richtungsgewichtung' zusätzlich zu der ‚Distanzgewichtung' wird bei unserer *kantenerhaltenden Grauwertinterpolation* zwar vermieden, aber die Idee des ‚shadowing' wird sich bei uns wiederfinden (siehe die Definition des Sichtbarkeitsbereichs auf Seite 59). Dieses ‚shadowing' ist ganz eng verbunden mit einer weiteren Modifikation, die ebenso wie Punkt 2, von SHEPARD angedacht und später vergessen oder ignoriert wurde.

> When the foregoing interpolation algorithm is used in city and regional planning, where the data points represent the centers of data collection areas (such as census tracts), it is assumed that whenever two areas are adjacent, there is some logical relationship between them. If some physical barrier such as a river, railroad or expressway separates the areas, the logical relation may be attenuated. Through the inclusion of barriers a user may specify discontinuities in the metric space in which d_i is calculated to simulate this attenuation. [Sh68, S. 521]

Die Idee der ‚Barriere' ist gerade in unserer *kantenerhaltenden Grauwertinterpolation* von entscheidender Bedeutung. Die bei uns zu berücksichtigenden Grauwertkanten haben den von SHEPARD angesprochenen Barrierecharakter, denn sie trennen Helligkeitswerte, die zu unterschiedlichen Grauwertflächen gehören. Weder die globale Shepard-Interpolation, noch die bisher bekannten daraus abgeleiteten Interpolationsverfahren sind in der Lage, solche zusätzlichen Daten zu nutzen. In der Folge sind sie auch nicht in der Lage, visuell ansprechende Bilder aus den Helligkeitsabtastwerten zu rekonstruieren (bzw. zu interpolieren).

Wir werden im folgenden eine lokale Variante, die *Modified Quadratic Shepard*-Methode (MQS) vorstellen, wie sie von RENKA in [Re88b] implementiert wurde. Wir stellen die MQS-Methode hier vor, da wir unsere *kantenerhaltende Grauwertinterpolation* später mit ihr vergleichen werden.

Modified Quadratic Shepard (Prinzip): *Gegeben sind Punkte (x_i, y_i), sowie die Werte f_i (einer unbekannten Funktion f) für $i = 1, 2, \ldots, N$. Weiter sind gegeben zwei Zahlen $n_w, n_q \in \mathbb{N}$. Dann ist der Interpolationswert nach der Modified Quadratic Shepard-Methode an einer Stelle (x, y) gegeben durch*

$$MQS(x,y) := \Big(\sum_{k=1}^{N} w_k(x,y) \cdot Q_k(x,y)\Big) \Big/ \Big(\sum_{k=1}^{N} w_k(x,y)\Big).$$

Die Gewichtsfunktionen w_k sind definiert durch

$$w_k(x,y) := \left(\frac{\phi\Big(r_{w,k} - ||(x,y) - (x_k, y_k)||\Big)}{r_{w,k} \cdot ||(x,y) - (x_k, y_k)||}\right)^2$$

mit der Abschneidefunktion $\phi\colon \mathbb{R} \to \mathbb{R}$

$$\phi(x) = \begin{cases} x & \text{falls } x > 0 \\ 0 & \text{falls } x \leq 0 \end{cases}.$$

Dabei wird der Radius $r_{w,k}$ so gewählt, daß in der (euklidischen) $r_{w,k}$-Umgebung um den Punkt (x_k, y_k) n_w andere der Punkte (x_i, y_i) liegen.

Die quadratische Funktion Q_k ist eine lokale Approximation von f an der Stelle (x_k, y_k) und ist definiert durch

$$\begin{aligned} Q_k(x,y) := {} & \overline{c}_{k,1}(x-x_k)^2 + \overline{c}_{k,2}(x-x_k)(y-y_k) + \\ & + \overline{c}_{k,3}(y-y_k)^2 + \overline{c}_{k,4}(x-x_k) + \overline{c}_{k,5}(y-y_k) + f_k. \end{aligned}$$

Dabei minimieren die Koeffizienten $\overline{c}_k \in \mathbb{R}^5$ die folgende Summe

$$\sum_{\substack{j=1 \\ j \neq k}}^{N} q_j(x_k, y_k)\Big(c_{k,1}(x_j - x_k)^2 + \cdots + c_{k,5}(y_j - y_k) + f_k - f_j\Big)^2$$

mit den Gewichtsfunktionen

$$q_j(x,y) := \left(\frac{\phi\Big(r_{q,j} - ||(x,y) - (x_j, y_j)||\Big)}{r_{q,j} \cdot ||(x,y) - (x_j, y_j)||} \right)^2 .$$

Analog zu der Wahl der Radien $r_{w,k}$ werden die Radien $r_{q,j}$ so gewählt, daß in einer $r_{q,j}$-Umgebung um den Punkt (x_j, y_j) n_q andere der Punkte (x_i, y_i) liegen.

Für weitere Details zur Auswahl der Radien $r_{w,k}$ und $r_{q,k}$, sowie für Details der Implementierung siehe [Re88a]. Der FORTRAN77-Quelltext zu [Re88b] ist u.a. in der elektronischen Bibliothek des Konrad-Zuse-Zentrums für Informationstechnik Berlin (oder jedem anderen NETLIB-Spiegel) per Anonymous-FTP zu beziehen (Rechner: `elib.zib-berlin.de`, Datei `660.Z` im Verzeichnis `/netlib/toms`). Dieser Quelltext liegt auch der von uns benutzten C-Implementation der MQS-Methode zugrunde, mit welcher einige die folgenden Abbildungen erzeugt wurden.

Die Abbildungen 2.12 und 2.13 auf der nächsten Seite wurden beide mit der MQS-Methode erzeugt. Bei Abbildung 2.12 wurden 32×32 Abtastwerte und bei Abbildung 2.13 64×64 Abtastwerte für die Interpolation verwendet. Bei beiden Bildern wurden die freien Parameter n_w und n_q zu $n_w = n_q = 6$ gewählt, um den Einfluß der Abtastwerte möglichst lokal zu halten. RENKA empfiehlt in [Re88a] die Werte $n_w = 19$ und $n_q = 13$ (siehe Abbildungen 2.14 und 2.15).

Wir haben darüber hinaus noch andere Parameterkombinationen getestet, aber die Erfahrung zeigt, daß die visuelle Qualität der nach der MQS-Methode erzeugten Grauwertbilder in erster Linie von der Anzahl der Abtastwerte abhängt und daß die wählbaren Parameter n_w und n_q wenig bis keinen Einfluß haben.

Abbildung 2.12: MQS-Interpolation mit 32×32 Abtastwerten und den Parametern $n_w = n_q = 6$.

Abbildung 2.13: MQS-Interpolation mit 64×64 Abtastwerten und den Parametern $n_w = n_q = 6$.

Die MQS-Methode ist, im Gegensatz zur globalen Shepard-Interpolation, in der Lage, Grauwertverläufe gut zu reproduzieren. Beispiele von solchen Verläufen finden wir in den Abbildungen 2.13 und 2.15 im Bereich des Huts und der Schulter. Was die visuelle Qualität der interpolierten Bilder beeinträchtigt, ist die ‚zackige' Ausprägung der Helligkeitsverläufe in der Nähe von Helligkeitskanten (des Originals). Dieses ist aber nicht weiter verwunderlich, da die MQS-Methode nicht dafür ausgelegt wurde, besonders gut Helligkeitskanten zu reproduzieren.

2.3 Kantenerhaltende Grauwertinterpolation

In den beiden vorherigen Abschnitten haben wir die *globale Shepard-Interpolation* und die *Modified-Quadratic-Shepard*-Methode vorgestellt. Dabei haben wir die globale Shepard-Interpolation ausgewählt, weil diese Interpolation sozusagen den Urvater einer ganzen Klasse von Interpolationsverfahren darstellt, der sogenannten *distanzgewichteten Interpolationsverfahren.*

Die MQS-Methode ist eine der modernsten Varianten aus dieser Klasse, sie ist aber gleichzeitig ein Verfahren, welches ein breites Anwendungsspektrum abdecken soll. Deswegen ist es auch leicht einsichtig, daß dieses allgemein ausgelegte Verfahren im direkten Vergleich mit unserer *kantenerhaltenden Interpolation* bei der Interpolation von Bilddaten schlechter abschneiden wird. Leider sind uns keine weiteren Verfahren bekannt, die sich speziell mit

Abbildung 2.14: MQS-Interpolation mit 32×32 Abtastwerten und den Parametern $n_w = 19$ und $n_q = 13$.

Abbildung 2.15: MQS-Interpolation mit 64×64 Abtastwerten und den Parametern $n_w = 19$ und $n_q = 13$.

der Interpolation von Bilddaten beschäftigen, sonst hätten wir sie in den Vergleich mit einbeziehen können.

Die folgenden Abbildungen zeigen neben schon bekannten Bildern auch die Ergebnisse der *kantenerhaltenden Grauwertinterpolation*. Abbildung 2.16 auf der nächsten Seite zeigt das vorverarbeitete Ausgangsbild, (Original siehe Abbildung 2.7 auf Seite 47) von welchem die Abtastwerte stammen und die Abbildung 2.17 auf der nächsten Seite zeigt die aus diesem Ausgangsbild maschinell extrahierte Kanteninformation. Beide Verarbeitungsstufen wurden schon in Kapitel 1 beschrieben.

Abbildung 2.18 auf Seite 57 zeigt das Ergebnis einer globalen Shepard-Interpolation von 64×64 Abtastwerten des abstrahierten Bildes (Abbildung 2.16) bei einem Parameter $\mu = 2$. (Zur Veranschaulichung des Einflußes der Parameterwahl siehe Abbildung 2.8 bis 2.11 auf Seite 48.) Durch Anwendung der weiter vorne angeführten MQS-Methode entstand Abbildung 2.19 auf Seite 57. Diese Abbildung basiert auf den selben Daten wie Abbildung 2.18. Die bei der MQS-Methode wählbaren Parameter n_w und n_q wurden zu $n_w = n_q = 6$ gewählt.

Die Abbildungen 2.20 und 2.21 auf Seite 58 sind die Ergebnisse der *kantenerhaltenden Grauwertinterpolation*, wie sie im folgenden vorgestellt wird. Die Abbildungen basieren auf den Kanteninformationen aus Abbildung 2.17 und benutzen 32×32 (Abb. 2.20) bzw. 64×64 Abtastwerte (Abb. 2.21) aus Abbildung 2.16 auf der nächsten Seite (dem ‚abstrahierten‘ Bild).

Abbildung 2.16: Vorverarbeitete Version des Testbilds Lena.

Abbildung 2.17: Kanteninformation des Testbilds Lena.

Abbildung 2.18: Shepard-Interpolation ($\mu = 2$) aus 64×64 Grauwerten.

Abbildung 2.19: MQS-Interpolation ($n_w = n_q = 6$) aus 64×64 Grauwerten.

Abbildung 2.20: *Kantenerhaltende Grauwertinterpolation* aus 32×32 Werten.

Abbildung 2.21: *Kantenerhaltende Grauwertinterpolation* aus 64×64 Werten.

Die *kantenerhaltende Grauwertinterpolation* greift auf die Barriereidee zurück, die auch schon von SHEPARD formuliert wurde (siehe Seite 52). Bei dieser Interpolation werden nur solche Abtaststellen p_i zur Berechnung eines Interpolationswertes an einer Stelle p benutzt, die nicht ‚hinter' einer Kante liegen, d.h. es werden die Werte nicht berücksichtigt, die außerhalb des Sichtbarkeitsbereichs eines Pixels p liegen. Wie in Kapitel 1 schon dargestellt, gibt es unterschiedliche Interpretationsmöglichkeiten der endlich vielen Zahlen im Rechner, die unser Bild ausmachen. Bei der Definition des Sichtbarkeitsbereichs $\mathcal{S}(p)$ legen wir die Flächeninterpretation der diskreten Kantendaten zugrunde.

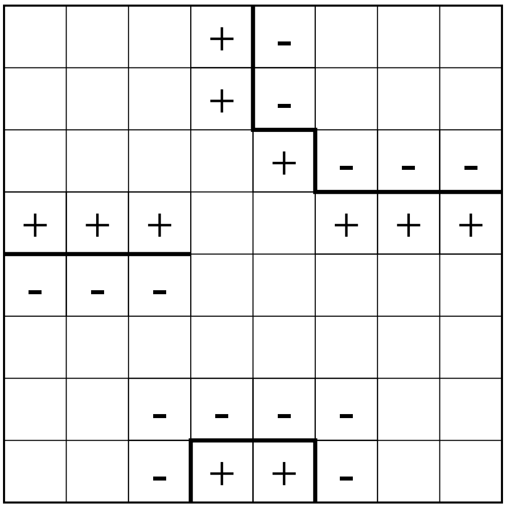


Abbildung 2.22: Kanteninformation k und Menge der Kantenstücke $\mathcal{K}(k)$.

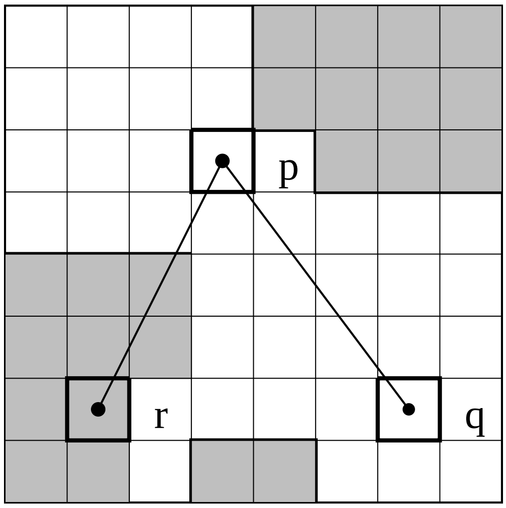


Abbildung 2.23: Sichtbarkeitsbereich $\mathcal{S}(p)$ (hell markiert).

Definition: *Gegeben sei die Kanteninformation* $k\colon \mathcal{R} \to \{+,-,0\}$. *Der Sichtbarkeitsbereich* $\mathcal{S}(p)$ *eines Pixel* $p \in \mathcal{R}$ *ist die Menge*

$$\mathcal{S}(p) = \{q \in \mathcal{R} \mid q \neq p \;\wedge\; \overline{pq} \cap s = \emptyset \quad \forall s \in \mathcal{K}(k)\}.$$

Dabei ist $\overline{pq}$ *die Verbindungsstrecke der Pixel* p *und* q *(siehe Seite 13), und* $\mathcal{K}(k)$ *ist die zu der Kanteninformation* k *gehörende Menge von Kantenstücken.*

Zur Verdeutlichung der obigen Definition haben wir in Abbildung 2.22 eine Kanteninformation k für ein 8×8-Bild angegeben. Die zu dieser Kanteninformation gehörende Menge $\mathcal{K}(k)$ von Kantenstücken ist mit einer stärkeren Strichdicke eingezeichnet. Der besseren Übersichlichkeit wegen sind die Nullwerte der Kanteninformation in der Zeichnung ausgespart worden. Der helle

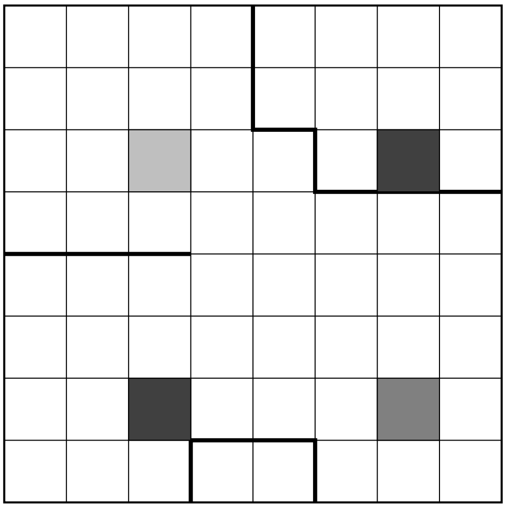


Abbildung 2.24: Vier Abtastwerte.

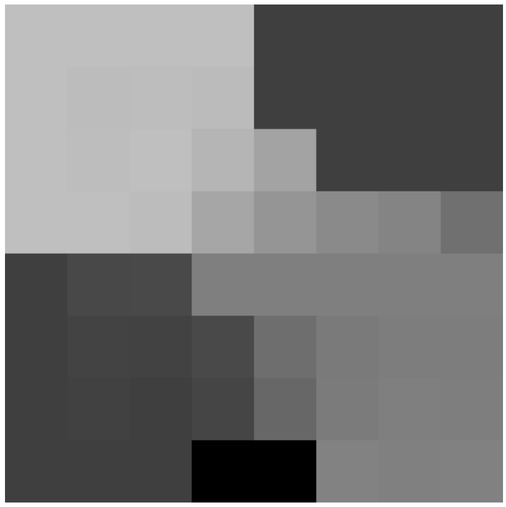


Abbildung 2.25: Interpolationsergebnis aus Abbildung 2.24.

Bereich in Abbildung 2.23 zeigt den Sichtbarkeitsbereich $\mathcal{S}(p)$ des Pixels p. Mit obiger Definition stellt man leicht fest, daß das Pixel q im Sichtbarkeitsbereich $\mathcal{S}(p)$ liegt, das Pixel r aber nicht (denn die Verbindungsstrecke $\overline{pr}$ schneidet ein Kantenstück). Für die Berechnung eines Interpolationswertes an der Stelle p werden jetzt nur noch Abtastwerte f_i benutzt, deren Stellen p_i im Sichtbarkeitsbereich $\mathcal{S}(p)$ von p liegen. Dadurch wird die grundlegende Interpolationformel $S^{(2)}(p)$ modifiziert zu $S_k(p)$.

Definition: *Gegeben seien paarweise verschiedene Punkte p_i und reelle Werte f_i, $(p_i, f_i) \in \mathcal{R} \times \mathbb{R}$ für $i = 1, 2, \ldots, N$. Die Interpolation im Sichtbarkeitsbereich $\mathcal{S}(p)$ eines Pixels $p \in \mathcal{R}$ ist gegeben durch*

$$S_k(p) = \begin{cases} f_i & \textit{falls } p = p_i \\ \Big(\sum\limits_{p_i \in \mathcal{S}(p)} \dfrac{f_i}{||p - p_i||^2} \Big) \Big/ \Big(\sum\limits_{p_i \in \mathcal{S}(p)} \dfrac{1}{||p - p_i||^2} \Big) & \textit{sonst} \end{cases} \tag{2.2}$$

Wir können die Interpolation S_k dahingehend interpretieren, daß sie aus der Shepard-Interpolationsformel 2.1 auf Seite 37 durch Änderung der Metrik, wie sie SHEPARD zur Simulation von Barriere-Verhalten vorschlug, entstanden ist. Ersetzen wir nämlich in der Shepard-Interpolationsformel $S^{(2)}(p)$ die euklidische Metrik $||p - p_i||$ durch eine ‚Pseudometrik' $d(p, p_i)$ (es ist keine Metrik, da sie die Dreiecksungleichung verletzt), so geht die Shepard-Formel in unsere Formel 2.2 über. Dazu muß die ‚Pseudometrik' $d(p, p_i)$ folgendermaßen definiert sein:

$$d(p, q) = \begin{cases} ||p - q|| & \text{falls } q \in \mathcal{S}(p) \\ +\infty & \text{sonst} \end{cases} .$$

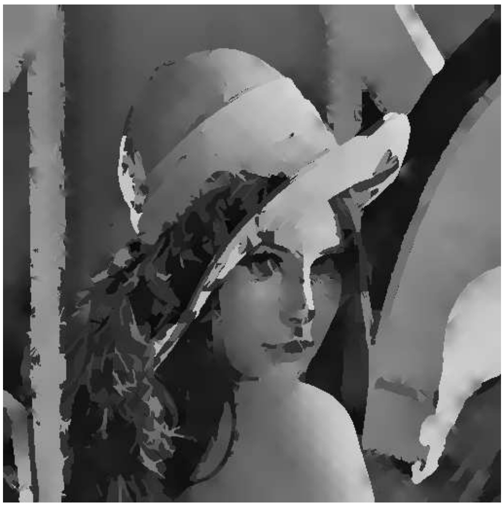


Abbildung 2.26: Wert $w = 50$.

Abbildung 2.27: Wert $w = 200$.

Folgen wir der Konvention $\frac{1}{\infty} = 0$, so führt diese Definition direkt zu S_k.

Leider hat die so erhaltene Interpolationsformel noch den Fehler, daß sie nicht überall definiert ist. In unserem einfachen Beispiel von oben, sieht man dies sofort. Nehmen wir einmal an, wir hätten vier Abtastwerte gegeben, welche wie in Abbildung 2.24 verteilt sind. Dann ist es nicht möglich für die zwei Pixel in der untersten Zeile Werte zu berechnen. Dieses ist in Abbildung 2.25 daran zu erkennen, daß dort noch schwarze Flecken zu sehen sind.

Wir sind jetzt vor die Aufgaben gestellt, ein Verfahren zu entwickeln, welches die positiven Eigenschaften der Interpolationsformel S_k (Kantenerhaltung) weiterhin besitzt, gleichzeitig aber für alle Pixel eines Bildes auch Werte liefert.

Ein möglicher Ansatz wäre, die Abtaststellen so zu legen, daß mit der Interpolationsformel S_k für alle Pixel des Bildes immer ein Wert berechnet werden kann. Diese Vorgehensweise verlagert aber die Schwierigkeit auf die Auswahl der Eingangsdaten. Wir wollen lieber den Weg beschreiten, das Interpolationsverfahren so auszulegen, daß es auch bei nicht gut gewählten Eingangsdaten überall Ausgangsdaten produziert.

Ein weiterer Ansatz könnte sein, alle Pixel des Bildes, welche durch die Interpolationsformel keine Werte erhalten haben, mit einem festen Wert w, der irgendwie gewählt wird, zu belegen. Dieser Ansatz liefert nicht nur visuell wenig ansprechende Bilder, wie die Abbildungen 2.26 (für $w = 50$) bzw. 2.27 (für $w = 200$) zeigen, sondern er vernachlässigt auch vorhandene Informationen. Wir wissen nämlich, daß sich an Kanten die Helligkeit sprunghaft ändert und wir wissen in welche Richtung (dunkler/heller), denn dieses ist in unserer Kanteninformation mitkodiert. Was wir nicht wissen, ist die absolute Höhe des Helligkeitssprungs.

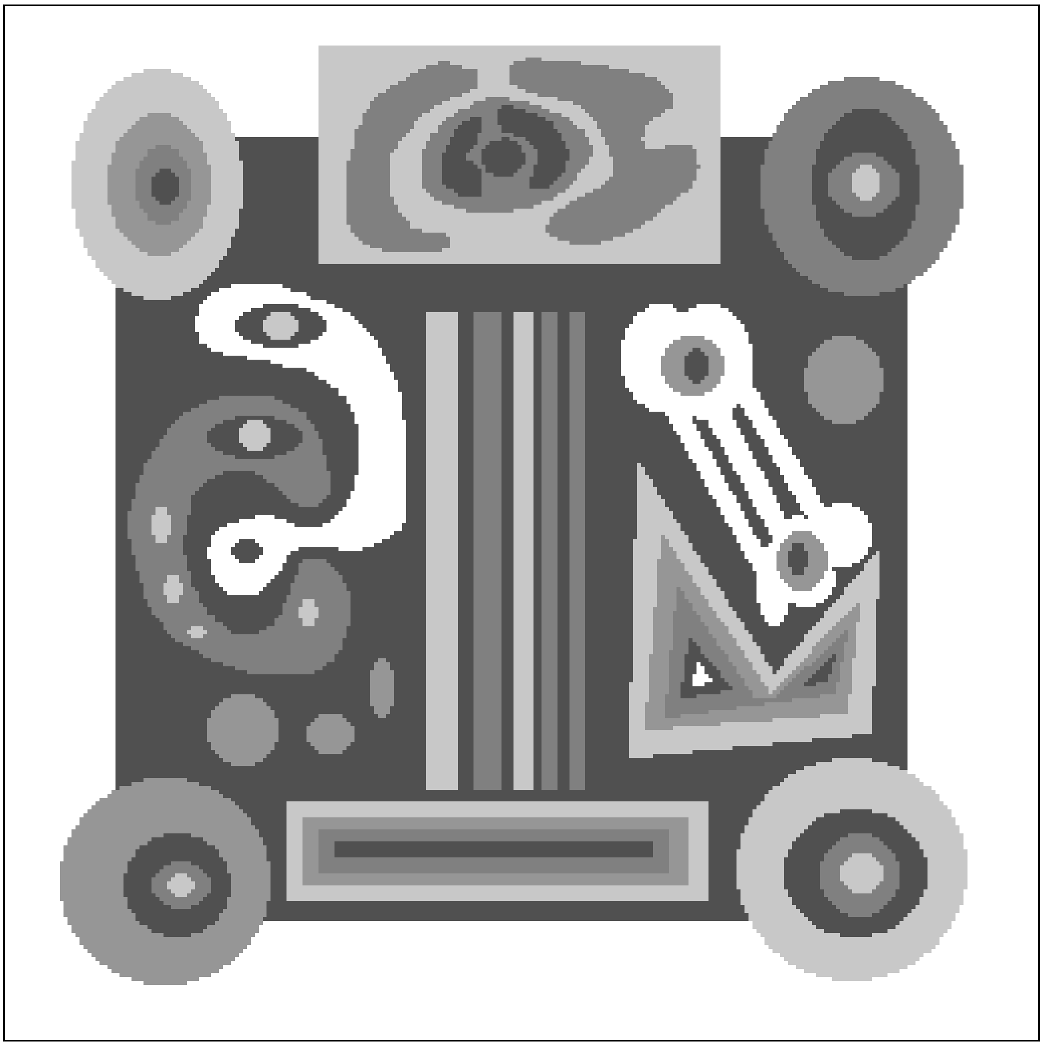


Abbildung 2.28: Testmuster zur Beschreibung der Probleme bei der kantenerhaltenden Interpolation

Um ein wenig Gefühl zu bekommen, was für einzelne Situationen, sprich Problemefälle bei der Interpolation, bei natürlichen Bildern auftreten können, wollen wir das folgende Testmuster betrachten. In diesem sind viele Probleme prototypisch isoliert.

Dieses Testmuster (Abbildung 2.28) ist selbstverständlich im Vergleich zu natürlichen Bildern noch sehr einfach, da alle Grauwertflächen konstant sind. Es ist auch von seiner (topologischen) Struktur her einfach aufgebaut, zeigt aber schon alle typischen Strukturen, die auch bei natürlichen Bildern auftreten. Es enthält kompakte Objekte – im Sinn kleines Verhältnis von Umfang zu Fläche –, länglich ausgedehnte Objekte, ‚konvexe' und nicht ‚konvexe' Objekte.

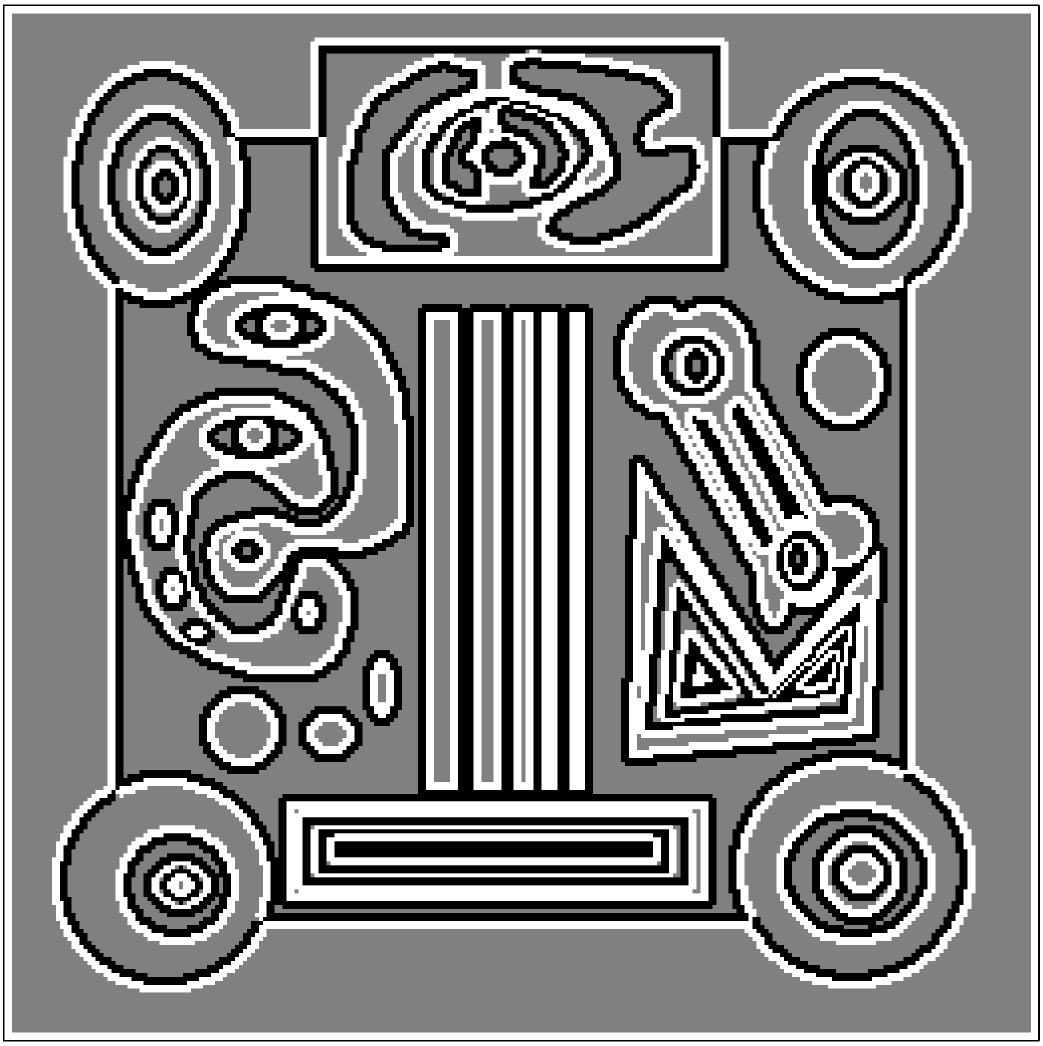


Abbildung 2.29: Kanteninformation aus Abbildung 2.28

Verschiedene Objekte sind mit unterschiedlichen Helligkeitsabfolgen ineinander geschachtelt. Die fünf länglichen Rechtecke in der Mitte erzeugen dünne Linien im Hintergrund. An einigen Stellen sind Flächen gleichen Grauwerts über teilweise nur ein Pixel breite ‚Stege' miteinander verbunden. Dieses sind besonders kritische Stellen für die Kantendetektion. Weitere für die Kantendetektion kritische Stellen sind die Tripelpunkte, in denen drei unterschiedliche Grauwertflächen direkt aneinandergrenzen. Hier ist es der weiße Hintergrund, das dunkelgraue Hintergrundrechteck und das Mittelgrau der Ovale, welche die vier Ecken des Hintergrundrechtecks verdecken.

Zur Vereinfachung der Diskussion bezeichnen wir einige der Gebiete.

Das TESTMUSTER besteht aus einem weißen Hintergrund und einem großen dunkelgrauen *Hintergrundrechteck*, auf welchem sich alle anderen Objekte ganz oder teilweise befinden. In der Mitte des Hintergrundrechtecks finden sich fünf *lange Rechtecke* und unter diesen vier geschachtelte Rechtecke, die *Rechteckkonfiguration*. Links der langen Rechtecke befindet sich ein mittelgraues *Embryo* mit *Auge* und *Flecken*, sowie ein weißer *Fisch* mit *Auge*. Rechts der langen Rechtecke befindet sich ein weißer *Knochen* mit *Augen* und unter diesem die *Dreieckskonfiguration*. Über den vier Ecken des Hintergrundrechtecks liegen vier *Ellipsenkonfigurationen*, jede besteht aus vier geschachtelten Ellipsen.

Nach dieser Sprachregelung wollen wir uns die aus diesem TESTMUSTER automatisch erzeugte Kanteninformation etwas genauer ansehen. Die Abbildung 2.29 auf der vorherigen Seite zeigt diese Kanteninformation, welche aus dem TESTMUSTER 2.28 auf Seite 62 durch den in Kapitel 1 beschriebenen Prozeß erzeugt wurde. Der Parameter σ wurde zu $\sigma = 0.7$ gewählt.

Zur genaueren Betrachtung typischer kritischer Stellen, haben wir die in der Abbildung 2.30 markierten Regionen R1, R2 und R3 vergrößert. Diesen Ausschnittsvergrößerungen sind die entsprechenden Ausschnittsvergrößerungen aus der Kanteninformation gegenübergestellt.

Die Region R1, ein 32×32 Pixel großer Ausschnitt, ist ausgewählt worden, um die Probleme mit dünnen Strukturen zu diskutieren. Die Region R1 im TESTMUSTER besteht aus verschiedenen Bereichen mit drei unterschiedlichen Helligkeitswerten. Wir wollen sie für die folgende Diskussion *dunkelgrau*, *mittelgrau* und *hellgrau* nennen.

Die Region R1 besitzt im unteren Teil drei dunkelgraue Gebiete, die durch ein mittelgraues Gebiet voneinander getrennt sind. Im oberen Teil finden wir zwei mittelgraue Gebiete, welche durch ein hellgraues getrennt werden. Interessant sind nun u.a. drei Stellen in dem mittelgrauen Gebiet, das die dunkelgrauen trennt, wo dieses nur ein Pixel breit ist. Eine Stelle finden wir am unteren Rand der Region R1, dort werden zwei dunkelgraue Bereiche getrennt. Die beiden anderen Stellen finden wir rechts nahe der Mittelsenkrechten. Dort werden zum einen das rechte dunkelgraue Gebiet von dem mittelgrauen getrennt und zum anderen das rechte dunkelgraue Gebiet von dem hellgrauen.

Betrachten wir nun diese Stellen in der Kanteninformation, so stellen wir fest, daß sich an den Stellen der dunkel-mittel-dunkelgrau-Übergänge des Testmusters, entsprechend Minus-Plus-Minus-Übergänge befinden. Der Plus-Bereich ist dabei nur ein Pixel breit und entsteht durch Überlagerung der Plus-Bereiche aus den Minus-Plus- und Plus-Minus-Übergängen, welche zu den dunkelgrau-mittelgrau- und mittelgrau-dunkelgrau-Übergängen des Testmusters gehören.

