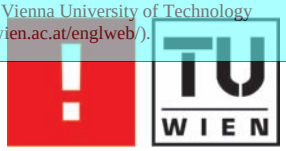

FAKULTÄT FÜR !NFORMATIK

Transformation von nicht-photorealistischen Bild- zu Videoeffekten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Medieninformatik

eingereicht von

Nicole Brosch
Matrikelnummer 0325237

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuerin: Ao. Univ. Prof. Mag. DI Dr. Margrit Gelautz
Mitwirkung: DI Jürgen Platzer

Wien, 25. September 2009

(Unterschrift Verfasserin) (Unterschrift Betreuerin)

Transformation von nicht-photorealistischen Bild- zu Videoeffekten

DIPLOMARBEIT

25. September 2009

Nicole Brosch
1180 Wien, Wallrißstraße 64/6
nici.brosch@gmail.com

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. September 2009 _____

(Unterschrift Verfasserin)

In dieser Arbeit wird nach Luise F. Pusch [68] geschlechtergerecht formuliert.

Abstract

This diploma thesis deals with the automated production of non-photorealistic animations from real video sequences. Stylization of video data poses the challenge of producing temporally coherent results. Video objects that are present in consecutive frames should be experienced as a functional unit, to enable users to regard the video as an animation, with disturbing artifacts such as flickering largely suppressed. Therefore the work starts with a survey of different approaches described in the literature to enforce temporal coherence, and we put an emphasis on cartoon-like effects operating on image-space data.

The main contribution of the diploma thesis is to enlarge and enhance a library of image and video processing algorithms, which is already in use for educational purposes. In this context, a new cartoon-like image effect and, building on that, several video effects are developed. Existing data structures are enhanced, and keyframe interpolation as well as various image and video processing algorithms are added to the library.

The implemented image effect creates a new style orientated on basic characteristics of cartoons and gives the users the opportunity to modify the resulting abstraction. For this purpose, a color reduction is first applied to flatten details. Additional steps (highlighting edges, adding brightness and color detail, modifying image sharpness or contrast) offer parameters to control the influence of several characteristics of the result.

The respective video effects refer to different approaches described in the literature to enhance temporal coherence. One of those video effects is based on techniques which process a video stream globally and attempt to include more than only one frame in the color reduction process. Another video effect relates colors of adjacent frames. A third video effect defines, inspired by approaches working with frame differences, a decision function to reduce flickering.

A variety of stylization results are illustrated and evaluated. To verify and compare the effectiveness of the implemented effects in combination with different techniques, frame differences are analyzed. Finally, the additional benefit of the framework and, in particular, of the added effects for educational purposes is discussed.

Kurzfassung

Das automatische Erzeugen von nicht-photorealistischen Animationen aus echten Videosequenzen ist ein aktuelles Problem der Computergraphik mit starker anwendungsorientierter Relevanz. Wird dabei von Daten im Bildraum ausgegangen, erstreckt sich das Forschungsgebiet auch über das Feld der Mustererkennung und der digitalen Bildbearbeitung. Beim Stilisieren einer Videosequenz besteht eine große Herausforderung darin, zeitlich kohärente Ergebnisse zu erzeugen. Die in den Frames dargestellten Videoobjekte sollen trotz zeitlicher Veränderungen als zusammengehörige Einheit empfunden werden können, sodass die Sequenz in weiterer Folge als Animation wahrnehmbar ist. Die vorliegende Arbeit gibt daher zunächst einen Überblick über verschiedene Ansätze Kohärenz zu erzwingen und geht dabei im Speziellen auf cartoonartige Stilisierungen im Bildraum ein.

Das primäre Ziel der Diplomarbeit ist es, eine bereits in der Lehre eingesetzte Bibliothek für Bild- und Videoverarbeitungsalgorithmen zu erweitern. Dafür wurden ein neuer, cartoonartiger Bildeffekt und darauf aufbauend mehrere Videoeffekte implementiert. Im Zuge dieser Änderungen werden außerdem in der Bibliothek existierende Datenstrukturen erweitert und um Keyframeinterpolation sowie weitere Bild- und Videoverarbeitungsroutinen ergänzt.

Der implementierte Bildeffekt entwickelt einen an Basiseigenschaften von Cartoons orientierten Stil, den BenutzerInnen zusätzlich variieren können. Zu diesem Zweck führt eine erste Abstraktion eine Farbreduktion aus und entfernt Details. Weitere Schritte (Betonung von Kanten, Modifikation von Helligkeits- und Farbwerten, Variation von Schärfe und Kontrast) schaffen gleichzeitig Parameter, welche die Ausprägung der einzelnen Merkmale im Ergebnis bestimmen.

Die Videoeffekte bauen auf unterschiedliche, in der Literatur präsentierte Strategien für den Erhalt zeitlicher Kohärenz auf. Ein Videoeffekt versucht, angelehnt an global konzipierte Ansätze, mehr als nur ein Frame in die Farbreduktion einzubeziehen. Ein weiterer Effekt stellt eine Beziehung zwischen den Farben benachbarter Frames her. Ein dritter Videoeffekt führt, inspiriert durch Ansätze, die aufgrund von Differenzen zwischen benachbarten Frames Arbeitsschritte definieren, eine Entscheidungsfunktion ein, die darauf abzielt, räumliche Farbsprünge zu verhindern.

Um die Effektivität der implementierten Effekte zu verifizieren und zu vergleichen, werden Differenzen zwischen Frames untersucht. Ein zusätzlicher, bereits existierender Videoeffekt wird adaptiert und in die Vergleiche eingebunden. Schlussendlich wird der Mehrwert des Frameworks und insbesondere der implementierten Effekte für den Lehreinsatz analysiert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Problemstellung

Cartoons und anders stilisierte nicht-photorealistische Animationen werden seit Anfang der Filmgeschichte studiert und weiterentwickelt. Stop-Motion, 2D- oder 3D-Computeranimation sowie experimentellere *Found Footage* Techniken werden sowohl in der Entertainment Industrie für Trickfilme, Musikvideos, (Fernseh-)Werbung oder Ähnliches als auch in der Visualisierung für die vereinfachte Darstellung von Inhalten in der Lehre oder auch zur Erstellung von animierten Grafiken eingesetzt. Diese Vorgehensweisen setzen in der Regel viel Handarbeit, künstlerische Begabung, Equipment und Kenntnis im Umgang mit Programmen, beispielsweise für die Modellierung von Charakteren, voraus. Gerade im Feld der professionellen, zweidimensionalen Animation produzieren viele Studios noch per Hand und erstellen beginnend bei *Story Board* und *Layouts* über *In-Betweening* bis zur finalen Aufnahme, Schritt für Schritt dem traditionellen Animationsprozess folgend, in Teams von 50 bis 300 Personen, tausende Zeichnungen, um eine Episode einer Zeichentrickserie zu produzieren [14].

Aber auch computergestützte Verarbeitung, die beispielsweise seit 1987 von Walt Disney eingesetzt wird, bringt viel Aufwand mit sich. Es müssen zumindest Entwürfe der Schlüsselpositionen einer Animation gezeichnet und in ein System importiert sowie *Templates* für die Bewegung der Charaktere und *Layer* erstellt werden. Ein Zeichentrickfilm bleibt somit, trotz der effizienteren Ausführung der traditionellen Arbeitsschritte am Computer, vor allem Handarbeit, die entweder auf einem Grafiktablett oder ganz klassisch mit Stift und Papier bzw. Folie durchgeführt wird [14]. Neben der Möglichkeit, die Schritte des Animationsprozesses am Computer auszuführen, existieren weitere Hilfestellungen für einzelne Arbeitsgänge, so können zum Beispiel rotoskopische Interfaces die Erstellung komplexer Layouts vereinfachen.

Diese Diplomarbeit beschäftigt sich damit, den enormen Arbeitsaufwand und sowohl technische als auch künstlerische Vorkenntnisse bei der Erstellung von Animationen zu mini-

mieren und geht dabei nicht von gezeichneten, sondern von gefilmten Bildern einer Videosequenz, sogenannten *Frames*, aus. Die Anwendung eines Videoeffekts soll dieses Grundmaterial entsprechend einfacher Parameter automatisch stilisieren, sodass das Ergebnis animiert wirkt.

Dieser bekannte Ansatz wirft weitere Fragen auf und stellt neue Herausforderungen. Aus der künstlerischen Perspektive betrachtet, muss der individuelle Stil eines Menschen kopiert oder ein neuer kreiert werden, ein Prozess, der nur schwer zu automatisieren ist. Charakteristische Merkmale des Stils haben, neben dem dramaturgischen Aufbau, großen Einfluss auf die emotionale Gestaltung einer Szene. Durch Farbwahl, Grad der Abstraktion, Kontraste, Schärfe und vieles mehr beeinflussen sie die Bildkomposition, welche allerdings auch durch die Kameraführung und andere Komponenten bestimmt wird. [63]

Technisch gilt es vor allem die zeitliche Kohärenz zu wahren. Es genügt nicht, die einzelnen Bilder unabhängig von einander oder ihrem Inhalt zu stilisieren, da das zu unkontrolliertem Flimmern und Flackern führen würde. Aus diesem Grund müssen, um brauchbare Ergebnisse zu erzielen, Methoden eingesetzt und entwickelt werden, die diese Fehler verhindern.

Die Transformation von Bild- zu Videoeffekt, welche neben der Bild- und Videoverarbeitung auch Themen der Mustererkennung bzw. *Computer Vision* sowie der Computergraphik bzw. des *nicht-photorealistischen Renderings* streift, und die damit einhergehenden Herausforderungen sind ein aktuelles Forschungsgebiet. Es existieren bereits unterschiedliche Ansätze, Zeichenstile zu adaptieren und dabei zeitlich kohärent zu arbeiten. Dazu gehören unter anderem animierte impressionistische [52][34] oder kubistische Gemälde [43], Aquarelle [46][82] oder Mosaike [16][17]. Besonders interessant sind cartoonartige Videoeffekte [93][9], die ein Ergebnis möglichst nahe am Zeichentrickfilm anstreben. Sie treten im Allgemeinen durch Komplexität und Rechenaufwand hervor. Diese Punkte führen wie die vordergründige Funktionalität des Effekts und das Ringen nach einem hochwertigen Resultat zu weiteren interessanten Untersuchungspunkten.

1.2 Ziele und Beitrag

Bei der Transformation von Bild- zu Videoeffekt können allgemein als Hauptziele der Erhalt von zeitlicher Kohärenz und die Erzeugung eines visuell ansprechenden Ergebnisses formuliert werden. Barrieren wie Arbeitsaufwand und künstlerische Begabung sollen gelockert werden, sodass zumindest für den privaten Bereich das Erstellen einer Animation erleichtert wird.

Ein Großteil der existierenden Ansätze bewegt Effektprimitive entsprechend eines Vektorfelds, welches die Bewegung der Objekte im Video beschreibt. Sie versuchen dieses, erstmals von Litwinowicz [52] präsentierte, mächtige und gleichzeitig einfache Konzept durch Einführung neuer Parameter [28][34], statistische Berechnungen [46] oder eine globalere Anwendung dieses Prozesses [82][43] zu verbessern. Die diesem Ansatz folgenden Systeme kämp-

fen meist mit dem selben Problem, nämlich der Ungenauigkeit der Schätzungsverfahren für die Berechnung des Vektorfelds. In Ermangelung einer generellen Lösung für die Ermittlung eines fehlerfreien Bewegungsfelds kann nur versucht werden, die dadurch entstehenden Inkonsistenzen zu reduzieren.

Die zweite große Gruppe von Ansätzen nimmt sich dem Problem auf deutlich komplexere Weise an. Anstelle der fix vorgegebenen Effektprimitive treten Videoobjekte mit semantischer Bedeutung, die im Allgemeinen durch zusätzliche manuelle Intervention extrahiert und von Frame zu Frame verfolgt werden. Oft beschränken sie BenutzerInnen in der Wahl des Videomaterials und fordern eine gleichbleibende Kameraposition, die Vermeidung von Verdeckungen oder die Wahl möglichst kontrastreicher Hintergründe [49][1]. Auch wenn dadurch das Ergebnis verbessert werden kann, sind die Schätzungsverfahren für die Verfolgung einzelner Elemente wieder ein nur schwer überwindbares Hindernis.

Diese Diplomarbeit präsentiert Ansätze, zeitliche Kohärenz zu erzwingen, und versucht dabei cartoonartige Videoeffekte zu erzielen. Diese und der Bildeffekt, von dem sie sich ableiten, werden im Rahmen der *Video Effect Library*, einer Sammlung von Bild- und Videoverarbeitungsalgorithmen, die in der Laborübung „Videoverarbeitung“ verwendet wird, implementiert und sollen diese inhaltlich ergänzen. Da in bisherigen Videoeffekten die Robustheit durchwegs stark von Schätzungsverfahren zur Berechnung von Bewegung abhängt, wird versucht, bewusst auf diese zu verzichten und mit anderen Methoden (z.B. dem Mitteln von Farben) über die Frame-zu-Frame Anwendung eines Bildeffekts hinauszugehen. Ähnlich dem zuerst angesprochenen Konzept (Verschiebung von Effektprimitiven) wird auch in dieser Arbeit ein möglichst einfacher bzw. zu studentischen Vorkenntnissen passender Lösungsweg angestrebt, um den Videoeffekt gegebenenfalls in der Lehrveranstaltung „Videoverarbeitung“ als Lehrmaterial einsetzen zu können.

Trotz der automatischen Verarbeitung der Videosequenz beabsichtigt diese Arbeit BenutzerInnen Freiheiten bei der Variation von Ergebnissen zu bieten. So soll das Anpassen verschiedener Parameter, insbesondere die Anzahl der vorkommenden Farben, das Ergebnis von Bild- und Videoeffekten maßgeblich beeinflussen können. In diesem Zusammenhang muss das Framework, in das die Effekte implementiert werden, um das Setzen mehrerer Keyframes in einer Videosequenz und die Interpolation der gewählten Parameter erweitert werden.

Um diese Ziele zu erreichen, werden zuerst ein Bildeffekt und darauf aufbauend unterschiedliche Videoeffekte implementiert. Nach der obligatorischen Farbreduktion, welche die für Cartoons typischen großen einfärbigen Flächen produziert, erfolgt die Verarbeitung in mehreren optionalen Schritten. Einer davon setzt ein weiteres charakteristisches Merkmal, die Konturen, um. Sie werden mit Hilfe von Kantenerkennung, automatischer Schwellenwertselektion und Schärferegulierung ermittelt.

Das Problem der zeitlichen Kohärenz wird mittels Vermeiden von Farbsprüngen im Er-

gebnis adressiert. Es wird angenommen, dass innerhalb der, durch die Farbreduktion entstehenden, einfärbigen Flächen wenig und eher langsame Bewegung stattfindet. Für die Transformation von Bild- zu Videoeffekt wird deshalb die Farbreduktion mit einer auf globalen Entscheidungen basierenden Heuristik stabilisiert, sodass ähnliche Bilder mit gleichen Parametern ähnliche Ergebnisse zur Folge haben. In Anlehnung an global konzipierte Ansätze wird versucht in einem Videoeffekt mehr als nur ein Frame in die Farbreduktion einzubeziehen. Bei einem zweiten Videoeffekt wird mittels Distanzmatrix eine Beziehung zwischen den Farben in benachbarten Frames hergestellt. Ein dritter Videoeffekt führt, inspiriert durch Ansätze, die aufgrund von Differenzen adjazenter Frames Transformationen definieren, eine Entscheidungsfunktion ein, welche darauf abzielt räumliche Farbsprünge zu verhindern. Zwei weitere, experimentellere Videoeffekte bieten Spielraum für Versuche. So wird überprüft, ob eine Verarbeitung im HSB anstelle des RGB Farbraums Vorteile bringt oder wie das Einbinden von Bewegungsinformation mit ihrer Fehleranfälligkeit die Qualität der Farbreduktion beeinflusst.

Zu Vergleichszwecken und als weitere Ergänzung der Bibliothek wird zudem die von Winnemöller et al. [93] beschriebene Videoabstraktion adaptiert, welche verspricht auch ohne zusätzliche Transformationsschritte bei einer Frame-zu-Frame Anwendung zeitlich kohärente Ergebnisse zu liefern. Als Abschluss wird nicht nur die visuelle Qualität der Ergebnisse, sondern auch ihre Brauchbarkeit für den Lehreinsatz evaluiert.

1.3 Struktur der Arbeit

Die Arbeit ist in sieben Kapitel gegliedert, wobei das zweite einen Überblick über bereits existierende Ansätze gibt und diese einander gegenüberstellt. Es wird eine Reihe an Versuchen, die aus einem Video automatisch Animationen erzeugen, präsentiert. Dabei sollen im Speziellen verschiedene Techniken Erwähnung finden, die über die Frame-zu-Frame Anwendung eines Bildeffekts hinausgehen und ein zeitlich kohärentes Ergebnis liefern. In diesem Zusammenhang werden auch cartoonartige Effekte sowie die damit verwandte Rotoskopie genauer betrachtet.

Im Zuge dieser Recherchen werden Algorithmen für die Verbesserung der zeitlichen Kohärenz entwickelt. Im Rahmen eines in der Lehre („LU Videoverarbeitung“) verwendeten Frameworks, der Video Effect Library (VEL), werden ein neuer Bildeffekt und darauf aufbauend mehrere Videoeffekte implementiert. Außerdem wird zu Vergleichszwecken ein weiterer Effekt adaptiert, welcher verspricht, auch ohne zusätzliche Transformationsschritte flüssige Ergebnisse zu liefern. Das dritte Kapitel beschäftigt sich mit diesen Bild- und Videoeffekten sowie ihren Algorithmen.

Da es sich um ein stark visuelles Thema handelt, werden im vierten Kapitel dieser Arbeit Ergebnisse der Effekte präsentiert und gegenübergestellt. Darüber hinaus bietet das Kapitel Raum für experimentelle Modifikationen der entwickelten Algorithmen, Effektvariationen und

visuelle Beispiele der Modifikation von Parametern.

Das fünfte Kapitel gibt einen Einblick in die aktuelle Version der Implementierung der Bild- und Videoeffekte sowie des Frameworks. Es wird auf Funktionalität und Zusammenspiel der relevanten Klassen eingegangen. Zusätzlich werden Ergebnisse von Laufzeitmessungen wichtiger Methoden zusammengefasst.

