Lehrstuhl für Integrierte Systeme
Technische Universität München

Ein Modell zur Beschreibung und Implementierung von Bildanalysefunktionen

Stephan Herrmann

Lehrstuhl für Integrierte Systeme
Technische Universität München

Ein Modell zur Beschreibung und Implementierung von Bildanalysefunktionen

Stephan Herrmann

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. habil. Gerhard Rigoll
Prüfer der Dissertation:
1. Priv.-Doz. Dr.-Ing. habil. Walter Stechele
2. Univ.-Prof. Dr.-Ing. Klaus Diepold

Die Dissertation wurde am 25.08.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 17.02.2005 angenommen.

Vorwort

Die in dieser Dissertation dokumentierte Arbeit ist während meiner Tätigkeit als wissenschaftlicher Assistent am Lehrstuhl für Integrierte Systeme der Technischen Universität München durchgeführt worden. Deshalb gilt mein erster Dank Herrn Prof. Dr.-Ing. I. Ruge, seinem Nachfolger Herrn Prof. Dr. sc. techn. A. Herkersdorf und Herrn PD Dr.-Ing. W. Stechele dafür, dass Sie mir die Möglichkeit gegeben haben, an diesem Lehrstuhl an dem hochinteressanten Thema Videosignalverarbeitung zu arbeiten und zu forschen. Dabei wurde ich stets sowohl administrativ als auch fachlich unterstützt.

Besonders hervorheben möchte ich meinen Dank, dass es mir ermöglicht wurde, auf zahlreichen Treffen (Konferenzen, Projekt- und Standardisierungstreffen) mit internationalen Experten des Arbeitsgebiets meinen Horizont zu erweitern. Hier möchte ich zunächst die Arbeitsgruppe des COST211-Projektes erwähnen, die ein wichtiges Diskussionsforum für meine Arbeit darstellte. Außerdem möchte ich mich für die im Rahmen des europäischen Forschungsprojekts MPEG Fo(u)r Mobiles (M4M) offen geführten Diskussionen mit Herrn Prof. F. Meyer, PhD und Herrn Prof. Dr. J.−C. Klein vom *Centre de Morphologie Mathématique, Ecole des Mines de Paris* bedanken, die gerade in der Startphase meiner Arbeit von großer Bedeutung waren. Genauso gilt mein Dank den Kolleginnen und Kollegen der europäischen Projekte BUSMAN[1] und SCHEMA[2], sowie der MPEG-Arbeitsgruppe.

Außerdem möchte ich auch meinen jetzigen und ehemaligen Kolleginnen und Kollegen am Lehrstuhl für Integrierte Systeme für das sehr angenehme Arbeitsklima und die vielen kritischen und fruchtbaren Diskussionen danken. Dabei möchte ich Dr. Hubert Mooshofer, Dr. Andreas Hutter, Dr. Thomas Wild, Dr. Ulrich Niedermeier, Dr. Nuria Pazos, Raul Medina Beltran de Otalora, Paul Zuber, Rainer Ohlendorf und Jose Martinez hervorheben, die stets bereit waren, ihr Wissen mit ihren Kolleginnen und Kollegen zu teilen.

Schließlich möchte ich auch meine Freunde und Verwandte nennen, die mich während der Durchführung der Arbeit moralisch und aktiv beim Korrekturlesen unterstützten. Dafür möchte ich besonders meiner Frau Sabine, meinen Söhnen Richard und Viktor, meinen Eltern und meinen Freunden Angelika Kammermeier, Nikolai Sefzig und Nicola Flint danken.

[1] Bringing User Satisfaction to Media Access Networks

[2] SCHEMA Network of Excellence on Content-Based Semantic Scene Analysis & Information Retrieval

Kurzfassung

In dieser Arbeit wird ein Modell zur Beschreibung, Implementierung und Analyse von Bildverarbeitungsfunktionen vorgestellt. Dabei liegt der Schwerpunkt des Einsatzgebietes des Modells bei der Entwicklung und Implementierung von Algorithmen und Systemen für Bildanalyseanwendungen. Dazu gehören Anwendungen zur Objektsegmentierung und zur Merkmalextraktion. Die Bildanalyse stellt eine grundlegende Technologie dar, die neue Funktionalitäten im Bereich der Verwendung von multimedialen Daten (insbesondere Videodaten) ermöglicht. Neue Funktionalitäten, die durch Bildanalyse ermöglicht werden, sind u.a. interaktives Video oder die Automatisierung der Erkennung von semantischen Inhalten.

Damit diese Technologien verfügbar werden, müssen Algorithmen sowohl noch entwickelt, als auch in Produkten implementiert werden. Um einen effizienten Entwurf und eine effiziente Umsetzung des Algorithmus zu ermöglichen, müssen folgende Bedingungen erfüllt sein: Die Spezifikation des Algorithmus und seiner Bildverarbeitungsfunktionen sollte möglichst einfach und klar sein, damit Wissen innerhalb und zwischen Gruppen von Forschern und Entwicklern ausgetauscht werden kann. Bei der Implementierung eines funktionalen Modells des Algorithmus ist es anzustreben, dass Software-Module wieder verwendet werden, damit die Funktion des Algorithmus innerhalb kurzer Zeit verifiziert werden kann. Schließlich sollte es möglich sein, bereits während der Software-Algorithmensimulation Komplexitätsabschätzungen durchzuführen, um festzustellen, ob der entwickelte Algorithmus auch sinnvoll implementierbar ist.

Im Bereich von Bibliotheken zum Entwurf der Simulations-Software sind die Voraussetzungen für einen effizienten Entwurf weitgehend erfüllt. Hingegen sind für eine eindeutige und einfache Spezifikation, sowie für eine Abschätzung der Komplexität des Algorithmus keine geeigneten Werkzeuge verfügbar.

Deshalb wurde in dieser Arbeit ein Modell entworfen, mit dem Bildverarbeitungsfunktionen beschrieben werden können. Das Modell erlaubt dabei das Abbilden von Bildverarbeitungsfunktionen, bei denen die Eingangs- und Ausgangsdaten in $2-$dimensionalen Feldern (Bilder) organisiert sind. Das Modell unterteilt in den Adressierungsteil und den Verarbeitungsteil der Bildverarbeitungsfunktionen. Der Adressierungsteil entspricht dabei einem von drei Adressierungstypen, der Inter-Adressierung (zwei Eingangsbildpunkte aus zwei Bildern), der Intra-Adressierung (ein Eingangsbildpunkt und seine Nachbarschaft) oder der rekursiven Adressierung (zur Darstellung von Aus-

breitungsprozessen). Der Adressierungsteil ist dabei charakteristisch für das gesamte Anwendungsgebiet Bildanalyse (bzw. Bildverarbeitung).

Der Verarbeitungsteil gibt der Bildverarbeitungsfunktion ihre spezifischen Eigenschaften. Dieser Teil ist also charakteristisch für eine spezielle Anwendung. Das Modell beschreibt die Schnittstellen der Verarbeitungsfunktionen. Die Funktionalität der Verarbeitungsfunktion selber kann frei gewählt werden und ist daher nicht im Modell enthalten. Das Modell deckt auch Bildverarbeitungsfunktionen ab, bei denen ein Teil der Eingangs- und Ausgangsdaten durch Indizierung adressiert wird. Dabei muss aber für den Ablauf der Funktion die Bildorganisation im Vordergrund stehen. Für Funktionen, bei denen die Daten nicht als Bilder organisiert sind, ist das Modell ungeeignet. Eine weitere Einschränkung des Modells ist, dass jeder Bildpunkt nur einmal verarbeitet wird. Ausnahmen sind bei Ausbreitungsprozessen möglich, aber die Abbildbarkeit der Funktion auf das Modell ist dann nicht mehr gewährleistet.

Die Trennung in Adressierung und Verarbeitung erlaubt es, die Adressierungsfunktionen zentral in einer Bibliothek zu implementieren. Dadurch ergibt sich als wesentliche Eigenschaft des Modells, dass beide Aspekte des Entwurfs, das Designen der Algorithmen (mit Schwerpunkt Qualität) und die Implementierung (mit Blickrichtung Komplexität), effizient abgedeckt werden. Dabei unterstützt die Modellbildung zuerst die Möglichkeit Algorithmen zu beschreiben und eindeutig zu spezifizieren, was die Grundlage für Kommunikation und Nutzung von Synergien ist. Durch eine zentrale Software-Implementierung der Adressierungsfunktionen können diese in Bildverarbeitungsfunktionen wieder verwendet werden. Da die Adressierungsfunktionen den wesentlichen Anteil des Implementierungsaufwands einer Bildverarbeitungsfunktion ausmacht, ist dadurch eine Plattform geschaffen worden, mit der Algorithmen schnell in einem funktionalen Modell implementiert werden können. Durch eine konzepthafte Architekturbeschreibung für Bildverarbeitungsfunktionen und besonders der Adressierungsfunktionen, ist es dann auch möglich, Komplexitätsmerkmale zu identifizieren und bereits während der Entwicklung von Algorithmen zu messen.

An einem Beispielalgorithmus, der einen signifikanten Satz an Methoden zur Objektsegmentierung vereint, wurde nachgewiesen, dass das entwickelte Modell eine sehr gute Abdeckung der verwendeten Bildverarbeitungsmethoden für die Bildanalyse hat. Es wurde auch nachgewiesen, dass Bildverarbeitungsfunktionen effizient implementiert werden können. Anhand von Profilen der Mikroinstruktionen (RISC-Befehle) unterschiedlicher Typen von Bildverarbeitungsfunktionen wurde gezeigt, wie ihre Rechenkomplexität verglichen werden kann. Durch Profile der Makroinstruktionen (komplexe Elementaraufgaben) wurde demonstriert, wie Komplexitätswerte bestimmt werden können, die von der Implementierungsplattform (Hardware oder Software) un-

abhängige Komplexitätsmerkmale repräsentieren.

Neben der allgemeinen Modellbildung ist auch eine Verbesserung am Verfahren zur Umsetzung von Ausbreitungsprozesse durchgeführt worden. Diese Verbesserung hat Einfluss auf die korrespondierende Adressierungsfunktion und somit auch auf alle Bildverarbeitungsfunktionen mit diesem Adressierungstyp. Das verbesserte Ausbreitungsverfahren setzt LIFOs im Vergleich zu den bisher verwendeten FIFOs ein. Dadurch ergeben sich Vorteile für eine Hardware-Implementierung von hierarchischen Ausbreitungsprozessen und für Architekturen mit Fließbandverarbeitung. Die Verwendung von LIFOs hat aber keinen Einfluss auf die Qualität der Bildverarbeitungsfunktionen und des Algorithmus. Für die LIFOs wurde auch ein verbessertes Speicherkonzept entwickelt. Der Vorteil dieses Konzepts, der in vielen Fällen geringere Speicherbedarf, ist anhand eines Profils der Makroinstruktionen belegt worden.

Insgesamt ermöglicht die Verwendung des Modells einen durchgängigen Entwurf des Algorithmus, von der Auswahl des grundlegenden Wirkungsprinzips des Algorithmus, über die Qualitätsoptimierung (z.B. durch Verwendung angepasster Filter), die Optimierung der Rechenleistung, bis hin zur Abschätzung der Implementierbarkeit auf einer Hardware-Plattform. Dadurch lassen sich vom Anfang der Algorithmenentwicklung an Auswirkungen auf die Rechenkomplexität und die Rechenzeit auf einer Hardware-Plattform analysieren.

Abstract

This thesis presents a model for describing, implementing and analyzing image processing functions. The application domain for these image processing functions is image analysis, covering object segmentation and feature extraction. Image analysis is an enabling technology to implement new functionalities for consuming multimedia data, especially video. New functionalities relying on image analysis are interactive video or the automatic extraction of the semantics of visual data.

To provide the required technology, both the design of algorithms and their implementation in products are still to be done. To provide an efficient development, the following prerequisites should be fulfilled. Specifying the algorithm and its image processing functions must be as simple and comprehensive as possible, thus enabling the exchange of knowledge inside and between researcher groups. It is desirable to re-use existing software modules for implementing the functional model of the algorithm. This will speed up the verification process of the algorithm's functionality. Finally, it should be possible to estimate the processing complexity already during the software simulations of the algorithm. Thus, it is possible to determine whether the algorithm can be implemented with a reasonable amount of hardware resources.

With respect to software libraries, the requirements for an efficient implementation of the simulation software are met. In contrast to this, there are neither suitable tools available to do a simple and non-ambiguous specification nor to estimate the complexity of an algorithm.

Due to these facts, this thesis proposes a model covering image processing functions. The model is dedicated to functions which have 2-dimensional arrays (images) as input and output data . The model does a break-up into the addressing and the processing part of the image processing function. The addressing part complies with one of the three addressing types: inter addressing (which have two pixels from two images as input), intra addressing (which have one pixel and its neighbors as input), and recursive addressing (to implement expansion processes). The addressing part is characteristic of the whole application domain of image analysis (or image processing).

The processing part defines the specific properties of the image processing function. Therefore, this part is characteristic of the individual application. The model specifies the interface of the processing functions. The functionality of the processing function can be selected arbitrarily, and thus is not included in the model.

However, it is also possible to use data that is indexed based on the pixel values, provided that the image structure enforces the scheduling of the processed data. The model is not meant for functions which are not organized as images. A second constraint is that each pixel is processed only once. Exceptions are possible for expansion processes. However, violating this rule means that the mapping of the function to the model might fail.

The break-up into the addressing and processing part allows implementing the addressing functions centrally in a library. This leads essentially to the consequence that the usage of the model covers both aspects of the development process efficiently: the algorithm design (with the focus on a quality optimization) and the algorithm implementation (stressing the complexity optimization). First, the model allows for the algorithm to be described and specified unambiguously. This is the foundation for communication and benefiting from synergies. Furthermore, the centrally implemented addressing functions are designed for re-use in image processing functions. In fact, this library provides a platform for the rapid software implementation of algorithms into functional models, because the addressing functions contribute the major part to the complexity of an image processing function. Using a conceptual architecture model for the generic components, it is possible to identify complexity features of the algorithm and to measure them already during the functional verification of the algorithm.

The implementation of the example algorithm that combines a significant set of methods for object segmentation, proved that the model has a very high coverage of image-processing functions used for image analysis. It has been shown that new image-processing functions can be designed and implemented very efficiently. Profiles on microinstruction level[3] for the different image processing function types (addressing schemes) have been analyzed to show how to compare the computational complexity of function of these types. Macroinstruction[4] profiles have been used to show how to measure the computational complexity features that are independent from the implementation platform (e.g., hardware or software).

Besides providing a generalized model for the different classes of image processing functions, this thesis also proposes optimizations for the expansion processes, which are typical of image analysis. These optimizations improve the corresponding addressing function, and thus all functions using this addressing type. The optimized scheme utilizes LIFOs instead of FIFOs. The benefits of this new scheme are with respect to a hardware implementation of hierarchical expansion processes, and for architectures using a pipelined

[3]RISC instructions

[4]complex elementary tasks

processing. The usage of LIFOs has no effect on the quality of the image processing functions and the algorithm. Furthermore, a new memory concept improving the implementation of the LIFOs was presented. Thus, the required amount of memory used in the LIFOs is much lower in most cases. The increased memory efficiency has been proved using a macroinstruction profile.

Using this image processing model, algorithms can be developed consistently. This covers the selection of the principle of the algorithm, the algorithm design, the quality optimization, the optimization with respect to computational complexity, and the estimation as to whether the algorithm can be implemented on hardware architecture that is compliant to the proposed hardware concept. It is possible to estimate from the beginning of the development the effects of changes in the algorithm on the computational complexity and the execution time on hardware architecture.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Künftige Anwendungen im Bereich der digitalen Bildsignalverarbeitung werden mehr Funktionalität anbieten, als das einfache Abspielen von Videos. Eine Möglichkeit ist die objektbasierte Videokodierung, bei der die Information, welche Bildpunkte zu einem Objekt gehören, erzeugt wird und bis zum Abspielen des Videos erhalten bleibt. Durch diese Objektmaske kann mit den Videoobjekten während des Abspielens interagiert und auf Zusatzinformationen zum Objekt zugegriffen werden.

Ein wesentliches Forschungsgebiet ist dabei die Erzeugung der Objektmasken. Dieser Prozess wird Objektsegmentierung genannt. Die Objektsegmentierung analysiert dabei unterschiedliche Merkmale des Bildinhaltes, was mit Bildanalyse bezeichnet wird. Bildanalyse wird auch zur Merkmalextraktion eingesetzt. Dabei wird die Bild- oder Merkmalanalyse nicht eingesetzt, um Objektmasken, sondern um Deskriptoren zu erzeugen. In beiden Fällen wird die Bildinformation in einer "quantisierten" Form dargestellt, die abhängig von den Objekten im Bild ist. Bildanalyse ist somit die Technik für die Anwendungsfälle Objektsegmentierung und Merkmalextraktion.

In vielen Fällen werden Bildanalysealgorithmen von unterschiedlichen Institutionen entwickelt, die dann später miteinander verglichen werden. Dies wird z.B. in wissenschaftlichen Projektgruppen oder bei der Entwicklung von Standards gemacht. Als Konsequenz des Vergleichs wird dann ein Algorithmus ausgewählt, der in die Referenzplattform oder den Standard übernommen wird. Als Basis für die Bewertung wird auf der einen Seite die Qualität des Algorithmus beurteilt. Auf der anderen Seite ist es auch wünschenswert, die dafür benötigte Rechenleistung zu bewerten, besonders wenn die Algorithmen vergleichbar gute Ergebnisse bzgl. der Qualität liefern. Zur Beurtei-

lung der Qualität werden oft die Randbedingungen für die Messung genau definiert. Elemente, die dabei definiert werden, sind der Testdatensatz und die Sollergebnisse für diesen Datensatz. Ein anderer Aspekt ist, dass die Ergebnisse idealer Weise mit unabhängigen Implementierungen reproduzierbar sein sollten. Insbesondere werden bei der Standardisierung Verifikationen der Ergebnisse von unterschiedlichen Institutionen gefordert. Leider ist eine Spezifikation der Technik, die genau genug ist, um reproduzierbare Ergebnisse mit unabhängigen Implementierungen zu erreichen, in der Regel nicht vorhanden. Dies liegt daran, dass solche Spezifikationen keine eindeutige und klare Form haben oder aber unvollständig sind.

Noch schlimmer sieht es bei der Ermittlung der Rechenkomplexitäten aus. Hier hängt die Rechenzeit von den Fähigkeiten des Entwicklers der Software-Implementierung, dem Grad der Optimierung und vom Algorithmus ab. Idealer Weise sollte aber nur der Algorithmus selbst Einfluss auf die Komplexitätsmessung haben.

Die Beurteilung von Qualität und Komplexität wird auch benötigt, wenn ein Algorithmus nach seinem Entwurf in optimierter Form implementiert werden soll. Dabei verschlechtert sich oft die Qualität des Algorithmus, wenn zugunsten der Rechenleistung Vereinfachungen durchgeführt werden müssen. Umgedreht führt eine Qualitätsoptimierung in der Regel zu erhöhten Anforderungen an die Rechenleistung. Um hier die geeignete Balance zwischen Komplexität und Qualität einstellen zu können, muss man beides messen können. Außerdem muss man in der Lage sein, schnell unterschiedliche Vereinfachungen im Algorithmus zu simulieren. So kann man den Preis ermitteln, der auf Seiten der Qualität bezahlt werden muss, wenn man auf eine bestimmte Art und Weise die benötigte Rechenleistung reduziert. Um diese Auswirkung der Vereinfachungen schnell simulieren zu können, benötigt man eine einfache und effiziente Art der Implementierung.

Ein typischer Entwicklungsablauf ist in Abbildung 1.1 dargestellt. Dieser Entwicklungablauf beginnt bei der Spezifikation, die sich an Anforderungen von Anwendungen orientiert. Danach wird ein Algorithmus entwickelt. Hierbei wird auf Wissen, das aus der Literatur oder dem vorhandenen Know-How kommt, aufgebaut. Im nächsten Schritt wird der Algorithmus optimiert. Auf der einen Seite kann dies in Richtung zur Verbesserung der Qualität gemacht werden. Dies geschieht durch Simulation und Beobachtung, wo das Modell, das die Realität in den Algorithmus überführt, verfeinert werden muss. Nicht selten erlebt man dabei, dass die vorhandenen Bildverarbeitungsfunktionen an die Verfeinerungen angepasst werden müssen. Der andere Zweig optimiert mit dem Ziel der Reduktion der Rechenleistung. Hierbei wird in der Regel das Modell, das die Realität auf den Algorithmus abbildet, vereinfacht. Dies stellt einen Widerspruch zur Optimierung der Qualität dar, da dort das

Modell häufig verfeinert wird. Auf der anderen Seite gibt es auch Teile der Optimierungsstrategien, die nicht gegeneinander arbeiten. So hat die Entwicklung von angepassten Filtern, oder die Parallelisierung nicht unbedingt einen negativen Einfluss auf ein anderes Optimierungsziel. Schließlich wird nach allen Optimierungen der Algorithmus auf einer Plattform von vielen möglichen implementiert. Dabei bestimmt die Zielplattform die Randbedingungen, in welche Richtung stärker optimiert werden muss oder kann.

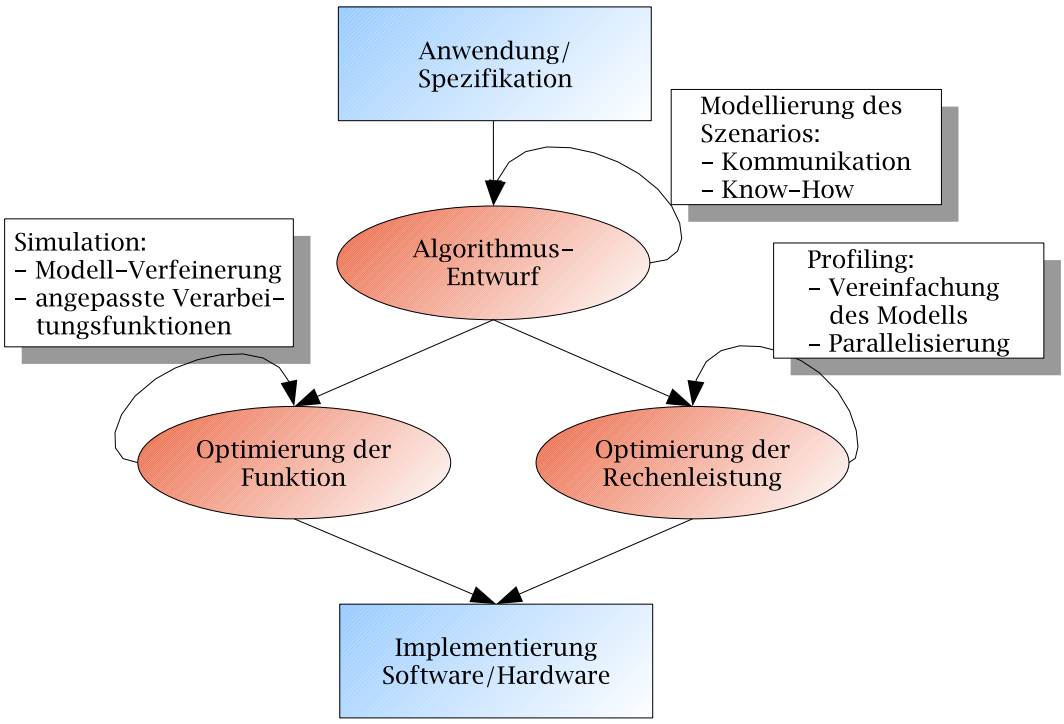


Abbildung 1.1: Arbeitsablauf bei der Algorithmenentwicklung und -implementierung

Ein Hauptproblem bei dem dargestellten Entwurfsablauf ist, dass beide Optimierungen mehr oder weniger getrennt voneinander ablaufen. Dies liegt nicht nur an den unterschiedlichen Interessen- und Erfahrungsschwerpunkten der Entwickler. Häufig benötigt man auch andere Simulationsmodelle, um bestimmte Eigenschaften des Algorithmus zu beobachten. Für die Qualitätsbeurteilung sind C−Modelle sehr gut geeignet. Mit diesen Modellen ist es aber nicht möglich, unmittelbar Architekturaspekte, wie Parallelisierungen, zu berücksichtigen. Dafür werden dann häufig die Architekturen mit einer Hardware-Beschreibungssprache modelliert.

1.2 Ziel der Arbeit

Aus den eben beschriebenen Anwendungsfällen lassen sich die allgemeinen Zielsetzungen dieser Arbeit ableiten. Als zentrales Element dieser Arbeit soll ein Modell entwickelt werden, mit dem man Bildverarbeitungsfunktionen exakt und einfach spezifizieren kann. Dieses Modell sollte dabei auf Bildverarbeitungsfunktionen, die in der Bildanalyse eingesetzt werden, allgemein anwendbar sein. Die Abbildung der Bildverarbeitungsfunktionen auf das Modell stellt dann eine Methode dar, um diese Funktionen effizient zu beschreiben und zu spezifizieren. Aufbauend auf diesem abstrakten Modell wird ein Software-Modell entwickelt. Die Entwicklung von Bildverarbeitungsfunktionen mit dem Software-Modell stellt dann eine Methode dar, um Bildanalysealgorithmen effizient zu implementieren. Effizient bedeutet dabei, dass die Zeit zur Implementierung möglichst kurz sein sollte. Daneben bedeutet hier effizient auch, dass der implementierte Code unter Berücksichtigung des generischen Modellcharakters weitgehend optimiert ist. Außerdem soll eine Methode entwickelt werden, mit der die Rechenkomplexität weitgehend unabhängig von einer speziellen Plattform ermittelt werden kann. Dafür soll ein abstraktes Hardware-Modell entwickelt werden, an dem dann allgemeine Komplexitätsmaße definiert werden können. Somit stellt diese Arbeit ein Bildglied zwischen dem Algorithmendesign und der Algorithmenimplementierung dar.

Anhand des Modells sollen auch die Verarbeitungsschritte der Bildverarbeitungsfunktionen betrachtet werden. Hierbei soll besonders auf Ausbreitungsprozesse eingegangen werden. Durch eine Diskussion auf dieser Ebene sollen Möglichkeiten zur Verbesserungen der Abläufe identifiziert werden.

Die Beurteilung der funktionalen Qualität von Bildverarbeitungsalgorithmen wird in dieser Arbeit nicht betrachtet. Dies ist sinnvoll, da hierfür neben dem Algorithmus auch seine Anwendung berücksichtigt werden muss. In dieser Arbeit steht aber das Anwendungsgebiet Bildanalyse im Zentrum und nicht die Betrachtung einer Anwendung. Dabei wird Bildanalyse in dieser Arbeit in Richtung Objektsegmentierung betrachtet.

1.3 Aufbau der Arbeit

In den folgenden Kapiteln dieser Arbeit werden zuerst Beispiele für Bildverarbeitungsanwendungen gezeigt, die auf Objektsegmentierung bzw. Bildanalyse basieren. Da sich diese Arbeit an einem Beispielalgorithmus orientiert, wird im Folgenden nur noch ein Anwendungsgebiet der Bildanalyse, die Objektsegmentierung, betrachtet. Im Anschluss an die Übersicht der Bildverar-

beitungsanwendungen wird der Stand der Technik beschrieben. Hier wird zuerst ein Überblick über Algorithmen gegeben, um die Randbedingungen für das abstrakte Modell für Bildverarbeitungsfunktionen festzulegen. Um das Verständnis über die Funktionsweise der einzelnen Bildverarbeitungsfunktionen zu verbessern, ist auch die Betrachtung von speziellen Architekturen zur Umsetzung der Algorithmen notwendig. Insbesondere werden hier Architekturen für Ausbreitungsprozesse diskutiert, da diese für Bildanalysealgorithmen charakteristisch sind. Dann wird zur Einordnung des Software-Modells und der damit verbundenen Methodik zum Implementieren von Bildverarbeitungsfunktionen ein Überblick über Bildverarbeitungsbibliotheken gegeben. Außerdem werden in diesem Kapitel noch Entwicklungsumgebungen zur Erstellung von Bildverarbeitungsalgorithmen betrachtet. Für die Diskussion der Methodik zur Komplexitätsanalyse wird schließlich noch ein Überblick über Prozessorarchitekturen gegeben. Hierbei ist ein wesentlicher Aspekt eine Betrachtung von Parallelisierungskonzepten, die mit unterschiedlichsten Architekturen realisiert werden.

Am Ende des Kapitels "Stand der Technik" werden Lücken bzgl. der einzelnen Arbeitsschwerpunkte aufgezeigt. Diese Lücken sind im Wesentlichen:

- fehlende Spezifizierbarkeit
- ungeeignete Systematisierung von Bildverarbeitungsfunktionen
- Unvollständigkeit von Entwicklungsumgebungen für Algorithmen
- fehlende Flexibilität der Entwicklungsumgebungen zum Ausbalancieren zwischen Qualität und Rechenleistung
- fehlende Möglichkeiten zur objektiven und plattformunabhängigen Messung der Rechenkomplexität
- fehlendes abstraktes Hardware- und Komplexitätsmodell

Daraus wird die konkrete Aufgabenstellung abgeleitet. Hier spielt die Systematisierung in Form einer Modellbildung eine zentrale Rolle. Die Modellbildung wird dabei auf unterschiedlichen Ebenen durchgeführt, um die einzelnen Lücken im Stand der Technik zu schließen.

Zur vollständigen Festlegung des Startpunktes der Arbeit wird das Einleitungskapitel mit der Beschreibung eines konkreten Algorithmus zur Objektsegmentierung abgeschlossen. Dieser Algorithmus stellt dabei eine sinnvolle Vereinigung der vorher vorgestellten Algorithmen zur Objektsegmentierung dar. Er ist somit repräsentativ im Sinne der verwendeten Bildverarbeitungsfunktionen. Obwohl dieser Algorithmus sehr gute Segmentierungsergebnisse

erzielt, dient er hier nur als repräsentatives Beispiel, an dem sich die weitere Arbeit orientiert. Der Algorithmus ist nicht als universell bzgl. der Anwendbarkeit zu betrachten.

In Kapitel 2 werden dann die eigenen Arbeiten beschrieben. Hier wird zuerst das abstrakte Modell für Bildverarbeitungsfunktionen erläutert. Dabei wird zunächst das Modell basierend auf dem Stand der Technik hergeleitet. Besonderer Schwerpunkt wird hierbei wieder auf Ausbreitungsprozesse gelegt, die im Rahmen dieser Arbeit erstmalig umfassend diskutiert werden. Außerdem wird an dieser Stelle ein neues Prinzip für die Darstellung von Ausbreitungsprozessen vorgestellt, das dann in das Modell mit einfließt. Aufbauend auf diesem Modell wird dann die Methode beschrieben, um die Bildverarbeitungsfunktionen auf das Modell abzubilden.

Danach wird das Software-Modell für das abstrakte Modell beschrieben. Die Anwendung dieses Modells ist dann eine Methode zur effizienten Implementierung von Bildverarbeitungsfunktionen. Diese ist sehr eng mit der Methode zur Abbildung auf das abstrakte Modell verbunden.

Schließlich wird noch das Hardware-Modell vorgestellt, das einem Konzept für die Implementierung von konkreten Architekturen entspricht. Es werden in diesem Zusammenhang allgemein sinnvolle Parallelisierungen beschrieben, die in diesem Konzept umgesetzt werden. Anschließend wird die Struktur von Hardware- und Software-Modell noch verglichen. Aufbauend auf der fast identischen Struktur werden dann einfache Komplexitätsmaße definiert. Daraus lässt sich dann die Methode ableiten, mit der die Komplexität des Algorithmus bestimmt werden kann. Diese Komplexität repräsentiert auf der einen Seite die Laufzeit des Algorithmus sowie auf der anderen Seite den Bedarf an Ressourcen, die der Algorithmus benötigt. Im konkreten sind diese Ressourcen der benötigte LIFO Speicher oder der Speicher für indizierte Daten, die während der Verarbeitung verwendet werden.

In Kapitel 3 werden dann die Ergebnisse der Arbeit dargestellt. Es wird gezeigt, dass sich die Funktionen des Beispielalgorithmus auf das Modell abbilden lassen. Daneben werden auch die Grenzen des Modells beschrieben. Diese Grenzen bestehen darin, dass jeder Bildpunkt nur einmal verarbeitet wird, bzw. dass die Bildinformationen in einer Bildstruktur organisiert sind. Die Methode zum Abbilden von Bildverarbeitungsfunktionen auf das abstrakte Modell sowie auf das Software-Modell wird dann anhand eines Beispiels beschrieben, das gleichzeitig klar die Effizienz beim Umsetzen der Funktionalität auf Softwarefunktionen belegt. Darauf folgend wird anhand von Profilen nachgewiesen, dass die Modelle für Komplexitätsmessungen verwendbar sind. Dabei wird mit einem Instruktionsprofil des Software-Modells demonstriert, wie die Komplexitäten von unterschiedlichen Bildverarbeitungsfunktionen miteinander verglichen werden können. Daneben wird

mit einem Profiling der Makroinstruktionen eine Messung der verwendeten Speicherbandbreite zum Bildspeicher durchgeführt. Anhand eines weiteren Profils der Makroinstruktionen wird außerdem der maximale Füllstand des LIFO-Speichers ermittelt. Komplexere Makroinstruktionsprofile werden hier nicht gezeigt, da diese bereits in einer weiterführenden Arbeit zum Erstellen einer optimierten Implementierung des Hardware-Modells durchgeführt wurden [1].

Abschließend werden weitere Beispiele genannt, bei denen Algorithmen mit dem Software-Modell implementiert wurden. Dadurch wird zusätzlich belegt, dass das Modell eine sehr große Abdeckung an Bildverarbeitungsfunktionen hat.

1.4 Stand der Technik

1.4.1 Grundlagen und Anwendungen der Objektsegmentierung

Zuerst sollen in diesem Kapitel Anwendungen gezeigt werden, die eine Bildanalyse voraussetzen. Wie schon in der Motivation erwähnt wurde, wird in dieser Arbeit die Bildanalyse für die Unterteilung von Bildern in einzelne Objekte, die so genannte Objektsegmentierung, betrachtet. Dabei stellten Anwendungen mit einer MPEG−4 Kodierung die Domäne der Zielanwendungen dar. Bild 1.2 zeigt ein System eines MPEG−4 Codecs. Die Bilddaten, die in diesem Bild mit einer Kamera aufgenommen werden, werden zuerst im Segmentierer in einzelne Objekte zerlegt. Mögliche Algorithmen für diese Zerlegung sind in Kapitel 1.4.3 dargestellt. Dann werden die Objekte einzeln kodiert, so dass jeweils ein Video-Elementar-Bitstrom die Daten eines Videoobjekts enthält. Im Folgenden werden die Elementar-Bitströme dann zu einem System-Bitstrom gemultiplext. Dieser Bitstrom wird dann zum Decoder übertragen. Im Demultiplexer des Decoders werden aus dem System-Bitstrom wieder die Elementar-Bitströme gewonnen und einzeln decodiert. Die decodierten Videoobjekte werden mit dem so genannten *Compositor*, dem inversen Segmentierer, wieder zu einer Szene zusammengestellt und auf dem *Display* dargestellt.

Auf den ersten Blick sieht das in Abbildung 1.2 angezeigte Videobild genauso aus, wie bei einer herkömmlichen bildbasierten Kodierung. Allerdings hat hier der Decoder noch zusätzlich die Objektinformationen verfügbar, so dass der Benutzer z.B. durch klicken auf Objekte mit diesen interagieren kann. So kann er z.B. in einem multimedialen Warenkatalog durch Anklicken von Objekten im Video Bestellinformationen zu den geklickten Waren er-

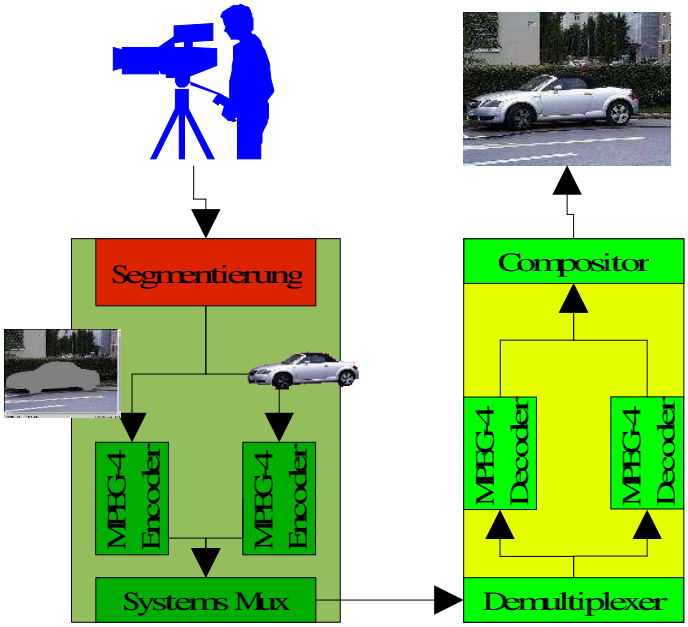


Abbildung 1.2: MPEG−4-Video-Codec bestehend aus Encoder-System (Kamera, Segmentierer, Encoder und Multiplexer) und Decoder-System (Demultiplexer, Decoder, Compositor und Display).

halten. In Schulungsvideos werden dann nicht Bestellinformationen *verlinkt* sein, sondern weitere Videos, in die bei Bedarf (per Klick) verzweigt wird. Diese Funktionalität wird bereits auf Web-Seiten auf Standbildern angewendet. Hier wird diese Funktion durch die Verwendung von Tranzparenzinformation, die von einigen Bild-Dateiformaten unterstützt wird, ermöglicht. MPEG−4 erweitert diese Interaktionsmöglichkeiten auf Videos.

Bei der virtuellen Videokonferenz werden nur noch die Elementar-Bitströme, die die Konferenzteilnehmer darstellen, kodiert und übertragen, um Bitrate und Rechenleistung zum Dekodieren zu sparen. Auf den Terminals können die Teilnehmer der virtuellen Videokonferenz, an einen virtuellen Konferenztisch gesetzt, dargestellt werden. Dadurch wird der nicht übertragene Hintergrund mit einem anwendungsnahen, vordefinierten Hintergrund ersetzt.

Die Objektsegmentierung, die in diesen Beispielen benötigt wird, setzt eine komplexe Bildanalyse voraus. Analysiert wird dabei die Homogenität der Bilddaten hinsichtlich von Merkmalen im Bild- oder Videoinhalt, wie Farbe, Bewegung oder Tiefe. Eine Bildanalyse, für die das Bildverarbeitungsmodell eingesetzt werden kann, wird auch bei der Merkmalextraktion durchgeführt.

Dabei werden mit der Bildanalyse Werte für festgelegte Merkmale im Bild berechnet und in Deskriptoren abgespeichert. Anwendungen, bei denen eine Merkmalextraktion durchgeführt wird, sind die inhaltsbasierte Suche (z.B. Suche nach Bildern mit ähnlicher Farbverteilung basierend auf MPEG−7 Deskriptoren) und die Klassifikation von Bildern. Beispiele für einfache Klassifikationen sind die Bestimmung von *Indoor* und *Outdoor* Bildern, die automatische Erkennung von Torszenen in Sportberichten oder die Detektion von Ereignissen in Überwachungssystemen. Neben dieser einfachen Klassifikation kann die Merkmalextraktion auch auf Objekte angewendet werden, um die Semantik der Objekte automatisiert zu bestimmen. Im umgekehrten Fall kann die Objektsegmentierung auch durch die Merkmalextraktion unterstützt werden, indem die Segmente einer übersegmentierten Maske (generiert z.B. durch eine Farbanalyse) zu Objekten zusammengeführt werden.

1.4.2 Prinzip der Objektsegmentierung

Nachdem nun Beispiele für die Anwendung von Objektsegmentierung gegeben wurden, sollen nun die grundlegenden Prinzipien der Objektsegmentierung erläutert werden. Zuerst lassen sich Objekte wie folgt definieren [2]:

Definition 1.4.1 *Ein Objekt ist eine örtlich zusammenhängende Menge von Bildpunkten, die ein Homogenitätskriterium erfüllen.*

Prinzipiell kann die Objektdefinition in zwei Richtungen interpretiert werden. Zum einen kann auf Homogenität innerhalb des Objektes hin analysiert werden und zum anderen auf die Unstetigkeiten, die zwischen den Objekten vorhanden sind. Bei der Analyse von Unstetigkeiten entstehen in der Regel keine geschlossenen Konturen. Hier müssen die Kanten so weitergeführt werden, dass sie das Objekt vollständig umschließen. Im Fall, dass die Homogenität analysiert wird, entstehen immer Flächen und somit auch immer eine geschlossene Kontur. Hier werden aber die Unstetigkeiten beim Ausbreiten im homogenen Bereich umflossen, was zu inkorrekten Ergebnissen führt. Der Unterschied zwischen Unstetigkeiten und Homogenitäten liegt im Wesentlichen nur in der Lage des Schwellwertes, bis zu welchem Elemente als homogen beurteilt werden oder aber ab welchem eine Unstetigkeit erkannt wird.

In der realen dreidimensionalen Welt ist die Homogenität der örtlichen Lage (Zusammenhang) bestimmend für Objekte. Das heißt, dass alle Elemente eines Objektes zusammenhängend sind und sich auch zusammenhängend bewegen. Zwischen Objekten können sich Freiräume befinden, die auch für begrenzte Zeiträume verschwinden können, wenn sich die Objekte berühren.

Eine zielstrebige Umsetzung dieses Homogenitätskriteriums wäre also die Erzeugung einer Tiefenkarte, bei der für jeden Bildpunkt die Tiefe im Bild abgelegt ist. Anschließend kann die Tiefenkarte analysiert und Regionen mit homogener Tiefe gefunden werden. Algorithmen, die dies tun, sind in der Regel sehr leistungsfähig. Allerdings kommen diese Algorithmen nicht mit normalem Videomaterial aus, da leider die Unstetigkeiten der Tiefeninformation an Objektgrenzen verloren gehen, wenn die dreidimensionale Realität in ein zweidimensionales Bild projiziert wird. Nun können Bildpunkte direkt benachbart sein, selbst wenn sie in der 3D Welt durch unterschiedliche Tiefe weit auseinander lagen. Dies ist ein dramatischer Informationsverlust, da man nun nach anderen Homogenitätskriterien suchen muss. Neben den neuen Homogenitätskriterien muss aber das Kriterium des örtlichen Zusammenhangs weiter gelten, da Elemente, die in der 3D Welt benachbart sind auch in der 2D Welt benachbart bleiben, solange das Objekt bei der Projektion nicht verdeckt wird.

In der 2D Welt der Bilder werden häufig die Bildpunktwerte benutzt, um nach homogenen Regionen zu suchen. Die Bildpunktwerte sind dabei die von der Kamera aufgenommen Helligkeits- und Farbwerte. Da ein Objekt mit einer semantischen Bedeutung aber aus unterschiedlichen Regionen mit unterschiedlichen Farben bestehen kann, hat die aus einer Farbanalyse resultierenden Segmentmaske in der Regel viel zu viele Regionen. Sie ist also stark übersegmentiert. Zusätzlich zu den Regionen mit unterschiedlichen Farben kommen noch unterschiedliche Ausleuchtungen, wie Schatten und Reflexionen, hinzu. Der große Vorteil der Farbanalyse besteht aber darin, dass die Bilddaten unbearbeitet und ohne Degradierung verwendet werden, was zu einer hohen Genauigkeit der Objektgrenzen führt.

Neben der Analyse der Farbwerte kann auch nach homogenen Mustern (Textur) im Bild gesucht werden. Dabei kann nach konstanten Ortsfrequenzen, die das Muster ausmachen, gesucht werden. Durch die Frequenzfilterung entsteht aber eine Unschärfe der Ortsauflösung, mit der bestimmte Frequenzen auftreten. Durch diese Unschärfe von abgeleiteten Merkmalen hat die resultierende Segmentmaske auch eine höhere Unschärfe, was einer niedrigeren Ortsauflösung entspricht.

Ein wichtiges und häufig verwendetes Merkmal ist die Bewegung. Wie bei der Textur liegt hier häufig eine Unschärfe des Segmentierungsergebnisses vor, da für die Berechnung der Bewegungsvektoren nicht nur der Bildpunkt, sondern auch seine Umgebung benutzt wird. Außerdem kann bei sehr vielen Bildpunkten nicht die tatsächliche Bewegung ermitteln werden. So lassen sich nur bei Ecken im Bildinhalt vertrauenswürdige Bewegungsvektoren ermitteln. Hingegen ist es an Kanten nur möglich die Bewegung in Richtung der Normalen zur Kante zu ermitteln und in Bereichen ohne Textur

in keine Richtung. Auf der anderen Seite stellt die Bewegung eine wichtige Information dar, da Objekte, die örtlich zusammenhängend sind, sich auch zusammenhängend bewegen. Diese Information hat also einen direkten Bezug zur Objektdefinition und lässt sich unter der Voraussetzung, dass sich das Objekt unterscheidbar von anderen Objekten bewegt, auch als vollwertiger Ersatz zur Objektdefinition betrachten.

1.4.3 Algorithmenübersicht

Nachdem nun die grundlegenden Prinzipien der Objektsegmentierung erläutert wurden, werden in diesem Kapitel einige Segmentierungsalgorithmen vorgestellt. Einige der Algorithmen setzen bestimmte Randbedingungen voraus, die zu einer Vereinfachung des Segmentierungsproblems führen. In diesen Fällen können Objektmasken mit hoher Qualität bei niedriger Rechenleistung berechnet werden.