Anders ist dieses bei dem hellgrau-mittelgrau-dunkelgrau-Übergang (von oben nach unten). Bei diesem kommen von den beiden Kanten Wälle mit ver-

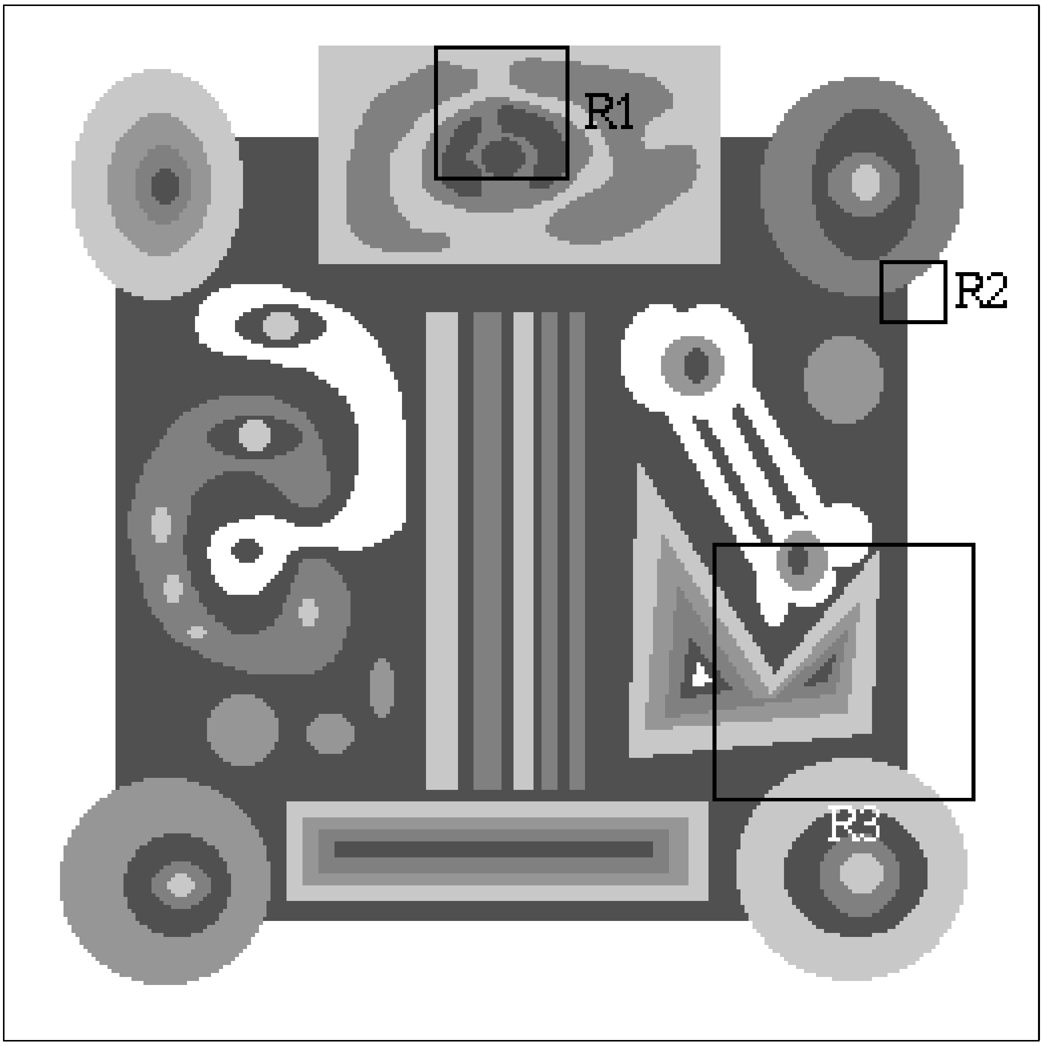


Abbildung 2.30: Die Regionen R1, R2 und R3 im TESTMUSTER

schiedenen Vorzeichen zur Überlagerung und während dieses bei räumlich eng benachbarten Kanten (ca. 2–4 Pixel Abstand) zur Bildung von zusätzlichen Plus-Minus-Übergängen führt, führt es in unserem Fall zweier räumlich sehr eng benachbarter Kanten (nur ein Pixel Abstand!) zum Verlust eines Übergangs. An dieser Stelle ist von beiden Kanten nur eine detektierbar.

Die Region R2 ist ein 16 × 16 Pixel großer Ausschnitt eines Tripelpunktes, d.h. eines Punktes, in welchem drei Gebiete unterschiedlicher Helligkeit zusammenstoßen. In solchen Punkten ist es prinzipiell nicht möglich, mit unserem Verfahren alle Kanten richtig zu detektieren. Trotzdem machen solche Punkte bei der Interpolation keine großen Probleme, wenn nur genügend viele Abtastpunkte vorliegen.

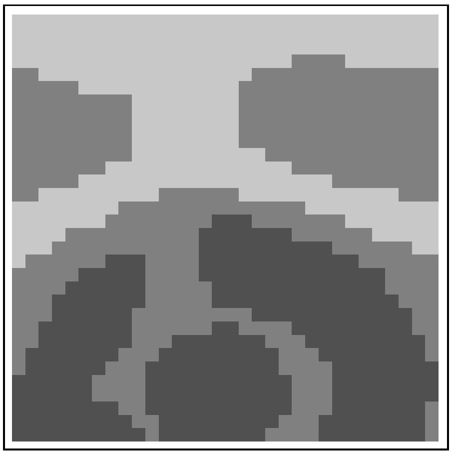


Abbildung 2.31: Ausschnitt R1

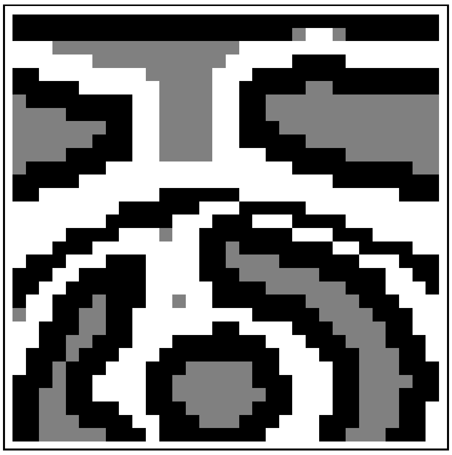


Abbildung 2.32: Ausschnitt R1

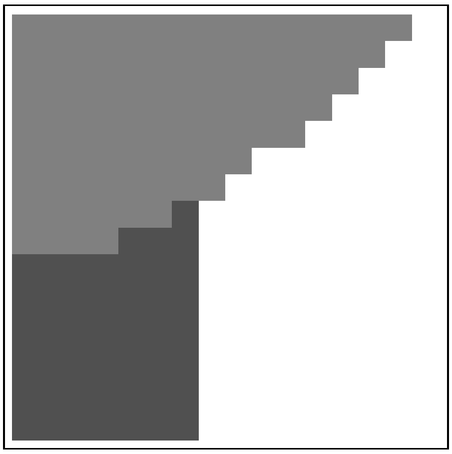


Abbildung 2.33: Ausschnitt R2

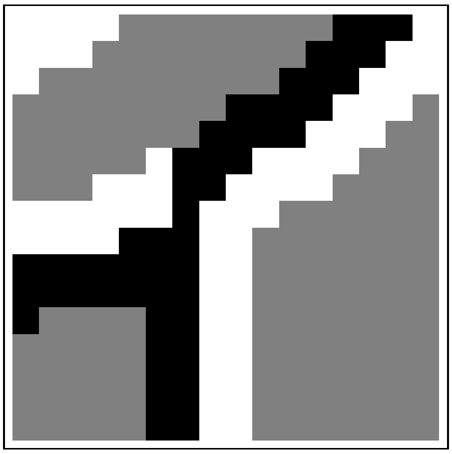


Abbildung 2.34: Ausschnitt R2

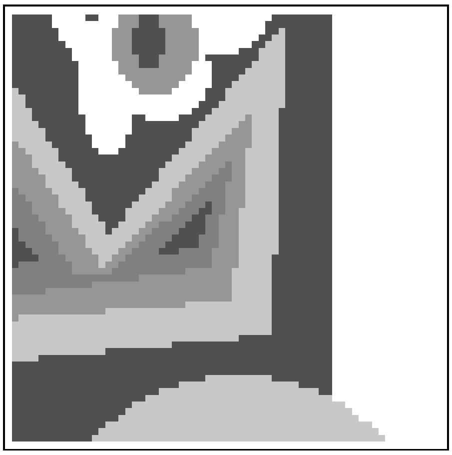


Abbildung 2.35: Ausschnitt R3

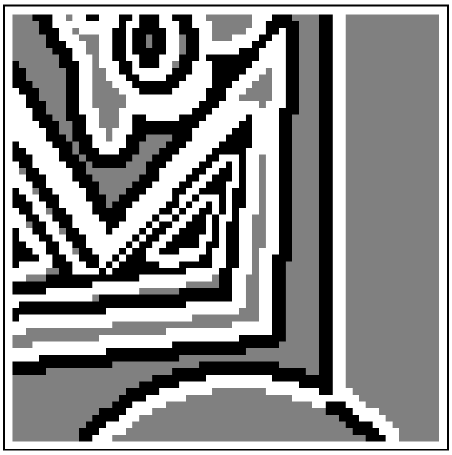


Abbildung 2.36: Ausschnitt R3

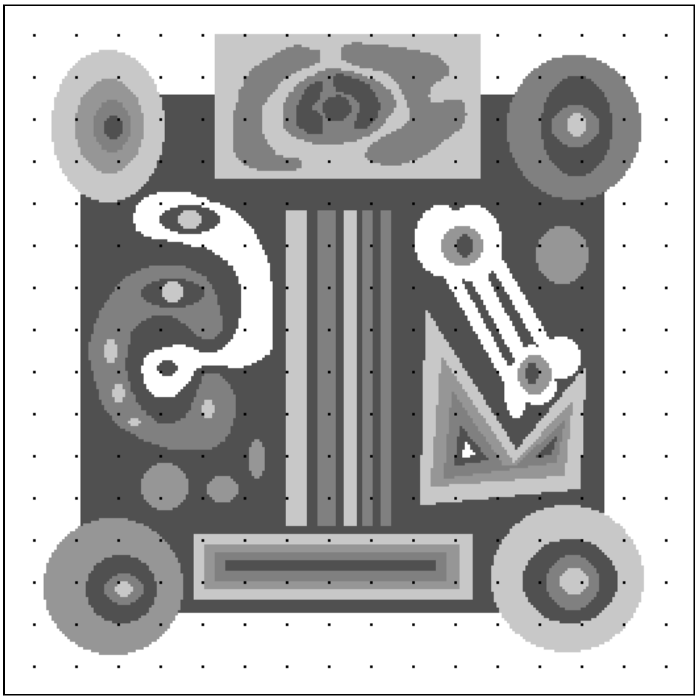


Abbildung 2.37: Auswahl der Abtastwerte für die Interpolation.

Region R3 ist ein Ausschnitt der Größe 64×64 Pixel in welcher die Problemtypen aus R1 und R2 noch einmal versammelt sind. Betrachten wir beispielsweise einen waagerechten Streckenzug startend in dem dunkelgrauen Dreieck in der Mitte der Dreieckskonfiguration und endend rechts im weißen Hintergrund, so liegen auf diesem Streckenzug fünf Kanten (im Sinne von Helligkeitssprüngen). Folgen wir dem gleichen Streckenzug in der Kanteninformation, so finden wir dort sieben Plus-Minus- bzw. Minus-Plus-Übergänge. Dieses ist auf zwei dunkel-mittel-hell Kantenfolgen in der Dreieckskonfiguration zurückzuführen. Betrachtet man Region R3 in aller Ruhe, so findet man dort alle schon besprochenen Probleme, sowie deren Kombinationen.

Doch kommen wir von der Kantendetektion zurück zu unserem Interpolationsverfahren. Da wir unser Augenmerk nicht auf eine geschickte Wahl der Abtaststellen legen wollen, benutzen wir ein reguläres Gitter. Dabei sind die 4-Nachbarn jeweils 16 Pixel voneinander entfernt, d.h. aus unserem 256×256 Pixel großem TESTMUSTER benutzen wir 16×16 Pixel als Abtastwerte, wie in Abbildung 2.37 zu sehen. Es wird dann für jedes Pixel p versucht, aus der Kanteninformation und den Abtastwerten einen neuen Grauwert zu berechnen. Dazu wird die Interpolationformel S_k (von Seite 60) ausgewertet, aber mit dem Unterschied, daß aus Rechenzeitgründen nicht der gesamte Sichtbarkeitsbereich $\mathcal{S}(p)$ berücksichtigt wird, sondern nur der Teil von $\mathcal{S}(p)$, der in

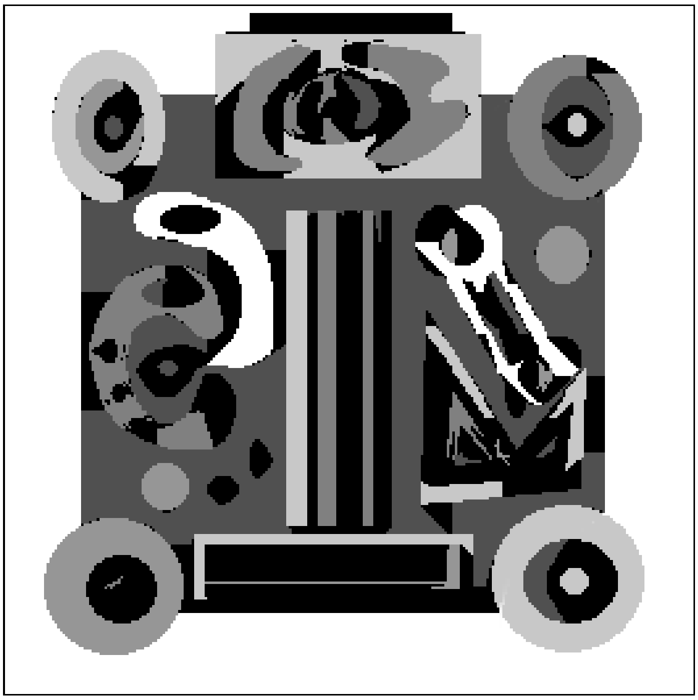


Abbildung 2.38: Erster Interpolationsschritt.

einer (abgeschlossenen) Umgebung $U_r(p)$ ($r \in \mathbb{Z}$) des Pixels p (in der Maximumsnorm) liegt. Es wird also für jedes Pixel eine zu Formel 2.2 auf Seite 60 analoge Formel ausgewertet, bei der die Summation über $\mathcal{S}(p)$ durch eine über $\mathcal{S}(p) \cap U_r(p)$ ersetzt wird. Das Ergebnis für $r = 17$ zeigt Abbildung 2.38. Wie zu erwarten war, konnte bei dieser Interpolation nicht zu jedem Pixel auch ein Grauwert berechnet werden (diese Stellen sind in der Abbildung schwarz wiedergegeben).

Wir wollen im folgenden die Ursachen untersuchen, die dazu führen, daß an bestimmten Stellen keine Werte berechenbar sind. Und wir wollen ausgehend von den dabei gewonnenen Erkenntnissen eine Heuristik entwickeln, welche es gestattet durch einen iterativen Prozeß zu einem vollständigen Bild (überall definierte Interpolation) zu gelangen.

Als erstes ist natürlich klar, daß der Übergang von $\mathcal{S}(p)$ zu $\mathcal{S}(p) \cap U_r(p)$ zusätzliche Stellen ohne berechenbare Grauwerte produziert, nämlich immer dann, wenn $p_i \in \mathcal{S}(p)$ aber $\{p_j \mid j = 1, \ldots, N\} \cap \mathcal{S}(p) \cap U_r(p) = \emptyset$ gilt. Oder anders ausgedrückt: Ist die Distanz von p zu einer Abtaststelle p_i größer als r, so wird sie nicht berücksichtigt – auch wenn keine Abtaststelle mit einer kleineren Distanz zu p existiert! Durch diesen Effekt erzeugte schwarze Flächen, oder Teile von solchen, finden wir in Abbildung 2.38 einige: Z.B. in der linken Bildhälfte sind Teile der Grenzen des dunkelgrauen Hintergrundrechtecks zu der Rückenpartie des Embryos auf diese Weise entstanden. Solche schwarzen Flächen sind durch waagerechte oder senkrechte Grenzen gekennzeichnet, die nicht durch Kanteninformationen begründet werden.

Eine weitere Ursache für nicht berechnete Werte ist ein leerer Sichtbarkeitsbereich, d.h. $\mathcal{S}(p) \cap \{p_j \mid j\} = \emptyset$. Im Gegensatz zu der Situation oben ist diese von außen gegeben, d.h. bei fest gegebener Kanteninformation läßt sich diese Situation nur durch Anpassung des Samples (der Abtastwerte) beheben. Genau diesen Weg wollen wir hier nicht gehen; er bleibt natürlich als zusätzliche Option offen! Die Situation $\mathcal{S}(p) \cap \{p_j \mid j\} = \emptyset$ kann unterschiedlich bedingt sein, und diese verschiedenen Randbedingungen führen zu unterschiedlichen Strategien.

Zum einen kann das Pixel p in einem Gebiet liegen, in welchem keine Abtaststelle liegt. Beispiele hierfür sind zwei der langen Rechtecke, das Auge des Fisches und die Flecken des Embryo.

Zum anderen kann das Pixel p in einem Gebiet mit Abtaststelle liegen, aber durch die Gestalt des Gebiets liegt die Abtaststelle außerhalb des Sichtbarkeitsbereich $\mathcal{S}(p)$. Beispiele hierfür sind die Zwischenräume zwischen den langen Rechtecken, sowie Teile der Dreieck- und Rechteckkonfiguration.

Damit ist ein Großteil der schwarzen Stellen erklärt. Es tauchen aber auch vereinzelt schwarze Pixel in der Interpolation auf, deren Auftauchen wir jetzt noch verstehen wollen. Betrachten wir die mittelgraue Ellipse rechts des Knochens. In Abbildung 2.38 auf der vorherigen Seite sehen wir, daß dort für einige Stellen keine Interpolationswerte berechnet werden konnten (schwarze Pixel rechts und links). Ein Vergleich mit Abbildung 2.37 auf Seite 67 zeigt, daß diese Stellen im Hintergrundrechteck liegen. Betrachten wir jetzt für die rechten Pixel die in der Nähe liegenden Abtaststellen im Hintergrundrechteck, so finden wir, daß diese Abtaststellen aufgrund der Ellipsenkrümmung außerhalb des Sichtbarkeitsbereichs liegen.

Ein etwas anderes Bild ergibt sich für die linke Seite. Betrachten wir für diese Seite die Abtaststellen im Hintergrundrechteck, so finden wir im Norden und Süden auch Abtaststellen, die aufgrund der Ellipsenkrümmung nicht im Sichtbarkeitsbereich liegen. Im Nordwesten finden sich allerdings Abtaststellen innerhalb des Sichtbarkeitsbereichs; diese sind aber zu weit entfernt (Interpolation nur mit Abtaststellen innerhalb $U_{17}(p)$!).

Bevor wir die für das Gesamtverfahren verwendete Heuristik vorstellen, wollen wir noch kurz einen Effekt ansprechen, der auf der Erzeugung zusätzlicher Kanten (im Sinne von Plus-Minus-Übergängen) bei der Kantendetektion beruht (wie schon weiter oben besprochen). Im Norden der Region R1 gibt es in der Abbildung 2.38 auf der vorherigen Seite eine vier Pixel lange waagerechte Linie, welche vollständig (!) im hellgrauen Hintergrund liegt, und die nächste Abtaststelle im hellgrauen Hintergrund ist nur zwei

Pixel entfernt! Betrachten wir den entsprechenden Bereich in der Kanteninformation (in Abbildung 2.29 oder in Abbildung 2.32 ganz links oben), so stellen wir fest, daß der weiß-hellgrau-dunkelgrau-Übergang, da er räumlich klein ist, in der Kanteninformation eine neue Kante erzeugt, der keine Helligkeitskante im Original entspricht. Die Abtaststelle liegt nun auf der einen Seite der künstlich erzeugten Kante und die vier schwarzen Pixel auf der anderen. Mit solchen Effekten müssen wir auch in natürlichen Bildern rechnen. Nur können wir sie dort nicht so schön isolieren!

Nachdem wir jetzt die verschiedenen äußeren Umstände diskutiert haben, die dazu führen können, daß im ersten Interpolationsschritt an einem Pixel p kein Interpolationswert berechnet werden kann, wollen wir jetzt das Gesamtverfahren vorstellen.

Das Verfahren ist in vier Schritte oder Phasen eingeteilt. Der erste Schritt, die *I-Phase*, ist die Anwendung der Interpolationsformel mit der Umgebung $U_r(p)$. D.h. im ersten Schritt wird für jedes Pixel p, das noch keinen Wert hat, die folgende Formel $S_{k,U_r}(p)$ ausgewertet:

$$S_{k,U_r}(p) = \left(\sum_{q \in \mathcal{W} \cap U_r(p) \cap \mathcal{S}(p)} \frac{f(q)}{||p-q||^2} \right) \Bigg/ \left(\sum_{q \in \mathcal{W} \cap U_r(p) \cap \mathcal{S}(p)} \frac{1}{||p-q||^2} \right). \tag{2.3}$$

Dabei ist $\mathcal{W}$ die Menge der Pixel $q \in \mathcal{R}$ für die ein Wert $f(q)$ vorliegt und der für die Interpolation genutzt werden soll. Im ersten Schritt umfaßt $\mathcal{W}$ nur die Abtaststellen p_i. Diesen ersten Schritt haben wir schon ausführlich diskutiert. Für die Wahl der Umgebungsgröße r hat sich bewährt, den um eins vergrößerten Gitterabstand der Abtaststellen zu benutzen.

Für den zweiten Schritt, die *L-Phase*, geben wir uns eine kleine natürliche Zahl n vor und wiederholen für die Pixel p ohne bisher berechneten Wert die Interpolation mit wachsender Umgebungsgröße. Es werden nacheinander $S_{k,U_{2r}}(p)$, $S_{k,U_{3r}}(p)$ bis $S_{k,U_{nr}}(p)$ ausgewertet und zwar so lange, bis eine Formel einen Wert liefert oder die vorgegebene Umgebungsgröße nr erreicht ist. Dieser Schritt kompensiert den Fehler, der durch Einführung von U_r gemacht wird, fast vollständig. Günstige Wahlen für n sind $n = 2$ oder $n = 3$. Wir wählen $n = 2$. Größere n führen schnell dazu, daß der Performancegewinn durch die Einführung von U_r wieder egalisiert wird, und zu große n führen durch Mehrfachberechnung an Stellen p mit $\mathcal{W} \cap \mathcal{S}(p) = \emptyset$ (keine Werte im Sichtbarkeitsbereich) schnell zu längeren Laufzeiten als eine Auswertung von S_k für alle Pixel ohne die U_r-Einschränkung.

Der dritte Schritt, die *G-Phase*, ist dadurch gekennzeichnet, daß die schon berechneten Grauwerte als neue Abtastwerte betrachtet werden und die Interpolation mit verkleinerter Umgebung U_g neu aufgesetzt wird. Dieses wird insgesamt m-mal wiederholt. In jedem dieser Durchläufe wird dabei die Men-

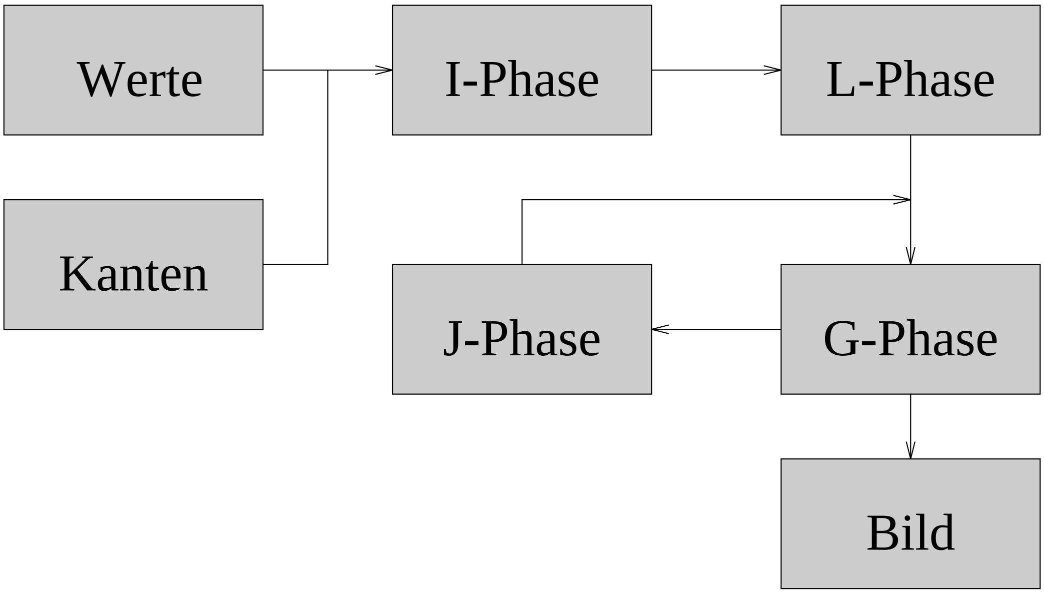


Abbildung 2.39: Die einzelnen Phasen der Interpolation

ge $\mathcal{W}$ größer. Dieses Vorgehen hat zur Folge, daß die schon berechneten Grauwerte sich langsam in Gebiete ausdehnen, die von den jetzigen Grauwerten aus erreichbar sind ohne Kanten zu passieren. D.h. die Grauwerte können um Kanten herumfließen, aber nicht über Kanten hinweg. In unserer Implementation wählen wir standardmäßig $g = \frac{r}{4}$ und $m = 16$, so daß die Grauwerte über eine maximale Distanz von $4r$ (im TESTMUSTER sind dies 68 Pixel) wachsen können.

Im vierten Schritt, der *J-Phase*, benutzen wir erstmalig die Vorzeichen der Kanteninfomation. Jeder Grauwert, der an einer Kante liegt, wird auf die andere Seite der Kante transportiert und um eine global gewählte Sprunghöhe s erhöht bzw. erniedrigt, abhängig davon, ob der Ausgangswert auf der Minus- bzw. Plus-Seite der Kante lag. Dieses ist der Mechnismus um Grauwerte in Gebiete zu transportieren, die anfänglich keine Abtastwerte erhalten haben.

Danach schließen sich im Wechsel weitere G- und J-Phasen an, bis alle Pixel des Bildes einen Grauwert erhalten haben. Daß auf diese Weise wirklich jedes Pixel einen Wert erhält und das Verfahren also abbricht, ist klar. Denn wir iterieren einen Prozeß, der die Wegzusammenhangskomponenten füllt, im Wechsel mit einem, der diese Wegzusammenhangskomponenten überwinden kann.

Die folgenden Abbildungen zeigen unser TESTMUSTER nach Durchlaufen der oben genannten Phasen der Interpolationsheuristik. Die Abbildung 2.40 auf Seite 73 zeigt noch einmal das schon weiter oben gezeigte TESTMUSTER nach dem ersten Interpolationsschritt. Die schwarzen Flächen sind die Stellen, an denen (noch) keine Werte berechnet werden konnten. Der nächste Schritt des Verfahrens wird jetzt nur noch für diese schwarzen Stellen durchgeführt. Und das unter Beibehaltung der ursprünglichen Abtastwerte (siehe Abbildung 2.37 auf Seite 67).

In der Abbildung 2.41 auf der nächsten Seite sehen wir das TESTMUSTER nach der L-Phase der Interpolationsheuristik. Ein Vergleich mit Abbildung 2.40 zeigt, daß viele Effekte der Reichweitenbeschränkung durch Einführung der Umgebung U_r kompensiert werden konnten. Die Abbildung 2.41 zeigt aber auch, daß dieses nicht vollständig erreicht wurde. Die Rückenpartie des Embryos ist ein gutes Beispiel dafür. Obwohl eine Verlängerung der L-Phase, also eine Wahl von $n \geq 3$, eine weitere Minderung der Anzahl schwarzer Pixel bringen würde, ist dieses nicht zu empfehlen. Wie wir schon weiter vorne angemerkt haben, verlängert sich dadurch die Rechenzeit unnötig stark. Der Effekt im Bild ist außerdem in der Regel nicht besonders groß und wird in der nachfolgenden G-Phase auch erreicht. Wir wollten aber auch nicht völlig auf die L-Phase verzichten, da sie die Anzahl verlässlicher Ausgangswerte für die G-Phase erhöht.

Während in der I- und L-Phase die neuen Grauwerte noch direkt aus den anfänglichen Abtastwerten berechnet werden, werden in den nachfolgenden Phasen alle neuen Grauwerte aus allen schon berechneten Werten berechnet. D.h. in den G- und J-Phasen wird die Menge $\mathcal{W}$ in der Formel 2.3 immer durch alle schon berechneten Werte ersetzt. Dieses führt in der G-Phase dazu, daß die vorhandenen Grauwerte harmonisch in die schwarzen Flächen hineinwachsen (‚growing'). Die Abbildung 2.42 auf Seite 74 zeigt einige Zwischenschritte dieser 16-stufigen Iteration, während die Abbildung 2.43 auf Seite 75 das TESTMUSTER am Ende der ersten G-Phase zeigt. Alle auf der Abbildung 2.43 noch schwarzen Flächen sind Wegzusammenhangskomponenten, in denen zu Anfang keine Abtastwerte lagen. Die einzige Ausnahme bilden einige Stellen in der Dreieckskonfiguration.

In der nachfolgenden J-Phase (‚jump') wird allen schwarzen Pixeln, die an einer Kante liegen, auf deren anderer Seite schon Werte berechnet wurden, nun auch ein Wert zugewiesen. Dieses geschieht unter Berücksichtigung der in der Kanteninformation enthaltenen Kenntnis der helleren bzw. dunkleren Seite der Kante. Dadurch, daß wir eine fiktive Sprunghöhe global für alle Kanten festsetzen, erhalten wir natürlich nicht die Grauwerte des Originals. Aber durch diese Methode wird wenigstens der relative Helligkeitsunterschied zwischen den an den Kanten zusammenstoßenden Grauwertflächen erhalten. Die Abbildung 2.44 auf Seite 75 zeigt das TESTMUSTER nach der ersten J- und einer der zweiten G-Phase, in welcher sich die Grauwerte in den Komponenten wieder ausgedehnt haben.

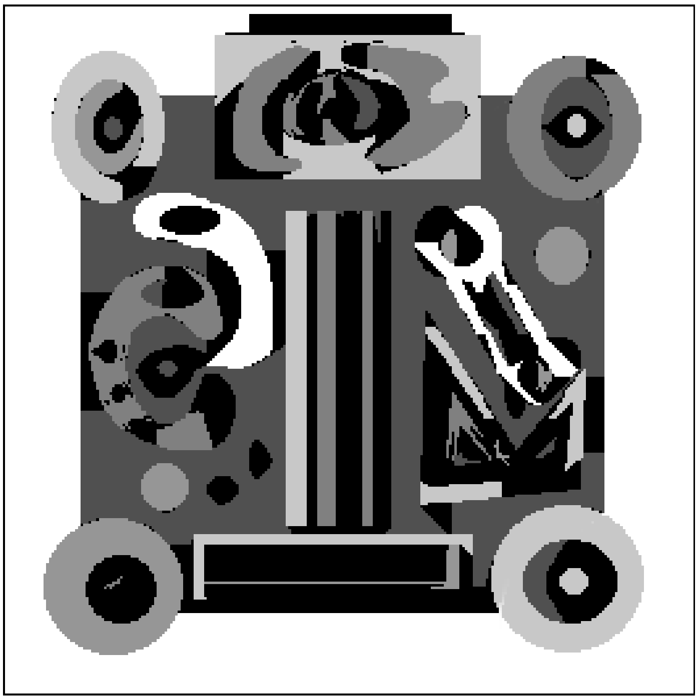


Abbildung 2.40: TESTMUSTER nach I-Phase.

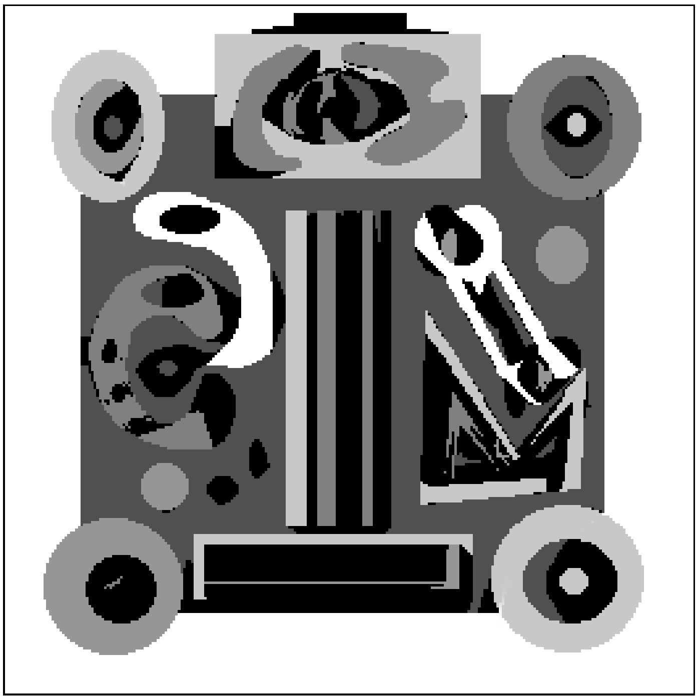


Abbildung 2.41: TESTMUSTER nach L-Phase.