Der Einsatz des Frameworks und der darin befindlichen Bild- und Videoverarbeitungsalgorithmen in der Lehre legt es nahe, auch die im Zuge dieser Arbeit implementierten Effekte bezüglich des Lehreinsatzes zu evaluieren.

Die Arbeit schließt mit einem umfassenden Fazit und Ideen für zukünftige Weiterentwicklungen.

Kapitel 2

Transformation von Bild- zu Videoeffekt

Dieses Kapitel bietet eine Übersicht und Gegenüberstellung verschiedener Algorithmen im Bereich des *Non-Photorealistic Rendering* (NPR) und der automatischen Rotoskopie [37], welche versuchen zeitlich kohärente Ergebnisse zu erzielen. Deren Funktionsweisen sowie Vor- und Nachteile vorhandener Systeme stellen die Basis für die Entwicklung eigener Algorithmen dar.

2.1 Nicht-photorealistische Bild- und Videoeffekte

Nicht-photorealistische Effekte haben, im Gegensatz zu photorealistischen Effekten, das Ziel existierende, künstlerische Stile zu adaptieren oder neue zu kreieren. Sie verändern das gegebene Grundmaterial (z.B. Bild, Video oder 3D Szene) beispielsweise so, dass es wie ein Gemälde (z.B. [60] oder [26]), Bleistiftskizze (z.B. [27]) oder Mosaik (z.B. [53]) aussieht. Um künstlerische Effekte zu erzielen, wird unter anderem das Grundmaterial aus Primitiven (z.B. Pinselstriche [52] oder Polygone [53]) aufgebaut, Zufälligkeit hinzugefügt (für Rauschen, Verformungen, z.B. [34]) oder eine Technik wie Kantenerkennung angewandt (z.B. [11]) [48]. Die Verarbeitung erfolgt dabei automatisch, halbautomatisch oder interaktiv (z.B. [26]). In den späten Achtzigern und frühen Neunzigern erschien hauptsächlich Literatur, die sich mit der Modellierung künstlerischer Medien beschäftigte und Intervention von Seiten der BenutzerInnen benötigte (z.B. [26]) [7]. Ein Beispiel für ein solches System ist das, im Jahre 1990 präsentierte, von Haeberli [26]. Darin werden die Farben zwar aus einem als Vorlage dienenden Bild extrahiert, das Effektbild aber weitgehend von BenutzerInnen selbst auf eine anfänglich leere Fläche gezeichnet [26]. In den späten Neunzigern wurde der Wunsch nach rein automatischer Verarbeitung des Grundmaterials stärker, sodass Systeme aufkamen, die allein vom Bildinhalt des Inputs gesteuert wurden [48]. Litwinowicz [52] präsentierte beispielsweise 1997 den ersten automatischen Effekt, der nur mit Hilfe von (statischen 2D-) Bilddaten ein sich am Ausgangsmaterial orientierendes Effektbild aus Pinselstrichen aufbaut. Der zugehörige Algorithmus und

seine Erweiterung werden in einem späteren Abschnitt (siehe Abschnitt 2.2.2) genauer erläutert. Sind, wie beim eben genannten Effekt [52], Grundmaterial und Ergebnis jeweils ein Bild, wird vom Prozess der Stilisierung im Allgemeinen als *Bildeffekt* gesprochen. Im selben Paper [52] erweitert Litwinowicz seinen Bildeffekt (erstmals) für die Anwendung auf Videos, was einen weiteren, noch immer anhaltenden Trend in der NPR Literatur darstellt (z.B. [93], [71], [82]). Haben Grundmaterial und Ergebnis, wie bei dieser Erweiterung, zusätzlich eine zeitliche Komponente, kann die stattgefundene Verarbeitung als *Videoeffekt* bezeichnet werden. Videoeffekte können beispielsweise in der Postproduktion für die Stilisierung von Filmmaterial angewendet werden.

Existierende NPR Methoden verarbeiten hauptsächlich statische, zweidimensionale Bilder (und verschiedene Parameter) zu einem Effektbild. Wenn diese Bildeffekte Frame für Frame auf Videos oder Bildsequenzen angewendet werden, laufen ihre Ergebnisse im Allgemeinen nicht flüssig [43]. Die zeitliche Kohärenz geht verloren, sodass die Frames des Videos nicht mehr einheitlich wirken. Plötzliche Änderungen der Effektprimitive, auf der Bildebene fixierte Artefakte (z.B. *shower door effect* [82][60]) und andere ungewollte Effekte stören den Fluss der Bilder und lenken vom eigentlichen Inhalt der Bildsequenz ab. Der Zusammenhang zeitlicher Veränderungen der Objekte geht verloren, sodass das Ergebnisvideo nicht als Animation wahrgenommen werden kann. Um visuell ansprechende und zeitlich kohärente Videoeffekte zu erzeugen, bedarf es also in der Regel zusätzlicher Schritte, welche über die Frame-zu-Frame Anwendung eines Bildeffekts hinausgehen. Diese Schritte und die Videoeffekte, die sie einsetzen, sind das Thema der folgenden Abschnitte.

2.2 Kohärenz bei Videoeffekten

In den folgenden Abschnitten dieses Kapitels werden verschiedene Algorithmen vorgestellt, die bei der Transformation von Bild- zu Videoeffekt versuchen, die angesprochenen Artefakte (z.B. plötzliche Änderungen oder Flimmern) zu vermeiden. Gängig sind zum Beispiel folgende Mittel zur Verbesserung der zeitlichen Kohärenz:

- Das Binden der Animation an Oberflächen durch geometrische Informationen. [60][83] [27][53] [41]
- Die Verwendung des *Optical Flows* als zeitliche Komponente. [52][43][16][34]
- Das Identifizieren und Aktualisieren von veränderten Bereichen in Nachbarframes (*Frame Differencing*). [34]

Der erste Punkt bezieht sich hauptsächlich auf NPR im Objektraum, also dreidimensionale Szenen wie Modelle, und zweidimensionale Szenen mit geometrischen Informationen der einzel-

nen Objekte [53] oder zusätzlicher Tiefeninformation [41]. Der zweite und der dritte Punkt gehen vom Bildraum (2D), also Bildern oder einzelnen Frames, aus. Diese Thematik stellt den Schwerpunkt der Diplomarbeit dar. Die implementierten Videoeffekte (siehe Kapitel 3.2.3 und Kapitel 4.1.1) nehmen Bezug auf Ansätze, die mit Frame Differencing und in einem experimentelleren Effekt mit Optical Flow arbeiten.

Die beiden aufgezählten NPR Techniken im Bildraum sind nicht in der Lage, komplizierte Videoeffekte wie automatische Rotoskopie [71], *Video Tooning* [77] oder das Erzeugen von Bewegungslinien [70] zu fassen. Erst das Hinzufügen eines weiteren Punkts, dem

- Erhalt der zeitlichen Kohärenz durch Vektorisierung [1][49][9][77],

zu der bereits begonnenen Liste, vervollständigt diese mit Effekten, die *high level* Videoanalyse bedingen. Objekte mit semantischer Bedeutung (also Objekte mit Entsprechungen in der realen Welt, z.B. ein Ball) sollen dazu aus dem Video (durch Segmentierung) extrahiert und von Frame zu Frame verfolgt (*getrackt*) werden [72]. Auch dieser Punkt ist für die implementierten Effekte (siehe Kapitel 3) relevant. Das Segmentieren des Grundmaterials ist bei den entwickelten Algorithmen ein wichtiger Schritt zur Identifizierung zusammengehöriger Regionen und der Erzeugung einer cartoonartigen Darstellung.

2.2.1 Oberflächenbindung

In der Computergraphik-Literatur findet man eine Vielzahl von Effekten, die Kohärenz durch Oberflächenbindung erzwingen und Aufschlüsse für die Transformation von Bild- zu Videoeffekt liefern. Dazu gehören unter anderem Effekte, die 3D Szenen wie Gemälde (*Painterly Rendering* [60], Aquarelle [83]) oder handgezeichnet (*Pen and Ink Rendering* [27]) aussehen lassen. Auch zweidimensionale Szenen mit zusätzlichen geometrischen Informationen der einzelnen Objekte können als Quelle für Effekte herangezogen und ihre Oberflächeninformation ausgenützt werden [53]. Neben der bereits angesprochenen Differenz des Grundmaterials (Objektraum nicht Bildraum) gibt es einen weiteren gravierenden Unterschied, welcher das Konzept der Kohärenz durch Oberflächenbindung von anderen abhebt. Die exakte Bewegung der Objekte in einer Animation ist schon vorher bekannt und muss nicht, wie im zweidimensionalen Fall, mit Schätzungsverfahren aus dem Video berechnet werden. Der im folgenden beschriebene Algorithmus von Meier [60] ist ein Beispiel für einen Effekt, der mit Oberflächenbindung arbeitet.

Partikelsystemgeneriertes Painterly Rendering [60] Der Erhalt der zeitlichen Kohärenz wird in [60] durch den Einsatz von *Partikel Rendering Methoden*, wie im folgenden Pseudocode beschrieben, erreicht.

Abbildung 2.1: Diese Abbildung zeigt drei Frames aus [60], welche mit dem System Meier [60] erzeugt wurden.

Listing 2.1: Pseudocode zur Oberflächenbindung von Partikeln

```
1  Erzeuge Bildrepräsentation durch Partikel;
2  Für jedes Frame der Animation {
3          Erzeuge Referenzbild mit geometrischer Information;
4          Transformiere Partikel entsprechend der Animationsparameter;
5          Sortiere Partikel nach Distanz vom Sichtpunkt;
6          Für jedes Partikel {
7                  transformiere auf Bildschirmraum;
8                  Attribute der Pinselstriche ermitteln;
9                  Zeichne;
10         }
11 }
```

In einem ersten Schritt werden *Partikel Sets*, die verschiedene geometrische Objekte wie z.B. eine Ebene repräsentieren, generiert. Die Ausgangsfläche wird dazu in Dreiecke zerlegt, welche mit *Partikeln*, also Elementen mit zeitlich veränderbaren Eigenschaften, gefüllt werden (siehe Zeile 1 in Listing 2.1). Die einzelnen Partikel werden anschließend in Bildschirmkoordinaten transformiert und nach der Distanz vom Sichtpunkt aus sortiert. Jeder Pinselstrich rendert aufgrund der Eigenschaften des Referenzbilds (z.B. Farbe an der entsprechenden Position) einen Partikel. Zusätzliche Parameter wie die Größe der Pinselstriche oder deren Texturen können von BenutzerInnen beeinflusst werden und ermöglichen es, verschiedene Effektvariationen zu erstellen. Abbildung 2.1 zeigt ein mögliches Ergebnis.

Dadurch, dass Partikel und damit auch die gerenderten Pinselstriche jeweils an ihre fixe Position auf einem dreidimensionalen Objekt gebunden sind, läuft die Animation flüssig. Die hier beschriebene Vorgehensweise dient als Grundlage für die Entwicklung von auf Optical Flow basierenden Videoeffekten (siehe Abschnitt 2.2.2) und ähnelt damit konzeptuell vielen dieser Ansätze.

2.2.2 Optical Flow

Dieser Abschnitt befasst sich mit der Transformation von Bild- zu Videoeffekt (im Bildraum) unter zu Hilfenahme des Optical Flows, welcher Bewegungsrichtung und Geschwindigkeit der einzelnen Bildpunkte angibt, sodass für jeden Übergang von Frame zu Frame bekannt ist, welcher Bildpunkt des ersten Frames sich an welche Stelle des zweiten Frames bewegt hat. Die zwei grundsätzlichen Möglichkeiten, diese Vektorfelder zu schätzen, sind Differenzialtechniken (Ableitungen der Bildintensitäten) und Vergleiche von Regionen in benachbarten Frames. Da die detaillierte Beschreibung der verschiedenen Ansätze und Algorithmen den Rahmen der Diplomarbeit sprengen würde, werden an dieser Stelle an vertiefender Information interessierte LeserInnen auf den Artikel von Barron et al. [19] verwiesen, welcher einen ausführlichen Überblick über diverse Techniken bietet. Grob zusammengefasst verläuft die Ermittlung des Optical Flows in folgenden drei Schritten [19]:

- Rauschunterdrückung
- Extraktion der Grunddaten (Ableitungen oder Regionen)
- Ableitung eines zweidimensionalen Vektorfelds und zu diesem Zweck Treffen von Annahmen über das Bewegungsfeld

Ein wesentlicher Parameter zur Erhaltung der zeitlichen Kohärenz im Ergebnisvideo ist die Qualität des gewählten Schätzungsverfahrens für die Berechnung von Bewegung. Im Allgemeinen liegen dessen Schwächen bei schwankenden Lichtverhältnissen und nicht texturierten Regionen. Aus Berechnungsfehlern bei der Schätzung resultieren zusätzlich Folgefehler, die sich im Video als unkontrollierte Bewegung und Flackern (*swimming*) äußern [7]. Nehmen diese Fehler überhand, so besteht laut Prägnanzgesetz der Gestalttheorie [91] die Gefahr, dass Objekte von BetrachterInnen nicht mehr als solche wahrgenommen werden. Da es keine allgemeine Lösung für die Berechnung eines fehlerfreien Bewegungsfelds gibt (*correspondence problem*), kann nur versucht werden diese Inkonsistenzen zu minimieren.

Dieser Abschnitt demonstriert außerdem beispielhaft verschiedene Möglichkeiten zur Erzeugung nicht-photorealistischer Videoeffekte. Während einige Effekte sequenziell an das Problem herangehen [52][34][28], verarbeiten andere das Video als eine zusammengehörige Einheit [16] oder versuchen einen Mittelweg zu finden [82]. Der Großteil der hier besprochenen Videoeffekte setzt den Optical Flow in einem Zwischenschritt ein, welcher eine Verbindung zur zeitlichen Dimension darstellt. Die Verarbeitung beginnt in der Regel mit der Anwendung eines statischen Bildeffekt Algorithmus auf das erste Frame und wird in einer durch das Vektorfeld abgewandelten Form weitergeführt. Besonders beliebt ist dieser Ansatz im Bereich des *Painterly Rendering*, bei dem Pinselstriche entlang des Vektorfelds von Frame zu Frame bewegt

werden [52][34][28]. Einige Verfahren des Painterly Rendering werden im Folgenden genauer beschrieben.

Painterly Rendering

Mit Painterly Rendering wird die, erstmals von Litwinowicz [52] für Videos erweiterte, Methode bezeichnet, welche Bilder in eine Menge von Pinselstrichen transformiert. Werden die einzelnen Frames eines Videos unabhängig von einander verarbeitet, sind die Pinselstriche fix auf der Bildebene verteilt. BetrachterInnen bekommen den Eindruck als würde die Bewegung hinter einem „Duschvorhang“ stattfinden (*shower door effect* [60]). Das zentrale Problem bei der Erweiterung von Bild- zu Videoeffekt ist die Platzierung dieser Pinselstriche im zeitlichen Verlauf, um diesen Eindruck zu verhindern. Die Effektprimitive sollten sich im Idealfall mit den Objekten einer Szene mitbewegen, was durch die Bewegung der Pinselstriche entlang des Optical Flow Vektorfelds erreicht werden soll.

Dieser Lösungsansatz wurde von unterschiedlichen Systemen auf verschiedene Weise eingesetzt. Das ist auch bei folgenden Transformationen von Bild- zu Videoeffekten der Fall:

- Impressionistischer Effekt von [52]
- Interaktives Painterly Rendering von [34]
- Layerbasiertes Zeichnen von [28]
- Paintbrush Transformation von [46]

Diese Systeme und ihre Varianten des Lösungsansatzes sollen anschließend näher betrachtet werden.

Impressionistischer Effekt [52] Der impressionistische Effekt von Litwinowicz [52], welcher erstmals automatisch und rein datenabhängig auf der Bildebene arbeitete, übersetzt im Bildeffekt ein Originalbild in mehrere gleich große Pinselstriche und konzentriert sich aus künstlerischer Perspektive auf deren charakteristische Merkmale, welche sich auch in der Definition ihrer Parameter widerspiegeln:

Radius zur Steuerung der Quantifizierung bzw. des Detailgrads

Position im Bild

Richtung normal zum lokalen Bildgradienten oder fixer Wert

Farbe interpolierte Farbe des Originalbilds an dieser Position

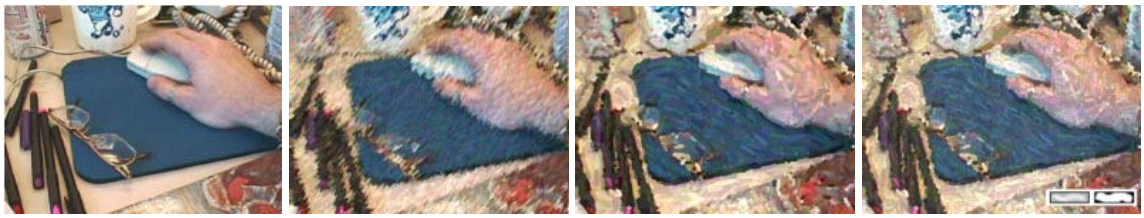

Abbildung 2.2: Ergebnisse bei verschiedenen Einstellungen des impressionistischen Effekts aus [52]. Das *erste Bild* zeigt das Originalbild. Das *zweite Bild* das Ergebnis mit der fixen Pinselstrich-Richtung von *45* Grad, und das *dritte Bild* resultiert aus der Berechnung von Richtungen normal zum Gradienten. Im *vierten Bild* wurde eine Textur als Pinselstrich verwendet.

Textur eine Alpha Maske, die dem Pinselstrich eine bestimmte Struktur geben kann

Diese Primitive werden im Effektbild je nach Einstellungen, z.B. zentriert in jedem zweiten Pixel, angeordnet und in einer zufälligen Reihenfolge nach dem Algorithmus in Listing 2.2 gezeichnet. Im Zuge der Verarbeitung werden ihre Parameter modifiziert, um durch Unregelmäßigkeiten ein handgemachtes Aussehen zu erlangen. Bis auf die erwähnten Modifikationen ist ihre Größe und Form innerhalb eines Bilds global fix definiert.