Das gängigste Verfahren, um Objektmasken zu generieren, ist die *Blue-Screen*-Technik. Hier wird der Vordergrund vor einem einfarbigen Hintergrund aufgenommen. Der Hintergrund ist dabei meist blau, woher dieses Verfahren auch den Namen hat. Der Hintergrund kann aber auch andere Farben haben. So wird z.B. auch ein grüner Hintergrund eingesetzt. Später, nach der Aufnahme, wird dann das Vordergrundobjekt mit einem *Compositor* (siehe Abbildung 1.2) über ein Hintergrundbild geblendet. Die klassischen Anwendungen hierfür sind der Wetterbericht im Fernsehen, Musikvideos, bei denen der Künstler vor einem synthetischen Hintergrund zu sehen ist oder Tricks bei der Produktion von Hollywood-Filmen. Auch ein Teil der Testsequenzen, die bei den *Core*-Experimenten zur Entwicklung des MPEG−4 Standards benutzt wurden, wurde mit der Blue-Screen-Technik aufgenommen. Eine wesentliche Einschränkung der Blue-Screen-Technik ist, dass die Farbe des Hintergrunds nicht im Vordergrundobjekt vorkommen darf, da diese Teile dann später nicht vom Hintergrund unterschieden werden können und bei der Überblendung transparent, also unsichtbar sind. Ein zweiter, gravierender Nachteil dieser Technik ist, dass sie praktisch nur im Studio eingesetzt werden kann, so dass die Aufnahme bereits mit hohen Kosten verbunden ist. Auf bestehendes Videomaterial, dass konventionell aufgenommen wurde, kann diese Technik nicht angewendet werden. Dafür liefert diese Technik aber sehr hochwertige Ergebnisse bei sehr niedriger Rechenkomplexität. Analog zur Blue-Screen-Technik können beim *Chroma-Keying* auch zur Speicherung der Objektmaske alle Bildpunkte, die zum Hintergrund gehören, mit einer vorgegebenen Farbe markiert werden.

In [3] wurde ein Verfahren angewendet, das keinen einfarbigen Hintergrund benötigt. Wie zuvor muss aber der Hintergrund bekannt sein. Dafür

wird ein Bild des Hintergrunds ohne Vordergrundobjekt aufgenommen und abgespeichert. Ist später das Vordergrundobjekt im Bild, lässt es sich leicht im Differenzbild zum bekannten Hintergrund erkennen. Wie zuvor bei der Blue-Screen-Technik hat auch dieses Verfahren eine sehr niedrige Rechenkomplexität und liefert auch gute Ergebnisse. Fehler in der Segmentmaske können hier entstehen, wenn es lokal im Bild zu geringen Unterschieden zwischen dem verdeckten Hintergrund und dem Vordergrundobjekt kommt. Im Gegensatz zur Blue-Screen-Technik lässt sich diese Situation schwerer vermeiden, da der Hintergrund eine deutlich höhere Komplexität (im Sinne der Struktur und Farbverteilung) haben kann. Eine Einschränkung, die sich durch dieses Verfahren ergibt, ist, dass die Kamera statisch sein muss. Dies gilt nicht nur für die Kamera, sondern auch für die Beleuchtungsverhältnisse. Diese Technik wird bei virtuellen Telekonferenzsystemen oder auch in [4] angewendet. Hier werden Passanten, die sich in einer bekannten Umgebung, einer zuvor aufgenommenen Einkaufsstraße, befinden, in eine synthetische Umgebung eingeblendet. Bild 1.3 zeigt ein Beispielbild dieser Anwendung. Da sich die Kamera außen befindet, ist der Hintergrund wechselnden Lichtverhältnissen unterworfen, so dass der Hintergrund permanent neu kalibriert werden muss.

In [5] wird der *Change-Detection-Mask*-Algorithmus (CDM) vorgestellt. Im Gegensatz zu den vorher vorgestellten Algorithmen wird kein bekannter Hintergrund vorausgesetzt. Der CDM-Algorithmus basiert auf der Erkennung von Änderungen im Bildinhalt, die durch die Bewegung des Vordergrundobjektes entstehen. Wie beim Einsatz des Referenzbildes muss auch hier die Kamera und die Beleuchtung statisch sein. So kann zuerst die Differenz zwischen einem Bild und dem Vorhergehenden berechnet werden. In Bereichen, in denen das Vordergrundobjekt strukturiert ist, entsteht dann bei Bewegungen des Objektes ein Unterschied zwischen den Bildern. Um diese Differenz zu erkennen, reicht dabei selbst eine geringe Strukturierung und eine geringe Bewegung aus. Eine weitere Quelle für Unterschiede zwischen den Bildern kann Rauschen sein. Deshalb wird der Schwellwert für eine Änderungserkennung aus der Intensität des Rauschens berechnet. Nach Anwendung des Schwellwertes auf das Differenzbild entsteht eine Maske, die viele Bildpunkte des Objektes aber auch Bildpunkte des Hintergrundes markiert hat. Um Lücken in Vordergrund zu schließen und Inseln im Hintergrund zu löschen, werden morphologische Filter angewendet. Da sich nicht alle Lücken in der Vordergrundmaske schließen lassen, werden die Bildpunkte der Maske noch in einem Speicher für eine bestimmte Anzahl an Bildern gespeichert. Jeder Bildpunkt, der neu oder erneut als Teil des Objektes erkannt wurde, wird in den Speicher eingetragen und verbleibt dort wieder für die angegebene Anzahl an Bildern. Ein weiterer Effekt, der bis dahin unberücksichtigt bleibt,

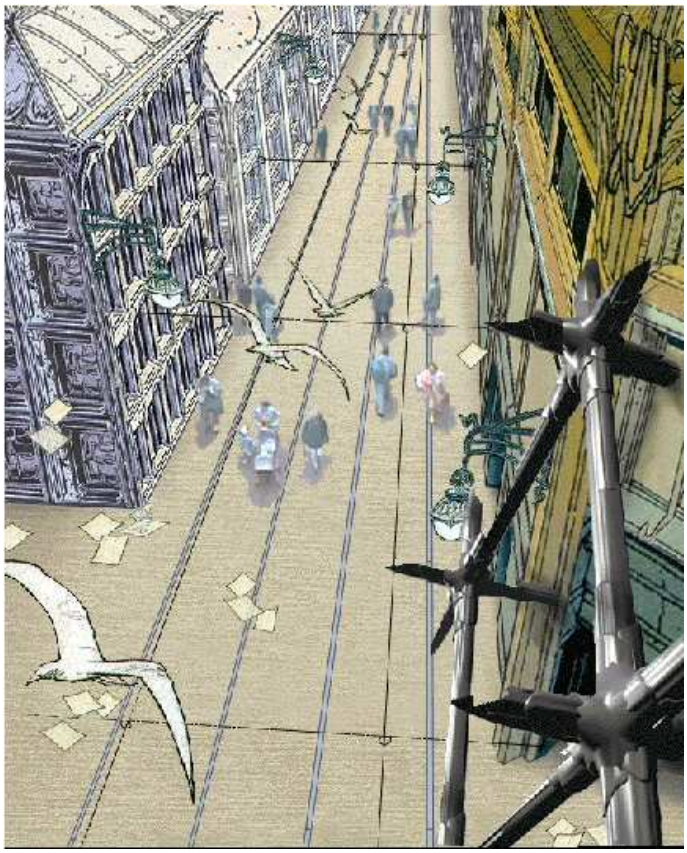


Abbildung 1.3: Urbicande-la-Neuve-Anwendung: Passanten (natürliche Vordergrundobjekte) werden in eine virtuelle Stadt Urbicande-la-Neuve eingeblendet.

ist, dass nicht zwischen dem bewegten Vordergrund und dem aufgedeckten Hintergrund unterschieden werden kann. Deshalb wird nicht nur das zeitliche Verhalten in Form des Differenzbildes analysiert, sondern auch noch die Helligkeitsinformation. Dabei wird die Maske, die sich im Speicher befindet, noch so an das Objekt angepasst, dass die Objektgrenze zum lokalen Maximum des Grauwertgradientens verschoben wird. Die endgültige Objektmaske entspricht dann der Maske im Speicher. Zur Segmentierung des nächsten Bildes wird die Maske im Speicher bewegungskompensiert und neue Bildpunkte aus dem Prozess der Änderungserkennung hinzugefügt.

In [6] wird der CDM-Algorithmus so erweitert, dass er auch mit bewegter Kamera funktioniert. Dafür wird das Vorgängerbild vor der Differenzbildung noch global bewegungskompensiert. Für die globale Bewegungskompensation wird eine Schätzung mit einem 8−Parameter-Modell eingesetzt. Damit las-

sen sich Rotationen, Dehnungen und Zooms inklusive perspektivischer Verzerrungen modellieren.

In [7] wird ein Algorithmus zur modellbasierten Bewegungsschätzung beschrieben. Die globale Bewegungskompensation lässt sich dabei nur auf den Hintergrund anwenden. Bereiche, die zu einem bewegten Vordergrundobjekt gehören, erzeugen dabei einen hohen Schätzfehler. Bereiche, bei denen die globale Bewegungskompensation keinen Schätzfehler hat, werden in ein Hintergrundmosaik eingetragen. So entsteht ein Mosaik, das Lücken an den Stellen hat, an denen sich das Vordergrundobjekt bewegt. Das Mosaik wird dann weiter benutzt, um die globale Bewegung des nächsten Bildes zu schätzen. Aufgedeckter Hintergrund und durch die Kamerabewegung neu hinzugekommenen Bildbereiche ergänzen dabei das Mosaik. Schließlich entsteht ein vollständiges Mosaik. Nun kann wie bei der Segmentierung mit bekanntem Hintergrund das Vordergrundobjekt einfach durch Differenzbildung heraussegmentiert werden. Dies ist dann identisch mit der Beurteilung des Schätzfehlers, als das Vordergrundobjekt nicht in das Hintergrundmosaik eingetragen wurde.

Eine andere Möglichkeit, um die Bewegung von Objekten für die Objektsegmentierung zu benutzen, ist in [8] beschrieben. Hier wird eine Segmentmaske vorausgesetzt, die eine feine Granularität hat, also übersegmentiert ist. Diese Segmentmaske wird mit einem *Recursive-Shortest-Spanning-Tree*-Algorithmus (RSST) [9] unter Benutzung von Farbinformationen berechnet. Dieser Algorithmus ist im Folgenden noch detaillierter beschrieben. Basierend auf der Farbsegmentmaske wird dann durch Zusammenlegung von Farbsegmenten die Objektmaske erzeugt. Als Kriterium für die Zusammenlegung der Segmente wird dabei die Bewegungsinformation verwendet. Dabei müssen die Farbsegmente eines Objektes die Bewegung eines Bewegungsmodells erfüllen. Die Zusammenlegung der Objekte wird wie bei der Farbsegmentierung mit einem RSST-Algorithmus erledigt. Im Vergleich zu den auf Änderung der Bildpunktwerte basierenden Verfahren, wird hier nicht nur in Vordergrund und Hintergrund unterschieden, sondern in mehrere Vordergrundobjekte, die sich unterschiedlich bewegen, und dem Hintergrund. Anders ausgedrückt werden hier nicht nur die Parameter des Bewegungsmodells für den Hintergrund berechnet, sondern auch für alle Vordergrundobjekte. Allerdings muss sich die Bewegung der Vordergrundobjekte von der Bewegung des Hintergrundes deutlich unterscheiden. Die Änderungserkennung hingegen kommt mit minimalen Bewegungen des Vordergrundes aus.

Ein Verfahren, das eine Mischung aus dem eben beschriebenen RSST-Algorithmus, basierend auf der Bewegung, und der Änderungserkennung darstellt, ist in [10] vorgestellt. Hier wird auch von einer Farbsegmentmaske ausgegangen. Allerdings wird nur das Bewegungsmodell des Hintergrundes

berechnet. Alle Farbsegmente, die das Bewegungsmodell des Hintergrundes erfüllen, werden dabei zum Hintergrund. Alle anderen werden als Vordergrund betrachtet.

Bei den beiden eben vorgestellten Algorithmen zur Bewegungssegmentierung, die eine Farbsegmentierung voraussetzen, wird die Farbsegmentmaske bei jedem Bild neu berechnet. Die Farbsegmentmaske ist also nicht kohärent über die Zeit. Zusammenfassungsinformationen können deshalb nicht von einem Bild zum nächsten übertragen werden. In [11] ist ein Algorithmus zur Bewegungssegmentierung beschrieben, der eine zeitlich kohärente Farbsegmentmaske voraussetzt. Der verwendete Algorithmus zur Farbsegmentierung ist in Kapitel 1.6 detailliert beschrieben und ist der Referenzalgorithmus für diese Arbeit. Die Bewegungssegmentierung benutzt einen RSST-Algorithmus zur Zusammenlegung der Farbsegmente. Es entstehen dabei für alle Objekte Bewegungsmodelle. Hier werden Bewegungsmodelle mit 6, 8 oder 12 Parametern berechnet. Zur Berechnung der Bewegungsmodelle werden Bewegungsvektoren eines *Block-Matching*-Algorithmus benutzt, aus denen sich mit Regressionsformeln die Parameter des Bewegungsmodells der einzelnen Segmente berechnen lassen. Zur Beurteilung der Ähnlichkeit der Modelle unterschiedlicher Segmente wird die Genauigkeit des Gesamtmodells beurteilt. In einem Vorverarbeitungsschritt lassen sich Gewichtungsfaktoren für die Bewegungsvektoren ermitteln, so dass Vektoren, die vertrauenswürdig sind, stärkeren Einfluss haben. In folgenden Bildern kann die Zusammenfassungsinformation der vorhergehenden Bilder zur Initialisierung benutzt werden, da ein Farbsegment über die Zeit seine Segmentnummer behält.

Eine Bewegungsschätzung kann auch eingesetzt werden, um die Tiefeninformation der aufgenommenen Szene zu rekonstruieren [12]. Hierbei wird nicht die Bewegung zwischen zwei aufeinanderfolgenden Bildern analysiert, sondern zwischen zwei Bildern die gleichzeitig, aber von unterschiedlichen Positionen aufgenommen wurden. Es wird also zur Aufnahme eine Stereokamera benötigt. Bei Videoaufnahmen ist dabei die Synchronität der Kameras von großer Bedeutung, so dass Unterschiede zwischen den Bildern (Disparität) ausschließlich durch die unterschiedliche Tiefen und der daraus resultierenden Parallaxe entstehen. Im Gegensatz zur normalen Bewegungsschätzung lassen sich hier durch die bekannte Verschiebung der Kameras nicht nur an Ecken vertrauenswürdige Bewegungsvektoren schätzen, sondern auch an Kanten, die senkrecht zur Basis der Kamera liegen. Für alle anderen Bildpunkte müssen die Tiefenwerte geschätzt werden. Das wesentliche Problem hierbei ist die Tatsache, dass der Tiefenwert an einer Kante nur in eine Richtung, nämlich zum Vordergrundobjekt hin, Bedeutung hat [13]. Dadurch ergeben sich viele Bildbereiche, für die keine zuverlässige Tiefeninformation berechnet werden kann. Eine Lösung wäre die Bewegungsschätzung basierend auf gesamten

Segmenten. Hierbei besteht eine Wechselwirkung zwischen der Segmentmaske und der Tiefenberechnung. Zum einen wird für die Berechnung der Tiefenkarte die Segmentmaske benötigt, wobei zum anderen die Segmentmaske wiederum von der Tiefenkarte abhängt. Als Konsequenz muss eine Eingangssegmentmaske verwendet werden, die z.B. auf Basis einer Farbanalyse erzeugt wurde.

Eine zweite Möglichkeit zur Erstellung einer Tiefenkarte setzt eine andere Form einer Spezialkamera voraus. Bei der ZCam [14] wird nicht nur die Helligkeit und Farbe aufgenommen, sondern auch mit einem Spezialsensor die Tiefeninformation. Das Tiefenbild hat dann aber eine niedrigere Bildauflösung als das Farbbild der eigentlichen Kamera. Es werden dann die im Folgenden vorgestellten Methoden zur Farbsegmentierung angewendet. Jedoch werden nun statt der Farb- und Helligkeitswerte die Werte der Tiefenkarte verwendet.

Es gibt eine Vielzahl von Algorithmen, die Farbinformationen verwenden. Es kann dabei noch weiter in Algorithmen unterschieden werden, die globale Eigenschaften im Bild oder dem Objekt analysieren oder lokale Informationen einzelner Bildpunkte verwenden.

Ein Beispiel für die Verwendung von globalen Informationen ist die Histogrammanalyse [15]. Zuerst werden die Bildpunktwerte quantisiert. Dies ist erforderlich, da bei einer 24−Bit Darstellung der Farbwerte deutlich mehr Farbwerte im Histogramm vorliegen (ca. 16 Mio.), als es Bildpunkte in gängigen Videoformaten gibt (z.B. bei CIF ca. 100.000). Nach der Quantisierung werden die Farbwerte aller Bildpunkte in ein Histogramm eingetragen. Ausgehend von den Maxima im Histogramm wird das Histogramm in einzelne Zonen aufgeteilt. Die Nummern der Zonen lassen sich dann wieder den korrespondierenden Bildpunkten als Segmentnummern zuordnen. Ein maßgeblicher Faktor für die Eigenschaft dieser Segmentierungsmethode liegt in der Quantisierung. Hier wird zuerst eine lineare Quantisierung ausgeführt und später eine nichtlineare auf das Histogramm angewendet.

Eine andere Methode der Quantisierung wäre die kombinierte Durchführung von nichtlinearer Quantisierung und Cluster-Bildung in der globalen Statistik. Dies wird z.B. in [16] beschrieben. Hier wird die Cluster-Bildung mit dem *K−Means*-Algorithmus [17] durchgeführt. In [18] ist die K−Means Funktion noch so erweitert, dass auch der örtliche Zusammenhang berücksichtigt wird.

Globale Eigenschaften eines Segmentes werden beim *Active-Contours*-Algorithmus [19] analysiert. Hier ist eine Startmarkierung vorausgesetzt. Die Startmarkierung ist eine geschlossene Linie, innerhalb derer das zu segmentierende Objekt oder ein Teil des Objektes liegt. Von den Bildpunkten, die innerhalb der Markierung liegen, werden globale Eigenschaften berechnet.

Die Markierungslinie wird anschließend so verschoben, dass eine Energiefunktion, die auf den globalen Eigenschaften basiert, minimiert wird. Die Markierung kann dabei nach innen oder nach außen verschoben werden. Die Verschiebung wird solange fortgesetzt, bis das Minimum der Energiefunktion erreicht ist. In einigen Implementierungen, wird die Verschiebungen nur an einigen Stützstellen berechnet. Der Konturverlauf zwischen den Stützstellen wird dann mit einer *Spline*-Funktion interpoliert [20].

Eine andere Form der Optimierung einer globalen Energiefunktion in Abhängigkeit von einer Segmentierungsmaske wird mit der Relaxation [2] implementiert. Hier ist das eigentliche Ziel die Maximierung der bedingten Wahrscheinlichkeit, mit der die Segmentmaske zu dem Bild gehört. Diese Wahrscheinlichkeit ist maximal, wenn die globale Energiefunktion minimal ist. Unter der Annahme, dass ein Bild ein Markov-Feld ist [21], lässt sich die Energiefunktion durch lokale Minimierung auch global minimieren. Deshalb wird bei der Relaxation die Segmentnummer eines Bildpunktes so umkonfiguriert, dass das lokale Potential minimiert wird. Im einfachsten Fall lässt sich das Potential aus den Differenzen zu den benachbarten Bildpunkten berechnen [22]. Es werden aber nur die Differenzen zu Bildpunkten mit gleicher Segmentnummer berücksichtigt, da die Bildpunktwerte innerhalb eines Segments voneinander abhängig sind. Sind die Bildpunktwerte voneinander unabhängig, das heißt, wenn sie unterschiedliche Segmentnummern haben, wird eine Konstante dem lokalen Potential hinzuaddiert. Für jeden Bildpunkt werden unterschiedliche Segmentnummern ausprobiert. Schließlich wird der Bildpunkt dem Segment zugeordnet, für den das lokale Potential minimal ist. Die lokale Optimierung wird idealer Weise mit einem Zufallsmuster auf unterschiedliche Bildpunkte angewendet. In der Praxis ist die Abarbeitungsfolge nicht zufällig, sondern oft regulär. Dadurch kann die Konvergenz in Richtung der Abarbeitungsfolge beschleunigt werden. Diese unnatürliche Bevorzugung der Ausbreitungsrichtung kann dadurch umgangen werden, dass die neuen Segmentnummern erst gültig werden, wenn alle Bildpunkte einmal bearbeitet wurden. In diesem Fall neigt die Segmentmaske aber zum Schwingen und konvergiert nicht. In jedem Fall werden mehrere Iterationen auf das Bild angewendet. Die Anzahl der Iterationen ist von der Qualität der Segmentmaske beim Starten der Relaxation abhängig. Deshalb eignet sich die Relaxation, um eine Feinkorrektur auf eine Segmentmaske anzuwenden. Es werden sozusagen lokale Spannungen durch fehlerhafte Verläufe von Segmentgrenzen entspannt.

Die Wahrscheinlichkeit, dass man bei der Optimierung der Energiefunktion in einem lokalen Minimum endet, ist recht groß. Wie in Abbildung 1.4 dargestellt ist, muss dann dem System wieder Energie zugeführt werden, damit die Potenzialbarriere mit der Höhe H überwunden werden kann. An-

schließend kann die Optimierung auch das globale Minimum erreichen.

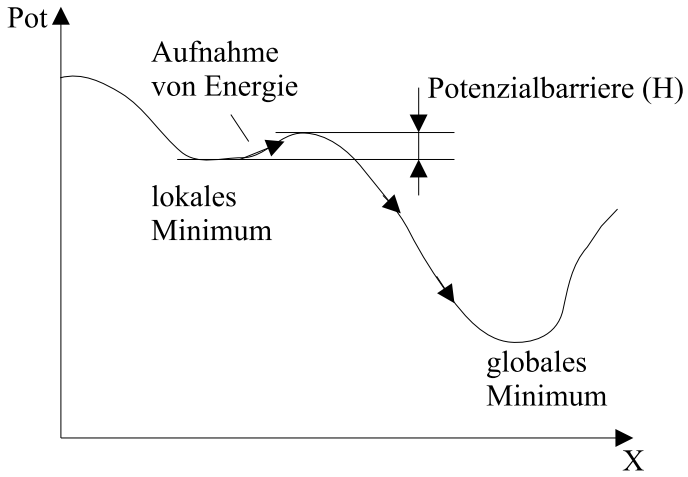


Abbildung 1.4: *Simulated Annealing* (simulierte Ausglühung): Durch langsames Abkühlen des Systems kann dem System aus der "Wärme" Energie hinzugeführt werden, um lokale Energieminima zu überwinden. Bei der Erstarrung befindet sich das System dann im globalen Minimum.

Die Energie, die benötigt wird, um das lokale Minimum zu überwinden, kann aus der "Umgebungstemperatur" entnommen werden. Die Umgebungstemperatur wird in der Berechnung des lokalen Potentials durch eine Zufallskomponente modelliert. Die Amplitude der Rauschkomponente entspricht dabei der Temperatur. Von Iteration zu Iteration wird die Temperatur langsam abgesenkt. Deshalb wird hierbei von simulierter Abkühlung oder *Simulated Annealing* gesprochen. Simulated Annealing wurde dabei ursprünglich bei der Simulation des Erstarrungsprozesses von Stahl eingesetzt.

Wie in [21] beschrieben wurde, kann die Relaxation noch um einen Linienprozess erweitert werden. Im Sinne der Bildverarbeitung heißt das, dass Konturverläufe zwischen Segmenten, die kompliziert und daher unerwünscht sind, mit einem Strafpotential belegt werden. In Abbildung 1.5 sind einige Konturverläufe dargestellt, die unerwünscht sind.

Durch die oft hohe Anzahl an Iterationen ist die Relaxation ein Algorithmus, der eine sehr hohe Rechenleistung benötigt. In [23] wird, um die Rechenkomplexität der Relaxation zu reduzieren, die Relaxation auf Blöcke angewendet. Dabei wird mit Berücksichtigung eines Linienprozesses der Konturverlauf eines Blockes in einem Schritt beurteilt. Der bildpunktgenaue Konturverlauf wird dann später mit einer Relaxation, die nur auf Blöcke, die ein Konturstück beinhalten, angewendet.

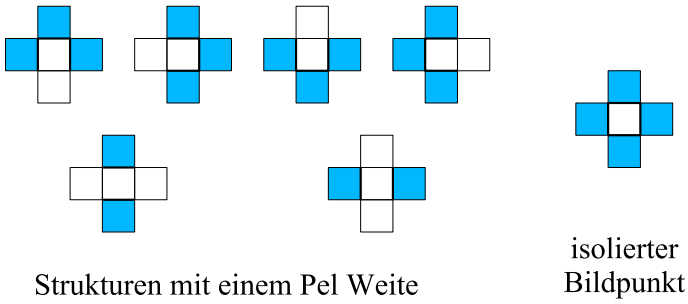


Abbildung 1.5: Beim Linienprozess in der Relaxation werden ungünstige Konturverläufe zwischen Segmenten mit einem Strafpotential beaufschlagt. Dargestellt sind Beispiele für ungünstige Konturverläufe. Das Strafpotential hängt dabei von der Art des Konturverlaufs ab.

Wie bereits oben erwähnt wurde, ist die Relaxation ein Werkzeug, um Segmentgrenzen zu verschieben. Dabei wird einem Segment der Bildpunkt "weggenommen" und einem anderen hinzugefügt. Ein anderes Werkzeug, dass auch die Ähnlichkeit zu den Nachbarn beurteilt, ist der Wasserscheide-Algorithmus (*Watershed*) [24]. Bei dem Wasserscheide-Algorithmus handelt es sich um ein sehr leistungsfähiges und effizientes Werkzeug zur Objektsegmentierung, das in vielen Algorithmen verwendet wird [25][26][27][28][29].

Abbildung 1.6 zeigt die Funktionsweise des Wasserscheide-Algorithmus, der anhand seiner Analogie zur Geologie erklärt wird. Gezeigt ist das Gradientengebirge mit drei Tälern, den Kerngebieten der Segmente. An den Segmentgrenzen (Objektkanten) ergeben sich dagegen hohe Gradientenwerte. Jedes Tal bzw. lokale Minimum erhält eine eigene Segmentnummer und ist gleichzeitig die Startregion für den folgenden Ausbreitungsprozess. Während des Ausbreitungsprozesses werden die Täler mit Wasser aufgefüllt. Wenn die dabei entstehenden Seen drohen zusammenzulaufen, wird zwischen ihnen ein Damm errichtet. Wenn das gesamte Gebirge überflutet ist, bleiben nur noch die Dämme zwischen den Seen übrig. Die Dämme sind die Grenzen zwischen den Segmenten.

Dadurch, dass jedes lokale Minimum ein eigenes Segment erzeugt, erhält man eine Segmentierung mit sehr vielen kleinen Segmenten. Die Segmentmaske ist stark übersegmentiert, und die Segmente haben keine semantische Bedeutung.

Um die Übersegmentierung zu verhindern, wird das Bildmaterial fast immer vor der Identifikation von lokalen Minima vereinfacht. Dazu können Tiefpassfilter oder Rangordnungsfilter eingesetzt werden. Diese Filter sind aber nicht so gut geeignet, da sie die Kanten zwischen den Objekten verän-

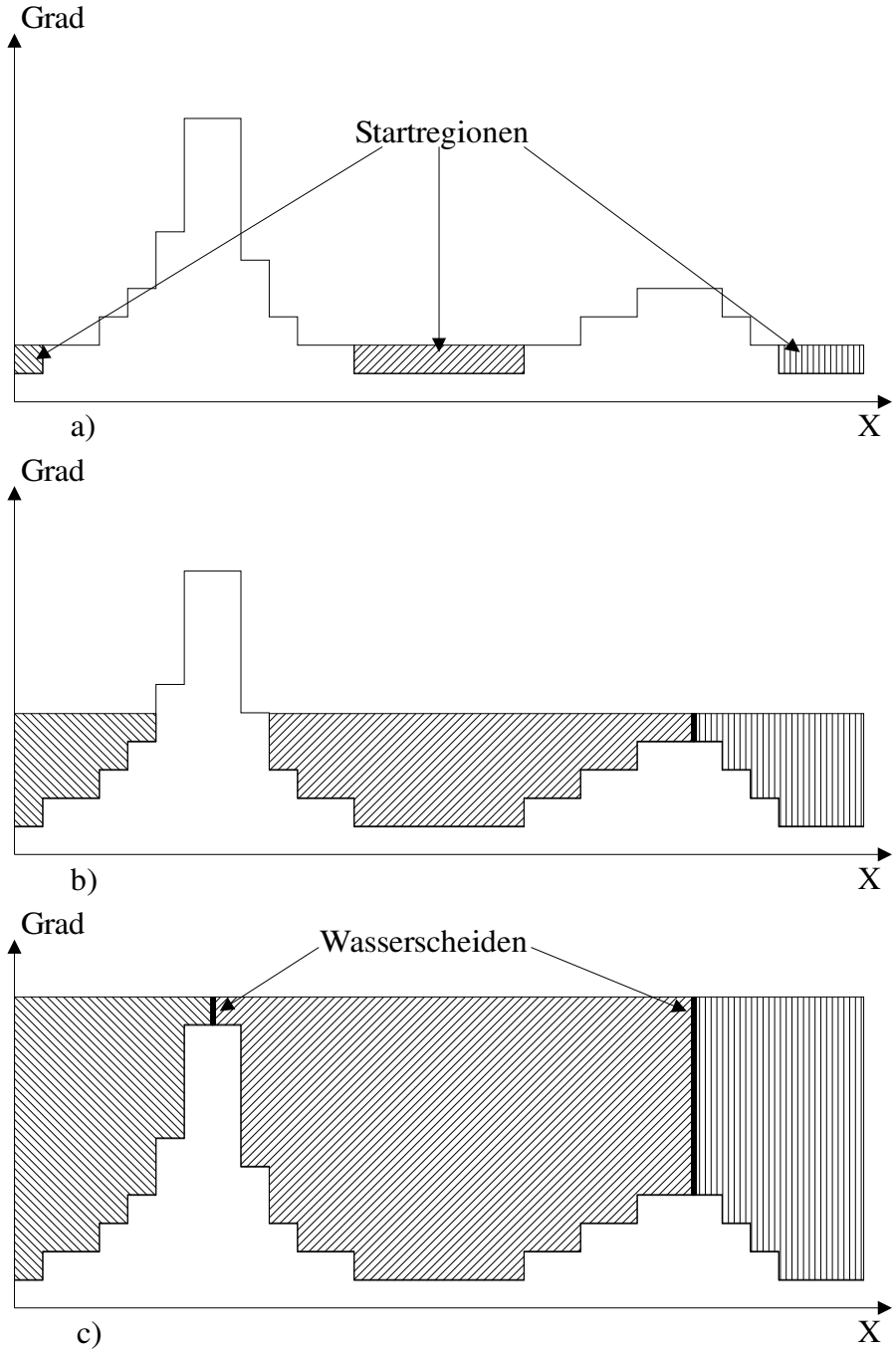


Abbildung 1.6: Darstellung des Wasserscheide-Prinzips. a) Die lokalen Minima (Kernzonen) sind die Startregionen der gleichnamigen Segmente. b) Die Täler der Regionen werden geflutet. c) Das Gradientengebirge ist vollständig überflutet. Die Grenzen zwischen den Seen (Wasserscheiden) bilden die Segmentgrenzen.

dern und andererseits zwar die lokalen Minima reduzieren, aber nicht auflösen. Wesentlich besser geeignet sind morphologische Filter, die sowohl lokale Minima auslöschen können, als auch den Funktionsverlauf an den Kanten erhalten. Abbildung 1.7 stellt die Funktion eines morphologischen Filters, dem *Leveling* [30][31], dar. Die Funktion F wird dabei unter Verwendung der Marker-Funktion M "flacher" gemacht. Lokale Maxima und Minima der Bildpunktwerte werden gelöscht.

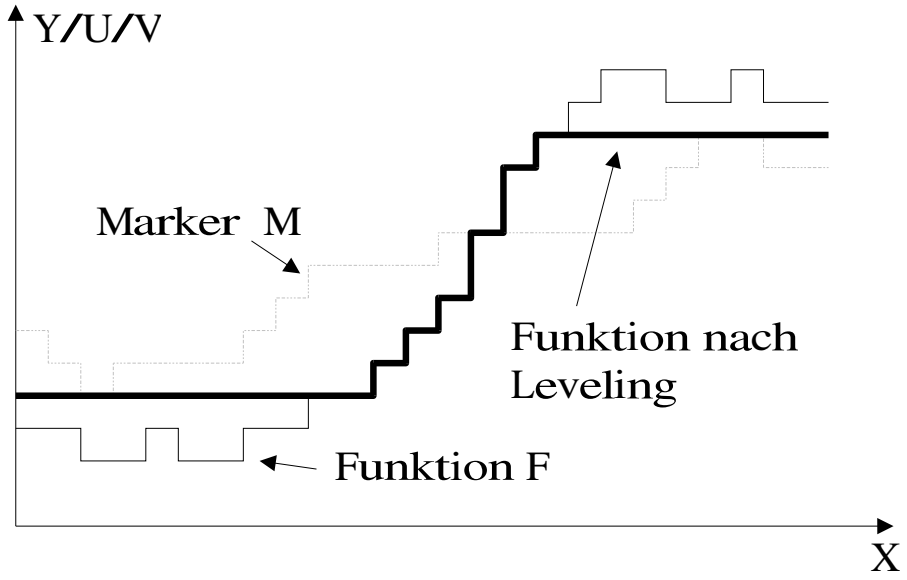


Abbildung 1.7: Funktionsweise des morphologischen Filters *Leveling*

Die Funktionsweise soll anhand des *morphologischen Schließens*, dargestellt im linken Teil von Bild 1.7, erklärt werden. In diesem Bereich ist die Funktion F kleiner als die Marker-Funktion M, so dass lokale Minima geschlossen werden. Ausgehend von den lokalen Minima von M wird das Maximum von F an der aktuellen Position und dem Minimum der Werte M in der Nachbarschaft der aktuellen Position ermittelt.

$$M_x = max(F_x, min_{nx \in Nhood_x}(M_n x)) \tag{1.1}$$

Dieser Wert wird dann wieder in M gespeichert. Wenn die Verarbeitung von den Minima ausbreitend geschieht, können sich lokale Minima von M soweit in ihre Umgebung ausbreiten, bis der Wert von F größer wird als der sich ausbreitende Wert von M. Von den Stellen an ist der Funktionsverlauf vom neu berechneten M mit dem von F identisch. Dies entspricht exakt dem gefor-

derten Verhalten, dass lokale Minima gefüllt werden, ohne dass der Kantenverlauf sich ändert. Die Startwerte von M können z.B. durch eine Dilatation von F ermittelt werden.

$$M_x = max_{nx \in Nhood_x}(F_n x) \tag{1.2}$$

Flattenings, zu denen das Schließen gehört, werden wie im Beispiel beschrieben dazu benutzt, um lokale Minima zu löschen. Flattenings können umgekehrt auch zum Auflösen lokaler Maxima verwendet werden. Diese Funktion wird als (morphologisches) Öffnen bezeichnet:

$$M_x = min(F_x, max_{nx \in Nhood_x}(M_n x)) \tag{1.3}$$

Neben den einseitigen Flattenings gibt es auch die im Bild 1.7 gezeigte beidseitige Erweiterung, dem s.g. Leveling (beidseitiges Abschneiden von Minima und Maxima).

Die Entwicklung der morphologischen Filter ist wesentlich durch die Arbeiten am *Centre de Morphologie Mathématique* (CMM) an der *Ecole des Mines* geprägt worden. Eine Übersicht über die morphologischen Filter ist in [32] aufgeführt.

Neben der Vereinfachung der Bildstruktur existiert noch eine andere Methode, die Übersegmentierung zu reduzieren. Der Wasserscheide-Algorithmus kann auch so angewendet werden, dass er keine neuen Segmente erzeugt, wenn ein neues lokales Minimum entdeckt wird. In diesem Fall kann das lokale Minimum z.B. einfach überflutet werden. Man benötigt dann so genannte *Startmarker*, die durch den Wasserscheide-Algorithmus expandiert werden. In diesem Fall stellt sich dann aber wieder das Problem, wie man mit einem automatischen Prozess die Startmarker erzeugen kann. In Abbildung 1.8 wird an einem einfachen Beispiel die Leistungsfähigkeit des Wasserscheide-Algorithmus gezeigt. In a) ist das Bild mit manuell gesetzten Startmarkern dargestellt. Marker für den Hintergrund sind dabei schwarz (dunkel) und Marker für den Vordergrund grün (hell) dargestellt. Bild b) zeigt dann die resultierende Segmentierungsmaske nach der Ausbreitung der Startmarker mit dem Wasserscheide-Algorithmus und c) das Vordergrundobjekt.

Neben dem Wasserscheide-Algorithmus wird auch das RSST-Verfahren (siehe Seite 14) eingesetzt. Bei der Farbsegmentierung unter Einsatz des RSST-Verfahrens [33][34] werden einzelne, benachbarte Regionen rekursiv in der Reihenfolge ihrer Ähnlichkeit hinsichtlich der Farbwerte zusammengefasst. Der Zusammenfassungsgraph hat dabei die Struktur eines binären Baums. Zum Anfang des Zusammenfassungsprozesses bildet jeder Bildpunkt eine eigene Region. Als Abbruchkriterium für die rekursive Zusammenfassung dient die Anzahl der Segmente. Im Gegensatz zum Wasserscheide-Algorithmus wächst eine Region dabei immer um eine komplette weitere

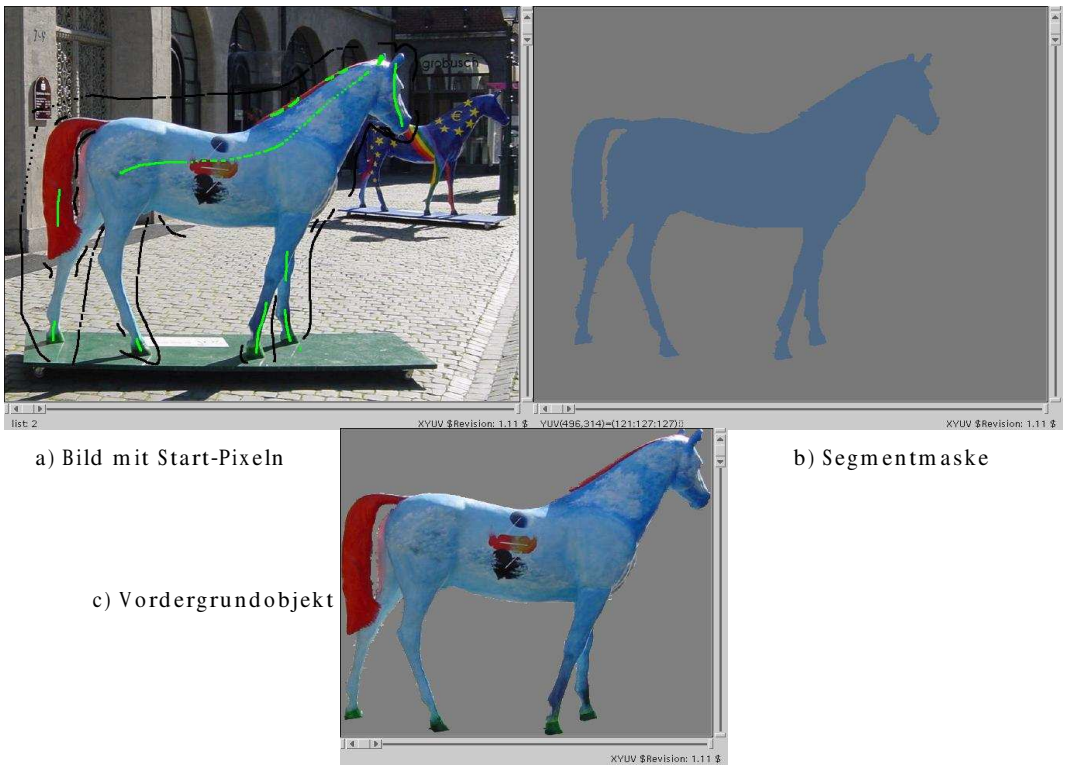


Abbildung 1.8: Semi-automatischer Wasserscheide-Algorithmus: a) Originalbild mit manuell gesetzten Startmarkern für Vordergrund (grün/hell) und Hintergrund (schwarz), b) Segmentmaske nach der Wasserscheide-Ausbreitung der Startmarker, c) Vordergrund.

Region und nicht um einzelne Bildpunkte. Der RSST Algorithmus kann aber auch benutzt werden, um die Segmente einer übersegmentierten Farbsegmentierungsmaske, die z.B. mit einem Wasserscheide-Algorithmus erzeugt wurden, zusammenzufassen [25]. Dabei muss, wie oben bereits beschrieben, nicht unbedingt die Farbe als Zusammenfassungskriterium benutzt werden, sondern z.B. auch bis zum Erreichen einer bestimmten Anzahl an Segmenten Farbinformation und dann später Bewegungsinformation [35].

Einer andere Methode der Zusammenfassung von Segmenten ist der hierarchische Wasserscheide-Algorithmus. Hier werden auch benachbarte Segmente in Abhängigkeit ihrer Ähnlichkeit zusammengefasst. Allerdings werden die Ähnlichkeiten zwischen den Segmenten vor dem Zusammenfassen berechnet und verändern sich nicht mit dem Zusammenfassen von Segmenten. So ist der Zusammenfassungsgraph auch ein Baum, der allerdings nicht binär sein muss.

Neben den bisher vorgestellten Analysemethoden gibt es auch Systeme zur Objektsegmentierung, die mehrere Analysemethoden kombinieren. Hier sind im Wesentlichen zwei Ansätze vorhanden. Im ersten Ansatz werden die Analysemethoden voneinander unabhängig ausgeführt und liefern dementsprechend auch unabhängige Segmentmasken. Mit einer *Inference-Engine* wird dann aus den unabhängigen Segmentmasken die endgültige Objektmaske erstellt. Ein Beispiel hierfür ist das COST *Analysis Modell* (AM) [36]. Das COST AM, das Referenzsystem der europäischen Projektgruppe COST[1] 211, integriert dabei einen RSST-basierten Farbsegmentierungsalgorithmus (Seite 22) mit einer Bewegungsanalyse. Bei der Bewegungsanalyse kann zwischen zwei Algorithmen ausgewählt werden, der RSST-basierten Bildung von unterschiedlichen Bewegungsmodellen für die einzelnen Objekte (Seite 14) oder dem CDM (Change-Detection-Mask)-Algorithmus (Seite 12). In späteren Weiterentwicklungen ist dann der CDM-Algorithmus um eine globale Bewegungskompensation und eine Schattenerkennung [37] erweitert worden. Außerdem ist die Farbanalyse gegen den in Kapitel 1.6 beschriebenen Algorithmus ausgetauscht worden. Mit dem *Rule-Based-Processor* werden dann die Masken der Farbanalyse mit der Maske der Bewegungsanalyse kombiniert.

Eine andere Möglichkeit ist die Verwendung eines hierarchischen Objektmodells. Dies wird z.B. in [25] beschrieben. Hier wird für die Farbanalyse der Wasserscheide-Algorithmus (Seite 19) eingesetzt. Im Folgenden werden die Farbsegmente mit einem RSST-Algorithmus zusammengelegt. Dabei wird zuerst die Farbähnlichkeit benutzt und anschließend die Bewegungsinformation. In [11] wird zusätzlich ein hierarchisches Objektmodell beschrieben, das aus Farbsegmenten, Elementen mit einfachen Formen und Objekten mit zusammenhängender Bewegung besteht. Der Algorithmus benutzt den in Kapitel 1.6 beschriebenen Algorithmus zur Farbsegmentierung, eine Formanalyse und eine modellbasierte Bewegungsanalyse.

Ein abstraktes Segmentierungssystem ist mit dem *Kernel of Analysis for new Multimedia Technologies* (KANT) beschrieben [38]. Das COST AM ist eine Implementierung des KANT-Modells. Über das COST AM hinaus erlaubt das KANT-Modell noch die Möglichkeit, dass die Bewegungsanalyse bereits Ergebnisse der Farbanalyse verwendet. So wird neben der Verwendung einer Inference-Engine auch die Bildung von hierarchischen Objektmodellen ermöglicht. Außerdem umfasst das KANT-Modell noch eine benutzerunterstützte Segmentierung (*supervised Segmentation*).

Die QIMERA-Plattform [34] ist eine Integrationsumgebung für Segmentierungsalgorithmen. Es dient im Wesentlichen zum Vergleich der unter-

[1] European Cooperation in the field of Scientific and Technical Research

schiedlichen Segmentierungsalgorithmen in einer einheitlichen grafischen Benutzeroberfläche. Im späteren Verlauf wird diese Plattform auch benutzt werden, um eine Inference-Engine zu entwickeln, mit der die Ergebnisse der Algorithmen kombiniert werden können. Die QIMERA-Plattform ist außerdem im SCHEMA Referenzsystem [39][40] integriert worden. Mit dieser Software wird eine Ähnlichkeitssuche für Bilder implementiert (siehe Bild 1.9). Das Besondere dieser Suchmaschine ist die Verwendung von Objekten und den zugehörigen Formbeschreibungen.

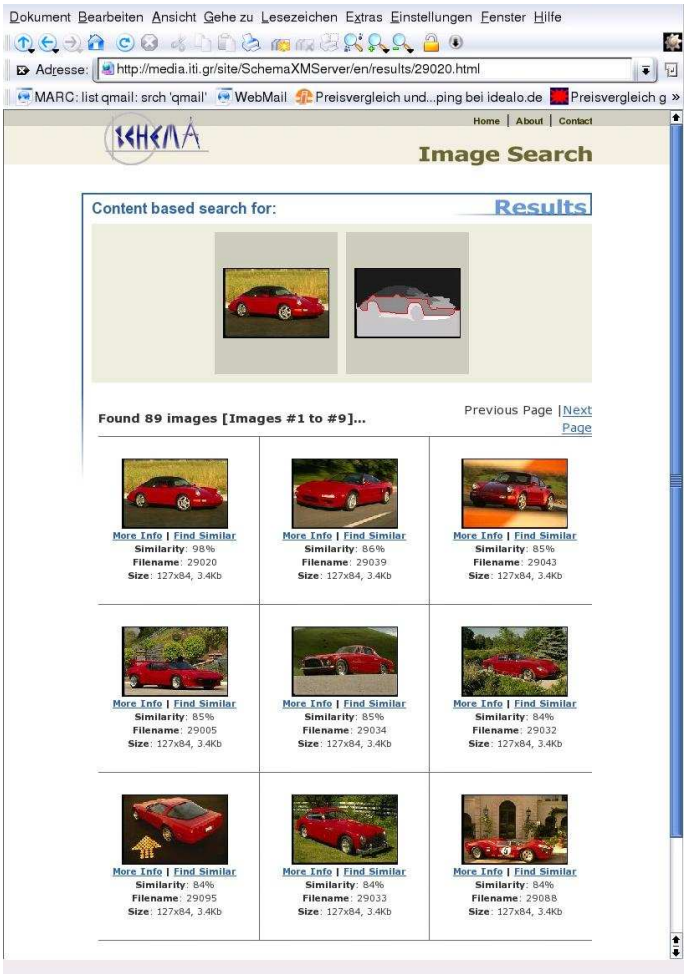


Abbildung 1.9: Screenshot der SCHEMA-Bildsuchmaschine. Das Besondere dieser Suchmaschine ist Verwendung von Objekten in Bildern und Formdeskriptoren zur Ähnlichkeitssuche.