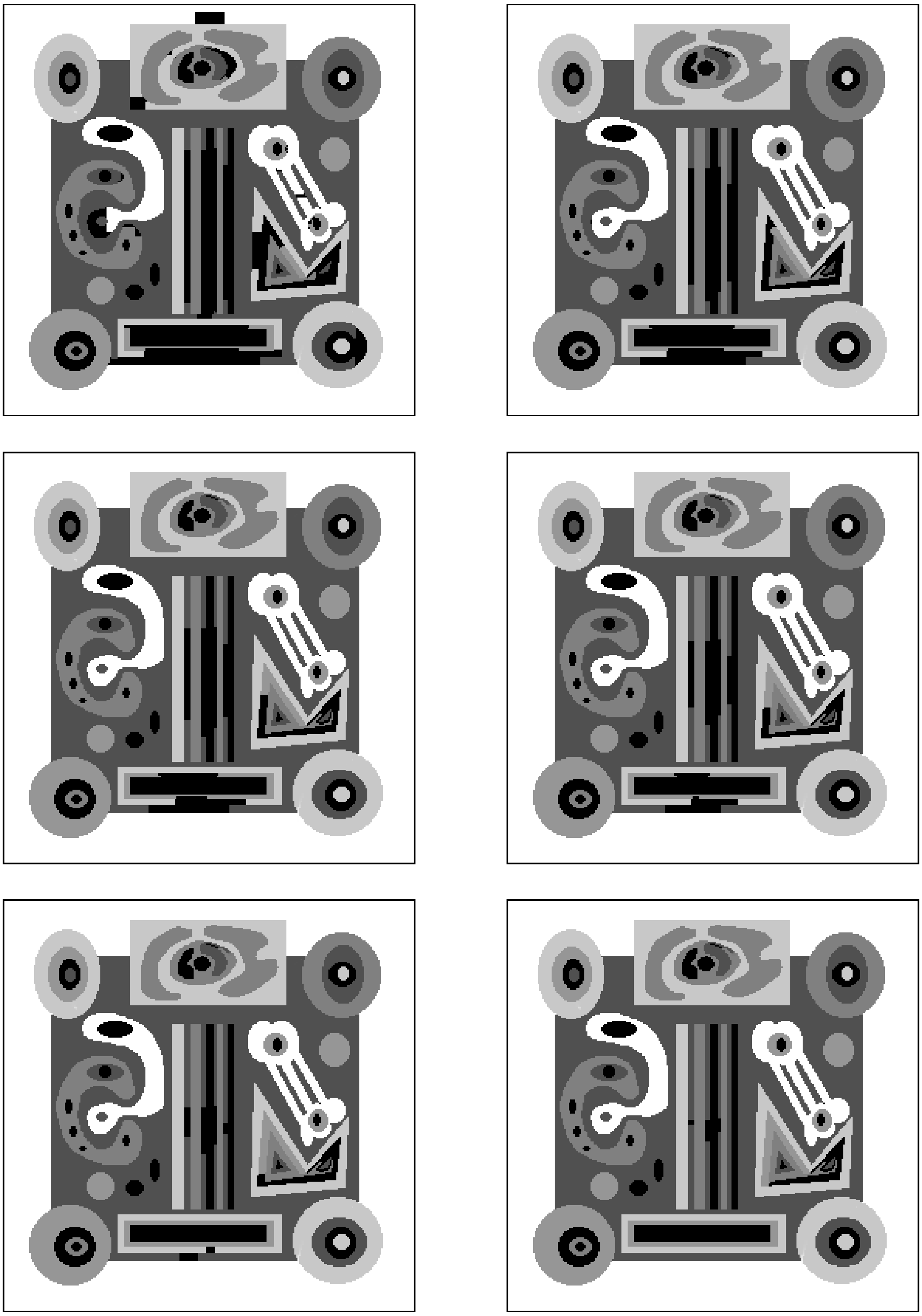


Abbildung 2.42: Verschiedene Zwischenbilder während der Iteration in der 1. G-Phase. (Iterationstufen 2, 5, 7, 9, 12 und 14)

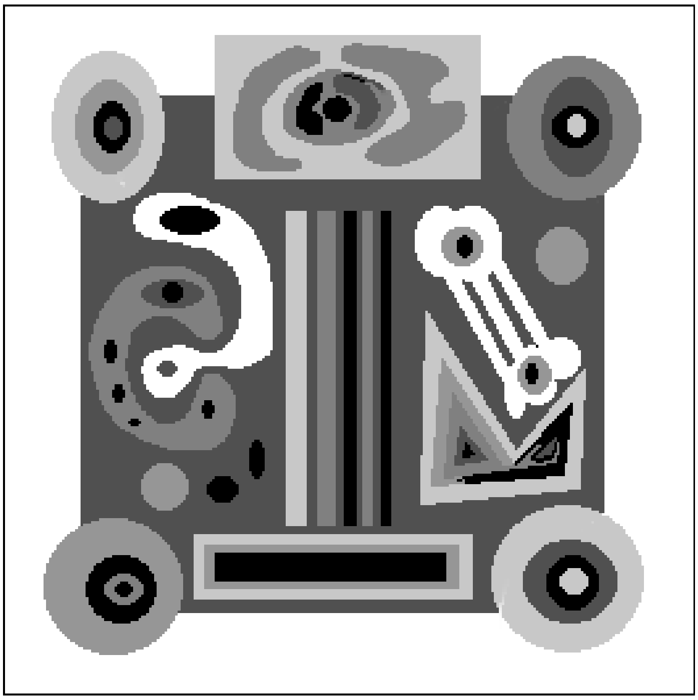


Abbildung 2.43: TESTMUSTER nach 1. G-Phase.

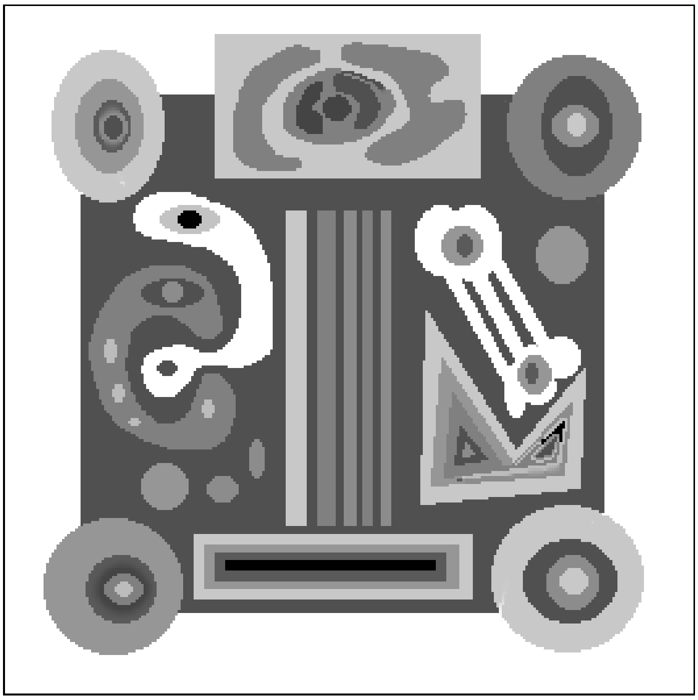


Abbildung 2.44: TESTMUSTER nach 2. G-Phase.

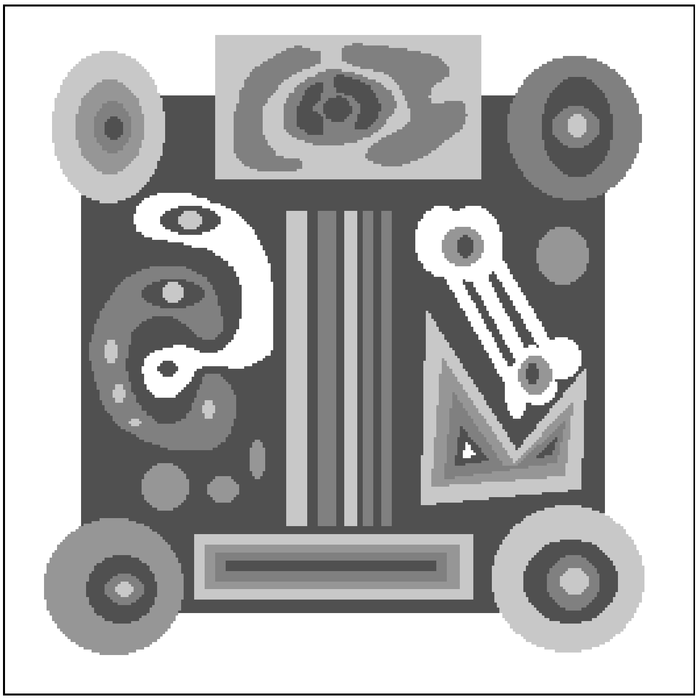


Abbildung 2.45: Testmuster Original.

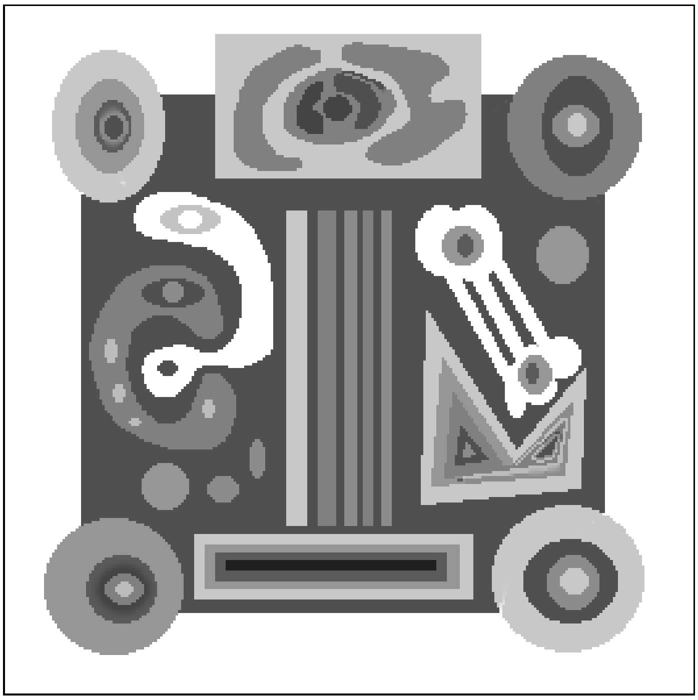


Abbildung 2.46: Testmuster Rekonstruktion.

Auch jetzt noch sind einige Stellen ohne Werte, insbesondere Teile der Dreieckskonfiguration. Nach einer weiteren J- und G-Phase hat aber jedes Pixel einen Wert zugewiesen bekommen und das auf diese Weise ‚interpolierte' Bild weicht nur an einigen Stellen, und an diesen nicht eklatant, vom Original ab.

Nach der nun erfolgten detaillierten Darstellung des benutzten *kantenerhaltenen Interpolationsverfahren* an einem hochgradig artifiziellen TESTMUSTER, wollen wir dieses Verfahren auf natürliche Bilder anwenden. Durch diese Anwendung demonstrieren wir zum einen ein distanzgewichtetes Interpolationsverfahren, welches in der Lage ist, zusätzliche Strukturdaten (hier: Kanteninformationen) zu berücksichtigen. Zum anderen zeigen wir, daß die in Kapitel 1 entwickelte Methode wirklich Daten extrahiert, die zu einer Abstraktion des bearbeiteten Bildes führen, die aber noch für den Menschen wesentliche Aspekte des ursprünglichen Bildes beinhaltet. Diese Extraktion liefert Informationen, auf denen weitere Verfahren des *Maschinellen Sehens* aufsetzen können.

Die Interpolationen von Daten, die aus Originalbildern gewonnen wurden finden sich im Anhang A.

Kapitel 3

Bildkompression durch Abstraktion

Wir haben in dem letzten Kapitel ein Verfahren vorgestellt, mit welchem es möglich ist, aus der Kanteninformation, zusammen mit der Grauwertinformation, wieder ein Bild zu rekonstruieren. Dieses rekonstruierte Bild enthält nicht mehr alle Details des Originalbildes, aber alle für eine spätere Verarbeitung ‚wichtigen' Bilddetails. Dabei wurde ‚wichtig' nur implizit durch die Parameterwahl des Abstraktors A_σ definiert. Wir bezeichneten im ersten Kapitel den gesamten Vorgang der Gewinnung der Kanteninformation und der Grauwertinformation als Abstraktion, da wir bei diesem Vorgang bewußt auf Informationen verzichteten. Im zweiten Kapitel (zusammen mit Anhang A) haben wir demonstriert, daß in den so erhaltenen Kanten- und Grauwertinformationen noch genug Bildspezifika des Ausgangsbilds enthalten sind, um ein visuell ansprechendes Bild rekonstruieren zu können.

Der Erhalt großer oder größerer Bildstrukturen ist aber nur der eine Aspekt des Begriffs Abstraktion. Der andere Aspekt ist der Verlust oder bewußte Verzicht auf Bilddetails, der sich auch in der Möglichkeit einer kompakten Speicherung ausdrücken sollte.

Ziel dieses Kapitels wird es sein, die aus der Abstraktion eines Ausgangsbildes erhaltene Grauwert- und Kanteninformation verlustfrei zu komprimieren. Zur Anwendung kommen dabei nur Standardverfahren, die an anderer Stelle schon ausführlich besprochen wurden.

Insgesamt ergibt sich das Bild aus Abbildung 3.1 auf der nächsten Seite. Fassen wir die Hintereinanderausführung von Abstraktion und verlustfreier Kompression bzw. Dekompression und Rekonstruktion zusammen, so können wir diese Hintereinanderausführung als ein verlustbehaftetes Bilddatenkompressionsverfahren betrachten. Auf dieser Ebene können wir unser Verfahren mit anderen verlustbehafteten Bilddatenkompressionsverfahren vergleichen.

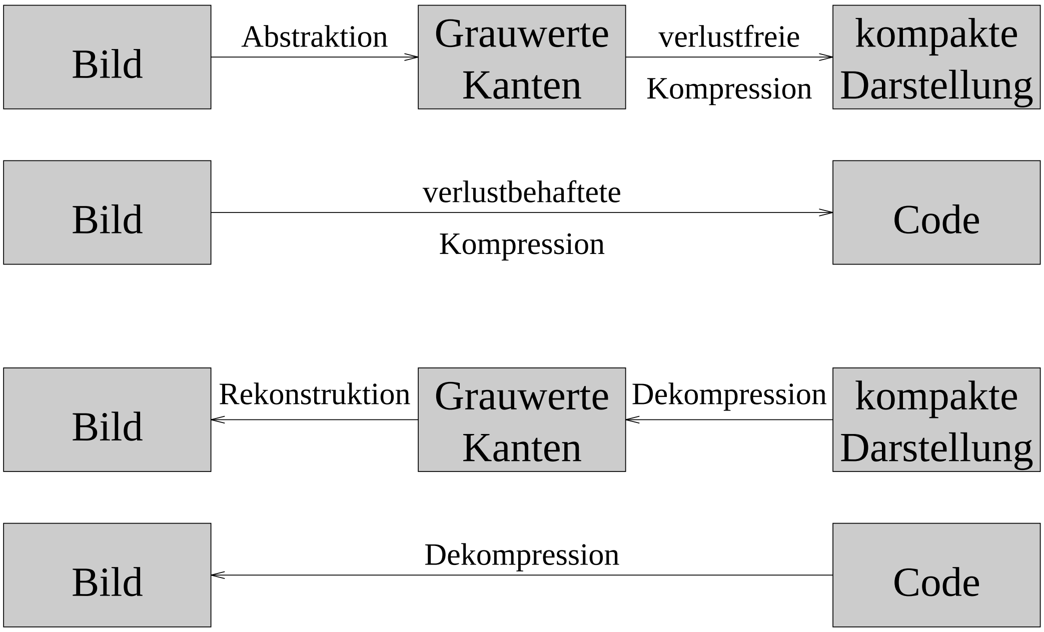


Abbildung 3.1: Hintereinanderschaltung unserer Verfahren ergibt ein verlustbehaftetes Bilddatenkompressionsverfahren.

Diese Vergleiche werden den letzten Teil des Kapitels bilden, während die ersten beiden Teile sich der verlustfreien Kompression der Grauwert- und Kanteninformation widmen.

3.1 Kompression der Grauwertinformation

Die in diesem Abschnitt verwendetete Grauwertinformation wurde aus Testbildern gewonnen, die mit dem Abstraktor A_σ vorverarbeitet wurden. Diese Testbilder sind zum Teil weit verbreitete Bilder (sie werden in der Tabelle 3.1 auf der nächsten Seite mit ‚standard' bezeichnet) und zum Teil Bilder der Arbeitsgruppe um Prof. Aurich (sie sind mit ‚local' bezeichnet). Bei den Bildern mit der Bezeichnung ‚local' handelt es sich ausschließlich um mit einer Videokamera aufgenommene Bilder. Sie sind so entstanden, wie in Kapitel 1 beschrieben. Diese Bilder bestehen aus nicht quadratischen Pixeln, weswegen sie in der Druckwiedergabe das Seitenverhältnis von 3:2 aufweisen. Die beiden Bilder mit der Bezeichnung ‚PhotoCD' stammen von einer ebensolchen, d.h. sie wurden mit einer Fotokamera aufgenommen und dann vom entwickelten Diapositiv eingescannt. Von der PhotoCD wurden sie in den Rechner eingelesen und als Grauwertbild (Y-Komponente des YUV-Modells) abgespeichert. Über die Testbilder mit der Bezeichnung ‚standard' ist nicht

Bild			Ursprung	A_σ
Nr.	Name	Größe in Pixel		$\sigma =$
1	Airplane	512×512	standard	0.5
2	Baboon	512×512	standard	1.5
3	Beton	512×512	local	1.5
4	Casino	512×512	PhotoCD	1
5	Collie	256×256	standard	1
6	Fanta	512×512	local	1
7	Gabi	512×512	local	1.5
8	Grundig	512×512	local	1.5
9	Halle	512×512	PhotoCD	1
10	IC	512×512	local	1.5
11	Lena	512×512	standard	1
12	Maus	512×512	local	1.5
13	Peppers	512×512	standard	1
14	Radio	256×256	local	1.5
15	Sailboat	512×512	standard	1
16	Schlumpf	512×512	local	0.5
17	Tastatur	512×512	local	1.5
18	Telefon	512×512	local	1
19	Tiffany	512×512	standard	1

Tabelle 3.1: Die verwendeten Testbilder.

viel bekannt, außer daß sie häufig in der Literatur als Vergleichsbilder benutzt werden. Eine Ausnahme ist das Testbild LENA. Bei diesem handelt es sich um einen mit einer Farbkamera aufgenommenen Druck (PLAYBOY November 1972). Von diesem Farbbild existieren zwei Grauwertversionen. Zum einen die Grünkomponente des RGB-Modells und zum anderen die Y-Komponente des YUV-Modells. Die Testbilder sind im Anhang A abgedruckt.

Um die Grauwertinformation zu bekommen, wurden die Bilder mit A_σ geglättet und dann mit Faktor 16 unterabgetastet (wie in Kapitel 1 beschrieben). Aus einem 512×512 (bzw. 256×256) Pixel großen Bild bekommen wir so 32×32 (bzw. 16×16) Byte Grauwertinformation. Der Parameter σ, mit welchem der Filter A_σ bei jedem einzelnen Bild benutzt wurde, ist der Tabelle 3.1 zu entnehmen, und er wurde dem Charakter des jeweiligen Bildes entsprechend gewählt (d.h. bei strukturreichen Bildern kleiner als bei strukturarmen). Da die so erhaltenen Grauwerte weniger korreliert sind, als das bei normalen Bilddaten zu erwarten ist, werden die Grauwerte als Textstring betrachtet und mit gängigen Verfahren zur Textkompression komprimiert.

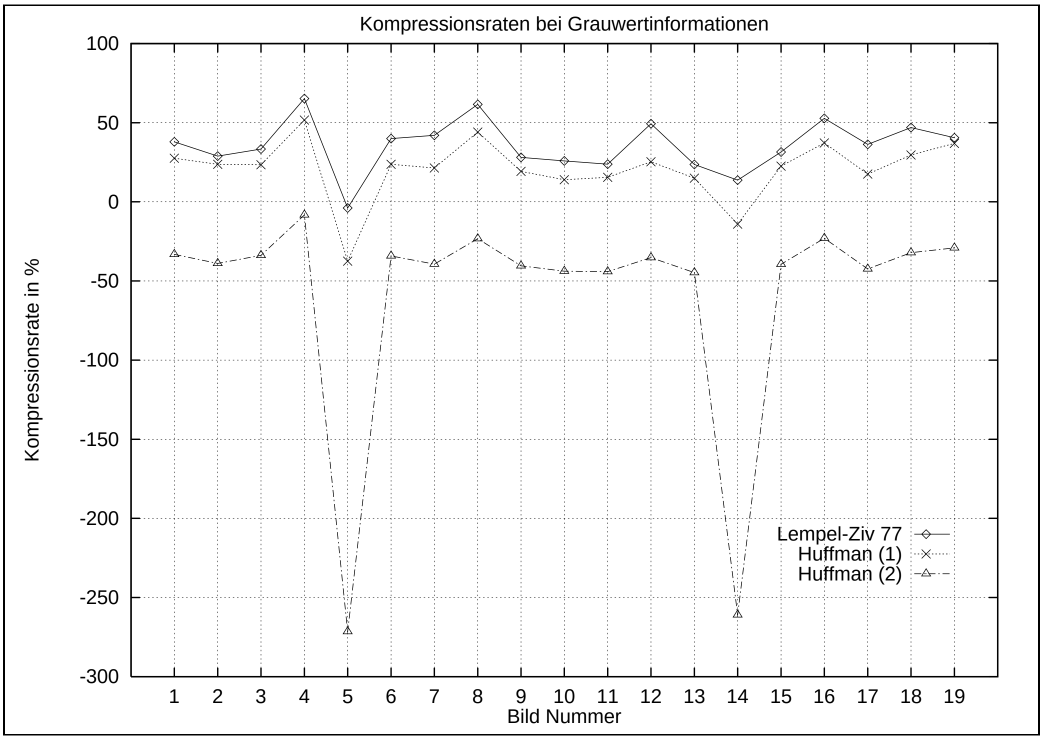


Abbildung 3.2: Kompressionraten bei der Grauwertinformation für LZ77- und statische Huffman Kompression.

Eine Serialisierung der Abtastwerte erhalten wir, indem wir diese zeilenweise abspeichern und dann das Resultat als (Byte-) Text betrachten.

Die hier zur Untersuchung herangezogenen Verfahren umfassen LZ77-, Huffman- und arithmetische Kompression. Es wurden nur Textkompressionsverfahren berücksichtigt, für die wir auch eine Implementation finden konnten. Alle Programme, die hier zur Kompression benutzt werden, sind entweder frei verfügbar oder von den Programmautoren zu Forschungszwecken zur Verfügung gestellt worden.

In der Abbildung 3.2 sind die Kompressionsraten über alle Testbilder für eine Implementation des Lempel-Ziv-Algorithmus [LZ77] und zwei unterschiedliche Implementationen des Huffman-Algoritmus aufgetragen.

Die Implementation des LZ77 stammt aus dem GNU-Projekt (gzip). Die Implementation ‚Huffman (1)' stammt von Mark Nelson (siehe [Ne92]) und ‚Huffman (2)' von Bill Demas.

Ist l die Länge der Ausgangsdatei (1024 byte für 512×512-Bilder) und ist c die Länge der komprimierten Datei, so ist die Kompressionsrate gegeben durch $1 - c/l$. Folglich bedeuten negative Kompressionsraten, daß die komprimierte Datei länger ist als die Ausgangsdatei. Das ist für alle drei Implementationen für Bild 5 der Fall. Bild 5 bildet zusammen mit Bild 14 eine Ausnahme unter den Testbildern, da beide Bilder nur 256×256 Pixel groß sind und die zu komprimierende Grauwertdatei mit 256 Byte schon relativ klein ist. Wie wir der Abbildung 3.2 auf der vorherigen Seite auch entnehmen können, leiden beide Huffman-Implementationen darunter, daß sie die Statistik mitübertragen müssen. ‚Huffman (1)' benutzt nur 256 Byte Statistik, während ‚Huffman (2)' 768 Byte Statistik benutzt (d.h. auch überträgt). Das ist auch der Grund, warum ‚Huffman (2)' durchgängig negative Kompressionsraten erzielt.

Unter bestimmten Bedingungen haben statische Huffmankodierungen gegenüber dynamischen (d.h. adaptiven) Vorteile, siehe dazu z.B. [MP85]. (Bei statischen Huffmankodierungen wird die relative Häufigkeit der Symbole vorab über die gesamte Datei ermittelt und dann mitübertragen, während bei der adpativen Variante die relative Häufigkeit der Symbole zu jedem Zeitpunkt aus den bislang übertragenen Symbolen bestimmt wird.) Aber in unserem Fall ist es notwendig, für die Kompression solch kleiner Dateien auf die Übertragung der Statistiken zu verzichten und statt dessen die Symbolhäufigkeiten adaptiv zu ermitteln. Diesen Weg gehen die Implementationen ‚adaptiver Huffman' von Nelson und ‚Splay Tree' von Kim Kokkonen und Sean O'Connor. Für Details des Algorithmus ‚Splay Tree' siehe [Jo88]. Eine Einführung in die Datenstruktur ‚Splay Tree' in Zusammenhang mit Suchbäumen findet sich auch bei Ottmann und Widmayer in [OW93]. Ein einführender Artikel zu adaptiver Huffmankodierung ist [CH84]. Für Details des Algorithmus ‚adaptiver Huffman' siehe [Ne92].

Die Abbildung 3.3 auf der nächsten Seite zeigt die Kompressionsraten der adaptiven Huffman-Implementation und der Splay-Tree-Implementation, während Abbildung 3.4 Kompressionsraten für eine arithmetische Kodierung mit verschiedenen (statistischen) Modellen zeigt. Das Programm zur arithmetischen Kodierung 1. und 2. Ordnung wurde von Nelson geschrieben (vgl. [Ne92]). Eine einführende Arbeit in das Thema ‚arithmetische Kodierung' ist [WNC87], von diesen Autoren stammt auch das Programm ‚CACM87'. Für weitere Literatur siehe das ausführliche Literaturverzeichnis von [HV92].

Während der Code von Witten, Neal und Cleary (CACM87) nur ein Modell nullter Ordnung benutzt, kann der Code von Nelson statistische Modelle unterschiedlicher Ordnung benutzen. Zum Problem der Kontexmodellierung siehe [HL92].

Ein Modell nullter Ordnung achtet bei der Kodierung eines Zeichens nicht auf den Kontext (die anderen Zeichen vorher). Im Gegensatz dazu bestimmt ein Modell n-ter Ordnung die Wahrscheinlichkeitsverteilung für das momentan zu kodierende Zeichen aus den zuletzt gesehenen n Zeichen. So ist z.B.

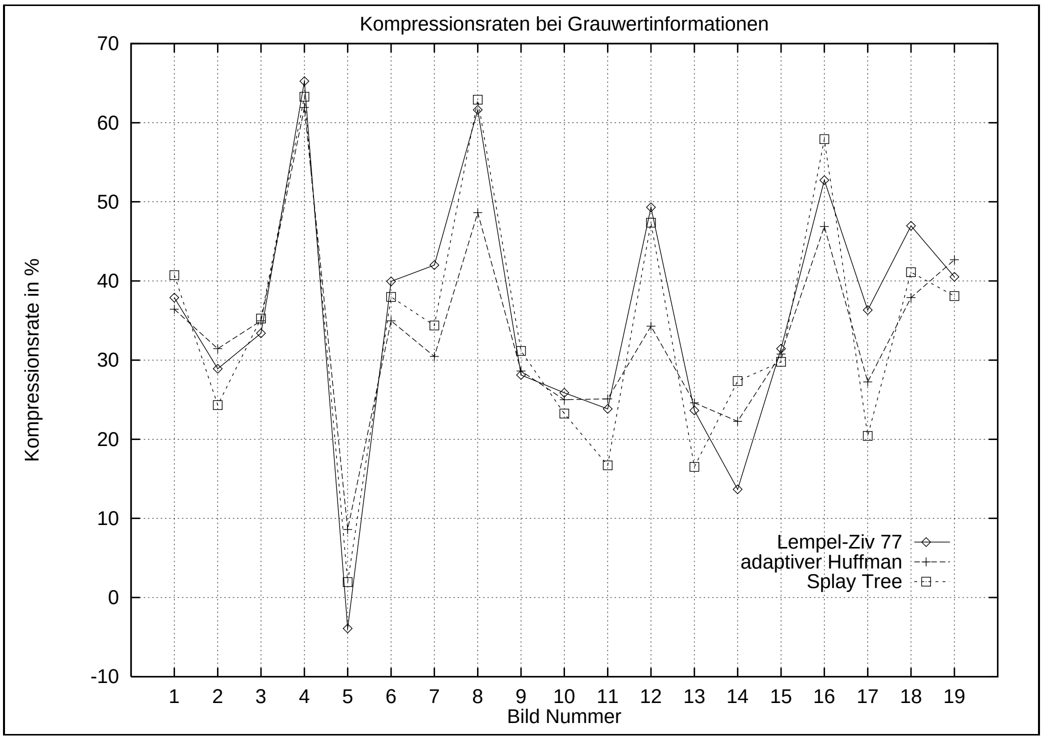


Abbildung 3.3: Kompressionsraten bei der Grauwertinformation für adaptive Huffman-Kompression.

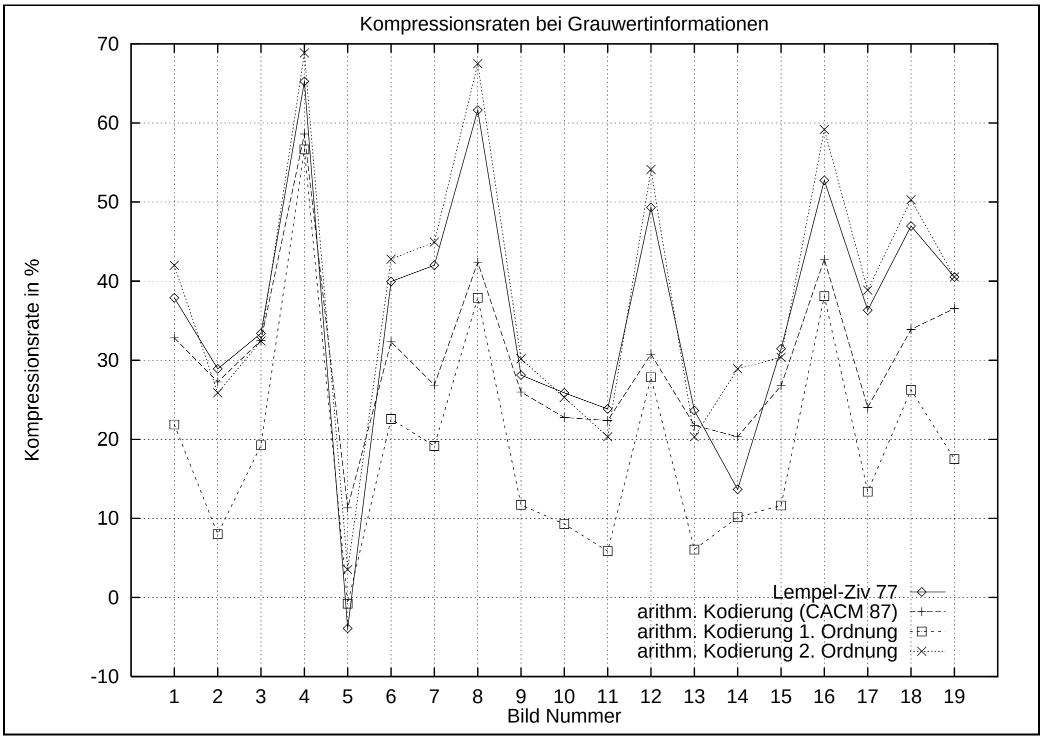


Abbildung 3.4: Kompressionsraten bei der Grauwertinformation für arithmetische Kodierungen mit unterschiedlichen Modellen.

die Wahrscheinlichkeit dafür, daß das nächste zu kodierende Zeichen ein ,u' ist, signifikant höher, wenn das letzte Zeichen ein ,q' war (normalen Text vorausgesetzt).

Ein genauer Vergleich der Abbildungen 3.2, 3.3 und 3.4 zeigt, daß es am günstigsten ist, zur Kompression der Grauwertdaten die ,arithmetische Kodierung zweiter Ordnung' von Nelson zu benutzen. Die hier nicht angegebenen Kompressionsraten für die arithmetische Kodierung mit Modellen höherer als zweiter Ordnung sind wieder kleiner als die der Kodierung zweiter Ordnung. Das liegt daran, daß drei oder mehr aufeinander folgende Grauwerte nicht mehr stark korreliert sind.

3.2 Kompression der Kanteninformation

War es bei der Grauwertinformation noch einigermaßen naheliegend, die Grauwertinformation als Textstring zu betrachten und dann den Textstring zu komprimieren, ist dieses bei der Kanteninformation nicht so naheliegend. Denn die Kanteninformation enthält die Information über die Kantenverläufe in unserem Bild, also eine zweidimensionale Information, die durch einen einfachen Durchlauf zeilenweise nicht adäquat serialisiert werden kann. Wir werden im folgenden eine Serialisierungsmethode vorstellen, mit welcher wir aus der trinären Kanteninformation einen Textstring erzeugen können. Unter Berücksichtigung der speziellen Struktur des Textstrings werden wir dann ein Kompressionsverfahren für diesen String auswählen.

3.2.1 Serialisierung mittels Quadtreedurchlauf

Der Quadtree ist eine in der Bildverarbeitung weit verbreitete Baumstruktur zur Repräsentation von (quadratischen) Bildern der Seitenlänge 2^k. Jeder Knoten des Quadtrees, der kein Blatt ist, besitzt genau vier Söhne. Die Blätter des Quadtrees repräsentieren (bei uns) einfarbige Blöcke des Bildes der Größe $2^{k-s} \times 2^{k-s}$ (s ist die Stufe des entsprechenden Knotens; die Wurzel besitzt Stufe 0). Auf eine formale Definition wollen wir an dieser Stelle verzichten. Für eine Übersicht über die Verwendung von Quadtrees verweisen wir auf Samet [Sa84].

Wir wollen uns an dieser Stelle mit einem Beispiel begnügen. An diesem Beispiel werden wir sehen, wie wir durch rekursive Unterteilung von nicht einfarbigen Blöcken unserer Kanteninformation einen Quadtree aufbauen können, der genau die Kanteninformation repräsentiert.

Wir betrachten dazu die Abbildung 3.5. Das gesamte 8×8 Bild (Abbildung 3.5(a)) wird durch die Wurzel des Baumes aus Abbildung 3.6 re-

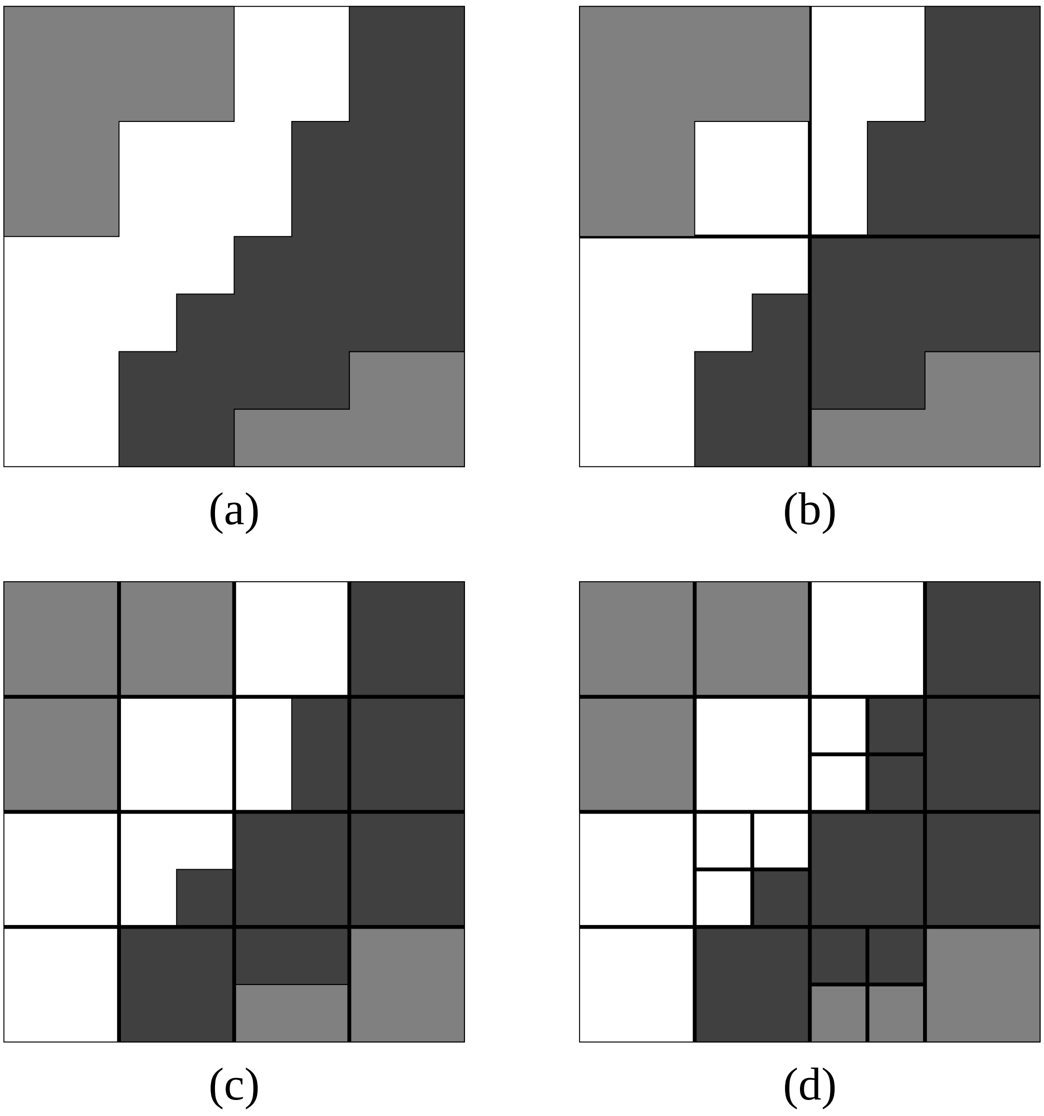


Abbildung 3.5: Ein 8×8 Beispiel mit den entsprechenden Quadtreeunterteilungen.