Listing 2.2: Pseudocode zum Impressionistischen Videoeffekt

```
1 Intensitätsbild(Originalbild);
2 GaußFilter(Intensitätsbild); //Glätten
3 SobelFilter(geglättetesBild); //Kantenerkennung
4
5 Solange nicht die maximale Länge oder eine Kante erreicht ist {
6         Zeichne einen Pinselstrich weiter in Gradientenrichtung;
7 }
```

In der Implementierung wird zum Glätten des Intensitätsbilds mit einem Gauß Filter [85] gefaltet. Durch diesen Vorgang werden Rauschen und Details reduziert. In dem daraus resultierenden unscharfen Bild erfolgt anschließend mittels Sobel Filter [85] die Kantenerkennung. Als Ergebnis gibt der Gradient die Richtung der Pinselstriche an (siehe Ziele 5-7 im Pseudocode). Pixel mit einem Gradienten bei oder nahe Null werden durch das Ergebnis der Gradienteninterpolation der Nachbarpixel ersetzt. Abbildung 2.2 zeigt Ergebnisse des Effekts mit verschiedenen Einstellungen.

Der aus diesem Bildeffekt abgeleitete Videoeffekt verarbeitet das Originalvideo sequenziell. Das erste Frame wird wie im Bildeffekt beschrieben abstrahiert. Alle anderen Frames greifen für weitere Berechnungen jeweils auf ihr Vorgängerframe zurück. Inspiriert durch die Oberflächenbindung der Effektprimitive in [60] ist das Ziel dieser Transformation zum Videoeffekt, Pinselstriche mit den Pixeldaten und dadurch auch mit den Objekten mitzubewegen.

Sie werden dem Vektorfeld des Optical Flows entsprechend verschoben und ergeben das neue Frame. Damit das Ergebnisvideo eine annähernd konstante Dichte an Pinselstrichen aufweist (erkannt durch *Delaunay Triangulation* [75]), müssen in leeren Bereichen neue Pinselstriche hinzugefügt und in zu dichten Bereichen vorhandene gelöscht werden.

Es können sich aufgrund der Abhängigkeit von der Qualität der Optical Flow Berechnung schleichend Fehler entwickeln. Erst händisches Nachbearbeiten des Vektorfelds ermöglicht diesem Effekt in der Praxis trotzdem ein befriedigendes Ergebnis zu erzielen. Eine weitere Quelle für Unstetigkeiten ist das Hinzufügen von Pinselstrichen zur Vermeidung von Löchern, welches durch abrupte Änderungen Flimmern verursachen kann. [52]

Dieses einfache und zugleich mächtige Konzept hat die zugehörige Literatur maßgeblich beeinflusst. Als Vorreiter für viele Painterly Rendering- und andere Videoeffekte sind seine Schwächen und ihre Vermeidung ein wichtiges Thema für später entwickelte Ansätze. Durch Modifikationen dieses von [52] im Jahre 1997 präsentierten Algorithmus versuchen die im Anschluss vorgestellten Nachfolger die Nachteile gezielt zu verhindern.

Interaktives Painterly Rendering [34] Einer dieser von [52] inspirierten Algorithmen ist der interaktive Ansatz von [34]. Der ihm zugeordnete Bildalgorithmus [32] verwendet im Gegensatz zum Impressionistischen Effekt [52] verschieden große Pinselstriche R in einem Bild, um die Genauigkeit zu steuern und die Zeichentechnik von KünstlerInnen zu imitieren. Mit großen Pinselstrichen wird anfänglich ein grobes Effektbild gezeichnet, welches später in detailreichen Bildbereichen verfeinert wird.

Listing 2.3: Pseudocode zum Interaktiven Painterly Rendering

```
1  paint(Originalbild, Arbeitsbereich, PinselstrichGrößen R[]){
2          Für jeden Pinselstrich R vom größten zum kleinsten {
3                  GaußFilter (Originalbild); //Pinselstrichgröße
4                  Erzeuge damit einen gleichmäßigen Raster;
5                  Für jede Zelle im Raster {
6                          Berechne Farbdifferenz;
7                          Wenn Farbdifferenz > Schwellenwert T {
8                                  Zeichne den Pinselstrich;
9                          }
10                 }
11         }
12 }
```

Ein weiterer Unterschied zum zuvor beschriebenen Algorithmus ist die Art und Weise, mit der Pinselstriche verarbeitet werden. So produziert hier die Methode „zeichne den Pinselstrich“ (siehe Zeile 8 im Listing 2.3) *Kontrollpunkte eines B-Splines* (formgebende Punkte einer Kur-

Abbildung 2.3: Die *erste Reihe* zeigt Ergebnisse aus [34] eines Videos, das im Gegensatz zur *zweiten Reihe* ohne Optical Flow verarbeitet wurde.

ve) [30], welche normal zur Gradientenrichtung bzw. entlang von Objektkonturen verlaufen und später unter Verwendung der Graphikhardware gerendert werden. Die Farbdifferenz von Originalbild und Pinselstrich wird mit einem Schwellenwert verglichen und damit über die Länge des Pinselstrichs entschieden. [32]

Eine Variation des Bildeffekts entsteht durch das Hinzufügen einer Höhenvariable zu jedem Pinselstrich und eines Höhenfelds für das gesamte Bild. So kann zusätzlich eine Berechnung der Lichtverhältnisse (*bump mapping* [30]) vorgenommen werden [33]. Der Prozess startet bei einem Bild mit der initialen Höhe Null. Die Höhe ändert sich unter anderem durch die Anzahl der an dieser Position bereits gezeichneten Pinselstriche. Für eine genauere Beschreibung dieser Variante siehe [33].

Ein auf dem Bildeffekt aufbauender Videoeffektalgorithmus (Abbildung 2.3, zweite Reihe) zieht das Vektorfeld des Optical Flows in seine Berechnungen mit ein [34]. Die Kontrollpunkte werden mit Hilfe des Bewegungsfelds auf das aktuelle Frame abgebildet und Bereiche, die sich stark vom Vorgängerframe unterscheiden, erneuert. Pinselstriche können sich dadurch, wie in [52], mit dem Objekt mitbewegen und werden immer wieder neu gezeichnet, ohne dass dieser Prozess sichtbar wird. Auf die Identifizierung verschiebungsbedingter Löcher wird hier verzichtet. Durch simples übereinander Zeichnen der Frames scheinen vorhergehende Schichten durch und beugen somit leeren Flächen vor.

Trotz der Einführung verschieden großer, übereinander gezeichneter Pinselstriche in [34] können die Artefakte nicht eliminiert werden. Swimming wird durch die eingeführte Entscheidungsfunktion (siehe Zeile 7 im Listing 2.3) nicht verhindert, sondern nur auf die Bereiche lokalisiert, in denen starke Bewegung stattgefunden hat. Eine alternative Variante des Videoeffektalgorithmus, die ohne Optical Flow auskommt, wird in Abschnitt 2.2.3 beschrieben.

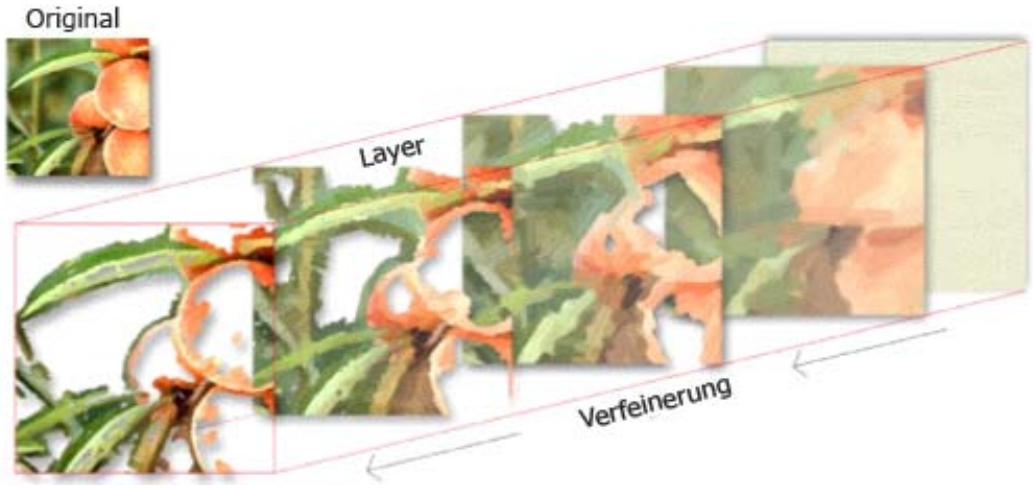

Abbildung 2.4: Diese Abbildung aus [28] visualisiert den Verfeinerungsprozess der Layer. Je größer der Radius der Pinselstriche, desto gröber ist das Effektbild. Durch Hinzufügen von Details, in Form von Layern mit kleineren Pinselstrichen auf bzw. um Kanten, wird das Effektbild verfeinert.

Layerbasiertes Zeichnen [28] Der Effekt von Hays und Essa [28] erweitert die Painterly Rendering Algorithmen von Hertzmann und Perlin [34] sowie Litwinowicz [52] um das Konzept der *Layer* und das der dynamischen Eigenschaften der Effektprimitive. Wie in [34] gibt es Pinselstriche verschiedener (fixer) Größen, die das Bild sukzessive verfeinern (siehe Abbildung 2.4). In [28] werden sie jedoch nur in Bereichen von starken Kanten bzw. Gradienten erzeugt (siehe Abbildung 2.4). Layer, also die Menge aller Pinselstriche mit einem bestimmten Radius, haben zusätzlich zu den in [52] definierten Eigenschaften (Farbe, Richtung, Position, Textur) einige weitere. Die zwei Wichtigsten davon sind Transparenz und Anzahl der vergangenen Frames.

Transparenz ermöglicht unter anderem ein langsames Ein- bzw. Ausblenden, anstelle des abrupten Löschens in [34], und vermindert Flimmern im Ergebnisvideo. Letztere Eigenschaft ist einer der Hauptunterschiede zu den beiden anderen Ansätzen.

Anzahl der vergangenen Frames, die für die Berechnung der Farbe einzelner Pinselstriche gemittelt werden soll.

Bildeffekt und Videoeffekt beim ersten Frame verarbeiten das Bild mit dem selben statischen Algorithmus (ab Zeile 5 im Listing 2.4). Je nachdem welchen *Style* BenutzerInnen wählen, werden die Eigenschaften verschieden initialisiert. Dazu gehören die Anzahl der verwendeten Layer (und damit Pinselgrößen) und ein Schwellenwert, der für die Definition besonders starker Gradienten herangezogen wird. Im weiteren Verlauf wird das Ausgangsbild für jeden Layer eines Frames mittels eines Gauß Filters, dessen Filtergröße dem Radius der Pinsel-

striche entspricht, geglättet. Geglättete Bilder für Layer mit kleinen Pinselstrichen sind somit schärfer als geglättete Bilder für Layer mit großen Pinselstrichen.

Anschließend werden durch Anwendung eines Sobel Filters [85] Gradienten berechnet. Sie dienen im Gegensatz zu jenen in [34] nicht gleich zur Orientierung der Pinselstriche. Der Schwellenwert extrahiert „starke Gradienten“, welche anschließend für alle übrigen Pinselstriche global interpoliert werden. Mit zwischen zehn und 200 solchen Basis-Pinselstrichen soll dieser Ansatz verglichen mit bisherigen Methoden ([34] und [52]) zu einem kohärenteren Ergebnis führen.

Als letzte Schritte, die noch als Teil des Bildeffekts anzusehen sind, werden eine Kantenerkennung durchgeführt und schlussendlich Pinselstriche gezeichnet. Der *Canny Edge Detector* [85] wird dazu auf die zuvor geglätteten Versionen des Frames bzw. für den detailreichsten Layer auf das Frame selbst angewandt. Analog zu [52] werden die Pinselstriche an den Kanten geclippt.

Listing 2.4: Pseudocode zum Layerbasierten Zeichnen

```
1  Für jedes Frame {
2          Ab dem zweiten Frame {
3                  Optical Flow Berechnungen();
4          }
5          Für jeden Layer {
6                  GaußFilter (Frame); //entspricht Pinselstrichgröße
7                  SobelFilter(GaußFrame); //Gradient
8
9                  Wenn Layer < detailliertester Layer{
10                         GaußFilter (Frame);
11                 }
12                 Canny (GaußFrame); //Kantenerkennung
13                 Interpoliere Gradienten der Pinselstriche;
14
15                 Für jeden Pinselstrich {
16                         Ab dem zweiten Frame zum neuen Platz bewegen;
17                         Wenn notwendig entfernen oder hinzufügen;
18                         Wenn notwendig Transparenz ändern;
19                         Bis maximale Länge erreicht ist { //Kanten
20                                 Zeichne den Pinselstrich;
21                         }
22                 }
23         }
24 }
```

Im Videoeffekt folgen, in einem Zwischenschritt (siehe Zeile 16 im Listing 2.4), die Pinselstriche dem Vektorfeld des Optical Flows an ihren Platz im neuen Frame. Liegt die neue Position des Pinselstrichs auf einem „starken Gradienten“, so dient in Zukunft auch dieser als Basis für die Interpolation der Gradienten. Da die Berechnung der Gradienten sehr sensibel gegenüber Rauschen ist, wird im Videoeffekt darauf geachtet, dass sie sich pro Frame höchstens um einen konstanten Wert (z.B. ein Grad) ändern. [28]

Ähnlich dem Impressionistischen Effekt von Litwinowicz [52] versucht auch das Layerbasierte Zeichnen [28] Löcher aufzufüllen oder Redundanz zu entfernen. Gerade beim Hinzufügen und Löschen von Pinselstrichen spielt Transparenz eine wichtige Rolle, da die beiden Vorgänge nicht abrupt erfolgen müssen. Folgende Punkte verdeutlichen wann und wie diese neue Eigenschaft eingesetzt wird:

- Pinselstriche mit einer neuen Position außerhalb des Frames werden gelöscht.
- Fallen die Mittelpunkte zweier Pinselstriche des selben Layers nach der Neupositionierung aufeinander, wird einer der beiden kontinuierlich gelöscht, also von Frame zu Frame immer transparenter, bis er schlussendlich ganz verschwindet. Die Redundanz würde zu Flackern im Ergebnisvideo führen.
- Der unterste (gröbste) Layer wird in einer pseudo-zufälligen Reihenfolge (nicht *scanline*) nach Löchern, also Bereichen, in denen kein Pinselstrich gefunden werden kann, durchsucht und gegebenenfalls ein Pinselstrich mit der Markierung *new* hinzugefügt.
- Höhere, also detailreichere, Layer werden analog nach Löchern durchsucht. Hier dürfen nur Bereiche in der Umgebung von Kanten erneuert werden. Im höchsten Layer werden also neue Pinselstriche nur auf bzw. neben starken Kanten gesetzt (siehe Abbildung 2.4). Sie sind anfangs immer sehr transparent und gewinnen von Frame zu Frame an Deckkraft.
- Entfernen sich Pinselstriche höherer Layer zu weit von der ihnen zugeordneten Kante, werden sie immer transparenter, bis sie vollständig verschwinden.

Wie bereits angesprochen, werden Pinselstriche an den zuvor erkannten Kanten geclippt. Im Videoeffekt ist ihr Wachstum zusätzlich über die Zeit an eine bestimmte Länge gebunden. Zeitliche Inkonsistenzen, welche oft durch Rauschen im Video hervorgerufen werden, würden ohne die zusätzliche Beschränkung des Längenwachstums zu wiederholten und rapiden Änderungen der Länge führen. Die Farbe eines Pinselstrichs wird durch Farben eines Pixels im Frame und dem Mittel seiner Farben in Vorgängerframes berechnet, um plötzliche Farbänderungen zu verhindern. Je nach Style können zufällige, leichte Abweichungen hinzugefügt werden (siehe Abbildung 2.5).

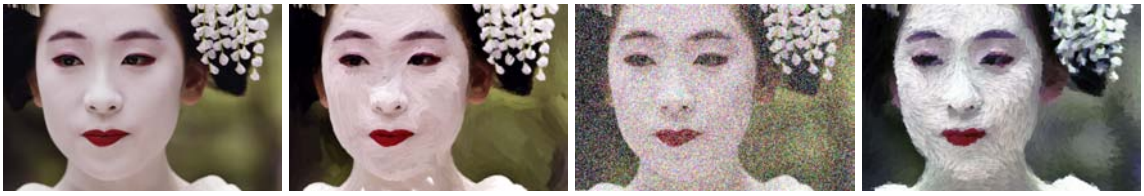

Abbildung 2.5: Diese Abbildung zeigt Original- und Effekt-Bilder aus [29], die durch Layer-basiertes Zeichnen erzeugt wurden. Das *zweite Bild von links* verwendet den Style „Impressionismus“, das *Dritte* „Van Gogh“ und das *Vierte* „Pointillismus“. In den Effekt-Bildern sind die Farbmodifikationen der verschiedenen Styles deutlich zu erkennen.

Diese Abwandlung des initialen Konzepts von [52] ist sehr robust und überwindet die Artefakte, die in vorangegangenen Arbeiten beim Hinzufügen oder Entfernen von Pinselstrichen entstanden sind. Wie ihre Vorgänger kämpfen Hays und Essa [28] allerdings weiterhin mit der Ungenauigkeit von Schätzungsverfahren für die Berechnung des Bewegungsfelds. Bei fehlerhaften Vektoren haften die Pinselstriche nicht mehr an ihren Objekten und produzieren ein unsauberes Ergebnis.

Paintbrush Transformation [46] Eine weitere Modifikation des ursprünglichen Optical Flow Algorithmus und des von Hertzmann und Perlin präsentierten Lösungsansatzes [34] (siehe Abschnitt 2.2.3) wurde von Kovács und Szirányi [46] entwickelt. Sie verwenden im Gegensatz zu bisher vorgestellten Painterly Rendering Ansätzen nicht runde, sondern rechteckige Pinselstriche, welche hierarchisch pro Layer (bzw. Größe) auf ihre berechneten Positionen platziert werden. Zusätzlich können sie optional den Kanten des Originalbilds folgen.

Der initiale Bildeffekt von Szirányi und Tóth [79][78] basiert auf der Konstruktion von starken Kanten, sodass jeder Layer, von einer gegebenen Distanz betrachtet, für das menschliche visuelle System die selbe Szenerie wie das Originalbild (siehe Abbildung 2.6) zeigt. Eine stochastische Entscheidungsfunktion, basierend auf einem globalen Fehler, soll dieses Konzept im Effekt realisieren. Während des Verfeinerungsprozesses wird ein Pinselstrich nur dann gesetzt, wenn dadurch das Effektbild zum Originalbild konvergiert. Dazu wird vor und nach der Platzierung ein Histogramm des geglätteten Fehlerbilds (aktueller Stand der Transformation im Vergleich zum Originalbild) berechnet. Gibt es eine Verbesserung, so wird die Veränderung beibehalten.