1.4.4 Implementierung ausgewählter Bildverarbeitungsfunktionen für die Bildanalyse

In diesem Abschnitt werden nun einige Architekturen vorgestellt, die für Ausbreitungsprozesse in der Bildsignalverarbeitung eingesetzt werden. Den Ausbreitungsprozessen kommt hier eine besondere Rolle zu, da sie das charakteristische Element der Aufgabenstellung ist, zusammenhängende Elemente zu identifizieren und zu verarbeiten.

Einer der einfachsten Ausbreitungsprozesse ist das Markieren (Nummerieren) einer zusammenhängenden Fläche. Hier wird mit einem Bildpunkt angefangen, der mit einer Segmentnummer markiert wird. Nun sollen alle Bildpunkte, die mit dem Bildpunkt zusammenhängen und das Homogenitätskriterium erfüllen, mit der selben Nummer markiert werden. Eine Architektur, die hierfür eingesetzt werden kann, ist der PIMM1 [41]. Dabei wird ein rekursives Filter eingesetzt, das die Homogenität mit den vorher bearbeiteten Bildpunkten überprüft. Wird eine Homogenität festgestellt, dann bekommt der Bildpunkt die Segmentnummer des Bildpunktes, zu dem das Homogenitätskriterium erfüllt ist.

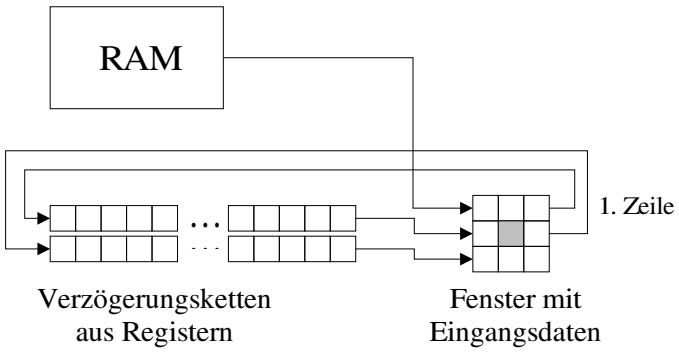


Abbildung 1.10: Verzögerungsketten für die horizontale Verschiebung eines Eingangsbildbereichs. Die Daten aus der 1. und 2. Zeile werden in den Verzögerungsketten so verzögert, dass sie für die Bearbeitung der nächsten Zeilen wieder aus den Verzögerungsketten geladen werden.

Wie Abbildung 1.10 zeigt, wird im PIMM1 mit einer 3x3 Bildpunktnachbarschaft gearbeitet. Das 3x3 Bildpunkt große Eingangsfenster wird horizontal über das Bild geschoben, bis die Zeile zu Ende ist. Dann wird am Anfang der nächsten Zeile fortgefahren. Der beim Schieben neu hinzukommende Bildpunkt der dritten Zeile wird aus dem Bildspeicher geladen. Der Bildpunkt aus der dritten Zeile, der aus dem Eingangsfenster wegfällt, wird

in eine Verzögerungskette geschrieben. Nachdem der nächste Bildpunkt bearbeitet wurde, kommt ein neuer Bildpunkt in die Verzögerungskette, und alle vorhergehenden Bildpunkte in der Verzögerungskette werden um eine Position weitergeschoben. Die Verzögerungskette ist nun so lang, dass der Bildpunkt wieder aus der Verzögerungskette herauskommt, wenn er in der nächsten Zeile in das 3x3 Eingangsfenster geladen werden muss. Der verzögerte Bildpunkt wird dann in die zweite Zeile des Eingangsfensters geladen. Zwischen der zweiten und der ersten Zeile befindet sich ebenfalls eine Verzögerungskette. So muss immer nur ein Bildpunkt aus dem Bildspeicher geladen werden, wobei die passenden Bildpunkte der ersten und zweiten Zeile aus den Verzögerungsketten geladen werden können. Da die Segmentnummer nur aus den bereits bearbeiteten Bildpunkten übernommen werden kann, werden bei dieser Operation nur zwei Zeilen benötigt. Für andere Bildverarbeitungsfunktionen werden aber alle drei Zeilen des Eingangsfensters benötigt. Mit dieser Abarbeitungsfolge lassen sich nur Bildpunkte markieren die im *kausalen* Bereich ihrer Nachbarn liegen. Deshalb sind unter Umständen mehrere Arbeitsschritte nötig, um die gesamte Fläche zu markieren. Eine genaue Beschreibung dafür ist in Kapitel 2.1.4 gegeben. In diesem Kapitel stehen die Architektur und die Eigenschaft, dass pro Durchlauf über das Bild jeder Bildpunkt nur einmal geladen wird, im Vordergrund.

In [24] ist eine andere Variante der Implementierung vorgeschlagen. Hier wird die Implementierung eines Wasserscheide-Algorithmus beschrieben. Diese Architektur lässt sich aber genauso auf das Markieren von Segmenten anwenden und wird deshalb hier auch an diesem Beispiel erläutert.

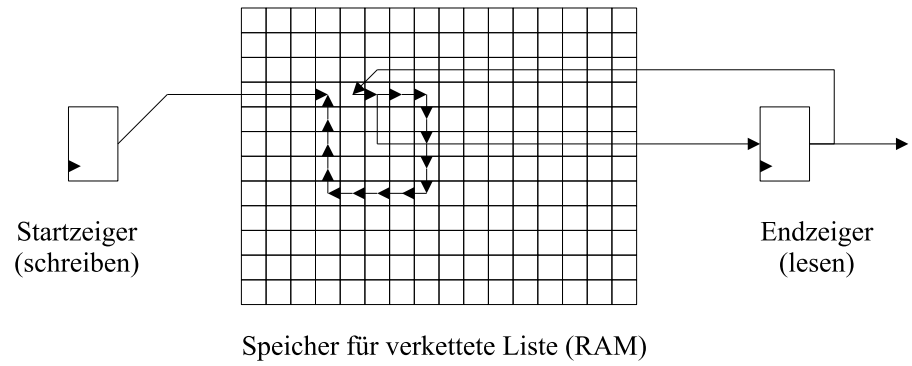


Abbildung 1.11: FIFO-Architektur für Bildverarbeitungsfunktionen mit Ausbreitungscharakteristik. Jeder Bildpunkt wird einmal bearbeitet. Für seine Ausbreitung muss die komplette Nachbarschaft in das Eingangsfenster geladen werden.

Wie in Abbildung 1.11 zu sehen ist, wird hier mit einer FIFO (*First-In-First-Out*-Speicher) gearbeitet. In die FIFO werden die Nachbarn des zuletzt ausgelesenen Bildpunktes eingetragen, die

- das Homogenitätskriterium (Segmentzugehörigkeitskriterium) erfüllen und
- noch nicht in der FIFO eingetragen waren.

Für die Bildpunkte, die aus der FIFO ausgelesen werden, wird die komplette Nachbarschaft geladen und für die Nachbarn die beiden oben genannten Kriterien geprüft. Wichtig ist an dieser Stelle, dass bei der Verarbeitung mit der FIFO-Architektur jeder Bildpunkt nur einmal bearbeitet wird. Für jeden Bildpunkt muss dann immer die gesamte Nachbarschaft geladen werden. Die FIFO hat im Gegensatz zu den Verzögerungsketten der vorher beschriebenen Architektur keine feste Länge. Deshalb werden für die FIFO, die eine Maximalgröße von der Anzahl der Punkte im Bild hat, ein Start- und ein Endzeiger benötigt.

Für den Wasserscheide-Algorithmus ist eine erweiterte Architektur, basierend auf der FIFO-Architektur, in [42] vorgeschlagen worden. Im Gegensatz zur einfachen FIFO-Architektur sind hier, wie in Abbildung 1.12 dargestellt, mehrere FIFOs vorhanden. Die Nachbarbildpunkte werden nach ihrer Priorität in unterschiedliche FIFOs eingetragen, die dann mit abnehmender Priorität nacheinander ausgelesen werden. Diese Architektur wird als hierarchische FIFO (HFIFO) bezeichnet.

Die Gesamtanzahl der FIFO-Einträge entspricht hier auch der Anzahl der Punkte im Bild. Der FIFO-Speicher ist auch zweidimensional mit den gleichen Dimensionen wie das Bild implementiert. Jedes FIFO-Element gehört zu einem Bildpunkt, so dass die Koordinaten des FIFO-Elements die gleichen Werte haben, wie die des korrespondierenden Bildpunktes. Eine Referenz zum Bildpunkt muss also nicht gespeichert werden. Der FIFO-Speicher entspricht einer verketteten Liste. Das heißt, dass jedes FIFO-Element die Adresse zum nächsten Element der FIFO enthält (und dadurch auch die Adresse zum nächsten Bildpunkt). Jede FIFO der hierarchischen FIFO hat einen Startzeiger, der auf das als nächstes zu lesende Element dieser Hierarchiestufe zeigt und einen Endzeiger, der auf das Element zeigt, an dem das nächste Element dieser Hierarchiestufe angehängt wird.

Der FIFO-Speicher kann als RAM Block implementiert werden, um Chipfläche zu sparen. Hingegen ist es sinnvoll, die Start- und Endzeiger als Register auszuführen. Dadurch kann parallel auf alle Start- und Endzeiger zugegriffen werden, was für die Initialisierung deutliche Vorteile hat. Außerdem

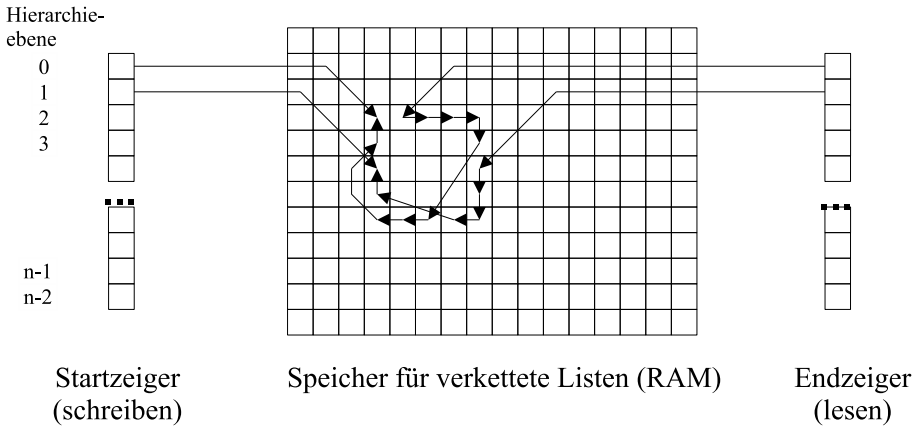


Abbildung 1.12: Hierarchische FIFO-Architektur nach Lemonnier für Ausbreitungsprozesse mit variabler Geschwindigkeit. Jeder Bildpunkt kann nur in genau einer FIFO einmal eingetragen werden, so dass er nur einmal bearbeitet wird. Für die Ausbreitung des Bildpunktes müssen dann alle seine Nachbarn in das Eingangsfenster geladen werden.

wird so nur ein Speicherzugriff benötigt, um die FIFO-Daten zu lesen oder zu schreiben, da die Zeiger direkt verfügbar sind.

Basierend auf der hierarchischen FIFO-Architektur werden dann in dieser Arbeit weitere Konzepte der Speicherverwaltung und Implementierung der Verzögerungsketten entwickelt und diskutiert.

1.4.5 Werkzeuge zum Entwurf der Algorithmen

Neben den algorithmischen Grundlagen ist für diese Arbeit auch eine Betrachtung des Standes der Technik hinsichtlich gängiger Entwurfsabläufe relevant. Hier wird in der Regel mit dedizierten Software-Funktionen gearbeitet, bei der jede Bildverarbeitungsfunktion direkt implementiert wird. Häufig wird dabei mit einer Fließkommarepräsentation für die Darstellung von Bildpunktwerten gearbeitet. Zum einen wird dadurch eine Degradierung der Ergebnisse durch die Quantisierung von Ganzzahlen ausgeschlossen. Zum anderen braucht sich der Entwickler von Algorithmen in der Regel keine Gedanken über mögliche Überläufe des Bereichs der Bildpunktwerte zu machen.

Für die Entwurfskomplexität bedeutet diese Art der Algorithmenentwicklung, dass jede Bildverarbeitungsfunktion immer wieder aufs Neue implementiert wird und nur in den seltensten Fällen eine Wiederverwendung stattfindet. Ergebnisse betreffend der Rechen- und Implementierungskomplexität

hängen stark von den Fähigkeiten des Entwicklers und dem gewählten Grad der Optimierung ab. Sie lassen sich nicht mit anderen Algorithmen von anderen Entwicklern vergleichen.

In anderen Fällen wird innerhalb einer Arbeitsgruppe eine Bibliothek entwickelt, die grundlegende Bildverarbeitungsfunktionen enthält. Dies setzt auch die Verwendung eines einheitlichen Datenmodells voraus, das bei der Implementierung der Bibliothek zugrunde gelegt wurde. So sind neben der Speicherverwaltung (*interleaved* oder *non-interleaved*) auch die Datentypen (Byte, Word, Integer oder Float) festgelegt. Des Weiteren ist die Anzahl der Bildkanäle, die verarbeitet werden können, festgelegt. Die gleiche Funktion benötigt bei unterschiedlichen Datenmodellen oder veränderter Anzahl der Bildkanäle unterschiedliche Implementierungen. Wird die Bibliothek um eine neue Funktion erweitert, muss die gesamte Funktion neu geschrieben werden. Ein Beispiel für diesen Entwurfsablauf ist [43]. Hier liegen eine große Zahl an vordefinierten Bildverarbeitungsfunktionen und Filtern bereits als kompilierte Objektdateien vor. Diese können dann in einer Entwicklungsumgebung, die auf einem Perl-Interpreter basiert, zur Laufzeit kombiniert werden. Die Wiederverwendung der implementierten Bildverarbeitungsfunktionen ist hier bereits recht hoch. Über die Verwendung im eigenen Labor hinaus wurde diese Bibliothek auch im Entwicklungspaket Aphelion [44] vertrieben. In der kommerziellen Variante ist aber die Erweiterung der Bibliothek durch das beim Benutzer nicht vorhandene Know-How hinsichtlich der Interfaces der Bibliothek nur sehr eingeschränkt möglich. Komplexitätsmessungen können nur bedingt von einem Algorithmus auf einen anderen übertragen werden. Die Varianz an Optimierung und Programmierstil ist schon deutlich gegenüber dedizierten Implementierungen reduziert. Da aber diese Art von Bibliothek über die Zeit gewachsen ist und unterschiedliche Entwickler daran beteiligt waren, lässt sich eine Vergleichbarkeit nicht garantieren.

Ein anderer Weg, der bei Entwicklungsumgebungen eingeschlagen wird, ist mit Envision bzw. SPW von Cadence [45][46] implementiert. Der Umfang an Bildverarbeitungsfunktionen umspannt gängige Filter und Komponenten, die häufig in der Videokompression eingesetzt werden. Funktionen zur Bildanalyse sind nicht vorhanden. Bei diesem Entwurfswerkzeug liegt der Schwerpunkt auf der grafischen Benutzeroberfläche, und der automatischen Umsetzung des Systems in ein Hardware-Modell. Erweiterungen des Funktionsumfangs sind hier zwar möglich, aber wegen des recht komplizierten Speichermodells nicht einfach durchzuführen. Allerdings können hier unterschiedliche Entwurfsmodelle integriert werden. So kann das Software-Modell für die Algorithmenentwicklung mit einem Modell zur Timing-Simulation bzw. zum Hardware-Entwurf ergänzt werden. Ergebnisse bezüglich der Rechenkomplexität können hier bedingt miteinander verglichen werden. Die Vergleichbar-

keit kann aber auch hier nicht garantiert werden.

Im QIMERA-Projekt [34] wird ebenfalls mit einer grafischen Benutzeroberfläche gearbeitet. Hier liegt das Hauptaugenmerk bei der Integration von fertigen Algorithmen zur Objektsegmentierung in eine einheitliche Test- und Simulationsumgebung. Im Vergleich zu den bisher vorgestellten Entwurfsumgebungen ist hier die Granularität der Module deutlich gröber. Die Entwicklung der einzelnen Analysemethoden ist im Wesentlichen abgeschlossen oder wird zumindest nicht in dieser Umgebung weitergeführt. Algorithmenentwicklung in dieser Umgebung konzentriert sich auf die Kombination der einzelnen Segmentierungsmasken in einer *Inference-Engine*. Das QIMERA-System ist hier erwähnt, um ein Beispiel für Entwurfswerkzeuge auf einer höheren Ebene (Auswahl eines kompletten Algorithmus) zu geben.

Eine weitere Variante von Methoden für die Algorithmenentwicklung in der digitalen Bildsignalverarbeitung ist die ImageLib [47] von Intel. Hier besteht die Bibliothek aus Filterfunktionen. Die Filter sind mit einer Funktion implementiert die für Intel CPUs optimiert ist. Die Erweiterbarkeit der Bibliothek beschränkt sich auf den Entwurf von Filtern die in dieser Arbeit der Intra-Adressierung entsprechen. Die Filter lassen sich aber leicht implementieren. Letztendlich werden nur die Funktionen zum Filtern eines Bildpunktes neu geschrieben und optimiert und dann mit einer Funktion, die das Filter auf alle Bildpunkte des Bildes anwendet, kombiniert. Aus diesem Konzept ergibt sich eine deutlich höhere Wiederverwendung der Bibliothekselemente und auch eine hohe Vergleichbarkeit der Laufzeitkomplexität, da wesentliche Teile der Bildverarbeitungsfunktion wieder verwendet werden. Dieses Konzept der Bibliothek ist im Wesentlichen eine Untermenge des in dieser Arbeit vorgestellten Modells zur Verarbeitung von Bildern. Die Zielsetzung bei der ImageLib ist aber insbesondere die Optimierung des Codes für Intel CPUs.

1.4.6 Allgemeine Architekturen für die Bildverarbeitung

Ein dritter, wesentlicher Gesichtspunkt dieser Arbeit ist die Entwicklung eines Konzepts, mit dem das Bildverarbeitungsmodell als Hardware-Architektur implementiert werden kann. Deshalb wird in diesem Kapitel ein kurzer Überblick über den Stand der Technik hinsichtlich Architekturen gegeben. Hier werden die Architekturen nach ihrem Grad und der Art der Parallelisierung sowie nach ihrer Flexibilität klassifiziert. Bei der Parallelisierung kann in *Single Instruction Single Datapath* (SISD), *Single Instruction Multiple Datapaths* (SIMD) und *Multiple Instructions Multiple Datapaths* (MIMD) unterschieden werden.

Bei SISD Architekturen ist ein Datenpfad vorhanden, in dem eine Operation auf einen Satz Daten angewendet wird. Eine Parallelisierung findet hier nur in Form einer Fließbandverarbeitung statt. In SIMD Architekturen können dann in mehreren Datenpfaden unterschiedliche Daten verarbeitet werden. Dabei ist charakteristisch, dass alle Datenpfade die gleiche Operation auf die unterschiedlichen Daten anwenden. Diese Art der Parallelisierung setzt voraus, dass die Daten unabhängig voneinander verarbeitet werden können. Auf der anderen Seite benötigt man nur ein Steuerwerk, um alle Datenpfade zu kontrollieren. In machen Operationen ist es nötig, bedingte Verzweigungen einzusetzen. In diesem Fall können alle Datenpfade wieder exakt die gleiche Sequenz an Befehlen benutzen. Dabei werden einige Befehle nicht in allen Datenpfaden ausgeführt. Die Bedingung, ob ein Befehl ausgeführt wird oder nicht, kann dabei in der Verarbeitungseinheit des Datenpfades berechnet werden. Der Nachteil dieses Prinzips ist, dass alle möglichen Verzweigungen sequentiell aufgerufen werden müssen, aber dann teilweise nicht ausgeführt werden. Dieses Konzept wird als autonome SIMD-Architektur bezeichnet [48].

Noch mehr Flexibilität bei gleicher Parallelisierung hat man bei MIMD Architekturen. Hier sind die Datenpfade komplett unabhängig voneinander. Es sind nicht nur die Daten unterschiedlich, die verarbeitet werden, sondern auch die Operationen, die auf die Daten angewendet werden. Trotz der größeren Flexibilität muss man hier auch die Datenabhängigkeiten berücksichtigen. Man hat aber größere Freiheit durch die Wahl der Reihenfolge der Daten und der zugehörigen Operationen.

Bei Mischformen zwischen SIMD und MIMD sind mehr Datenpfade vorhanden als Steuereinheiten. Dies bedeutet, dass eine Steuereinheit die Verarbeitung in mehreren Datenpfaden kontrolliert. Es entstehen mehrere SIMD-Einheiten, die zusammen ein MIMD-System ergeben. Ist die Zuordnung der Datenpfade zu den Steuereinheiten statisch, so wird dies als eine *Clustered*-SIMD-Architektur [49] bezeichnet. Ist die Zuordnung dynamisch, spricht man von einer assoziativen Steuerung [50].

Eine andere Art der Klassifikation von Architekturen ist die Unterteilung in programmierbare Prozessoren, konfigurierbare Prozessoren und dedizierten Prozessoren oder ASICs (anwendungsspezifische Integrierte Schaltungen). Zu den programmierbaren Prozessoren gehören Standardprozessoren, die als *Complex-Instruction-Set-Computer* (CISC) oder *Reduced-Instruction-Set-Computer* (RISC) ausgeführt sind. CISC Prozessoren erlauben die Verarbeitung von komplexen Befehlen, die mehrere Taktzyklen benötigen. Sie haben also ein Steuerwerk. RISC-Prozessoren sind dadurch charakterisiert, dass sie kein Steuerwerk haben und somit alle Befehle in einer Taktperiode ausführen.

Außerdem gibt es zwei unterschiedliche Architekturvarianten: die *von−Neumann*-Architektur und die *Harvard*-Architektur. Die von−Neumann-Architektur ist dadurch gekennzeichnet, dass nur eine Speicherschnittstelle vorhanden ist, über die Programmbefehle geladen und Daten geladen bzw. gespeichert werden. Bei der Harvard-Architektur gibt es mehrere Speicherschnittstellen, eine für Programmbefehle und eine oder mehrere für Daten. Durch diese Parallelisierung der Ressourcen kann der Engpass der Speicherbandbreite vom und zum Prozessor erweitert und entschärft werden.

Eine Harvard-Architektur wird bei *digitalen Signalprozessoren* (DSP) eingesetzt. Digitale Signalprozessoren sind Prozessoren, die eine an das Anwendungsgebiet angepasste Verarbeitungseinheit haben, die so genannte MAC-Einheit (Multiply and Accumulate). Mit ihr können parallel eine Multiplikation und eine Addition durchgeführt werden. Dieser Arbeitsablauf kommt besonders häufig bei digitalen Filtern vor, weswegen DSPs häufig für deren Berechnung eingesetzt werden.

Eine weitere Variante von programmierbaren Prozessoren stellt das Konzept von *Very-Long-Instruction-Word*-Prozessoren (VLIW) dar. Hier werden mehrere Befehle gleichzeitig in einem Befehlswort geladen, die an ihre jeweilige Verarbeitungseinheit weitergeleitet werden. Die Verarbeitungseinheiten können dabei unterschiedliche Befehlssätze haben und sind daher nicht gleichwertig. VLIW Prozessoren gehören zu den MIMD Architekturen. Die Auswahl der Befehle und Daten muss so gewählt werden, dass keine Datenabhängigkeiten verletzt werden. Diese Datenabhängigkeiten werden durch Umsortieren der Befehlsreihenfolge aufgelöst. Die Aufgabe, Abhängigkeiten aufzulösen, wird dabei vom Compiler erledigt. Allerdings lässt sich in der Regel keine 100%ige Auslastung der Verarbeitungseinheiten erreichen, da die Abhängigkeiten nicht immer aufgelöst werden können.

Ein programmierbarer Prozessor kann vom Prinzip her für beliebige Aufgaben eingesetzt werden, wie z.B. Bildanalyse oder Textverarbeitung. Im Gegensatz dazu kann mit einem dedizierten Prozessor (ASIC) nur eine einzige Aufgabe gelöst werden. Für eine andere Aufgabe wird ein anderer Prozessor benötigt. Der Vorteil von dedizierten Prozessoren ist die genaue Anpassung an die zu lösende Aufgabe.

Dies betrifft z.B. eine angepasste Anzahl an Registern, in denen Zwischenergebnisse gespeichert werden können. Diese Register brauchen auch keinen universellen Zugang zum Lesen und Schreiben der Daten, wie es bei den Registern eines Mikroprozessors nötig ist. Auf einem ASIC können die Register gezielt zwischen den Ressourcen für die Verarbeitung eingesetzt werden. Demgegenüber haben Instruktionsprofile [51][52][53] ergeben, dass bei Standardprozessoren meist ca. 40% aller Maschinenbefehle zum Laden oder

Speichern von Daten (Zwischenergebnissen) benötigt werden.

Auf einem ASIC können zudem auf die in den Registern abgelegten Daten gleichzeitig zugegriffen werden. Dadurch wird die parallele Verarbeitung der Daten ermöglicht. Alles in allem können auf einem ASIC die vollen Parallelisierungsmöglichkeiten, die bei der Lösung der Aufgabe möglich sind, ausgenutzt werden. Bei Standardprozessoren hingegen ist die Abarbeitungsreihenfolge immer sequentiell oder wie bei VLIW Prozessoren maximal mit Anzahl der Verarbeitungspfade parallel.

Des Weiteren entfallen bei ASICs Taktzyklen zum Laden der Befehle, was allerdings bei der Harvard-Architektur auch parallel zum Laden, Verarbeiten und Speicher von Daten gemacht werden kann. Bei ASICs ist das Programm in einem Steuerwerk abgelegt, das den zeitlichen Ablauf festlegt. Das Steuerwerk entspricht also der Mikroprogrammierung einer CPU. Ein weiterer Vorteil der statischen Programmierung in einem Steuerwerk eines ASICs ist, dass (bedingte) Verzweigungen keinen eigenen Befehlszyklus benötigen.

Durch alle diese Anpassungen kann ein ASIC das ihm gestellte Problem deutlich effizienter lösen. Dies betrifft sowohl die Anzahl der benötigten Taktzyklen als auch die vom Chip aufgenommene elektrische Leistung. CPUs habe demgegenüber aber in der Regel deutlich höhere Taktfrequenzen, da sich durch den vielfältigen Einsatz eine extrem hohe Optimierung lohnt.

Eine Möglichkeit der Kombination von programmierbaren und dedizierten Lösungen ist die Erweiterung des Befehlssatzes mit komplexen Befehlen, die in der Abarbeitung der Aufgabe besonders häufig wiederkehren. Ein einfaches Beispiel hierfür wäre die MAC-Einheit bei DSPs. Aber in der Regel sind die dedizierten Ergänzungen deutlich komplexer. In einigen Ansätzen lassen sich die Befehlserweiterungen zur Laufzeit konfigurieren. Hierbei wird der Standardprozessor um einen FPGA-Block (*Field-Programmable-Gate-Array*) erweitert. In dem FPGA-Block befinden sich dann eine Reihe von LBAs (*Logic-Block-Array*), deren genaue Funktion über ein Konfigurationsregister festgelegt wird. Über weitere Konfigurationsregister kann dann noch die Zusammenschaltung der LBAs konfigurieren werden. So kann in dem konfigurierbaren Block eine Funktion implementiert werden, die dann während des Programmablaufs aufgerufen wird und die Funktion beschleunigt ausführt. Dann kann zur Laufzeit die Funktion des Blockes umkonfiguriert werden und eine andere Funktion beschleunigt abgearbeitet werden.

FPGAs lassen sich auch alleine, ohne programmierbaren Prozessor einsetzen. Dabei werden FPGAs bevorzugt eingesetzt, wenn sich wegen zu niedriger Stückzahlen die Fertigung eines ASIC nicht lohnt, oder der Prototyp eines ASICs getestet werden soll. Dabei sind mit einem FPGA im Vergleich zum ASIC nur niedrige Taktfrequenzen möglich (ca. 1/3 bis 1/6) [54][55].

Eine andere Kombinationsmöglichkeit von konfigurierbaren Modulen ist

der Einsatz in einem ASIC. Hier kann dem ASIC in engen Grenzen noch eine Flexibilität hinzugefügt werden. Dies macht z.B. Sinn, wenn ein großer Teil der Aufgabe fest steht und nur ein relativ kleiner Teil flexibel sein muss.

Im Folgenden werden die oben genannten Konzepte noch mit Beispielen belegt. Der typische Vertreter für Standardprozessoren ist z.B. der Intel-Pentium−IV-Prozessor der in Standard-PCs eingesetzt wird. Hierbei handelt es sich um einen CISC-Prozessor, der eine Fließbandverarbeitung implementiert. So kann ein Befehlsdurchsatz von annähernd einem Befehl pro Takt erreicht werden. Der Prozessor hat eine Verlustleitung von 66 Watt (bei 1,8 GHz Taktfrequenz) [56][57] und eine Komplexität von ca. 55×10^6(mit Northwood-Kern) bzw. 125×10^6 (mit Prescott-Kern) Transistoren [58]. Neben den normalen arithmetischen und logischen Befehlen unterstützt die Verarbeitungseinheit auch die parallele Verarbeitung von Sub-Worten. Diese Architektur ist als Multimediaerweiterung (MMX) [59] bekannt. Dabei können die Sub-Worte als unabhängige Daten mit niedrigerer Bitbreite interpretiert (z.B. 64 Bit als 8x 8 Bit breite Datenwörter) und gleichzeitig im SIMD-Verfahren verarbeitet werden. Vergleichbare Architekturen werden auch bei den kompatiblen Konkurrenzprodukten verwendet (AMD [60]). Bei den UltraSparc-Prozessoren heißt dieses Konzept *Visual-Instruction-Set* (VIS) [61]. Bei den UltraSparc-Prozessoren handelt es sich um RISC-Prozessoren. Da hier jeder Verarbeitungsschritt des Programms als eigener Maschinenbefehl sichtbar ist, wurde diese Architektur in dieser Arbeit bevorzugt für die Erstellung von Instruktionsprofilen verwendet.

Im Vergleich zu den vorher beschriebenen Multimediaerweiterungen MMX oder VIS hat der TRIO-Prozessor [62] neben der *General-Instruction-Unit* (GPU) eine *Image-Processing-Unit* (IPU). Diese auf Bildverarbeitungsfunktionen ausgerichtete Verarbeitungseinheit unterstützt Befehle für z.B. Faltungen, Erosion, Dilatation. Die Bildverarbeitungsbefehle sind hier auf 3x3 Nachbarschaftssystem beschränkt. Ein weiteres Beispiel, bei dem ein Prozessor mit zusätzlichen Verarbeitungseinheiten und Befehlen erweitert wurde, ist die Bitstream-Engine [63]. Hier wurde ein MIPS-Core mit Funktionen für das Decodieren von MPEG−4-Video-Bitströmen erweitert.

Neben dedizierten Erweiterungen der Verarbeitungseinheit gibt es auch Architekturen, die für die Erweiterungen konfigurierbare Logik verwenden. Beispiele hierfür sind die Garp-Architektur [64], die einen MIPS−II-Prozessor verwendet oder die A7−Familie [65] von Triscend, die einen ARM7TDMI-Prozessor und 3200 konfigurierbare Logikzellen kombiniert.

Ein Beispiel für digitale Signalprozessoren ist der DSP96000. Er basierte auf einer Harvard-Architektur mit bis zu drei Datenbussen (je nach Ausführung). Die Verarbeitungseinheit verfügte über eine 96 Bit breite Verarbeitungseinheit für Fließkommazahlen (FPU). Ein Single-Chip-

Multiprozessorsystem ist der DSP TMS320C8X (von Texas Instruments) [66]. Auf dem Chip befand sich ein RISC-Prozessor und mehrere DSPs. Wegen der komplizierten Programmierung konnte sich dieser Prozessor allerdings nicht durchsetzen.

Der TSM320C6X (auch von Texas Instruments) [67] ist hingegen ein Vertreter der VLIW-Prozessoren. Je nach Ausführung verfügt er über 8 bzw. 14 Funktionseinheiten. Die Variante TSM320C64x stellte auch Sub-Wort-Parallelität, Bit-Level-Verarbeitung und Spezialbefehle für fehlerkorrigierende Codes zur Verfügung. Ein weiterer prominenter Vertreter der VLIW-Prozessoren ist der TriMedia (Philips) [68], der speziell für Multimediaanwendungen entworfen wurde.

Der HiPAR-Prozessor [69] wird über einen VLIW-Bitstrom programmiert. Intern arbeitet er als autonomes SIMD-Array mit 4 (HiPAR-4) bzw. 16 (HiPAR-16) Datenpfaden. Dabei kann jeder Datenpfad auf einen privaten Cache, auf den externen globalen Speicher und auf einen gemeinsamen Matrixspeicher zugreifen. Auf den Matrixspeicher wird mit virtuellen $2D-$Adressen zugegriffen.

Architekturen, die nur für Bildverarbeitungsaufgaben einsetzbar sind, sind z.B. der PIMM1 oder der PIMM10. Der PIMM1 [41] ist ein Spezialprozessor mit einer 3x3 Matrix für Bildpunktdaten. Zur Optimierung des Ladens der Eingangsdaten werden Verzögerungsketten verwendet. Somit müssen Bildpunkte nicht mehrfach geladen werden. Der PIMM1 kann für reguläre Abarbeitungsfolgen verwendet werden. Unterstützte Verarbeitungsfunktionen sind z.B. die Erstellung von Distanzfeldern, Erosion oder Dilatation. Die Verarbeitungsfunktion wird über ein 34 Bit Konfigurationsregister ausgewählt. Der PIMM1 kann Graubilder mit 8 Bit pro Bildpunkt verarbeiten. Zur Unterstützung von 16 Bit breiten Bildkanälen können zwei PIMM1-Prozessoren parallel geschaltet werden. Der PIMM10 [42] ist eine Erweiterung des PIMM1. Hier können z.B. zwei Bildpunkte parallel verarbeitet werden.

Die in [70] und [71] vorgeschlagenen Architekturen beschleunigen die Berechnung von Erosion bzw. Dilatation, und [72] bzw. [73] das Wasserscheide-Verfahren. Dies sind Beispiele, mit denen einzelne, rechenintensive Bildverarbeitungsfunktionen implementiert werden. Daneben existieren noch Spezialarchitekturen für die Abarbeitung kompletter Algorithmen. So wird mit [74] eine Fahrbahnranderkennung und mit [75] eine klassifikationsbasierte Segmentierung für automatische Fertigungslinien durchgeführt.

1.4.7 Bedarf an Weiterentwicklungen

In diesem Kapitel werden Lücken im Stand der Technik identifiziert, aus denen sich im folgenden Kapitel die konkrete Aufgabenstellung ableiten lässt.

Dabei strukturiert sich dieses Kapitel nach den im Kapitel 1.1 beschriebenen Zielsetzungen dieser Arbeit.

Bei einer Betrachtung von Spezifikationsaspekten kann man feststellen, dass selbst nach dem Lesen der veröffentlichten Algorithmen aus Kapitel 1.4.3 diese nicht reproduzierbar nachprogrammiert werden können. Der Grund ist, dass die Spezifikationen der Bildverarbeitungsfunktionen in der Regel unvollständig sind. Dadurch ist ein Aufbauen auf den in Veröffentlichungen beschriebenen Algorithmen in der Regel sehr aufwendig oder nicht möglich. Ein wesentlicher Grund für diesen Zustand ist das Fehlen eines exakten Formats für die Beschreibung und Spezifikation der Bildverarbeitungsfunktionen in Textform. Deshalb wird z.B. bei der Standardisierung häufig eine Referenz-Software [76][77] erstellt, die u.a. als exakte Beschreibung der Funktionen dient. Dieser Ansatz wird auch in einigen Forschungsprojekten verwendet [78][34]. Leider ist in diesen Referenz-Systemen die relevante Information oft im Quellcode der Software verteilt und schwer lokalisierbar.

Ein weiterer Punkt, der bei der Spezifikation schwer ins Gewicht fällt, ist die Tatsache, dass es keine geeignete systematische Betrachtung der Bildverarbeitungsfunktionen gibt. Hier wird meist in klassische, statistische oder morphologische Methoden, nach dem mathematischen Ursprung unterteilt. Diese Kategorisierung ist aber wegen der fehlenden Trennung in allgemeinen und spezifischen Teil und wegen des fehlenden Bezugs zur Implementierung für eine Spezifikation ungeeignet. Während für klassische Filter das Verständnis noch relativ einfach ist, sind Bildverarbeitungsfunktionen, die auf Ausbreitungsprozessen basieren, deutlich komplexer in ihrer Funktion. Eine systematische Betrachtung von Ausbreitungsprozessen, wie sie in dieser Arbeit in den Kapiteln 2.1.4, 2.1.5, 2.2 und 2.3 durchgeführt wird, ist in der Literatur nicht zu finden. Eine systematische Betrachtung zeigt aber, dass für die Ausbreitungsprozesse nicht alle Prinzipien beschrieben sind, so dass bisher nur suboptimale Prinzipien angewendet wurden.

Zur Umsetzung der Bildverarbeitungsfunktionen in einer Simulations- oder Referenz-Software wurden einige Beispiele gezeigt. In diesem Abschnitt werden dabei die damit verbundenen allgemeinen Implementierungsmethoden, im Sinne der Erstellung einer Simulationsumgebung, betrachtet. Für eine Beurteilung der Methoden können die Erweiterbarkeit, sowie die Abdeckung (Grad der Vollständigkeit) betrachtet werden. Beide Faktoren zusammen bestimmen dann den Grad der Wiederverwendbarkeit.

Für die Beurteilung der Erweiterbarkeit der Bibliothek spielt der Aufwand, um neue Funktionen hinzuzufügen, eine wichtige Rolle. Hier unterscheidet sich die ImageLib von den anderen Konzepten (Envision, Aphelion). Bei der ImageLib wird ein wesentlicher Teil der Bildverarbeitungsfunktion wieder verwendet und nur die Filterfunktion für einen Bildpunkt neu ge-

schrieben. Bei Envision oder Aphelion hingegen muss die gesamte Bildverarbeitungsfunktion neu implementiert werden, also neben der Verarbeitungsfunktion für einen Bildpunkt auch die Schleife zum Laden und Speichern der Bildpunkte. Das Konzept der Wiederverwendung der Adressierungsschleifen hat die ImageLib mit der AddressLib, dem Software-Modell dieser Arbeit, gemeinsam. Außerdem ist für die Erweiterbarkeit der Bibliothek die Komplexität der Schnittstelle zu beachten. Je einfacher und regulärer die Schnittstelle ist, desto leichter kann die Bibliothek erweitert werden.

Neben der Erweiterbarkeit, spielt auch der Grad der Vollständigkeit eine große Rolle. So implementiert die ImageLib nur Filterfunktionen. Ausbreitungsprozesse können mit diesem Funktionstyp nur durch rekursive Filter implementiert werden. Funktionen, wie die Differenzbildberechnungen, lassen sich hingegen gar nicht auf Filter abbilden. Eine detaillierte Beschreibung zu diesen Funktionstypen befindet sich in Kapitel 2.1.2 und 2.1.3. Die Funktionsbibliothek von Envision enthält nur einfache Funktionen wie Farbraumkonvertierungen, oder lineare Filter. Zusätzlich sind spezielle Funktionen für die Bildkompression vorhanden. Funktionen für die Bildanalyse, insbesondere Funktionen, die auf Ausbreitungsprozessen beruhen, sind nicht vorhanden. Die Aphelion-Bibliothek hat einen recht umfangreichen Funktionssatz. Werden aber spezielle Filter für neue Algorithmen benötigt, stößt man auf die Grenzen der Bibliothek. Zwar hat diese Bibliothek einen sehr großen Umfang, aber dennoch ist sie begrenzt. In diesem Fall muss dann die Erweiterbarkeit mit neuen Funktionen deutlich einfacher sein, also Teile der Bibliothek zur Erstellung neuer Funktionen wieder verwendbar sein. Dafür müssen aber die Schnittstellen von wieder verwendetem zum ersetzten Code deutlich und einfach sein.

Ein weiterer Aspekt beim Entwurf einer Simulations-Software ist das Umsetzen von Vereinfachungen. Dazu gehört die Verwendbarkeit von unterschiedlichen Datenmodellen für die Bildpunktwerte und die Anwendung von unterschiedlichen Bildkanälen. Beide Konzepte werden von keiner der referenzierten Bibliotheken unterstützt. Dieser Aspekt betrifft nicht nur die funktionale Verifikation, sondern auch die Optimierung und das Ausbalancieren zwischen Qualität und Rechenleistung des Algorithmus. Zur Beurteilung der benötigten Rechenleistung eines Algorithmus werden zwei Komponenten benötigt:

- das Erstellen von Profilen
- die Interpretation der Profile

Die Profile werden dabei mit der Simulations-Software erstellt. Dabei gibt es zwei Möglichkeiten. Bei der Verwendung eines Lauftzeitprofils, wie z.B. bei

Verwendung des gprof [79], oder eines Instruktionsprofils, bei Verwendung des iprof [52], muss der Optimierungsgrad bekannt sein. Dieser ist aber nur schwer messbar. Alternativ kann zusätzlicher Code zum Zählen der Durchläufe von wichtigen Programmabschnitten eingebaut werden. Werden die Bildverarbeitungsfunktionen mit vielen unabhängigen Code-Segmenten implementiert, ist die Vertrauenswürdigkeit, dass an allen Stellen der Code korrekt eingebaut wurde, sehr gering. Dies trifft insbesondere zu, wenn der Code von unterschiedlichen Programmierern und Institutionen hinzugefügt wird. So wäre von den vorgestellten Bibliothekskonzepten nur die ImageLib geeignet, um Profile zu erstellen.

Nachdem ein Profil erstellt worden ist, muss es nun interpretiert werden. Idealer Weise ist dabei die Interpretation von der Architektur weitgehend unabhängig. Trotzdem wird eine Architektur benötigt, um die Profile zu bewerten. Diese Architektur kann dabei abstrakt bleiben, da sie nur für die Definition der Komplexitätsmaße benötigt wird. Das Komplexitätsmodell kann sich auch auf eine sequentielle Architektur beziehen, bei der jeder Arbeitsschritt in der Verarbeitugssequenz sichtbar ist.

Ein RISC-Prozessor würde diese Randbedingungen erfüllen. Realistischer für die Abschätzung der Rechenzeit an einer optimierten Architektur wäre aber ein Prozessor, der an das Anwendungsgebiet angepasst ist. Dabei steht das Anwendungsgebiet im Vordergrund und nicht die Optimierung für einen speziellen Algorithmus. Als geeignete Startpunkte werden die im Kapitel 1.4.4 beschriebenen Architekturen verwendet. Diese Startpunkte decken jeweils einen Teilbereich der benötigten Funktionen optimal ab. Eine für das gesamte Anwendungsgebiet sinnvolle Architektur müsste die unterschiedlichen Ansätze kombinieren. Programmierbare Lösungen werden für optimierte Implementierungen nicht betrachtet, da sie nicht oder nur in sehr geringem Maß die Randbedingungen des Anwendungsgebiets Bildanalyse berücksichtigen.

Eine der wichtigsten Randbedingungen ist, dass die selbe Operation auf alle Bildpunkte, die verarbeitet werden sollen, angewendet wird. Bei einem CIF-Bild (352x288 Bildpunkte) ist dies ca. 100.000mal der Fall. Dabei macht es wenig Sinn die Befehlssequenz zum Ausführen der Operation 100.000mal zu laden. Massiv parallele Systeme sind auf der anderen Seite auch ungeeignet, da rekursive Prozesse den hohen Grad der Parallelität nicht ausnutzen können. Somit sind diese Architekturen nur für einen Teilbereich der Funktionen sinnvoll. Insbesondere die Funktionen, die charakteristisch für die Bildanalyse sind, sind wegen ihrer Rekursivität ausgeschlossen. Einige Umsetzungen der Parallelisierungskonzepte, die in den im Kapitel 1.4.6 vorgestellten Architekturen verwendet werden, werden trotzdem bei dem Hardware-Modell berücksichtigt.

Ein grundsätzlich anderer Weg wird bei Envision eingeschlagen. Hier können unterschiedliche Implementierungsmodelle für die Simulation und Hardware-Implementierung vorhanden sein. Die Rechenzeit kann dann mit einer Simulation des synthetisierbaren Verilog- oder VHDL-Modells ermittelt werden. Erstellen und Interpretieren des Profils finden somit quasi in einem Schritt statt. Wegen der extrem hohen Simulationszeiten sind aber nur eingeschränkt Simulationen zur Ermittlung der Rechenzeit durchführbar. Außerdem gilt der ermittelte Wert dann ausschließlich für die implementierte Architektur und ist nicht mehr nur vom Algorithmus abhängig.

Obwohl die hier durchgeführte Arbeit für das Anwendungsgebiet Bildanalyse gültig sein soll, wird ein ausgewählter Algorithmus für die Verifikation der Arbeit als Beispiel benötigt. Dabei soll der Algorithmus eine möglichst große Abdeckung der typischen Bildverarbeitungsfunktionen haben und dadurch universell im Sinne der verwendeten Methoden sein. Dies trifft aber auf keinen der vorgestellten Algorithmen zu, so dass ein neuer Algorithmus verwendet werden muss. Aus der großen Abdeckung der verwendeten Methoden soll im Rahmen dieser Arbeit keine universelle Anwendbarkeit des Algorithmus abgeleitet werden.