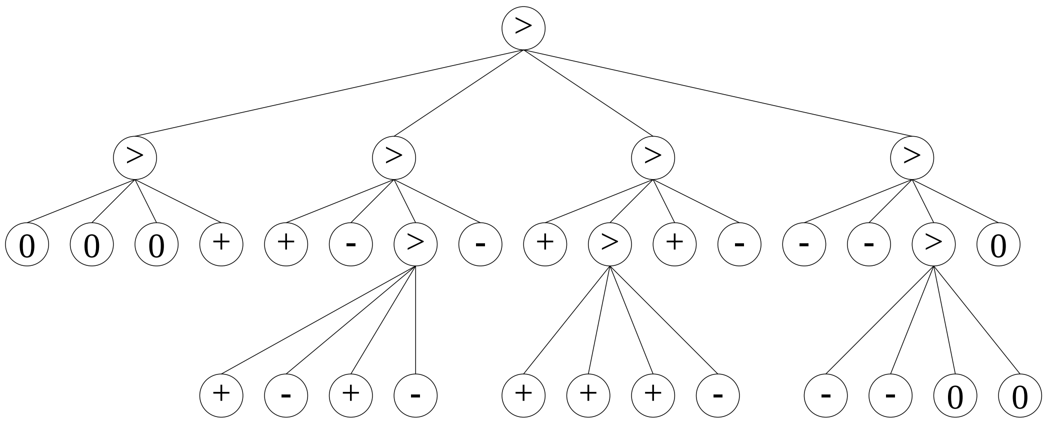


Abbildung 3.6: Der zum Beispiel aus Abbildung 3.5 gehörende Quadtree.

präsentiert. Das Zeichen `>` in der Wurzel zeigt an, daß der 8×8 Block nicht einfarbig ist und weiter unterteilt werden muß. Er wird in vier 4×4 Blöcke unterteilt (Abbildung 3.5(b)), welche durch die vier Söhne der Wurzel des Quadtrees repräsentiert werden. Die Söhne besitzen eine fest definierte Reihenfolge; bei uns entsprechen die Söhne (von links nach rechts betrachtet) den Teilblöcken in NW-, NO-, SW- und SO-Richtung (Kompaßangabe).

Da alle vier 4×4 Blöcke nicht einfarbig sind, bekommen alle vier Söhne der Wurzel das Symbol `>` zugeordnet. Im zweiten Unterteilungsschritt (vgl. Abbildung 3.5(c)) bekommen die Knoten des Quadtrees, die zu einfarbigen Blöcken gehören, das entsprechende Symbol (`+`, `-` oder `0`) zugeordnet. Die restlichen drei 2×2 Blöcke werden ein weiteres Mal unterteilt (Abbildung 3.5(d)). Auf diese Weise erhalten wir den Quadtree aus Abbildung 3.6.

Wenn wir in der oben festgelegten Reihenfolge der Knoten des Quadtrees einen Preorderdurchlauf durch den Baum machen, so liefert uns dieses ein Wort (Textstring) über den Symbolen {`+ - 0 >`}. In diesem Beispiel ist das der String:

```
>>000+>+->+-+-->+>+++-+->-->--000
```

Es ist klar, daß sich, bei vorgegebener Durchlaufreihenfolge der Söhne eines inneren Knotens, die Kanteninformation und der String eineindeutig entsprechen. Aus diesem Textstring läßt sich die Kanteninformation zurückgewinnen, also ist keine Information verlorengegangen. Wir könnten jetzt dazu übergehen, die schon vorher benutzten verlustfreien Textkompressionsmethoden auf diesen Text anzuwenden. Das werden wir jetzt aber noch nicht tun. Statt dessen gehen wir noch einmal zu unserer Kanteninformation zurück.

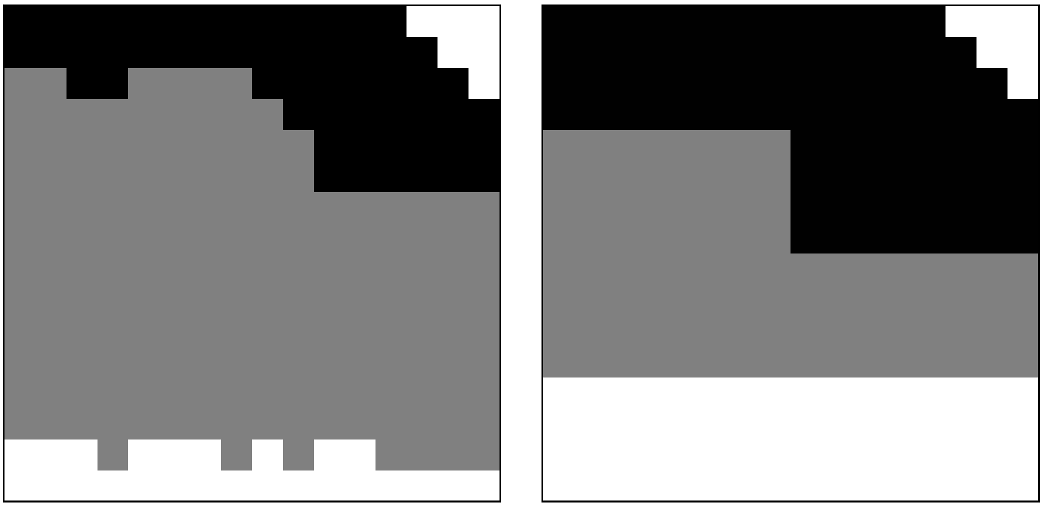


Abbildung 3.7: Beispiel für die Wirkungsweise des Quadtreegrenzen-Ausgleichs (links: vorher; rechts: hinterher).

3.2.2 Quadtreegrenzen-Ausgleich

Betrachten wir unsere Kanteninformation noch einmal genauer. Die für die Bildrekonstruktion wesentliche Information ist die genaue Lokalisierung der Plus-Minus-Übergänge, denn diese repräsentieren Kanten im Bild. Diese Plus-Minus-Übergänge werden gebildet von Plus-Bereichen und Minus-Bereichen. Die Null-Bereiche erfüllen bei dieser Markierung der Kanten durch Übergänge nur eine Pufferfunktion. Sie ermöglichen den Übergang von Plus nach Minus ohne dadurch eine Kante zu markieren.

Wir sehen also, daß die für die Rekonstruktion wesentliche Information in der genauen Lokalisierung von Plus-Minus-Übergängen besteht. Die andere wesentliche Information besteht in der Tatsache, daß bestimmte Bereiche nicht zusammenstoßen. Die wichtige Funktion der Null-Bereiche ist ihre Trennfunktion. Die genaue Lokalisierung der Null-Plus- und Null-Minus-Übergänge ist unbedeutend. Aber genau diese unbedeutenden Übergangslokalisierungen haben wir in dem oben erzeugten Textstring mitkodiert.

Die Länge des Textstrings läßt sich signifikant vermindern, wenn wir es erlauben, daß die Übergänge zu den Null-Bereichen auf Blockgrenzen liegen, die zu Knoten möglichst kleiner Stufe im Quadtree gehören. Wir erlauben damit nur die Verschiebung der Übergänge zu den Null-Bereichen, immer unter der Nebenbedingung, daß keine neuen Plus-Minus-Übergänge entstehen und die genaue Lokalisierung der alten erhalten bleibt.

Die ursprüngliche Kanteninformation (als Abbildung betrachtet) wird dabei durch eine andere ersetzt, die aber von unserem Rekonstruktionsverfahren in identischer Weise interpretiert wird.

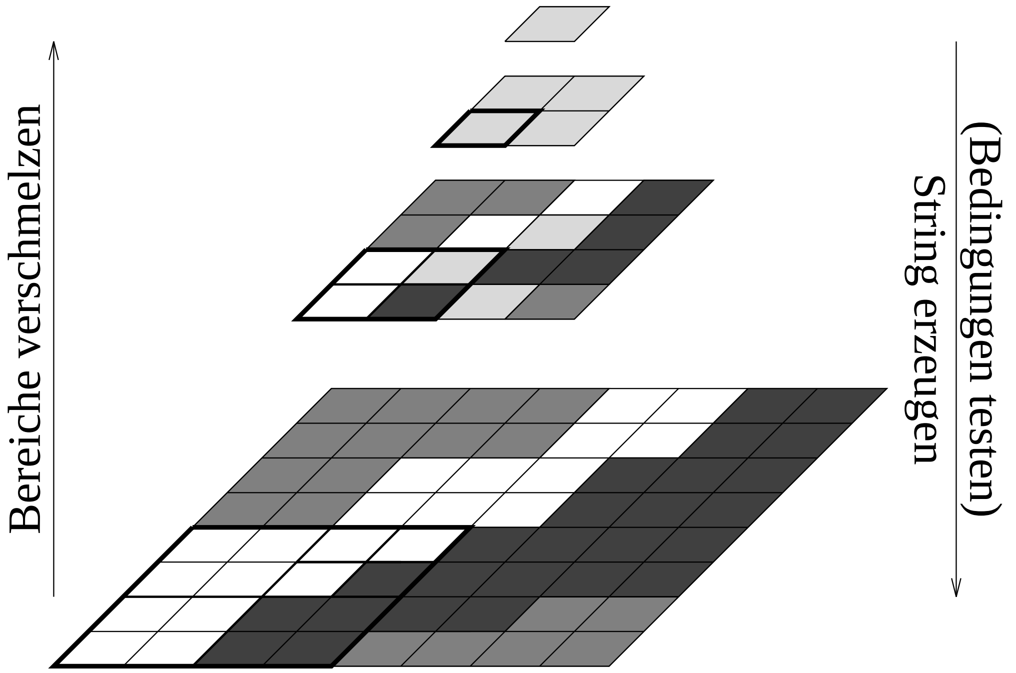


Abbildung 3.8: Das Beispiel aus Abbildung 3.5 als Bildpyramide.

Betrachten wir die ursprüngliche Kanteninformation in Abbildung 3.7 auf der vorherigen Seite (links), so benötigt der dazu gehörende Textstring 97 Symbole. Durch entsprechendes Ausrichten der nicht bedeutungstragenden Null-Übergänge (Abbildung 3.7 rechts) konnte der Textstring auf 33 Symbole verkürzt werden.

Wir haben einen Algorithmus entwickelt und implementiert, der einen solchen Quadtreegrenzen-Ausgleich unter den oben genannten Randbedingungen durchführt. Wie wir schon an Abbildung 3.7 sehen können, liefert unser Algorithmus keine minimalen Textstrings (in Sinne kürzester Länge). Dieses liegt daran, daß wir für den Quadtreegrenzen-Ausgleich eine schnelle heuristische Greedy-Strategie benutzen, die einfach zu implemetieren ist.

Der Algorithmus arbeitet nicht direkt auf dem Quadtree selbst, da in diesem Nachbarschaftsbeziehungen von Bildblöcken nicht direkt auswertbar sind, sondern er arbeitet auf einer *Bildpyramide*. Abbildung 3.8 zeigt das Beispiel aus Abbildung 3.5 als Bildpyramide. Die unterste Ebene dieser Pyramide wird von dem Bild selbst gebildet und ist mit den Symbolen `+`, `-` und `0` belegt (in der Abbildung 3.8 farblich kodiert). In jeder jeweils nächsthöheren Ebene sind vier Blöcke der aktuellen Ebene zusammengefaßt. Sind die Blöcke nicht alle mit dem selben Symbol markiert (d.h. sind sie nicht von gleicher Farbe), so wird ihnen das Symbol `>` (in Abbildung 3.8 die Farbe ‚Hellgrau') zugeordnet. Wir können unseren Textstring ebenfalls aus dieser

Bildpyramide bekommen. Dazu werden im ersten Durchgang von unten nach oben die Ebenen der Pyramide mit den vier Symbolen versehen. In einem Durchlauf von oben nach unten wird dann der Textstring erzeugt (analog zum Preorderdurchlauf im Quadtree).

Der Algorithmus zum Quadtreegrenzen-Ausgleich arbeitet vom Prinzip her analog. Nur benutzt er zusätzlich zwei neue Symbole, nämlich `p` und `m`. Die Interpretation der Symbole ist im Algorithmus die folgende:

`0 + -`: die vier ‚Sohnblöcke' sind einheitlich mit einem dieser Symbole markiert,

`p` bzw. `m`: die vier ‚Sohnblöcke' sind mit einer Kombination der Symbole `0`, `+` und p (bzw. `0`, `-` und m) markiert,

`>`: sonstige Kombination von Symbolen in den ‚Sohnblöcken'.

Mit den neuen Symbolen `p` bzw. `m` werden die Blöcke markiert, in welchen die Plus- bzw. Minus-Null-Übergänge in den Null-Bereich hinein ausgedehnt wurden. Bei der späteren Stringerzeugung werden wir die neuen Symbole `p` bzw. `m` wie die alten Symbole `+` bzw. `-` behandeln, und dadurch effektiv den entsprechenden Bereich in den Null-Bereich hinein ausdehnen.

Der Algorithmus zum Quadtreegrenzen-Ausgleich arbeitet in drei Phasen. In der ersten Phase wird analog der normalen Stringerzeugung die Pyramide ebenenweise mit den nun sechs Symbolen gefüllt. Die Regel dafür ist oben angegeben. In der zweiten Phase werden von oben nach unten die Übergangsbedingungen getestet, d.h. es wird ebenenweise für jeden der Knoten getestet, ob ein zusätzlicher Plus-Minus-Übergang erzeugt wurde. Ein zusätzlicher Übergang wurde erzeugt, wenn der getestete Knoten mit einem Knoten aus seiner 8-Nachbarschaft eine der drei Symbolkombinationen (`p`,`-`), (`p`,`m`) oder (`m`,`+`) erzeugt. In diesem Fall werden die neuen Symbole in dieser Kombination durch das Symbol `>` ersetzt. Dadurch wird der neu erzeugte Übergang wieder aufgelöst. In der dritten Phase wird in einem Preorderdurchlauf der Textstring erzeugt, dabei werden die neuen Symbole wie die alten Symbole `+` und `-` behandelt.

Die Abbildungen auf Seite 92 bis 94 zeigen die Resultate des Quadtreegrenzen-Ausgleichs bei den Kanteninformationen der Testbilder LENA (Testbild-Nr. 11), GRUNDIG (Testbild-Nr. 8) und SAILBOAT (Testbild-Nr. 15). Dabei konnten die Längen der Textstrings entsprechend Tabelle 3.2 vermindert werden.

Die Abbildung 3.9 zeigt eine Übersicht über die Verringerung der Längen der Textstrings durch den Quadtreegrenzen-Ausgleich für alle Testbilder.

Testbild		Länge String in Byte		Verringerung
Nr.	Name	vorher	nachher	in %
8	Grundig	23592	11708	50.4
11	Lena	81912	51908	36.6
15	Sailboat	95124	60808	36.1

Tabelle 3.2: Verringerung der Länge des Textstrings durch Quadtreegrenzen-Ausgleich.

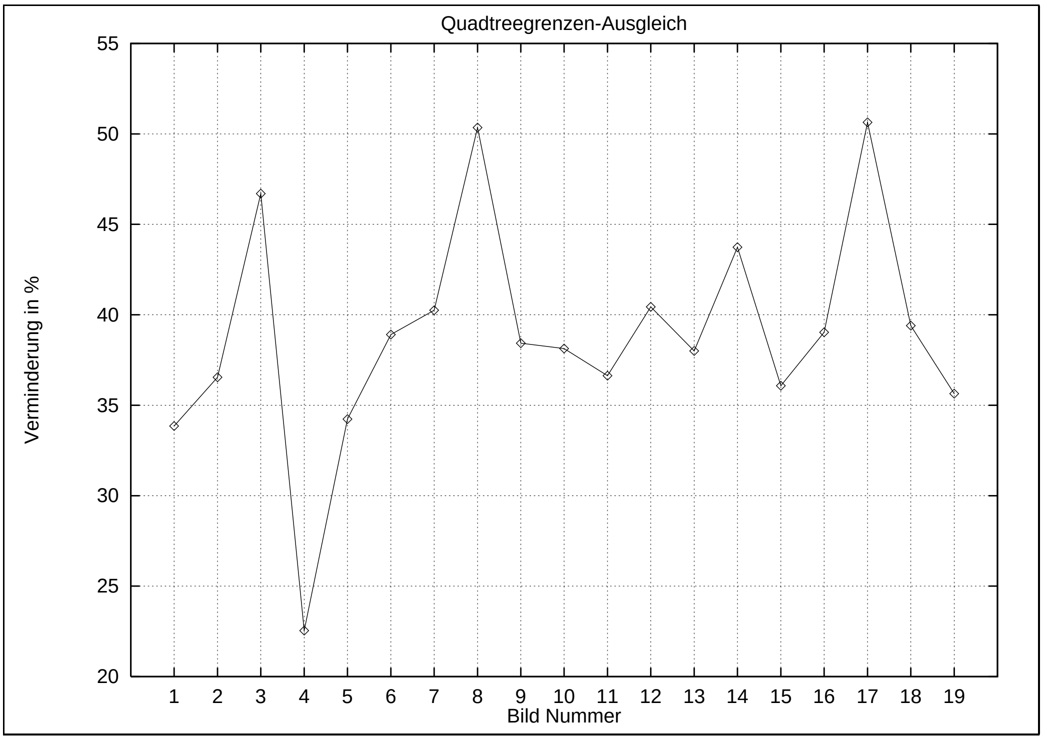


Abbildung 3.9: Verringerung der Länge des Textstrings durch Quadtreegrenzen-Ausgleich für alle Testbilder.

3.2.3 Kompression des Textstrings

Wie schon in Abschnitt 3.1 wollen wir auch in diesem Abschnitt zur Kompression nur Standardverfahren verwenden, für die wir auch Implementierungen vorweisen können. Aber anders als in Abschnitt 3.1 ist die Verwendung der vorhandenen Implementierungen in diesem Fall nicht so unproblematisch wie vorher. Denn alle Implementierungen gehen davon aus, daß sie Textstrings mit potentiell 256 verschiedenen Zeichen zu kodieren haben. Dieses ist in unserem Fall aber nicht so, da wir garantieren können, daß in unserem String

Abbildung 3.10: Kanteninformation des Testbilds LENA

Abbildung 3.11: Ausgeglichene Kanteninformation.

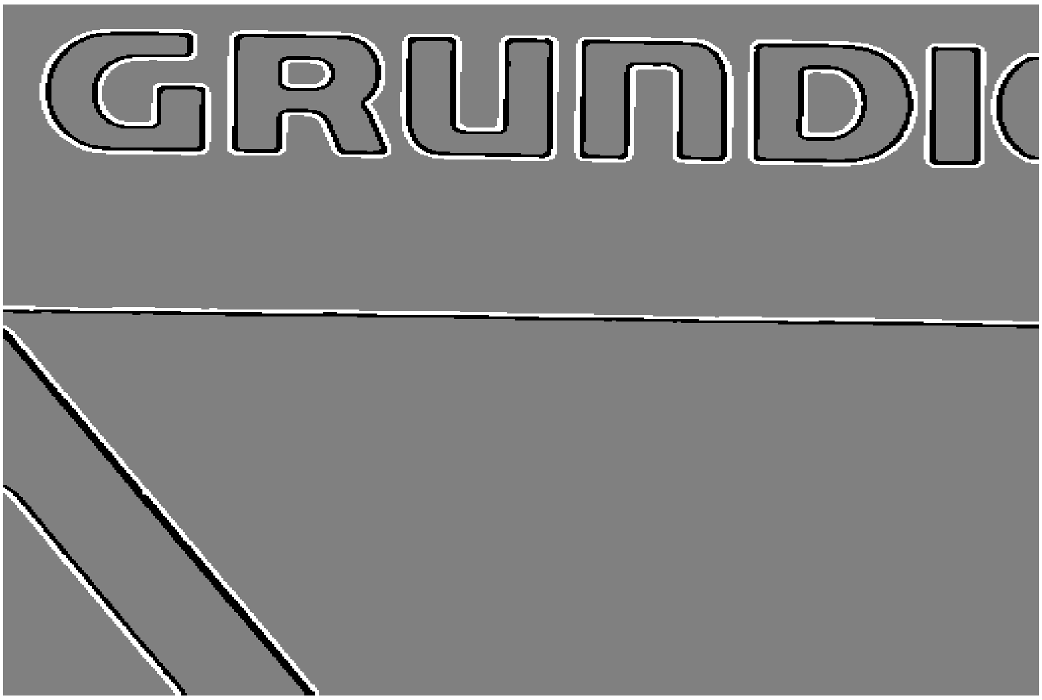


Abbildung 3.12: Kanteninformation des Testbilds GRUNDIG

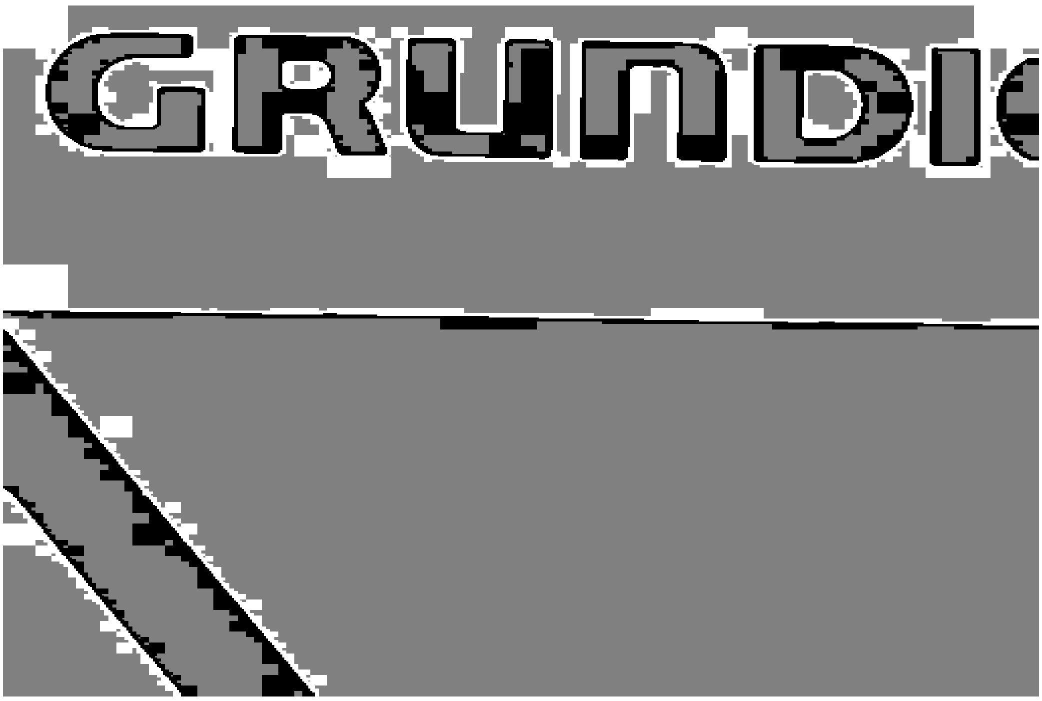


Abbildung 3.13: Ausgeglichene Kanteninformation.

Abbildung 3.14: Kanteninformation des Testbilds SAILBOAT

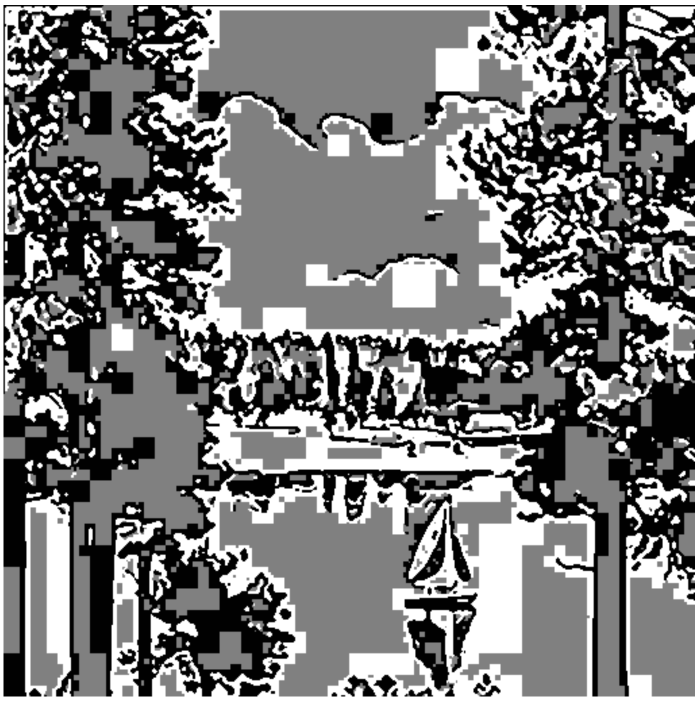


Abbildung 3.15: Ausgeglichene Kanteninformation.

nur genau vier verschiedene Zeichen vorkommen. Aus diesem Grund sind die in den folgenden Abbildungen angegebenen Kompressionsraten für die einzelnen Kompressionsverfahren etwas kritischer zu lesen.

Angegeben ist in den Abbildungen die Länge der komprimierten Datei in Byte bezogen auf die Länge der unkomprimierten Textdatei in Byte. Dabei belegt ein Symbol (wie es bei Text üblich ist) 1 Byte, also 8 Bit. Da wir aber nur vier unterschiedliche Symbole benutzen, könnten wir unseren String aber auch mit nur 2 bit/Symbol abspeichern. Auf diese Weise erreichten wir ohne Anstrengung eine Kompressionsrate von 75%. Es kann also im folgenden nur darum gehen, die trivialerweise erreichbare Kompressionsrate von 75% zu überbieten.

In diesem Licht betrachtet schneiden die Huffman-Verfahren nicht sonderlich gut ab. Die Abbildungen 3.16 und 3.17 auf der nächsten Seite zeigen die Ergebnisse der Kompression der Huffman-Verfahren für alle Testbilder im Vergleich zur LZ77-Implementierung. Wie wir sehen, erreicht keine der Huffman-Implementierungen die 75%-Grenze.

Woran liegt das? Alle diese Verfahren verwenden zur Modellierung des Zeichenstroms nur die relativen Häufigkeiten der einzelnen Zeichen. Die relative Häufigkeit wird dabei entweder vorab über die ganze Datei (statisch) oder aber während der Kodierung aus den bisher kodierten Zeichen (adaptiv) ermittelt. Die Implementierungen ‚Huffman (1)' und ‚Huffman (2)' benutzen die statische Variante, während die adaptive Variante von den Implementierungen ‚adaptiver Huffman' und ‚Splay Tree' benutzt wird. Ein weiterer Nachteil dieser Verfahren ist, daß sie zur Übertragung eines kodierten Zeichens eine ganzzahlige Anzahl von Bits benötigen. Unter diesen Umständen ist eine gute Kompression nur zu erreichen, wenn ein einzelnes Zeichen sehr häufig vorkommt. Diese Aussage läßt sich auch genauer quantifizieren.

Es seien vier Symbole mit den relativen Häufigkeiten $n_1 \geq n_2 \geq n_3 \geq n_4$ und $\sum_{i=1}^{4} n_i = 1$ gegeben. Ein String der Länge N (über diesen Symbolen) läßt sich mit 2 bit/Symbol, also in insgesamt $2N$ Bit speichern. Ein Huffman-Code benutzt zur Speicherung (oder Übertragung) des Symbols mit der relativen Häufigkeit n_1 ein Bit, für das Symbol mit der relativen Häufigkeit n_2 zwei Bit und für die anderen beiden Symbole drei Bit. Im Mittel wird also ein kodiertes Zeichen $n_1+2n_2+3n_3+3n_4 = 1+n_2+2(n_3+n_4)$ Bit benötigen. Damit ein Zeichen mit (im Mittel) weniger als 2 Bit übertragen werden kann, muß $n_3+n_4 \leq n_1$ gelten (einfache Rechnung). Dieses ist auch sofort einleuchtend: Der Verlust bei der Übertragung der drei Bit langen Zeichen darf nicht größer sein, als der Gewinn bei der Übertragung des häufigsten Symbols.

Abbildung 3.18 zeigt die relativen Häufigkeiten der Symbole in den Textstrings der einzelnen Testbilder nach dem Quadtreegrenzen-Ausgleich. Wie wir sehen, ist die Bedingung $n_3 + n_4 \leq n_1$ bei fast allen Bildern verletzt.

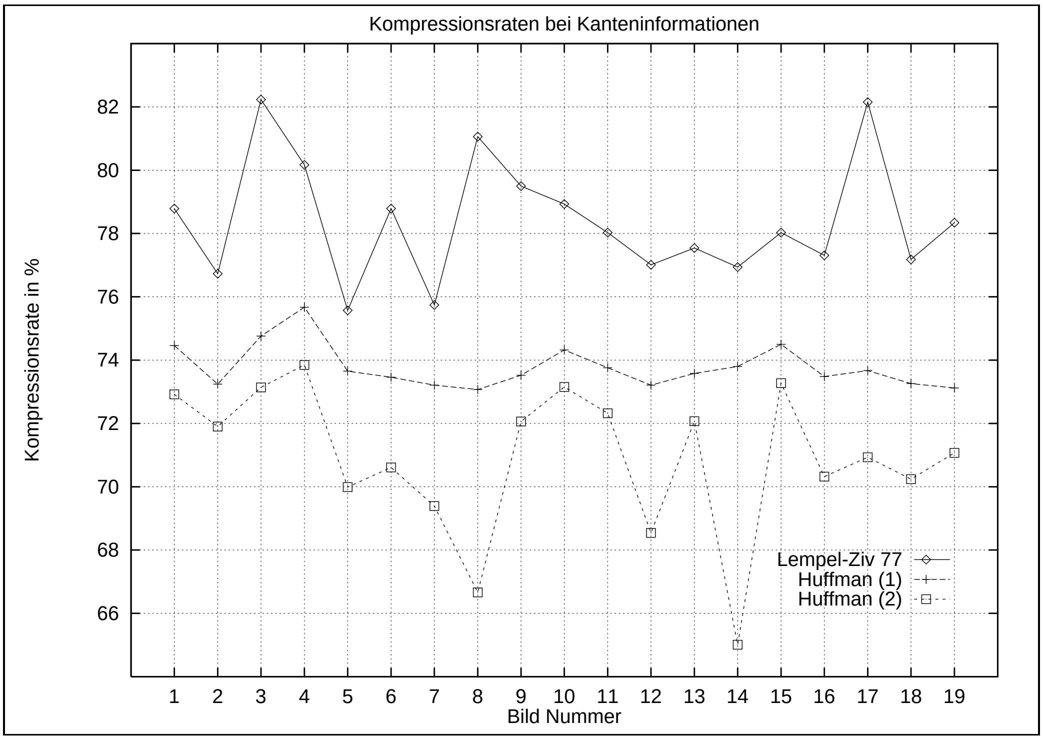


Abbildung 3.16: Kompressionsraten bei der Kanteninformation für LZ77- und statische Huffman-Kompression.

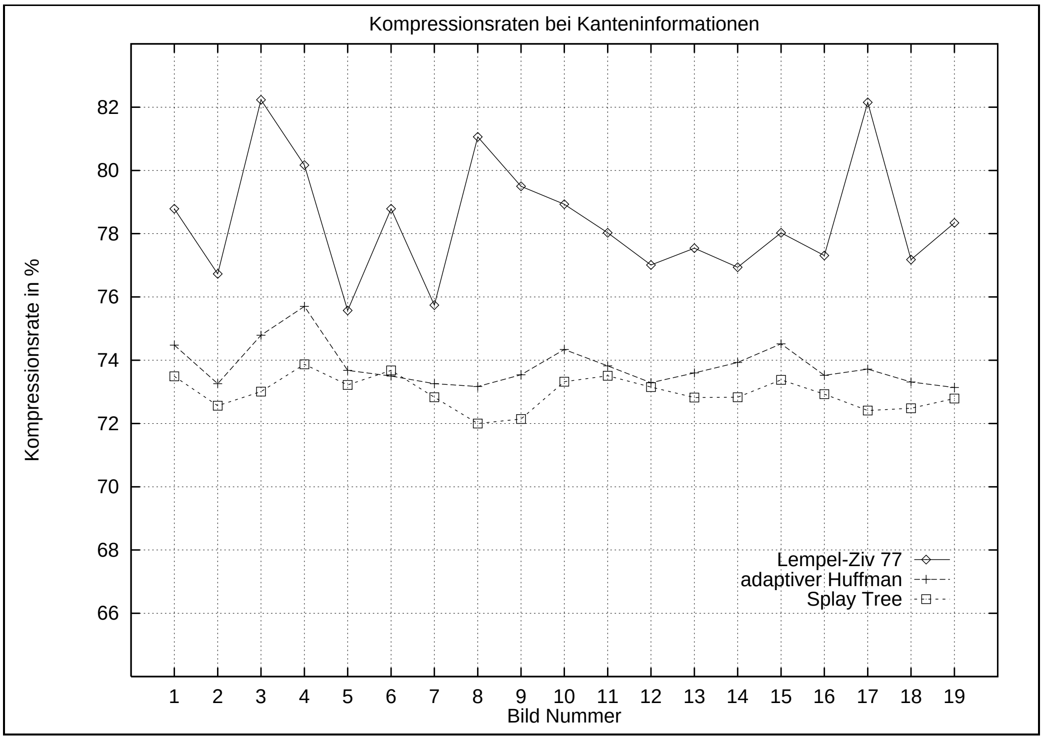


Abbildung 3.17: Kompressionsraten bei der Kanteninformation für adaptive Huffman-Kompression.