Im Videoeffekt kombinieren Kovács und Szirányi [46] die stochastische Paintbrush Transformation mit dem Vektorfeld der Bewegungserkennung. Mit Hilfe von Optical Flow werden die Bereiche, in denen Bewegungen stattgefunden haben, erkannt und mittels Bildeffektalgorithmus transformiert [47]. Ein weiterer Unterschied zu bisher betrachteten Methoden ist, dass zusätzlich zum ersten Frame auch regelmäßig Keyframes mit dem Bildeffektalgorithmus verar-

Abbildung 2.6: Diese Abbildung aus [78] zeigt Originalbild (*links*) und Effektbilder (*Mitte* und *rechts*) zu verschiedenen Verarbeitungsschritten. Zu einem früheren Schritt (*Mitte*) wird durch große Pinselstriche ein grobes Ergebnis erzielt. Das endgültige Resultat wird im rechten Bild gezeigt. Durch feine Pinselstriche wurde das grobe Bild des früheren Verarbeitungsschritts detailreicher.

beitet werden. Diese regelmäßige Aktualisierung hat eine Minderung der langsam entstehenden Artefakte zur Folge. Swimming konnte durch die Einführung von Keyframes nicht verhindert werden.

Andere Effekte

Neben der Einflussnahme des Optical Flows auf die Position von Pinselstrichen, kann er auch mit anderen Effektprimitiven oder auf andere Art und Weise zur Beibehaltung der Kohärenz beitragen. Dieser Abschnitt soll einen Rahmen für Videoeffekte mit solch einer alternativen Verwendung bieten.

Aquarell [82] Besonders der Schritt vom Aquarell zum Videoeffekt stellt eine Herausforderung dar. Gerade die Eigenschaften (u.a. Texturen), die ein Bild wie mit Wasserfarben gemalt aussehen lassen, produzieren in Videos sichtbare Artefakte. In [82] wird versucht, diese durch Bereiche und Texturen, die mittels morphologischer Operationen berechnet und entsprechend des Optical Flows bewegt werden, zu verhindern.

Im Bildeffekt wird das Originalbild C gemäß Formel 2.1 abstrahiert. Mit einer zuvor gewählten Textur P (siehe Abbildung 2.7 *c*), welche die Grundlage für das mit Wasserfarben gemalte Erscheinungsbild bietet, wird eine neue, modifizierte Farbe C' errechnet [82].

$$C' = C(1 - (1 - C)(P - 0.5)) \tag{2.1}$$

Für die Stilisierung wird zusätzlich eine zusammengesetzte morphologische Operation (auf jeden einzelnen Farbkanal) angewandt. Sie soll sowohl besonders helle, als auch dunkle Details entfernen. Die Vorgangsweise ist folgende:

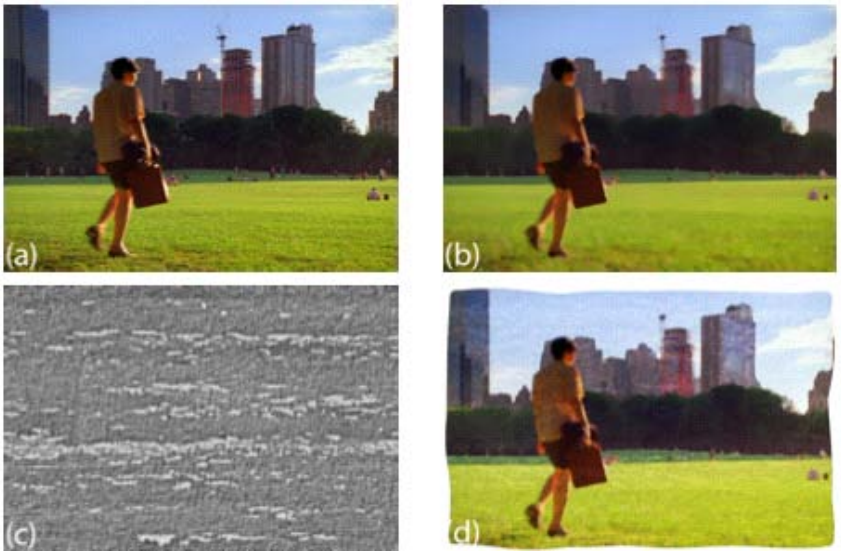

Abbildung 2.7: Ein Frame des Inputvideos *(a)* wurde mit Hilfe der morphologischen Operationen abstrahiert *(b)*. Die Textur *(c)* wird nach Formel 2.1 miteinbezogen, sodass das Ergebnisbild *(d)* entsteht. [82]

- Zuerst wird *closing* [85], also eine *Dilatation* gefolgt von einer *Erosion*, durchgeführt.
- Anschließend wird *opening* [85], also eine *Erosion* gefolgt von einer *Dilatation*, durchgeführt.

Im Videoeffekt werden beide Hauptmerkmale des Bildeffekts, *texture mapping* und morphologische Operationen, unter Berücksichtigung des Optical Flows von Frame zu Frame weitergetragen. Die Textur wird an die sich von Frame zu Frame bewegenden Pixel angehängt, indem ihre Koordinaten entsprechend des Bewegungsvektors versetzt werden. Dadurch entsteht eine, sich mit der Zeit steigernde, Verformung der Textur, welche durch periodisches Zurücksetzen auf das Original und eine Mischung mehrerer aufeinander abgestimmter Textur-Layer verstärkt wird. Im Gegensatz zu den bisher behandelten Videoeffekten arbeitet dieser nicht nur in eine Richtung, sondern lässt einen Layer vom ersten zum letzten Frame und den anderen vom letzten zum ersten Frame durchlaufen. Die angewandte Textur P ist also immer eine Mischung aus zwei Layern. Zusätzlich wird P mit einem der aktuellen Verformung (ein Wert berechnet durch verschiedene Matrixoperationen, siehe [82]) abgeleiteten Wert gewichtet. Ein zweites Layer-Paar einer visuell sehr ähnlichen Textur und einer halb so langen Erneuerungsperiode verhindert sichtbare zyklische Veränderungen des Kontrasts und verbessert damit die zeitliche Kohärenz.

Auch beim morphologischen Filter gibt es eine temporäre Adaption für den Videoeffekt, welche die bei der Frame-nach-Frame Anwendung auftauchenden Artefakte verhindert. Das neue 3D-Strukturelement für Closing und Opening ist eine Menge von langsam auftauchenden

und verschwindenden 2D-Nachbarschaften, die in mehreren aufeinander folgenden Frames definiert ist. Diese Pixelnachbarschaft wird anschließend entsprechend des Optical Flows für das nächste Frame verändert.

Zusätzlich zur globaleren Anwendung des initialen Optical Flow Konzepts, welche versucht Ungenauigkeiten des Bewegungsfelds auszugleichen, wird auch hier auf die Transparenz (der 2D-Nachbarschaften) gesetzt, welche sich schon beim Layer basierten Painterly Rendering [28] bewährt hat. Diese Kombination verspricht die Produktion automatisch generierter, zeitlich kohärenter und visuell ansprechender Animationen.

Video Cubes [16][43] Die bisher vorgestellten Videoeffekte haben mit Ausnahme des gerade beschriebenen Effekts, welcher in beide Richtungen vorgeht, die Videosequenz vom ersten bis zum letzten Frame sequenziell verarbeitet und versucht, durch verschiedene Methoden, wie beispielsweise [40], Artefakte zu verhindern. Die im Folgenden beschriebene Transformation zum Videoeffekt hat ein konträres Verhalten. In [43] wird ein Video als Raum-Zeit-Volumen, oder *video cube*, betrachtet und der Effekt auf mehrere Frames gleichzeitig angewandt. Dadurch können lokale Entscheidungen basierend auf globalem Wissen über die Videodaten getroffen werden [43].

Vor dem eigentlichen Rendern werden Videos einer Raum-Zeit-Analyse unterzogen und mit Hilfe von u.a. Optical Flow sogenannte *rendering solids* konstruiert, die im Raum-Zeit-Volumen existieren. Diese Funktionen bilden reelle Zahlen (Zeitpunkt t im Intervall $[t_1, t_2]$) auf m-dimensionale Merkmalsvektoren R ab, sodass sie zu einem bestimmten Zeitpunkt die Parameter für das Rendern eines NPR Primitives zur Verfügung stellen: [16]

$$S : R_{[t1,t2]} \rightarrow R^m \tag{2.2}$$

Es gibt verschiedene Rendering Solids S, wie beispielsweise *worm-like* Rendering Solids [16]. Eines dieser Solids ist durch die Bewegungskurve eines Punktes durch Raum und Zeit ($[t_1, t_2]$) definiert. Alle Positionen zusammen ergeben wurmartige Kurven, die sich durch das Video ziehen. Entlang des Bewegungsablaufs werden Parameter wie Farbe und Orientierung (Gradient) gespeichert. Um mehrere Rendering Solids zu erzeugen, wird der von [40] adaptierte *2D streamline placement*-Algorithmus verwendet. Das Videovolumen wird dabei als gleichmäßiges Gitter strukturiert. Folgt man, beginnend beim ersten Frame, dem Optical Flow durch dieses Gitter, werden Kurven (*streamlines*) erzeugt. Nähert sich eine Kurve einer neuen Zelle, werden die Nachbarzellen auf andere Kurven überprüft, um zu verhindern, dass sie sich zu sehr annähern. In diesem Fall oder beim Vorfinden vom Rand des Videovolumens ist ihr Ende erreicht. Entlang ihres Pfads wird nach dem Startpunkt für eine neue Kurve gesucht.

Als Ergebnis erhält man Punkte von Kurven, durch die schlussendlich B-Splines gelegt werden, um den Umfang der Daten zu verringern. Beim anschließenden Rendern werden

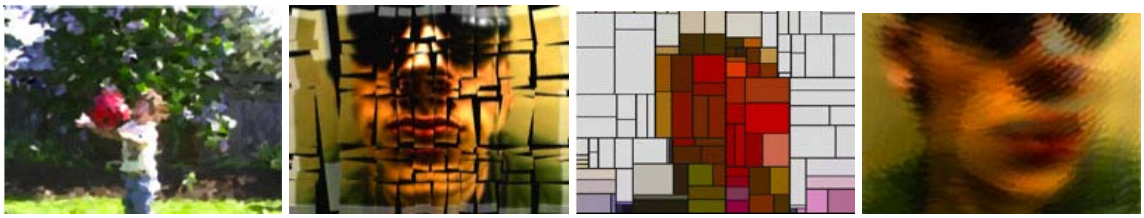

Abbildung 2.8: Diese Abbildungen aus [16] zeigen verschiedene mit Video Cubes gerenderte Stile: impressionistisch, als Fotomosaik, kubistisch und abstrakt. Sie wurden mit unterschiedlichen Rendering Solids erzeugt.

zum Zeitpunkt jedes Frames Parameter wie Position, Farbe und Skalierung für die Bewegungsabläufe der Punkte ermittelt und dementsprechend eine gewählte Maske (z.B. ein runder Pinselstrich) ins Frame gesetzt. Abbildung 2.8 zeigt Ergebnisse, die mit zuvor beschriebenen (*erstes Bild*) und anderen, hier nicht näher erklärten, Rendering Solids erzielt wurden [16].

Ein weiteres Beispiel sind die *shard-like* Rendering Solids, welche hier nur oberflächlich behandelt werden. Sie entsprechen Geraden, die ein Bild in mehrere inhaltsunabhängige Polygone aufteilen (siehe Abbildung 2.8, *zweites Bild*) und jeweils von einem Rand des Frames zum nächsten reichen. Im ersten Schritt, der Vorverarbeitung, werden Grenzen (Liniensegmente) der Polygone gefunden und zwischen Keyframes interpoliert. Aufgrund des Wissens über die zukünftige Position dieser Grenzen kann die Berechnung des Bewegungsvektors vereinfacht werden. Beim Rendern haben BenutzerInnen unter anderem die Möglichkeit, Inhalte der Polygone zu tauschen, zu vergrößern oder zu bewegen.

Die präsentierten Systeme, welche Optical Flow als zeitliche Komponente einsetzen, berechnen durchwegs Bewegungspfade für Effektprimitive. Sie alle verwenden zusätzlich unterschiedliche Strategien (z.B. Transparenz [28] oder globalere Verarbeitung [43]), die Ungenauigkeiten der Bewegungsberechnung für BetrachterInnen nicht sichtbar machen sollen. Wobei sich die Strategien selbst auch negativ auf die zeitliche Kohärenz auswirken können [52]. Obwohl die entwickelten Systeme durchaus auch kohärente Ergebnisse produzieren, werden aufgrund dieser Beobachtung in der Diplomarbeit Ansätze angestrebt, die ohne die Berechnung eines Bewegungsfelds auskommen oder dieses auf eine vielleicht weniger für die Kohärenz kritische Weise einsetzen (siehe Kapitel 3 und Kapitel 4.1.1). Außerdem soll in dieser Diplomarbeit, zumindest pro Frame, räumlich auf einem höheren Level gearbeitet werden. Anstatt das Video aus, weitgehend vom Bild unabhängigen, Effektprimitiven aufzubauen, wird dieses analysiert und Farbhäufungen gefunden (siehe Kapitel 3.1). In [43] und [82] wurde darauf hingewiesen, dass alleiniges Einbeziehen des Vorgängerframes oft nicht ausreicht, um gute Ergebnisse zu erzielen. Ein im Zuge dieser Diplomarbeit implementierter Effekt soll daher die Möglichkeit bieten mehrere Frames in seine Berechnungen einzubeziehen (siehe Kapitel 3.2.2). Folgender

Abschnitt bespricht eine, ebenfalls für die entwickelten Videoeffekte relevante, Alternative zur Verwendung von Optical Flow Daten.

2.2.3 Frame Differencing

Ein mit den Optical Flow Techniken verwandter Ansatz, das *Frame Differencing*, wurde in der Literatur ebenfalls bereits für den Erhalt der zeitlichen Kohärenz eingesetzt. Dieses Verfahren ermittelt durch eine einfache Differenzbildung der Farbwerte zwischen aufeinander folgenden Frames ob und in welchen Regionen Farbveränderungen stattfinden. Dieser Ansatz läuft in der Regel schneller als der Optical Flow, da seine Berechnung deutlich weniger aufwendig ist. Bei diesem Algorithmus führen Lichtunterschiede, Kamerabewegungen sowie Rauschen zu fehlerhaften und somit unruhigen Ergebnissen. [7]

Ein Beispiel für die Anwendung dieses Ansatzes zeigt eine alternative Version des Interaktiven Painterly Renderings von Hertzmann und Perlin [34], welches aus Abschnitt 2.2.2 als Optical Flow Technik bekannt ist. Im Folgenden soll die zweite, noch nicht behandelte Variante der Transformation des im Bildeffekt aus Pinselstrichen aufgebauten Effektbilds für einen Videoeffekt beschrieben werden.

Interaktives Painterly Rendering mit Frame Differencing [34] In der zweiten Variante des Interaktiven Painterly Rendering Algorithmus von [34] wird das erste Frame der Videosequenz mit dem Bildeffektalgorithmus (siehe Abschnitt 2.2.2) verarbeitet. Jedes weitere Frame wird dadurch konstruiert, dass sukzessive Bereiche der Vorgängerframes, die sich stark verändert haben, erneuert werden. Der Ansatz ist bemüht, das in [52] entstehende Flimmern im Ergebnis zu verringern, indem gleichbleibende Bereiche unverändert bleiben. Bei alleinigem Übereinanderzeichnen kann z.B. durch Rauschen ein eigentlich statischer Bereich immer wieder erneuert und damit das Ziel eines kohärenten Ergebnisses nicht erreicht werden. Aus diesem Grund wird die Differenzbildung zwischen benachbarten Frames, F_t und F_{t+1}, als Bedingung verwendet. Um die Entscheidung, ob viel Bewegung stattgefunden hat bzw. ob eine Aktualisierung erfolgt, nicht pro Pixel treffen zu müssen, werden dabei mehrere Pixel zu einem Bereich $(i, j) \in M$ zusammengefasst. Das hat weiters den Vorteil, dass Ausreißer vernachlässigt werden. Die Differenzen der Pixelfarben eines Bereichs des aktuellen Frames von denen des Vorgängerframes werden summiert und entsprechend der Anzahl der Pixel in einem Bereich M gemittelt [34]:

$$\frac{1}{M} \sum_{(i,j)\in M} ||F_{t+1}(i, j) - F_t(i, j)|| > T \tag{2.3}$$

Übersteigt die Summe von Differenzen in einer Bildregion einen Schwellenwert T, so wird sie als verändert erkannt. Um mit dieser Technik auch Überblendungen zu identifizieren, wird die

Abbildung 2.9: *Oben*: Drei Frames aus dem Film *A Scanner Darkly* [51]. *Unten*: Drei Frames aus dem Film *Waking Life* [50]. Während erstere Bildabfolge reale Objekte enthält, hat der Künstler bei der zweiten Bildabfolge den gefilmten Bildinhalt deutlich modifiziert.

Summe für jede Pinselstrichgröße und jeden Bereich über mehrere Frames hinweg aktualisiert.

Auch diese Transformation von Bild- zu Videoeffekt mittels Differenzbildung bietet keine optimale Lösung. In [34] wird deshalb empfohlen die Frame-Rate zu verringern, damit das Flimmern für BetrachterInnen weniger störend wirkt. Nichtsdestotrotz gibt es Systeme, die sich von diesem Lösungsansatz inspirieren ließen. Kovács und Szirányi [47] aktualisieren ebenfalls Bereiche von Frames, in denen Bewegung stattgefunden hat. Der Unterschied liegt lediglich in der Erkennung dieser Bereiche, so greifen sie nicht auf ein einfaches Frame Differencing zurück, sondern interpretieren die Daten aus dem Optical Flow Feld (siehe Abschnitt 2.2.2). Auch ein im Zuge dieser Diplomarbeit entstandener Videoeffekt definiert eine Entscheidungsfunktion basierend auf Differenzen adjazenter Frames (siehe Kapitel 3.2.3).

2.2.4 Vektorisierung

Eine deutlich komplexere Herangehensweise an die Erzeugung eines zeitlich kohärenten Videoeffekts ist die Analyse des Grundmaterials zum Zweck der Extraktion und Verfolgung einzelner Segmente. Anstelle fix vorgegebener Primitive treten Videoobjekte mit semantischer Bedeutung (z.B. [1]), deren zeitliche Veränderungen für die korrekte Darstellung des Ergebnisvideos wichtig sind. Diese Videoobjekte werden in der Regel durch Segmentierung extrahiert und in einer System-internen Repräsentation aus grafischen Primitiven (z.B. Kurven [95]) gespeichert. Dieser Vorgang wird als *Vektorisierung* bezeichnet.