1.5 Aufgabenstellung

Aufbauend auf einem Beispielalgorithmus zur Objektsegmentierung, der eine Farbanalyse durchführt, soll im Rahmen dieser Arbeit ein Modell entworfen werden, auf das Bildverarbeitungsfunktionen abgebildet werden können. Die Bildverarbeitungsfunktionen, die auf das Modell abgebildet werden, sollen die Randbedingungen erfüllen, dass die Bildpunktdaten als zweidimensionales Bild organisiert sind und jeder Bildpunkt nur ein einziges Mal verarbeitet werden muss. Durch Abbildung der Bildverarbeitungsfunktionen auf das Modell sollen diese einfach und eindeutig beschrieben werden können. Ein besonderer Schwerpunkt bei der Entwicklung des Modells soll auf Ausbreitungsprozesse gelegt werden, da diese charakteristisch für die Bildanalyse sind. Es sollen systematisch Prinzipien dargestellt werden, mit denen Ausbreitungsprozesse realisiert werden können. Aus dieser systematischen Darstellung soll ein Ausbreitungsprinzip identifiziert werden, das allgemeingültig ist und Ausbreitungsprozesse effizienter darstellen kann als die bisher bekannten Prinzipien.

Das abstrakte Bildverarbeitungsmodell soll im nächsten Schritt in ein Software-Modell übertragen werden. Durch die Implementierung der Bildverarbeitungsfunktionen mit dem Software-Modell soll eine Methode zur Verfügung gestellt werden, mit der sich die Bildverarbeitungsfunktionen auf effizi-

ente Weise implementieren lassen.

Im dritten Schritt soll ein Hardware-Modell abgeleitet werden, das zum Software-Modell analog ist. Das Hardware-Modell soll Optimierungen, die für das Anwendungsgebiet charakteristisch sind, einschließen. Optimierungen, die bei speziellen Bildverarbeitungsfunktionen möglich sind, werden dabei nicht berücksichtigt, da diese nicht für das gesamte Anwendungsgebiet allgemeingültig sind. So kann das Hardware-Modell zur Abschätzung der Rechenkomplexität eines Algorithmus verwendet werden. Durch die Analogie zum Software-Modell sollen Profile mit einer Software-Simulation erstellt werden können, die mit dem Hardware-Modell bzgl. der Rechenkomplexität beurteilt werden können. Die Ermittlung der Rechenzeiten auf konkreten Architekturen in Abhängigkeit der Rechenkomplexität ist nicht Gegenstand der Arbeit und soll nur kurz diskutiert werden.

Die Arbeitsschritte sollen dann anhand des Beispielalgorithmus verifiziert werden. Dabei soll die Gültigkeit des Modells durch die Implementierung des Algorithmus mit dem Software-Modell nachgewiesen werden. Dadurch wird auch die Funktionstüchtigkeit des neu entwickelten Prinzips für die Darstellung von Ausbreitungsprozessen, das auf einer LIFO-Architektur beruht, verifiziert. Das abstrakte Modell soll anschließend benutzt werden, um eine Beispielfunktion zu spezifizieren.

Anhand der zuvor spezifizierten Bildverarbeitungsfunktion soll diese mit dem Software-Modell implementiert werden. Dabei soll die Effizienz beim Implementieren von Bildverarbeitungsfunktionen gezeigt werden.

Durch eine Software-Simulation soll zunächst ein Instruktionsprofil für den Algorithmus mit einer Testsequenz erstellt werden. Das Instruktionsprofil soll dann analysiert werden. Unter Verwendung von Instruktionsprofilen sollen auch die Laufzeiten der wesentlichen Typen von Bildverarbeitungsfunktionen miteinander vergleichen werden. Mit der Software-Simulation soll außerdem bei den Ausbreitungsprozessen der maximale LIFO-Füllstand gemessen werden. Für die Ermittlung eines Komplexitätsmerkmals, das zur Abschätzung der Laufzeit des Algorithmus auf einer konkreten Architektur verwendet werden kann, soll die Speicherbandbreite zum Laden und Speichern der Bildpunktdaten aus bzw. in den Bildspeicher gemessen werden.

Als wissenschaftliche Neuerungen ergibt sich dadurch eine modellhafte Beschreibung von Bildverarbeitungsfunktionen. Insbesondere wird hier der Schwerpunkt auf die Analyse von Funktionen mit Ausbreitungsprozessen gelegt. Die systematische Analyse dieser Funktionen wird unterschiedliche Arbeitsmodi herleiten, aus der sich dann die Parametrisierung ableiten lässt. Neben der Modellbeschreibung wird in dieser Arbeit auch, ausgehend von den in der Literatur dargestellten Prinzipien für Ausbreitungsprozesse, ein neues und optimiertes Prinzip, das LIFO-Prinzip, entwickelt. Analogien und

Unterschiede zu den bisher bekannten Prinzipien werden diskutiert.

Über die Analyse der Bildverarbeitungsfunktionen hinaus wird das Modell noch um ein Software- und Hardware-Modell erweitert, so dass konsistent Bildverarbeitungsfunktionen spezifiziert, mit Software simuliert und die Komplexität des Algorithmus implementierungsunabhängig analysiert werden kann. Dabei stellt die Durchgängigkeit eine wichtige Eigenschaft dar.

1.6 Ein ausgewählter Segmentierungsalgorithmus

In diesem Kapitel wird nun ein Segmentierungsalgorithmus beschrieben, der aus den in Kapitel 1.4.3 beschriebenen Algorithmen unterschiedliche Bildverarbeitungsfunktionen einsetzt. Damit soll der Algorithmus die nötige Breite an verwendeten Bildverarbeitungsfunktionen haben, um das Bildverarbeitungsmodell abzuleiten und dessen Sinnvollheit nachzuweisen. Er dient als Beispiel, an dem die Randbedingungen festgelegt werden, unter denen das Modell für Bildverarbeitungsfunktionen gültig sein soll.

Dieser Farbsegmentierungsalgorithmus benutzt sowohl lokale Informationen der Bildpunkte als auch globale Informationen der Segmente. Dies führt zu einer sehr leistungsfähigen Kombination, da die Vorteile beider Varianten kombiniert werden.

Abbildung 1.13 zeigt den Aufbau des hier benutzten Algorithmus zur Farbanalyse. Dieser Algorithmus basiert auf dem Wasserscheide-Algorithmus. Um eine extreme Übersegmentierung zu vermeiden, werden vor dem Ausbreitungprozess des Wasserscheide-Algorithmus Start- oder Markerzonen definiert. Die Startzonen, der in die Szene neu hinzukommenden Objekte, werden mit drei Schritten gefunden. Zuerst werden alle zusammenhängenden Regionen, die eine vorgegebene Mindestgröße überschreiten, mit einer neuen Segmentnummer gekennzeichnet. Zusammenhängend bedeutet hier, dass die Bildpunkte örtlich benachbart sind und ein Homogenitätskriterium erfüllen. Homogen bedeutet hier, dass die Differenz zwischen den Bildpunkten unter einer vorgeschriebenen Grenze bleibt. Das Finden von zusammenhängenden, homogenen Flächen basiert nur auf den lokalen Bildpunktwerten. Nach dem Finden von neuen Segmenten werden die gefundenen Segmente noch anhand ihrer Histogramme auf ihre Homogenität hin überprüft. Dieser Schritt benutzt globale Eigenschaften des Segments. Im Histogramm kann man sehen, ob es eine oder mehrere Charakteristika hinsichtlich der im Histogramm eingetragen Eigenschaft im Segment gibt. Wird mehr als eine Charakteristik gefunden, dies bedeutet es gibt neben dem Hauptmaximum

Abbildung 1.13: Algorithmus zur Farbanalyse. Nach der Bewegungskompensation und dem Löschen von nicht kompensierbaren Bildpunkten werden in den Iterationsschleifen neue Segmente gefunden. Anschließend werden noch die nicht Segmenten zugewiesenen Bildpunkte mit dem Wasserscheide-Algorithmus den existierenden Segmenten zugeordnet. Die abschließende Relaxation kann die Segmentkonturen noch fein korrigieren.

im Histogramm noch weitere lokale Maxima, dann werden alle Bildpunkte gelöscht, die nicht der Hauptcharakteristik entsprechen. In Histogrammen können folgende Eigenschaften eingetragenen sein:

- der Grauwert (Y)
- die Aktivität der Grauwerte ($max_{n \in Nhood}(Y_n) - min_{n \in Nhood}(Y_n) =$ (morphologischer Gradient)
- die beiden Farbwerte (U und V)
- die Aktivität der Farbwerte (vgl. Aktivität des Grauwerts)
- die Differenz bzw. Summe der Farbwerte und
- deren Aktivität

Dies sind die derzeit im Algorithmus implementierten Eigenschaften, die getestet werden können. Es ist aber einfach möglich, weitere Segmenteigenschaften zu extrahieren und mit einem Histogramm auf Homogenität zu prüfen. Außerdem wird hier angemerkt, dass es unter Umständen nicht trivial ist, das Grauwerthistogramm zu beurteilen, da sich hier unterschiedliche Beleuchtungen (z.B. Schatten) direkt bemerkbar machen. Auf der anderen Seite sind die Farbwerte der YUV-Darstellung, also die U- und V-Werte, weitgehend unabhängig von der Ausleuchtung. Die Differenz bzw. Summe der Farbwerte kann die Abweichung im Segment verstärken und somit die Maxima im Histogramm besser voneinander trennen. Die Histogrammanalyse wird nur benötigt und angewendet, wenn sich das Segment seit der letzten Histogrammanalyse verändert hat.

Nachdem durch die Histogrammanalyse Bildpunkte von einem Segment entfernt worden sein können, können aus den geschlossenen Flächen einzelne Fragmente entstanden sein. Deshalb wird noch ein Fragmentierungstest durchgeführt. Werden Fragmente gefunden, wird ihnen eine neue Segmentnummer zugewiesen. Sind die Fragmente kleiner als ein vorgegebener Schwellwert, werden die Fragmente wieder gelöscht.

Diese drei Schritte zum Finden neuer Segmente werden mit unterschiedlichen Schwellwerten für die Segmentgröße und die erlaubte Bildpunktdifferenz mehrmals durchgeführt. Durch diese Iterationen kann die Bildanalyse mit unterschiedlichem Bildmaterial und unterschiedlichen Objekttypen arbeiten. Die Parameterabfolge legt dabei die Rangordnung der Objekttypen fest. Die Iterationen werden beendet, wenn eine maximale Anzahl an Parametern benutzt wurden, oder wenn eine vorgegebene Anzahl an Segmenten erreicht (überschritten) wurde. Zusätzlich können die selben Parameter mehrfach benutzt werden, bis sich die Segmentmaske nicht mehr ändert oder eine Maximalzahl an Iterationen durchgeführt wurde. Wenn bereits eine bewegungskompensierte Segmentmaske vom Vorgängerbild zur Initialisierung existiert, werden mit diesen Schritten nur neue Objekte gefunden. In diesem Fall kann der Parametersatz in Abhängigkeit von dem nicht wieder gefundenen Bildbereich reduziert werden. Zur Vereinfachung wurde mit dem letzten Parametersatz des vorhergehenden Bildes weiter gesucht. Wenn keine Segmentmaske zur Initialisierung vorliegt, wie beim ersten Bild der Sequenz oder nach einem Szenenschnitt, wird wieder mit dem vollständigen Parametersatz gearbeitet.

Nachdem die Startzonen der Segmente festgelegt wurden, werden die bisher noch nicht zugeordneten Bildpunkte durch Ausbreitung der Startzonen mit dem Wasserscheide-Algorithmus den Segmenten zugeordnet.

Nach dem Wasserscheide-Algorithmus können noch Segmente, die an Segmentkanten entstehen, bzw. dünne Verästelungen an den Segmenten entfernt

werden. Dies geschieht mit einer Erosion der Segmentmasken um ein Bildelement und einer anschließenden Dilatation, bis wieder jeder Bildpunkt einem Segment zugeordnet ist. Durch diesen Schritt werden auch kleine Segmente gelöscht, die eine eigene deutliche Charakteristik aufweisen und von daher für die Segmentierung wichtig sind. Diese Segmente unterliegen durch das häufige Löschen und Wiederfinden einer erhöhten zeitlichen Instabilität.

Wenn alle Bildpunkte eine Segmentnummer haben, können die Grenzen der Segmente noch mit wenigen Iterationen einer abgewandelten Relaxation leicht korrigiert werden. Bei der Software-Implementierung der Relaxation in dieser Arbeit gehen nicht nur die Bildpunktwerte in die Summe der Kosten ein, sondern auch die Differenz zu dem Mittelwert des Segments. Dadurch können wieder lokale und globale Eigenschaften benutzt werden. Es handelt sich also dabei um eine Mischung aus Relaxation und *Active-Contours*-Algorithmus.

In allen Abschnitten des Algorithmus zur Farbanalyse werden die Informationen erstellt und ausgewertet, wann ein Segment entstanden ist. Dabei wird unterschieden in:

- mit dieser Iteration entstanden
- in diesem Bild entstanden
- bereits im vorherigen Bild vorhanden gewesen oder aus einem Segment im vorherigen Bild (durch Fragmentierung) entstanden (im Fragmentierungsfall wird noch zusätzlich die Nummer des Segments im Vorgängerbild gespeichert)

Dies bedeutet zwar einigen Aufwand in der Implementierung der Farbanalyse, verbessert aber die zeitliche Kohärenz der Segmentmaske erheblich. Über die Kenntnis der Nummer eines Segments im Vorgängerbild lässt sich auch die Zusammenfassungsinformation von möglichen, folgenden Analysemethoden zur Initialisierung der Zusammenfassung wieder verwenden.

In [11] werden Möglichkeiten beschrieben, die Rechenkomplexität des Algorithmus zur Farbanalyse zu reduzieren und somit die Ausführungsgeschwindigkeit zu erhöhen. Mit der höheren Ausführungsgeschwindigkeit ist allerdings auch eine Verschlechterung der Genauigkeit der Segmentmaske verbunden. Somit bilden die Anforderungen an die Rechenleistung und Qualität einen Gegensatz (*Trade-Off*).

Eine Möglichkeit, die benötigte Rechenleistung zu reduzieren, ist das Überspringen der Suche nach neuen Segmenten für eine bestimmte Anzahl von Bildern. Dies bedeutet auch, dass nach der Bewegungskompensation der Segmentmaske des vorherigen Bildes keine Histogrammanalysen der Segmente gemacht werden. Zur Verbesserung der Qualität, kann unter bestimmten

Bedingungen die Suche nach neuen Segmenten erzwungen werden. Dies ist sinnvoll, wenn durch die Bewegungsschätzung erkannt wurde, dass mindestens ein neues Objekt im Bild vorhanden ist. Dieses neue Objekt lässt sich nicht im vorherigen Bild finden. Deshalb werden während der Bewegungsschätzung die Blöcke in der Region des neuen Objekts einen hohen Wert der minimalen Summe der Absolutdifferenzen (SAD) aufweisen. Liegt der SAD-Wert über einem vorgegebenen Schwellwert, gilt der Block als nicht im Referenzbild gefunden und die Segmentnummern der Bildpunkte werden nicht aus dem vorherigen Bild (Referenzbild) prädiziert.

Gibt es mehr als eine vorgegebene Anzahl an nicht nachverfolgten Bildpunkten, wird davon ausgegangen, dass ein neues Objekt in die Szene eingetreten ist. Durch diese Maßnahme zur Rechenleistungsreduktion werden z.B. bei der Foreman-Sequenz (in QCIF-Auflösung) für Bilder ohne Suche nach neuen Objekten ca. 336×10^6 und für Bilder mit Suche nach neuen Objekten ca. 1373×10^6 RISC-Instruktionen für die Analyse benötigt. Damit kann eine deutliche Beschleunigung erreicht werden (ca. Faktor 4), ohne dass die Qualität der Segmentierung darunter erkennbar leidet.

Weitere Möglichkeiten, die benötigte Rechenleistung zu verringern, sind:

- die Reduktion der Anzahl der zu prüfenden Kriterien
- die Reduktion der Anzahl der Iterationen bis zum Abbruch
- die Reduktion der Anzahl der Segmente bis zum Abbruch
- die Auflösung der Iterationsschleifen durch Benutzung einfacherer Methoden zur Quantisierung der Bildpunktwerte
- die Reduktion der Anzahl der Iterationen für die Relaxation

Die ersten vier Punkte beziehen sich dabei auf das Finden der homogenen Kernzonen von den Segmenten.

Der Algorithmus zur Farbsegmentierung lässt sich in zwei Ebenen unterteilen, den Bildverarbeitungsfunktionen und dem *High-Level*-Algorithmus, der diese kombiniert. Bildverarbeitungsfunktionen lassen sich dabei wie folgt definieren:

Definition 1.6.1 *Eine Bildverarbeitungsfunktion verwendet für die Berechnung von Ergebnissen für alle Bildpunkte eines Bildes oder Bildbereichs eine beliebige aber feste Operation.*

Abschließend werden noch die Bildverarbeitungsfunktionen, die in der Farbanalyse benutzt werden, aufgelistet. Es wird deshalb noch einmal auf Abbildung 1.13 auf Seite 43 verwiesen, in der die Struktur des Algorithmus

zur Farbanalyse mit den beiden Iterationsschleifen für die Parametervariation und für Iteration mit einem Parameter dargestellt ist. In den einzelnen Abschnitten werden folgende Bildverarbeitungsfunktionen benutzt:

- Kopieren einer Bildregion von einer anderen Stelle eines anderen Bildes zur Bewegungskompensation
- Berechnung der Bildpunkteigenschaft für die Histogrammanalyse (je Eigenschaft)
- Das Finden neuer Startzonen verwendet:
  - Bestimmung der Größe von homogenen zusammenhängenden Flächen
  - Nummerierung der homogenen zusammenhängen Flächen
- Die Histogrammanalyse je Eigenschaft benutzt folgende Bildverarbeitungsfunktionen:
  - Erstellen der Histogramme pro Segment aus den vorberechneten Eigenschaften
  - Histogrammauswertung, die Suche nach Nebencharakteristika in den Histogrammen
  - Löschen der Segmentnummer bei Bildpunkten die nicht der Hauptcharakteristik entsprechen

  Die Histogrammauswertung ist nicht mit einer Bildverarbeitungsfunktion realisiert und wird im Folgenden als Teil des High-Level-Algorithmus betrachtet.
- Der Fragmentierungstest besteht aus:
  - Suchen nach Fragmenten
  - Neunummerierung von Fragmenten
  - Löschen von zu kleinen Fragmenten
- Wasserscheide-Algorithmus
- Das Entfernen von linienförmigen Segmenten und Ästen von Segmenten (Kantensegmenten) benutzt:
  - Erosion der Segmente
  - Dilatation der Segmente

- Die Relaxation benötigt:
  - Extraktion der Mittelwerte von Helligkeit und Farbe aller Segmente
  - mehrere Iterationen der Relaxation

Dieser Algorithmus benutzt also auf der einen Seite Bildverarbeitungsfunktionen für lokale und (Segment-)globale Eigenschaften und auf der anderen Seite klassische, morphologische und statistische Bildverarbeitungsfunktionen.

Neben den Bildpunktoperationen, die in dem hier beschriebenen Algorithmus benutzt werden, werden in der Bewegungsschätzung noch

- die Berechnung von Absolutdifferenzbildern
- mit Aufsummierung der Absolutdifferenzen (SAD)

benötigt. Außerdem werden noch

- die Differenzbildung (inklusive SAD-Berechnung) von Bildern mit Koordinatentransformationen (zur globalen Bewegungsschätzung)
- die Skalierung von Bildern (zur Konvertierung von 4:2:0 nach 4:4:4 Farbauflösung)
- die Expansion von geodätischen Skeletten (zur Formanalyse)

als charakteristisch und wichtig empfunden.

Kapitel 2

Modelle und Methoden

2.1 Abstraktes Modell für Bildverarbeitungsfunktionen

2.1.1 Grundprinzip des Modells

Nachdem nun ein Überblick über den Stand der Technik gegeben und ein Beispielalgorithmus festgelegt wurde, wird in diesem Kapitel die eigene Arbeit vorgestellt. Dies ist die Ableitung des abstrakten Modells für die Bildverarbeitungsfunktionen, das Software- und Hardware-Modell sowie die Methoden zur Spezifikation, Implementierung und Analyse.

Wie in Definition 1.6.1 beschrieben, ist die Basis des Modells, dass für jeden Bildpunkt, der für die Verarbeitung adressiert wird, ein Ergebnis berechnet wird. Dabei werden die Eingangsbildpunkte geladen, verarbeitet und das Ergebnis gespeichert. In der Regel wird dieses Ergebnis dann in einem neuen Bildspeicher abgelegt, so dass ein Ergebnisbild, das dann z.B. weiter verarbeitet werden kann, entsteht. In Sonderfällen werden die Ergebnisse, die pro Bildpunkt berechnet werden, nicht in Form eines Ergebnisbildes benötigt, sondern zu globalen Ergebnissen zusammengefasst. Global bedeutet, wie in Kapitel 1.6 an Beispielen verdeutlicht wurde, dass diese Informationen sich auf das gesamte Bild bzw. auf ein Segment (Teilbereich des Bildes) beziehen. Beispiele hierfür sind:

- die Berechnung des umgebenden Rechtecks eines Segmentes, bei der alle Koordinaten der Bildpunkte des Segments mit einem Minimum- bzw. Maximum-Operator zusammengefasst werden
- die Aufsummierung aller Bildpunkte eines Segments zur Berechnung der Fläche eines Segments

In beiden Fällen werden nicht nur einzelne Bildpunkte verarbeitet, sondern eine Bildpunktmenge. Neben der Abarbeitungsfolge, in der Ergebnisse für die einzelnen Bildpunkte berechnet werden, werden für die Berechnung der Ergebnisse noch unterschiedliche Konfigurationen von Eingangsdaten benötigt. Die Abarbeitungsfolge und die Konfiguration der Eingangsdaten stellen den Adressierungsteil der Bildverarbeitungsfunktionen dar. Die Adressierungsmethoden lassen sich klassifizieren, weshalb die Bildverarbeitungsfunktionen in die beiden Teile Adressierung und Verarbeitung der einzelnen Bildpunkte unterteilt werden. Das Modell umfasst nicht die einzelnen Verarbeitungsmethoden, die pro Bildpunkt ausgeführt werden. Auf der anderen Seite definieren diese Operationen, welche Daten benötigt werden, um das Ergebnis für einen Bildpunkt zu berechnen. Deshalb legt die Verarbeitungsmethode fest, mit welcher Adressierungsmethode sie verwendet werden kann.

Zur weiteren Beschreibung des Modells werden noch die verwendeten Adressierungsmethoden erklärt. Die folgende Klassifikation der Bildverarbeitungsfunktion nach den Adressierungsarten ist ein wichtiges Ergebnis dieser Arbeit. Dabei wird in die drei Adressierungsklassen unterschieden:

- Inter-Adressierung (für die Verarbeitung von zwei Eingangsbildern)
- Intra-Adressierung (für die Verarbeitung eines Eingangsbildes)
- rekursive Adressierung (für die Darstellung von Ausbreitungsprozessen)

Die rekursive Adressierung wird dabei noch in die Varianten mit Stapelspeicher(n) und FIFO-Speicher(n) unterschieden. Jede Klasse von Adressierungsmethoden kennt dabei eine Vielzahl von Parametern, die die genaue Adressierungsmethode festlegen. Eine Übersicht über die Varianten wird in Kapitel 2.1.5 gegeben.

In jedem Fall geht das Modell davon aus, dass jeder Bildpunkt pro Aufruf der Bildverarbeitungsfunktion genau einmal verarbeitet wird. Diese Annahme gilt auch in der Regel. Eine bekannte Ausnahme von diesem Prinzip stellt das *Texture-Padding* des MPEG−4 Standards dar. Hier gibt es die Möglichkeit, dass in einem teiltransparenten Block die nicht transparenten Werte der Randbildpunkte in den transparenten Bereich kopiert werden, dann aber im weiteren Verlauf der Zeile noch weitere nicht-transparente Bildpunkte folgen. In diesem Fall wird zum ersten Randbildpunkt zurückgesprungen und der transparente Bereich mit dem Mittelwert der beiden begrenzenden, nichttransparenten Bildpunkte erneut aufgefüllt. Hier wollte man eigentlich eine Ausbreitung der Textur (Bildpunktwerte) durchführen, die auch ohne das Zurückspringen zum ersten Randpunkt möglich gewesen wäre.

Neben dieser Einschränkung ist das Modell auch für die blockbasierte Verarbeitung in Video-Kompressionsstandards nicht optimal geeignet. So kann

man das Modell zwar gut für den Block-Matching-Algorithmus anwenden, aber die Randbedingungen beim Block-Matching sind sehr viel enger und es gibt auch weitere Regularitäten, so dass spezielle Implementierungen [80] deutlich effizienter sein können. Genauso gibt es für Frequenztransformation, insbesondere für die in der Videokodierung verwendete DCT, optimierte Implementierungsvarianten [81][82], die auf der Ausnutzung von speziellen Regularitäten basieren. Die Betrachtung dieser individuellen Regularitäten liegt aber außerhalb der Aufgabenstellung.

Im Folgenden werden nun die einzelnen Adressierungsmethoden beschrieben.

2.1.2 Inter-Adressierung

Diese Adressierungsklasse wird eingesetzt, wenn man für die Berechnung des Ergebnisses an jedem Bildpunkt des Ergebnisbildes zwei Bildpunkte aus unterschiedlichen Bildern bzw. Bildbereichen, wie in Abbildung 2.1 dargestellt, benötigt.

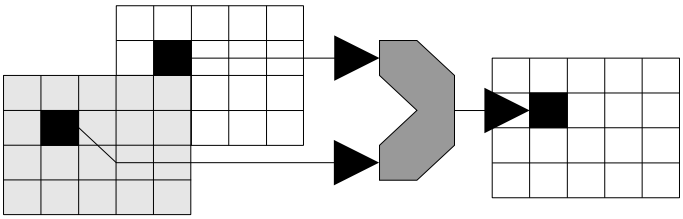


Abbildung 2.1: Inter-Adressierung. Das Ergebnis an jeder Bildposition wird aus zwei Bildpunkten aus unterschiedlichen Bildern bzw. Bildbereichen berechnet.

Anwendungsbeispiele für die Inter-Adressierung sind die Berechnung von (Absolut-)Differenzbildern oder die Berechnung des Mittelwerts der Differenzenquadrate (Mean-Square-Error= MSE) bei der Block-Matching-Bewegungsschätzung.

Zur Realisierung des zweiten Beispiels werden in dem verwendeten Software-Modell bereits zwei Abwandlungen der Inter-Adressierung benötigt. Zum einen müssen nicht die Ergebnisse an den Bildpunktpositionen gespeichert werden, sondern nur das aufsummierte Ergebnis, und zum anderen werden nur Teilbereiche der Eingangsbilder verarbeitet, die sich zusätzlich an unterschiedlichen Positionen in den Bildern befinden.

Um eine Bewegungsschätzung mit höheren Bewegungsmodellen durchzuführen, muss außerdem die Möglichkeit vorhanden sein, dass auf eines der

Eingangsbilder eine Koordinatentransformation angewendet wird. Das Softwaremodell für die Inter-Adressierung erlaubt dabei die Verwendung beliebiger Transformationsfunktionen, die durch einen Zeiger auf diese Funktion ausgewählt werden. Zusätzlich unterstützt das Software-Modell unterschiedliche Interpolationen von Bildpunktwerten, die durch die Transformation bedingt zwischen den Bildpunktpositionen liegen können.

Die spezielle Funktion, die mit der Bildverarbeitungsfunktion implementiert ist, ist dann nur durch die Parametrisierung der Adressierungsfunktion und durch die Verarbeitungsfunktion, die für einen Bildpunkt angewendet wird, festgelegt. Die Verarbeitungsfunktionen, die mit der Inter-Adressierung kombiniert werden, benötigen die Werte von zwei Eingangsbildpunkten als Eingangsdaten und die Bildpunktwerte des Bildpunktes im Ergebnisbild als Ausgangsdaten.

2.1.3 Intra-Adressierung

Die Intra-Adressierung wird bei Filteranwendungen eingesetzt. Dabei werden als Eingangsdaten für die Filter ein Bildpunkt, der sich an der korrespondierenden Stelle zum Ergebnisbildpunkt befindet, und seine Nachbarn verwendet. Abbildung 2.2 stellt diesen Sachverhalt dar.

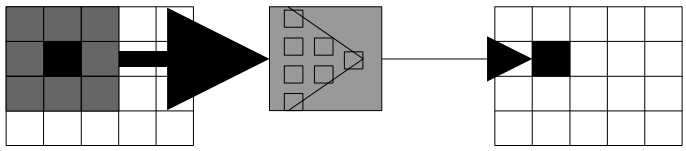


Abbildung 2.2: Intra-Adressierung. Das Ergebnis an jeder Bildposition wird aus dem zentralen Bildpunkt und seinen Nachbarn berechnet.

Obwohl das abstrakte Modell beliebige Nachbarschaftssysteme zulässt, beschränkt sich das Software-Modell auf die Nachbarschaftssysteme, die in Bild 2.3 dargestellt sind. Größere Nachbarschaftsysteme können im Software-Modell nur benutzt werden, wenn sich das Filter in kleinere, unterstützte Filter separieren lässt. Diese Einschränkung erscheint sinnvoll, da die Verwendung von Filtern mit mehr als 25 Eingangswerten wegen der hohen Rechenkomplexität fragwürdig ist.

Bei Filtern gibt es prinzipiell die Situation, dass einige Eingangswerte außerhalb des Eingangsbildes liegen. Deshalb unterstützt das Software-Modell dieser Adressierungsklasse folgende Möglichkeiten der Randwertersetzung:

- Ersetzung durch konstante Werte

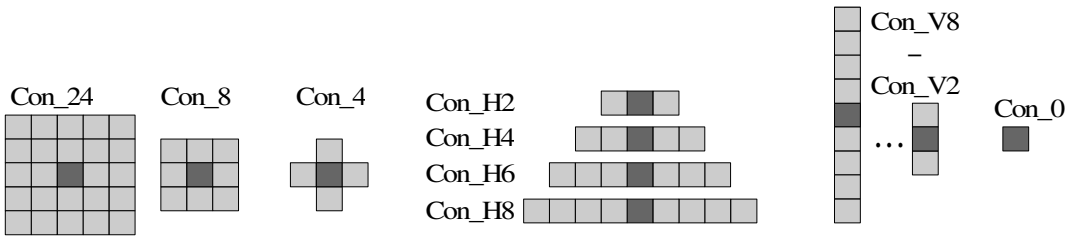


Abbildung 2.3: Von dem Software-Modell unterstützte Nachbarschaftssysteme. 2−dimensionale Nachbarschaftssysteme: CON_24 (Zentral-Bildpunkte mit 24 Nachbarn), CON_8 (mit 8 Nachbarn), CON_4 (mit 4 Nachbarn), eindimensionale Nachbarschaften (horizontal und vertikal): CON_H8/CON_V8, bis CON_H2/CON_V2 und ohne Nachbarn: CON_0

- Ersetzung mit einem Halteglied 0−ter Ordnung

Weitere denkbare sinnvolle Randwertersetzungen wären:

- eine periodische Fortsetzung
- eine gespiegelte periodische Fortsetzung

Die beiden letztgenannten Varianten werden im Software-Modell derzeit nicht unterstützt, da sich die für die Berechnung der Ersetzungswerte benötigten Operationen stärker von denen der Halteglieder unterscheiden und zu komplex sind. Wird dennoch eine der beiden Varianten benötigt, so kann das Bild um den nötigen Rand erweitert und die (gespiegelte) periodische Fortsetzung durch eine Inter-Adressierung mit Koordinatentransformation erzeugt werden. Schließlich kann dann die eigentliche Verarbeitung unter Verwendung der Intra-Adressierung durchgeführt werden.

Die Intra-Adressierung wird auch benötigt, wenn die örtliche Auflösung des Bildes geändert wird. Bei der Vergrößerung des Bildes (Überabtastung) wird für jeden Bildpunkt des Ausgangsbildes ein Wert berechnet. Die maximale Vergrößerung hängt dabei von der Größe des Filters zur Berechnung der Ausgangswerte ab. Wie bei der Randwertersetzung lassen sich nicht vorhandene Bildpunkte des Eingangsbildes durch eine Konstante (z.B. 0) bzw. durch Anwendung eines Haltegliedes 0−ter Ordnung ersetzen. Die Eingangsdaten für die Verarbeitungsfunktion (z.B. einem Tiefpassfilter) haben also bereits die örtliche Auflösung des Ergebnisbildes. Bei der Verkleinerung des Bildes (Unterabtastung) wird wie vom Modell gefordert auch für jeden Bildpunkt des Ausgangsbildes der Wert ermittelt. Diese Werte können z.B. durch Filterung (Anti-Aliasing) der Eingangswerte berechnet werden, die im Gegensatz zur Vergrößerung die örtliche Auflösung des Eingangsbildes haben.

Eine weitere Option ist die Unterscheidung von nicht rekursiven und rekursiven Filtern. Bei rekursiven Filtern sind Eingangsbild und Ausgangsbild identisch und die Ergebnisse an der aktuellen Bildpunktposition werden wieder in die Eingangswerte der nächsten Bildpunktposition zurückgekoppelt. Rekursive Filter werden unter anderem eingesetzt, um beliebig berandete und zusammenhängende Flächen zu bearbeiten [83]. Da rekursive Filter eine Kausalitätsrichtung haben, lässt sich mit einem einfachen Abarbeiten unter Umständen nicht die gesamte Fläche bearbeiten. Deshalb muss die Intra-Adressierung unterschiedliche Abarbeitungsrichtungen unterstützen. Die Anzahl der nötigen Abarbeitungen des Bildes mit unterschiedlichen Kausalitätsrichtungen, um eine zusammenhängende Fläche zu bearbeiten, hängt dabei von der Form und den gewählten Richtungen ab. Abbildung 2.4 stellt dar, wie zusammenhängende Flächen mit der Intra-Adressierung bearbeitet werden können.

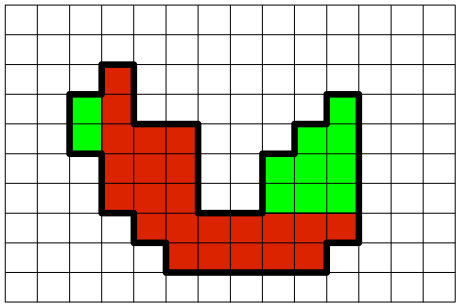


Abbildung 2.4: Verarbeitung von zusammenhängenden, beliebig berandeten Flächen mit der Intra-Adressierung. Beim ersten Bearbeitungsschritt (zeilenweise von links nach rechts, von oben nach unten) werden die dunklen Bildpunkte markiert. Mit dem zweiten Bearbeitungsschritt (zeilenweise von rechts nach links, von unten nach oben) werden die übrigen, hellen Bildpunkte markiert.

Die Schnittstelle zu den Verarbeitungsfunktionen, die mit der Intra-Adressierung kombiniert werden, benötigen als Eingangsdaten einen Zeiger auf die Bildpunktdaten des Bildausschnitts, der verarbeitet werden soll. Für die Ausgangsdaten werden die Bildpunktwerte des Bildpunktes im Ergebnisbild zurückgegeben. Zusätzlich stehen der Funktion als Eingangsdaten noch die Koordinaten des Ergebnisbildpunktes zur Verfügung.

2.1.4 Rekursive Adressierung

Eine weitere Möglichkeit die Rekursion zu implementieren, die zur Verarbeitung von beliebig berandeten und zusammenhängenden Flächen benötigt wird, ist der Einsatz von rekursiven Abarbeitungsfolgen [83].

Die Verarbeitungsmethoden sind wie bei der Intra-Adressierung Filter. Da bei der rekursiven Adressierung aber die örtliche Struktur im Sinne einer zusammenhängenden Fläche eine große Rolle spielt, machen Nachbarschaftssysteme, die mehr als die direkten Nachbarn umfassen, keinen Sinn. Deshalb werden hier nur die CON_8 und die CON_4 Nachbarschaften aus Abbildung 2.3 angewendet. Für Bildpunkte aus dem Nachbarschaftssystem wird überprüft, ob zwischen ihnen und dem zentralen Bildpunkt das Homogenitätskriterium erfüllt ist. Ist es nicht erfüllt und gibt es auch keinen anderen Pfad über den das Homogenitätskriterium zwischen den Bildpunkten erfüllt ist, gehören die Bildpunkte nicht zur gleichen zusammenhängenden Fläche. Ist das Kriterium erfüllt, so gehören die Bildpunkte zur gleichen zusammenhängenden Fläche. Das Homogenitätskriterium legt also fest, welches Kriterium die zusammenhängende Fläche definiert. Ist also das Kriterium erfüllt, wird der benachbarte Bildpunkt in die Liste der zu verarbeitenden Bildpunkte eingetragen. Wenn dieser Bildpunkt verarbeitet wird, wird wieder das Homogenitätskriterium für seine Nachbarn überprüft. Alle Nachbarn, die das Kriterium erfüllen, werden dann wieder zur Verarbeitung in die Liste eingetragen. Dieses Verhalten, dass ein Bildpunkt die Verarbeitung seiner Nachbarn bedingt, und diese Nachbarn die ihrer Nachbarn usw., erzeugt das rekursive Verhalten dieser Adressierungsvariante. In Abbildung 2.5 ist dargestellt, wie die Abarbeitungsfolge der Bildpunkte mit einer rekursiven Adressierung aussieht. Die Abarbeitungsfolge ist dabei mit dem Farbverlauf dargestellt. Im Vergleich dazu wird hier noch einmal Abbildung 2.4 referenziert, wobei dort die Rekursion mit der Verarbeitungsfunktion realisiert ist.

Im Gegensatz zur Inter- bzw. Intra-Adressierung werden insgesamt drei Funktionen benötigt, um das Verhalten von Bildverarbeitungsfunktion mit der rekursiven Adressierung zu spezifizieren. Diese drei Funktionen sind:

- ein Startkriterium, welches festlegt, ob ein Bildpunkt ein gültiger Startpunkt für die rekursive Abarbeitungsfolge ist
- ein Nachbarschafts- oder Homogenitätskriterium, welches ermittelt ob die Homogenität zwischen dem Bildpunkt und den Nachbarn erfüllt ist
- eine Verarbeitungsfunktion

Das Startkriterium wird dabei mit einer regulären Abarbeitungsfolge, genau wie bei der Intra-Adressierung, getestet. Wenn die Startpunkte für die

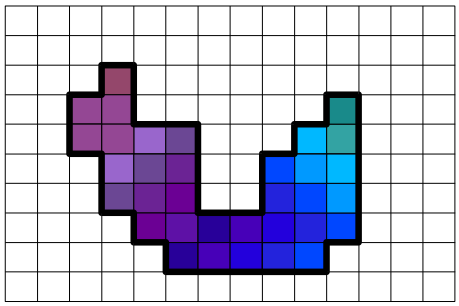


Abbildung 2.5: Verarbeitung von zusammenhängenden beliebig berandeten Flächen mit der FIFO-Adressierung. Die Abarbeitungsfolge wird durch den Farbverlauf dargestellt

rekursive Abarbeitungsfolge bereits bekannt sind, lassen sich die Startpunkte auch aus einer Startpunktliste lesen. In der hier gewählten Implementierung der rekursiven Adressierung wird das Start- und das Nachbarschaftskriterium noch durch einen Statusspeicher ergänzt, der den Verarbeitungszustand der Bildpunkte abspeichert. Der Statusspeicher soll das Verhindern von Endlosschleifen vereinfachen, die bei der rekursiven Adressierung entstehen können, wenn sich die Bildpunkte gegenseitig wieder aufrufen. Durch das Abfragen des Verarbeitungszustands lässt sich das wiederholte Aufrufen eines Bildpunktes leicht vermeiden. Dafür wird die folgende Unterscheidung der Zustände, die ein Bildpunkt annehmen kann, benötigt:

- noch nicht in die Liste der zu verarbeitenden Bildpunkte eingetragen
- in die Liste der zu verarbeitenden Bildpunkte eingetragen
- bereits bearbeitet (und somit aus der Liste ausgelesen)

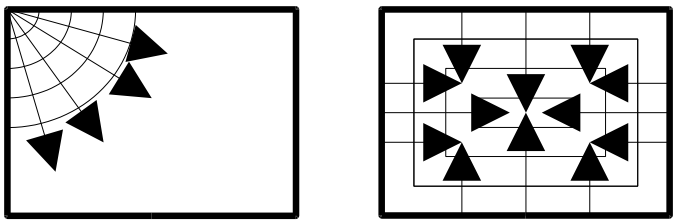
Außerdem kann ein Bildpunkt ungültig sein, falls er außerhalb des gültigen Bildbereichs liegt. Dieser Zustand wird aber nicht abgespeichert, da er sich einfach aus den Koordinaten berechnen lässt. Gemäß dem verwendeten Modell für die Bildverarbeitungsfunktionen wird mit der Verarbeitungsfunktion ein Ergebnis für den aktuellen Bildpunkt berechnet. Häufig sind bei Bildverarbeitungsfunktionen, die die rekursive Adressierung benutzen, die Verarbeitungsfunktionen sehr einfach. So ist oft das Ergebnis ein konstanter Wert, wie z.B. die Segmentnummer bei der Nummerierung von Segmenten, oder die Bildpunktdaten werden kopiert, wie z.B. die Segmentnummer beim Wasserscheide-Algorithmus. Im zweiten Fall ist ein direktes Kopieren der Information, wegen der gleichzeitigen Ausbreitung mehrerer Segmente nicht

möglich. Vielmehr muss zuerst festgestellt werden, von welchem Nachbarbildpunkt die Werte kopiert werden. Eine Möglichkeit die Suche nach dem Nachbarn zu umgehen ist, wenn man in dieser Situation von dem normalen Verarbeitungsmodell, das mit *Process-on-queue-out* bezeichnet wird, abweicht und die Ergebnisse für die Nachbarn berechnet, die in die Liste der zu bearbeitenden Bildpunkte eingetragen werden. Dieses Modell wird dann mit *Process-on-queue-in* bezeichnet. Prinzipiell ist die normal Verarbeitung, bei der ein Ergebnis für den aktuellen Bildpunkt berechnet wird, die allgemeinere, da hier zur Berechnung alle Eingangsdaten vorliegen. Demgegenüber lässt sich der Process-on-queue-in-Modus nur anwenden, wenn als Eingangsinformation nur die Werte des zentralen Bildpunktes benötigt werden.

Eine zentrale Komponente der rekursiven Adressierung ist die Liste, in die alle Bildpunkte, die verarbeitet werden sollen, eingetragen werden. Sie kann in zwei Varianten implementiert werden. In diesem Abschnitt wird die Adressierung mit einem *First-In-First-Out*-Speicher (FIFO) erläutert, die in der Literatur [24] häufig eingesetzt wird. Diese Struktur ist bereits in Kapitel 1.4.4 mit Hinblick auf die Implementierung betrachtet worden. An dieser Stelle wird nun der funktionale Ablauf beschrieben.

Beim Einsatz einer FIFO zum Erzeugen der Abarbeitungsfolge werden die Nachbarn des aktuellen Bildpunktes, die das Homogenitätskriterium erfüllen, in die FIFO geschrieben (*queue-in*). Eingetragen werden dabei die Koordinaten (Adresse) des Bildpunktes. Wenn alle Bildpunkte, die vorher in die FIFO geschrieben wurden, aus der FIFO gelesen wurden, wird der Bildpunkt wieder als aktueller Bildpunkt aus der FIFO gelesen (*queue-out*). Wenn man also mit einem Bildpunkt, dem Startbildpunkt, anfängt, dann werden zuerst alle Nachbarn mit einer örtlichen Distanz von 1 in die FIFO geschrieben. Beim Lesen und Verarbeiten der Bildpunkte mit dem Distanzwert 1, werden deren Nachbarn wieder eingetragen, die dann eine örtliche Distanz von 2 zum Startpunkt haben. Deren Nachbarn haben dann wieder einen Distanzwert, der um 1 größer ist. Die Konsequenz ist, dass die Reihenfolge, in der die Bildpunkte verarbeitet werden, nach der örtlichen Distanz zum Startpunkt geordnet ist. Die Abarbeitungsfolge verhält sich wie eine sich ausbreitende Wellenfront. Dieses Verhalten stellt sich auch ein, wenn man nicht mit einem Startpunkt beginnt, sondern eine Menge von Startpunkten hat (z.B. die Peripheriepunkte einer Form). Abbildung 2.6 zeigt dieses Verhalten für einen Startpunkt und eine Startpunktmenge, die sich aus den Randpunkten der Form zusammensetzt. In beiden Fällen erfüllen Bildpunkte, die außerhalb der Form liegen, nicht das Homogenitätskriterium.

Die beiden Varianten, mit nur einem Startpunkt und mit einer Startpunktmenge, werden wie folgt bezeichnet:



a) Ausbreitung mit einem Startpunkt (Ecke links oben)

b) Ausbreitung mit einer Startpunktmenge (Randpunkte)

Abbildung 2.6: Ausbreitungsfront bei der rekursiven Adressierung mit einer FIFO: a) bei Verwendung eines einzelnen Startpunktes und b) bei Verwendung einer Startpunktmenge.

- *non-collected*-Modus, bei dem nach dem Finden eines gültigen Startpunktes sofort mit der rekursiven Adressierung angefangen wird
- *collected*-Modus, bei dem zuerst alle Startpunkte gesammelt werden und dann die rekursive Adressierung von der Startpunktmenge anfängt

Der non-collected-Modus wird häufig eingesetzt, wenn nur eine Fläche bearbeitet werden soll und das Distanzfeld keine Rolle spielt. Beispiele sind der Fragmentierungstest oder die Nummerierung von Segmenten. Der collected-Modus wird benutzt, wenn man unterschiedliche Segmente parallel bearbeiten will und das Distanzfeld bei der Ausbreitung eine Rolle spielt (z.B. beim Wasserscheide-Algorithmus, zum Positionieren der Segmentgrenze bei einer mehrere bildpunktbreiten Gratlinie).