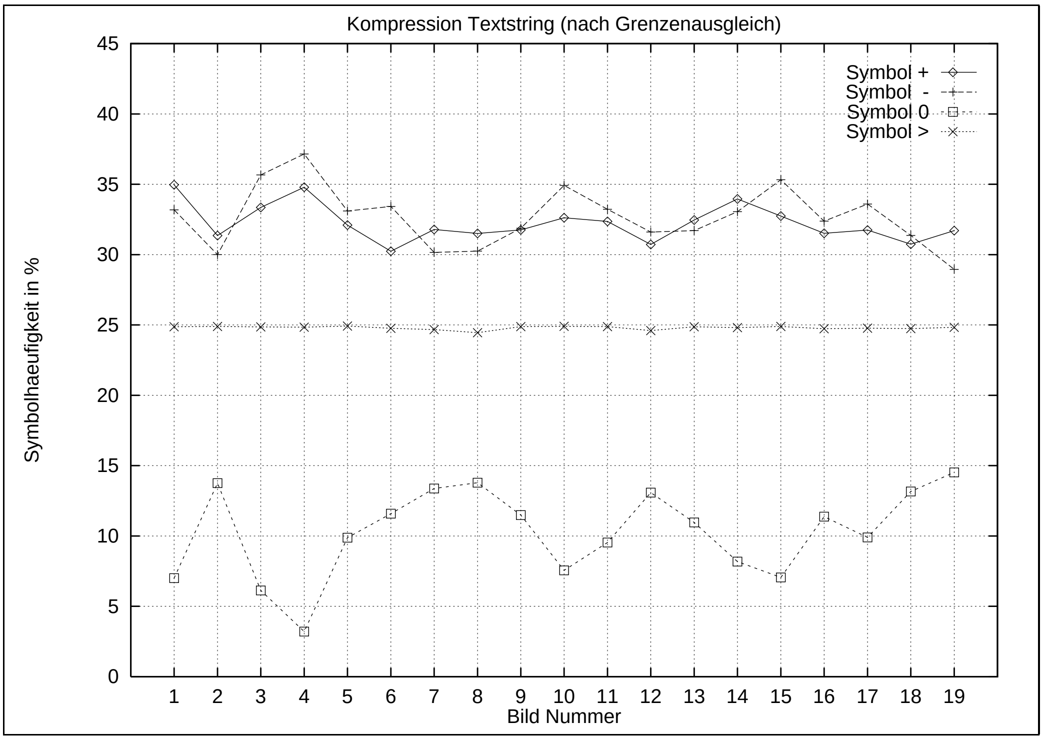


Abbildung 3.18: Symbolhäufigkeiten nach dem Quadtreegrenzen-Ausgleich.

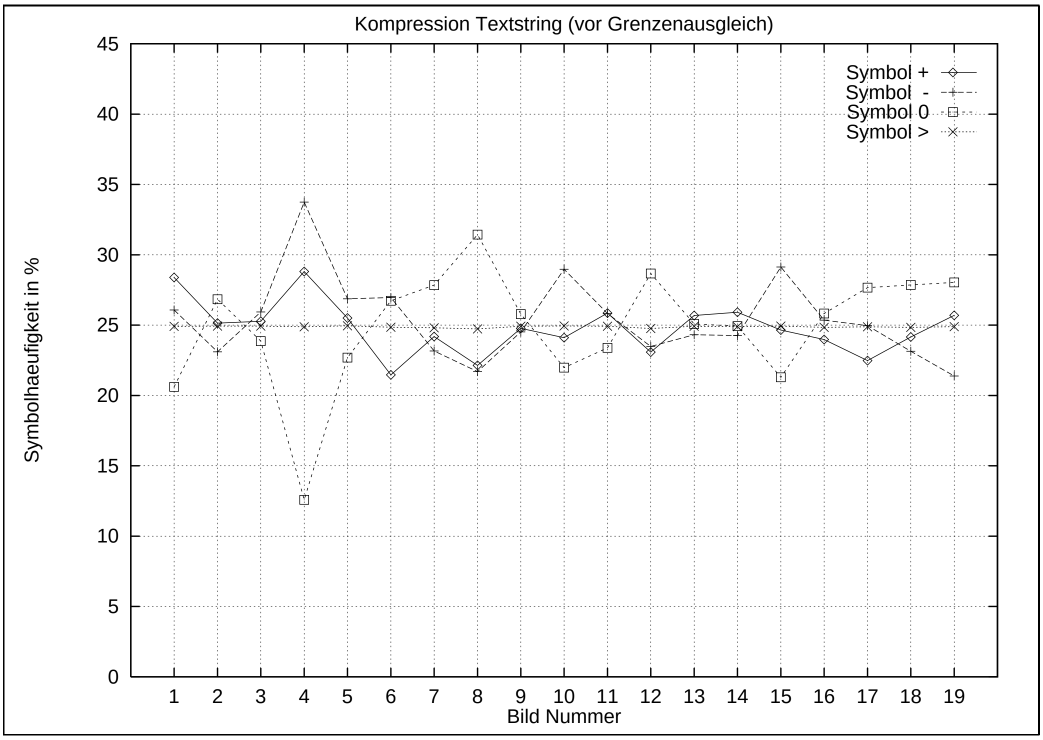


Abbildung 3.19: Symbolhäufigkeiten vor dem Quadtreegrenzen-Ausgleich.

D.h. mit den oben skizzierten Verfahren ist an dieser Stelle prinzipiell keine Kompression erreichbar. (Dieses Ergebnis ist auch unabhängig von der Implementierung des Verfahrens! Eine nachträgliche Rechtfertigung der Vernachlässigung speziell an unsere ‚4-Symbol-Situation' angepaßter Huffman-Implementierungen.) Vergleichen wir aber die relativen Häufigkeiten der einzelnen Symbol vor dem Quadtreegrenzen-Ausgleich (siehe Abbildung 3.19) mit der danach (Abbildung 3.18), so sehen wir, daß der Quadtreegrenzen-Ausgleich diese Situation nicht verursacht hat. Ganz im Gegenteil: Die relativen Häufigkeiten der Symbole vor dem Quadtreegrenzen-Ausgleich war fast ‚gleichverteilt'. Eine Situation in der Kompressionsverfahren ohne Kontextmodellierung (Huffman, arithmetische Kodierung nullter Ordnung) fast keine Kompression erreichen. Unser Quadtreegrenzen-Ausgleich trägt also in doppelter Weise zur Kompression der Kanteninformation bei: Zum einen verkürzt er die erzeugten Textstrings (zum Teil erheblich), zum anderen ‚verzerrt' er die Häufigkeitsverteilung und trägt so zur besseren Komprimierbarkeit des resultierenden Textstrings bei.

Die Abbildung 3.20 auf der nächsten Seite zeigt die Kompressionsraten durch arithmetische Kodierung mit Kontextmodellen verschiedener Ordnung. Bei einem Kontextmodell nullter Ordnung gehen in die Symbolvorhersage nur die relativen Häufigkeiten der Symbole ein. Dieses Verfahren ist bezüglich der Modellierung direkt mit den Huffman-Verfahren vergleichbar (es benutzt allerdings ein anderes Übertragungskonzept, eben arithmetische Kodierung). Bei einem Kontextmodell n-ter Ordnung bilden die letzten n Symbole den Kontext, in welchem die Vorhersage für das aktuell zu übertragende Symbol erfolgt. Für eine einführende Darstellung zur Kontextmodellierung siehe den Beitrag von Hirschberg und Lelewer [HL92]. Genaueres zu den Details der benutzten Implementierung findet man in Nelson's Buch [Ne92].

Wie wir der Abbildung 3.20 auf der nächsten Seite entnehmen können, steigt die Kompressionsrate mit zunehmender Ordnung des Modells zuerst an, um bei Ordnung 4 ein Maximum zu erreichen. Bei Ordnungen jenseits von vier, fällt die Kompressionsrate wieder ab. Der Fall der Ordnung 5 ist aus Gründen der Illustration noch mit in die Abbildung 3.20 aufgenommen worden. Die Ordnungen bis einschließlich neun sind noch gerechnet worden, wurden aber aus Gründen der Übersichtlichkeit und da sie keine neuen Einsichten vermitteln, nicht in die Abbildung mit aufgenommen.

Die Abbildung 3.21 schließlich zeigt noch einmal einen Vergleich der Kompressionsraten der arithmetischen Kodierung zweiter und vierter Ordnung mit dem wörterbuchbasierten LZ77-Kompressionsalgorithmus. Wir sehen, daß in allen Fällen die arithmetische Kodierung mit einem Kontextmodell vierter Ordnung die besten Kompressionsraten erreicht. Aus diesem Grund ist sie für uns die Methode der Wahl zur Kompression unserer Kanteninformation bzw. der aus ihr gewonnenen Textstrings.

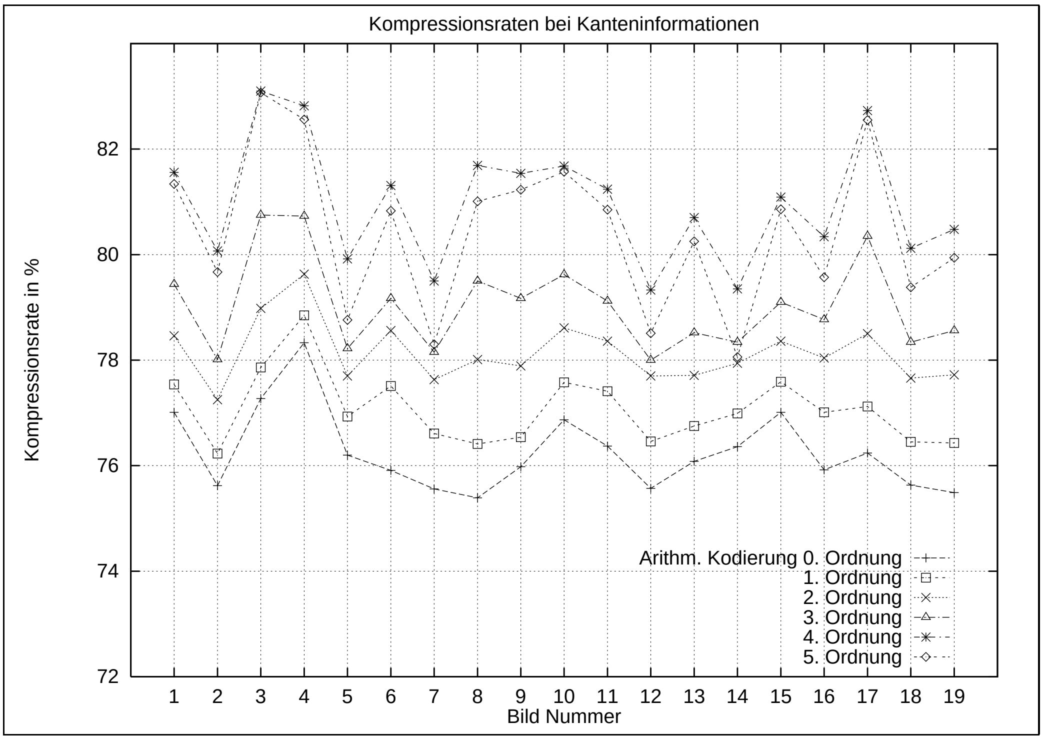


Abbildung 3.20: Kompressionsraten bei der Kanteninformation für arithmetische Kodierung mit Kontextmodellen der Ordnung 0 bis 5.

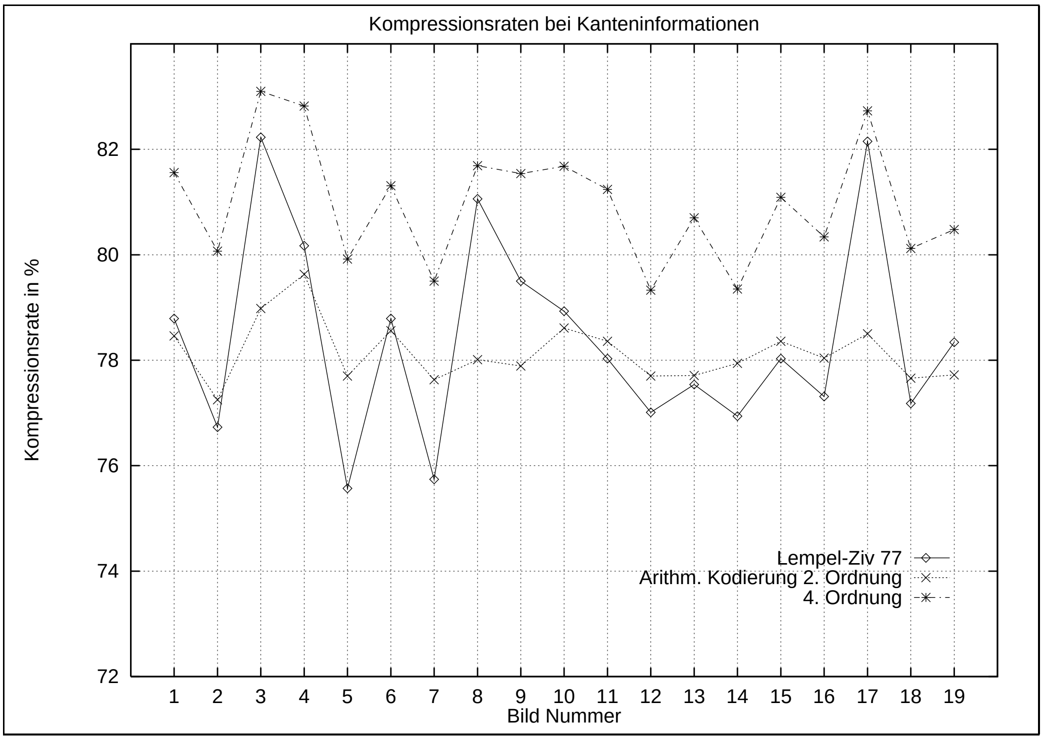


Abbildung 3.21: Kompressionsraten bei der Kanteninformation für arithmetische Kodierung mit Kontextmodellen der Ordnung 2 und 4 im Vergleich zu LZ77.

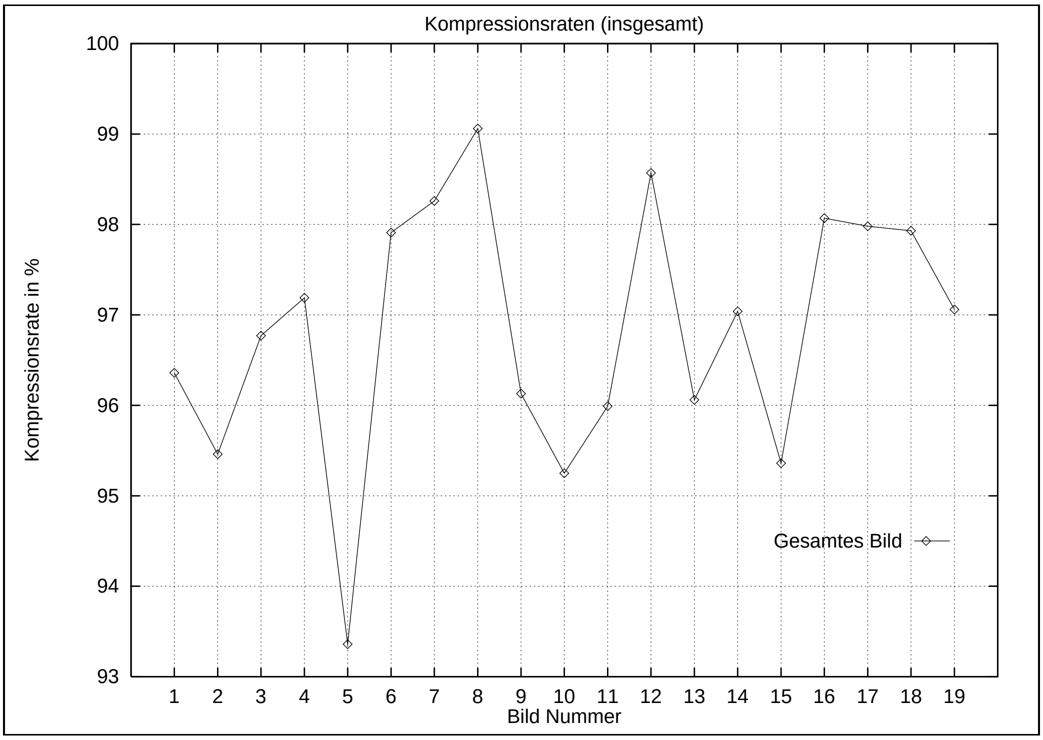


Abbildung 3.22: Kompressionsrate für das Gesamtverfahren.

Insgesamt bekommen wir eine Möglichkeit unsere Ausgangsdaten für die Rekonstruktion sehr kompakt abzuspeichern. Abbildung 3.22 zeigt die Kompressionsraten für das Gesamtverfahren, welches wir erhalten, wenn wir die Grauwertinformation mit einer arithmetischen Kodierung zweiter Ordnung und die Kanteninformation nach dem Quadtreegrenzen-Ausgleich mit einer arithmetischen Kodierung vierter Ordnung komprimieren. Wir haben bei der Auswahl der Kompressionsmethode nur Wert auf eine hohe Kompressionsrate gelegt. Wir haben bewußt an dieser Stelle darauf verzichtet, andere Auswahlkriterien für die Kompressionsmethoden zu berücksichtigen, da es uns nur um die *Möglichkeit* ging, eine hohe Kompressionsrate zu erzielen. Andere Auswahl- bzw. Designkriterien umfassen z.B.

- schnelle Kompression und/oder Dekompression,
- leichte Hardware- und/oder Software-Implementierbarkeit,
- niedriger Speicherverbrauch,
- inkrementelle Dekomprimierbarkeit

und sind stark vom Einsatzgebiet der Kompressionsmethode abhängig.

3.3 Vergleich mit Bilddatenkompressionsverfahren

Wie wir in der Einleitung zu diesem Kapitel schon beschrieben haben (siehe Seite 80), können wir den gesamten Prozeß, beginnend beim Ausgangsbild, über die verlustbehaftete Abstraktion und verlustfreie Kompression der erhaltenen Grauwert- und Kanteninformation, als verlustbehaftete Bilddatenkompressionsmethode betrachten. Die Dekompressionsmethode bestünde dann aus der Dekompression der Grauwert- und Kanteninformation und anschließender Bildrekonstruktion (-interpolation).

Dieses Verfahren besitzt für Anwendungen in dem Bereich *Maschinelles Sehen* interessante Eigenschaften, die es von anderen Bilddatenkompressionsverfahren unterscheidet. Diese Eigenschaften wollen wir im folgenden aufzeigen und wir wollen erläutern, worum wir glauben, daß sie es Wert sind, weiterentwickelt zu werden. Das Kompressionsverfahren ist zur Zeit noch nicht laufzeitoptimiert implementiert.

Was die Praktikabilität von Bilddatenkompressionsverfahren angeht, so mögen die beiden folgenden Aussagen zur *fraktalen Bildkompression* zur Illustration dienen. Diese Aussagen haben wir dem von Jean-loup Gailly betreuten Comp.Compression-FAQ (Version 25. Feb. 1995) [FAQ] entnommen. Tal Kubo (`kubo@zariski.harvard.edu`) wird dort zitiert mit:

> According to Barnsley's book "Fractals Everywhere", this method is based on a measure of deviation between a given image and its approximation by an IFS code. The Collage Theorem states that there is a convergent process to minimize this deviation. Unfortunately, according to an article Barnsley wrote for Byte a few years ago (Jan. 1988, U.D.), this convergence was rather slow, about 100 hours on a Cray, unless assisted by a person.
>
> Barnsley et al are not divulging any technical information beyond the meager bit in "Fractals Everywhere". The book explains the idea of IFS codes at length, but is vague about the application of the Collage theorem to specific compression problems.
>
> There is reason to believe that Barnsley's company has *no algorithm* which takes a given reasonable image and achieves the compression ratios initially claimed for their fractal methods. The 1000-to-1 compression advertised was achieved only for a "rigged" class of images, with human assistance. The best unaided performance I've heard of is good lossy compression of about 80-1.
>
> [FAQ, Subject 17]

Yuval Fisher (`fisher@inls1.ucsd.edu`) erwidert:

> Their performance has improved dramatically beyond what they were talking about in Byte a few years ago. Human assistance to the compression is no longer needed and the compression time is reasonable, although the more time and compute power you throw at the compression, the smaller the resulting file for the same level of quality. [FAQ, Subject 17]

Unser Kompressionsverfahren benötigt in seiner jetzigen Form nicht besonders viel Zeit für Kompression und Dekompression, so daß wir hoffen können, daß es weiter optimiert wird, falls seine positiven Eigenschaften im Bereich *maschinelles Sehen* erst richtig erkannt sind.

Die Kompressionsrichtung benötigt ca. 15 Minuten auf einem PC mit Intel 486DX50-Prozessor unter Linux und ca. 2 Minuten auf einem PowerExplorer der Firma Parsytec mit 8 PowerPC-Prozessoren (601). Die Dekompressionsrichtung benötigt ca. 2 Minuten auf einem 486DX50 und ist auf dem PowerExplorer noch nicht implementiert. Keine der jetzigen Programmversionen ist laufzeitoptimiert, da zur Zeit die Idee des Verfahrens im Vordergrund steht.

Wir werden unsere Methode mit einigen anderen Methoden zur Bilddatenkompression vergleichen und werden zu diesem Vergleich als alleiniges Vergleichskriterium die ‚visuelle Qualität‘ der dekomprimierten Bilder heranziehen. Dazu wollen wir in diesem Fall unter ‚visuelle Qualität‘ die Möglichkeit verstehen, diese Bilder sinnvoll weiteren Bildverarbeitungsprozessen (im Bereich *maschinelles Sehen*) unterziehen zu können. Wir möchten explizit darauf hinweisen, daß die Vergleichsmethoden nicht mit dem Designziel ‚visuelle Qualität‘ in obigen Sinne entwickelt wurden. Da aber alle uns bekannten verlustbehafteten Bilddatenkompressionsverfahren nicht das Gewicht auf spätere maschinelle Weiterverarbeitung der komprimierten Bilder legen, haben wir für den Vergleich keine andere Möglichkeit als diesen Ansatz.

3.3.1 Vergleichsverfahren

Wir wollen auch in diesem Kapitel unserer Philosophie treu bleiben und nur solche Verfahren in den Vergleich einbeziehen, für die uns Implementierungen vorliegen. Mit den uns vorliegenden Implementierungen decken wir drei große Bereiche von Bilddatenkompressionsmethoden ab, die sich in ihrem methodischen Vorgehen unterscheiden. Die Implementierungen sind allerdings meist keine Produktionsversionen, sondern Beispielimplementierungen, die die Grundlage der Technik oder Softwaretechnologie demonstrieren sollen. Auch sind die implementierten Algorithmen nicht immer ‚State of the Art‘.

Schon deswegen nicht, da sich im Bereich Bilddatenkompression Geld verdienen läßt und die besten Algorithmen (bzw. deren Implementationen) Softwarepatenten unterliegen.

Die zum Vergleich benutzen Implementationen decken die Bereiche

- Transformationskodierungen,
- fraktale Bildkompression und
- Waveletkodierungen

ab. Wir wollen im folgenden kurz die methodisch unterschiedlichen Ansätze der Bereiche skizzieren, müssen für die genauen Details dann aber auf andere Literatur verweisen.

Transformationskodierungen

Bei den Transformationskodierungen wird das Ausgangsbild in gleichgroße $n \times n$-Blöcke unterteilt (meist 8×8-Blöcke) und diese Blöcke werden dann einzeln komprimiert. Das Teilbild in einem dieser Blöcke $\mathcal{B}$ wird dann als Graph einer Funktion f betrachtet. Bezüglich der kanonischen Basis $(\chi_p)_{p \in \mathcal{B}}$ (χ_p ist die charakteristische Funktion zum Pixel p) läßt sich die Funktion f dann darstellen als $f = \sum_{p \in \mathcal{B}} f_p \cdot \chi_p$, dabei ist f_p der Grauwert des Pixels p.

Die Transformationskodierung besteht im ersten Schritt daraus, die Funktion f bezüglich einer anderen (wählbaren) Basis $(\varphi_j)_{j=1,\ldots,n^2}$ darzustellen: $f = \sum_j c_j \cdot \varphi_j$. Der Übergang von den Grauwerten f_p zu den *Transformationskoeffizienten* c_j wird durch eine invertierbare $n^2 \times n^2$-Matrix T vermittelt. Die Datenkompression wird dadurch erreicht, daß nicht alle möglichen Transformationskoeffizienten c_j zugelassen werden (zweiter Schritt). Der Bereich der c_j wird quantisiert und die quantisierten Koeffizienten c'_j werden dann (meist) mittels einer Entropiekodierung E kodiert.

In der Regel werden nicht direkt die quantisierten Transformationskoeffizienten als Symbole der Kodierung benutzt, sondern die Koeffizienten bestimmen über statistische Modelle die Kodierungssymbole. Beispielsweise werden im Baseline-Modus des JPEG-Standards die Symbole aus Kombinationen der Größenordnung der Transformationskoeffizienten und aus Run-Length-Kodes für Nullfolgen von Transformationskoeffizienten gebildet.

Als Möglichkeiten für die Entropiekodierung der so erzeugten Symbole bieten sich alle Verfahren aus Abschnitt 2 an. Der ganze Vorgang läßt sich schematisch so darstellen:

$$(f_p) \xmapsto{T} (c_j) \xmapsto{Q} (c'_j) \xmapsto{E} \text{Code}$$
$$\text{Code} \xmapsto{E^{-1}} (c'_j) \xmapsto{T^{-1}} (f'_p)$$

Nur bei der Quantisierung Q geht Information verloren.

Die Auswahl der Basis (φ_j) und damit von T, die Wahl der Quantisierung Q und die Wahl des Modells zur Erzeugung der Kodierungssymbole aus den Transformationskoeffizienten haben entscheidenden Einfluß auf die Kompressionsrate und die Qualität des dekomprimierten Bilds f'.

Es wurden in der Vergangenheit verschiedene Transformationen T in diesem Zusammenhang untersucht. Unter ihnen waren

- die Karhunen-Loève-Transformation,
- die Walsh-Hadamard-Transformation,
- die Haar-Transformation,
- die diskrete Fourier-Transformation und
- die diskrete Kosinus-Transformation.

Die Transformationskodierung mittels diskreter Kosinus-Transformation ist mittlerweile in den JPEG-Standard eingeflossen. Für genauere Einzelheiten zu den einzelnen Transformationen und der Transformationskodierung im allgemeinen möchten wir auf [GW92, Seite 374ff] verweisen.

Die von uns verwendete Implementierung ist die Implementierung des Baseline-JPEG durch die ‚Independent JPEG Group' um TOM LANE. Eine Einführung in die genauere Arbeitsweise von Baseline-JPEG findet man in [GW92], [Wa91] und [FAQ, Subject 75]. Die letzten Details über JPEG findet man in [PM92], dessen Autor PENNEBAKER auch Editor des eigentlichen Standarddokuments ISO 10918-1 JPEG DIS ist, welches einen Anhang von [PM92] bildet.

Fraktale Bildkompression

Die Grundidee für die *fraktale Bildkompression* ist relativ einfach zu beschreiben, wenn wir den ganzen Sprachjargon weglassen, der sich in diesem Bereich eingebürgert hat.

Wir identifizieren unser (kontinuierlich gedachtes) Bild $f\colon R \to \mathbb{R}$ mit seinem Graphen $\operatorname{graph}(f) \subset R \times \mathbb{R}$ ($\operatorname{graph}(f) \in \mathcal{G}$). Mit $\mathcal{G}$ werde die Menge der (kompakten) Graphen von Bildern bezeichnet. Die Menge $\mathcal{G}$ wird mit der Hausdorff-Metrik ausgestattet. Versehen mit dieser Metrik ist die Menge $\mathcal{G}$ ein vollständiger metrischer Raum. Also haben wir die Fixpunktabbildung Fix, die jeder kontrahierenden Abbildung $\mathcal{F}\colon \mathcal{G} \to \mathcal{G}$ ihren Fixpunkt $\operatorname{Fix}(\mathcal{F})$ zuordnet.

Die Grundidee der *fraktalen Bildkompression* ist es, zu einem gegebenen Bild f eine ‚gut beschreibbare' kontrahierende Abbildung $\mathcal{F}$ zu finden mit $\mathcal{F} \in \mathrm{Fix}^{-1}(\mathrm{graph}(f))$. Dabei bezieht sich ‚gut beschreibbar' auf die Anzahl von Bits, die zur Speicherung der Abbildung $\mathcal{F}$ benötigt werden. Genau genommen wird die Abbildung $\mathcal{F}$ nicht in der Faser von $\mathrm{graph}(f)$ gesucht, sondern es wird zugelassen, daß der Fixpunkt von $\mathcal{F}$ ein wenig von f abweicht.

Der Name *fraktale Bildkompression* ist in der Methodik begründet, mit welcher versucht wird, dieses Invertierungsproblem zu lösen. Man versucht dabei auszunutzen, daß in natürlichen Bildern spezielle Ausschnitte anderen ähnlich sind (Stichwort: Selbstähnlichkeit). Der Versuch ähnliche Bildausschnitte aufeinander abzubilden führt auf eine Strategie, die kontrahierende Abbildung $\mathcal{F}$ stückweise zu definieren. Der wesentliche Begriff ist der Begriff der ‚partiellen iterierten Funktionensysteme' (PIFS).

Definition: *Ein partielles iteriertes Funktionensystem (PIFS) ist eine endliche Menge von Abbildungen*

$$w_i = v_i \times u_i \colon D_i \times \mathbb{R} \to R_i \times \mathbb{R} \qquad (1 \leq i \leq n)$$

mit $D_i \subset R$ *und* $R = \bigcup_{i=1}^{n} R_i$ *(disjunkt).*

Ein solches PIFS induziert auf natürliche Weise eine Abbildung $\mathcal{F}_{\text{PIFS}}$ zwischen Graphen

$$\mathcal{F}_{\text{PIFS}}(g) := \bigcup_{i=1}^{n} w_i(g \cap (D_i \times \mathbb{R}))$$

und sind die Abbildungen v_i kontrahierend (bzgl. euklidischer Metrik), so ist auch die Abbildung $\mathcal{F}$ kontrahierend.

Die Kunst besteht dann in der geeigneten Wahl einer Zerlegung $(R_i)_i$ der Bildebene (bzw. Pixelmenge) und in der geeigneten Wahl der Abbildungen w_i, so daß der Fixpunkt der Abbildung $\mathcal{F}_{\text{PIFS}}$ dem Ausgangsbild sehr ähnlich ist. Für die Wahl der Zerlegung $(R_i)_i$ gibt es unterschiedliche Alternativen bzw. Strategien. Man kann z.B. gleichgroße Blöcke (8×8-Blöcke o.ä.), Dreiecke einer – beliebigen oder dem Bild angepaßten – Triangulierung, Blöcke aus einer Quad- oder HV-Treedarstellung oder etwas anderes wählen.

Die Abbildungen w_i verschafft man sich meist über den Umweg über einen sog. Domainpool $\mathcal{D}$, einer Menge, die mögliche Kandidaten für die Domains D_i enthält. Für festes R_i versucht man dann ein $D \in \mathcal{D}$ und eine affin-lineare Abbildung $w_i = v_i \times u_i \colon D \times \mathbb{R} \to R_i \times \mathbb{R}$ zu finden, so daß $w_i(\mathrm{graph}(f) \cap (D \times \mathbb{R}))$ und $\mathrm{graph}(f) \cap (R_i \times \mathbb{R})$ möglichst wenig voneinander abweichen. Bei fest gewählter Zerlegung $(R_i)_i$ und fest gewähltem Domainpool $\mathcal{D}$ bleibt dann im wesentlichen ein Suchproblem übrig, welches noch mit unterschiedlichen Suchstrategien angegangen werden kann.

Wir können an dieser Stelle nicht weiter in die Details gehen und müssen auf die Literatur verweisen. Eine ausführliche, aber nicht sehr tief gehende, Einführung in die Ideenwelt der *fraktalen Bildkompression* stellt das Buch von M. Barnsley und L. Hurd [BH92] dar. Eine wesentlich kompaktere und genauere Darstellung findet man bei Y. Fisher in [Fi92] oder dem sehr ähnlichen Artikel [FJB92]. Einen guten Überblick über den Stand der Technik liefert das Buch ‚Fractal Image Compression: Theory and Application' von Y. Fisher [Fi94], welches auch ein sehr ausführliches Literaturverzeichnis zu diesem Thema besitzt.

Die hier verwendete Implementierung stammt von Y. Fisher und ist in [Fi94, Kapitel 3] besprochen. Der Quelltext findet sich in [Fi94, Anhang A] und ist unter `http://inls.ucsd.edu/y/Fractals/` zu beziehen. Fisher's Wahl der Zerlegung $(R_i)_i$ und des Domainpools $\mathcal{D}$ ist quadtreebasiert.

Waveletkodierung

Ähnlich wie bei den Transformationskodierungen wird auch bei den Waveletkodierungen das ursprüngliche Bild bezüglich einer anderen Basis dargestellt. Aber anders als bei den Transformationskodierungen wird das Bild nicht vorher in Blöcke aufgeteilt, sondern als Ganzes transformiert. Das kann man natürlich auch mit z.B. der diskreten Kosinustransformation machen. Das Problem dabei ist, daß man keine gute Interpretation für die vielen Koeffizienten hat, die dabei auftauchen würden und daß die Berechnung sehr aufwendig werden würde. Beide Probleme werden durch die spezielle Konstruktion der Waveletbasen, auf die wir hier nicht eingehen können, gemildert.

Bedingt durch ihre Konstruktion besitzen die Waveletbasen eine besondere, eine ‚pyramidale' Struktur, die es ermöglicht, den Koeffizienten eine anschauliche Interpretation zu geben und diese über effektive Filteransätze aus der Signalverarbeitung zu berechnen. Wir möchten an dieser Stelle die Interpretation der Waveletkoeffizienten erläutern, müssen für alle weiteren Details aber auf die Literatur verweisen.

In einer Waveletbasis lassen sich die Basisfunktionen so numerieren, daß man aus dem ersten Viertel der Waveletkoeffizienten eine verkleinerte Version des Originalbildes mit Auflösung $n/2 \times n/2$ Pixel berechnen kann, wenn das Ausgangsbild eine Auflösung von $n \times n$ Pixeln besessen hat. Aus diesem verkleinerten $n/2 \times n/2$-Bild zusammen mit den restlichen drei Vierteln der Waveletkoeffizienten, den ‚Hochpaßanteilen' in senkrechter, waagerechter und diagonaler Richtung, wird dann das Originalbild rekonstruiert.