Ein Beispiel für einen solchen Ansatz ist die *Rotoskopie* aus dem Jahre 1917. Dieser Ansatz hat in seinen frühen Jahren AnimatorInnen sehr viel Handarbeit abverlangt. Damals musste über das gefilmte Grundmaterial, Frame für Frame, ein neues Bild gezeichnet werden, um aus dem Endergebnis eine Animation zu erzeugen [20]. Die aktuellere, automatische Roto-

skopie versucht die BenutzerInnen-Intervention zu minimieren, sodass man auch ohne Wissen über Modellierung oder teures Equipment eine Animation erstellen kann [45]. KünstlerInnen zeichnen in regelmäßigen Abständen in Keyframes ein Objekt, dessen Konturen anschließend in Splines oder ähnliche Datenstrukturen übersetzt und in den Intervallen zwischen den Keyframes linear interpoliert werden [71]. Beispiele für Filme, die mit dieser Technik nachbearbeitet wurden, sind *Waking Life* [50] und *A Scanner Darkly* [51] unter der Regie von Richard Linklater. Die aus diesen Filmen ausgewählten Bildfolgen in Abbildung 2.9 machen deutlich, dass sich die Rotoskopie nicht auf das reine Nachzeichnen von Konturen beschränkt, sondern auch künstlerische Modifikationen der Bildinhalte zulässt. Außerdem wird Rotoskopie dazu verwendet in der Postproduktion Schauspieler in neue Szenen zu platzieren oder einzelne Objekte durch CGI (*computer-generated imagery* [71]) zu ersetzen.

Weitere Techniken, die sich durchaus an der automatischen Rotoskopie orientieren, aber weniger Freiraum zur künstlerischen Gestaltung der Objekte bieten, werden unter dem Begriff *Video Tooning* zusammengefasst. Sie zielen darauf ab, ein Video ohne oder mit geringem Einfluss von Applikations-BenutzerInnen in einen Cartoon zu transformieren. Auch wenn die konkreten Effekte unterschiedliche Ergebnisse liefern, so verarbeiten sie die Videodaten üblicherweise in folgenden drei Schritten [37].

- Segmentierung
- Tracking
- Rendering

In weiterer Folge werden diese Schritte näher behandelt und verschiedene Beispiele aus dem Bereich Video Tooning angeführt. Die Schritte zeigen auf, wie ein Videoobjekt mit semantischer Bedeutung aus dem Grundmaterial extrahiert, verfolgt und seine System-interne Repräsentation unterschiedlich dargestellt werden kann.

Segmentierung

Der erste Schritt einer Vektorisierung ist für gewöhnlich die Segmentierung der Videopixel in Regionen mit semantischer Bedeutung, um später gegebenenfalls *higher level* Operationen wie Erkennung oder semantische Interpretation ausführen zu können [37]. Für dieses klassische Problem der Computer Vision gibt es zahlreiche Ansätze und Lösungen, so könnten Punkte im Bild als Vektoren im Merkmalsraum [10], das zugrunde liegende Bild als Graph [45] oder Segmente von BenutzerInnen selbst [1] definiert werden. Spezifisch für das Erstellen eines cartoonartigen Videoeffekts wurden bisher nur wenige davon verwendet. In der Literatur fanden sich grob zusammengefasst zwei Ansätze. Zum einen werden Videos mit dem *Mean Shift Algorithmus* oder mit Variationen dieses Algorithmus segmentiert, zum anderen - ähnlich der

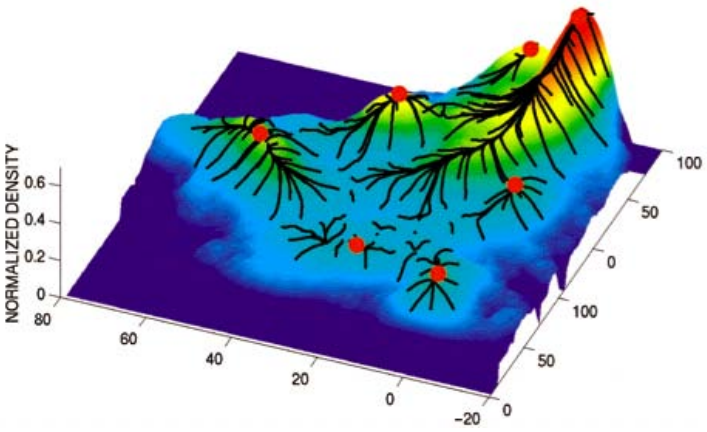

Abbildung 2.10: Dieses Diagramm aus [10] visualisiert Trajektorien der Farbänderungen, die sich durch die Dichteschätzungen ergeben. Die *roten Punkte* markieren die lokalen Maxima als Ergebnis des Algorithmus.

Rotoskopie - durch Definition der Objektkonturen in Bereiche geteilt, welche später weiterzuverarbeiten sind. Während der erste Ansatz also nach Flächen sucht, ergeben sich die Flächen beim Zweiten durch das Erkennen ihrer Grenzen. Im Folgenden sollen diese beiden Ansätze betrachtet und jeweils Anwendungsbeispiele gegeben werden. In Kapitel 3 werden außerdem die Segmentierungsalgorithmen, welche für die im Zuge dieser Diplomarbeit entstandenen Effekte verwendet werden, beschrieben.

Mean Shift Algorithmus und seine Variationen Ein in der Video Tooning Literatur häufig genannter Segmentierungsalgorithmus ist der Mean Shift Algorithmus [10]. Er wurde erstmals 1975 von Fukunaga und Hosteler [22] präsentiert, fand aber vor allem in den letzten Jahren bei Aufgaben der Computer Vision besonders Anklang [77][49][69][11][12].

Der Algorithmus basiert auf der Analyse eines mehrdimensionalen Merkmalsraums und kann im Zuge dessen, im Vergleich zu vielen anderen Segmentierungsalgorithmen (z.B. [96]), die Anzahl der benötigten Segmente ermitteln. Betrachtet man die Farbe eines Pixels als Punkt im Merkmalsraum, in diesem Fall der dreidimensionale Farbraum ($d = 3$), so sind Pixel mit ähnlicher Farbe darin als Häufungen (*Cluster*) erkennbar. Um diese Häufungen zu finden, müssen die Daten analysiert werden. Nimmt man nun weiter den Merkmalsraum als empirische Dichtefunktion $\widehat{f}$ einer Wahrscheinlichkeitsverteilung an, entsprechen die Häufungen den lokalen Maximalwerten der Dichtefunktion und das Problem wird zur Suche der Momente umformuliert [10][37]:

$$\widehat{f(x)} = \frac{1}{nh^d} \sum_{i-0}^{n} K(\frac{x - x_i}{h}) \tag{2.4}$$

Die Dichte $\widehat{f}(.)$ ist dabei an jedem Punkt $x \in [x_1, ...x_n]$ im Raum abhängig von den Farben

seiner Nachbarpixel $x_i \in [x_1, ...x_n]$ definiert. Ein *Kernel* K legt dabei fest welche bzw. wie viele Pixel den Dichtewert beeinflussen und wie stark der Einfluss der einzelnen Nachbarpixel ist. Ursprünglich ist ein solcher Kernel symmetrisch definiert. Die Stärke des Einflusses der einzelnen Nachbarpixel fällt mit der Entfernung von seinem Zentrum x. h entspricht dem Radius des Kernels und wird *Bandbreite* genannt. Sie beeinflusst den Grad der Abstraktion. Ist der Wert niedrig, so werden Objekte wahrscheinlicher in mehrere kleine Segmente aufgespalten. Bei höheren Werten werden kleine Farbvariationen vernachlässigt und das Objekt eher als eine einfärbige Fläche dargestellt. Der Mean Shift Algorithmus berechnet wiederholt die Dichten der Pixel (siehe Funktion 2.4) und nähert mit deren Hilfe die Pixelfarben so lange an das nächstgelegene Maximum an, bis sich die Farben kaum oder nicht mehr ändern. Zum Cluster eines gefundenen Maximums gehören alle Pixel, die sich während dieses Prozesses in die Richtung des Maximums bewegt haben (siehe Abbildung 2.10). [22][10]

Eine Adaption des Mean Shift Algorithmus für Videos (u.a. mit asymmetrischem Kernel) wird von [96] vorgeschlagen. Die Anwendung erfolgt für bessere raum-zeitliche Kohärenz auf alle Frames eines Videos gleichzeitig, sodass dreidimensionale Pixelvolumen entstehen. Für die Segmentierung von Videos mittels Mean Shift Verfahren sind hohe Speicherkapazität und die Einplanung von Laufzeiten bis zu einer Nacht erforderlich [49].

Um in Videos oder Bildern cartoonartige Effekte zu erzielen, wurde die Mean Shift Technik bereits von [11], [49], [3], [9], [6] und [77] erfolgreich eingesetzt. Zusammengehörige Segmente werden mit einer beliebigen oder der mittleren Farbe der zugehörigen Pixel eingefärbt.

In [77] können bei Keyframes nachträglich zusätzlich einzelne Segmente zu Gruppen zusammengeführt werden, indem BenutzerInnen im Interface Konturen um sie zeichnen. Die Information über Zusammengehörigkeit von Segmenten kann zwischen Keyframes interpoliert werden.

Ein weiteres Beispiel, das *interaktive System für effizientes Video Cartooning* von [49], stilisiert ein Originalvideo nach einigen Vorverarbeitungsschritten mit dem herkömmlichen Mean Shift. Ausgegangen wird dabei von einem Video, das in fixem Winkel aufgenommen wurde und keine überraschenden, schnellen Bewegungen der Kamera oder der Objekte aufweist. Zuerst wird eine Rauschunterdrückung (Bilateraler Filter [84]) ausgeführt, um das Video zu stabilisieren. In einem weiteren Schritt wird das Video in Vorder- und Hintergrund geteilt (*Background Subtraction*, siehe [49]), um das Datenvolumen und die damit verbundene Verarbeitungszeit zu reduzieren. Die beiden Teile werden voneinander unabhängig aufbereitet. Auf eine Bildsequenz, welche nur Vordergrundbilder des Videos enthält, wird (auf alle Einzelbilder gleichzeitig) der Mean Shift Algorithmus angewandt. Da aufgrund der fixen Kameraposition das Hintergrundbild statisch ist, wird es nur einmal mittels Mean Shift segmentiert. Durch Bewegungen und Überlagerung der Koordinaten des Vordergrunds auf dem Hintergrundbild ergibt sich schlussendlich ein gemeinsames Videovolumen. Wie in [77] werden auch hier sowohl au-

Abbildung 2.11: *Links* befinden sich zwei Ergebnisse der Mean Shift Segmentierung aus [77], in der *Mitte* eines aus [49] und *rechts* aus [9]. Bei diesen Bildern wurden keine weiteren Stilisierungen wie das Hinzufügen von Kanten etc. angewandt.

tomatisch als auch durch AnwenderInnen überflüssige Segmente eliminiert.

Die *VideoPaintBox* von Collomosse und Hall [9] verzichtet bewusst auf den für Videos optimierten Ansatz des Mean Shifts. Das System segmentiert mit dem EDISON Algorithmus (*Edge Detection and Image Segmentation* [24]), einer Variante des initialen Mean Shift Algorithmus, zunächst jedes Frame unabhängig von seinen Nachbarn. Dieser synergistische Ansatz kombiniert die Segmentierung (globale Information) mit Kantenerkennung (lokale Information), um die Qualität des Ergebnisses zu verbessern. Für Kantenpixel zwischen zwei Regionen wird aufgrund der Farbnähe und einer vom Gradienten abhängigen Gewichtung berechnet, zu welcher Region sie gehören.

Die Ergebnisse der Einzelbilder der drei Anwendungsbeispiele [77], [49] und [9] im Bereich Video Tooning sind durchaus vergleichbar (siehe Abbildung 2.11). Sie unterscheiden sich in Geschwindigkeit, der erreichten zeitlichen Kohärenz und den Beschränkungen, die sie dem originalen Videomaterial auferlegen. Eine weitere Anwendung des Mean Shifts abseits dieser Beispiele, welche jedoch bereits auch im Bereich des Video Toonings verwendet wurde (z.B. [3]), ist das Extrahieren von Objekten aus einem Video [2].

Da die für einen Cartoon typischen Konturen und die interne Darstellung im Allgemeinen nicht durch Segmentierung alleine erzeugt werden, müssen zusätzliche Algorithmen angewandt werden. Die oben gezeigten Beispiele [9][77][49] verwenden dafür unterschiedliche Ansätze:

- Sub-Volumen des Videos werden durch Grenzflächen (*stroke surfaces* [7]) repräsentiert und in Form von Flächen, Kanten und Punkten in Listen (*winged edge structure* [5]) gespeichert. Diese Listen können zusätzliche Information, z.B. die Farbe der Sub-Volumen, enthalten und diese über die Dauer des Videos glätten. [9]

- Verarbeitung der Segmente der Einzelbilder zu mittels Punkten und Kanten definierten Regionen und Ableitung von Konturen aus Segmentgrenzen. Um zeitliche Kohärenz auch bei den Kanten beizubehalten, werden sie zusätzlich geglättet und ihre Positionen gemittelt. [77]

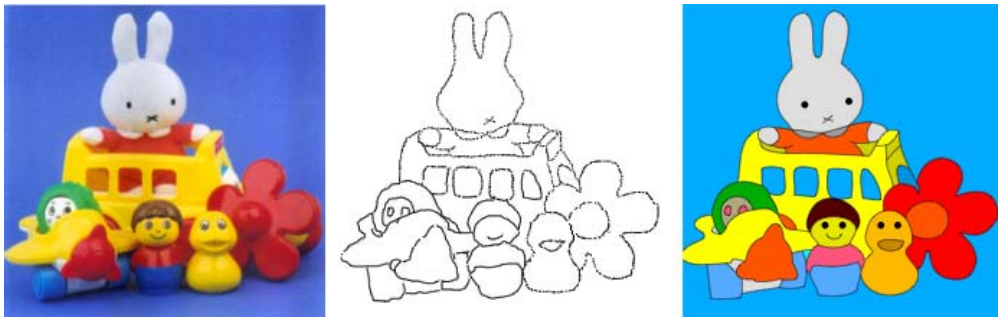

Abbildung 2.12: Diese Abbildung aus [1] zeigt Originalbild (*links*), die von BenutzerInnen definierten Konturen (*zweites Bild*) und das Ergebnis nach der Produktion von geschlossenen Flächen (*rechts*). Mit Hilfe der Skizzen der BenutzerInnen können sichtlich überzeugende Cartoon-Versionen des Originalbilds erzeugt werden.

- Bezier-Kurven-Approximation der Segmentgrenzen [49]

Konturbasierte Ansätze Eine andere Möglichkeit, ein Bild oder Video in Bereiche mit semantischer Bedeutung aufzubrechen, ist die Erkennung bzw. Definition von geschlossenen Konturen. Diese Art der Segmentierung geht einen weiteren Schritt in Richtung automatischer Rotoskopie und verlangt dementsprechend einen gewissen Grad an BenutzerInnen-Intervention.

In *SnakeToonz* [1] skizzieren BenutzerInnen die Konturen direkt auf das erste Frame und initialisieren damit Datenstrukturen (siehe Abbildung 2.12). Diese Skizze wird intern als *Snakes* [95], in diesem Beispiel eine auf Bézier Splines basierende, aktive Kontur, gespeichert. Sie passt sich automatisch an die Kanten des ihr zu Grunde liegenden Bilds an (*relaxation*) und schließt gegebenenfalls beim Skizzieren entstandene Lücken (*snapping*). Sind die Konturen definiert, kann jeder geschlossenen Fläche eine Farbe zugewiesen werden. Um das Färbungs- und PLP- (*point-location*) Problem zu lösen, wird eine doppelt verkettete Kantenliste verwendet. Für ein Ergebnis, wie es in Abbildung 2.12 durch ein einzelnes Frame angedeutet ist, müssen Frames des Originalvideos stark definierte Kanten, ausgeprägte Kontraste und möglichst keine Überdeckungen oder Änderungen der Perspektive aufweisen. In solchen Fällen haben BenutzerInnen einzugreifen, um das Ergebnis zu korrigieren. In der Animation neigen Objekträndern zum Flimmern, da während der Relaxation Ungenauigkeiten entstehen können [7].

Eine ähnliche Technik für das Erzeugen von Animationen aus Videos wird in [35] angewandt. Auch bei computerunterstützter Rotoskopie wie in [71] wird auf diesen Ansatz zurückgegriffen. Dabei werden von BenutzerInnen gezeichnete Kurven an Kanten im Bild angepasst. Weiters existiert eine Verallgemeinerung von Snakes, aktive Flächen [25], welche für die Segmentierung von Filmsequenzen eingesetzt werden kann.

Ein Vorteil der Repräsentation der Videosequenz durch Konturen, sei es als Ergebnis ei-

ner kantenbasierten Segmentierung oder der Ableitung der Segmentgrenzen, ist die nachträgliche Invarianz gegenüber der Größe des Videos. Das Ergebnis kann ohne Qualitätsverlust in verschiedenen Größen abgespielt werden, was sich auch auf die Datenmenge bei der Speicherung positiv auswirkt. Das Outputvideo kann ohne Probleme unter Verwendung von vektorbasierten Formaten wie Flash im Internet gestreamt werden [1]. Um diese Repräsentation der Daten zu erhalten, sind jedoch im Vorfeld komplexe Berechnungen notwendig.

Tracking

Beim Tracking werden die anhand der Segmentierung ermittelten Objekte von Frame zu Frame verfolgt. Die Hauptherausforderung dabei ist die zeitliche Kohärenz zu wahren, sodass es zu keinem oder möglichst wenig Flackern von Kanten, Farben oder Flächen kommt. Ungenaue Beobachtungen von Zielobjekten durch beispielsweise Rauschen oder Überdeckungen können zu Fehlern während des Verfolgungsprozesses führen und die Information über den Verlauf der Bewegung eines Objekts verfälschen [37]. Es existieren verschiedene Punkt-, Kernel- und Kontur-Tracker, welche je nach Repräsentation des zu verfolgenden Objekts unterschiedliche statistische, deterministische oder heuristische Ansätze wählen, um dieses Problem zu lösen [39]. Wie beim Optical Flow ist auch das Ziel der Tracking-Verfahren, der Lösung des *correspondence problems* möglichst nahe zu kommen.