In manchen Situationen ist es aber nicht ausreichend, die Abarbeitungsfolge nur nach der örtlichen Distanz zu ordnen. So kann es z.B. nötig sein, noch weitere Kriterien zu benutzten, um die Abarbeitungsfolge festzulegen. Beim Wasserscheide-Algorithmus richtet sich die Ausbreitungsgeschwindigkeit auch nach dem Grad der Homogenität der Bildpunkte. Implementiert wird dieses Verhalten mit hierarchisch geordneten FIFOs [84]. Durch Erweiterung des Nachbarschaftskriteriums, so dass es nicht nur festlegt, ob die Homogenität erfüllt ist, sondern auch wie stark die Homogenität ist, kann festgelegt werden, in welche FIFO, die eine Hierarchiestufe für die Stärke der Homogenität repräsentiert, ein Bildpunkt einsortiert wird. Die FIFOs werden dann in der Reihenfolge ihrer Hierarchien abgearbeitet. Innerhalb einer Hierarchiestufe wird die Abarbeitungsfolge durch die örtliche Struktur bestimmt. Da die Bildpunkte weiterhin nacheinander bearbeitet werden, kann

das Kriterium eine höhere Dimensionalität haben. Es muss sich nur auf die eindimensionale (lineare) Abfolge abbilden lassen.

Die FIFOs selber lassen sich mit der in Abbildung 1.11 (Seite 27) dargestellten Architektur implementieren. Eine Beschränkung dieser Architektur ist, dass jeder Bildpunkt nur einmal in die FIFOs eingetragen werden kann. Ein möglicher Konfliktfall kann eintreten, wenn ein Bildpunkt bereits in der FIFO mit einer niedrigen Priorität abgelegt ist und von einem anderen Nachbarn in eine FIFO mit einer höheren Priorität abgelegt werden soll. Eine Lösung für den Konfliktfall, die in Kapitel 2.2 vorgeschlagen wird, ist, dass jeder Bildpunkt mehrfach in die FIFOs eingetragen werden kann. Der Bildpunkt wird aber weiterhin nur einmal verarbeitet, nämlich beim ersten Auslesen. Dieser Modus wird mit *Multi-queue-in*-Modus bezeichnet.

Im allgemeinen Fall gibt es nicht nur einen Modus, bei dem ein Bildpunkt mehrfach in die FIFO eingetragen werden kann, sondern auch den Fall, dass ein Bildpunkt mehrfach verarbeitet wird. Ein Anwendungsfall hierfür ist das Abfahren einer Objektkontur bei der Extraktion des MPEG−7-Konturdeskriptors. Abbildung 2.7 zeigt, dass zum Abfahren einer Kontur ein Bildpunkt in unterschiedliche Richtungen mehrfach bearbeitet werden kann. Diese Variante der rekursiven Adressierung, die das mehrfache Verarbeiten eines Bildpunktes erlaubt, wird mit *Multi-queue-out*-Modus bezeichnet. Da in diesem Fall das Statusfeld nicht mehr berücksichtigt wird, muss nun das Nachbarschaftskriterium auch eine Abbruchbedingung einschließen, um Endlosschleifen zu verhindern.

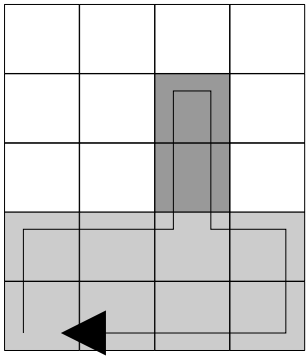


Abbildung 2.7: Konturtransformation von Teilflächen mit einer Stärke von einem Bildpunkt. Zum Abarbeiten der Kontur müssen die dunkelgrau markierten Bildpunkte zweimal, je mit unterschiedlicher Richtung, bearbeitet werden.

Das mehrfache Verarbeiten eines Bildpunktes liegt aber nicht mehr in-

nerhalb des gewählten Gültigkeitsbereichs des Modells. Betrachtet man aber das Beispiel der Konturtransformation genauer, stellt man fest, dass hier eigentlich die Konturelemente betrachtet werden und nicht die Bildpunkte. Jedes Konturelement wird hier wiederum nur ein einziges Mal verarbeitet. Auf dieser Basis lässt sich eine Abbruchbedingung formulieren. Die Konturtransformation lässt sich dann auf das Modell abbilden. Formal ist dies möglich, weil die mehrfache Verarbeitung in der Ausbreitungsphase stattfindet und zusätzliche Funktionalitäten sich als Abbruchbedingung formulieren lassen. Damit können die zusätzlichen Funktionalitäten in dem benutzerdefinierten Nachbarschaftskriterium implementiert werden und Eingriffe in den statischen Adressierungsteil sind nicht nötig.

2.1.5 Parametrisierung

In den drei vorherigen Unterabschnitten wurden die drei unterschiedlichen Adressierungsmethoden, die Inter-, Intra- und rekursive Adressierung, beschrieben. In diesen Kapiteln wurden bereits Variationen dieser Adressierungsmethoden genannt. Hier werden diese vervollständigt und noch einmal systematisch erläutert.

Ein Parameter, der sich auf alle drei Adressierungsmethoden anwenden lässt, ist die Anzahl der Bildkanäle des Eingangsbildes/der Eingangsbilder und des Ergebnisbildes. Beispielsweise können hier der Helligkeitskanal und die Farbkanäle gleichzeitig verarbeitet werden. Darüber hinaus gibt es noch oft einen Bildkanal für die Segmentnummer. Diese Kanäle können pro Eingangs- und Ergebnisbild ausgewählt werden. Neben den reinen Eingangs- und Ausgangskanälen unterstützt das Modell auch Kanäle, die als Eingangs- und Ausgangskanal gleichzeitig dienen. Ein Beispiel hierfür ist der Statuskanal, der bei der rekursiven Adressierung verwendet wird. Im abstrakten Modell kann es beliebig viele Kanäle geben. Bei der Implementierung in einem Software- oder Hardware-Modell hingegen muss die Anzahl beschränkt sein.

In vielen Fällen sind dabei die Eingangsbilder von den Ergebnisbildern entkoppelt. Praktisch können die Bilder jedoch den selben Bildspeicher referenzieren. Zum Beispiel passiert dies bei allen rekursiven Filtern. Hier müssen die Werte der Ergebnisbildpunkte auch sofort wieder in alle zwischengespeicherten Kopien der Werte der Eingangsbildpunkte geschrieben werden. Kopien der Eingangswerte werden angelegt, um z.B. die bereits aus dem Bildspeicher geladene Bildpunkte zur Verarbeitung des nächsten Bildpunktes wieder zu verwenden. Dies ist eine sinnvolle Optimierung für die Intra-Adressierung, bei der das Eingangsfenster mit der Nachbarschaft des zu verarbeitenden Bildpunktes immer um einen Bildpunkt weitergeschoben wird. Für diesen

Fall kann signalisiert werden, dass die Ergebnisse vor dem Verschieben wieder in den zentralen Eingangsbildpunkt der Nachbarschaft kopiert werden sollen.

Ein Parameter für die nicht-rekursiven Adressierungsmethoden (Inter-, Intra-Adressierung und Start-*Pixel-Scan*) ist das Schema, das die Reihenfolge der Verarbeitung der Bildpunkte festlegt. Mögliche Abarbeitungsschemen sind der horizontale *Scan* und der vertikale *Scan*. Meist wird ein horizontaler Scan verwendet, da die Bildpunkte in dieser Reihenfolgen übertragen bzw. aus dem Kamerasensor ausgelesen und im Speicher abgelegt werden. Dieses Scan-Schema ist außerdem für eindimensionale horizontale Filter geeignet. Allerdings ist ein horizontaler Scan für eindimensionale vertikale Filter hochgradig ungeeignet. In diesem Fall lassen sich durch horizontales Verschieben des Eingangsfensters keine Bildpunkte wieder verwenden. Dies ist hingegen bei Verwendung eines vertikalen Scans möglich, weswegen dieser hier sehr gut geeignet ist. Neben dem normalen horizontalen oder vertikalen Scan gibt es noch die Möglichkeit, die Bildpunkte horizontal oder vertikal mäanderförmig abzufahren. Bei den mäanderförmigen Scans wird am Zeilenanfang nicht die komplette Nachbarschaft neu geladen, sondern nur das Eingangsfenster um eine Zeile verschoben.

Eine weitere Möglichkeit des Scans ist bei der rekursiven Adressierung gegeben, wenn die Liste mit den Startpunkten für den Ausbreitungsprozess bereits bekannt ist. In diesem Fall werden nur die Bildpunkte einer Start-*Pixel*-Liste geladen.

Wie bereits vorher angedeutet wurde, muss auch nicht in jedem Fall das gesamte Bild verarbeitet werden. Für die Verarbeitung von rechteckigen Teilbereichen des Bildes, können die Koordinatenbereiche eingeschränkt werden. Zur Spezifikation des Teilbereichs, wird die Größe des Teilbereichs in zwei Parametern (Breite und Höhe) angegeben. Außerdem wird die linke obere Ecke des Teilbereichs festgelegt. Da im allgemeinen Fall die Teilbereiche im (in den) Eingangsbild(ern) und dem Ergebnisbild an unterschiedlichen Positionen liegen können, wird für jedes Bild gesondert die linke obere Ecke festgelegt. Ein Beispiel, bei dem Teilbereiche der Bilder im Eingangsbild und im Ergebnisbild an unterschiedlichen Positionen liegen, ist die Bewegungskompensation.

Eine weitere Variation ist das Verwenden von unterschiedlichen Nachbarschaftssystemen (siehe Abbildung 2.3 auf Seite 53). Diese können $0-$dimensional, $1-$dimensional oder $2-$dimensional sein. Bei der rekursiven Adressierung machen aber nur $2-$dimensionale Nachbarschaftssysteme Sinn. In der Regel werden hier auch nur Nachbarschaftssysteme verwendet, die nur die direkten Nachbarn enthalten (CON_4 oder CON_8). Bei der Verwendung von 1- oder $2-$dimensionalen Nachbarschaftssystemen besteht die

Möglichkeit, dass Bildpunkte der Nachbarschaft eines Bildpunktes außerhalb des gültigen Bildbereichs liegen. Dies ist bei horizontalen Nachbarn in den ersten und letzten Spalten und bei vertikalen Nachbarn in den ersten und letzten Zeilen der Fall. Abbildung 2.8 zeigt diesen Fall. Grau markiert sind die Bildpunkte der Nachbarschaft, die außerhalb des Bildes liegen. Für diese Bildpunkte muss angegeben werden, welcher Wert der Verarbeitungsfunktion übergeben wird.

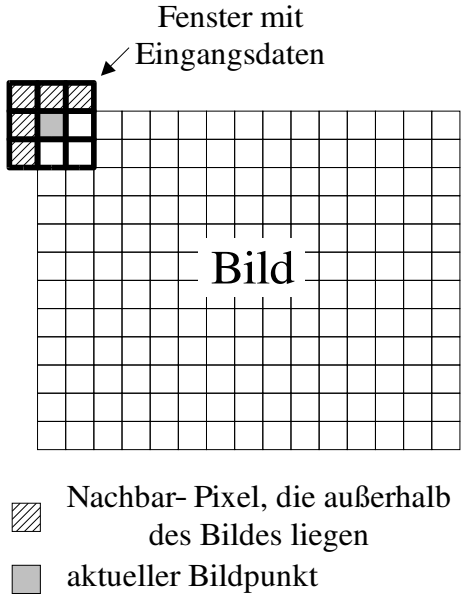


Abbildung 2.8: Beispiel, bei dem eine Randwertersetzung von Bildpunktwerten nötig ist, die außerhalb des gültigen Bildbereichs liegen. Die grau markierten Bildpunkte der CON_8 Nachbarschaft an der Bildpunktposition (0,0) liegen außerhalb des Bildes.

Weitere Parameter sind bereits in den vorhergehenden Unterkapiteln 2.1.2, 2.1.3 und 2.1.4 beschrieben worden. Tabelle 2.1 fasst die möglichen Parameter des Modells für Bildverarbeitungsfunktionen zusammen.

2.2 Entwicklung der rekursiven Adressierung mit hierarchischer dualer LIFO

In diesem Unterkapitel werden nun detailliert unterschiedliche Implementierungsvarianten der rekursiven Adressierung diskutiert. Als ein wesentliches Resultat dieser Arbeit wird dann ein neues Prinzip für Ausbreitungsprozesse

Parameter	Inter-Adressierung	Intra-Adressierung	Segment-Adressierung
Kanäle	X	X	X
rekursive Verarbeitung	-	X	-
Scan-Schemen	X	X	X
Verarbeitung von Teilbereichen	X	X	-
Randwertersetzung	-	X	X
Skalierung von Bildern	-	X	-
Koordinaten-transformation	X	-	-
Process-on-queue-in/out	-	-	X
Collected- / non-collected-Modus	-	-	X
Multi-queue-in	-	-	X
Multi-queue-out	-	-	X

Tabelle 2.1: Mögliche Parameter der Bildverarbeitungsfunktion in Abhängigkeit des Adressierungsverfahrens.

vorgestellt, das eine Verbesserung gegenüber bisherigen Implementierungen darstellt.

Die wohl einfachste Implementierung für Ausbreitungsprozesse mit rekursiver Adressierung ist aus der rekursiven Programmierung abgeleitet. Dabei ruft sich eine Funktion wieder solange selber auf, bis eine Abbruchbedingung erfüllt ist. Faktisch wird dabei ein Funktions-Stack auf- und abgebaut. Bei einer Anwendung des Prinzips auf die rekursive Adressierung bedeutet dies, dass diese Funktion eine Verarbeitungsroutine für den Bildpunkt abarbeitet und dann das Homogenitätskriterium für alle Nachbarn überprüft, die bisher noch nicht im Statuskanal (siehe Kapitel 2.1.4) als bereits auf dem Stack (LIFO) eingetragen markiert sind. Ist das Homogenitätskriterium erfüllt, ruft sich die Funktion selber wieder mit den entsprechenden Nachbarbildpunkten auf. Da nun der Nachbarbildpunkt der aktuelle Bildpunkt ist, wird der Bildpunkt verarbeitet und seine Nachbarn überprüft. Der Prozess wiederholt sich, bis für einen aktuellen Bildpunkt keine Nachbarn mehr vorhanden sind, die

das Homogenitätskriterium erfüllen und als nicht in der LIFO eingetragen markiert sind. Nachdem für alle gefundenen Nachbarbildpunkte die Funktion sich selber aufgerufen hatte, wird ein zweiter Teil der Verarbeitungsroutine ausgeführt und zum vorherigen Aufruf zurückgekehrt. Die Verarbeitung besteht hier also aus zwei Teilen. Der erste Teil wird ausgeführt, wenn der Bildpunkt in die LIFO eingetragen wird und der zweite Teil, nachdem der Bildpunkt wieder aus der LIFO ausgelesen wurde. Da zwischen den beiden Teilen der Verarbeitung andere Bildpunkte verarbeitet werden, müssen die Teilergebnisse für jeden Bildpunkt nach der ersten Teilverarbeitung in den Bildspeicher gesichert werden und vor der Ausführung des zweiten Teils wieder geladen werden. Ein Beispiel wird später bei der Beschreibung für die Erzeugung eines Distanzfeldes gegeben.

Bei jedem Aufruf der Funktion werden alle lokalen Variablen und Parameter der Funktion neu auf dem Programm-*Stack* [1] angelegt. Betrachtet man diese Vorgehensweise genauer, stellt man fest, dass nur die Bildpunktkoordinaten auf dem Stapelspeicher abgelegt werden müssen. Dies lässt sich mit einer expliziten Implementierung einer LIFO in dieser Funktion realisieren. Die Funktion muss sich dann nicht mehr selber aufrufen und die Implementierung wird deutlich effizienter.

Will man erreichen, dass die Abarbeitungsfolge, wie in Abbildung 2.9 dargestellt, entlang einer Kontur verläuft, muss die Reihenfolge, in der die Nachbarn geprüft werden, in einer bestimmten Art festgelegt werden. Dazu werden die Nachbarn, ausgehend von dem Nachbarn, von dem der aktuelle Bildpunkt aus aufgerufen wurde, im Uhrzeigersinn geprüft. Dabei muss der Nachbar, der zuvor der aktuelle Bildpunkt war, nicht geprüft werden, da dieser in jedem Fall als in die LIFO eingetragen markiert sein muss. Wird im Uhrzeigersinn geprüft, liegen die Kontur oder die bereits in die LIFO eingetragenen Bildpunkte links von der Ausbreitungsrichtung, wie in Abbildung 2.9 zu sehen ist. Mit dieser Abarbeitungsreihenfolge lassen sich nun Distanzfelder erzeugen, wenn alle Konturpunkte mit einem Distanzwert von Null vorausgesetzt werden. Dies ersetzt dann den collected-Modus, der bei dieser LIFO-Implementierung nicht direkt umgesetzt werden kann.

Im allgemeinen lassen sich Distanzfelder aber nicht fehlerfrei berechnen. Abbildung 2.10 zeigt ein Beispiel, bei dem das Distanzfeld von einem beliebigen Bildpunkt aus (weiß markiert mit dem Distanzwert 0) in einer beliebig berandeten Form erstellt werden soll. Hier sind die Distanzwerte auf der Kontur nicht bekannt. a) zeigt die Distanzwerte beim Aufbauen des Stapelspeichers und b) die endgültigen Distanzwerte nach dem Abbauen des Stapelspeichers. Der dunkelgrau markierte Bildpunkt auf der rechten Seite des

[1] Stack = Stapelspeicher oder LIFO (Last-In-First-Out)

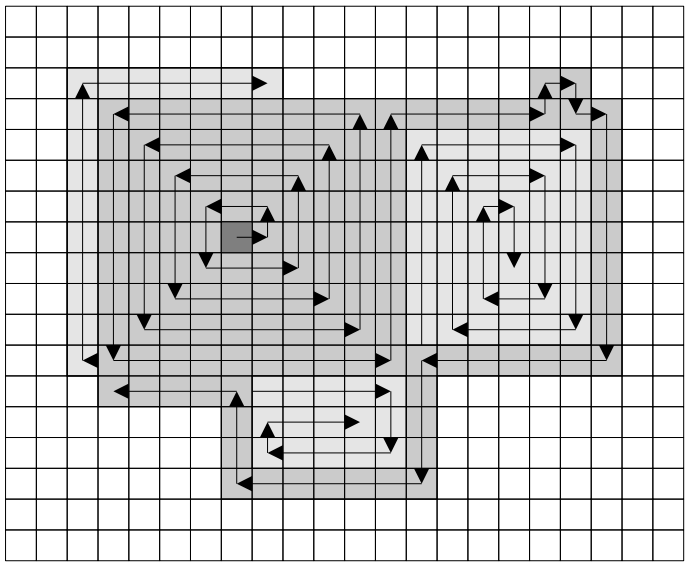
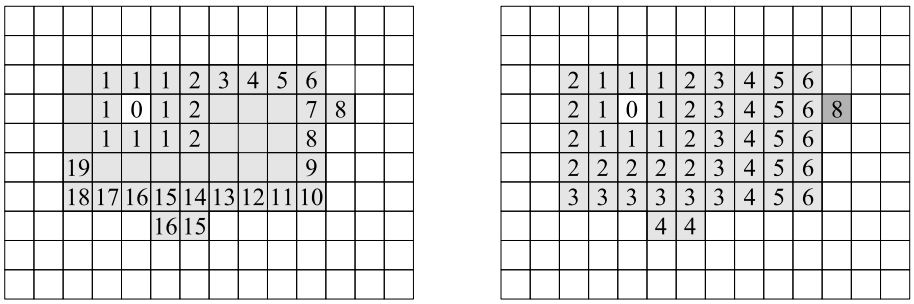


Abbildung 2.9: Schneckenförmige Ausbreitung mit einfacher LIFO. Dunkelgrau markiert ist der Startpunkt. Mittelgrau markierte Punkte werden beim Aufbau des Stacks bearbeitet und hellgrau markierte Punkte erst nach teilweisem Abbau des Stacks.

Objektes ist allerdings beim Hinweg (Aufbau des Stapelspeichers) schon wieder vom Stapelspeicher abgebaut worden und wird somit auf dem Rückweg nicht mehr verarbeitet. So kann der Distanzwert auch nicht mehr korrigiert werden.

Die Erzeugung eines Distanzfeldes wird aber nicht in jedem Fall benötigt. Beispiele hierfür sind das Markieren eines Segments mit homogener Farbe und Helligkeit mit einer Segmentnummer oder der Fragmentierungstest. Beide Funktionen werden in dem in Kapitel 1.6 vorgestellten Algorithmus, der als Grundlage für diese Arbeit dient, benutzt. In diesen Fällen hat die Implementierung mit einer einfachen LIFO den Vorteil, dass die Position des aktuellen Bildpunktes immer nur um eine Position verschoben wird. Dadurch kann das Laden der kompletten Nachbarschaft verhindert werden. Ein gravierender Nachteil dieser Implementierung ist allerdings, dass die Verarbeitung im allgemeinen Fall sowohl beim Hineinschreiben in als auch beim Auslesen aus der LIFO durchgeführt werden muss und somit zusätzliche Schreiboperationen nötig sind. Dies bedeutet auch, dass der Bildpunkt zweimal geladen werden muss, beim Aufbauen des Stapelspeichers, und beim Abbauen des Stapelspeichers. Da sich mit der einfachen LIFO nicht allgemeingültig Distanzfelder berechnen lassen, ist die Verwendbarkeit nicht immer garantiert,

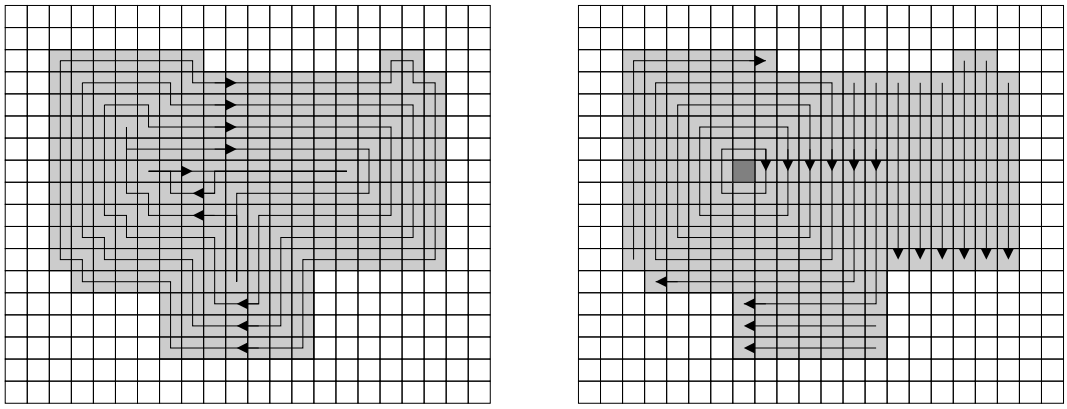


a) Distanzwerte beim Aufbauen des Stacks

b) Distanzwerte beim Abbauen des Stacks

Abbildung 2.10: Fehler bei der Erzeugung von Distanzfeldern mit einfacher LIFO. Der dunkelgrau markierte Bildpunkt hat einen inkorrekten Distanzwert.

so dass diese Implementierungsvariante im Folgenden nicht weiter betrachtet wird.



a) *collected* Modus

b) *non- collected* Modus

Abbildung 2.11: Abarbeitungsfolge der rekursiven Adressierung mit FIFO: a) collected Modus (Start vom Objektrand), b) non-collected Modus (Start vom dunkelgrau markierten Bildpunkt).

Die Implementierung, die sich im allgemeinen Fall anwenden lässt, ist die FIFO-Implementierung. Eine Architektur für diese Variante ist bereits in Kapitel 1.4.4 und die Funktionsweise in Kapitel 2.1.4 vorgestellt worden. An dieser Stelle wird nun die Ausbreitungsreihenfolge betrachtet. Abbildung 2.11

zeigt die Abarbeitungsreihenfolge unter Verwendung einer CON_8 Nachbarschaft mit a) allen Randpunkten als Startpunkte und b) einem Startpunkt. Die Ausbreitung verhält sich wie eine sich ausbreitende Wellenfront. Alle Bildpunkte, die sich auf einer Distanzebene zu den Startbildpunkten befinden, werden in einer zusammenhängenden Reihenfolge abgearbeitet, weshalb in jedem Fall Distanzfelder fehlerfrei berechnet werden können. Bei Verwendung einer CON_8 Nachbarschaft ergibt sich ein Distanzfeld mit Schachbrettdistanz:

$$d = max(|x_1 - x_2|, |y_1 - y_2|) \tag{2.1}$$

Eine CON_4 Nachbarschaft erzeugt ein Distanzfeld mit der Manhattendistanz:

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{2.2}$$

Der Drehsinn der Abarbeitungsfolge der Bildpunkte auf den einzelnen Distanzebenen zeigt immer in die gleiche Richtung. Dabei sind Sprünge zwischen den Positionen aufeinander folgender Bildpunkte möglich, so dass für jeden Bildpunkt immer die komplette Nachbarschaft geladen werden muss (im Vergleich zum Verschieben des Eingangsfensters, bei dem immer nur die neu hinzugekommenen Bildpunkte nachgeladen werden müssen).

Das in dieser Arbeit neu entwickelte Ausbreitungsprinzip ist weitgehend funktional äquivalent der FIFO-Ausbreitung. Dabei wird das LIFO-Prinzip mit dem FIFO-Prinzip zu einem neuen Ausbreitungsprinzip, dem dualen LIFO (DLIFO)-Prinzip, kombiniert. Gegenüber der oben vorgestellten einfachen LIFO werden hier alle Bildpunkte, die in die LIFO eingeschrieben werden können, gleichzeitig eingeschrieben. Für das Einschreiben der Nachbarn wird eine andere LIFO benutzt, als die, aus der gelesen wird. Wegen der Benutzung von zwei LIFOs wird dieses Konzept auch duale LIFO genannt. Durch die Trennung von Lese- und Schreib-LIFO lässt sich erreichen, dass alle Bildpunkte einer Distanzebene wie bei der FIFO-Ausbreitung in einer zusammenhängenden Reihenfolge abgearbeitet werden. Die Nachbarn dieser Bildpunkte haben einen Distanzwert, der genau um 1 größer ist, als der Distanzwert der aktuellen Lese-LIFO. So haben alle Bildpunkte der Schreib-LIFO wieder den gleichen Distanzwert. Ist die Lese-LIFO leer, wird aus der Schreib-LIFO die Lese-LIFO und umgedreht. Abbildung 2.12 zeigt das Blockbild der dualen LIFO mit den beiden LIFO-Speichern und den zugehörigen Zeigern auf das aktuelle Element des Stapelspeichers.

Im Vergleich zur FIFO-Ausbreitung (siehe Abbildung 1.11 auf Seite 27) ist zu sehen, dass die gleiche Anzahl an Zeigern benötigt wird (bei der FIFO-Ausbreitung ein Start- und ein Endzeiger). Der einzige funktionale Unterschied zur FIFO-Ausbreitung besteht in der alternierenden Drehrichtung bei jedem Wechsel des Distanzwerts während des Ausbreitungsprozesses. Die

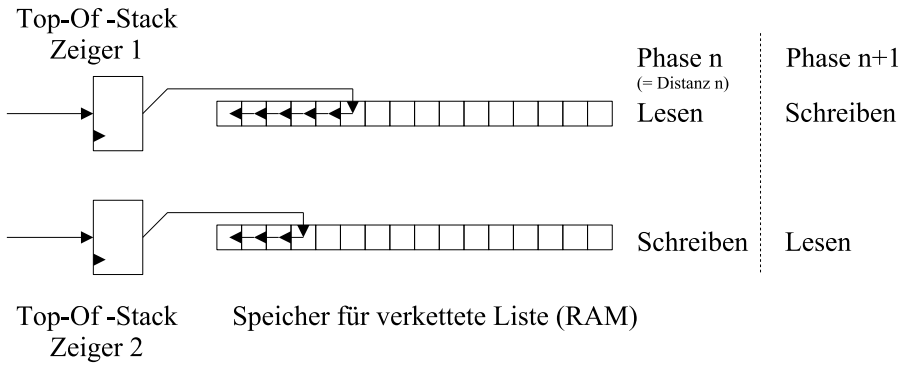
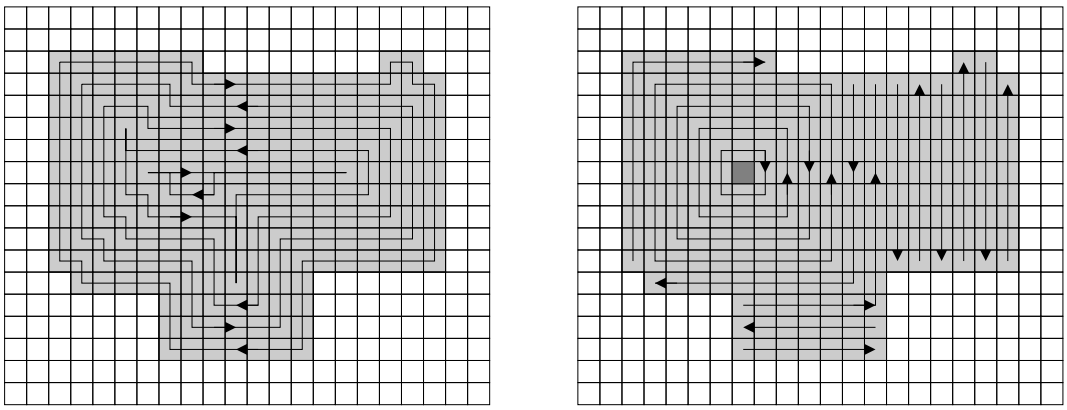


Abbildung 2.12: Architektur für die duale LIFO-Implementierung: die *Top-Of-Stack*-Zeiger zeigen auf die aktuellen Bildpunkte in der Lese- und Schreib-LIFO.

strenge Trennung nach Distanzebene in der Abarbeitungsreihenfolge bleibt erhalten. Die Reihenfolge innerhalb der Distanzebene spielt hingegen keine Rolle, oder zumindest ist in keiner der im ausgewählten Algorithmus benutzten Bildverarbeitungsfunktionen mit rekursiver Adressierung die Abarbeitungsfolge innerhalb eines Distanzwerts spezifiziert. Die Reihenfolge kann aber dennoch das Ergebnis der Funktionen beeinflussen, so dass die FIFO und die DLIFO nicht vollständig identisch sind. Unterschiede treten deshalb nur an Stellen auf, an denen das Ergebnis instabil ist und haben statistisch gesehen keinen Einfluss auf die Qualität des Algorithmus. Abbildung 2.13 zeigt den Ausbreitungsprozess unter Verwendung einer CON_8 Nachbarschaft mit a) collected und b) non-collected Modus.

Die Rechenkomplexitäten der DLIFO- und FIFO-Ausbreitung sind im Fall einer Software-Implementierung identisch. Ein wesentlicher Unterschied besteht aber bei einer Hardware-Implementierung. Wie in Kapitel 2.7 vorgeschlagen wird, bieten sich bei der Hardware-Implementierung der Einsatz einer zeitlichen Parallelisierung mit Fließbandverarbeitung an. In diesem Fall werden zuerst die Koordinaten des Bildpunktes aus der LIFO geladen (Stufe 1). Anschließend werden die Bildpunktwerte des Eingangsfensters aus dem Bildspeicher geladen (Stufe 2), das Homogenitätskriterium überprüft, und die Verarbeitung des Bildpunktes/der Bildpunkte (bei Process-on-queue-out/-in) durchgeführt (Stufe 3). Die Bildpunkte, die das Homogenitätskriterium erfüllen, werden dann in die LIFO eingetragen und die Ergebnisse in den Bildspeicher geschrieben (Stufe 4). Als Resultat kann festgestellt werden, dass das Auslesen aus der LIFO in der ersten Stufe des Fließbandes ausgeführt wird und das Einschreiben in der Vierten. Ist die LIFO leer, muss die Fließ-



a) c*ollected* Modus b) *non- collected* Modus

Abbildung 2.13: Abarbeitungsfolge der rekursiven Adressierung mit dualer LIFO: a) collected Modus (Start vom Objektrand), b) non-collected Modus (Start vom dunkelgrau markierten Bildpunkt).

bandverarbeitung warten, bis wieder neue Daten in die LIFO eingetragen werden. Ist der Füllstand der LIFO niedrig, kann es oft zu *Pipeline-Stalls* kommen. Im Fall einer leeren LIFO kann die Wartezeit verkürzt werden, wenn der LIFO-Speicher beim Einschreiben und Auslesen umgangen werden kann. Diese Technik wird auch mit *Bypassing* bezeichnet. Die Problembeschreibung ist dabei unabhängig davon, ob mit einer LIFO oder einer FIFO gearbeitet wird. Der wesentliche Unterschied besteht nun darin, dass bei der LIFO-Ausbreitung bei jedem Wechsel der Distanzebene die Wartezeit durch Bypassing verkürzt werden kann. Bei Verwendung einer FIFO muss hingegen Aufwand getrieben werden, um einen Wechsel des Distanzwerts festzustellen. Dadurch ergibt sich ein Vorteil durch Verwendung einer DLIFO gegenüber einer FIFO.

Noch deutlicher wird der Vorteil von LIFOs, wenn man die Erweiterung zur hierarchischen dualen LIFO betrachtet. Beim Einsatz von FIFOs in den niedrigeren Hierarchiestufen wird pro Stufe, wie bei jeder FIFO, ein Start- und ein Endzeiger benötigt. Die Architektur für hierarchische FIFOs wurde bereits in Abbildung 1.12 auf Seite 29 dargestellt. Setzt man normale LIFOs bei den niedrigeren Hierarchiestufen ein, wird für diese nur noch ein Zeiger pro Stufe benötigt. Der Einsatz einer normalen LIFO ist problemlos möglich, da in den niedrigeren Stufen nur die Startbildpunkte benötigt werden. Diese liegen immer auf dem selben Distanzwert. Der Ausbreitungsprozess nach der geodätischen Distanz auf der aktuellen Hierarchiestufe wird dann wieder mit einer dualen LIFO durchgeführt. Wahlweise könnte für die aktuelle Hierar-

chiestufe auch eine FIFO eingesetzt werden. Die Kombination von LIFOs für die niedrigeren Hierarchiestufen mit einer dualen LIFO wird im Folgenden als hierarchische duale LIFO (HDLIFO) bezeichnet. Dadurch, dass nun für die niedrigeren Stufen nur noch ein Zeiger auf das aktuelle Element benötigt wird, lässt sich der Implementierungsaufwand in Form von Chip-Fläche reduzieren. Das folgende Beispiel soll dies mit einer Überschlagsrechnung verdeutlichen. Wie in Kapitel 1.4.4 beschrieben wurde, werden die Zeiger mit Registern implementiert. Dies ermöglicht ein Zurücksetzen der Zeiger am Anfang der Bildverarbeitungsfunktion. Ein Register mit *Enable* und *Reset* benötigt pro Bit 9 Gatteräquivalente Chipfläche. Unter Annahme der FIFO-Architektur mit statischer Elementzuordnung werden bei einer Bildgröße von 352x288 Bildpunkten (CIF-Format) 18 Bit benötigt. Bei der Verwendung einer hierarchischen FIFO (HFIFO) mit 256 Hierarchiestufen werden für die Implementierung der Zeiger

$$N = 9[Gates/Bit] * 18[Bits/Zeiger] * 256[Stufen] * 2[Zeiger/Stufe] \quad (2.3)$$

$$= 82944[Gates]$$

benötigt. Bei der Verwendung einer HDLIFO wird hingegen nur 1 Zeiger pro Stufe benötigt und für die aktuelle Stufe noch ein weiterer Zeiger zur Ergänzung zu einer dualen LIFO:

$$N = 9[Gates/Bit] * 18[Bits/Zeiger] * \quad (2.4)$$

$$(256[Stufen] * 1[Zeiger/Stufe] + 1[Zeiger])$$

$$= 41634[Gates]$$

Wie in [85] gezeigt wurde, macht die Gatterkomplexität der Zeiger den Großteil der Gesamtkomplexität des HDLIFO Moduls aus. Durch Verwendung einer HDLIFO kann man beim wichtigsten Flächenfaktor also knapp 50% der Flächenkomplexität gegenüber der Verwendung einer HFIFO einsparen.

2.3 Listenspeicher mit dynamischer Elementzuordnung

Neben der Einführung der HDLIFO stellt die Entwicklung eines verbesserten Speicherkonzepts für die Implementierung des LIFO-Speichers ein wichtiges

Ergebnis dieser Arbeit dar. Im Vergleich zu dem in Kapitel 1.4.4 vorgestellten Speicherkonzept, die hier als statische Elementzuordnung bezeichnet wird, wird hier ein Speicherkonzept mit dynamischer Zuordnung vorgeschlagen. Dabei wird der Speicher wie in [42] mit verketteten Listen implementiert. Im Unterschied zur statischen Bildpunktzuordnung muss bei der dynamischen Elementzuordnung der LIFO-Speicher aber nicht die gleiche Größe haben wie das Bild. Abbildung 2.14 zeigt, dass die Architektur mit dynamischer Elementzuordnung aus dem Top-Of-Stack-Zeiger, einem Zeiger auf eine Liste mit unbenutzten LIFO-Elementen, einer Steuereinheit zur Initialisierung der Liste mit unbenutzten LIFO-Elementen und dem LIFO-Speicher selber besteht.

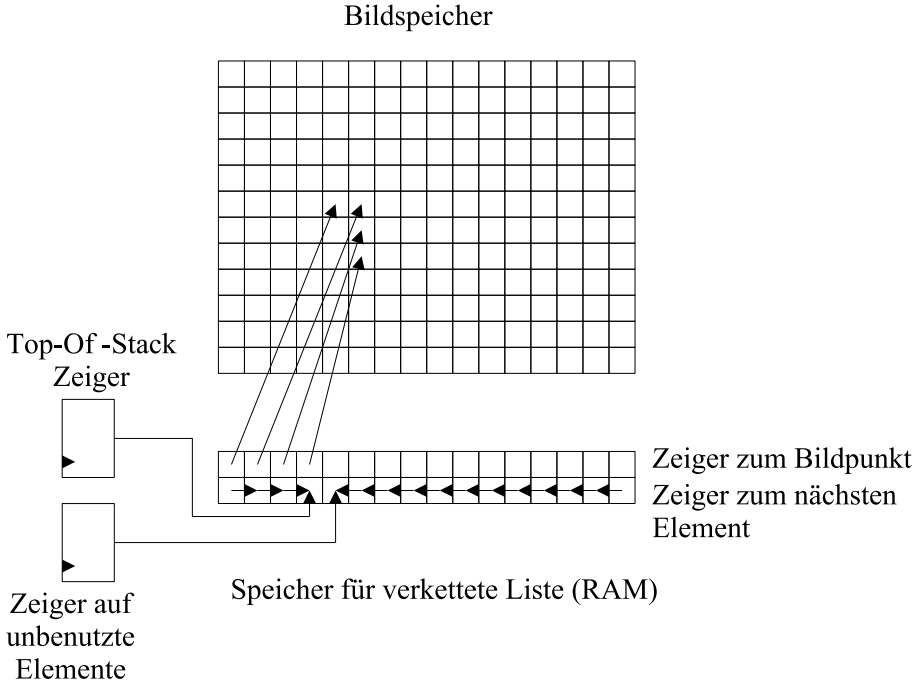


Abbildung 2.14: Konzept für den LIFO-Speicher mit dynamischer Zuordnung der Listenelemente. Der einfach verkettete Speicher kann eine andere Anzahl an Elementen haben als es Punkte im Bild gibt.

Jedes Element im LIFO-Speicher hat dabei jeweils einen Zeiger auf das vorhergehende Element der verketteten Liste und einen Zeiger auf den dazugehörigen Bildpunkt. Bei der statischen Zuordnung der Listenelemente hingegen ist der Zeiger auf den Bildpunkt nicht nötig, da der Bildpunkt die gleiche Adresse (Koordinaten) hat wie das Listenelement. Dieser Vorteil der

statischen Elementzuordnung wird aber durch eine andere Eigenschaft der dynamischen Elementzuordnung wieder kompensiert. Wie in Kapitel 2.1.4 bereits beschrieben, gibt es mögliche Konfliktfälle bei der Verwendung der HFIFO. Dieser Konfliktfall ist unabhängig von der Verwendung einer HFIFO oder HDLIFO und wird im Folgenden am Beispiel der HDLIFO erläutert. Der Konfliktfall tritt auf, wenn eine Bildpunkt, der bereits in einer LIFO eingetragen ist und dem somit ein Listenelement zugeordnet ist, mit einer höheren Priorität in eine andere LIFO eingetragen werden soll. Bei der statischen Bildpunktzuordnung gibt es genau ein Element im Listenspeicher, das aber bereits in einer Liste verwendet wird. Also muss der alte Eintrag aus der alten Liste ausgetragen und der neue Eintrag an die neue Liste angehängt werden. Abbildung 2.15 zeigt den Prozess des Austragens eines Elementes beim Neuzuordnen des Elementes in eine höher priorisierte LIFO.

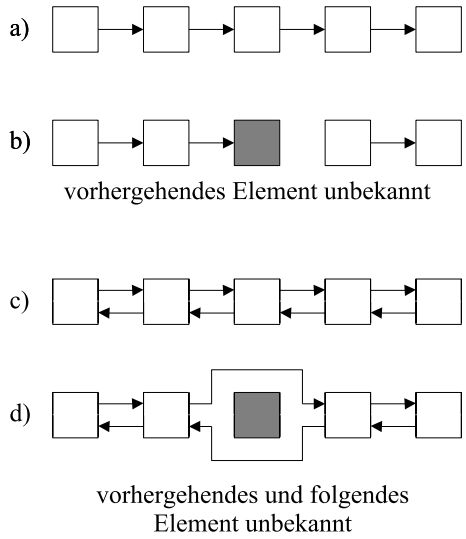


Abbildung 2.15: Austragen eines Elementes aus einer verketteten Liste: a) Zustand vor dem Austragen bei Verwendung einer einfach verketteten Liste, b) fehlerhafter Zustand nach dem Austragen des grau markierten Bildpunktes, c) Zustand vor dem Austragen bei Verwendung einer doppelt verketteten Liste, d) korrekter Zustand nach dem Austragen des grau markierten Bildpunktes.

Wenn, wie a) und b) zeigen, eine einfach verkettete Liste benutzt wird, dann lässt sich die Liste beim Austragen eines Listenelementes nicht fehlerfrei wiederherstellen. Der Zeiger vom vorhergehenden Element kann nicht auf das folgende Element gesetzt werden, da von dem auszutragenden Element keine Referenz zu dem vorhergehenden Element vorhanden ist. Es müsste also die

komplette Liste bis zu dem Element, das ausgetragen werden soll, durchsucht werden, um das vorhergehende Element zu identifizieren und den Zeiger auf das Folgeelement zu setzen. Eine andere Methode zur Lokalisierung des vorhergehenden Elementes ist die Verwendung einer doppelt verketteten Liste. Nun sind sowohl das vorhergehende also auch das folgende Element bekannt. Durch Neusetzen der Zeiger kann nun das Element fehlerfrei ausgetragen werden. Durch die Verwendung einer doppelt verketteten Liste muss jetzt aber der Speicher bei der statischen Elementzuordnung doppelt so groß sein.

Im Gegensatz dazu benötigt man bei der dynamischen Elementzuordnung nur eine einfach verkettete Liste. Da hier keine feste Zuordnung von Listenelement und Bildpunkt besteht, kann hier der Bildpunkt mehrmals in die LIFOs eingetragen werden. Beim Auslesen aus der LIFO muss dann nur noch überprüft werden, ob der Bildpunkt bereits nach dem Auslesen aus einer höher priorisierten LIFO verarbeitet wurde. Ist der Bildpunkt bereits als verarbeitet markiert, wird der Test des Homogenitätskriteriums für die Nachbarn und die Verarbeitung des Bildpunktes übersprungen und mit dem nächsten Bildpunkt fortgefahren. Auf diese Art wird zwar mehr LIFO-Speicher belegt, aber man spart Speicher da keine doppelte Verkettung benötigt wird. Für das wiederholte Auslesen des Bildpunktes wird zusätzliche Zeit benötigt, die allerdings auch beim Austragen eines Elementes aus einer LIFO benötigt wird. Da beim Austragen auf zwei unterschiedliche Elemente zugegriffen werden muss, ist hier der Zeitaufwand tendenziell größer als beim Auslesen eines Bildpunktes. Die dynamische Elementzuordnung kann man nicht nur auf LIFOs anwenden, sondern auch auf FIFOs.

Ein anderer Aspekt beim Vergleichen von statischer und dynamischer Zuordnung ist die Auslastung des LIFO-Speichers. Bei der statischen Elementzuordnung ist die Anzahl der Listenelemente identisch mit der Anzahl der Bildpunkte. In der LIFO sind aber immer nur die Bildpunkte der Ausbreitungsfront gleichzeitig eingetragen, so dass die Ausnutzung des Speichers relativ schlecht ist. Da sich nur die aktuelle Ausbreitungsfront in dem LIFO-Speicher befindet, braucht der Speicher auch nur die Größe des maximalen Füllstands zu haben. Der maximale Füllstand ist aber von dem Start- und Homogenitätskriterium, der Verarbeitungsfunktion, der Bildgröße sowie dem Bildinhalt abhängig. Es gibt Bildverarbeitungsfunktionen mit rekursiver Adressierung, deren Startkriterium alle Bildpunkte in die HDLIFO einträgt. Anschließend wird die Reihenfolge der Abarbeitung durch Änderung der Priorität des Bildpunktes festgelegt. Bildverarbeitungsfunktionen, die dies tun, sind z.B. Flattenings. In diesem Fall ist die benötigte Speichergröße gleich der *Worst-Case*-Größe, die der Bildgröße entspricht. Auf der anderen Seite verwendet aber nicht jeder Algorithmus solche Funktionen, so dass bei gleicher Speichergröße und der Verwendung einer dynamischen Elementzuordnung

auch größere Bilder verarbeitet werden können. Dies ist besonders bei einer Hardware-Implementierung von Bedeutung, da nach der Herstellung des Chips die Speichergröße nicht mehr verändert werden kann. Als Beschränkung ist nicht mehr die Bildgröße entscheidend, sondern nur, dass bei der Verarbeitung eines Bildes kein Speicherüberlauf auftritt. Der in Kapitel 1.6 vorgestellte Algorithmus kann mit einer LIFO-Größe von 64K Elementen[2] Bilder im CIF-Format (ca. 100K Bildpunkte) fehlerfrei bearbeiten. Die statische Elementzuordnung benötigt dafür hingegen mindestens die ca. 100K Elemente.