Das $n/2 \times n/2$-Bild erhält man völlig analog aus einem $n/4 \times n/4$-Bild, welches aus dem ersten $1/16$-tel der Waveletkoeffizienten berechnet wird. Der

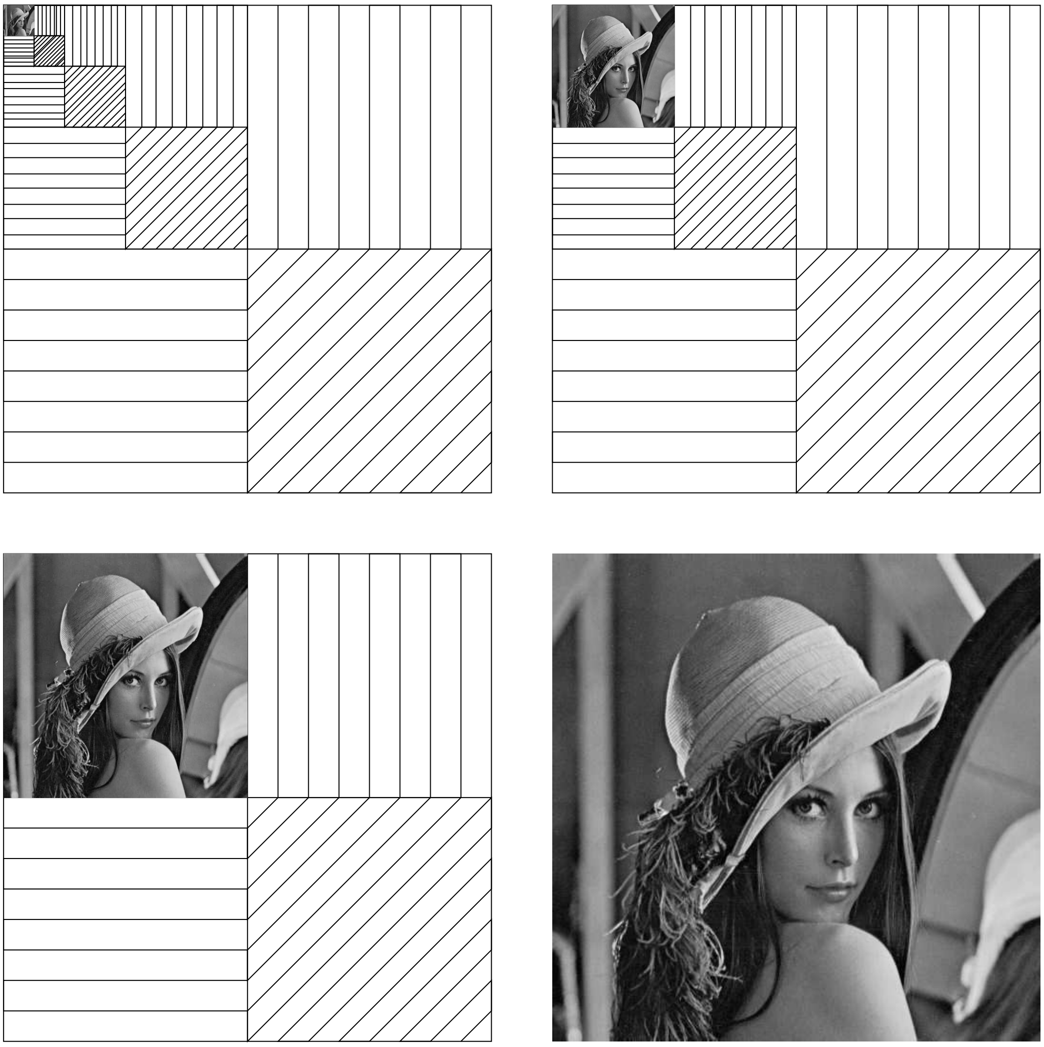


Abbildung 3.23: Die letzten Stufen bei der Waveletrücktransformation.

Prozeß startet mit den ersten vier Koeffizienten aus welchen ein 2×2-‚Bild' berechnet wird. Aus diesem, zusammen mit den nächsten 12 Koeffizienten, wird dann ein 4×4-‚Bild' berechnet usw.

Auf diesem Weg wird eine Bildpyramide berechnet, deren Bilder alle verschieden stark verkleinerte Versionen des Originalbilds sind. Deswegen läuft diese Methode bei verschiedenen Autoren auch unter der Bezeichnung *pyramidal subband coding*.

Durch diese Struktur ist jetzt leichter abschätzbar, welche Art von Fehler zu erwarten ist, wenn man die Waveletkoeffizienten zu Kompressionszwecken quantisiert (und dann natürlich entropiekodiert z.B. mit einem Huffmancode). Die unterschiedlichen Verfahren zur Kompression mittels Waveletkodierung unterscheiden sich im wesentlichen durch

- verschiedene Waveletbasen (d.h. durch verschiedene Filter zur Berechnung der Koeffizienten),
- verschiedene Quantisierungsmethoden (unterschiedliche Quantisierung der ‚Hochpaßanteile', abhängig von Richtung und Skalierung; Benutzung von modellbasierten oder Vektorquantisierungsmethoden),
- verschiedene Arten der Entropiekodierung.

Wir können an dieser Stelle nicht weiter die Theorie der Wavelets oder ihrer Anwendung verfolgen, da dieses nicht Thema dieser Arbeit ist und sicherlich auch den Umfang der Arbeit sprengen würde. Im Zusammenhang mit Bildkodierung beschreiben MALLET [Ma89] und ADELSON et. al. [ASH87] zwei unterschiedliche Blickwinkel auf denselben Gegenstand. Die Arbeit von ADELSON und SIMONCELLI [AS90] beschreibt (knapp) die hier verwendete Implementierung EPIC (Efficient Pyramid Image Coder), welche von SIMONCELLI implementiert wurde. Der Quellcode ist zu beziehen über `ftp://whitechapel.media.mit.edu/pub/epic.tar.Z`.

3.3.2 Visueller Vergleich der Kompressionsresultate

In diesem Abschnitt wollen wir an Hand dreier ausgewählter Testbilder die interessanten Eigenschaften unseres ‚Kompressionsverfahrens' (von dem wir im folgenden nur als *Kompression durch Abstraktion (KdA)* sprechen wollen) demonstrieren. Wir haben dafür die Bilder Nr. 8 (GRUNDIG), Nr. 10 (IC) und Nr. 15 (SAILBOAT) gewählt, da wir an jedem Bild etwas typisches demonstrieren wollen.

Das Testbild GRUNDIG (siehe Abbildung 3.24 auf der nächsten Seite) ist der typische Vertreter eines strukturarmen Bilds. Es besteht im wesentlichen aus einem großen Schriftzug auf einem, durch Schrift texturierten, Untergrund. Im linken unteren Teil des Bildes sieht man noch Teile des Kamerastativs und im rechten unteren Teil sieht man (wenn die Druckwiedergabe nicht allzu schlecht ist!) einen nicht ganz scharf begrenzten Schatten.

Wenn wir uns entscheiden, daß uns die Textur des Hintergrundes nicht interessiert, so können wir dieses Bild mittels KdA (also mit unserem Verfahren) auf ca. 1% seiner ursprünglichen Größe von 262144 byte komprimieren (siehe Abbildung 3.25 auf der nächsten Seite). Die genauen Dateilängen nach der Kompression sind in Tabelle 3.3 auf Seite 110 zusammengestellt.

Wir wollen an dieser Stelle anmerken, daß wir versucht haben, die wählbaren Parameter der einzelnen Vergleichsverfahren so zu justieren, daß die

Abbildung 3.24: Original des Testbilds GRUNDIG.

Abbildung 3.25: GRUNDIG komprimiert mit KdA. (0.95%)

Kompressionsverfahren (Implementierung)	Länge der Datei in byte	in %	Abbildung Nr.	Seite
Original	262144	100	3.24	109
KdA	2480	0.95	3.25	109
Transformationskodierung (JPEG)	3615	1.38	3.26	111
Transformationskodierung (JPEG)	5075	1.94	3.27	111
WaveletKodierung (EPIC)	2652	1.01	3.28	112
Fraktale Bildkompression (YF)	2550	0.97	3.29	112

Tabelle 3.3: Dateilängen für das Testbild GRUNDIG.

Verfahren die Bilder auf ungefähr den selben Prozentsatz der Ausgangsgröße komprimieren. Wo dieses möglich war, haben wir die zusätzlich freien Parameter so justiert, daß die Verfahren möglichst visuell gute Bilder produzieren, auch wenn wir höhere Rechenzeiten in Kauf nehmen mußten.

Das vorrangige Ziel war es aber, die von unserem Verfahren (KdA) erreichte Kompressionsrate zu erreichen. Dieses ist im Fall unseres ersten Testbilds GRUNDIG mit JPEG nicht möglich gewesen. Die von uns verwendete Implementierung von JPEG braucht für dieses Bild mindestens 1.38%, aber das dabei produzierte Bild (Abbildung 3.26 auf der nächsten Seite) ist nicht besonders ansprechend. Wir sehen hier ganz deutlich die Probleme, die eine Aufteilung des Bildes in Blöcke verursacht. Es ist schon erstaunlich, daß JPEG mit den hier zur Verfügung stehenden 7 bit/Block überhaupt etwas kodieren kann.

Komprimiert man das Bild GRUNDIG mit JPEG nicht ganz so stark (Abbildung 3.27 auf der nächsten Seite zeigt eine Kompression auf 1.94%), so wird die Kontur des Schriftzugs deutlich besser. Gleichzeitig entstehen neue Blockstrukturen im Hintergrund und an den Schattengrenzen im unteren rechten Bildteil. JPEG versucht auch die Hintergrundtextur zu rekonstruieren, das aber mit wenig Erfolg.

Da die Waveletkodierung (EPIC) nicht mit Blöcken arbeitet, hat sie die Möglichkeit bei größerer Kompression (Abbildung 3.28 auf Seite 112) weniger (Block-) Artefakte zu produzieren. Die Artefakte sind hier eher ‚welliger'. Auch EPIC produziert an den Stellen der Hintergrundbeschriftung etwas, das aber nichts mit Textur im weiteren oder der Originalbeschriftung im engeren Sinne zu tun hat.

Die fraktale Bildkompression (YF: für YUVAL FISHER) produziert ein eher ‚weicheres' und ‚unschärferes' Bild (Abbildung 3.29 auf Seite 112) mit wenig auffälligen Artefakten und schlechter Kantenreproduktion.

Das Testbild GRUNDIG ist für uns ein Beispiel für ein hochkomprimierbares Bild. Wir sehen, daß wir hohe Kompression nicht dadurch erreichen

Abbildung 3.26: Grundig komprimiert mit Jpeg (1.38%).

Abbildung 3.27: Grundig komprimiert mit Jpeg (1.94%).

Abbildung 3.28: Grundig komprimiert mit Epic (1.01%).

Abbildung 3.29: Grundig komprimiert mit YF (0.97%).

können, daß wir das Bild in Blöcke zerlegen und irgendwie blockweise kodieren. Das Bild muß als Ganzes betrachtet werden, damit große zusammenhängende Grauwertflächen nicht künstlich durch Blockgrenzen getrennt werden. Das Verfahren muß in der Lage sein, das Bild als Ganzes ‚betrachten' zu können.

Man sollte bei hohen Kompressionsraten nicht versuchen die Bildfehler, die auf jeden Fall gemacht werden, zu verstecken. Dieses klingt vielleicht zuerst paradox, weil man meint, daß die Fehler solcher Verfahren natürlich zu verstecken sind. Man will die Fehler ja nicht sehen! Die Verfahren JPEG, EPIC und YF versuchen Bilder dadurch zu komprimieren, daß sie Redundanzen in der Darstellung vermeiden – das ist in Ordnung – und dadurch, daß sie Bildfehler auf kleinen Skalen einführen, von denen man – bei mäßiger Kompression – annehmen kann, daß sie das menschliche Auge nicht deutlich wahrnimmt. (Eine analoge Situation hat man, wenn man eine Funktion in eine Fourierreihe entwickelt und dann die Funktion durch einen endlichen Abschnitt dieser Reihe approximieren will. Bei hoher Kompression, d.h wenn man nur wenige Glieder der Reihe benutzt, entstehen dann aber sehr ausgeprägte neue Strukturen.) Bei hoher Kompression machen sich diesen ‚kleinen Bildfehler' aber als neue Strukturen im Bild unangenehm bemerkbar, die das Verfahren erst eingeführt hat! Unser Verfahren läßt dagegen Bilddetails weg und bekommt auf diese Weise den Raum für hohe Kompressionsraten. Die Unterschiede bei hohen Kompressionsraten sind deutlich zu sehen.

Bei den Vergleichsverfahren wird versucht die Hintergrundbeschriftung zu erhalten, aber der kumulative Effekt der Fehler auf kleinen Skalen führt dazu, daß die Reste der Hintergrundbeschriftung eher als störende Artefakte empfunden werden. Wenn jemand nur die Bilder nach der Dekomprimierung sieht, ohne das Original zu kennen, ist nicht mehr klar, wie die Flecken auf die Bilder kommen. Dieser Effekt ist sehr gut in den Grauwertprofilen auf Seite 114 zu erkennen.

Bei unserem Verfahren wird die Hintergrundbeschriftung weggelassen, aber es werden keine kleinen Fehler an der Lokalisierung der Kanten von darzustellenden Objekten eingeführt. Aus diesem Grund ist die Kantenlokalisierung bei uns genauer als bei den Vergleichsverfahren. Das Weglassen von Bilddetails macht sich in einem schlechteren l_2-Abstand unseres Bildes zum Original bemerkbar. Die Tabelle 3.4 auf Seite 115 listet die l_2-Abstände der Vergleichsbilder (ohne JPEG bei 1.38%) zum Original. Unter dem l_2-Abstand zweier Bilder $P = (p_i)_i$ und $Q = (q_i)_i$ mit Pixeln p_i und q_i $(i = 1, 2, \ldots, 512^2)$ verstehen wir den Ausdruck

$$d(P, Q) = \left(\sum_{i=1}^{512^2} (p_i - q_i)^2 \right)^{\frac{1}{2}}.$$

Abbildung 3.30: Grundig mit eingezeichneter Zeile 175.

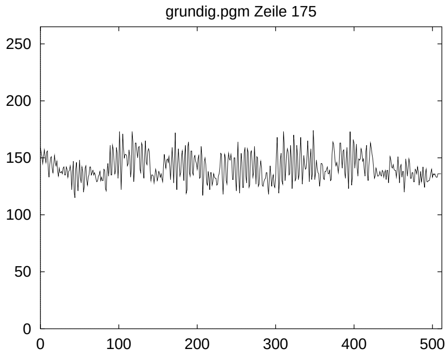


Abbildung 3.31: Grauwertprofil der Zeile 175 im Bild Grundig.

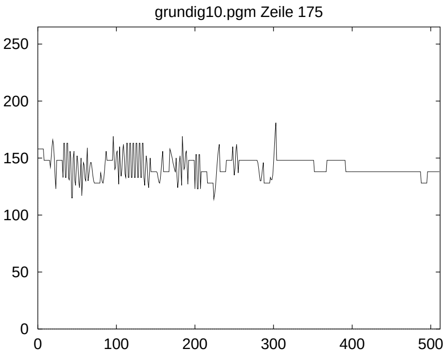


Abbildung 3.32: Grauwertprofil im Jpeg komprimierten Bild.

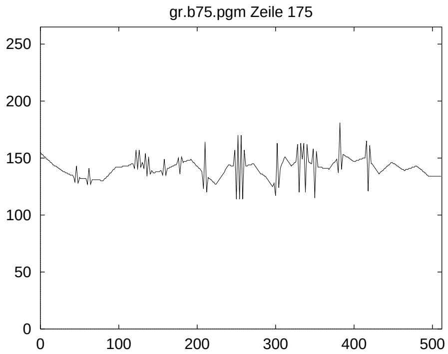


Abbildung 3.33: Grauwertprofil im Epic komprimierten Bild.

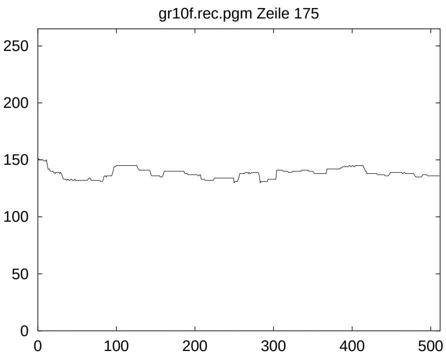


Abbildung 3.34: Grauwertprofil im YF komprimierten Bild.

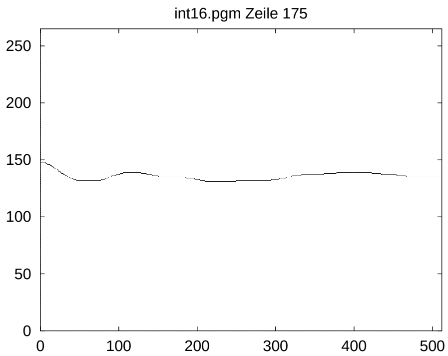


Abbildung 3.35: Grauwertprofil im KdA komprimierten Bild.

Verfahren	Abbildung	l_2-Abstand
KdA	3.25	3878
JPEG	3.27	3533
EPIC	3.28	3535
YF	3.29	3721

Tabelle 3.4: l_2-Abstände zum Testbild GRUNDIG.

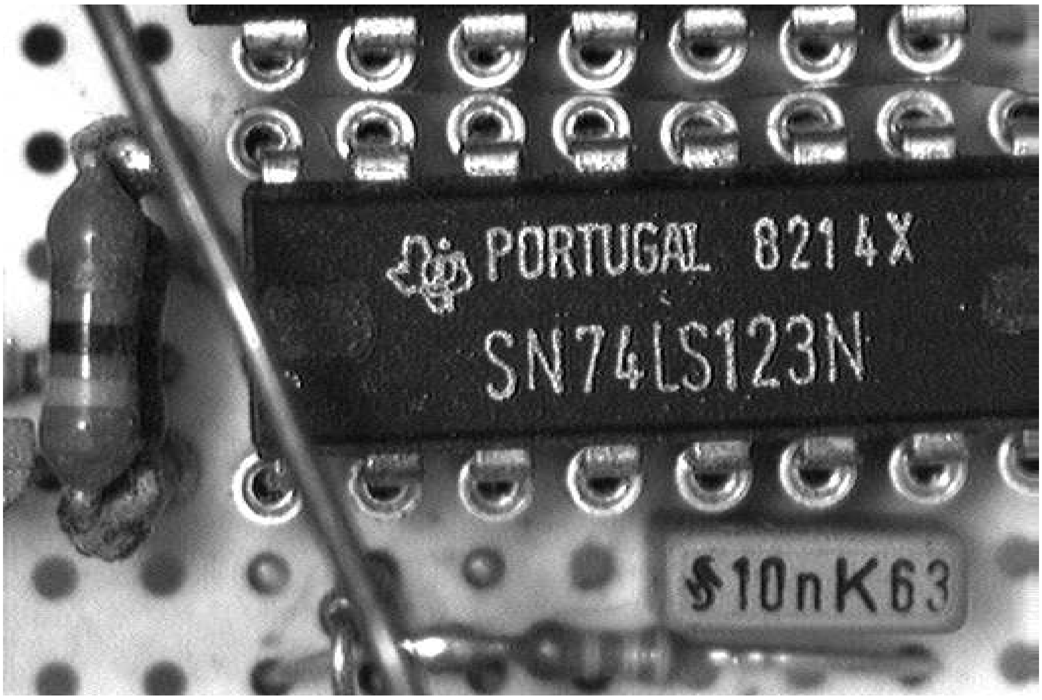


Abbildung 3.36: Original des Testbilds IC.

Wir sehen daran, daß der l_2-Abstand vom Original sicherlich kein adäquates Maß für visuelle Güte eines komprimierten Bildes ist, auch wenn er (oder verwandte Maße) in der Vergangenheit häufig so benutzt wurden.

Die Folgen der unterschiedlichen Komprimierungsstrategien werden an dem nächsten Beispiel, dem Bild Nr. 10 IC, besonders deutlich. Die nächsten Abbildungen zeigen das Original (Abbildung 3.36), die drei Vergleichsverfahren (JPEG: Abbildung 3.38; EPIG: Abbildung 3.39; YF: Abbildung 3.40) und unser Verfahren (Abbildung 3.37). Die Tabelle 3.5 enthält die Dateilängen nach der Kompression. Die Kompressionsrate liegt jeweils um 95%, also in einem Bereich, in welchem JPEG und auch die anderen Vergleichsverfahren für Menschen ansprechende Bilder produzieren. Der optische Vergleich ergibt auch keine großen Differenzen zwischen dem Original und den Vergleichsverfahren.

Abbildung 3.37: IC komprimiert mit KdA (4.77%).

Abbildung 3.38: IC komprimiert mit Jpeg (4.84%).

Abbildung 3.39: IC komprimiert mit Epic (4.86%).

Abbildung 3.40: IC komprimiert mit YF (4.87%).

Kompressionsverfahren (Implementierung)	Länge der Datei in byte	in %	Abbildung Nr.	Seite
Original	262144	100	3.36	115
KdA	12509	4.77	3.37	116
Transformationskodierung (Jpeg)	12696	4.84	3.38	116
WaveletKodierung (Epic)	12753	4.86	3.39	117
Fraktale Bildkompression (YF)	12763	4.87	3.40	117

Tabelle 3.5: Dateilängen für das Testbild IC.

Bei genauerer Betrachtung finden wir auch hier (soweit es die Druckwiedergabe erlaubt) die oben geschilderten Tendenzen wieder: Jpeg produziert leicht Blockartefakte (gut zu sehen im IC-Körper), Epic produziert einen ‚welligen' IC-Körper, während bei YF der Gesamteindruck eher unscharf ist. Insgesamt wirken die Bilder auf den menschlichen Betrachter (!) lebendiger als unsere Rekonstruktion (Abbildung 3.37). Das ist aber auch so gewollt, denn wir wollten durch Weglassen von kleinen Bilddetails die wesentlichen Bildbestandteile für eine spätere maschinelle Weiterverarbeitung kompakt speichern.

Stellen wir uns also vor, wir wollten auf den komprimierten Bildern z.B. eine Ziffernerkennung aufsetzen. In diesem Fall kollidiert die Methode, die Bildfehler vor dem menschlichen Betrachter in kleinen Strukturen zu verstecken, die außerdem noch für ‚Lebendigkeit' des Bildes sorgen, mit der später beabsichtigten Anwendung.

Die Abbildungen 3.41 bis 3.45 zeigen jeweils Vergrößerungen desselben 90×50 Pixel großen Ausschnitts der Gesamtansicht für jedes Verfahren. Da es die Methoden zur fraktalen Bildkompression erlauben, Bilder mit einer anderen Auflösung zu dekomprimieren als sie komprimiert wurden, haben wir zur Dekompression neben der Originalauflösung von 512×512 Pixeln auch die Auflösung 2048×2048 Pixel gewählt. Die Abbildung 3.46 zeigt den entsprechenden Ausschnitt. Die seriösen Vertreter der Richtung fraktale Bildkompression weisen natürlich darauf hin, daß die zusätzlichen Strukturen in der höheren Auflösung nichts mit Strukturen zu tun haben, die man bekäme, wenn man dieselbe Szene mit einer höherauflösenden Kamera aufnehmen würde. Mit anderen Worten, die zusätzlichen Strukturen sind durch den Prozeß der Dekomprimierung bedingt und haben nichts mit den Bildinhalten zu tun.

Die einzelnen Ausschnitte brauchen nicht allzu ausführlich kommentiert zu werden. Auch ohne Beschriftung wäre jeder Leser in der Lage, die einzelnen Ausschnitte den Verfahren zuzuordnen. Nach einer Durchsicht der einzelnen Ausschnitte dürfte auch klar sein, warum die Gesamtansicht des mit unserem

Abbildung 3.41: Original des Testbilds IC (Ausschnitt).

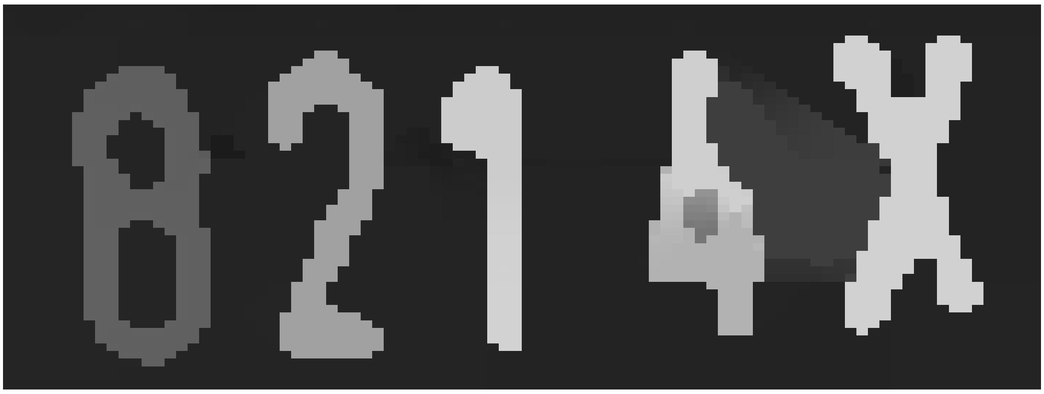


Abbildung 3.42: IC komprimiert mit KdA (Ausschnitt).

Abbildung 3.43: IC komprimiert mit JPEG (Ausschnitt).

Abbildung 3.44: IC komprimiert mit Epic (Ausschnitt).

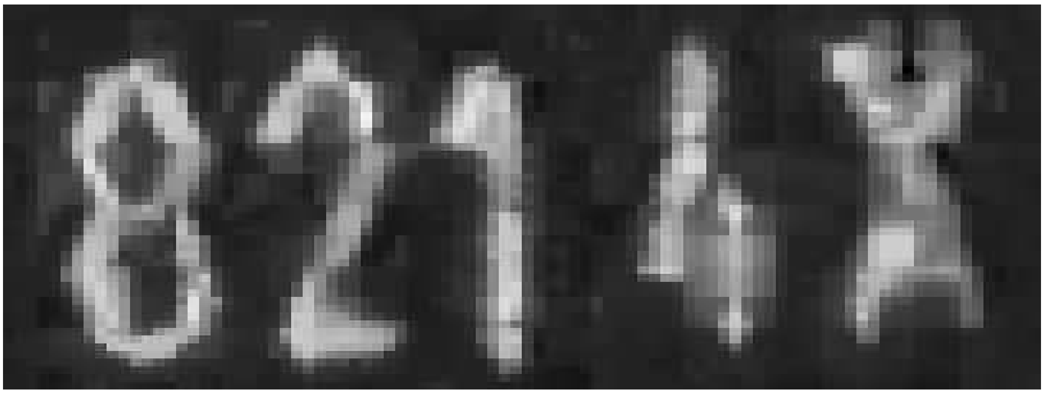


Abbildung 3.45: IC komprimiert mit YF (Ausschnitt).

Abbildung 3.46: IC komprimiert mit YF (Ausschnitt 4-fach vergrößert).

Verfahren rekonstruierten Bilds (Abbildung 3.37) so künstlich wirkt: Dem Bild fehlen die Grauwertfluktuationen auf kleiner Größenskala; dem Bild fehlt das, was spätere Bildverarbeitungsprozesse erschwert.

Das letzte hier verwendete Testbild ist das Bild Nr. 15 SAILBOAT. Wir haben das Bild hier ausgewählt, weil wir an dieser natürlichen Szene gut demonstrieren können, was es heißt, auf kleine Bilddetails zu verzichten. Der Leser möge einmal das Original (Abbildung 3.47) mit unserer Rekonstruktion (Abbildung 3.48) aus einer etwas größeren Entfernung (ca. 1–2 Meter) betrachten. Er wird bemerken, daß alle eklatanten Unterschiede zwischen den beiden Abbildungen nivelliert werden. Unsere Rekonstruktion verhält sich damit ähnlich wie eine Theaterkulisse, bei deren Ausführung auch auf alle aus der Entfernung nicht sichtbaren (also in diesem Sinne kleinen) Details verzichtet wird. Es ist u.a. diese Analogie zum menschlichen Sehen, die uns glauben läßt, mit unserem Ansatz in die richtige Richtung zu arbeiten.

Wir wollen hier nicht behaupten, daß sich die Bilder der Vergleichsverfahren (JPEG: Abbildung 3.49; EPIC: Abbildung 3.50; YF: Abbildung 3.51) nicht ähnlich verhalten, wenn wir sie (als Menschen) aus größerer Entfernung betrachten. Natürlich ist der Bildeindruck aus größerer Entfernung bei allen Verfahren ähnlich, denn Menschen können diese Grauwertfluktuationen auf kleinen Skalen aus größerer Entfernung nicht erkennen. Für den Menschen ist es mehr oder weniger egal, welches der Bilder ihm aus großer Entfernung präsentiert wird, da er sich von dem Bild einen globalen Eindruck verschaffen kann. Übliche Bildverarbeitungsverfahren jedoch arbeiten lokal und sind (sehr) sensitiv für Grauwertänderungen auf kleinen Skalen. Sie sind also in der Regel sehr sensitiv für die durch die Vergleichsverfahren eingeführten Kompressionsartefakte, die z.B. im Testbild SAILBOAT in den Bäumen im Vordergrund gut erkennbar sind. Noch besser erkennbar sind die Unterschiede an den Ausschnittsvergrößerungen des Segelboots (Größe 65×100 Pixel).

Wenn man Bilder kompakt speichern will, um sie später weiteren Bildverarbeitungsprozessen, im Sinne von Bilderkennung, zu unterziehen, so ist es nicht sinnvoll, sie unbearbeitet mit den Standardverfahren zur Bilddatenkompression zu speichern, denn diese Art der Kompression erzeugt. Diese erschweren die spätere Weiterverarbeitung.

Wenn jetzt der Eindruck entstanden ist, daß die Vorteile unserer Kompressionsmethode einzig auf der Abstraktion (d.h Vorverarbeitung mit dem Abstraktor A_σ und anschließender Interpolation) beruhen, so ist das falsch. Um dieses zu zeigen, haben wir mit den Verfahren JPEG und EPIC unsere Rekonstruktion 3.48 und nicht das Original auf etwa die Länge, die wir mit unserer Kompression erreichen, komprimiert (siehe Tabelle 3.6). Die entstandenen Bilder (JPEG: Abbildung 3.56; Epic: Abbildung 3.57) sowie die zugehörigen Ausschnittsvergrößerungen zeigen auch hier die typischen Artefakte der jeweiligen Kompressionsmethode.

Abbildung 3.47: Original des Testbilds SAILBOAT.

Abbildung 3.48: SAILBOAT komprimiert mit KdA (4.66%).

Abbildung 3.49: SAILBOAT komprimiert mit JPEG (4.85%).

Abbildung 3.50: SAILBOAT komprimiert mit EPIC (4.81%).

Abbildung 3.51: Sailboat komprimiert mit YF (4.51%).

Wir sehen also, daß die vorher angestellten Überlegungen zur Kompression unserer Bildrepräsentation (Grauwert- und Kanteninformation) nicht ein hübsches Beiwerk, sondern essentieller Bestandteil einer Kompressionsmethode sind, die es ermöglicht, die dekomprimierten Bilder sinnvoll mit Bildverarbeitungsprozessen weiterzubearbeiten.

Abbildung 3.52: Original des Testbilds SAILBOAT (Ausschnitt).

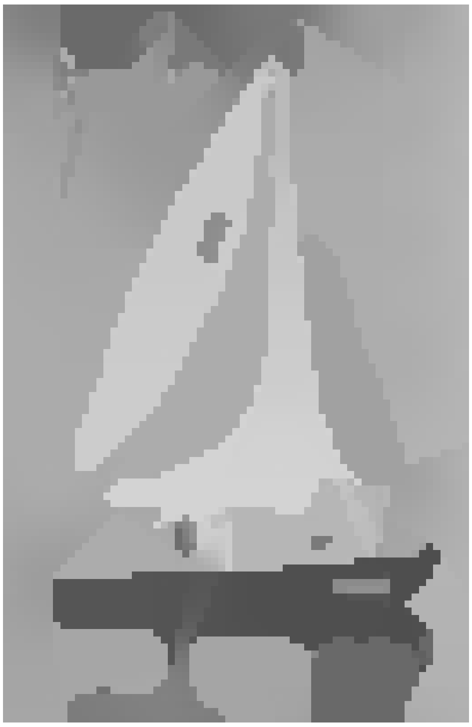


Abbildung 3.53: SAILBOAT komprimiert mit KdA (Ausschnitt).

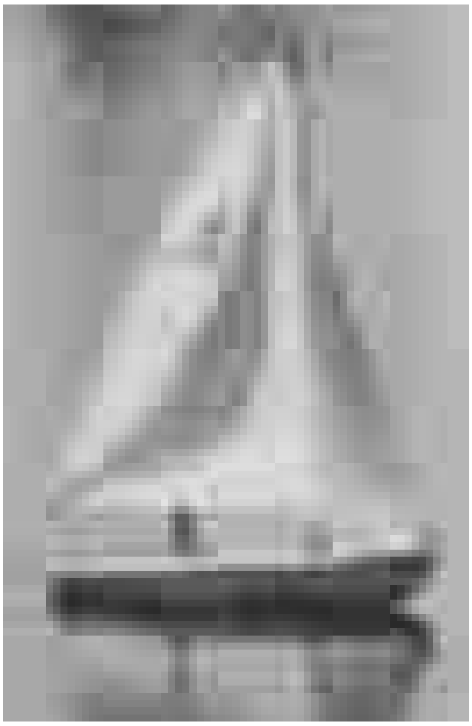


Abbildung 3.54: SAILBOAT komprimiert mit JPEG (Ausschnitt).

Abbildung 3.55: SAILBOAT komprimiert mit EPIC (Ausschnitt).

Abbildung 3.56: SAILBOAT Rekonstruktion komprimiert mit JPEG.

Abbildung 3.57: SAILBOAT Rekonstruktion komprimiert mit EPIC.

Kompressionsverfahren (Implementierung)	Länge der Datei in byte	in %	Abbildung Nr.	Seite
Original	262144	100	3.47	122
KdA	12216	4.66	3.48	122
Transformationskodierung (JPEG)	12707	4.85	3.49	123
WaveletKodierung (EPIC)	12600	4.81	3.50	123
Fraktale Bildkompression (YF)	11817	4.51	3.51	124
Rekonstruktion mit JPEG	12577	4.80	3.56	126
Rekonstruktion mit EPIC	12652	4.83	3.57	126

Tabelle 3.6: Dateilängen für das Testbild SB.

Abbildung 3.58: SAILBOAT Rekonstruktion komprimiert mit JPEG (Ausschnitt).

Abbildung 3.59: SAILBOAT Rekonstruktion komprimiert mit EPIC (Ausschnitt).