Collomosse und Hall [7][8] präsentieren eine Möglichkeit, (durch Mean Shift) ermittelte Regionen in einer internen Repräsentation zu speichern und aufgrund ihrer Farbe, Position und Form miteinander zu assoziieren. Ihre Heuristik sortiert pro Region $R_{t,1}$ eines Frames t alle Regionen $R_{(t+1),1}, R_{(t+1),2}, ...$ des Nachfolgeframes $t + 1$, welche sich in seinem Einflussgebiet (definiert durch einen Radius um das Zentrum von $R_{t,1}$) befinden, nach ihrer Ähnlichkeit. Beginnend mit der ähnlichsten Region werden so lange Assoziationen zu $R_{t,1}$ erstellt und sukzessive die Pixelanzahl der betrachteten Region von jener bei $R_{t,1}$ subtrahiert, bis die Differenz unter Null fällt oder der Ähnlichkeitswert einem Schwellenwert unterschreitet. Dieser Prozess wird für alle Frames wiederholt, bis schlussendlich die Videosequenz als eine Menge von Sub-Volumen (*Videoobjekte*) repräsentiert werden kann. In einem zweiten Schritt werden die entstandenen Verbindungen gefiltert und die durch Rauschen oder Übersegmentierung falsch erkannten Assoziationen entfernt. Kurzlebige Objekte (weniger als eine Sekunde) werden beispielsweise entfernt, indem ihre Region jeweils mit einer der benachbarten verschmilzt. Finales Glätten der Videoobjekte und ihrer Grenzen erhöht zusätzlich die Stabilität des Ergebnisses. In der Implementierung [7] wird dazu eine Baumstruktur verwendet. Dieser Algorithmus ist sehr robust und verspricht auch mit Kamerabewegungen, Lichtänderungen oder Verdeckungen umgehen zu können.

Ein weiterer Ansatz [77] unterstützt die (durch Mean Shift Clustering der gesamten Sequenz) entstandenen raum-zeitlichen Regionen mit einem zusätzlichen rotoskopischen Inter-

face, welches BenutzerInnen im Vorfeld erlaubt, die Konturen eines Objekts zu markieren. Die Information aus der Segmentierung kann verwendet werden, um die Konturen zwischen Keyframes zu interpolieren und damit saubere Objektgrenzen zu produzieren.

SnakeToonz [1] definiert, wie schon erwähnt, aktive Konturen (Snakes), welche sich an das zugrundeliegende Bild anpassen. Beim Tracking werden zunächst die Konturen des Vorgängerframes auf das aktuelle Frame übertragen und anschließend der *Shi-Tomasi Feature Tracker* [76] angewandt, um die Endpunkte im neuen Frame zu identifizieren. Da diese oft an Ecken liegen, können sie leichter wiedergefunden werden als andere Punkte entlang einer Linie. Der Shi-Tomasi Tracker beschreibt die Verschiebung δ eines Punkts x mit einem Modell einer affinen Transformation

$$\delta = Dx + d \tag{2.5}$$

mit der Translation d und der Verformungsmatrix D. In einem Fenster von 13x13 Pixel um den Punkt wird versucht, sechs Parameter für D und d zu finden, die den Fehler zwischen dem Fenster um den Punkt im vorherigen und aktuellen Frame minimieren. Die Verschiebungsdaten werden zusätzlich gespeichert, um Schätzungen für weitere Bewegungen der Punkte in kommenden Frames zu treffen und den Tracker zu stabilisieren. Die übrigen Punkte werden aufgrund einer, durch die Endpunkte abgeleiteten, Transformation verschoben. An der neuen Position der Snakes wird eine Relaxation mit den Kanten des aktuellen Frames durchgeführt. Beim Auftauchen neuer Objekte in der Szene, Verdeckungen oder im Fehlerfall können oder müssen BenutzerInnen interaktiv in diesen Prozess eingreifen und nachträglich Konturen hinzufügen oder entfernen. Willkürliche und besonders komplexe Szenen können von einem Videoeffekt mit diesem Tracker nicht verarbeitet werden, da die Verformungsparameter in diesem Fall häufig nicht richtig berechnet werden.

Die im vorherigen Abschnitt kurz erwähnten, aktiven Flächen [25] verwenden Kanteninformation und Interpolation zwischen Keyframes, um die Position im neuen Frame zu finden. In [56] hängt die aktuelle Lage einer Kontur vom Ergebnis des Trackings, ausgehend vom vorhergehenden Keyframe und vom nachfolgenden Keyframe bis zum aktuellen Frame, ab. Die Kombination von Ergebnissen beginnend an verschiedenen Punkten des Videos verspricht nicht nur in dieser Variante, sondern auch allgemein eine Verbesserung des Trackings.

Yilmaz et al. [39] bieten eine ausführliche Übersicht über aktuelle Objekt-Tracking Systeme. Ihr Paper kann für weiterführende Informationen zu diesem Thema herangezogen werden.

Im Gegensatz zu den bereits präsentierten Beispielen, gibt es auch Video Tooning Systeme, welche gänzlich auf Tracking verzichten. Das System von Hong et al. [49] trifft zum einen Annahmen bezüglich des Inputvideos (fixe Kameraposition und fixer Winkel) und vertraut zum anderen auf die Qualität der gemeinsamen Segmentierung aller relevanten Videodaten. Da eine

raum-zeitliche Verbindung der Frames schon durch die gemeinsame Verarbeitung gegeben ist, wird auf das Tracking der Segmente verzichtet. Auch in dieser Diplomarbeit wird weniger die räumliche Struktur, Form oder Position eines Segment, sondern vielmehr seine Farbe gemittelt (siehe Kapitel 3.2).

Rendering

Schlussendlich werden die durchgehend im Video identifizierten Effektprimitive auf unterschiedliche Weise präsentiert. Segmente werden oft mit ihren durchschnittlichen Originalfarben [77] und Objektgrenzen durch geschwungene, schwarze Linien repräsentiert, deren Dicke beispielsweise vom Farbunterschied der Segmente, die sie trennen, abhängt [49]. Einige Systeme, wie SnakeToonz [1], gehen weiter und erlauben BenutzerInnen selbst, geschlossenen Flächen Farben zuzuteilen oder einzelne ausgewählte Objekte vor einen neuen Hintergrund zu setzen [1][77][9]. Auch die VideoPaintBox [9] nutzt die Möglichkeit der raum-zeitlichen Videobeschreibung, um das Ergebnisvideo in verschiedenen Stilen darzustellen. So können beispielsweise mit etwas Mehraufwand zeitlich kohärente

- Linienzeichnungen,
- gezeichnete farbige Regionen (ohne Linien),
- gemischte Medien (Mischung aus gerenderten Segmenten und originaler Bildinformation) oder
- Painterly Renderings

erzeugt werden. Die VideoPaintBox ermöglicht darüber hinaus, dass Bewegungslinien [70] hinzugefügt werden können.

Das Video Tooning System von [77] bietet zwar keine Stile für die Darstellung des gesamten Ergebnisses, jedoch für die Darstellung der Kurven. So können BenutzerInnen Eigenschaften wie Linienfarbe und Liniendicke selbst beeinflussen.

2.2.5 Andere Methoden

Dieser Abschnitt beschäftigt sich mit Methoden zur Erhaltung der zeitlichen Kohärenz, die entweder nicht in eine der anderen drei Kategorien fallen, da sie sehr speziell und nur auf ihren Videoeffekt zugeschnitten sind (Video Mosaik [17]), oder den Bildeffekt so definieren, dass sich die Kohärenz auch ohne Zwischenschritt von selbst ergeben soll (Cartoon Abstraktion [93]).

Abbildung 2.13: Diese Abbildung aus [17] zeigt zwei Frames des Video Mosaiks. Ein Frame des Ergebnisvideos ist aus Frames der Quellvideos aufgebaut.

Video Mosaik [17] Der in [17] beschriebene Effekt fügt dem klassischen Mosaik [15] eine dritte Dimension, die Zeit, hinzu. Analog zum Bildmosaik gibt es ein Original- und mehrere Quellvideos, aus denen die Mosaikteile generiert werden (siehe Abbildung 2.13). Im Gegensatz zum Bildeffekt wird beim Videoeffekt nicht nach dem am besten geeigneten Bild für einen bestimmten Bereich, sondern nach dem am besten passenden Frame bzw. der am besten passenden Teilsequenz eines Videos gesucht. Die Herausforderung bei der Transformation vom Bild- zum Videoeffekt ist nicht nur ein Frame eines Videos zu finden, das zum aktuellen Zeitpunkt am besten passt. Denn um den zeitlichen Zusammenhang nicht zu verlieren, sollten die Folgeframes des Quellvideos auch über mehrere Frames des Originalvideos hinweg verwendet werden können. Um das zu gewährleisten, wird in [17] eine Entscheidungsfunktion formuliert und so gewichtet, dass Videos mit passenden Folgeframes bevorzugt werden. In [17] werden einige Kriterien für das Finden einer zu einem Mosaikteil passenden Teilsequenzen der Quellvideos formuliert. Dazu gehören auch Folgende:

- Mittelwert der Farben
- Verteilung der Farben als struktureller Aspekt
- Bewegungs-Aspekte

Klein et al. führen in [17] für die letzten beiden Kriterien eine Wavelet-Transformation in den Farbkanälen jedes Quellvideos aus. Index und Vorzeichen der 30 größten Wavelet Koeffizienten werden anschließend für die Suche gespeichert. Für eine effiziente Durchführung der für die Suche notwendigen Vergleiche wird eine Modifikation des *fast image querying system* von [18] verwendet. Wie stark der Einfluss der einzelnen Kriterien ist, wird in der Entscheidungsfunktion durch, von BenutzerInnen variierbare, Gewichte bestimmt. Sind schlussendlich alle Teile für ein Frame ermittelt, wird zwecks weiterer Verbesserung der Kohärenz eine Farbanpassung zum lokalen Mittel der Teile durchgeführt.

Abbildung 2.14: Diese Abbildung aus [93] zeigt ein Bild im Verlauf der Verarbeitung. Das abstrahierte Bild (*abstracted*) entsteht durch mehrmalige Faltung mit einem Bilateralen Filter und anschließendem Hinzufügen von *DoG-Kanten* des Originalbilds. Das quantifizierte Bild (*quantized*), ganz rechts, wird auf 12 Helligkeitswerte reduziert.

Cartoon Abstraktion [94][93] Die von Winnemöller [93] präsentierte cartoonartige Abstraktion eines Videos geht davon aus, dass markante Eigenschaften in einem Videoeffekt vor allem durch Helligkeit und Farbkontraste erzeugt werden können. Durch das Stärken von Kontrasten bzw. Kanten und Reduzierung von Rauschen bzw. Weichzeichnen von vermeintlich unwichtigen Details sowie Quantifizierung der Helligkeitswerte will das Abstraktions-Framework von [94][93] dies erreichen. Der erste Schritt der Abstraktion in [93] ist die wiederholte Anwendung eines Filters (*Bilateraler Filter* [66], siehe Kapitel 3.3.1), welcher das Bild glättet, aber Kanten stehen lässt bzw. allgemein den Kontrast erhöhen oder verringern kann. Das Beispiel in Abbildung 2.14 macht sichtbar, dass dadurch bereits am Anfang des Algorithmus das Originalbild cartoonartiger wirkt.

Im zweiten Schritt werden parallel eine Quantifizierung der Helligkeitswerte und eine Kantenerkennung durchgeführt. Bei der Quantifizierung (siehe Kapitel 3.3.3) werden die Helligkeitswerte auf mehrere *Bins* (Intervalle) der selben Größe aufgeteilt und somit mehrere Werte durch nur einen Wert repräsentiert. Ein Parameter kontrolliert den Verlauf von einem Bin zum nächsten, sodass entweder mehr langsame Verläufe oder abrupte Farbänderungen entstehen. Dieser Parameter kann entweder fix vorgegeben, also unabhängig vom bearbeiteten Bild sein, oder als Funktion des Gradienten der Helligkeitswerte gewählt werden. Zusätzlich besteht die Möglichkeit z.B. *eye-tracking* [11] generierte Daten einzubinden, um Kontraste gezielt zu beeinflussen. Aus der Sicht des Videoeffekts dient diese Quantifizierung, zumindest die Farben betreffend, dazu die zeitliche Kohärenz des Ergebnisvideos trotz des Effekts aufrecht zu erhalten. Mittels weicher Quantifizierung werden kleine Helligkeitsänderungen, die sonst durch

alleiniges *Binning* sichtbar gemacht würden, verhindert.

Die Kantenerkennung erfolgt mit einem, für den Erhalt der zeitlichen Kohärenz modifizierten, DoG Operator (*difference-of-Gaussians*, siehe Kapitel 3.3.2) [93]. Die letztendlich darzustellenden Kanten werden in [93] mit einem fixen Schwellenwert extrahiert und entsprechend ihrer Stärke geglättet. Schlussendlich werden im RGB Farbraum kleine Ungenauigkeiten bei der Kombination von farbreduziertem Bild und Kantenbild verhindert und das Ergebnis geschärft (siehe Kapitel 3.3.4). Diese Abfolge an simplen und vergleichsweise schnell ausführbaren Bildverarbeitungsoperationen funktioniert nicht nur als Bildeffekt, sondern soll laut [93] auch bei einer Frame-zu-Frame Anwendung ohne weitere Zwischenschritte zu flüssigen Ergebnissen führen.

Der in dieser Diplomarbeit entwickelte Bildeffekt verwendet, inspiriert durch den Effekt von Winnemöller [93], eine ähnliche Methode zur Stärkung von Kontrasten (siehe Kapitel 3.1.4) und glättet Kanten ebenfalls in Abhängigkeit ihrer Stärke (siehe Kapitel 3.1.3). In Kapitel 3.3 wird die Cartoon Abstraktion ein weiteres Mal aufgegriffen und in einer leicht modifizierten Form als eine auf der CPU laufende Variante implementiert. Die visuellen Ergebnisse dieses Effekts werden in Kapitel 4 jenen anderer im Zuge dieser Diplomarbeit entstandenen Videoeffekte gegenübergestellt.

Kapitel 3

Algorithmen für Bild- und Videoeffekte

Gestützt auf die Recherchen in Kapitel 2 werden ein Bildeffekt, und darauf aufbauend, unterschiedliche Videoeffekte implementiert. Die VEL (Video Effect Library), eine Bibliothek für Bild- und Videoalgorithmen, wird um, in der Bibliothek noch nicht implementierte, Ansätze aus der Literatur für den Erhalt von zeitlicher Kohärenz in Videos erweitert. Aufgrund der Abhängigkeit der meisten Methoden von Schätzungsverfahren für die Berechnung der stattgefundenen Bewegung und der Probleme, die hinsichtlich der zeitlichen Kohärenz entstehen, wird versucht, auf diese Berechnungen zu verzichten und mit anderen Techniken (z.B. dem Mitteln der Farben von Segmenten) über eine Frame-zu-Frame Anwendung des Bildeffekts hinauszugehen. Ähnlich dem einfachen und gleichzeitig mächtigen Konzept der Platzierung von Effektprimitiven entsprechend eines Vektorfelds wird auch im Zuge dieser Diplomarbeit versucht, einfache bzw. studentischen Vorkenntnissen entsprechende Ansätze zu wählen, sodass diese gegebenenfalls in der Lehrveranstaltung „LU Videoverarbeitung“ eingesetzt werden können. Mit der Intention, gezielt leicht verständliche Algorithmen zu entwickeln, heben sich die im Folgenden präsentierten cartoonartigen Effekte von vielen anderen Techniken (z.B. [77]) im Bereich des Video Toonings ab.

Grob zusammengefasst werden in dieser Diplomarbeit zwei charakteristische Merkmale eines Trickfilms, die Erzeugung von einfärbigen Flächen durch Farbreduktion und die Betonung von Konturen durch Kantenerkennung, simuliert. In Anlehnung an global arbeitende Videoeffekte (z.B. [46]) versucht die erste Transformation von Bild- zu Videoeffekt, Inkonsistenzen, insbesondere Farbsprünge, durch das Miteinbeziehen mehrerer Frames zu vermeiden (siehe Abschnitt 3.2.2). Ein weiterer Videoeffekt stellt eine Beziehung zwischen den Farben benachbarter Frames her, um damit ein stabileres Ergebnis zu liefern (siehe Abschnitt 3.2.1). Eine dritte Transformation von Bild- zu Videoeffekt basiert auf einer Entscheidungsfunktion, welche Unterschiede in adjazenten Frames einbezieht und Flimmern reduziert (siehe Abschnitt 3.2.3).

Dieses Kapitel klärt Details der Funktionsweise und Parameter dieser Effekte. Zusätzlich wird der Algorithmus eines von [93] adaptierten Videoeffekts behandelt, da dieser als weitere

Ergänzung der VEL implementiert wurde.

3.1 ColorReduction Bildeffekt

Der *ColorReduction Bildeffekt* strebt eine Abstraktion des Originalbilds mit einem Ergebnis möglichst nahe an einem *Comic* oder dem Einzelbild eines Cartoons an. Ausgehend von deren Basiseigenschaften, wie der Einfachheit durch homogene, einfärbige Flächen, wird versucht, einen eigenen Stil zu entwickeln und gleichzeitig BenutzerInnen Variationsmöglichkeiten zu bieten, um diesen leicht abzuwandeln. Grundlegende stilistische Unterschiede von Comics wie etwa ihr Detailgrad, Liniendichte und -stärke sowie Farbe und Schärfe sind aus diesem Grund variabel [59].

Die künstlerische Vorgehensweise einer Großzahl an Cartoonistinnen und Cartoonisten [59] sowie die traditionellen Arbeitsschritte bei der Erstellung eines Zeichentrickfilms [14] beginnen mit einer groben Linienzeichnung und Umrissen semantisch bedeutender Objekte, welche später um Details und Farbe erweitert werden. Der *ColorReduction Bildeffekt* geht bei der Verarbeitung des Originalbilds ähnlich vor, beginnt aber nicht bei Umrissen, sondern homogenen Flächen, welche in der Regel Objekten entsprechen. Er besteht aus nur einem obligatorischen Schritt, der Farbreduktion (siehe Abschnitt 3.1.1). Zusätzlich können BenutzerInnen das Resultat des Effekts deutlich modifizieren, indem Parameter geändert und damit dem Effekt weitere Schritte hinzugefügt werden (siehe Abbildung 3.1). Diese werden, nach der Farbreduktion, von dem im Zuge dieser Diplomarbeit entwickelten Algorithmus in folgender Reihenfolge abgearbeitet.