Ein letzter wichtiger Aspekt ist die bei der dynamischen Speicherzuordnung notwendige Verwaltung der unbenutzten Elemente, da jedes Listenelement zeitlich nacheinander mehrfach verwendet werden kann. Wenn ein Bildpunkt in den LIFO-Speicher eingetragen werden soll, wird ein Element aus der Liste mit den unbenutzten Elementen entnommen und an eine der LIFOs der HDLIFO angefügt. Beim Auslesen eines Elementes wird in den Top-Of-Stack-Zeiger der aktuellen LIFO der Verkettungszeiger des Top-Of-Stack-Elementes kopiert. Somit ist der Bildpunkt aus der LIFO ausgetragen. Das Element wird dann in die Liste mit den unbenutzten Elementen eingetragen, indem der Wert des Top-Of-Stack-Zeigers dieser Liste in den Verkettungszeiger des nun unbenutzten Elementes kopiert wird und der Top-Of-Stack-Zeiger auf die Adresse des Elementes zeigt. In [85] wurden verschiedene Abläufe zum Ein- und Austragen von Elementen untersucht und beschrieben. Abbildung 2.16 zeigt die Kopiervorgänge beim gleichzeitigen Ein- und Austragen von Elementen. In dem Beispiel sind die Lese- und Schreib-LIFOs verschieden. Bei diesem Ablauf wird nur auf zwei Speicherstellen gleichzeitig zugegriffen. Bei Verwendung eines Dual-Port-RAMs[3] kostet dann die Verwaltung der leeren Liste keine zusätzliche Zeit.

Am Anfang des Ausbreitungsprozesses muss der Zustand der LIFOs initialisiert werden. Insbesondere muss die Liste mit den Elementen angelegt werden. Um nicht im Speicher die Verkettung erzeugen zu müssen, können beim ersten Zuweisen der Elemente die unbenutzten Elemente durch Zählen bestimmt werden. Sind Elemente nach einem Auslesen wieder unbenutzt, werden sie an die Liste mit den unbenutzten Elementen angehängt. Erreicht der Zähler dann das letzte Listenelement, können die nächsten unbenutzten Elemente aus der nun aufgebauten Liste entnommen werden. Da die Verwendung des Zählers zur Bestimmung von unbenutzten Elementen einfacher ist als die Listenverwaltung, bietet es sich an, vor dem Aufruf einer

[2]Standardeinstellung bei der Software. In den 64K Elementen ist eine großzügige Sicherheit noch vorhanden.

[3]Ramdom-Access-Memory mit zwei Schnittstellen über die gleichzeitig auf den Speicher zugegriffen werden kann.

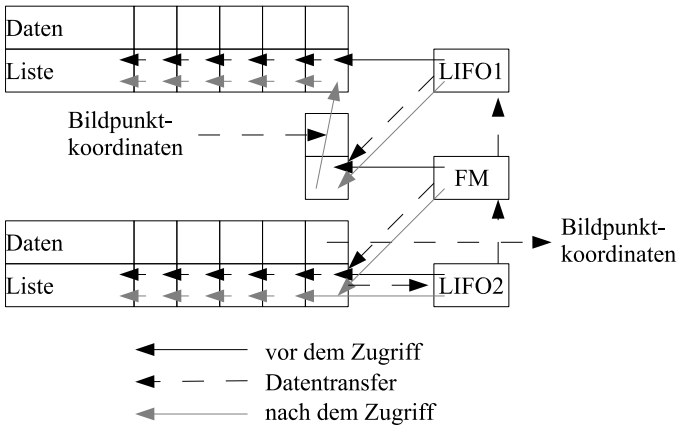


Abbildung 2.16: Gleichzeitiges Ein- und Austragen von Elementen bei Verwendung von LIFOs mit dynamischer Elementzuordnung. Gezeigt sind die Verkettungen und Datentransporte. Schwarze Pfeile stellen die Verkettung vor dem Zugriff dar, gestrichelte die Datentransporte und graue den Zustand nach dem Ein- und Austragen. An LIFO1 wird ein Element angehängt und aus LIFO2 ausgetragen. FM ist die Liste mit unbenutzten Elementen (Free Memory).

Bildverarbeitungsfunktion mit Ausbreitungsprozess die LIFO zu initialisieren. Dann wird in vielen Fällen ausschließlich der Zähler zur Bestimmung unbenutzter Elemente verwendet, vorausgesetzt die LIFO verfügt über ausreichend Elemente. In Abbildung 2.17 ist das gleichzeitige Eintragen von zwei Bildpunkten in unterschiedliche LIFOs und der Verwendung des Zählers für unbenutzte Elemente dargestellt. Aus Profilen, die in [1] durchgeführt wurden, kann man sehen, dass nur selten mehr als zwei Nachbarn in eine LIFO eingetragen werden, so dass die Architektur das gleichzeitige Eintragen von bis zu zwei Elementen in die LIFOs erlaubt.

2.4 Methode zum Abbilden der Funktionen auf das abstrakte Modell

Nachdem nun das abstrakte Modell für Bildverarbeitungsfunktionen beschrieben wurde, lässt sich eine Methode formulieren, mit der sich Bildverarbeitungsfunktionen auf das Modell abbilden lassen. Dabei legt das Modell fest, welche Bildverarbeitungsfunktionen mit dem Modell dargestellt werden können. Dies entspricht einer Methode für die Synthese von Bildverarbei-

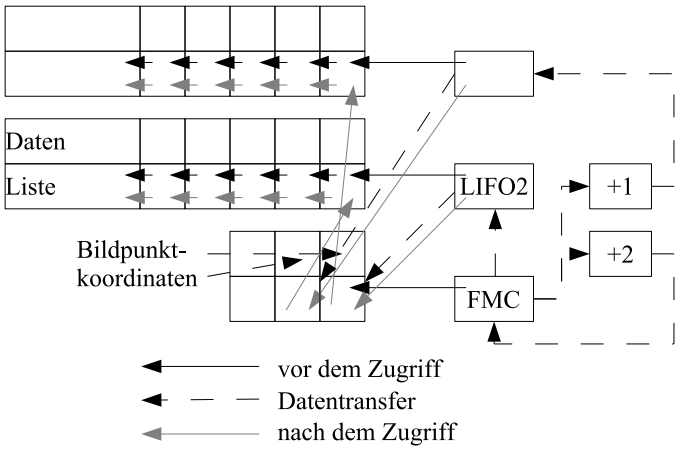


Abbildung 2.17: Gleichzeitiges Eintragen von zwei Elementen in unterschiedliche LIFOs. Unbenutzte Elemente werden durch den Zähler FMC bestimmt (Free Memory Counter). Schwarze Pfeile stellen die Verkettung vor dem Zugriff dar, gestrichelte den Datentransfer und graue den Endzustand.

tungsfunktionen. Der umgedrehte Fall, also die Abbildung einer Funktion auf das Modell, entspricht dann einer Analyse.

Hierbei wird zuerst identifiziert, welche Adressierungsklasse verwendet wird. Dadurch wird in Inter-, Intra-, und rekursive Adressierung unterteilt. Anschließend werden dann die nötigen Parameter der Adressierung bestimmt (siehe Tabelle 2.1), wie z.B. das Scan-Schema oder die Anzahl der Bildkanäle am Eingang bzw. Ausgang. Schließlich muss nur noch betrachtet werden, mit welcher Operation die einzelnen Bildpunkte verarbeitet werden. Mit diesen Informationen lässt sich eine Bildverarbeitungsfunktion so auf das Modell abbilden, dass die Synthese eindeutig ist. Das heißt, dass es unter Umständen mehrere Möglichkeiten gibt, wie eine Funktion abgebildet werden kann. Dies wird klar, wenn man sich vorstellt, dass einige Parameter bei bestimmten Bildverarbeitungsfunktionen nicht relevant sind, wie z.B. die Abarbeitungsfolge der Bildpunkte bei einer Gradientenberechnung. Werden diese Parameter festgelegt, lässt sich aus den Modellparametern eindeutig die Funktion bestimmen.

2.5 Software-Modell

Das abstrakte Modell für Bildverarbeitungsfunktionen ist mit der AddressLib als Software implementiert worden. Die AddressLib setzt dabei eine reduzier-

te VOP[4]-Struktur der MPEG−4 MoMuSys[5] Referenz Software voraus. Diese Datenstruktur hat 4 Bildkanäle, einen für die Luminanz, zwei für die Farbwerte und einen für die Segmentmaske. Für das Bild sind die Geometriedaten verfügbar. Die einzelnen Bildkanäle haben auch Geometrieinformationen, da z.B. bei einer 4:2:0 Farbauflösung die Farbkanäle nur die halbe Größe in X- und Y-Richtung haben. Jeder Bildkanal hat seinen eigenen Speicher, so dass die MoMuSys-Datenstruktur eine *non-interleaved* Datenstruktur ist. Die Software-Implementierung des Modells für Bildverarbeitungsfunktionen unterstützt dabei aber nur Bildkanäle mit gleicher Auflösung. Das heißt, dass bei Bildern mit z.B. 4:2:0 Farbauflösung entweder nur der U- und V-Kanal oder der Y- und Segmentmaskenkanal gleichzeitig verarbeitet werden können. Es bietet sich daher an, Bilder mit 4:4:4 Farbauflösung zu benutzen, damit alle Bildkanäle gleichzeitig verarbeitet werden können.

Jeder Bildkanalspeicher unterstützt Integer-Daten mit 8 Bit ohne Vorzeichen (*unsigned char*) oder 16 Bit mit Vorzeichen (*short*) sowie eine Fließkommadarstellung (*float*). Der gewählte Datentyp ist in der Kanaldatenstruktur abgespeichert und legt die Größe des Bildkanalspeichers fest.

Jede der drei Adressierungsmethoden ist mit einer eigenen Funktion implementiert. Für die Inter-Adressierung mit Koordinatentransformation und für die HDLIFO sind jeweils noch zusätzliche Funktionen definiert. Als Programmiersprache wurde dabei C verwendet. Im Folgenden sind die C−Prototypen der Funktionen dargestellt:

Inter-Adressierung

```
void inter_proc(
  MomVop *result, MomVop *input1, MomVop *input2,
  int reschannels,int in1channels,int in2channels, int scanmode,
  TCoor winx,TCoor winy,TCoor resposx,TCoor resposy,
  TCoor in1posx,TCoor in1posy,
  TCoor in2posx,TCoor in2posy,
  void (*operation)(TPixel *res, TPixel *in1, TPixel *in2));

void intertrans_proc(
  MomVop *result, MomVop *input1, MomVop *input2,
  int reschanels, int in1chanels, int in2chanels, int scanmode,
  TCoor winx, TCoor winy, TCoor resposx, TCoor resposy,
  TCoor in1posx, TCoor in1posy,
```

[4]Video-Objekt-Plane

[5]ACTS Projekt "Mobile Multimedia Systems" der Europäischen Union. Im Rahmen dieses Projektes wurde eines der MPEG−4 Software-Referenzsysteme erstellt.

```
  void in2trans(TCoor width, TCoor height,
  TCoor inx,TCoor iny, float *outx, float *outy),
  void (*operation)(TPixel *res, TPixel *in1, TPixel *in2));
```

Intra-Adressierung

```
void intra_proc(
  MomVop *resvop, MomVop *invop, MomImage *aux,
  int reschannels, int inchannels,
  int connect, int scanmode,
  TChan ahold, TChan yhold, TChan uhold, TChan vhold, TChan axhold,
  TCoor winx, TCoor winy, TCoor resposx, TCoor resposy,
  TCoor inposx,TCoor inposy, TSizeConv conv,
  void (*operation)(TPixel *, TPixel *, TCoor, TCoor));
```

rekursive Adressierung

```
void dlifo_proc(
  MomVop *result, MomVop *in1, MomImage *aux, MomImage *marker,
  int reschannels, int in1channels,
  int connect, int scanmode, int processmode,
  TChan ahold, TChan yhold, TChan uhold, TChan vhold, TChan axhold,
  int (*startcrit)(TPixel *in,TCoor x,TCoor y),
  int (*neighcrit)(TPixel *in,int pos,TCoor x,TCoor y),
  void (*operation)(TPixel *res, TPixel *in, TCoor x, TCoor y));

void hdlifo_proc(
  MomVop *resvop, MomVop *in1, MomImage *aux, MomImage *marker,
  int reschannels, int in1channels,
  int connect, int scanmode, int processmode,
  short startlevel, short stoplevel, short initfifo,
  TChan ahold, TChan yhold, TChan uhold, TChan vhold, TChan axhold,
  int (*startcrit)(TPixel *in, TCoor x, TCoor y),
  int (*neighcrit)(TPixel *in, int pos, TCoor x, TCoor y),
  void (*operation)(TPixel *res, TPixel *in, TCoor x, TCoor y));
```

Wie man in den Prototypendefinitionen erkennen kann, hat jede dieser Funktionen eine definierte Schnittstelle zu der(den) Verarbeitungsfunktion(en). Die Verarbeitungsfunktionen werden der Adressierungsfunktion bekannt gemacht, indem der Zeiger auf diese Funktionen an den jeweiligen Parametern übergeben wird. Die Inter-Adressierung hat eine Verarbeitungsfunktion mit einem Bildpunktwert für das Ergebnis und zwei für die beiden

Bildpunktwerte am Eingang. Die Bildpunkte werden dabei formal als Zeiger auf eine Struktur übergeben, die Werte für den Alpha-, Y-, U-, V- und Zusatzkanal zusammenführt. Jeder dieser Kanalwerte ist durch eine *Union* vom Type int oder float implementiert. Zusätzlich wird bei der Inter-Adressierung mit Koordinatentransformation noch der Zeiger auf die Funktion zur Berechnung der Koordinatentransformation, die auf das zweite Eingangsbild angewendet wird, angegeben.

Die Verarbeitungsfunktion für die Intra-Adressierung hat wie zuvor einen Zeiger auf die Ergebniswerte als auch einen Zeiger auf das Fenster der Eingangswerte. Dabei ist das Eingangsfenster immer als Feld mit 25 Werten angegeben. Wie in Abbildung 2.18 gezeigt ist, hat der zentrale Bildpunkt der Nachbarschaft immer den Indexwert 12. a) zeigt die Nummerierung der Bildpunkte im Fall der Verwendung eines zweidimensionalen Eingangsfensters. Dies hat dann eine Größe von 5x5 Bildpunkten. Ist die mit dem Parameter connect ausgewählte Nachbarschaft kleiner, sind die Werte in der Nachbarschaft nur an den ausgewählten Stellen definiert. Bei den eindimensionalen Nachbarschaften hat der zentrale Bildpunkt auch den Indexwert 12. Die Nachbarn sind dann aber linear von diesem Bildpunkt aus adressiert. Für diese Fälle sind die Nummerierungen in b) (horizontal) und c) (vertikal) gezeigt. Neben den Bildpunktwerten hat die Intra-Adressierung noch die Koordinaten des Ausgangsbildpunktes als Eingangsdaten.

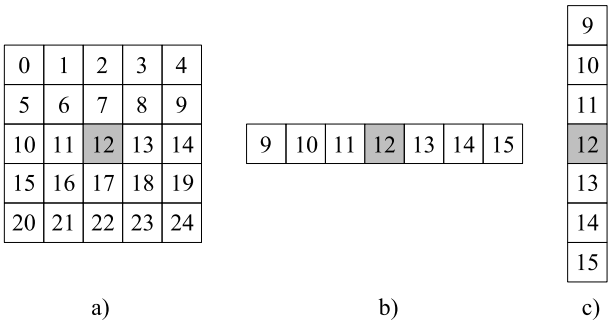


Abbildung 2.18: Nummerierung der Nachbarn des Eingangsfensters: a) für 2D und CON_0 Nachbarschaftssysteme, b) für horizontale und c) für vertikale Nachbarschaftssysteme. Der grau markierte Bildpunkt ist dabei immer der zentrale Eingangsbildpunkt und hat in jedem Fall den Indexwert 12.

Die rekursive Adressierung hat Zeiger für die drei Funktionen, Startkriterium, Nachbarschaftskriterium und Verarbeitungsfunktion. Die drei Funktionen benutzten als Eingang für die Bildpunktwerte einen Zeiger auf das Eingangsfenster. Die Nummerierung der Nachbarn ist die gleiche wie bei

der Intra-Adressierung. Das Start- und das Nachbarschaftskriterium liefern ihre Ergebnisse als Rückgabewert der Funktion. Das Nachbarschaftskriterium wird dabei für jeden Nachbarn des Nachbarschaftssystems aufgerufen. Die Verarbeitungsfunktion hat die gleiche Schnittstelle wie die der Intra-Adressierung.

Zusammenfassend sind alle Parameter und ihre möglichen Wertebereiche in Abhängigkeit der einzelnen Adressierungsfunktion in Tabelle 2.2 dargestellt.

Neben der Parametrisierung wird in diesem Kapitel noch auf die Struktur der Software-Implementierung eingegangen. Wie bereits beschrieben, verwendet das Modell für jede der Adressierungsfunktionen eine eigene C−Funktion. Außerdem haben die Inter-Adressierung mit Unterstützung für eine Koordinatentransformation und die HDLIFO-Adressierung jeweils eine eigene Funktion. Alle fünf Funktionen sind in ihrer Struktur ähnlich aufgebaut. Die unterschiedlichen Scan-Schemen sind aus Gründen der Optimierung jede für sich implementiert und werden durch eine *switch*-Anweisung ausgewählt. Innerhalb der Scan-Schemen sind dann die Schleifen für die Bearbeitung der Bildpunkte. Jede Iteration der Schleife stellt einen so genannten Bildpunktzyklus dar. Pro Bildpunktzyklus werden die Adressen zu den Bildpunktwerten durch Inkrementieren und Dekrementieren berechnet. Bei der rekursiven Adressierung werden die Adressen auch aus den Koordinaten, die aus der LIFO gelesen werden, berechnet. Dann werden die Bildpunktewerte in das Feld mit den 25 Bildpunkten des Eingangsfensters geladen. Da die verschiedenen Adressierungsfunktionen eine unterschiedliche Anzahl an Eingangsdaten laden müssen, wird das Laden der Eingangsdaten je nach Anzahl in der Software mit unterschiedlichen Funktionen oder Code-Segmenten implementiert. Dann werden die Eingangsdaten in der/den Verarbeitungsfunktion(en) verarbeitet. Schließlich werden die Ergebnisse in das Ergebnisbild geschrieben und bei der rekursiven Adressierung die Nachbarn in den LIFO-Speicher eingetragen. Alle diese Arbeitsschritte eines Bildpunktzyklus, im Folgenden auch Makroinstruktionen genannt, sind als Funktionen oder aber als zusammenhängende Code-Segmente implementiert. Tabelle 2.3 gibt eine vollständige Liste der in der AddressLib verwendeten Makroinstruktionen wieder. Dabei werden die Makroinstruktionen in Relation zu den Adressierungsfunktionen, die sie verwenden, gesetzt.

In Tabelle 2.3 ist zu sehen, dass es eine Reihe von Lade-Instruktionen gibt. Mit den Lade-Instruktionen werden die für die Verarbeitung benötigten Bildpunktdaten in einen Puffer, der so genannten Eingangsmatrix, geladen. Die Daten der Eingangsmatrix werden dann der Verarbeitungsfunktion übergeben.

Da die rekursive Adressierung nur sinnvoll bei 2D-Nachbarschaften ein-

Parameter	Parameter-name	Inter-Adres.	Intra-Adres.	rekursive Adres.
Kanäle	reschannels	α, Y, U, V	α, Y, U, V	α, Y, U, V
	inchannels	-	α, Y, U, V	α, Y, U, V
	in1channels	α, Y, U, V	-	-
	in2channels	α, Y, U, V	-	-
Zusatzkanal	aux	-	X	X
Statuskanal	marker	-	-	X
Nachbar-schaften	connect	-	CON_0-24, CON_H2-24, CON_V2-24	CON_4, CON_8
Scanschema	scanmode	horizontal, vertikal, hor.-Mäan., ver.-Mäan.,	horizontal, vertikal, hor.-Mäan., ver.-Mäan.,	horizontal, vertikal, hor.-Mäan., ver.-Mäan., Liste,
zusätzliche (oder- Ver-knüpfung)		Transform.-interpol.: keine, lin. int, lin. float,		Collected, Multi In,
Verarbeitungs-modell	processmode	-	-	Process-on-queue-in / queue-out /
Startpunkt-liste	list	-	-	X
	listsize	-	-	X
Hierarchie-ebenen	startlevel	-	-	0-stoplevel, startlevel-255,
	stoplevel	-	-	
	initlifo	-	-	X
Randwert-ersetzung	*hold	-	konstant, Halteglied	konstant, Halteglied
Koordinaten-bereich	winx/y,	X	X	-
	resposx/y,	X	X	-
	inx/y	in1x/y	X	-
		in2x/y	X	-
Skalierung	conv	-	X	-
Koordinaten-transform.	in2trans	X	-	-

Parameter	Parameter-name	Inter-Adres.	Intra-Adres.	rekursive Adres.
Verarbeitungsfunk.	startcrit neighcrit operation	- - X	- - X	X X X
Datentypen	-	unsigned char, short, float	unsigned char, short, float	unsigned char, short, float
indizierte Adress.	-	beliebig	beliebig	beliebig

Tabelle 2.2: Wertebereiche der Parameter bei der Software-Implementierung

MI	Funktion	Adressierungsmethoden
ad	Berechne Bildpunkt Adresse	alle
qo	Lese Aus der LIFO (queue-out)	rekursive Adressierung
ld()	Lade Nachbarschaft	Intra-, rekursive Adressierung
ld_2	Lade 2 Bildpunkte	Inter-Adressierung
shift	Schiebe Eingangsmatrix	Intra-Adressierung
preld()	Lade fehlende Punkte nach	Intra-Adressierung
ldix	Lade indizierte Daten	in Verarbeitungsfkt.
proc	Verarbeite Daten	alle
st	Speicher Ergebnis	alle
stn()	Speicher Ergebnis für Nachbar-bildpunkte (Process-on-queue-in)	rekursive Adressierung
stix	Speicher indizierte Daten	in Verarbeitungsfkt.
qi()	Schreibe Bildpunkte in die LIFO (queue-in)	rekursive Adressierung

Tabelle 2.3: Makroinstruktionen (MI), die in der AddressLib verwendet werden

gesetzt wird und wegen der wahlfreien Positionierung der Eingangsdaten, werden hier die Lade-Instruktionen *ld(8)* und *ld(4)* eingesetzt, die eine komplette Nachbarschaft aus dem Bildspeicher in die Eingangsmatrix laden.

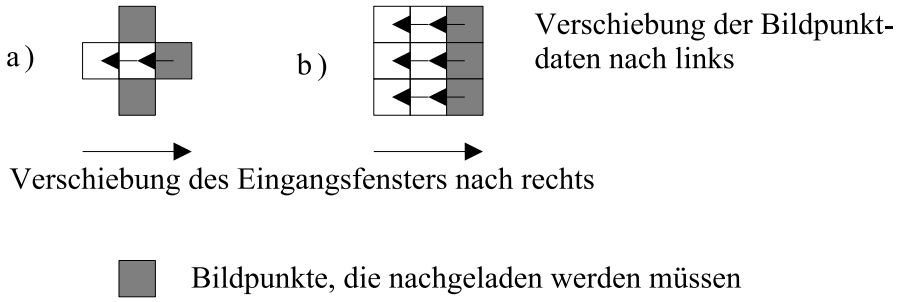


Abbildung 2.19: Nachladen der Bildpunkte der Eingangsmatrix am Beispiel einer Verschiebung nach links.
a) CON_4-Nachbarschaft: die zu ladenden Bildpunkte (grau) liegen versetzt und
b) CON_8 Nachbarschaft: die zu ladenden Bildpunkte (grau) liegen in einer Spalte (die X-Koordinate ist konstant)

Im Fall der Intra-Adressierung (bzw. der Suche nach Startpunkten für die rekursive Adressierung, die einer Intra Adressierung entspricht) werden am Zeilenanfang, bzw. beim Einsatz der mäanderförmigen Abarbeitung am Bildanfang, alle Daten der Eingangsmatrix geladen. Hierfür wird die Makroinstruktion *ld(8)* (*ld(25)*) verwendet. In den folgenden Bildpunktzyklen werden die Daten mit der Makroinstruktion *shift* in der Eingangsmatrix verschoben und eine Spalte bzw. Zeile der Eingangsmatrix mit dem Befehl *preld(v3)* (*preld(v5)*) oder *preld(h3)* (*preld(h5)*) nachgeladen. Wie im Beispiel mit der CON_8 und CON_4 Nachbarschaft in Bild 2.19 gezeigt wurde, werden beim Nachladen der Bildpunkte in der Matrix immer drei (5) Bildpunkte nachgeladen. Da das Nachladen einer Zeile bzw. Spalte bei der CON_8 Nachbarschaft einfacher ist, findet das Laden einer CON_4-Nachbarschaft (Fall a) in Bild 2.19) keine Verwendung, selbst wenn eine CON_4-Nachbarschaft in der Verarbeitung eingesetzt wird.

Bei der Verwendung von eindimensionalen Nachbarschaften werden die Makroinstruktionen *ld(h2)*, *ld(h4)*, *ld(h6)*, ... bis maximal *ld(h24)*, und *ld(v2)*, *ld(v4)*, *ld(v6)*, ... bis *ld(v24)* benutzt, um die Eingangsdaten am Anfang der Zeile oder des Bildes in die Matrix zu laden. Außerdem werden diese Lade-Instruktionen verwendet, wenn bei den eindimensionalen Nachbarschaften die Matrix orthogonal zur ihrer Ausrichtung verschoben wird, also wenn eine horizontale Nachbarschaft vertikal verschoben wird oder umgedreht. Wird

die eindimensionale Nachbarschaft in Richtung ihrer Ausrichtung verschoben, wird nur ein Bildpunkt nachgeladen. Hierfür wird die Makroinstruktion *preld(1)* verwendet. Dieser Befehl wird auch für das Nachladen des Bildpunktes bei einer CON_0 Nachbarschaft benutzt.

Mit der Lade-Instruktion *ld_2* werden die beiden benötigten Eingangsbildpunkte der Inter-Adressierung geladen.

Daneben werden noch weitere Makroinstruktionen zum Adressieren, Laden und Speicher benötigt. Mit *ldix* und *stix* werden indizierte Daten für die Verarbeitung geladen und gespeichert. Das Laden beinhaltet dabei auch das Adressieren der Daten, wofür auch die Bildpunktdaten selber benötigt werden. Mit *st* werden die Ergebnisdaten für den Ergebnisbildpunkt in den Speicher geschrieben. Für den Fall, dass eine rekursive Adressierung mit Process-on-queue-in-Modus verwendet wird, wird das Bildpunktergebnis für die Nachbarbildpunkte mit der *stn()*-Instruktion gespeichert. Mit dem Parameter werden die gültigen Nachbarn angegeben, für die das Ergebnis gespeichert werden soll. Diese Nachbarn werden außerdem mit *qi()* in die LIFO eingetragen. Mit dem *qo*-Befehl wird dann bei der rekursiven Adressierung der nächste zu verarbeitenden Bildpunkt aus der LIFO wieder ausgelesen. Analog dazu wird bei regulären Adressierungsfunktionen (Inter-, Intra-Adressierung und Start-Scan) die Makroinstruktion *ad* verwendet, um die Bildpunktadresse zu berechnen.

Außer der Makroinstruktion *proc* (siehe Tabelle 2.3) werden alle Makroinstruktionen für das Laden und Speichern der Bildpunktdaten bzw. für die Berechnung der Bildpunktadressen verwendet. Auf der einen Seite unterstreicht dies die Bedeutung der zentralen Realisierung der Speicherzugriffsfunktionen. Auf der anderen Seite sind aber die Verarbeitungsfunktionen nicht Bestandteil der AddressLib, so dass die Makroinstruktion *proc* auch nur die Schnittstelle zur Verarbeitung darstellt.

2.6 Eine Methode zum Entwurf von Software-Implementierungen der Funktion

Wie aus der Beschreibung des Software-Modells hervorgeht, sind das abstrakte Modell und das Software-Modell weitgehend identisch. Der wesentliche Unterschied zwischen den Modellen besteht in den Einschränkungen, dem das Software-Modell gegenüber dem abstrakten Modell unterliegt.

Als Konsequenz können zur Abbildung einer Bildverarbeitungsfunktion auf das Software-Modell die Modellparameter des abstrakten Modells ver-

wendet werden. Durch die Einschränkungen im Software-Modell lassen sich nun aber nicht mehr alle beliebigen Funktionen abbilden. In diesem Fall muss überprüft werden, ob und wie sich eine Bildverarbeitungsfunktion so separieren lässt, dass die Randbedingungen durch die Einschränkungen nicht mehr verletzt werden. Beispiele hierfür sind die getrennte Verarbeitung von Bildkanälen, wenn die Anzahl der benötigten Kanäle die Anzahl der vorhandenen Kanäle übersteigt oder die Separation von komplexen Filtern in mehrere einfache.

Neben der Parametrisierung der Adressierungsfunktion wird dann noch die Funktion zur Verarbeitung eines Bildpunktes benötigt. Diese verwendet die in der Prototypendefinition gegebene Schnittstelle passend zur Adressierungsfunktion. Dann wird diese Verarbeitungsfunktion durch ihren Zeiger als Parameter im Aufruf der Adressierungsfunktion referenziert. Ein Beispiel dafür ist in Abschnitt 3.1.2 gegeben.

2.7 Hardware-Modell

Neben dem Software-Modell, der AddressLib, wird für das abstrakte Verarbeitungsmodell in dieser Arbeit noch ein Konzept für eine Hardware-Implementierung, auch als AddressEngine bezeichnet, vorgeschlagen. Zu beachten ist, dass die AddressEngine ein Hardware-Modell für die Bildverarbeitungsfunktionen ist und keine konkrete Implementierung. Das heißt, dass die AddressEngine eine Grundarchitektur darstellt, die dann durch Optimierung in reale Implementierungen weiterentwickelt werden kann. Dies ist mit dem Conian [1] gemacht worden. Das abstrakte Hardware-Modell der AddressEngine erlaubt es, Merkmale für die Bestimmung der Komplexität von Algorithmen unabhängig von Optimierungen zu definieren.

Dabei ist das Hardware-Modell im Wesentlichen kompatibel zum Software-Modell und unterliegt nur folgenden Einschränkungen:

- Speicherorganisation: Die für die Algorithmenentwicklung optimierte Speicherorganisation mit unabhängigen Speicherbereichen pro Bildkanal (yyy..., uuu..., vvv...) ist nicht für die Ausführungsgeschwindigkeit optimiert. Deshalb wird vorgeschlagen, dass für eine Hardware-Implementierung ein überlappendes (*interleaved*) Speichersystem (yuv, yuv, yuv, ...) verwendet wird.

- Datentypen: In einer Hardware-Implementierung lassen sich keine Ressourcen dynamisch hinzufügen. Deshalb wird ein Datentyp dediziert für die Bildpunktdaten (pro Kanal) implementiert. Für Y-, U-, und V-Kanal werden die typischen 8 Bit pro Kanal vorgeschlagen, der Alpha-

und Zusatzkanal mit 16 Bit (zur Unterstützung von mehr als 256 Segmentnummern im Alpha-Kanal bzw. Distanzwerten > 255 im Zusatzkanal) und für den Marker-Kanal werden 2 Bit (für die 4 Zustände *ungültig*, *noch_nicht_verarbeitet*, *in_der_Queue* und *verarbeitet*) benötigt. In der Summe werden dann 58 Bit pro Bildpunkt benötigt.

- Größe des Eingangsfensters: bei 58 Bit pro Bildpunkt und 25 Bildpunkten würden 1450 Register plus Leitung zum Verdrahten mit der Verarbeitungseinheit benötigt werden. Um dies in praktikablen Grenzen zu halten wird das Eingangsfenster auf 3x3 bzw. 9x1 oder 1x9 Bildpunkte beschränkt.

- Verarbeitungseinheit: Die Verarbeitungseinheit kann nicht wie beim Software-Modell außerhalb des Modells implementiert werden, sondern muss Teil der Hardware-Architektur sein. Die Verarbeitungsfunktionen sind somit hinsichtlich der möglichen Flexibilität der Verarbeitungseinheit eingeschränkt.

- Indizierte Adressierung: Laden und Speichern indizierter Daten sind Teile der Verarbeitungsfunktion und somit auch eingeschränkt. Es wird vorgeschlagen eine zweifache Indizierung zu ermöglichen.

Unter Berücksichtigung der vorgeschlagenen Einschränkungen kann der in Kapitel 1.6 vorgestellte Algorithmus mit der AddressEngine implementiert werden.

Abbildung 2.20 zeigt das Blockdiagramm und Abbildung 2.21 das *Pipeline*-Modell der AddressEngine. Wie zu sehen ist, verfügt die AddressEngine über ein Modul für die Adressierung, das zum Laden und Speichern der Bildpunktwerte benötigt wird. Dieses Modul wird dediziert implementiert, da sich hiermit die bestmögliche Anpassung und Optimierung an das Anwendungsgebiet erreichen lässt. Dediziert implementiert sind auch das Steuerwerk und die HDLIFO. Außerdem gibt es die Registermatrix und das Ergebnisregister, die die Schnittstelle zur Verarbeitungseinheit darstellen. Die Verarbeitungseinheit muss, wie oben erwähnt, eine gewisse Flexibilität zur Unterstützung von unterschiedlichen Verarbeitungsfunktionen zur Verfügung stellen.

Zur Implementierung der flexiblen Verarbeitungseinheit gibt es mehrere Möglichkeiten. Die erste Variante wäre die Integration eines programmierbaren Prozessors. Eine solche Variante könnte die Eingangsdaten aus der Registermatrix laden, verarbeiten und das Ergebnis im Ergebnisregister speichern. Die nötige Flexibilität bei nur einer ALU (*Arithmetic-Logic-Unit*) wird hier durch das Speichern von Zwischenergebnissen in einem lokalen Speicher

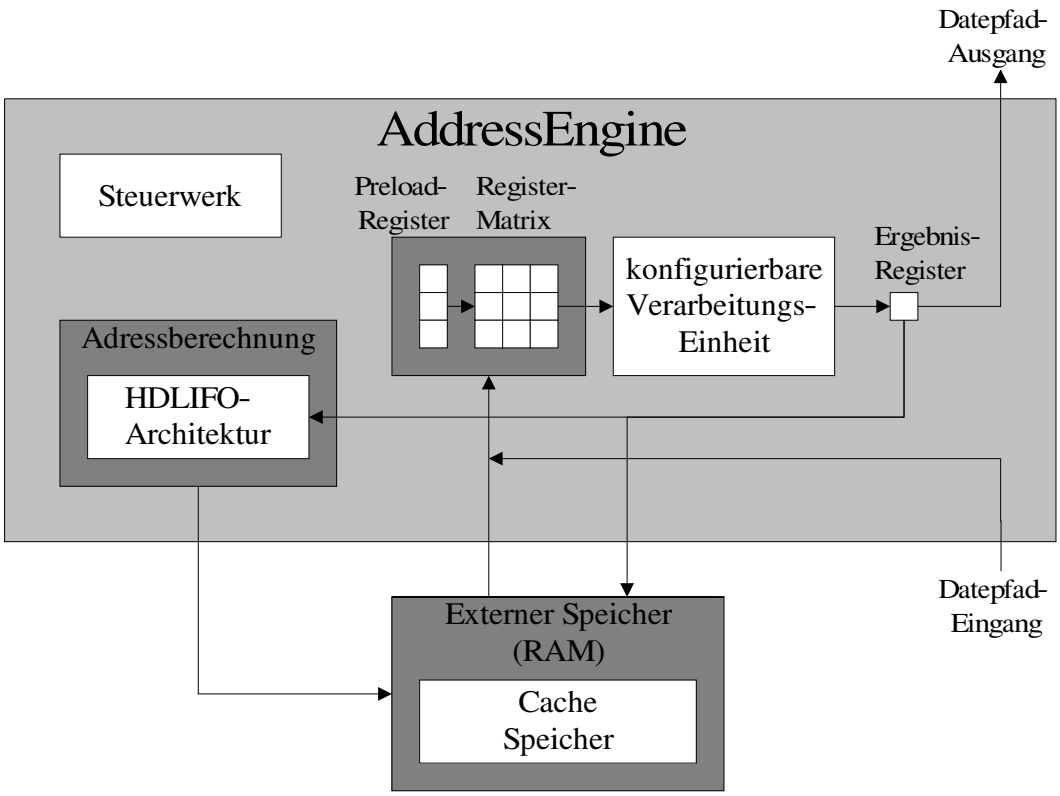


Abbildung 2.20: Blockschaltbild der AddressEngine

(RAM) und dem sequentiellen Abarbeiten realisiert. Bei einer solchen Implementierung wird in dem Prozessor für jeden Bildpunktzyklus die komplette Befehlsliste geladen und abgearbeitet. Dies ist bei der Verarbeitung eines Bildes mit CIF-Auflösung ca. 100.000mal der Fall. Außerdem wird auch nur die Adressierung beschleunigt. Die Verarbeitung wäre, ein Prozessor mit vergleichbarer Leistung vorausgesetzt, genauso schnell wie bei der Implementierung mit dem Software-Modell.

Eine weitere Variante wäre die Integration eines FPGA-Blocks. Wie im Kapitel 1.4.6 beschrieben wurde, hat diese Variante den Nachteil, dass die erreichbaren Taktfrequenzen sehr begrenzt sind. Dies liegt an der sehr hohen Granularität der Konfigurierbarkeit. Konkret heißt das, dass die Logikeinheiten auf der Ebene von einzelnen Gattern und Bits konfiguriert werden können. Man erreicht dadurch auf der anderen Seite eine sehr hohe Flexibilität, die nur durch die Größe des FPGA-Blocks begrenzt ist. Allerdings wird eine derartige Flexibilität nicht benötigt [86]. Zur Verarbeitungszeit muss dann noch die Zeit für das Konfigurieren des FPGAs hinzugezählt werden.

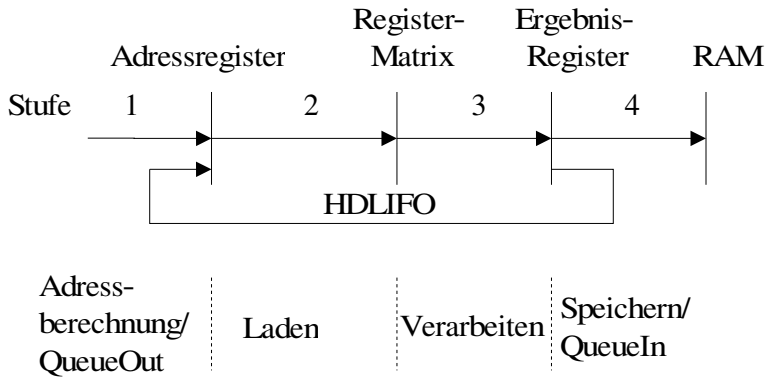


Abbildung 2.21: Pipeline-Modell der AddressEngine

Eine dritte Variante wäre eine unabhängige Implementierung aller von einem Algorithmus eingesetzten Verarbeitungsfunktionen. Diese Variante wurde z.B. beim PIMM1 umgesetzt. Hier wird jede Verarbeitungsfunktion mit einem eigenen Block implementiert. Durch die Konfiguration wird dann eine der Verarbeitungsfunktionen ausgewählt. Diese Variante ist eine Lösung für einen einzigen speziellen Algorithmus. Andere Algorithmen, die andere Bildverarbeitungsfunktionen verwenden, lassen sich nicht mit dieser Architektur implementieren. Andererseits sind diese Verarbeitungsfunktionen dann ohne zusätzlichen Overhead implementiert, da die Blöcke keine generischen Komponenten haben.

Ein Kompromiss aus der zweiten und dritten Variante ist eine Architektur, die elementare Verarbeitungsschritte zur Verfügung stellt. Diese können dann über die Konfiguration in weiten Grenzen miteinander kombiniert werden. Elementar ist in diesem Zusammenhang eine Teiloperation, die im Allgemeinen als atomarer Bestandteil in den Verarbeitungsfunktionen vorkommt und sich mit anderen elementaren Blöcken zur Verarbeitungsfunktion kombinieren lässt. Im Vergleich zum FPGA-Block sind diese Blöcke deutlich größer. So werden z.B. häufig Addiererbäume (lineare Filter) oder Sortiererbäume (Rangordnungsfilter) eingesetzt. Diese Blöcke werden dann dediziert implementiert. Die Bitbreite der arithmetischen und logischen Verarbeitungsschritte ist festgelegt. Da diese Blöcke monolithisch implementiert sind, sind diese auch deutlich schneller. Im Gegensatz zu den dedizierten Verarbeitungsfunktionen sind im begrenzten Maße generische Komponenten für die Verknüpfung der elementaren Arbeitsschritte nötig. Die Ausführungszeit wird daher geringfügig länger sein. Andererseits lassen sich nun aber auch neue Verarbeitungsfunktionen implementieren, die sich aus Kombinationen der elementaren Blöcke erstellen lassen. In [86] ist für das AddressEngine-

Konzept unter Berücksichtigung des in dieser Arbeit ausgewählten Algorithmus eine Verarbeitungseinheit entworfen worden.

Das Konzept der AddressEngine hat neben der Schnittstelle zum zentralen Bildspeicher noch einen Datenpfadeingang und -ausgang, um mehrere AddressEngines zusammenzuschalten. Durch das Benutzen der Datenpfadschnittstellen werden die Zugriffe auf die Bilddaten entzerrt und durch die sonst nötige Aufsummierung der Speicherbandbreiten der Engpass zum Speicher vermieden.

2.8 Parallelisierung im Hardware-Modell im Vergleich zum Software-Modell

Durch die Implementierung als Hardware-Modell kann gegenüber dem Software-Modell eine deutliche Beschleunigung erreicht werden. Diese Beschleunigung wird durch Parallelisierung erzielt. In diesem Kapitel sollen Möglichkeiten gezeigt werden, wie Parallelisierung benutzt werden kann, um Algorithmen beim Einsatz des Hardware-Modells zu beschleunigen.

Parallelisierung kann auf unterschiedlichen Ebenen durchgeführt werden. Diese Konzepte wurden bereits in Kapitel 1.4.6 beschrieben. Die unterste Ebene ist die Gatterebene. Auf ihr werden die Verzögerungszeiten der einzelnen Logikberechnungen optimiert. Ein Beispiel hierfür wäre die parallele Berechnung des Übertrags in einem *Carry-Lock-Ahead*-Addierer, die in diesem Fall von Entwurfwerkzeugen automatisch durchgeführt wird und somit auf jeder Standardarchitektur zu finden ist. Nur auf massiv parallelen Systemen werden oft wegen der Vielzahl an Verarbeitungseinheiten diese so einfach wie möglich implementiert.

Eine Parallelisierung, die nur bedingt im Software-Modell vorhanden ist, das auf einem programmierbaren Standardprozessor implementiert ist, ist die Integration von mehreren Verarbeitungseinheiten. Dadurch können Teilergebnisse einer komplexen Verarbeitungsfunktion parallel berechnet werden, sofern diese unabhängig voneinander berechnet werden können. Ein Beispiel für diese *Ressource*-Parallelisierung ist die Implementierung eines Addiererbaums bei dem viele Additionen gleichzeitig durchgeführt werden. Eine weitere wichtige Parallelisierung auf dieser Ebene ist die gleichzeitige Verarbeitung der Bildkanäle.

Im Software-Modell werden die einzelnen Bildpunktzyklen nacheinander ausgeführt. Dabei werden die Maschinenbefehle des Bildpunktzyklus nacheinander abgearbeitet. Parallelisierung findet hier in Form von Fließbandverarbeitung statt. Dabei werden die Maschinenbefehle überlappend abge-

arbeitet, wodurch der Durchsatz von Maschinenbefehlen auf in der Regel einen Befehl pro Takt erhöht wird. Bei der im Hardware-Modell eingesetzten Fließbandverarbeitung werden analog dazu Bildpunktzyklen in den einzelnen Stufen des Fließbandes überlappend verarbeitet. Dabei werden die unterschiedlichen Makroinstruktionen eines Bildpunktzyklus in jeweils einer Station des Fließbandes ausgeführt. Dadurch werden mehrere Makroinstruktionen gleichzeitig ausgeführt. Im Vergleich zu den Maschinenbefehlen besteht im Software-Modell jede Makroinstruktion aus mehreren Maschinenbefehlen, wobei eine Makroinstruktion im Hardware-Modell wenige Taktzyklen benötigt. Die Anzahl der Taktzyklen hängt dabei von der o.g. Gatter- und Ressource-Parallelisierung ab. Diese Art von Fließbandverarbeitung entspricht in etwa der Parallelisierung in einer VLIW-Architektur, wobei die dort erreichbare Flexibilität durch die Monotonität der Bildpunktzyklen nicht benötigt wird. Für die Fließbandverarbeitung im Hardware-Modell bietet sich folgende Zuordnung von Makroinstruktionen und Fließbandstufen an (siehe Abbildung 2.21):

1. Stufe: Berechnen der Bildpunktadresse bzw. Auslesen der Adresse aus der LIFO
2. Stufe: Laden der Eingangsdaten in die Registermatrix
3. Stufe: Laden der Indexdaten und Verarbeitung der Eingangsdaten
4. Stufe: Speichern des Ergebnisses und ggf. Eintragen der Nachbarn in die LIFO

Auf der nächsten Parallelisierungsebene werden Bildpunkte, die voneinander unabhängig verarbeitet werden können, gleichzeitig verarbeitet. Dieses Konzept entspricht einer SIMD Architektur. Virtuell könnte diese Parallelisierungsebene im Software-Modell durch parallele *Threads* in einer *Multi-Tasking*-Umgebung erreicht werden, aber faktisch bleibt die Sequentialität auf einem Standardprozessor erhalten. Deshalb unterstützt das Software-Modell auch keine Parallelisierung auf dieser Ebene. Auf der Seite des Hardware-Modells ist die Unabhängigkeit der Eingangsdaten von großer Bedeutung. Die Unabhängigkeit ist aber stark von der gewählten Adressierungsvariante abhängig. So sind die Eingangsdaten bei der Inter-Adressierung prinzipiell voneinander unabhängig und können somit gleichzeitig verarbeitet werden. Das gleiche gilt für die Intra-Adressierung mit nicht-rekursiven Verarbeitungsfunktionen. Beim Einsatz von rekursiver Verarbeitung bestehen naturbedingt Abhängigkeiten zwischen den Bildpunkten. Hier ist eine Parallelisierung auf der Bildpunktebene nicht möglich. Bei der rekursiven Adressierung gibt es auch naturbedingt Abhängigkeiten zwischen den

benachbarten Bildpunkten. Allerdings bestehen die Abhängigkeiten nur in der Ausbreitungsrichtung. Bildpunkte auf einer Distanzebene sind unabhängig voneinander und können wiederum gleichzeitig verarbeitet werden. Eine Möglichkeit der Umsetzung der Parallelisierung auf Bildpunktebene wäre eine Architektur mit unabhängigen Datenpfaden, die eine gemeinsame LIFO benutzen. Unbeachtet bleibt dabei das Problem, dass bei der rekursiven Adressierung immer eine komplette Nachbarschaft an Bildpunkten geladen werden muss und somit die Speicherbandbreite im Vergleich zu den anderen Adressierungsmethoden maximal ist. Unter der Annahme, dass die Speicherbandbreite der begrenzende Faktor ist, lässt sich dann keine Steigerung der Ausführungsgeschwindigkeit durch eine gemeinsam genutzt LIFO erzielen.