Anhang A

Die Testbilder

Auf den folgenden Seiten sind alle Testbilder zu finden, die in der Arbeit benutzt wurden. Es werden jeweils das Original des Testbilds oben und die ‚abstrahierte' Version unten auf einer Seite dargestellt.

Das abstrahierte Bild wurde über den in Kapitel 1 beschriebenen Prozeß der Gewinnung von Grauwert- und Kanteninformation, zusammen mit der in Kapitel 2 beschriebenen Interpolation gewonnen. Der bei der Filterung mit dem Abstraktor A_σ benutzte Parameter σ ist für jedes Bild in den Bildunterschriften angegeben.

Zur Rekonstruktion wurde die automatisch erzeugte Kanteninformation, sowie 64×64 Abtastwerte bei den 512×512 Pixel großen Bildern und 32×32 Abtastwerte bei den beiden 256×256 Pixel großen Bildern benutzt. (Siehe auch Tabelle 3.1 auf Seite 81.) Das Interpolationsverfahren arbeitet mit der in Kapitel 2 angegebenen Standardparameterwahl. Die zur Rekonstruktion benötigte Datenrate ist in ‚bits per pixel (bpp)', sowie in % der Dateilänge des Originals angegeben.

Abbildung A.1: Original des Testbilds Nr. 1: AIRPLANE.

Abbildung A.2: Abstraktion AIRPLANE, $\sigma = 0.5$, Datenrate 0.29 bpp $\hat{=}$ 3.64%

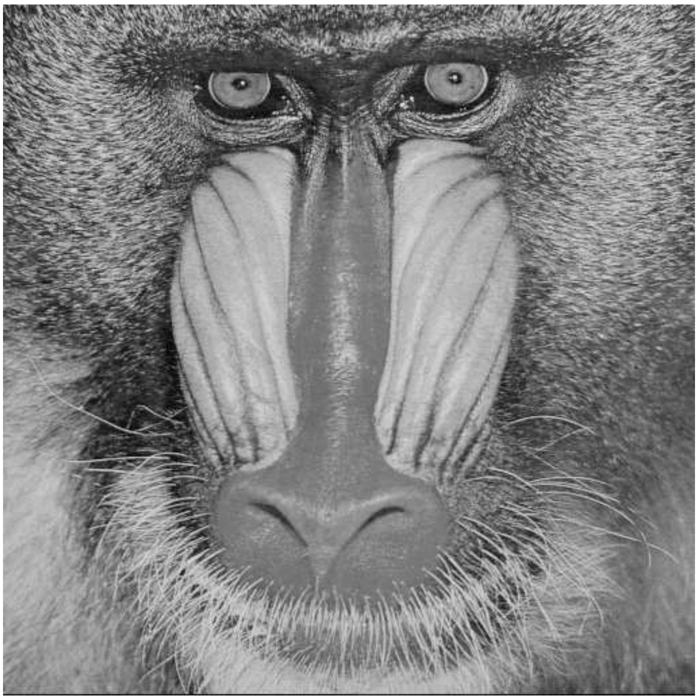


Abbildung A.3: Original des Testbilds Nr. 2: BABOON.

Abbildung A.4: Abstraktion BABOON, $\sigma = 1.5$, Datenrate 0.36 bpp $\hat{=}$ 4.54%

Abbildung A.5: Original des Testbilds Nr. 3: BETON.

Abbildung A.6: Abstraktion BETON, $\sigma = 1.5$, Datenrate 0.26 bpp $\hat{=}$ 3.23%

Abbildung A.7: Original des Testbilds Nr. 4: CASINO.

Abbildung A.8: Abstraktion CASINO, $\sigma = 1.0$, Datenrate 0.22 bpp $\hat{=}$ 2.81%

Abbildung A.9: Original des Testbilds Nr. 5: COLLIE.

Abbildung A.10: Abstraktion COLLIE, $\sigma = 1.0$, Datenrate 0.53 bpp $\hat{=}$ 6.64%

Abbildung A.11: Original des Testbilds Nr. 6: Fanta.

Abbildung A.12: Abstraktion Fanta, $\sigma = 0.5$, Datenrate 0.17 bpp $\hat{=}$ 2.09%

Abbildung A.13: Original des Testbilds Nr. 7: GABI.

Abbildung A.14: Abstraktion GABI, $\sigma = 1.5$, Datenrate 0.14 bpp $\hat{=}$ 1.74%

Abbildung A.15: Original des Testbilds Nr. 8: GRUNDIG.

Abbildung A.16: Abstraktion GRUNDIG, $\sigma = 1.5$, Datenrate 0.08 bpp $\hat{=}$ 0.94%

Abbildung A.17: Original des Testbilds Nr. 9: HALLE.

Abbildung A.18: Abstraktion HALLE, $\sigma = 0.5$, Datenrate 0.31 bpp $\hat{=}$ 3.87%

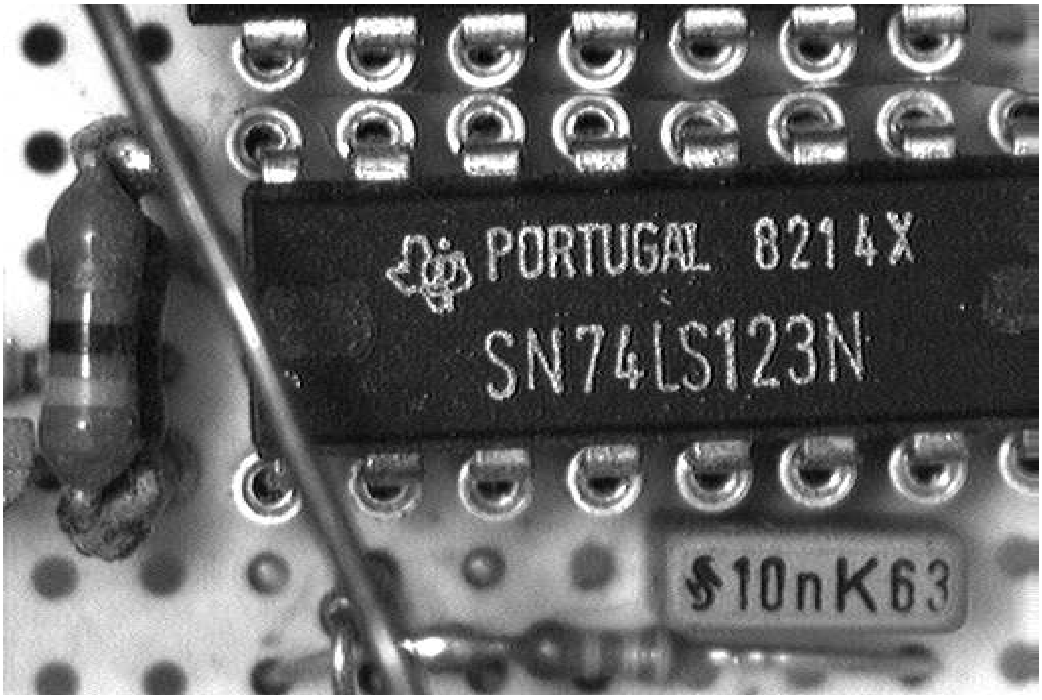


Abbildung A.19: Original des Testbilds Nr. 10: IC.

Abbildung A.20: Abstraktion IC, $\sigma = 1.5$, Datenrate 0.38 bpp $\hat{=}$ 4.75%

Abbildung A.21: Original des Testbilds Nr. 11: LENA.

Abbildung A.22: Abstraktion LENA, $\sigma = 1.0$, Datenrate 0.32 bpp $\hat{=}$ 4.01%

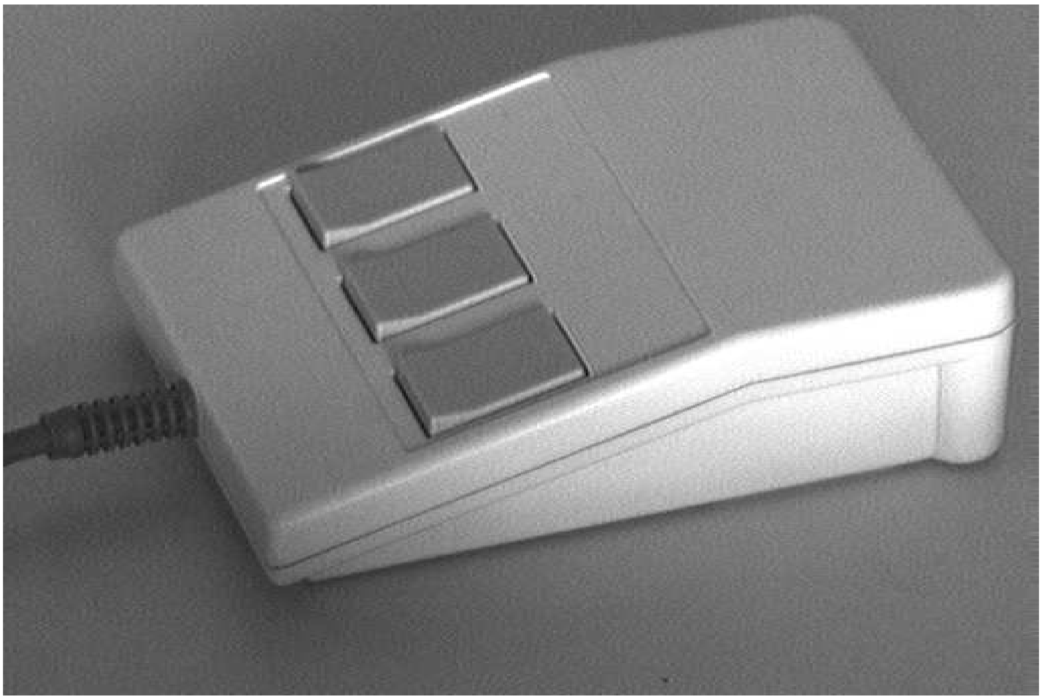


Abbildung A.23: Original des Testbilds Nr. 12: MAUS.

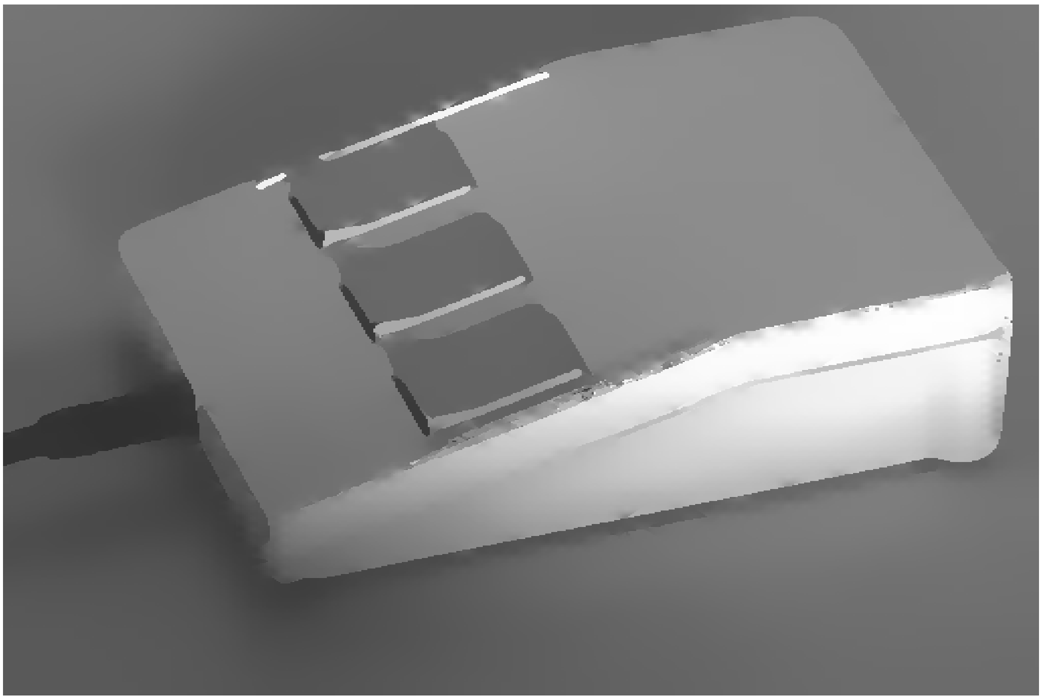


Abbildung A.24: Abstraktion MAUS, $\sigma = 1.5$, Datenrate 0.11 bpp $\hat{=}$ 1.43%

Abbildung A.25: Original des Testbilds Nr. 13: PEPPERS.

Abbildung A.26: Abstraktion PEPPERS, $\sigma = 1.0$, Datenrate 0.32 bpp $\hat{=}$ 3.94%

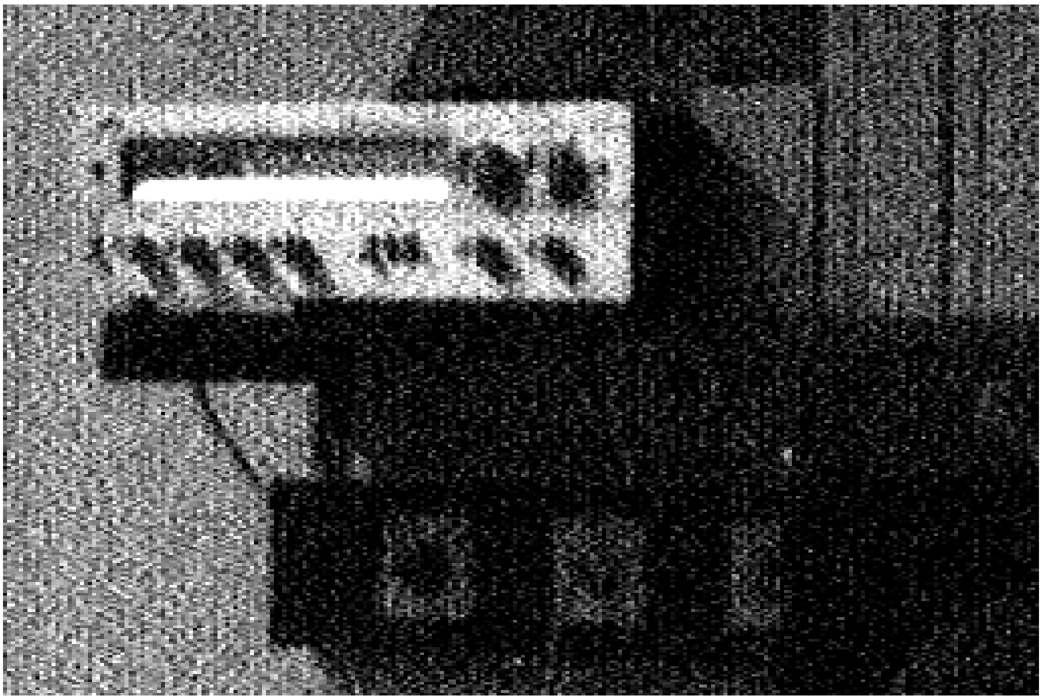


Abbildung A.27: Original des Testbilds Nr. 14: Radio.

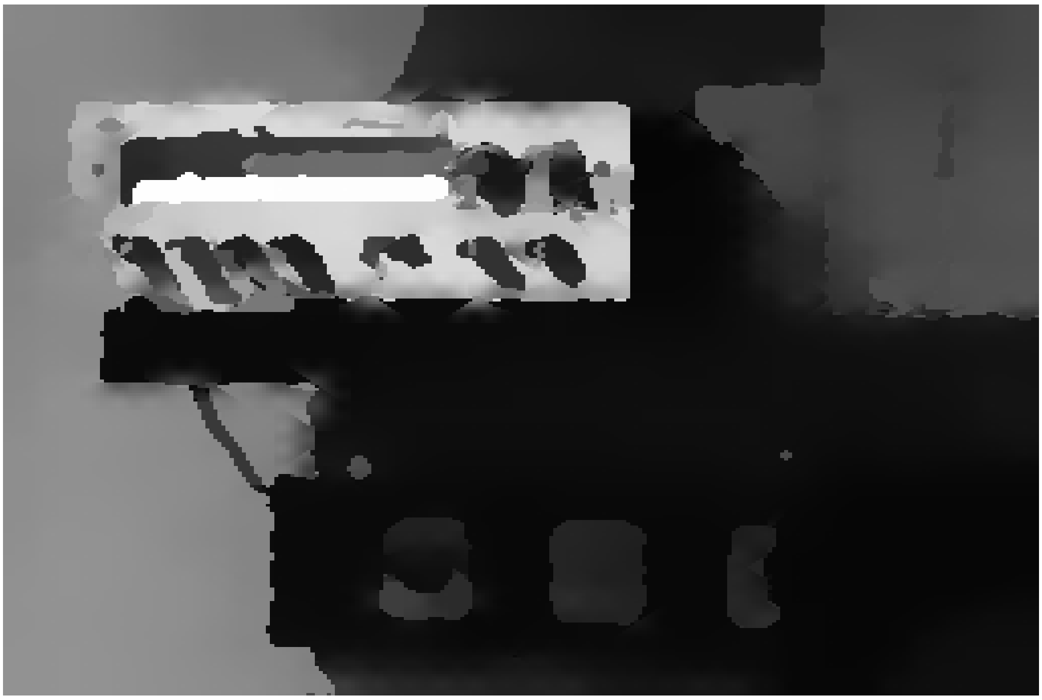


Abbildung A.28: Abstraktion Radio, $\sigma = 1.5$, Datenrate 0.24 bpp $\hat{=}$ 2.96%

Abbildung A.29: Original des Testbilds Nr. 15: SAILBOAT.

Abbildung A.30: Abstraktion SAILBOAT, $\sigma = 1.0$, Datenrate 0.37 bpp $\hat{=}$ 4.64%

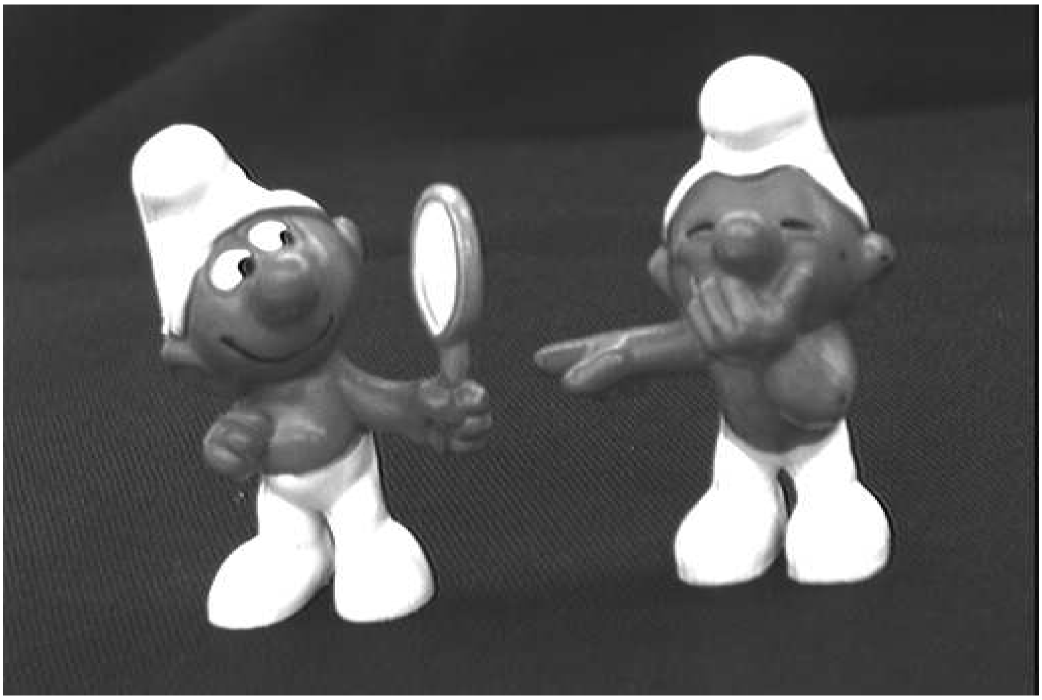


Abbildung A.31: Original des Testbilds Nr. 16: SCHLUMPF.

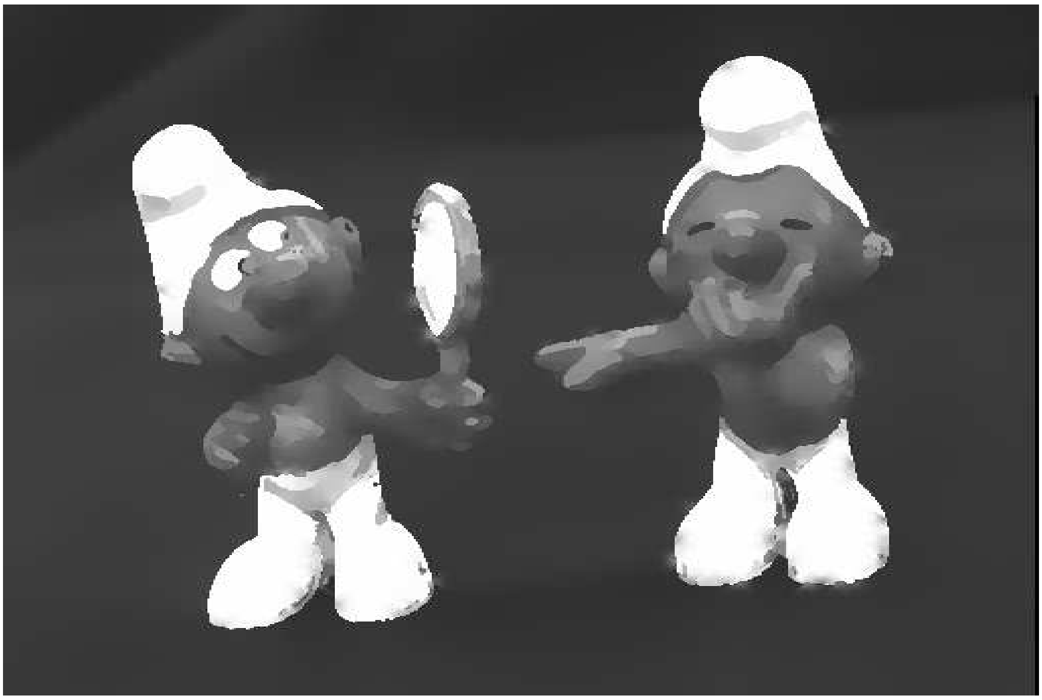


Abbildung A.32: Abstraktion SCHLUMPF, $\sigma = 0.5$, Datenrate 0.15 bpp $\hat{=}$ 1.93%

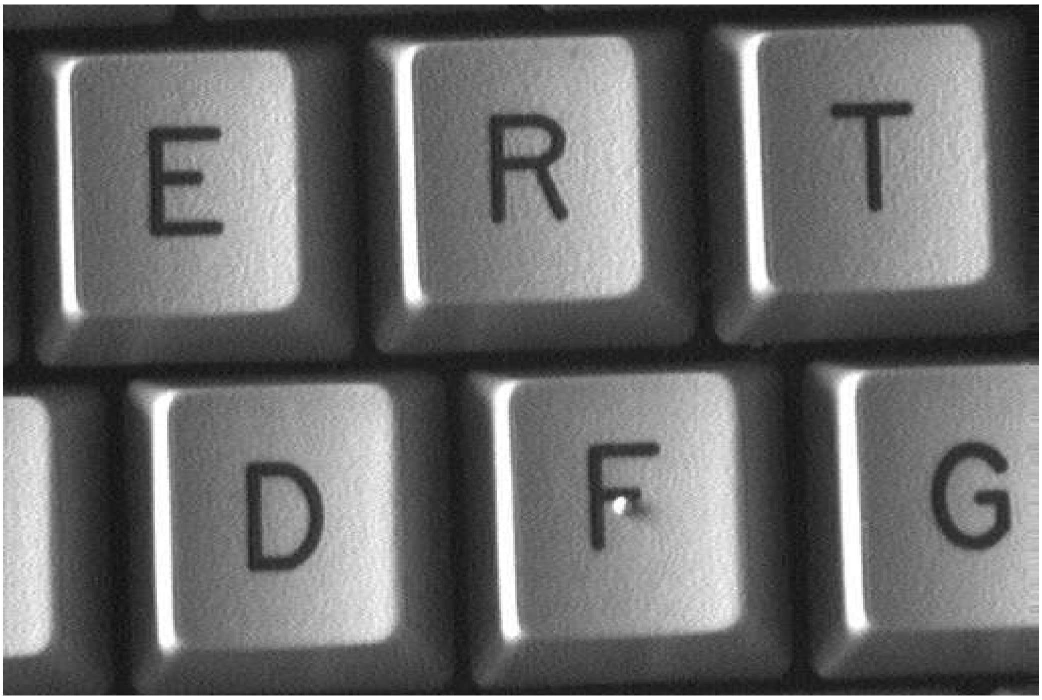


Abbildung A.33: Original des Testbilds Nr. 17: TASTATUR.

Abbildung A.34: Abstraktion TASTATUR, $\sigma = 1.5$, Datenrate 0.16 bpp $\hat{=}$ 2.02%

Abbildung A.35: Original des Testbilds Nr. 18: TELEFON.

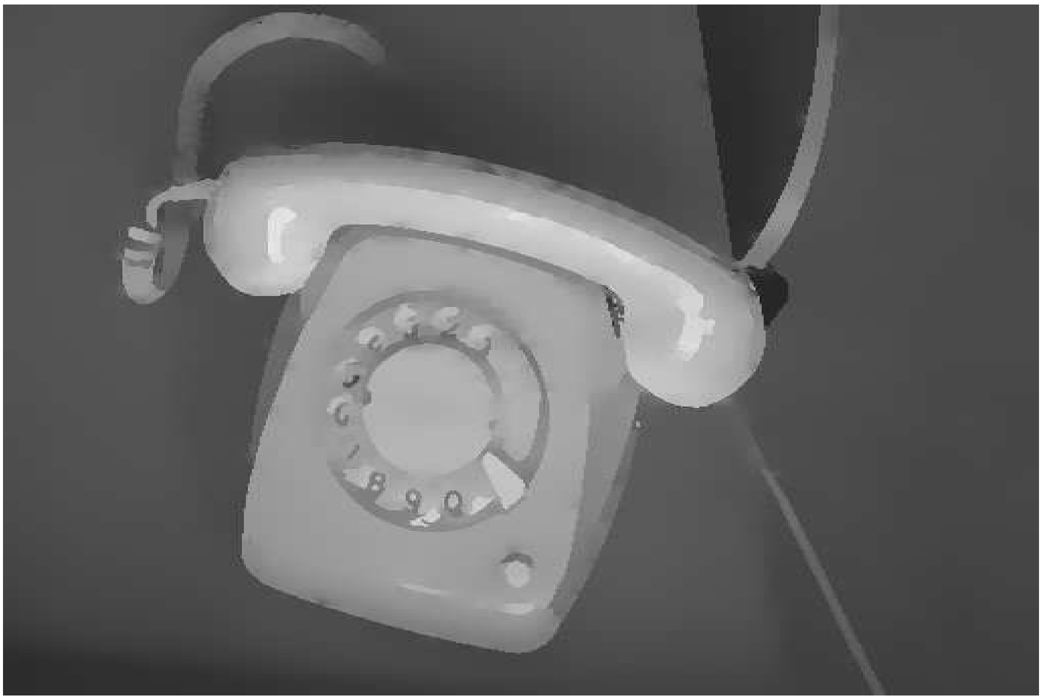


Abbildung A.36: Abstraktion TELEFON, $\sigma = 1.0$, Datenrate 0.17 bpp $\hat{=}$ 2.07%

Abbildung A.37: Original des Testbilds Nr. 19: TIFFANY.

Abbildung A.38: Abstraktion TIFFANY, $\sigma = 1.0$, Datenrate 0.24 bpp $\hat{=}$ 2.94%

Anhang B

A_σ Sequenzen

Auf den folgenden Seiten sind A_σ-Filtersequenzen zu finden. Es wurden die drei Testbilder BABOON, RADIO und CASINO mit dem Abstraktor A_σ (siehe Seite 19) gefiltert. Es wurden die Parameterwerte $\sigma = 0.5$, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.5 und 3.0 verwendet.

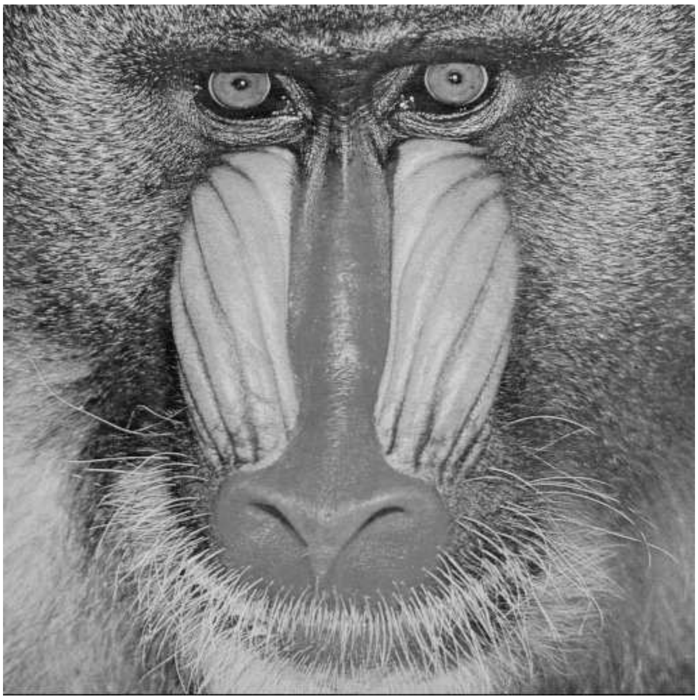


Abbildung B.1: Original des Testbilds BABOON.

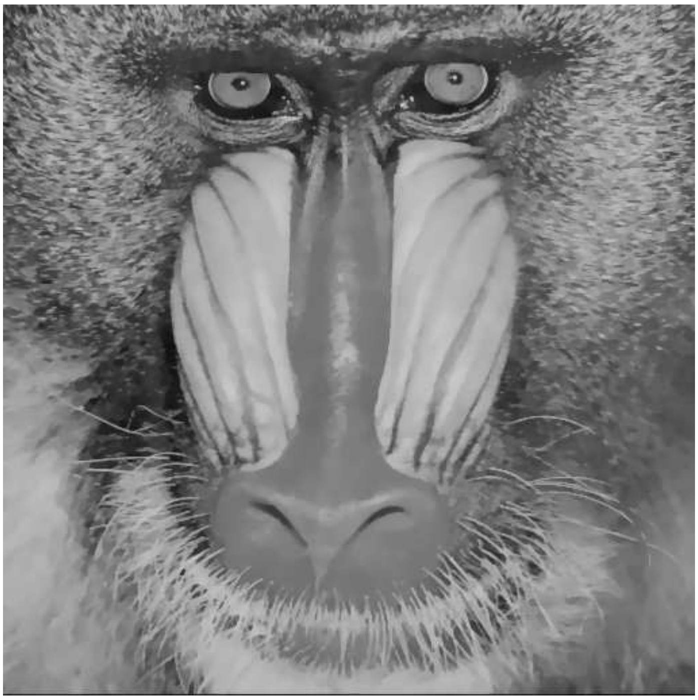


Abbildung B.2: BABOON gefiltert mit $A_{0.5}$.

Abbildung B.3: BABOON gefiltert mit $A_{0.75}$.

Abbildung B.4: BABOON gefiltert mit $A_{1.0}$.

Abbildung B.5: BABOON gefiltert mit $A_{1.25}$.

Abbildung B.6: BABOON gefiltert mit $A_{1.5}$.

Abbildung B.7: BABOON gefiltert mit $A_{1.75}$.

Abbildung B.8: BABOON gefiltert mit $A_{2.0}$.

Abbildung B.9: BABOON gefiltert mit $A_{2.5}$.

Abbildung B.10: BABOON gefiltert mit $A_{3.0}$.

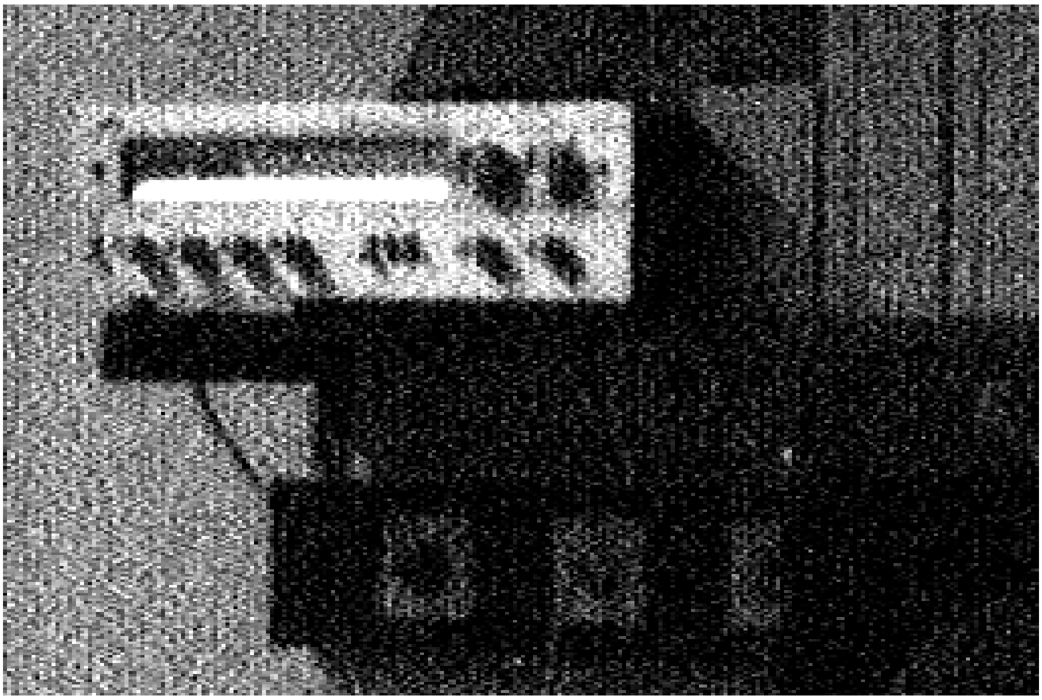


Abbildung B.11: Original des Testbilds RADIO.