Farbmodifikationen Nach der Farbreduktion können die Farben der Pixel auf verschiedene Weise modifiziert werden. Zu diesem Zeitpunkt des Algorithmus werden die Farben in einer *Look-Up-Table* (LUT) getrennt von den restlichen Pixelinformationen gespeichert und können so vor dem Zeichenvorgang effizient verändert werden (siehe Abschnitt 3.1.1). Es stehen dabei vier verschiedene Modifikationen zur Verfügung, die natürlich auch alle gleichzeitig anwendbar sind.

- Veränderung der Sättigung
- Verschiebung der Farbtöne
- Veränderung der Helligkeit
- Abbildung aller Farben auf Schwarz, Weiß und eine weitere beliebige Farbe

Detailsteigerung Dieser Schritt ermöglicht es einem Bild, den Output vorhergegangener Schritte um Details unterschiedlicher Ausprägung zu erweitern und bietet dabei verschiedene Einstellungsmöglichkeiten an.

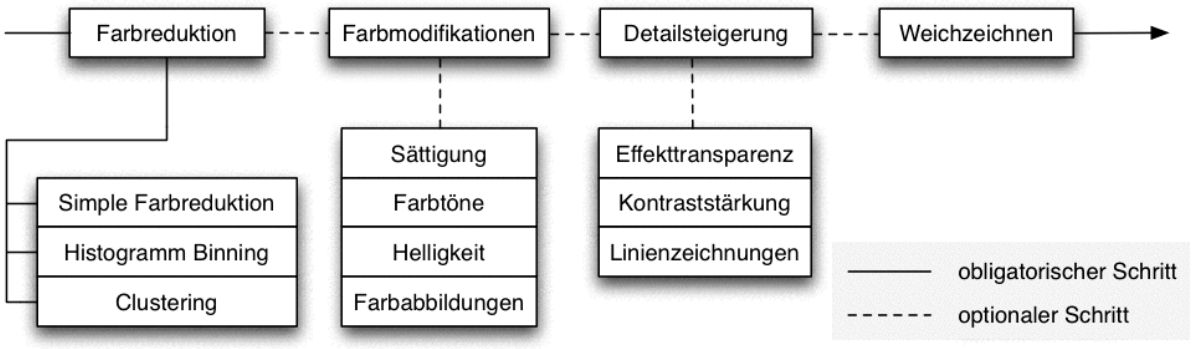

Abbildung 3.1: Der Bildeffektalgorithmus arbeitet in mehreren Schritten, welche in dieser Abbildung dargestellt werden. Abgesehen von der anfänglichen Farbreduktion, welche auf unterschiedliche Weise erfolgen kann, sind alle weiteren Schritte optional.

- Linienzeichnungen bzw. Kanten werden im Original oder im aktuellen Bild gesucht und mit gewählter Dichte und Deckkraft gezeichnet (siehe Abschnitt 3.1.3).
- Der Kontrast im Effektbild kann durch geringfügiges Streuen von Helligkeitswerten in einem spezifizierten Intervall und entsprechend niederfrequenten Bereichen des Originalbilds angehoben werden (siehe Abschnitt 3.1.4).
- Das Originalbild kann mehr oder weniger transparent auf das bisherige Ergebnis des Effekts gelegt werden.

Weichzeichnen Schlussendlich besteht die Möglichkeit, das gesamte Effektbild mittels Gauß Filter zu glätten.

3.1.1 Farbreduktion

Im Bildeffekt stehen drei verschiedene Arten der Farbreduktion zur Verfügung, die das Ziel haben, zusammenhängende Regionen mit dem Homogenitätskriterium Farbe zu produzieren und das Bild insgesamt zu vereinfachen. Im Kapitel über die Kohärenz durch Vektorisierung von Videomaterial (siehe Kapitel 2.2.4) wurde dieses Ziel durch Segmentierung erreicht. Beim *ColorReduction Bildeffekt* wird dazu ein auf *K-Means Clustering* [57] basierender Segmentierungsalgorithmus eingesetzt. Zwei weitere, simplere Methoden beruhen auf Binning der Farbkomponenten im Histogramm.

In der Literatur werden zum Erreichen dieses Ziels, neben der hier gewählten Identifikation von Häufungen, auch Effektprimitive erzeugt (siehe Kapitel 2.2.2) oder durch morphologische Operationen Details entfernt (siehe Kapitel 2.2.2). Ein wichtiges Kriterium bei der Entscheidung für diese Art der Abstraktion ist die Beeinflussbarkeit der Farbanzahl, um später

im Videoeffekt, mittels Keyframeinterpolation, auch Morphing-Effekte erzielen zu können. Die Farbanzahl bestimmt außerdem den Grad der Abstraktion und damit auch die Größe der homogenen Regionen. Die in dieser Diplomarbeit verwendeten Ansätze zur Farbreduktion werden in diesem Abschnitt ausführlich behandelt.

Simple Farbreduktion

Die *Simple Farbreduktion* quantifiziert die Farbbänder des RGB Farbmodells jeweils uniform um einen gewählten Faktor. Der Parameter für die Anzahl der Farben k muss dazu auf die Anzahl an Intervallen der Farbbänder x_R, x_G und x_B abgebildet werden. Da das Teilen der Farbbänder in gleich viele und gleich große Intervalle ($x_R = x_G = x_B$) nur in einigen wenigen Fällen möglich ist ($\sqrt[3]{k}$ ergibt nur eine ganze Zahl, wenn $k \in [8, 27, 64, 125, 216]$), wird der Teilungsprozess verallgemeinert. Die Farbbänder können in verschiedene Anzahlen an Intervallen geteilt werden. Die Kombination der für das jeweilige Intervall stehenden Werte muss jedoch weiterhin k Farben ergeben:

$$k = x_R x_G x_B \tag{3.1}$$

$$1 \leq x_R, x_G, x_B \leq k \tag{3.2}$$

Der Effekt zerlegt also die Farbanzahl k in drei nicht zwingend gleich große Faktoren x_R, x_G und x_B, welche für die Anzahl der Intervalle der Farbkomponenten Rot, Grün und Blau stehen. In der Implementierung wird dazu zunächst eine Primfaktorzerlegung in N Primzahlen p für k durchgeführt:

$$k = p_1 \cdot \cdots \cdot p_N = \prod_{i=1}^{N} p_k \tag{3.3}$$

In den meisten Fällen ergibt diese Zerlegung nicht exakt drei Faktoren p ($N = 3$), um diese auf x_R, x_G und x_B aufzuteilen. Aus diesem Grund müssen Neue ergänzt oder Vorhandene zusammengefasst werden. Der triviale erste Sachverhalt kann behoben werden, indem die übrigen Farbbänder durch nur einen Wert repräsentiert werden. Wird also beispielsweise Vier ($k = 4$) in seine Primfaktoren zerlegt, erhält man zwei Faktoren, $p_1 = 2$ und $p_2 = 2$. Da jedoch genau drei verschiedene Farbkomponenten zu kombinieren sind, um eine Farbe zu erhalten, wird ein dritter Wert benötigt. Mit dem Hinzufügen von $p_3 = 1$ ändert sich das Ergebnis der Multiplikation der Faktoren (Formel 3.3) nicht. Das dritte Farbband wird somit als ein Intervall gesehen und nur durch einen Wert repräsentiert. Auch der zweite Fall kann durch einen simplen Algorithmus gelöst werden. Sind zu viele Faktoren vorhanden, werden die niedrigsten solange paarweise miteinander multipliziert bis nur noch drei Zahlen vorhanden sind. Bei zum Beispiel $k = 12$ mit den Primfaktoren $p_1 = 2$, $p_2 = 2$, $p_3 = 2$ und $p_4 = 2$ werden zwei der Faktoren miteinander multipliziert ($p_3 = p_3 p_4$), um exakt drei Werte ($p_1 = 2$, $p_2 = 2$, $p_3 = 4$) zu be-

kommen. Die darauf folgende Verteilung der gegebenenfalls unterschiedlich großen Faktoren basiert auf der Sensibilisierung des menschlichen visuellen Systems auf verschiedene Farbtöne. So werden Grüntöne, auf die der Mensch im Allgemeinen am leichtesten reagiert, bevorzugt auf große Faktoren und Blautöne, bezüglich das Auge die geringste Auflösung hat, auf kleine Werte abgebildet [64].

Der Farbraum des Bilds wird auf die gewünschte Anzahl an Farben, welche durch Kombinationen der unteren Intervallgrenzen der Farbbänder entstehen, reduziert. Dabei müssen nicht alle Ergebnisfarben zwingend im resultierenden Bild vorkommen. So ist es möglich, dass der Farbraum zwar auf elf Farben reduziert wird, aber nur sieben davon tatsächlich in Verwendung sind.

Der Vorteil dieser Methode liegt in der Geschwindigkeit (siehe Kapitel 5.4). Aufgrund ihrer Einfachheit ist sie deutlich weniger rechenaufwändig als die anderen beiden Farbreduktionen. Bei der Qualität des Ergebnisbilds müssen allerdings deutliche Abstriche gemacht werden, da die Farbwahl - sowohl im Vergleich zu den Farben des Originalbilds als auch was die durch Farbhäufungen entstehenden Segmente betrifft - nicht optimal ist. Objekt- und Segmentgrenzen stimmen oft nicht miteinander überein. Im Videoeffekt kommt ersteres Problem besonders bei Veränderungen des Parameters für die Farbanzahl über ein Zeitintervall zum Tragen, denn benachbarte Werte können zu stark variierenden Ergebnisbildern führen. Das resultiert aus der ungleichen und für unterschiedliche Farbanzahlen variierenden Teilung der Farbbänder. Im Ergebnisvideo äußert sich diese Tatsache, besonders bei einem primen Farbparameter, in Farbsprüngen und Flackern (siehe Kapitel 4.2.1).

Histogramm Binning

Das *Histogramm Binning* erstellt im ersten Reduktionsschritt zunächst ein vereinfachtes Farbhistogramm und ermittelt dazu jeweils die Anzahl $H(.)$ der Pixel mit einer bestimmten Farbe i (siehe Abbildung 3.2). Die Summe dieser Anzahlen entspricht dabei der Anzahl Pixel N des Originalbilds.

$$N = \sum_{i=1}^{512} H(f(i)) \tag{3.4}$$

Um die möglichen Farbkombinationen (256^3) und damit den Arbeits- und Speicheraufwand während des Zählvorgangs zu reduzieren, wird jedes der drei Farbbänder (Rot, Grün und Blau) standardmäßig in acht gleich große Intervalle (Bins) geteilt. Das Mittel aus oberer und unterer Intervallgrenze repräsentiert dabei das Intervall. Die Kombination der Mittelwerte ergibt 8^3 ($= 512$, siehe Formel 3.4) Farben, wodurch bereits zu diesem Zeitpunkt eine Verminderung der Farbanzahl i um den Faktor 32768 erzielt wird. Dieses Zusammenfassen der ursprünglichen Farben und damit auch ihrer Häufigkeiten innerhalb eines Intervalls wird in Formel 3.4 durch

die Funktion $f(.)$ angedeutet.

Der zweite Reduktionsschritt wählt anschließend die häufigsten k Farben im Histogramm (siehe Abbildung 3.2, *rote Markierungen*).

$$I_R = max(H(f(i)), k) \tag{3.5}$$

Das hat unter anderem den Vorteil, dass eine exakte Abbildung vom Parameter Farbanzahl und den tatsächlichen Farbkombinationen im Ergebnisbild möglich ist. Darüber hinaus werden global wichtigere Farben bevorzugt und Details vernachlässigt. Schlussendlich werden die Pixel des Originalbilds aufgrund ihrer Farbähnlichkeit (euklidische Distanz) auf diese k Farben abgebildet und das Effektbild I_R erstellt.

Dieser Ansatz ist sowohl im Videoeffekt stabiler als auch visuell ansprechender als die Simple Farbreduktion. Aufgrund der gleichmäßigen Teilung der Farbbänder und der von den Bildinformationen abhängigen Farbwahl haben ähnliche Bilder auch im Effektbild wahrscheinlicher die gleichen Farben. Bei der Verminderung oder Erhöhung der Farbanzahl im selben Bild sind nicht so gravierende Unterschiede wie bei der Simplen Farbreduktion möglich.

Clustering

Der dritte Ansatz sieht das Problem der Farbreduktion als Segmentierungsproblem. Diese Sichtweise wurde bereits von anderen cartoonartigen Effekten (siehe Kapitel 2.2.4) angenommen und unterschiedlich umgesetzt. In der Video Tooning Literatur werden häufig Varianten der Mean Shift Segmentierung eingesetzt, welche zwar gute Ergebnisse liefern, aber nicht erlauben, die Anzahl der Farben zu modifizieren, und in der Regel lange Laufzeiten haben. Das in dieser Diplomarbeit angestrebte Ziel ist es, ein Bild im RGB Raum auf k Farben zu reduzieren und diese möglichst optimal zu wählen, sodass der globale Fehler möglichst gering ausfällt. In [62] wird einer der einfachsten Segmentierungsalgorithmen, der *k-Means Algorithmus* [57], dafür empfohlen, weshalb er in der Farbreduktion mittels Clustering Verwendung finden soll. Folgender Pseudocode skizziert seinen Ablauf.

Listing 3.1: Pseudocode zum k-Means Algorithmus

```
1 Initialisiere(Zentren);
2 Solange sich die Zentren noch ändern und
3         die maximale Anzahl an Iterationen nicht erreicht ist  {
4                 Gehe zum nächsten Pixel;
5                 Pixelfarbe ermitteln;
6                 Welches der Zentren passt am besten zur Farbe?
7                 Erneuere die Position der Zentren;
8 }
```

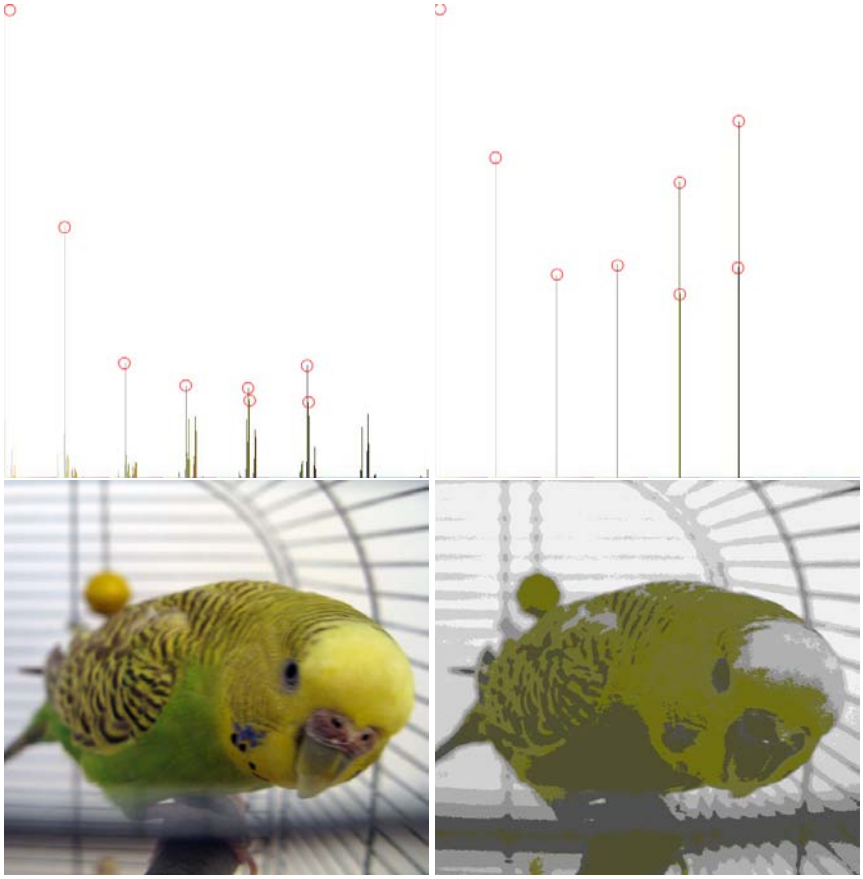

Abbildung 3.2: Die *erste Spalte* enthält das Testbild (*unten*) und das dazugehörige Histogramm (*oben*). In der *zweiten Spalte* ist das Ergebnis des zweiten Reduktionsschritts zu sehen. Die Farben des Originalbilds wurden auf die häufigsten Farben (in diesem Fall acht) abgebildet. Im Histogramm (*darüber*) sind die häufigsten Bins *rot markiert*.

Bei diesem Vorgang werden Pixel mit ähnlichen Farben x_i gruppiert, wobei die Gruppenfarbe, ein sogenanntes Zentrum c_j, für jeden zusätzlichen Wert angepasst wird. Die neue Farbe entspricht dem Mittel aller Farben der diesem Zentrum zugeordneten Pixel. Es besteht die Möglichkeit, dabei einzelne Farbbänder verschieden zu gewichten, worauf im Effekt aber nicht zurückgegriffen wird. Eine Wiederholung dieser Schritte bezweckt, folgende Zielfunktion, und damit den globalen Fehler, zu minimieren. Die Differenz des Originalbilds vom segmentierten Bild mit den Farben c_j soll möglichst gering sein.

$$I_R = \sum_{j=0}^{k} \sum_{i=1}^{N} ||x_i - c_j||^2. \tag{3.6}$$

Das Clustering terminiert, wenn die Startzentren stabil sind oder die maximale Anzahl an Iterationen erreicht ist.

Die Anzahl der gewählten Farben bzw. Cluster ist einer der Parameter, die BenutzerInnen verändern können, um das Ergebnis maßgeblich zu beeinflussen. Dies gilt in der Literatur als gravierender Nachteil des k-Means Algorithmus [37][96], wird aber in dieser Diplomarbeit bewusst als Vorteil genützt. Ein weiterer Kritikpunkt am k-Means Algorithmus ist die Wahl der Startzentren, welche normalerweise per Zufall in der Menge der Pixelfarben erfolgt. Für den Bild- und damit später auch für den Videoeffekt erweist sich dieser Ansatz als ungeeignet. Jede Farbreduktion hätte andere Grundfarben, wodurch das Ergebnis von ein und demselben Bild mit gleich vielen Farben anders aussehen könnte. Aus diesem Grund erfolgt im Bildeffekt die Wahl der Startzentren genau wie die der Farben beim *Histogramm Binning* (siehe Formel 3.5). Die k häufigsten Farben im Originalbild werden als Startzentren gewählt, wodurch die zufällige Entscheidung der klassischen Variante von einer globalen Entscheidung ersetzt wird. Wie beim Mean Shift (siehe Abschnitt 2.2.4) dienen bei dieser k-Means Modifikation Maxima, die sich aus allen Datenpunkten ergeben, als Basis für weitere Berechnungen.