Insgesamt gesehen wird für das Hardware-Modell keine Parallelisierung auf Bildpunktebene vorgeschlagen, da sie sich nicht allgemeingültig im Verarbeitungsmodell anwenden lässt. Ein weiteres Argument dagegen ist die Größe der Registermatrix und der dazugehörigen Schnittstelle zur Verarbeitungseinheit. Diese Größe ist bereits bei einer 3x3 Matrix mit 522 Bits kritisch.

In der obersten Ebene der Parallelisierung lassen sich unabhängige Teile des Algorithmus gleichzeitig im Sinne einer MIMD-Implementierung bearbeiten. Im Fall des Farbsegmentierungsalgorithmus sind diese Teile die verschiedenen Bildverarbeitungsfunktionen. Wird eine Software-Implementierung eingesetzt, gilt das gleiche wie bei der Parallelisierung auf Bildpunktebene. Wird mit nur einem Prozessor gearbeitet, können die Bildverarbeitungsfunktionen in einzelnen *Threads* laufen, wobei eine Beschleunigung nicht erreicht wird. Durch den Einsatz eines Multi-Prozessorsystems könnten dann die Threads auf unterschiedlichen Prozessoren laufen und die Ausführzeit verringert werden. Die Beschleunigung wird dann durch den Steuer- und Synchronisierungsaufwand etwas reduziert, so dass die Beschleunigung nicht linear mit der Anzahl der Prozessoren skaliert. Außerdem ist durch Datenabhängigkeiten die sinnvolle, maximale Anzahl der Prozessoren begrenzt. Bei auftretenden Datenabhängigkeiten können aber die Prozessoren überlappend arbeiten, so dass bereits Daten weiterverarbeitet werden können, wenn schon alle nötigen Eingangsdaten nach der Verarbeitung in der vorhergehenden Bildverarbeitungsfunktion verfügbar sind. Da diese Ebene der Parallelisierung außerhalb des Verarbeitungsmodells liegt, sind im Wesentlichen die gleichen Aussagen für das Hardware-Modell zu treffen. Hier lassen sich aber noch spezielle Strukturen implementieren, die die Kommunikation in einem Multi-Prozessorsystem beschleunigen. Deshalb wird im Hardware-Modell auch die Kommunikation über den Datenpfadeingang und -ausgang vorgeschlagen. Abbildung 2.22 zeigt ein Beispiel für eine MIMD-Implementierung mit mehreren AddressEngines. Jeder Prozessor kann auf (Bild-)Daten im globalen Speicher zugreifen. Zur Entzerrung der Bandbreite an diesem zentralen Spei-

cher sind die AddressEngines noch zu einem Datenpfad zusammengeschaltet. Zur Implementierung des High-Level-Algorithmus (zur Kombination der Bildverarbeitungsfunktionen) ist noch ein zusätzlicher Standardprozessor im System vorhanden.

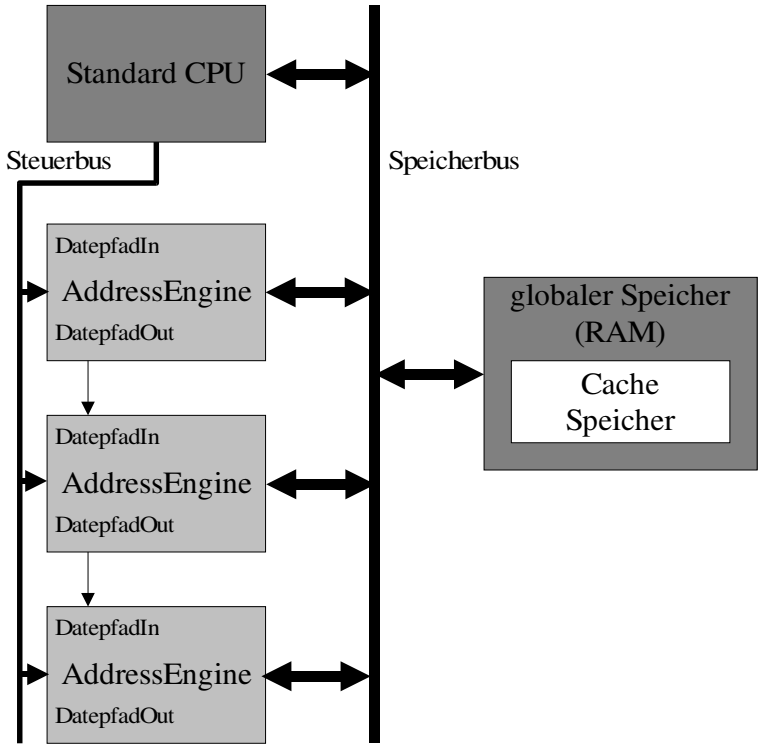


Abbildung 2.22: Blockschaltbild eines AddressEngine-MIMD-Systems: Mehrere AddressEngines können über den globalen Speicher kommunizieren oder Daten über die Datenpfadverkettung austauschen. Der High-Level-Algorithmus wird auf dem Standardprozessor ausgeführt.

Zum Erreichen von Skalierbarkeit könnten in einem solchen System auch die LIFOs zusammenwirken. Dann könnten auch Funktionen oder Bildformate bearbeitet werden, die sonst einen LIFO-Überlauf produzieren würden. Ebenso könnten die Verarbeitungseinheiten zusammengeschaltet werden, um eine komplexe Verarbeitungsfunktion in Teilfunktionen zerlegt abzuarbeiten. Diese Konzepte wurden aber im Rahmen dieser Arbeit bzw. weiterführenden Arbeiten aus Zeitgründen bisher nicht weiterverfolgt.

Die Parallelisierung auf der Ebene von Bildverarbeitungsfunktionen lässt sich auch einsetzen, um eine Parallelisierung auf Bildpunktebene zu implementieren. So ist es möglich, Datenunabhängigkeit vorausgesetzt, dass

Teilbereiche eines Bildes auf unterschiedlichen Prozessoren laufen. Obwohl dafür nur eine SIMD-Architektur nötig wäre, wird dann aber eine MIMD-Architektur verwendet.

Alles in Allem erlaubt das Hardware-Modell eine Reihe von Parallelisierungen, die eine Implementierung auf der AddressEngine gegenüber dem Softwaremodell deutlich beschleunigen. In [1] ist eine konkrete Implementierung der AddressEngine vorgeschlagen. Diese erreicht eine Beschleunigung um den Faktor 4,5, wobei diese Architektur mit 200 MHz mit einer Software-Implementierung auf einem Intel Pentium IV mit 2 GHz und MMX-Optimierung verglichen wurden.

2.9 Methode zum Analysieren der Komplexität

Im vorhergehenden Kapitel wurde das Software-Modell mit dem Hardware-Modell hinsichtlich der Parallelisierbarkeit verglichen. In diesem Kapitel wird die Struktur der beiden Implementierungsmodelle betrachtet und Aussagen abgeleitet, mit denen Ergebnisse eines *Profilings* der Software-Implementierung auf die Hardware-Implementierung übertragen werden können. Dies ist deshalb möglich, weil die Strukturen von Software- und Hardware-Modell weitgehend identisch sind. Diese bezieht sich vor allen Dingen auf die verwendeten Makroinstruktionen. Dadurch wird der Ablauf in beiden Modellen fast identisch. Tatsächlich kann die Software als ausführbare Spezifikation für den Hardware-Entwurf betrachtet werden. Trotz der großen Übereinstimmung gibt es folgende Unterschiede:

- Für die Ladeinstruktionen ld und preld sind im Hardware-Modell mehrere Implementierungen vorhanden, um unterschiedliche Nachbarschaftssysteme dediziert zu laden. Im Software-Modell erfolgt diese Unterscheidung durch Parametrisierung der Ladeinstruktionen.

- Die Reihenfolge, in der das Verschieben des Eingangsfensters und das Nachladen der neu hinzugekommenen Bildpunkte ausgeführt werden, ist bei der Software-Implementierung vorzugsweise anders, als bei der Hardware-Implementierung. Im Fall der Software wird das Verschieben zuerst durchgeführt, da dann für das Nachladen der Bildpunkte kein zusätzlicher Speicher benötigt wird. Prinzipiell gilt das gleiche für die Hardware. Hier lässt sich aber wegen der Fließbandverarbeitung der zeitliche Ablauf effizienter gestalten, wenn die neuen Bildpunkte erst in einem *Preload*-Register zwischengepuffert werden. Dies liegt

daran, dass die Registermatrix erst verändert werden darf, wenn die Daten des vorhergehenden Bildpunktzyklus verarbeitet wurden. Ohne Preload-Register könnte das Laden der nächsten Bildpunkte erst erfolgen, wenn die Verarbeitung der vorhergehenden Daten abgeschlossen ist. Dies würde eine zeitliche Parallelisierung verhindern. Die Reihenfolge, in der die Instruktionen aufgerufen werden, spielt aber für die Übertragung von Profilen vom Software-Modell auf das Hardware-Modell keine Rolle.

Durch die Übereinstimmung von Software- und Hardware-Modell ist der Ablauf des Algorithmus identisch. Es werden unter Berücksichtigung der genannten Ausnahmen die gleichen Makroinstruktionen in der selben Reihenfolge für die Verarbeitung ausgeführt. Dadurch ist ein Profil der Makroinstruktionen, das mit dem Software-Modell erzeugt wurde, auch für das Hardware-Modell gültig. Außerdem werden auch die Bildpunkte in der gleichen Reihenfolge geladen (unter Vernachlässigung des Bypassing der LIFO bei der rekursiven Adressierung). Jede Operation zum Laden von Bildpunkten aus dem Bildspeicher im Software-Modell entspricht einem Laden über die Speicherschnittstelle zum externen Speicher. Durch das Zählen der Zugriffe auf den Bildspeicher beim Ausführen der Software kann die benötigte Speicherbandbreite des Algorithmus für die Hardware-Implementierung gemessen werden. Genauso verhält es sich mit dem maximalen oder aktuellen Füllstand der LIFO. Aus den Füllständen lassen sich unter Berücksichtigung der Fließbandverarbeitung die nötige Speichergröße der LIFO als auch die Anzahl der Stillstände des Fließbandes ermitteln. Daneben kann durch ein Software-Profil ermittelt werden, wie oft ein, zwei oder mehrere Bildpunkte gleichzeitig in die LIFO geschrieben werden. Diese Informationen wurden in [1] ermittelt und in [85] für den Entwurf der HDLIFO-Architektur verwendet.

Kapitel 3

Ergebnisse

3.1 Verifikation des Modells

3.1.1 Implementierbarkeit des Algorithmus

Die drei Ebenen des Modells, das abstrakte Modell, das Software-Modell und das Hardware-Modell, sind in dieser Reihenfolge zunehmenden Einschränkungen unterworfen. Dies bedeutet, dass ein Algorithmus, der sich mit dem Software-Modell implementieren lässt, automatisch auch zum abstrakten Modell konform ist. Übersteigt der Algorithmus nicht die zusätzlichen Beschränkungen der Hardware-Architektur, lässt sich daraus auch automatisch die Implementierbarkeit auf der Hardware-Architektur ableiten. Soll ein Bildverarbeitungsalgorithmus entwickelt werden, der später mit einer Hardware-Implementierung beschleunigt werden soll (oder muss), so kann er unter Verwendung der AddressLib entworfen werden. Es kann nun bei der Entwicklung vorkommen, dass sich Bildverarbeitungsfunktionen nicht mit der AddressLib implementieren lassen. Dies kann zum einen an Inkompatibilität zum abstrakten Verarbeitungsmodell liegen, oder an den Einschränkungen des Software-Modells gegenüber dem abstrakten Modell. Für den ersten Fall wurde bisher eine Funktion (MPEG−4-Texture-Padding) identifiziert, die sich nicht auf das abstrakte Modell abbilden lässt. Dies lag an der Verletzung der Grundvoraussetzung, dass jeder Bildpunkt nur einmal pro Bildverarbeitungsfunktion bearbeitet wird. Der Algorithmenentwickler muss sich dann aber die Frage stellen, ob solche Funktionen wegen ihrer Irregularität noch für eine Implementierung geeignet sind und ob sich die Funktionalität auch mit Funktionen erreichen lässt, die zum abstrakten Modell konform sind. Auf der anderen Seite könnte auch das abstrakte Modell für diesen Sonderfall erweitert werden. Dafür müsste während des Scans die Bildpunktadresse bei einem bestimmten Ereignis auf einem Stapelspeicher (auch Stack oder LIFO genannt) gesichert

werden (push). Später, beim Auftreten des Rücksprungereignisses kann diese Adresse wieder vom Stapelspeicher geladen werden (pop) und der Scan an dieser Stelle erneut fortgefahren werden. Die Ereignisse müssten dann von der Verarbeitungsfunktion generiert werden. Da es sich hierbei aber um einen Einzelfall handelt und nicht vorhersehbar ist, welche Funktionen diese Funktionalität benutzen könnten, wird auf diese Erweiterung verzichtet. Bei einem zweiten Beispiel ist die AddressLib nicht geeignet, um eine optimierte Implementierung des RSST-Algorithmus (Seite 22) zu realisieren. Bei dieser Implementierung liegt keine bildbasierte Repräsentation der Bildpunktwerte vor. Die Bildpunkte der Regionen sind hier in verketteten Listen organisiert, so dass Regionen auf einfachste Weise durch verbinden der Listen zusammengeführt werden können [87]. Da es sich bei dieser Implementierung nicht um die Verarbeitung von Bildern handelt, sondern um die Verarbeitung von Listen, gehört diese Verarbeitungsfunktion auch nicht zum Anwendungsgebiet des Verarbeitungsmodells. Auf der anderen Seite könnte diese Methode auch das Modell erweitern, indem eine Listen-Adressierung als weitere Adressierungsmethode dem Modell hinzugeführt wird. Dies wurde aber im Rahmen dieser Arbeit nicht gemacht, da bisher keine weiteren Verarbeitungsfunktionen mit dieser Adressierungsvariante identifiziert wurden.

Neben möglichen Inkompatibilitäten zum Modell, können in manchen Fällen auch die Verarbeitungsfunktionen zu komplex sein, um mit dem Software- oder Hardware-Modell implementierbar zu sein. Ein Beispiel hierfür wäre die Verwendung von Filtern, die größere Nachbarschaften verwenden, als das Software- oder Hardware-Modell unterstützen. In diesem Fall kann unter Umständen die Begrenzung durch Separation des komplexen Filters in mehrere einfache überwunden werden. Ein anderes Beispiel wäre der Verzicht von Fließkommaverarbeitung in der Hardware-Implementierung. Hier lassen sich auch ganz einfach die Auswirkungen der Vereinfachung im Software-Modell ausprobieren, so dass Qualitätsverluste gegen den Implementierungsvorteil abgewogen werden können. Alternativ zu der Entscheidung, eine Vereinfachung durchzuführen, kann auch überprüft werden, ob eine Erweiterung der Möglichkeiten des Software- oder Hardware-Modells sinnvoll ist.

Im Rahmen dieser Arbeit wurde der in Kapitel 1.6 vorgestellte Algorithmus, der eine weite Abdeckung der im Kapitel 1.4.3 vorgestellten Algorithmen hat, mit dem Software-Modell erfolgreich implementiert. Durch Überprüfung der Bildverarbeitungsfunktionen des Algorithmus wurde festgestellt, dass sich der Algorithmus auch unter Berücksichtigung der im Hardware-Modell gemachten Einschränkungen mit der AddressEngine implementieren lässt. Dies begründet, dass das abstrakte Verarbeitungsmodell und die getroffenen Einschränkungen beim Software- und Hardware-Modell sinnvoll sind.

3.1.2 Effizienz der Implementierung und Spezifikation

Neben der Fragestellung, ob sich das abstrakte Verarbeitungsmodell auf einen sehr weiten Bereich von Bildverarbeitungsfunktionen anwenden lässt, ist auch die Effizienz, mit der diese Funktionen auf das Modell abgebildet und implementiert werden können, von großer Bedeutung. Die Effizienz soll anhand eines Beispiels demonstriert werden.

Bei dem Beispiel soll ein Wasserscheide-Algorithmus implementiert werden. Es wird davon ausgegangen, dass die Startpunkte für den Ausbreitungsprozess mit einer grafischen Benutzerschnittstelle manuell markiert wurden. Die markierten Startpunkte befinden sich dann in einer Liste, der Startpunktliste. Anschließend sollen von den markierten Bildpunkten aus durch den Ausbreitungsprozess des Wasserscheide-Algorithmus die Regionen erzeugt werden.

Zuerst wird die dafür nötige Adressierungsmethode identifiziert, die in diesem Fall die rekursive Adressierung mit einer HDLIFO und dem Connected-Modus ist. Die rekursive Adressierung benötigt drei Funktionen, das Startkriterium, das Nachbarschaftskriterium und die Verarbeitungsfunktion selber. Das Startkriterium ist einfach festzulegen, da es sich um die Punkte in der Startpunktliste handelt. Für alle diese Punkte trifft damit automatisch das Startkriterium zu. Wählt man als Scan-Schema für den Startpunkt-Scan den Listen-Scan aus, so muss das Startkriterium immer ein *Wahr* (Wert ungleich Null) zurückgeben. Das Nachbarschaftskriterium sortiert die Nachbarbildpunkte dann in die Hierarchiestufen ein, die dem Gradienten des Nachbarbildpunktes entsprechen. Zur Verbesserung der Auflösung wird hier der Gradient aus der Absolutdifferenz des zentralen Bildpunktes und des zu testenden Nachbarn ermittelt. Die Verarbeitungsfunktion kopiert schließlich die Regionnummer des Nachbarbildpunktes, der bereits eine Regionnummer, die niedrigste Absolutdifferenz (Gradient) und den niedrigsten Distanzwert hat. Da die Startpunkte auch verarbeitet werden, wird zusätzlich überprüft, ob der Bildpunkt, der gerade verarbeitet wird, bereits eine Regionnummer hat.

Das folgende Code-Beispiel in C−Syntax zeigt die Software-Implementierung dieses Wasserscheide-Algorithmus.

```
/* Startkriterium:*/

int Watersched_Start_Krit(TPixel *in,TCoor x,TCoor y)
{
  return 1;
```

```
}

/* Nachbarschaftskriterium:*/

int Watersched_Nachbar_Krit(TPixel *in,int pos,TCoor x,TCoor y)
{
  if (in[pos].a.i) return 0;

  /* _fifo_regin1 ist eine globale Variable*/

  _fifo_regin1=abs(in[12].y.i - in[pos].y.i) +
    abs(in[12].u.i - in[pos].u.i) +
    abs(in[12].v.i - in[pos].v.i);

  /* Hierarchiestufe muss zwischen 1 und 256 sein*/
  if (++_fifo_regin1 > 256) return 256;
  return _fifo_regin1;
}

/* Verarbeitungsfunktion:*/

void Watersched_Verarbeitung(TPixel *res,TPixel *in,
                             TCoor x,TCoor y)
{
  /* _fifo_regin2            Incrementwert fuer die zu
                             testenden Richtungen
                             CON\_8: 1
                             CON\_4: 2*/
  static short diff;      /* Differenz zwischen Bildpunkt
                             und Nachbar*/
  static short mindiff;   /* minimale Differenz*/
  static short dir;       /* Richtung des Nachbars*/
  static short pos;       /* Position des Nachbars*/
  static short minpos;    /* Position des Nachbars mit
                             minimaler Differenz*/
  static short mingeo;    /* minimaler Distanzwert vom
                             Bildpunkt mit minimaler
                             Differenz*/
  static char dirtopos[8]={11,6,7,8,13,18,17,16};
                          /* Umrechnung von Richtung zur
                             Position*/
```

```
/* Prozedur fuer Startpunkte*/
if (in[12].a.i) {
  res->a=in[12].a;
  res->y=in[12].y;
  res->u=in[12].u;
  res->v=in[12].v;
  res->ax.i=1;
  return;
}

/* Prozedur fuer nicht Startpunkte*/
mindiff=0x7fff;

/* Pruefe, ob es Nachbarbildpunkte mit Regionnummer gibt*/
for (dir=0; dir < 8; dir +=_fifo_regin2) {
  pos=dirtopos[dir];

  /* Suche Nachbarn mit minimaler Differenz*/
  if ((in[pos].a.i) && (in[pos].m == MARKER_DEQUEUED)) {
    diff=abs(in[12].y.i - in[pos].y.i) +
      abs(in[12].u.i - in[pos].u.i) +
      abs(in[12].v.i - in[pos].v.i);
    if (diff < mindiff) {
      mindiff = diff;
      minpos = pos;
      mingeo = in[pos].ax.i;
    }

    /* Wenn die Differnz gleich ist, waehle Nachbarn
       mit niedrigerem Distanzwert*/
    else if (diff == mindiff) {
      if (in[pos].ax.i < mingeo) {
        mindiff = diff;
        minpos = pos;
        mingeo = in[pos].ax.i;
      }

      /*Kein weiteres Kriteriumm wenn der Distanzwert auch
        gleich ist*/
    }
```

```
    }
  }

  /* Schreibe Alpha- und Distanzwert*/
  res->a=in[minpos].a;
  res->y=in[12].y;
  res->u=in[12].u;
  res->v=in[12].v;
  res->ax.i=_fifo_distlevel; /* _fifo_distlevel wird innerhalb
                                der hdlifo_proc bzw. dlifo_proc
                                generiert*/
  /*  res->ax=in[minpos].ax+1;*/
}
```

Die drei Funktionen werden dann der Funktion zur rekursiven Adressierung mit HDLIFO übergeben. Für den Funktionsaufruf wird ein *Makro* mit dem Namen Watershed erzeugt, das eine einfachere Schnittstelle hat, als die hdlifo_proc-Funktion.

```
/* MomVop *vop;
   MomImage geodist;
   int connect (CON\_4 oder CON\_8) */
#define Waterscheд(vop,geodist,connect) \
  { \
    _fifo_regin2 = ((connect)==CON_4?2:1); \
    hdfifo_proc(vop,vop,geodist,MARKER, \
                A_CH|AX_CH,Y_CH|U_CH|V_CH|A_CH|AX_CH, \
                connect, \
                SCAN_LIST|SCAN_COLLECT|SCAN_MULIN,QOUT_PROC, \
                0,255,0, \
                0,0,0,0,0, \
                &Waterscheд_Start_Krit, \
                &Waterscheд_Nachbar_Krit, \
                &Waterscheд_Verarbeitung); \
  }
```

Nach dem Implementierungsbeispiel soll noch eine vollständige Spezifikation der ausgewählten Bildverarbeitungsfunktion gegeben werden, die sich am Code orientiert. Wie in der Makrodefinition zu sehen ist, sind Eingangs- und Ausgangsbild (vop) identisch und der Bildkanalspeicher für das Distanzfeld (geodist) wird in der aufrufenden Funktion verwaltet. Über das *Define*

MARKER kann bestimmt werden, ob ein globaler Marker-Kanal benutzt wird. Der MARKER hat dann den Wert des Zeigers auf den Speicher. Alternativ kann der Marker-Kanal von der hdlifo_proc-Funktion angelegt werden. Der Parameter MARKER hat den Wert Null. Als Ergebniskanäle werden der Alpha- und der Zusatzkanal (für das Distanzfeld) benutzt, während durch eine Oderverknüpfung der *Flags* alle Kanäle des Bildes geladen werden. Über den Parameter *connect* kann gewählt werden, ob ein CON_4 oder CON_8 Nachbarschaftssystem verwendet werden soll. Diese Information wird auch über die globale Variable _fifo_regin2 an die Verarbeitungsfunktion weitergegeben. Als Scan-Schema wird der Listenmodus für den Start-Scan und der collected-Modus mit mehrfachem Einschreiben der Bildpunkte (aber einfacher Verarbeitung) für den Ausbreitungsprozess verwendet. Das Process-on-queue-out Verarbeitungsmodell ist ausgewählt worden. Es werden 256 Hierarchiestufen verwendet, und der LIFO-Speicher wird wegen der Verwendung der Startliste nicht initialisiert. Die Werte für alle Kanäle werden auf Null gesetzt, wenn der Bildpunkt außerhalb des Bildes liegt. Die Funktionen zur Verarbeitung sind bereits vorher und zusätzlich durch den C−Code spezifiziert worden.

Insgesamt werden für die Implementierung des Wasserscheide-Algorithmus 41 Code-Zeilen benötigt (2 für das Startkriterium, 5 für das Nachbarschaftskriterium und 34 für die Verarbeitungsfunktion). Die Zeit, die zur Implementierung benötigt wurde, lag bei ca. 30 Minuten. Diese Zeit ist deshalb sehr niedrig, weil die Adressierung der Bilddaten einfach aus der AddressLib wieder verwendet wurde und somit schon fertig war. Im Vergleich dazu benötigt man für die Implementierung der HDLIFO-Adressierung, die bei einer dedizierten Implementierung gemacht werden müsste, einige Tage. Ein einfaches Kopieren einer ähnlichen Funktion und Verändern der Kopie ist wegen der hohen Komplexität der Adressierungsvariante nicht einfach möglich. Ähnlich viel Zeit wird anschließend noch für die Optimierung benötigt. Hinzu kommt noch ein Vielfaches der Zeit, um Fehler im Code zu identifizieren und zu beheben. Die Fehlerbeseitigung ist sehr aufwendig, da für eine optimierte Implementierung in jedem Fall Zeigerarithmetik verwendet werden muss. Diese Schritte entfallen aber beim Einsatz der AddressLib, da die Adressierungsfunktionen wegen der zentralen Implementierung nur ein einziges Mal fehlerkorrigiert werden müssen.

Neben dem sehr geringen Implementierungsaufwand ist diese Implementierung des Wasserscheide-Algorithmus und der hier verwendeten Interpretation der Funktion, vollständig spezifiziert. Dabei benutzt die Spezifikation einen festen Satz an Parametern, die des Funktionsaufrufs der Adressierungsfunktion. Dadurch sind die Funktionsspezifikationen konsistent zueinander. Außerdem sind die Funktionsspezifikationen durch die Software Co-

dierung und die damit verbundene Ausführbarkeit einfach zu verstehen. Bei der Spezifikation des Algorithmus kann man sich dann auf den High-Level-Algorithmus, der die Bildverarbeitungsfunktionen kombiniert, und die Verarbeitungsfunktionen konzentrieren. Diese beiden Teile sind speziell für einen Algorithmus und hängen von dem jeweiligen Know-How des Entwicklers und der Arbeitsgruppe ab. Auf der anderen Seite verlangen die Adressierungsfunktionen kein spezielles Wissen über den individuellen Algorithmus, sondern setzen nur noch Kenntnisse über das Anwendungsgebiet voraus. Durch diese Unterteilung wird die Komplexität des Algorithmus gut partitioniert und das Verständnis über den Algorithmus deutlich vereinfacht. Dies wird umso bedeutsamer, da ein großer Teil der Code-Zeilen des Algorithmus für die Adressierungsfunktionen verwendet wird.

Neben der allgemeinen Spezifikation der Bildverarbeitungsfunktion dienen sowohl die AddressLib als auch die Verarbeitungsfunktionen als ausführbare Spezifikation für das Hardware-Modell, der AddressEngine.

3.2 Profil der Mikroinstruktionen

Mit der Implementierung basierend auf dem Software-Modell ist im Rahmen dieser Arbeit ein Profil der Maschinenbefehle erzeugt worden. Dabei wurde der iprof [52][51] verwendet. Das Profil gibt Informationen über die Anzahl der arithmetischen und logischen Befehle, Lade- und Speicherbefehle und über bedingte und unbedingte Sprünge. In [53] wurden die Ergebnisse des Profils, die im Rahmen dieser Arbeit gewonnen wurden, veröffentlicht. Für die Erzeugung der Profile wurde der Algorithmus auf einem UltraSparc-Prozessor [88][61] kompiliert. Hierbei handelt es sich um eine RISC-Architektur. Da alle Arbeitsschritte auf einem RISC-Prozessor mit einem eigenen Befehl realisiert sind, finden sich alle Aktionen des Algorithmus im Profil wieder. Deshalb ist das Profil des Algorithmus auf der Basis eines RISC-Prozessor besonders aussagekräftig. Die Instruktionen werden für die drei Ebenen des Algorithmus getrennt gezählt. Diese drei Ebenen sind der High-Level Algorithmus, die Adressierungsfunktionen und die Verarbeitungsfunktionen. Abbildung 3.1 zeigt die drei Ebenen der Implementierung.

Tabelle 3.1 stellt hier noch einige Ergebnisse zusammen. Dabei ist die Anzahl der Maschinenbefehle der drei Ebenen des Algorithmus gegliedert in arithmetische Befehle, Sprungbefehle, Lade- und Speicherbefehle und sonstige Befehle. Es werden Ergebnisse für unterschiedliche Sequenzen angegeben.

Wie man aus dem Profil ersehen kann, benötigen die Adressierungsfunktionen mehr als 90% der RISC-Befehle. Ca. $8-9\%$ werden für die Verarbei-

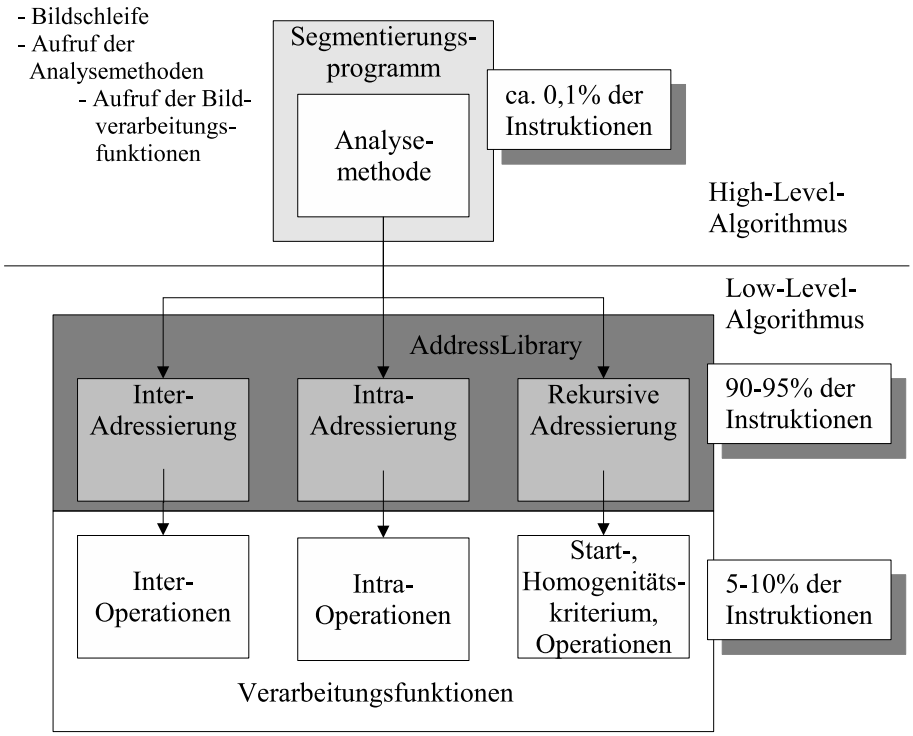


Abbildung 3.1: Hierarchien der Software-Implementierung des Algorithmus: Der High-Level-Algorithmus kombiniert die einzelnen Bildverarbeitungsfunktionen, die aus der Adressierung und der Bildpunktverarbeitung bestehen.

tungsfunktionen verwendet. Der High-Level-Algorithmus hat hingegen nur eine sehr geringe Rechenkomplexität mit weniger als 0,1% der Gesamtkomplexität. Da dieser Teil zugleich keine vorhersehbare Regularität aufweist, empfiehlt es sich, diesen Teil auf einem Standardprozessor zu implementieren. Dieser kann dann die AddressEngine oder einen zu ihr konformen Co-Prozessor ansteuern, um die Bildverarbeitungsfunktionen beschleunigt abzuarbeiten. Außerdem zeigt das Instruktionsprofil, dass insgesamt ca. 40% der RISC-Befehle für Lade- und Speicherbefehle verwendet werden. Die eigentliche *Nutzverarbeitung*, die auf der Verarbeitungsebene durchgeführt wird, beträgt nur ca. 8%. Berücksichtigt man dann noch, dass von diesen 8% nur 60% der Befehle tatsächlich für die Verarbeitung relevant sind, macht der *Overhead* der Instruktionen (inklusive Adressierung) ca. 95% der Gesamtbefehle aus. Unter der Annahme, dass sich der Overhead eliminieren oder durch Parallelisierung verdecken lässt, bietet der Algorithmus ein Optimierungspotential mit einem Faktor von 20. Dazu müsste dann aber auch die

Ebene/ Sequenz	Masch.- bef. $\times 10^6$	arith. Bef. $\times 10^6$	Sprünge $\times 10^6$	Load/ Store $\times 10^6$	sonstige Bef. x 10^6
Top-Level					
Foreman	3,07 (0,08%)	1,07 (34,85%)	0,83 (27,04%)	0,88 (26,71%)	0,35 (11,40%)
H1	2,26 (0,08%)	0,8 (35,40%)	0,62 (27,43%)	0,59 (26,22%)	0,25 (11,06%)
Mother & Daughter	2,20 (0,08%)	0,78 (35,45%)	0,61 (27,73%)	0,56 (25,45%)	0,25 (11,36%)
Adressierung					
Foreman	3505 (91,14%)	1134 (32,35%)	841 (23,99%)	1470 (41,94%)	60 (1,71%)
H1	2546 (91,38%)	826 (32,44%)	608 (23,88%)	1069 (41,99%)	43 (1,69%)
Mother & Daughter	2560 (91,32%)	828 (32,34%)	617 (24,10%)	1071 (41,84%)	44 (1,72%)
Verarbeitung					
Foreman	296 (7,78%)	70 (23,65%)	84 (28,38%)	123 (41,55%)	19 (6,42%)
H1	238 (8,54%)	60 (25,21%)	67 (28,15%)	96 (40,34%)	15 (6,30%)
Mother & Daughter	241 (8,60%)	60 (24,90%)	68 (28,22%)	98 (40,66%)	15 (6,22%)

Tabelle 3.1: Profil der RISC-Instruktionen beim Einsatz der Farbanalyse

Adressierung der Bildpunktdaten parallel zur Verarbeitung stattfinden. Dies würde den Einsatz einer dedizierten Hardware nahelegen.

Außerdem zeigt das Profil, dass die Gesamtkomplexität für die unterschiedlichen Sequenzen variiert. Dies liegt daran, dass zum Finden aller Startregionen bei der Foreman-Sequenz mehr Iterationen benötigt werden oder aber mehr Bildpunkte durch die Homogenitätsprüfung im Algorithmus wieder als unsegmentiert markiert werden.

Bemerkenswert ist auch der hohe Anteil an sonstigen Befehlen im Top-Level-Algorithmus und in der Verarbeitung. Sonstige Befehle sind im Wesentlichen Befehle zur Typenkonvertierung. Die Häufigkeit dieser Befehle im Top-Level-Algorithmus ist einfach nachvollziehbar. In der Verarbeitung hingegen ist dies nicht naheliegend. Der Grund hierfür liegt in der Unterstützung von unterschiedlichen Datenmodellen (Fließkomma- und Ganzzahlendarstel-

lung). Für eine Abschätzung der Komplexität sollten diese Befehle als Overhead betrachtet werden.

Ein weiteres interessantes Ergebnis ist die recht große Anzahl an Sprungbefehlen im Verarbeitungsteil. Dies deutet auf eine recht große Anzahl an nichtlinearen Filtern hin. Tatsächlich wird in vielen Filtern die Segmentzugehörigkeit der Bildpunkte berücksichtigt. Ein besonders deutliches Beispiel hierfür ist die Relaxation.

Neben dem Profil des gesamten Segmentierungsalgorithmus werden im Folgenden noch Bildverarbeitungsfunktionen der einzelnen Adressierungsmethoden miteinander verglichen. In diesem Fall ist das Profil auf einem CISC Prozessor (Pentium IV) durchgeführt worden. Bei dem Vergleich kommt es auf die Relation der Komplexitäten der Rechenzeit an und nicht auf die absoluten Komplexitäten, so dass hier auch die Verwendung eines CISC-Prozessors sinnvoll ist. Bei diesem Vergleich wurde ein Algorithmus entworfen, der Bildverarbeitungsfunktionen mit den wesentlichen unterschiedlichen Adressierungsmethoden verwendet. Wie in Tabelle 3.2 gezeigt ist, sind dies:

- die Inter-Adressierung
- die Inter-Adressierung mit CON_0 Nachbarschaft
- die Inter-Adressierung mit CON_8 Nachbarschaft
- die DLIFO-Adressierung mit CON_4 Nachbarschaft und collected-Modus
- die HDLIFO-Adressierung mit CON_4 Nachbarschaft und collected-Modus

Bei der Intra-Adressierung wird keine CON_4 Nachbarschaft gemessen, da diese keine Vereinfachung gegenüber der CON_8 Nachbarschaft darstellt. Ebenso wird keine LIFO-Adressierung im non-collected-Modus verwendet, da dieser prinzipiell keine andere Komplexität hat als der collected-Modus. Bei dem Profil werden nur die Instruktionen der Adressierungsebene gemessen. Trotzdem werden der Vollständigkeit halber der Algorithmus und die Verarbeitungsfunktionen beschrieben. Zuerst werden zwei Bilder eingelesen und die Absolutdifferenz der Bilder berechnet (Inter-Adressierung). Dann wird auf das Absolutdifferenzbild ein Schwellwert angewendet (Intra-CON_0), so dass nur noch eine binäre Information vorhanden ist. Es entstehen somit Regionen, in denen die Absolutdifferenz größer als der Schwellwert ist und Regionen, in denen die Absolutdifferenz kleiner ist. Für die entstehenden Regionen werden dann Distanzfelder berechnet (DLIFO-CON_4) und die lokalen Maxima der Distanzwerte (Intra CON_8) bestimmt (die s.g. Skelettpunkte). Anschließend

wird aus den Skelettpunkten wieder das Distanzfeld rekonstruiert (HDLIFO-CON_4). Durch einen Vergleich der Distanzfelder vor der Bestimmung der Skelettpunkte und nach der Ausbreitung der Skelettpunkte kann auf einfache Weise die Korrektheit des Ausbreitungsprozesses verifiziert werden. Die Rekonstruktion des Distanzfelds ist hier in zwei Varianten implementiert. In der ersten Variante werden die Skelettpunkte wie oben beschrieben mit einer Intra-Adressierung bestimmt und in eine Liste geschrieben. Für die Expansion wird dann kein Start-Scan durchgeführt, sondern die Startpunkte direkt aus der Liste verwendet. Im zweiten Fall wird die Skelettpunkt-Liste nicht verwendet, sondern ein horizontaler Startpunkt-Scan durchgeführt. Somit ist das Finden der Skelettpunkte und die Expansion in einem Schritt verknüpft. Nun können beide Implementierungsvarianten miteinander verglichen werden. Die Ergebnisse sind in der folgenden Tabelle zusammengefasst.

Adressierung	Funktion	Instruktionen $\times 10^6$	relativ zu Intra CON_0
Inter	absolut Differenz	11,07	39,83%
Intra CON_0	Schwellwert	27,79	100%
Intra CON_8	lokale Maxima Erkennung	66,66	239,87%
DLIFO CON_4	Distanzfeld	145,88	524,94%
HDLIFO CON_4 (Start Liste)	Expansion1	70,84	254,91%
Intra CON_8 + HDLIFO CON_4	Skelett + Expansion1	137,5	93,83% (rel. zu Expansion2)
HDLIFO CON_4 (SCAN_HO)	Expansion2	146,54	527,31%

Tabelle 3.2: Vergleich der Rechenkomplexitäten der Adressierungsmethoden. (Alle Bildverarbeitungsfunktionen verwenden nur die minimal nötige Anzahl an Kanälen zum Lesen und Schreiben der Bildpunktdaten.)

Die Ergebnisse dokumentieren die Relationen der Rechenzeitkomplexitäten der einzelnen Adressierungsmethoden. Hier ist zuerst festzustellen, dass für die Inter-Adressierung weniger Instruktionen benötigt werden, als für die einfachste Intra-Adressierung mit einer CON_0 Nachbarschaft. Vermutet wurde hier allerdings, dass die Intra-CON_0-Adressierung weniger komplex ist, da hier nur ein Eingangsbildpunkt geladen werden muss. Hingegen sind es zwei Eingangsbildpunkte bei der Inter-Adressierung. Die Intra-Adressierung

stellt aber eine Reihe von Parametern zur Verfügung, die innerhalb der Bildpunktschleife bearbeitet werden müssen. Dies sind die Möglichkeit der Skalierung und der Randwertersetzung. Die Randwertersetzung ist aber im Fall der Verwendung einer CON_0 Nachbarschaft nicht sinnvoll. So könnte die Intra-CON_0-Adressierung getrennt von der sonstigen Intra-Adressierung implementiert werden.

Außerdem ist zu sehen, dass die Intra-CON_8-Adressierung etwas weniger als dreimal so komplex ist wie die Intra-CON_0-Adressierung. Dies trifft die Erwartungen, da zum Laden der CON_8 Nachbarschaft bei der Intra-Adressierung in der Regel 3 Eingangsbildpunkte geladen werden müssen.

Im Vergleich zu einer Intra-Adressierung mit CON_8 Nachbarschaft ist die Verwendung eines reinen Ausbreitungsprozesses (HDLIFO mit Startliste) nur unwesentlich aufwendiger. Im Gegensatz dazu werden bei Ausbreitungsprozessen mit einem z.B. horizontalen Start-Scan des Bildes alle Nachbarschaften zweimal geladen (1x beim Start-Scan und 1x bei der Ausbreitung). Dadurch sind diese Adressierungsmethoden ungefähr doppelt so rechenintensiv wie Ausbreitungsprozesse mit einer Startpunktliste. Die Tabelle belegt außerdem, dass die Auftrennung in Start-Scan und Ausbreitungsprozess in zwei Bildverarbeitungsfunktionen (so dass der Start-Scan mit einer Intra-Adressierung durchgeführt wird) keinen wesentlichen Unterschied hinsichtlich der Rechenkomplexität ausmacht. Auf der einen Seite kann es vorkommen, dass das Startkriterium nicht alle Bildkanäle benötigt, die zum Ausbreiten verwendet werden müssen. In diesem Fall kann die Anzahl der Zugriffe auf den Bildspeicher durch eine Auftrennung in Start-Scan und Ausbreitung in zwei Bildverarbeitungsfunktionen optimiert werden. Auf der anderen Seite entsteht durch die Auftrennung ein Overhead in der Rechenzeit. Deshalb bietet es sich an, eine Auftrennung durchzuführen, wenn sich die Anzahl der zu lesenden Kanäle gut optimieren lässt.

Die Messungen wurden sowohl mit minimal nötiger Anzahl an Bildkanälen (in der Tabelle dargestellt) als auch mit genereller Verwendung des Y-, U- und V-Kanals durchgeführt. Die qualitativen Aussagen lassen sich auch auf den zweiten Fall übertragen.

3.3 Profil der Makroinstruktionen

Neben dem Profil der Mikroinstruktionen wurden in dieser Arbeit zwei einfache Ergebnisse basierend auf dem Profil der Makroinstruktionen erstellt. Dies ist auf der einen Seite die Messung der benötigten Bandbreite zum Bildspeicher und auf der anderen Seite der maximale Füllstand des LIFO-Speichers. Als Basis für die Speicherbandbreitenmessung wurde wieder das

Testprogramm mit den einzelnen Adressierungsmethoden verwendet. Dabei wurden alle Lese- und Schreibvorgänge von und zum Bildspeicher gemessen. Die Anzahl der Bildkanäle hat hierbei Einfluss auf die Ergebnisse. Die Bits pro Bildkanal beeinflussen hingegen nicht die Ergebnisse, obwohl eine Messung unter Berücksichtigung der Bit-Breiten möglich ist. Die Ergebnisse der Messung sind in Tabelle 3.3 zusammengefasst.