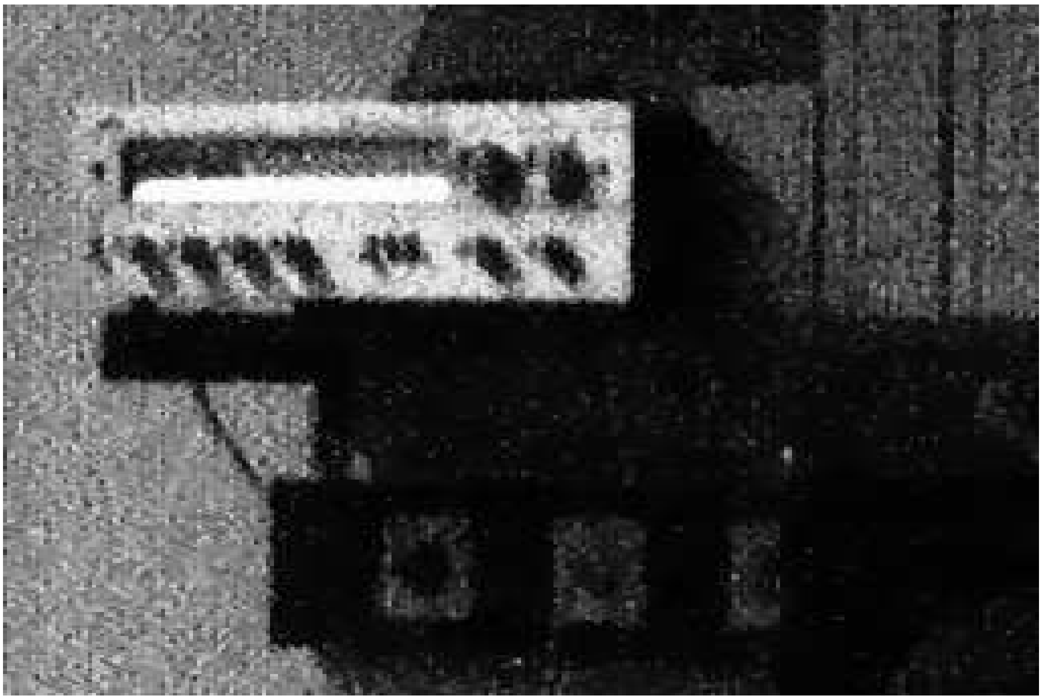


Abbildung B.12: RADIO gefiltert mit $A_{0.5}$.

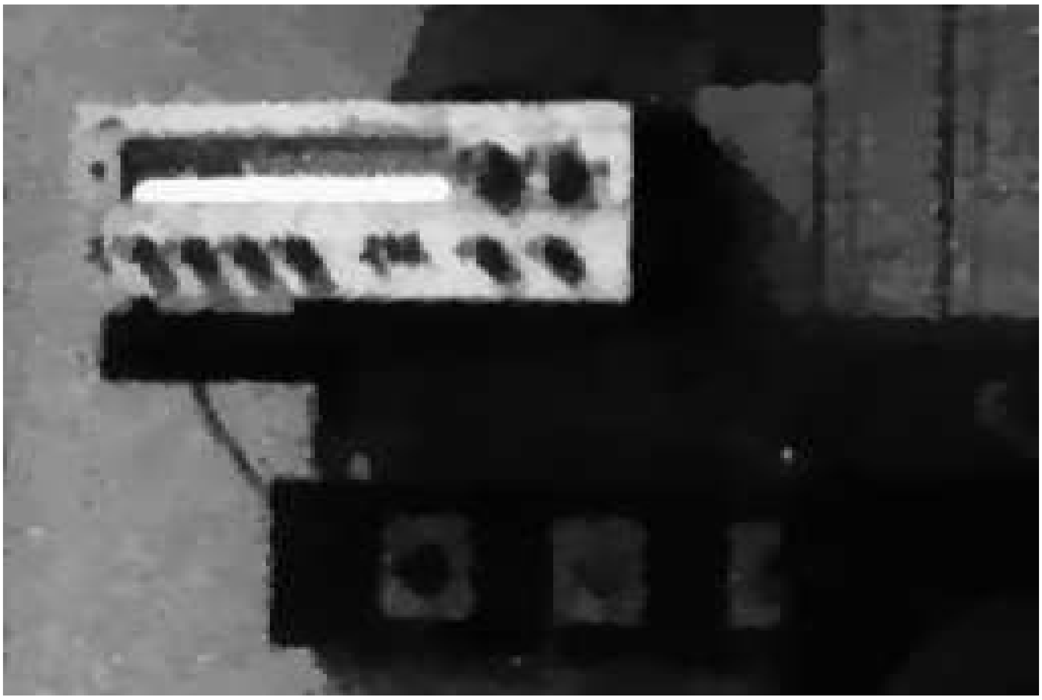


Abbildung B.13: RADIO gefiltert mit $A_{0.75}$.

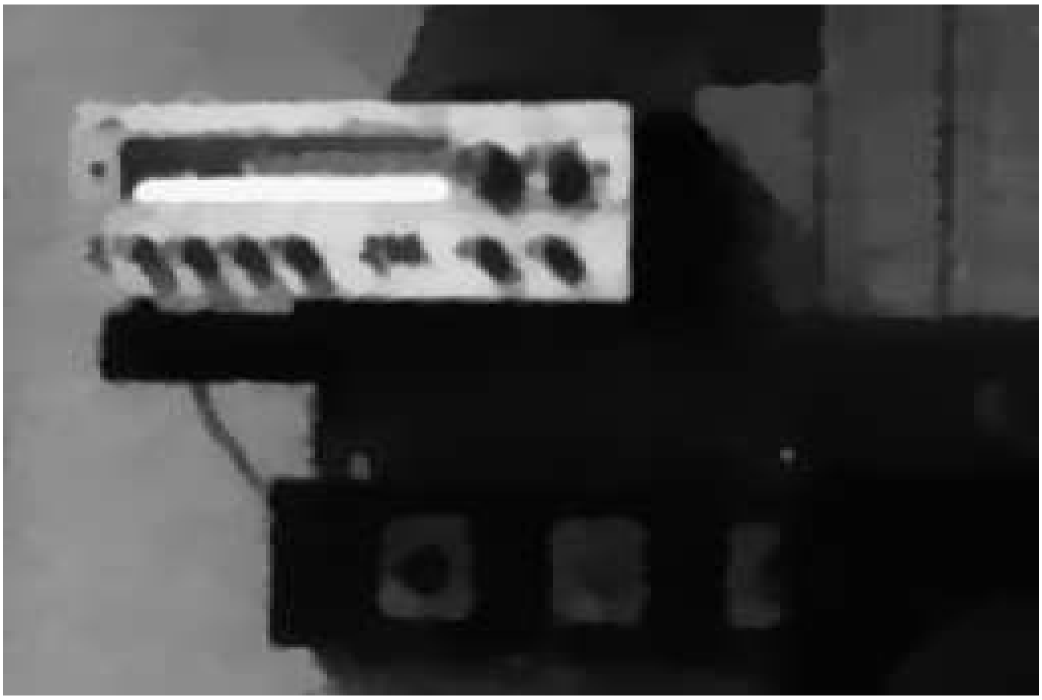


Abbildung B.14: RADIO gefiltert mit $A_{1.0}$.

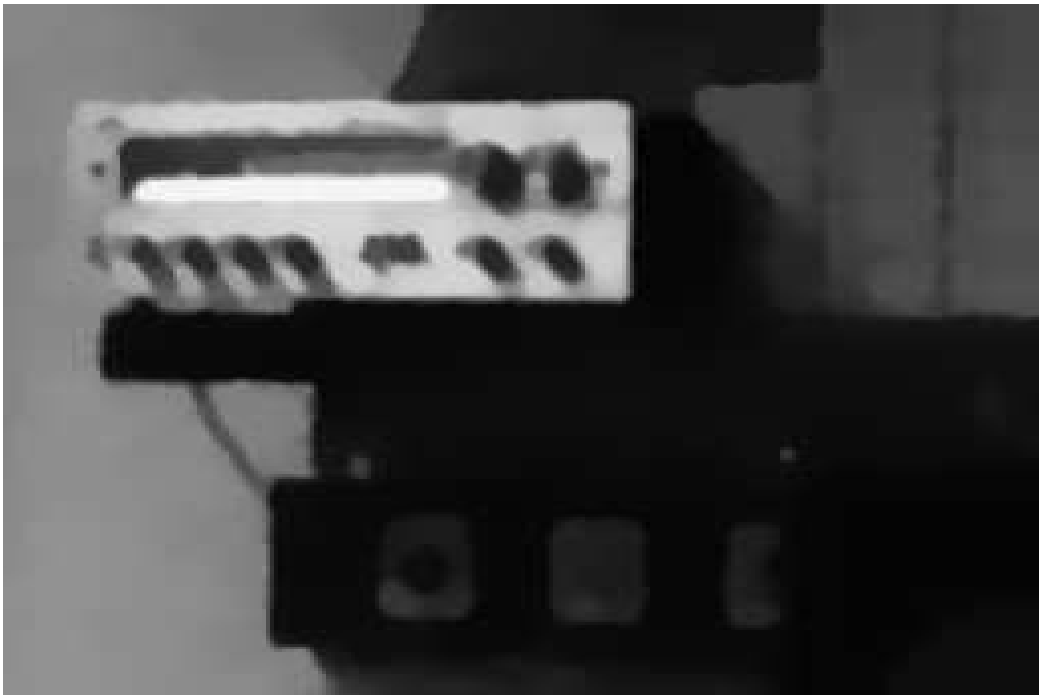


Abbildung B.15: RADIO gefiltert mit $A_{1.25}$.

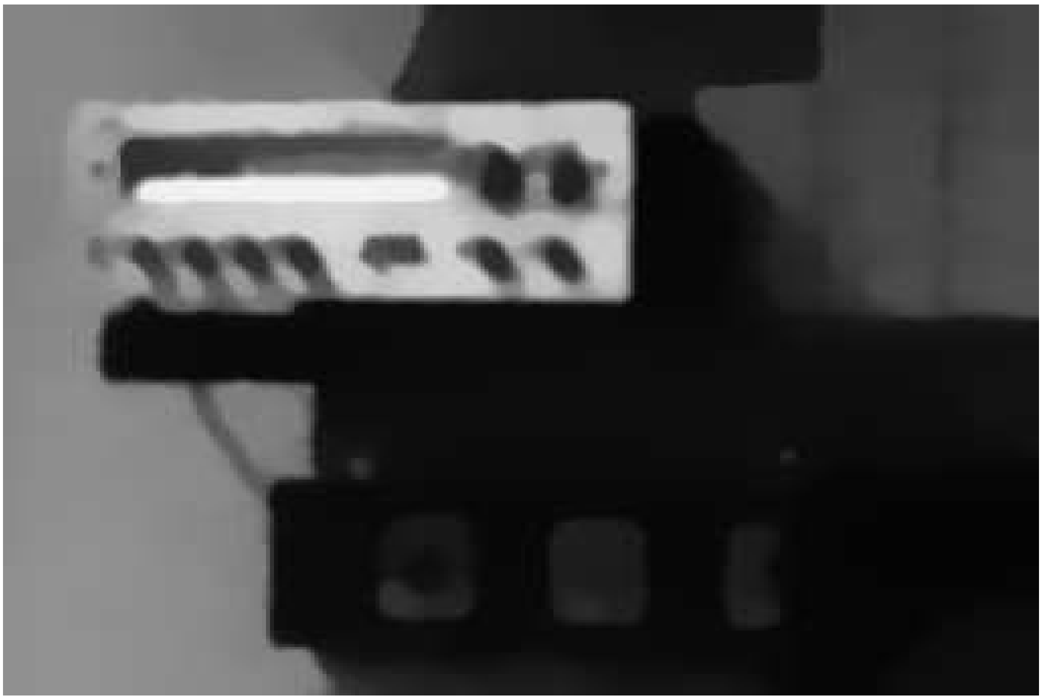


Abbildung B.16: RADIO gefiltert mit $A_{1.5}$.

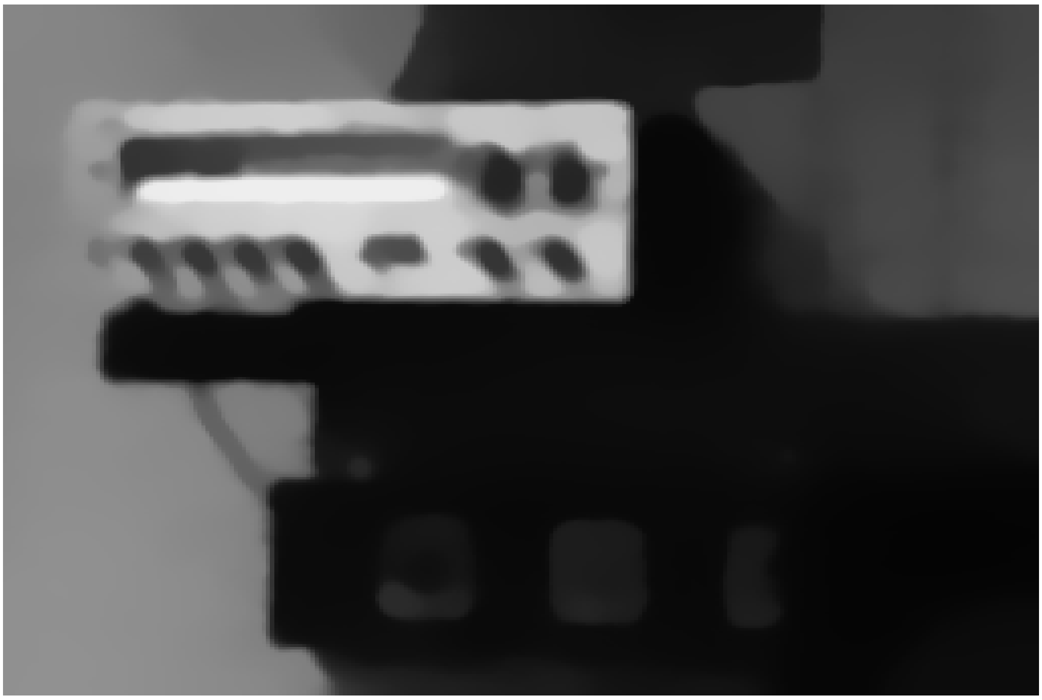


Abbildung B.17: RADIO gefiltert mit $A_{1.75}$.

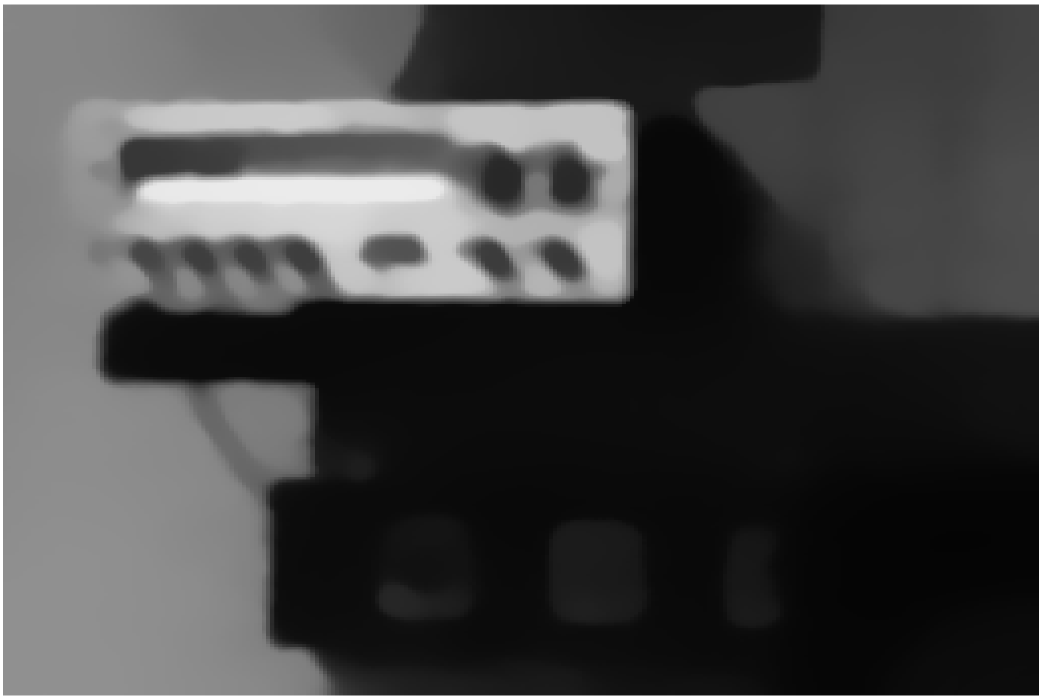


Abbildung B.18: RADIO gefiltert mit $A_{2.0}$.

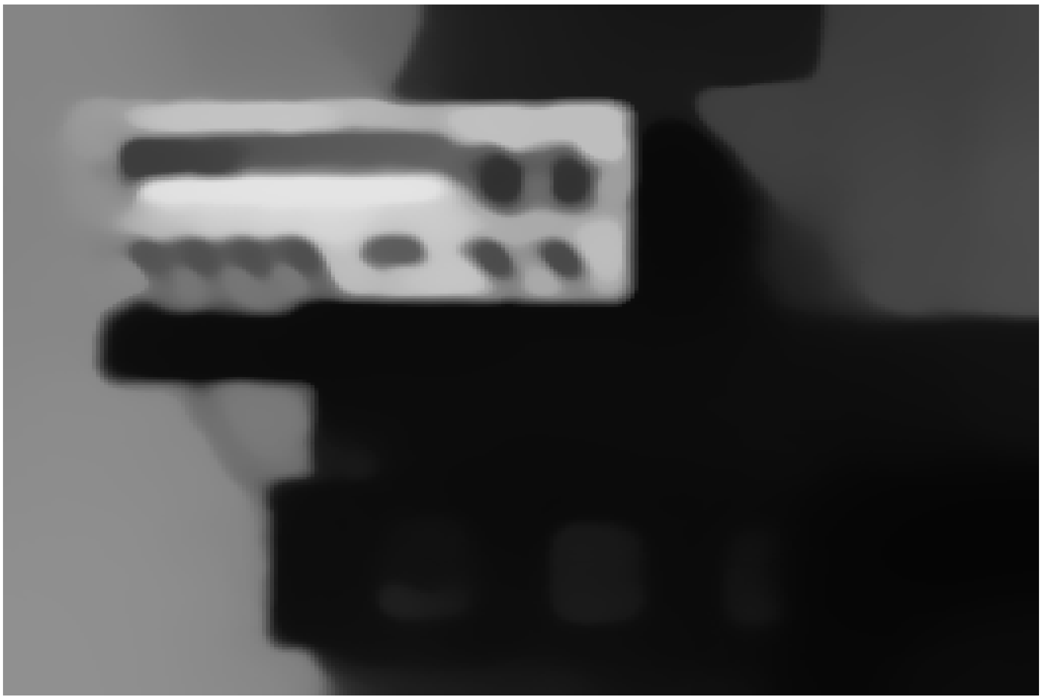


Abbildung B.19: RADIO gefiltert mit $A_{2.5}$.

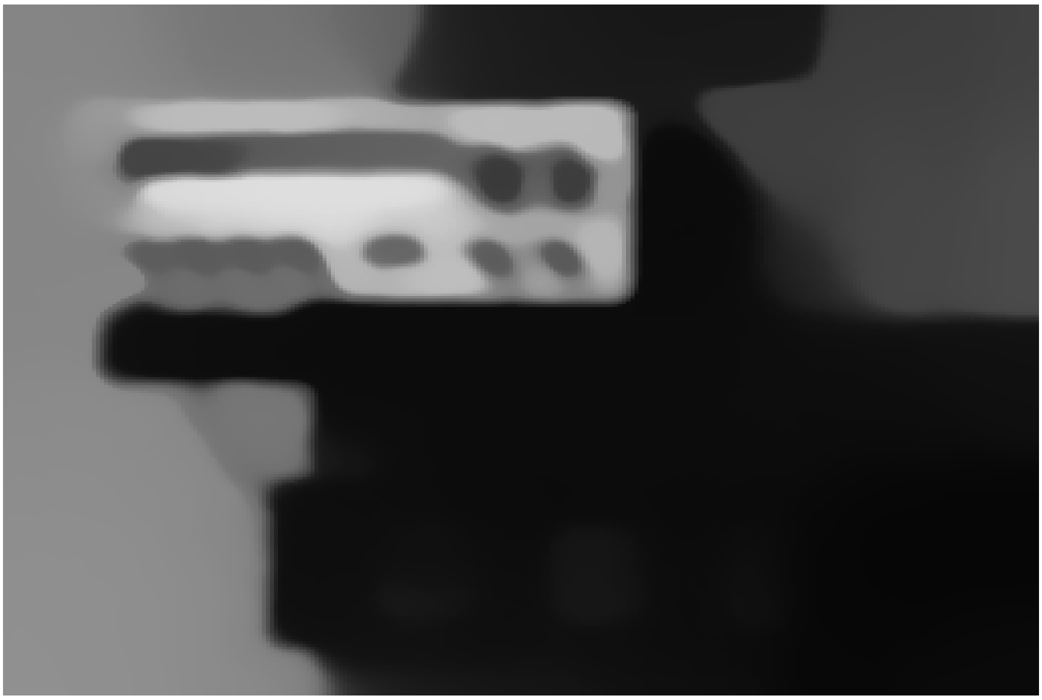


Abbildung B.20: RADIO gefiltert mit $A_{3.0}$.

Abbildung B.21: Original des Testbilds CASINO.

Abbildung B.22: CASINO gefiltert mit $A_{0.5}$.

Abbildung B.23: Casino gefiltert mit $A_{0.75}$.

Abbildung B.24: Casino gefiltert mit $A_{1.0}$.

Abbildung B.25: CASINO gefiltert mit $A_{1.25}$.

Abbildung B.26: CASINO gefiltert mit $A_{1.5}$.

Abbildung B.27: CASINO gefiltert mit $A_{1.75}$.

Abbildung B.28: CASINO gefiltert mit $A_{2.0}$.

Abbildung B.29: CASINO gefiltert mit $A_{2.5}$.

Abbildung B.30: CASINO gefiltert mit $A_{3.0}$.

Anhang C

Hinweise zur Druckqualität

In dieser Arbeit sind viele Bilder wiedergegeben. Die Aussagekraft der Bilder hängt sehr stark von der Druckqualität ab, mit der diese Arbeit ausgedruckt wird. Da die Arbeit auch als Postscript-Datei zu Verfügung gestellt werden soll, kann der Autor nicht für die Druckqualität jedes einzelnen Exemplars dieser Arbeit garantieren.

Um die Qualität der abgedruckten Bilder einschätzen zu können, dienen die folgenden zwei Abbildungen. An ihnen kann jeder die Qualität der Druckausgabe abschätzen.

Das Bild GRAUWERTRAMPE zeigt eine Grauwertrampe. Es enthält im Original, d.h. in der zugehörigen Postscript-Datei 256 verschiedene Grauwerte. In der Druckwiedergabe erscheint diese Rampe mehr oder weniger gestuft. Dieses ist darauf zurückzuführen, daß die meisten Postscript-Drucker keine 256 Graustufen darstellen können und folglich die 256 Graustufen auf die durch den Drucker darstellbaren Graustufen (meist zwischen 50 und 100) reduzieren.

Das Bild KREISE zeigt zwei kleine schwarze Kreise (Grauwert 255) auf einem homogenen grauen Hintergrund (Grauwert 128). Dieses Bild zeigt in der Druckwiedergabe meist keinen homogenen Hintergrund, welches meist in der Tonerzuführung des Druckers begründet liegt. Vereinzelt tauchen weitere Kreise oder Schatten von Kreisen auf; dieses meist bei Laserdruckern, deren Druckwalze nicht vollständig entladen wird.

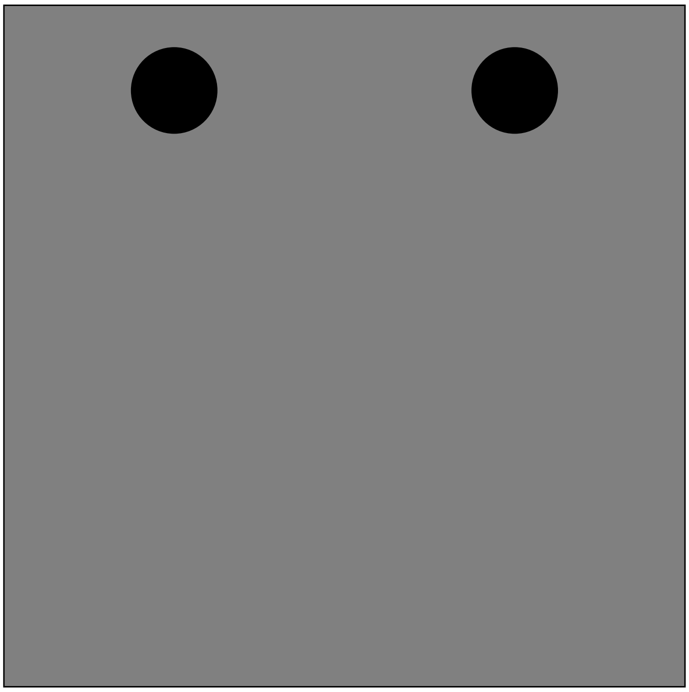


Abbildung C.1: Kreise.

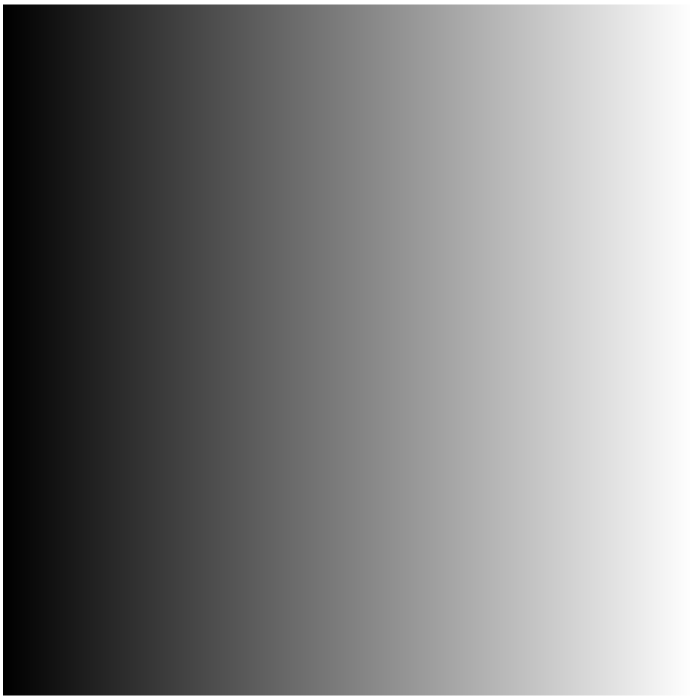


Abbildung C.2: Grauwertrampe.

Literaturverzeichnis

[AS90] ADELSON, EDWARD H.; SIMONCELLI, EERO P. *Subband Image Coding with Three-tap pyramids.* Picture Coding Symposium 1990, Cambridge MA.

[ASH87] ADELSON, EDWARD H.; SIMONCELLI, EERO P.; HINGORANI, RAJESH. *Orthogonal pyramid transforms for image coding.* Proc. SPIE Visual Commun. and Image Proc. II, Vol. 845, Cambridge MA 1987.

[AW95] AURICH, VOLKER; WEULE, JÖRG. *Non-Linear Gaussian Filters Performing Edge Preserving Diffusion.* In: Mustererkennung 1995. Springer Verlag 1995 (Reihe: Informatik Aktuell). (Proc. 17. DAGM-Symposium.)

[Ba77] BARNHILL, R.E. *Representation and approximation of surfaces.* In [Ri77, S. 69–120].

[BDL83] BARNHILL, R.E.; DUBE, R.P.; LITTLE, F.F. *Properties of Shepard's Surface.* Rocky Mountain Journal of Mathematics, Vol. 13, No. 2, 1983, S. 365–382.

[BH92] BARNSLEY, MICHAEL F.; HURD, LYMAN P. *Fractal Image Compression.* AK Peters Ltd, Wessley, Massachusetts, 1992.

[BZ87] BLAKE, ANDREW; ZISSERMAN, ANDREW. *Visual Rekonstruktion.* MIT Press, Cambridge, 1987.

[BrLo88] BROOMHEAD, D.S.; LOWE, D. *Multivariable Functional Interpolation and Adaptive Networks.* Complex Systems 2 (1988), S. 321–355.

[CH84] CORMACK GORDON V.; HORSPOOL R. NIGEL. *Algorithms for adaptive Huffman codes.* Information Processing Letters, Vol. 18, No. 3, März 1984, S. 159–165.

[Co87] COMAN, GH. *The remainder of certain Shepard type interpolation formulas.* Stud. Univ. Babes-Bolyai, Math. 32, No 4 (1987), S. 24–32.

[CoTa88] COMAN, GH.; TABULEA, L. *A Shepard-Taylor approximation formula.* Studia Univ. Babes-Bolyai, Math. 33, No 3, 1988, S. 65–73.

[DMT90] DELLA VECCHIA; MASTROIANNI; TOTIH. *Saturation of Shepard operators.* Approximation Theory and its Applications, Vol. 6, No 4, Dezember 1990, S. 76–84.

[FAQ] *Frequently Asked Questions* der Newsgroup `comp.compression`. Betreut von JEAN-LOUP GAILLY (`jloup@chorus.fr.`). Erhältlich z.B. über `ftp://rtfm.mit.edu/pub/usenet/news.answers/...`

[Fa86] FARWIG, REINHARD. *Rate of Convergence of Shepard's Global Interpolation Formula.* Mathematics of Computation, Vol. 46, No. 174, April 1986, S. 577–590.

[Fi92] FISHER, YUVAL. *Fractal image compression.* In SIGGRAPH' 92, 1992, Fractal Course Notes.

[Fi94] FISHER, YUVAL. *Fractal Image Compression: Theory and Application* Springer Verlag 1994.

[FJB92] FISHER, YUVAL; JACOBS, E.W.; BOSS, R.D. *Fractal Image Coding Using Iterated Transforms.* In [St92]

[Fr82] FRANKE, R. *Scattered Data Interpolation: Tests of Some Methods.* Mathematics of Computation, Vol. 38, No. 157, Januar 1982, S. 181–200.

[GoWi78] GORDON, W.J.; WIXOM, J.A. *Shepard's Method of 'Metric Interpolation' to Bivariate and Multivariate Interpolation.* Mathematics of Computation, Vol. 32, No. 141, Januar 1978, S. 253–264.

[GW92] GONZALES, RAFAEL C.; WOODS, RICHARD E. *Digital Image Processing.* Addison-Wesley, 1992.

[HL92] HIRSCHBERG, DANIEL S.; LELEWER, DEBRA A. *Context Modeling for Text Compression.* In [St92]

[HV92] HOWARD, PAUL G.; VITTER, JEFFREY SCOTT. *Practical Implementations of Arithmetic Coding.* In [St92]

[Jo88] JONES, DOUGLAS W. *Application of Splay Trees to Data Compression.* Communications of the ACM, Vol. 31, No. 8, August. 1988, S. 996–1007.

[LCS76] LORENTZ, G.G.; CHUI, C.K.; SCHUMAKER, L.L. *Approximation Theory II.* Academic Press 1976.

[Le93] LEVI, PAUL. *Lineare Kantenoperatoren und Filter.* in [Ra93]

[LZ77] LEMPEL, A.; ZIV, J. *A universal algorithm for sequential data compression.* IEEE Transaction on Information Theory. Vol. 23, No. 3, Mai 1977, S. 337–343.

[Ma82] MARR, DAVID. *Vision. A Computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman and Company, 1982.

[Ma89] MALLET, STEPHANE G. *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 11, No. 7, Juli 1989, S. 674–693.

[MH80] MARR, DAVID; HILDERTH, E. *Theory of edge detection.* Proc. R. Soc. London, B 207, 1980, S. 187–217.

[MoDa88] MOODY, JOHN; DARKEN, CHRISTIAN J. *Learning with Localized Receptive Fields.* In [THS88]

[MoDa89] MOODY, JOHN; DARKEN, CHRISTIAN J. *Fast learning in Networks of Locally-Tuned Processing Units.* Neural Computation 1, 1989, S. 281–294.

[MP85] MCINTYRE DAVID R.; PECHURA MICHAEL A. *Data Compression using static Huffman code-decode tables.* Communications of the ACM, Vol. 28, No. 6, Juni 1985, S. 612–616.

[Ne92] NELSON, MARK. *The Data Compression Book.* M & T Books, San Mateo 1992.

[OW93] OTTMANN, T.; WIDMAYER, P. *Algorithmen und Datenstrukturen.* BI-Wissenschaftsverlag 1993 (Reihe Informatik Band 70).

[PM92] Pennebaker, William B.; Mitchell, Joan L. *JPEG. Still Image Data Compression Standard.* Van Nostrand Reinhold, New York, 1992.

[Ra93] Radig, Bernd (Hrsg.) *Verarbeiten und Verstehen von Bildern.* R. Olbenbourg Verlag, 1993.

[Re88a] Renka, R.J. *Multivariate Interpolation of Large Set of Scattered Data.* ACM Transactions on Mathematical Software, Vol 14, No. 2, Juni 1988, S. 139-148

[Re88b] Renka, R.J. *Algorithm 660. QSHEP2D: Quadratic Shepard method for bivariate Interpolation of Scattered Data.* ACM Transactions on Mathematical Software, Vol 14, No. 2, Juni 1988, S. 149–150

[Ri77] Rice, John R. *Mathematical Software III.* Academic Press 1977.

[Sa84] Samet, Hanan. *The Quadtree and Related Hierarchical Datastructures.* Computing Surveys, Vol. 18, No 2, Juni 1984, S. 187–260.

[Schu76] Schumaker, L.L. *Fitting surfaces to scattered data.* In [LCS76, S. 203–268]

[Sh68] Shepard, Donald *A two dimensional interpolation function for irregulary-spaced data.* Proc. 23rd National Conference ACM, 1968, S. 517–524.

[St92] Stoerer, James A. (Ed.) *Image and Text Compression.* Kluwer Academic Publishers, Boston/Dordrecht/London 1992.

[Sz91] Szabados, J. *Direct and Converse Approximation Theorems for the Shepard Operator.* Approximation Theory and its Applications, Vol. 7, No. 3, September 1991, S. 63–76.

[THS88] Touretzky, Hinton, Sejnowski. *Proceedings of the 1988 Connectionist Models Summer School.* Morgan-Kaufmann, Publishers.

[Wa91] Wallace, Gregory K. *The JPEG Still Picture Compression Standard.* Communications of the ACM, Vol. 34 No. 4, April 1991, S. 31–44.

[We91] WERNTGES, H.W. *Approximation von Vektorfeldern durch Selbstorganisation neuronaler Netze mit Fehlerrückführung am Beispiel der inversen Kinematik eines planaren 4-Gelenk-Arms.* Dissertation Heinrich-Heine-Universität, Düsseldorf 1991.

[Weu94] WEULE, JÖRG. *Iteration nichtlinearer Gauß-Filter in der Bildverarbeitung.* Dissertation Heinrich-Heine-Universität, Düsseldorf 1994.

[WNC87] WITTEN, IAN H.; NEAL, RADFORD M.; CLEARY, JOHN G. *Arithmetic Coding for Data Compression.* Communications of the ACM, Vol. 30, No 6, Juni 1987, S. 520–540.

[Wo90] WOLPERT, DAVID H. *A Mathematical Theory of Generalisation.* Complex Systems 4, 1990, S. 151–249.

[Ze94] ZELL, ANDREAS. *Simulation Neuronaler Netze.* Addison-Wesley, 1994.