Adressierung	Funktion	Eingangs-kanäle	Lese - zugriffe	Ausg.-kanäle	Schreib-zugriffe
Inter	Absolut-differenz	Y	202.752	Y	101.376
Intra CON_0	Schwellwert	Y	101.376	Y	101.376
Intra CON_8	Skelett (lok. Max.)	Y	304.992	Y	101.376
DLIFO CON_4	Distanzfeld	Y,AX,M	2.215.428	AX,M	304.128
HDLIFO CON_4 (Start Liste)	Expansion1	A,AX,M	1.321.808	AX,M	290.538
Intra CON_8 + HDLIFO CON_4	Skelett + Expansion1	Y A,AX,M	1.626.800	- AX,M (nur HDLIFO)	290.538
HDLIFO CON_4 (SCAN_HO)	Expansion2	A,AX,M	2.171.142	AX,M	290.538

Tabelle 3.3: Messung der Speicherbandbreite der Adressierungsmethoden

Die Tabellen zeigen einen proportionalen Anstieg der benötigten Lese- bzw. Schreibzugriffe in Abhängigkeit von der Anzahl der Bildpunkte, die pro Bildpunktzyklus geladen werden und der Anzahl der Kanäle pro Bildpunkt. Zu geringfügigen Abweichungen zu den ganzzahligen Verhältnissen kommt es durch die irregulären Verarbeitungsmuster, wie z.B. bei Ausbreitungsprozessen. Bei der Intra-Adressierung mit einer CON_8 Nachbarschaft hingegen liegt die Abweichung an den Speicherzugriffen, die auf Bildpunkte gemacht werden, die außerhalb des gültigen Bildbereiches liegen. Implementierungsbedingt geschieht dies am Anfang und am Ende jeder Zeile. Im Vergleich der beiden Implementierungen zur Skelettexpansion sieht man, wie auch beim Profil der Mikroinstruktionen, ein mögliches Optimierungspotential bei der Durchführung des Start-Scans. Wird dieser mit einer vorgeschalteten Intra-CON_8-Adressierung durchgeführt, braucht unter Umständen nur ein Teil der Eingangskanäle geladen zu werden. Da nur die Zugriffe auf den Bildspeicher

gezählt werden, ergibt sich kein Overhead wie bei den Mikroinstruktionen. Das Profil der Makroinstruktionen zeigt somit nur die Komplexität, die unabhängig von der Implementierung ist, und inhärent zum Algorithmus ist. Damit kann das Profil auch auf andere Plattformen, wie z.B. die Address-Engine, übertragen werden. Wird neben dieser Komplexitätsbeurteilung des Algorithmus eine Abschätzung der tatsächlichen Laufzeit auf einer Architektur benötigt, müssen zusätzlich Architekturparameter berücksichtigt werden.

Die Speicherbandbreite wurde außerdem mit dem Algorithmus zur Objektsegmentierung gemessen. Dabei wurde die Speicherbandbreite für die Verarbeitung eines Bildes gemessen. Berücksichtigt wird dabei nur die Farbanalyse. Mit der Speicherbandbreite wird gleichzeitig der maximale Füllstand des LIFO-Speichers gemessen. Die Ergebnisse sind in Tabelle 3.4 aufgelistet.

Bild-nummer	Analyse-typ	Lese-zugriffe	Schreib-zugriffe	maximaler LIFO-Füllstand
0	Intra	243.267.497	66.597.675	5.728
1	Skip	12.626.478	4.871.784	1.328
2	Skip	12.626.823	4.871.841	1.328
3	Skip	12.626.220	4.871.743	1.319
4	Inter	66.120.200	21.025.565	2.552

Tabelle 3.4: Profil mit Lese- und Schreibzugriffen sowie dem maximalen LIFO-Füllstand (angewendet auf die Farbanalyse der Objektsegmentierung). Verwendet wurde die Akiyo-Sequenz mit dem Hintergrund der Bream2-Sequenz. Die Sequenz wurde in CIF Auflösung bearbeitet (ca 100.000 Bildpunkte).

Tabelle 3.4 zeigt, dass die Anzahl der Zugriffe auf den Bildspeicher für die einzelnen Bilder nicht konstant ist. Gut zu sehen ist, dass das erste Bild deutlich mehr Zugriffe auf den Bildspeicher benötigt als die folgenden Bilder. Die Ursache liegt darin, dass die Segmentmaske nicht aus vorhergehenden Bildern initialisiert werden kann. Somit werden viele (13) Iterationen benötigt, um alle Objekte im Bild zu finden. Da keine Information aus vorhergehenden Bildern verwendet werden, wird die Analyse auch als Intra-Analyse bezeichnet. Wie in Kapitel 1.6 beschrieben wurde, wird nicht in jedem Bild nach neuen Objekten gesucht. Anders ausgedrückt wird die Schleife zum Finden neuer Objekte übersprungen und nur eine Feinkorrektur der Segmentmaske durchgeführt. Dadurch ergeben sich deutlich weniger Zugriffe auf den Bildspeicher. Bilder, bei denen die Schleife übersprungen wird, haben den Wert

Skip in der Spalte "Analysetyp". Im vierten Bild wird nun wieder nach neuen Objekten gesucht. Da aber im Wesentlichen die Maske aus dem vorhergehenden Bild übernommen wurde, sind nur noch drei Iteration nötig, um die nicht Segmenten zugeordneten Bereiche zu analysieren. Dieser Analysetyp wird mit Inter-Analyse bezeichnet.

Unter der Annahme, dass für die Verarbeitung nach dem Finden von neuen Regionen ca. 12×10^6 Ladezugriffe auf den Bildspeicher durchgeführt werden, werden bei der Intra-Analyse pro Iteration ca. 17×10^6 Ladezugriffe benötigt. Überträgt man diesen Wert auf die Inter-Analyse, so ergibt eine Abschätzung ca. 63×10^6 Ladezugriffe für 3 Iterationen und die Feinkorrektur (vgl. Intra-Analyse). So wie hier die Analysetypen miteinander verglichen wurden, kann auch die Komplexität von unterschiedlichen Algorithmen betrachtet werden.

Nach der Messung der Anzahl der Speicherzugriffe wird nun der maximale Füllstand des LIFO-Speichers betrachtet. Dieser ist stark von den verwendeten Bildverarbeitungsfunktionen und vom Bildmaterial abhängig. In dem gewählten Beispiel muss der LIFO-Speicher mindestens 5728 Elemente umfassen. Als Relation dient die Größe des Bildes, das hier mehr als 100.000 Bildpunkte hat. Man sieht, dass zur Segmentierung des Bildes mit dem in Kapitel 1.6 vorgestellten Algorithmus nur ein Bruchteil des Speichers für die LIFO, gemessen an der Anzahl der Bildpunkte, vorhanden sein muss. Hier kann also im Fall einer Hardware-Implementierung an Speicher für die LIFOs gespart werden. Zur Abschätzung der Größe des Speichers sollte aber neben der Maximalgröße noch ein Sicherheitsfaktor angewendet werden, damit der LIFO-Speicher bei der Verwendung anderer Sequenzen nicht überläuft.

Sowohl im Fall einer Implementierung mit dem Software-Modell als auch bei einer Implementierung mit einer Hardware, die dem Konzept des Hardware-Modells folgt, ist die Anzahl der Lese- und Schreibzugriffe vom und zum Bildspeicher sowie der maximale LIFO-Füllstand identisch. Im Fall der Implementierung mit dem Software-Modell kommen zu diesen Speicherzugriffen noch alle Zugriffe auf den Speicher hinzu, um lokale und globale Variablen zu lesen und zu schreiben. Im Fall einer Hardware-Implementierung können demgegenüber die Zugriffe auf den Bildspeicher optimiert werden. Im Idealfall werden alle Bildpunktdaten für einen Bildpunktzyklus parallel in einem Taktzyklus geladen. Unter der Annahme, dass diese Speicherzugriffe der begrenzende Faktor bei der Hardware-Implementierung ist, kann die Laufzeit auf einer optimierten Hardware abgeschätzt werden. Diese Annahme scheint sinnvoll, da die Verarbeitungseinheit durch Fließbandverarbeitung einen sehr hohen Datendurchsatz erreichen kann und dadurch als unkritisch betrachtet werden darf. Eine Abschätzung der Laufzeit durch Messung der Zugriffe auf den Bildspeicher ist allerdings sehr optimistisch, da z.B. keine

Wartezyklen im Fall einer leeren DLIFO berücksichtigt werden. Dabei werden aber Wartezyklen mit zunehmender Fließbandlänge immer wahrscheinlicher. In jedem Fall werden aber deutlich weniger Speicherzugriffe benötigt, als bei einer Software-Implementierung.

In [1] wurden noch weitere Profile der Makroinstruktionen erzeugt. Dazu wurde in den Code-Segmenten der Makroinstruktionen Code zum Zählen der Anzahl der Aufrufe der Segmente eingeführt. Diese Technik ist nur zuverlässig anwendbar und führt zu vergleichbaren Ergebnissen, wenn die Code-Segmente nur ein einziges Mal vorhanden sind. Liegen hunderte von Kopien der Makroinstruktionen im Code vor, wie es bei einer typischen, dedizierten Implementierung der Fall ist, ist es so gut wie ausgeschlossen, dass der Zusatz-Code zum Zählen der Aufrufe überall korrekt eingefügt wurde.

3.4 Einsatzgebiete

Die AddressLib, das Software-Modell dieser Arbeit, wurde in mehreren Bildanalysesystemen verwendet. Dies sind:

- die MPEG−7 Referenz Software (XM-Software) [76]
- das COST AM (*Analysis Model*, Referenzsystem der COST211 Gruppe)
- das QIMERA-System [34]
- die Objektsegmentierungs-Software am LIS

Hierbei verwenden das COST AM und das QIMERA-System den in Kapitel 1.6 vorgestellten Algorithmus zur Farbanalyse. Neben diesem Farbanalysemodul sind in der Objektsegmentierungs-Software noch weitere Analysemodule unter Verwendung der AddressLib implementiert worden:

- Ein Block-Matching-Modul
- ein Modul zum Zusammenfassen von Farbsegmenten basierend auf Bewegungsinformation [89]
- ein Segmentierungsmodul für eine Formanalyse unter Verwendung von einfach geformten Elementen
- ein benutzerinteraktives Segmentierungsmodul
- ein Algorithmus zum Nachverfolgen von Gesichtern basierend auf einem Active-Contours-Algorithmus

In der MPEG−7 Referenz-Software wird die AddressLib benutzt, um aus visuellen Merkmalen MPEG−7 Beschreibungen zu extrahieren. Mit der AddressLib wurden die Extraktionsmethoden für folgende Beschreibungen implementiert:

- Scalable-ColorD
- FrameActivity, SummarizationDS und SegmentDS
- MosaicDS

Die AddressLib ließe sich auch auf die weiteren Extraktionsmethoden von visuellen Deskriptoren anwenden. Allerdings sind viele der Extraktionsfunktionen nicht für die Implementierung mit der AddressLib portiert worden. Bei der Extraktion des Scalable-Color-Deskriptor werden eine Farbraumkonvertierung, eine lineare Quantisierung und die Erstellung des Histogramms mit der AddressLib implementiert. Die drei Arbeitsschritte werden zusammen mit einem Aufruf der Intra-Adressierung durchgeführt. Die Funktion zum Verarbeiten eines Bildpunktes ruft dabei die drei Verarbeitungsfunktionen der einzelnen Arbeitsschritte auf. Auf eine Implementierung mit dem Hardware-Modell bezogen, entspricht dies einer Zusammenschaltung von drei AddressEngines über die Datenpfadeingänge und -ausgänge. Jede dieser AddressEngines führt dann einen der Verarbeitungsschritte aus.

Das SummarizationDS ist mit zwei Varianten implementiert. In einer Variante wird der *FrameActivity*-Wert mit einem *Kanade-Lucas-Tomasi-Feature-Tracker* (KLT) [90] berechnet und in einer zweiten Variante mit der Erkennung von neuem Bildinhalt basierend auf einer Block-Matching-Bewegungschätzung. Im SegmentDS wird dieser FrameActivity-Wert analysiert und Szenenschnitte erkannt. Zusätzlich wird eine Auswahl von *Key-Frames* durchgeführt.

Im MosaicDS wird eine globale Bewegungsschätzung mit 6 Parametern durchgeführt. In dieser Extraktionsfunktion wird die Inter-Adressierung mit Koordinatentransformation verwendet.

Der in dieser Arbeit verwendete Algorithmus ist Teil der Referenz-Software der Europäischen Projektgruppe COST 211. In dieser Referenz-Software wurden die integrierten Algorithmen in einem vergleichenden Verfahren ausgewählt. Die Integration in diese Software unterstreicht die Signifikanz des Algorithmus, der als Basis für den Entwurf des Modells für Bildverarbeitungsfunktionen diente. Seit kurzem ist der Algorithmus auch Teil der QIMERA-Software. Hier wurde der Algorithmus mit anderen Algorithmen verglichen. Die dort integrierten Algorithmen haben vergleichbare Qualität

(reproduzierbare Messungen liegen noch nicht vor), wobei der hier verwendete Algorithmus als einziger durch die zeitliche Prädiktion der Segmentmaske eine hohe zeitliche Kohärenz aufweist.

3.5 Bewertung

Mit der AddressLib ist eine signifikante Anzahl von Bildverarbeitungsalgorithmen für ein breites Spektrum an Bildanalyseanwendungen implementiert worden. In allen bekannten Fällen zeichnete sich der Algorithmenentwurf dadurch aus, dass die **Implementierung innerhalb kürzester Zeit** durchgeführt wurde und zuverlässig funktionierte. In der Phase zur Optimierung der Qualität des Algorithmus wurden einige Filter und Verarbeitungsfunktionen an die Anforderungen angepasst. Beispiele für angepasste Filter und Bildverarbeitungsfunktionen sind der Wasserscheide-Algorithmus mit der Verwendung der Bildpunktdifferenzen statt eines skalaren Gradientenwerts pro Bildpunkt, der Fragmentierungstest oder auch die Extraktion von Merkmalen, wie die Differenz oder Summe von U- und V-Kanal. Durch diese neu definierten und implementierten Funktionen, die durch die **gute Erweiterbarkeit** der Bildverarbeitungsfunktions-Bibliothek ermöglicht wurden, konnte die Qualität des Algorithmus deutlich verbessert werden.

Im Wesentlichen wird die schnelle Implementierbarkeit der Bildverarbeitungsfunktionen durch eine **verbesserte Strukturierung** der Bildverarbeitungsfunktionen sowie eine klare Strukturierung der Arbeitsabläufe erreicht. Hierbei spielt die klare Trennung in Know-How, das spezifisch für das Anwendungsgebiet ist, und Know-How, das spezifisch für die spezielle Anwendung ist, eine herausragende Rolle. Dadurch lassen sich Arbeitsabläufe, die spezifisch für das gesamte Anwendungsgebiet sind (die Adressierungsmethoden), wieder verwenden. Diese klare Strukturierung in einem Modell erlaubt dann auch eine **klare Spezifikation** der Bildverarbeitungsfunktionen, sowie des Algorithmus, der die Bildverarbeitungsfunktionen kombiniert.

Die Identifikation des generischen Teils der Bildverarbeitungsfunktionen, die Adressierungsmethoden, erlaubte es auch, diesen unabhängig vom spezifischen Teil, dem Algorithmus und den Verarbeitungsfunktionen, zu analysieren. Durch diese Analyse konnte eine **Verbesserung der Adressierungsmethode zur Darstellung von Ausbreitungsprozessen** erreicht werden. Hier wurden die aus der Literatur bekannten Ausbreitungsprinzipien systematisch betrachtet und ein neues Ausbreitungsverfahren mit einer hierarchischen dualen LIFO abgeleitet. Außerdem wurden die Implementierung des LIFO-Speichers durch die Verwendung einer dynamischen Elementzuordnung optimiert.

Ein weiteres wichtiges Merkmal des Algorithmenentwurfs mit der AddressLib ist die Möglichkeit sofort eine **Abschätzung der Komplexität** des Algorithmus durchzuführen. Gezeigt wurde dies an den Profilen der Mikro- und Makroinstruktionen. Hierfür sind keine Änderungen am Quellcode der Software-Implementierung des Algorithmus notwendig. Durch die Verwendung der AddressLib ist gewährleistet, dass die Profile unterschiedlicher Optimierungsgrade und -richtungen zueinander konsistent sind. Vereinfachungen, wie die Verwendung von Ganzzahlarithmetik statt Fließkommaarithmetik, lassen sich mit minimalem Aufwand ausprobieren. Als Konsequenz, kann der Algorithmus dadurch mit einer gleichzeitigen **Optimierungen bzgl. Qualität und Rechenzeit** verbessert werden.

Im Rahmen dieser Arbeit ist ein **Vergleich der Rechenkomplexitäten der einzelnen Adressierungsmethoden** durchgeführt worden. Es wurde gezeigt, dass Ausbreitungsprozesse nicht signifikant mehr Rechenzeit benötigen, als Filter mit einer vergleichbaren Nachbarschaft, die eine Intra-Adressierung verwenden. Zu beachten ist allerdings, dass die Implementierungskomplexität der Adressierung für Ausbreitungsprozesse höher ist, da LIFOs zur Erzeugung der Abarbeitungsfolge verwendet werden.

Die Verwendung der AddressLib zur Implementierung des Algorithmus ermöglicht es auf einfache Art ein **Profil der Makroinstruktionen** zu erstellen. Dieses Profil ist besonders aussagekräftig bzgl. Hardware-Implementierungen. Dies liegt daran, dass die Makroinstruktionen **unabhängig von der Implementierungsplattform** ausgeführt werden müssen. Dies verdeutlichen auch die drei Ebenen des Modells, das abstrakte Modell, das Software-Modell und das Hardware-Modell. Diese sind im Wesentlichen identisch. Unterschiede ergeben sich in Einschränkungen der Parametrisierung, die notwendig sind, um sinnvolle Implementierungen auf Software- und Hardware-Ebene zu erreichen. Durch die weitgehende Übereinstimmung der Modelle kann auch gewährleistet werden, dass eine **Übertragung des Algorithmus vom Software- auf das Hardware-Modell** möglich ist. Hierfür müssen lediglich die Einschränkungen des Hardware-Modells eingehalten werden.

Das Hardware-Modell (die AddressEngine) unterliegt keinen Optimierungen für einen speziellen Algorithmus. Es dient somit als Konzept, oder Grundarchitektur für reale Hardware-Implementierungen. Die AddressEngine ist aber in jedem Fall geeignet, um bereits frühzeitig als **Komplexitätsmodell** zur Abschätzung der Rechenzeit auf einer optimierten Architektur eingesetzt zu werden.

Kapitel 4

Zusammenfassung und Ausblick

Durch die Analyse eines Beispielalgorithmus zur Objektsegmentierung ist ein Modell für Bildverarbeitungsfunktionen mit Schwerpunkt auf Bildanalyse erstellt worden. Dabei wurde bei der Entwicklung des Algorithmus auf eine repräsentative Abdeckung an Funktionen für die Bildanalyse geachtet. Das resultierende Modell umfasst drei Ebenen: das abstrakte Modell, das Software-Modell und das Hardware-Modell. Das abstrakte Modell definiert die Modellparameter, die zur Abbildung der Bildverarbeitungsfunktion benötigt werden. Das Software-Modell ist als Funktionsbibliothek in der AddressLib implementiert. Das Hardware-Modell ist ein Architekturkonzept, das als AddressEngine bezeichnet wird. Zur Bildung des Modells wurden die gemeinsamen Komponenten der im Beispielalgorithmus verwendeten Bildverarbeitungsfunktionen identifiziert und im abstrakten Modell abgebildet. Diese gemeinsamen Komponenten sind die Adressierungsmethoden, die festlegen, welche Bilddaten in welcher Reihenfolge verarbeitet werden. Dabei wurden drei unterschiedliche Typen von Adressierungsmethoden gefunden, die Inter-, Intra- und rekursive Adressierung. Die Adressierungsmethoden sind charakteristisch für das Anwendungsgebiet (digitale Bildanalyse). Deshalb stellt das Modell in kompakter Form das Know-How für das Anwendungsgebiet bereit. Bildverarbeitungsfunktionen benötigen neben der Adressierung der Bildpunktdaten eine Funktion, mit denen die Bildpunktdaten verarbeitet werden. Diese Funktionen sind spezifisch für einen Algorithmus und repräsentieren das dafür nötige Know-How. Da alle möglichen Verarbeitungsfunktionen denkbar sind, ist nur die Schnittstelle der Funktionen Teil des Modells. Durch die Trennung in Adressierung und Verarbeitung lassen sich zwei wesentliche Vorteile erreichen:

1. Das Problem wird partitioniert und dadurch leichter überschaubar.

2. Bildverarbeitungsfunktionen können einfach und eindeutig spezifiziert

werden.

Das Software-Modell implementiert das abstrakte Modell in einer Funktionsbibliothek. Durch die Verwendung des Software-Modells zur Implementierung der Bildverarbeitungsfunktionen kann der Algorithmus innerhalb kürzester Zeit simuliert werden. Die Software-Implementierung zeichnet sich durch einen sehr niedrigen Aufwand zum Debuggen aus, da der wesentliche Teil der Software, die Adressierungsfunktionen, aus der Bibliothek verwendet wird und bereits weit reichend getestet ist. Durch Verwendung von Zeigerarithmetik in den Adressierungsfunktionen hat der resultierende Code bereits ein gutes Niveau der Optimierung. Hier lohnt sich der Optimierungsaufwand besonders, da er auf alle Bildverarbeitungsfunktionen wirkt. Zusätzlich hat eine Optimierung der Adressierungsfunktionen einen besonders großen Einfluss auf die Gesamtrechenzeit, da diese Funktionen hierzu den größten Anteil beitragen.

Schließlich ist noch ein Konzept für eine Hardware-Architektur vorgestellt worden, das konform zum Software-Modell ist. Dadurch kann das Hardware-Konzept als Komplexitätsmodell für die Ermittlung der Rechenkomplexität der Bildverarbeitungsfunktionen und des Bildverarbeitungsalgorithmus verwendet werden. Das Hardware-Modell bildet die inhärenten Merkmale des Anwendungsgebiets ab (wie auch das abstrakte und das Software-Modell). Die so erstellten Profile stellen nur die Eigenschaften dar, die durch die Komplexität des Algorithmus selber bestimmt werden und unabhängig von der Zielarchitektur sind. Auf der anderen Seite ist die tatsächliche Rechenzeit auf einer bestimmten Zielarchitektur auch von der Architektur abhängig. Im Ausblick werden Vorschläge gemacht, wie aus den beiden Teilinformationen, der Komplexität des Algorithmus und den Architekturparametern, die Rechenzeit abgeschätzt werden kann.

Zur Messung der Rechenkomplexität sind Profile der Makro- und Mikroinstruktionen erstellt worden. Die Profile der Mikroinstruktionen wurden verwendet, um die Komplexitäten der einzelnen Ebenen der Algorithmenimplementierung zu messen. Dabei hat sich gezeigt, dass der High-Level Algorithmus, der die Bildverarbeitungsfunktionen aufruft, nur einen verschwindend geringen Anteil an der Gesamtkomplexität hat. Mehr als 90% der Instruktionen werden für die Adressierungsmethoden verwendet. Dieser Teil ist der generische Teil, der für das Anwendungsgebiet spezifisch ist. Ungefähr 7−8% der Instruktionen werden für die eigentliche Verarbeitung der Bildpunkte verwendet. Eine Betrachtung der Komplexität und Flexibilität auf den drei Ebenen legt eine Partitionierung nahe, bei der die Bildverarbeitungsfunktionen mit einer angepassten Architektur implementiert werden und der High-Level-Algorithmus auf einem Standardprozessor.

Der relativ geringe Anteil der Verarbeitungsebene an der Gesamtkomplexität deutet außerdem auf ein gutes Optimierungspotential bei der Implementierung des Algorithmus hin. Geht man davon aus, dass eine Architektur zur Implementierung der Bildverarbeitungsfunktionen verwendet wird, die an dieses Aufgabengebiet angepasst ist, dann würden die Adressierung und die Verarbeitung parallel ablaufen. Dabei können sowohl noch innerhalb der Adressierung als auch der Verarbeitung Parallelisierungen durchgeführt werden. Zu erwarten ist dann, dass die Bandbreite zum Bildspeicher der begrenzende Faktor sein wird. Obwohl dies von der Architektur selber abhängig ist, wird dies in der Regel der kritische Teil sein, um eine balancierte Verteilung der Ressourcen-Auslastung zu erreichen. Deshalb wurde mit dem Profil der Makroinstruktionen eine Messung der Zugriffe auf den Bildspeicher durchgeführt. Vergleicht man das Profil der Makroinstruktionen mit dem der Mikroinstruktionen, so sieht man, dass z.B. bei der Verwendung einer einfachen Adressierung (Intra-CON_0) für die Verarbeitung eines Bildes im CIF-Format ca. 200.000 Zugriffe auf den Bildspeicher benötigt werden, bei knapp 28 Mio. RISC-Instruktionen. Dies ist ein aussagekräftiger Hinweis auf das Optimierungspotential bei Verwendung eines angepassten Prozessors.

Abschließend kann festgestellt werden, dass die gestellten Aufgaben gelöst wurden. Durch das entwickelte Modell können Bildverarbeitungsfunktionen und -algorithmen eindeutig und einfach spezifiziert werden. Durch die Anwendung des Software-Modells ist diese Spezifikation zudem ausführbar und leicht nachvollziehbar. Mit dem Software-Modell können Bildverarbeitungsfunktionen effizient implementiert werden. Dies betrifft sowohl den Prozess der Implementierung als auch die Laufzeit des Codes. Für Ausbreitungsprozesse ist ein verbessertes Verfahren mit einer hierarchischen dualen LIFO vorgeschlagen worden. Durch die Definition des Hardware-Modells können Profile unabhängig von der Zielarchitektur beurteilt werden. Insgesamt ergibt sich ein durchgängiger Entwurfsprozess, der für die Qualitätsoptimierung genauso geeignet ist, wie für eine Rechenzeitoptimierung.

Das in dieser Arbeitet entwickelte Software-Modell wurde in einer Reihe von Algorithmen verwendet. So wird es auch in der MPEG−7 Referenz Software verwendet, um Extraktionsmethoden für visuelle Merkmale zu implementieren. Unter Verwendung einer Hardware-Implementierung des Software-Modells könnten dann alle Extraktionsmethoden durch diesen speziellen Bildanalyseprozessor beschleunigt werden. Dieser Prozessor wäre dann, wie im Konzept des Hardware-Modells vorgeschlagen, für Bildanalysefunktionen optimiert. Bei der Verwendung einer flexiblen Verarbeitungseinheit wäre dieser aber nicht auf einen Algorithmus beschränkt.

Um die volle Leistungsfähigkeit der Verwendung des Modells für Bildverarbeitungsfunktionen zu erreichen, soll in zukünftigen Arbeiten noch ei-

ne konkrete Implementierung des Hardware-Modells durchgeführt werden. Optimierungen der in dieser Arbeit vorgestellten Grundarchitektur wurden bereits in einer weiterführenden Arbeit durchgeführt. In dieser Arbeit wurde auch eine Methodik entwickelt, wie durch die Verknüpfung von Profilen der Makroinstruktionen und Architekturparametern die Rechenzeit auf der Hardware-Architektur abgeschätzt werden kann. Neben dieser Methode könnten auch die Aufrufe der Makroinstruktionen in der Software benutzt werden, um eine taktgenaue Simulation der Architektur zu steuern. Dafür werden dann noch zwei Eingangsinformationen bzgl. der Hardware benötigt. Auf der einen Seite sind dies Annahmen der zeitlichen Abläufe der Makroinstruktionen, die dann später von der Architektur erfüllt werden müssen. Auf der anderen Seite werden vorgegebene Abhängigkeiten der Instruktionen und ihrer verwendeten Ressource benötigt. Diese Informationen könnten dann von einem Simulator gelesen werden, der mit der Software-Simulation gekoppelt ist. Wenn eine Makroinstruktion aufgerufen wird, werden im Simulator die zeitlichen Abfolgen der Makroinstruktion gestartet. Der Simulator erzeugt dann unter Berücksichtigung der Abhängigkeiten und Ressourcen-Konflikte der Instruktionen einen taktgenauen Ablauf der Verarbeitung auf dem Prozessor. Diese Simulation kann durchgeführt werden, bevor die Module des Prozessors auf Register-Transfer-Level (RTL) modelliert wurden.

Wenn eine reale Hardware-Implementierung der AddressEngine durchgeführt wurde, benötigt man noch eine Programmierumgebung, um den Prozessor in den Ablauf des Algorithmus einzubinden. Hier könnte eine abgewandelte Version der AddressLib verwendet werden. Durch Aufrufe der Bildverarbeitungsfunktionen kann dann die Parametrisierung der Adressierungsfunktionen auf dem Prozessor durchgeführt werden. Die Funktion übernimmt auch die Programmierung des Verarbeitungsteils und startet den Verarbeitungsprozeß. Offen bleibt dann, wie aus den C-Beschreibungen der Verarbeitungsfunktionen die Konfiguration der Verarbeitungseinheit abgeleitet werden kann. Hier wäre es denkbar, dass die Funktionen auf der einen Seite als C−Beschreibungen für die Software-Simulation und auf der anderen Seite als Konfigurationsmakro für die Hardware-Implementierung vorliegen. Eine Methodik für die automatische oder halbautomatische Umsetzung der C−Beschreibung in ein Konfigurationsmakro kann aber erst entwickelt werden, wenn die Verarbeitungseinheit implementiert ist. Bei der Entwicklung der Verarbeitungseinheit sollten aber bereits Aspekte des Programmiermodells berücksichtigt werden.

Langfristig wäre es denkbar, das Software- und Hardware-Modell als Produkt für den Entwurf von *Embedded Systems* mit Bildanalysefunktionen anzubieten. Hier könnte dann ein AddressEngine-*Core* mit einem Kamerasensor auf einem Chip kombiniert werden, um in der Kamera bereits die Bildana-

lyse durchzuführen. Anwendungen für diese eingebetteten Systeme könnten dann die Extraktion von MPEG−7-Deskriptoren für visuelle Merkmale oder eine Objektsegmentierung sein. Eine Anwendung für MPEG−7-Deskriptoren wäre z.B. ein verteiltes Überwachungssystem, bei dem viele Kameras eingesetzt werden. Jede Kamera könnte dann eigenständig Alarme auslösen. Ein Server entscheidet dann, was in unterschiedlichen Alarmsituationen mit den Daten gemacht werden soll. Der Server selber braucht aber nicht die Rechenleistung zu haben, um für hunderte von Kameras die Bildanalyse durchzuführen. Objektsegmentierung könnte in Automobilen benutzt werden, um einen Autopiloten zu implementieren. Hier würde eine Kamera mit integrierter Bildanalyseeinheit eine kompakte Komponente für ein solches System sein.

Literaturverzeichnis

[1] H. Mooshofer, "Entwurfsmethodik für eine flexible Architektur zur Videoobjekt-Segmentierung," *Dissertation am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 2001.

[2] A. Kaup, *Modelle zur regionorientierten Bildbeschreibung*, VDI-Fortschrittsberichte, Reihe 10, Nr. 381, VDI-Verlag, Düsseldorf, Juli 1995.

[3] S. Ohm, J.-R. Bauer, C. Herpel, A. Kaup und J. Spille, "Der MPEG-4 Multimedia-Standard und seine Anwendungen im MINT-Projekt," *Der Fernmeldeingenieur*, Nov. 1998.

[4] X. Marichal, "On-line Web Application using Image Segmentation," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'99*, Juni 1999.

[5] M. Wollborn und R. Mech, "A Noise Robust Method for Segmentation of Moving Objects in Video Sequences," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'97*, Bd. 4, pp. 2657–2660, 1999.

[6] R. Mech und M. Wollborn, "A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[7] A. Smolic, "Globale Bewegungsbeschreibung und Video Mosaiking unter Verwendung parametrischer 2-D Modelle, Schätzverfahren und Anwendungen," *Dissertation an der Rheinisch-Westfälischen Technischen Hochschule Aachen (RWTH), Fakultät für Elektrotechnik und Informationstechnik*, 2001.

[8] E. Tuncel und Onural L., "Utilization of the Recursive Shortest Spanning Tree Algorithm for Video Object Segmentation by 2-D Affine Mo-

tion Modeling," *IEEE Transactions on Circuits and Systems for Video Technology, CSVT*, Bd. 10, Nr. 5, pp. 776–781, 2000.

[9] O.J. Morris, M.J. Lee und A.G. Constantinidis, "Graph theory for image analysis: An approach based on the shortest spanning tree," *IEE Proceedings*, Bd. 133, pp. 146–152, 1986.

[10] P. Eren, Y. Altunbask und A.M. Tekalp, "Region Based Affine Motion Segmentation Using Color Information," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'97*, Bd. 4, pp. 3005, Apr. 1997.

[11] S. Herrmann, H. Mooshofer, H. Dietrich und W. Stechele, "A Video Segmentation Algorithm for Hierarchical Object Representations and its Implementation," *IEEE Transactions on Circuits and Systems for Video Technology, CSVT*, Bd. 4, Nr. 8, pp. 1204–1215, 1999.

[12] E. Izquierdo und M. Ghanbari, "Object Segmentation Using Disparity-Dependent Perona-Malik Model," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'99*, Juni 1999.

[13] C. Estermann, "Tiefenschätzung anhand des Schwenks einer einzigen unkalibrierten Kamera," *Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 2003.

[14] 3DV Systems, "Zcam," *http://www.3dvsystems.com/products/zcam.html*, 2004.

[15] N. O'Conner und S. Marlow, "Supervised Image Segmentation using EM-based Estimation of Mixture Density Parameters," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, pp. 27–32, Juni 1997.

[16] N.V. Boulgouris, I. Kompatsiaris, V. Mezaris und M.G. Strintzis, "Content-based Watermarking for Indexing Using Robust Segmentation," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'01*, pp. 129–132, 2001.

[17] J.A. Hartigan und M.A. Wong, "A K-means clustering algorithm," *Applied Statistics*, Bd. 28, Nr. 1, pp. 100–108, 1979.

[18] V. Mezaris, I. Kompatsiaris und M.G. Strintzis, "Still Image Segmentation Tools for Object-based Multimedia Applications," *International Journal of Pattern Recognition and Artificial Intelligence*, Bd. 18, no 4, pp. 701–725, 2004.

[19] D.J. Williams und M. Shah, "A Fast Algorithm for Active Contours and Curvature Estimation," *Computer Vision Graphics Image Process (CVGIP): Image Understanding*, Bd. 55, Nr. 1, pp. 14–26, 1992.

[20] M. Kass, A. Witkin und D. Terzopoulus, "Snakes: Active Contour Models," *First International Conference on Computer Vision*, pp. 259–268, 1987.

[21] S. Geman und D. Geman, "Stochastic relaxation, Gibbs distributions and the Byesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Bd. 6, pp. 721–741, 1984.

[22] J.-R. Ohm, *Digitale Bildcodierung*, Springer, Berlin, 1995.

[23] J. Kim und T. Chen, "A VLSI Architecture for Video-Object Segmentation," *IEEE Transactions on Circuits and Systems for Video Technology, CSVT*, Bd. 13, Nr. 1, pp. 83–96, 2003.

[24] L. Vincent und P. Soille, "Watershed in Digital Spaces: An Efficient Algorithm based on Immersion Simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Bd. 13, pp. 583–598, 1991.

[25] L. Garrido, F. Marques, M. Pardas, P. Salembier und V. Vilaplana, "A Hierarchical Technique for Image Analysis," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[26] F. Meyer, "Topographic Distance and Watershed Lines," *Signal Processing*, Bd. 38, pp. 113–125, 1994.

[27] F. Meyer und S. Beucher, "Morphological Segmentation," *Journal of Visual Communications and Image Representation, JVCIR*, Bd. 1, Nr. 1, pp. 21–46, 1990.

[28] J.B. Kim und H.J. Kim, "A Wavelet-Based Watershed Image Segmentation for VOP Generation," *16 th International Conference on Pattern Recognition, ICPR'02*, Bd. 1, Nr. 1, pp. 21–46, 2002.

[29] V. Grau, A.U.J. Mewes, M. Alcaniz, R. Kikinis und S.K. Warfield, "Improved watershed transform for medical image segmentation using prior information," *IEEE Transactions on Medical Imaging*, Bd. 23, Nr. 4, pp. 447– 458, 2004.

[30] F. Meyer, "From connected operators to levelings," *Proceedings of the fourth international symposium on Mathematical morphology and its applications to image and signal processing*, pp. 191–198, 1998.

[31] F. Meyer, "Levelings, Image Simplification Filters for Segmentation," *Journal of Mathematical Imaging and Vision archive*, Bd. 20, Issue 1-2, pp. 59–72, 2004.

[32] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.

[33] A. Alatan, E. Tuncel und L. Onural, "Object Segmentation via Rule-based Data Fusion," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[34] N. O'Connor, S. Sav, T Adamek, V. Mezaris, I. Kompatsiaris, T.Y. Lu, E. Izquierdo, C. Bennström und J. Casas, "Region and Object Segmentation Algorithms in the QIMERA Segmentation Platform," *Proceedings of Third International Workshop on Content-Based Multimedia Indexing, CBMI 2003*, 2003.

[35] P. Salembier und F. Marques, "Image and video segmentation tools for new multimedia services," *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on*, 1999.

[36] T. Sikora und H. Seguin, "The European COST211ter Group - Research on Redundancy Reduction Techniques and Content Analysis for Multimedia Services," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[37] J. Stauder und R. Mech, "Detection and Tracking of Moving Cast Shadows," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'99*, Juni 1999.

[38] P. Correia und F. Pereira, "Segmentation of Video Sequences in a Video Analysis Framework," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[39] V. Mezaris, R. Median, Y. Kompatsiaris, S. Herrmann und J. Casas, "M10566: Experimental results for a Search & Retrieval System using MPEG-7 still image descriptors," *ISO/IEC*, Mär. 2004.

[40] V. Mezaris und Y. Kompatsiaris, "SCHEMA XM Server Search Engine: http://media.iti.gr/site/SchemaXMServer," 2004.

[41] CMM, "PIMM1 User Guide," *Ecole des Mines de Paris, Centre de Morphologie Mathematique*, 1990.

[42] F. Lemonnier, "Architecture électronique dédiée aux algorithmes rapides de segmentation basés sur la morphologie mathématique," *Thèse de Doctorat en Morphologie Mathématique, ENSMP, TL-05/97/MM*, 1996.

[43] CMM, "Xlim3d," *http://cmm.ensmp.fr/xlim3d.html*, 2004.

[44] ADCIS, "Aphelion 3.2g," *http://www.adcis.net/*, 2004.

[45] C. Goldberg, "Visual Architect Bridges the Gap Between Systems and ASIC Designers," *http://www.edacafe.com/technical/papers/Cadence/archive*, Bd. 2, Issue 2, 1997.

[46] Cadence, "Cadence Multimedia Design Kit 2.0: http://www.cadence.com/datasheets/multimedia_design_kit.html," 2003.

[47] Intel, "Intel integrated performance primitives," *http://www.intel.com/software/products/ipp*, 2004.

[48] J. Kneip, M. Berekovic, J. Wittenburg, W. Hinrichs und P. Prisch, "An Algorithm Adapted Autonomous Controlling Concept for a Parallel Single-Chip Digital Signal Processor," *Journal of VLSI Signal Processing*, Bd. 16, pp. 31–40, 1997.

[49] W. Gehrke und K. Gaedke, "Associative Controlling of Monolithic Parallel Processor Architectures," *IEEE Transactions on Circuits and Systems for Video Technology, CSVT*, Bd. 5, Nr. 5, pp. 453–464, Okt. 1995.

[50] P. Pirsch, W. Gehrke, K. Gaedke und K. Hermann, "A Parallel VLSI Architecture for Object-Based Analysis-Synthesis Coding," *Proceedings of the IEEE Workshop on Visual Signal Processing and Communications*, pp. 136–140, 1994.

[51] P. Kuhn, "A portable Instruction Level Profiler for Complexity Analysis - Software," *MPEG96/M1056*, 1996.

[52] P. Kuhn, "iprof, anonymous ftp: ftp://ftp.lis.e-technik.tu-muenchen.de/pub/iprof," 1996.

[53] S. Herrmann, H. Mooshofer und W. Stechele, "A Toolbox Approach for Image Segmentation and Complexity Analysis," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'97*, Juni 1997.

[54] S.D. Brown, R.J. Francis, J. Rose und Z. Vranesic, *Field Programmable Gate Arrays*, Kluwer-Verlag, Dordrecht/Boston/London, 1992.

[55] L.H. Cooke, "Use of Configurable Cores in Platform based SoCs," *http://www.easic.com/technolgy/whitepapers.html*, 2000.

[56] P. Pirsch und H.-J. Stolberg, "Bei Licht betrachtet: Die Architektur des Pentium 4 im Vergleich zu Pentium III und Athlon," *CT, Heise Verlag*, Bd. 24, pp. 134–141, 2000.

[57] P. Pirsch und H.-J. Stolberg, "DecAthlon," *CT, Heise Verlag*, Bd. 24, pp. 228, 2000.

[58] C. Windeck, "Motorwechsel: Der Intel Pentium 4 mit 90-nm-Strukturen in der Praxis," *CT, Heise Verlag*, Bd. 4, pp. 160, 2004.

[59] M. Choudjury und J. Miller, "A 300 MHz CMOS Microprocessor with Multi-Media Technology," *IEEE International Solid-State Circuits Conference, ISSCC'97*, pp. 170–171/450, 1997.

[60] D. Draper. et al., "An x86 Microprocessor with Multimedia Extensions," *IEEE International Solid-State Circuits Conference, ISSCC'97*, p. 172/450, 1997.

[61] K. Normoyle et al., "Ultra-Sparc-IIi: Expanding the Boundaries of a System on a Chip," *IEEE Micro*, pp. 14–23, 1998.

[62] M.Y. Siyal und M. Fathy, "Triple RISC Image Operator for Real-Time Image Processing Applications," *Electronic Letters*, Bd. 32, Nr. 24, pp. 2224–2225, 1996.

[63] K.A. Jacob, "Architecture Design for Entropy Coding in Object-Based Video," *Dissertation am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 2002.

[64] T.J. Callahan, J.R. Hauser und J. Wawrzynek, "The Garp Architecture and C Compiler," *Computer*, Bd. 33, Nr. 4, pp. 62–69, 2000.

[65] Triscend, "Triscend A7S 32-bit Configurable System-on-Chip Family," *Product Brief, http://www.triscend.com/products/TextTechLit.html*, 2003.

[66] Texas Instruments, "TMS320C8x System-Level Synopsis," *Literature Number: SPRU113B, http://www.ti.com*, 2000.

[67] Texas Instruments, "TMS320C6000 CPU and Instruction Set Reference Guide," *Literature Number: SPRU189F, http://www.ti.com*, 2000.

[68] Philips, "Tri-Media TM-1300: Programmable Media Processor," *Datenblatt, Philips Semiconductors Trimedia Business Line, http://www.trimedia.philips.com.*

[69] K. Rönner und J. Kneip, "Architecture and Applications of the HiPAR Video Signal Processor," *IEEE Transactions on Circuits and Systems for Video Technology, CSVT*, Bd. 6, Nr. 1, pp. 163–173, 1996.

[70] P.J. Drongowski und K. Andress, "A Morphology-based Processor for Realtime Enhancement," *Proceedings of SPIE*, Bd. 1823, pp. 25–36, 1992.

[71] A. Chihoub, A. Tsai, M. La Valva, J. Avins und J. Turlip, "A Field Programmable Gate Array Implementation of a Systolic Architecture for a Morphology Engine," *Proceedings of SPIE*, Bd. 2064, pp. 95–106, 1993.

[72] J-C. Klein, F. Lemonnier, M. Gauthier und P. Peyrad, "Hardware Implementation of the Watershed Zone Algorithm based on a Hierarchical Queue Structure," *IEEE Workshop on Nonlinear Signal Processing*, 1995.

[73] D. Noguet, "A Massive Parallel Implementation of the Watershed Based on Cellular Automata," *Proceedings of the IEEE International Conference on Applications-Specific Systems , Architectures and Processors*, pp. 42–52, 1997.

[74] J. Schönfeld, "Kompakte Implementierung konturorientierter Bildverarbeitungssysteme mit VLSI-Bausteinen," *VDI-Fortschrittsberichte, Reihe 10, Nr. 321*, Juli 1994.

[75] J. Miteran, R. Bailly und P. Gorria, "Classification Board for Real-Time Image Segmentation," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'97*, Bd. 5, pp. 4069–4072, 1997.

[76] ISO/IEC 15938-6, "Text of ISO/IEC FCD 15938-6 Information Technology - Multimedia Content Description Interface - Part 6: Reference Software (Version 1.0)," *ISO/IEC*, Mär. 2001.

[77] ISO/IEC 14496-9, "MPEG-4 Reference Software," *ISO/IEC*.

[78] A.A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel und T. Sikora, "Image Sequence Analysis for Emerging Interactive Multimedia Services - The European COST 211 Framework," *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Representation and Coding of Images and Video 1*, Bd. 8, pp. 802–813, Nov. 1998.

[79] GNU, "http://www.gnu.org/manual/gprof-2.9.1/gprof.html," 2003.

[80] P. Kuhn, "Algorithms, Complexity-Analysis and VLSI-Architectures for MPEG-4 Motion Estimation," *Kluwer academic publishers*, p. 248, Juni 1999.

[81] Z. Yong und M. Zhang, "A Novel Superscalar Architecture for Fast DCT Implementation," *Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, 2000.

[82] G. Masera, A. Molino, G. Piccinini und M. Zamboni, "A Parametric DCT Architecture for H.263+ Mobile Terminals Video Streaming," *IEEE International Midwest Symposium on Circuits and Systems - MWSCAS 2002*, 2002.

[83] S. Herrmann, R. Sasportas, H. Mooshofer, J.-C. Klein, F. Meyer und W. Stechele, "Application of recursive methods for the object based video processing and feature extraction," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'99*, p. 121, Juni 1999.

[84] F. Meyer, "An Optimal Algorithm for the Watershed Line," *Dans RFIA*, pp. 847–857, 1991.

[85] M. Stuck, "Entwurf einer Hardwarearchitektur für hierarchische LIFOs," *Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 2001.

[86] T. Thile, "Entwurf eines konfigurierbaren Verarbeitungsteils für einen Coprozessor zur Segmentierung und Bildanalyse," *Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 1998.

[87] N. O'Connor, T. Adamek, S. Sav, N. Murphy und S. Marlow, "QIMERA: A Software Platform for Video Object Segmentation and Tracking," *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS'03*, 2003.

[88] Sun Micro Systems, "UltraSPARC Processors Family Brochure," *http://www.sun.com/processors/feature/USFamilyBrochure_FINAL.pdf*, 2004.

[89] H. Dietrich, "Bewegungsanalyse und Zusammenfassung von Segmenten in Video-Sequenzen," *Diplomarbeit am Lehrstuhl für Integrierte Schaltungen der Technischen Universität München*, 1998.

[90] J. Shi und C. Tomasi, "Good features to track," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR94*, pp. 593–600, 1994.