Dieser Ansatz produziert deutlich abgeschlossenere Flächen als die beiden anderen, ist aber auch am aufwändigsten. Durch die räumliche und farbliche Homogenität der Ergebnisse werden sowohl der Bild- als auch der Videoeffekt deutlich stabiler.

3.1.2 Distanzberechnungen für Farbabbildungen

Die aus der Farbreduktion gewonnenen Grundfarben c_j können in einem weiteren, optionalen Schritt auf drei Farben (Schwarz s, Weiß w und eine weitere beliebige Farbe b) abgebildet werden, sodass jede dieser drei Farben im Bild mindestens einmal vorkommt. Um das zu gewährleisten, wird eine dreidimensionale Distanzmatrix benutzt. Die Dimensionen stehen dabei für die unterschiedlichen Farbkomponenten (Rot R, Grün G und Blau B) der zu vergleichenden Farben. Die Elemente der Matrix für jede Dimension entsprechen den jeweiligen Differenzen der neuen Farbkomponenten von jenen der Grundfarben (siehe Abbildung 3.3).

Um für jedes Farbpaar einen ausschlaggebenden Wert zu bekommen, wird daraufhin die euklidische Distanz berechnet. Die Einträge der roten, der grünen und der blauen Matrizen werden zu diesem Zweck quadriert und addiert. Die photometrische Differenz von Schwarz zu den übrigen Farben d_{sj} wird unter Berücksichtigung aller Farbkomponenten auf beispielsweise folgende Art und Weise berechnet:

$$d_{sj} = \sqrt{(Bs - Bc_i)^2 + (Gs - Gc_i)^2 + (Rs - Rc_i)^2} \tag{3.7}$$

3.1.3 Kantenerkennung

Linienzeichnungen für Details werden im Bildeffekt durch drei simple Schritte wahlweise vom Original- oder Effektbild, in dem diese weitgehend den Segmentgrenzen entsprechen, generiert.

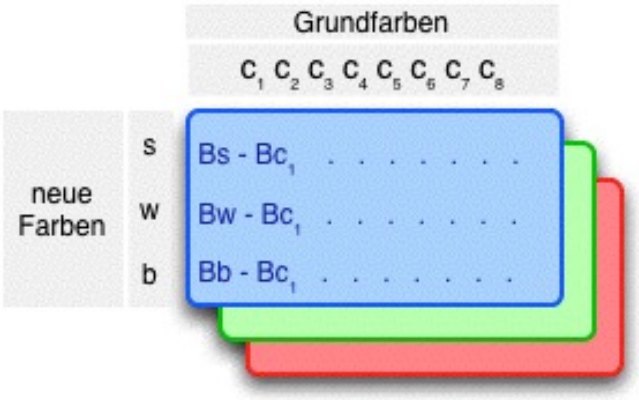

Abbildung 3.3: Die drei Matrizen (für Rot, Grün und Blau) beinhalten die Differenzen der neuen Farben von jenen der Grundfarben.

Der erste Schritt ist die Bestimmung der horizontalen und vertikalen Ableitungen, aus deren Information ein Kantenbild gewonnen werden kann. Das Kantenbild wird durch Faltung mit dem horizontalen h_x und vertikalen h_y Sobel Operator angenähert: [85]

$$h_y = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{3.8}$$

$$h_x = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{3.9}$$

Als zweiter Schritt der Kantenerkennung werden Kantenpixel mit einem globalen Schwellenwert τ von Nicht-Kantenpixeln getrennt. Dieser Wert wird mittels automatischer Schwellenwertselektion durch Histogrammanalyse bestimmt. Normalerweise wird im Grauwerthistogramm nach einem Minimum zwischen zwei Maxima oder Ähnlichem gesucht [74]. Dieser Ansatz ist für ein Kantenbild jedoch nicht zielführend, da der typische Verlauf seines Histogramms (siehe Abbildung 3.4) in der Regel nur ein Maximum im dunklen, kantenlosen Bereich hat und im hellen Bereich immer niedriger wird [31]. Die für den *ColorReduction Bildeffekt* und damit auch für die zugehörigen Videoeffekte entwickelte Methode nützt diese Verteilung aus. Das bereits generierte Kantenbild wird im ersten Schritt als Grauwerthistogramm mit 128 Bins - es werden also jeweils zwei Grauwerte zu einem zusammengefasst - dargestellt. Aus diesem Histogramm können anschließend Schwellenwerte ermittelt werden. Dazu gibt es in dem implementierten System folgende Möglichkeiten:

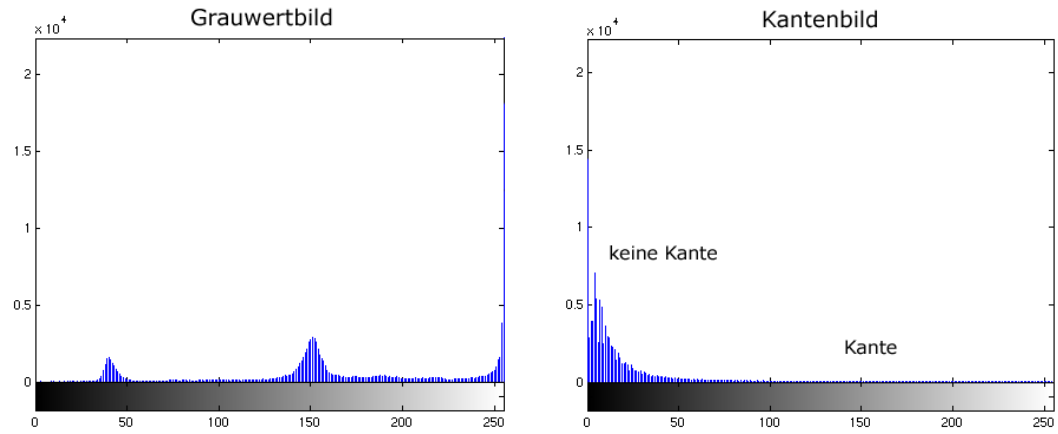

Abbildung 3.4: Das *linke Histogramm* wurde aus einem Grauwertbild erstellt. Ein geeigneter Schwellenwert für die Trennung von Vorder- und Hintergrund könnte bei einem Minimum zwischen zwei Maxima, also beispielsweise um 200 liegen. Das *rechte Histogramm* stammt von einem Kantenbild mit hellen Konturen auf schwarzem Hintergrund. Die Suche nach einem Minimum zwischen zwei Maxima ist aufgrund des typischen Verlaufs mit nur einem Maximum nicht möglich.

- Die z.B. 90 Prozent der Pixel, die beim typischen Verlauf im schwarzen, kantenlosen Bereich liegen, werden von der weiteren Berechnung ausgeschlossen. Die Mitte des Grenzbins zu den übrigen, unteren zehn Prozent ($p_{bottom} = 10$) kann als Schwellenwert verwendet werden.
- Alternativ können die Bins zusätzlich nach der Häufigkeit ihres Vorkommens sortiert und von den unteren 80 Prozent der Pixel das gewichtete arithmetische Mittel oder
- das Mittel der oberen und unteren Binintensitäten berechnet werden. Das liefert in der Regel Ergebnisse bei Intensitätswerten knapp unter 128.

Der Effekt verwendet für sinnvolle Prozentwerte ($p_{bottom} \in [3, 20]$) den ersten Ansatz, sodass BenutzerInnen die Häufigkeit der Kanten geringfügig beeinflussen können. Prinzipiell wird in der Implementierung dieser Diplomarbeit bei allen anderen Fällen auf den zweiten Ansatz zurückgegriffen. Da aber der Bildeffekt im User Interface nicht vorsieht, Werte außerhalb dieses Bereichs zu wählen, kommt der zweite Ansatz in der Praxis nicht zum Einsatz.

Die finalen Kanten werden mittels Schwellenwertbildung extrahiert und zusätzlich die Differenz der addierten Gradienten G_x, G_y, also der Kantenstärke, vom Schwellenwert τ mit

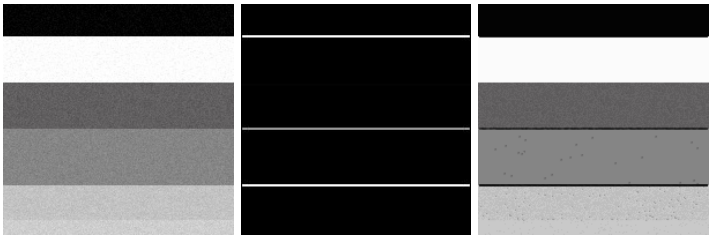

Abbildung 3.5: Das *erste Bild* ist das Originalbild mit Gauß'schem Rauschen. Das *zweite Bild* zeigt das Ergebnis der Faltung mit einem vertikalen Sobel Filter. Das *letzte Bild* ist das Ergebnis der Kantenerkennung nach Anwendung der geglätteten Stufenfunktion mit $\phi_e = 1.75$.

einem *Schärfeparameter* ϕ_e gewichtet:

$$E(\vec{x}, \phi_e, \tau) = \begin{cases} 1, & \text{wenn } \tau - (G_x(\vec{x}) + G_y(\vec{x})) > 0 \\ 1 + \tanh(\phi_e(\tau - (G_x(\vec{x}) + G_y(\vec{x})))), & \text{sonst} \end{cases} \tag{3.10}$$

Das Abdunkeln der Kanten entsprechend dem berechneten Wert $E(.)$ erfolgt mittels Multiplikation mit der Helligkeit (HSB Farbraum) im farbreduzierten Bild. Ist der Gradient eines Pixels $\vec{x}$ kleiner als der Schwellenwert, wird nach Funktion 3.10 das bereits farbreduzierte Bild an dieser Stelle nicht geändert (Multiplikation des Helligkeitswerts dieses Pixels mit $E(.) = 1$). Bei Gradienten, die größer als der Schwellenwert sind, wird im zweiten Teil der Stufenfunktion die Helligkeit der Kante bestimmt. Funktion 3.10 glättet so, analog zu der Stufenfunktion der Video-Abstraktion in [93] (siehe Kapitel 3.3.2), die Enden sowie Seiten und bewirkt, dass harte Kanten deutlich dunkler gezeichnet werden als weiche (siehe Abbildung 3.5). Sie nutzt dabei praktische Eigenschaften des Tangens Hyperbolicus aus. Er ist im Wertebereich [-1 ,1] definiert, sodass der negative Ausdruck $\tau - (G_x(\vec{x}) + G_y(\vec{x}))$ im zweiten Teil der abgerundeten Sprungfunktion 3.10 für positive ϕ_e auch nach Anwendung des Tangens Hyperbolicus negativ bleibt und zusätzlich $0 \leq E(.) \leq 1$ gilt. Für große bzw. durch den Schärfeparameter stark vergrößerte Differenzen konvergiert das Ergebnis gegen Null. Die anschließende Multiplikation des Helligkeitswerts eines Pixels im Effektbild mit $E(.)$ hat bei großen Gradienten also eine Verdunkelung des Pixels zur Folge.

Optional können BenutzerInnen zusätzlich die Deckkraft der Kanten bestimmen. Umgesetzt wird das durch gewichtete Mittelung der Originalfarbe des Pixels und der berechneten Kantenfarbe. Dadurch verschwinden schwache Kanten zuerst.

3.1.4 Kontrastverstärkung

Um einerseits Kontraste zu verstärken und andererseits Details in Form einer geringfügigen Streuung von Helligkeitswerten hinzuzufügen, wird eine Abbildung der HSB-Helligkeitsinfor-

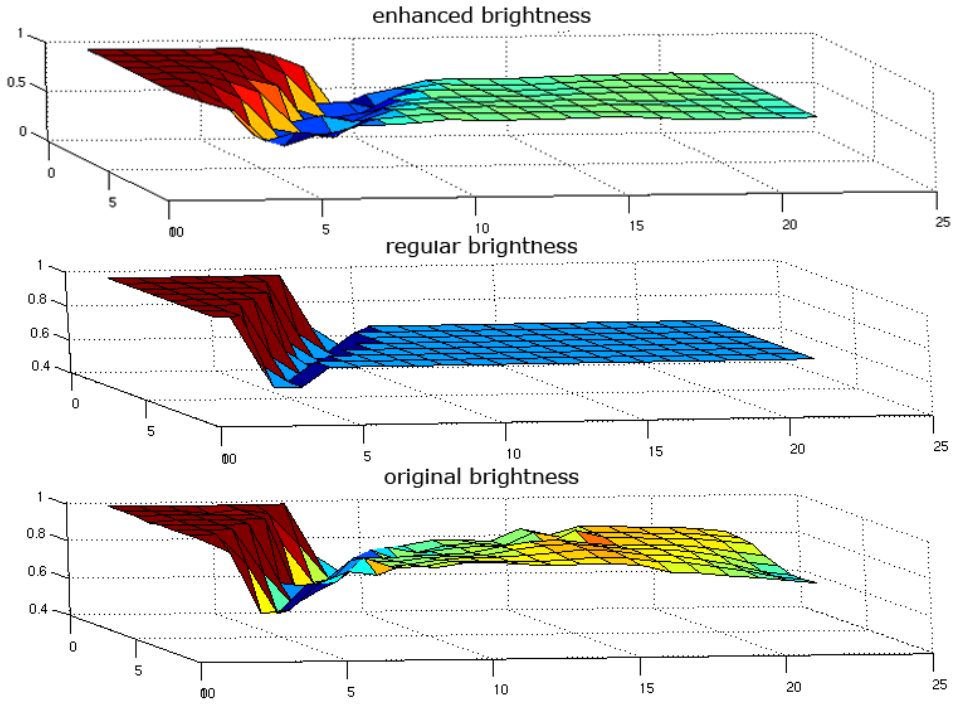

Abbildung 3.6: Die mit MATLAB erzeugten *Surface Plots* zeigen die Helligkeitswerte eines Originalbilds (*unten - original brightness*), die eines farbreduzierten Bilds (*in der Mitte - regular brightness*) und einer Variante mit gestreuten Werten (*oben - enhanced brightness*).

mation des Originalbilds $b(.)$ auf die des Effektbilds $B(.)$ definiert:

$$B_{new}(\vec{x}) = B(\vec{x}) - b(\vec{x}, \tau, G_x, G_y) \tag{3.11}$$

$$b(\vec{x}, \tau, G_x, G_y) = \begin{cases} \omega \frac{\sqrt{G_x^2+G_y^2}}{\tau}, & \text{wenn } \sqrt{G_x^2 + G_y^2} < \tau \\ \omega + 0.2, & \text{sonst} \end{cases} \tag{3.12}$$

Diese Funktion greift auf eine Finesse von Winnemöller et al. [93] zurück (siehe Abschnitt 3.3.3). Mittels Sobel Filter ermittelte Gradienten (G_x und G_y) dienen als Gewichte der Streuung. Die in dieser Diplomarbeit definierte Abbildung identifiziert zwar wie die Quantifizierung der Helligkeitsdaten in [93] durch die berechneten Gradienten niederfrequente Bereiche $\sqrt{G_x^2 + G_y^2} \in [0, \tau]$, jedoch nicht im abstrahierten, sondern im originalen Bild $b(.)$. Funktion 3.12 bildet im ersten Teil der Sprungfunktion die Gradienten in $[0, \tau]$ auf ein von BenutzerInnen definiertes Intervall $[0, \omega]$, welches die maximale Änderung der initialen Farbe definiert, ab. Das aus dieser Funktion resultierende Ergebnis entspricht der vorzunehmenden Abweichung von der aktuellen Helligkeit im Effektbild. Funktion 3.11 subtrahiert das Ergebnis von dem entsprechenden Helligkeitswert im Effektbild, sodass nicht eine Verminderung, sondern eine Erhöhung der Helligkeitsvariation entsteht (siehe Abbildung 3.6).

3.2 ColorReduction Videoeffekte

In diesem Abschnitt werden die vom *ColorReduction Bildeffekt* abgeleiteten Videoeffekte und die dabei stattfindende Transformation näher erläutert. Die Videoeffekte konzentrieren sich vor allem auf das Verhindern von Inkonsistenzen, welche hauptsächlich durch Farbsprünge entstehen. Dabei werden exemplarisch wichtige kohärenzverbessernde Konzepte aus der Literatur, wie die globalere Verarbeitung und das Einführen einer Entscheidungsfunktion, aufgegriffen. Bereits bei einigen Farbreduktionen des Bildeffekts (siehe Abschnitt 3.1.1) wurden Maßnahmen zur Farbsprungreduzierung getroffen und dadurch eine gute Basis für die Transformation zu einem Videoeffekt erarbeitet. So wurde beispielsweise beim Clustern eine globale Heuristik für die Wahl der Startzentren gewählt, die für ähnliche Bilder, wie benachbarte Frames, auch ähnliche Ergebnisse liefert.

Prinzipiell gibt es zwei Möglichkeiten, den Bildeffekt auf ein Video anzuwenden. Entweder auf alle Frames gleichzeitig, als ein 3D Raum-Zeit-Volumen (z.B. [43] oder [77]), oder pro Frame, gefolgt von einem Schritt, der zeitlichen Zusammenhang herstellt (z.B. [52]) und damit versucht, Kohärenz zwischen benachbarten Frames zu erzwingen. In der Computer Vision Literatur wird der zweite Ansatz auch als „2D plus Zeit "$(2D + t)$ Technik bezeichnet [45][9]. Bei den Videoeffekten in dieser Diplomarbeit wird vor allem auf die letztere Technik zurückgegriffen. Zum Einen legt das vorhandene Framework dies nahe, da üblicherweise die Ergebnisse einzelner Frames schon während der Verarbeitung des Videos in einer Vorschau zu sehen sind. Zum Anderen wäre die Forderung nach Keyframeinterpolationen beim zweiten Ansatz nur schwer zu realisieren gewesen. Weiters sind auch die übrigen Schritte des Bildeffekts zu beachten. Für diese müssten zusätzlich Adaptierungen, welche die Zeitachse miteinbeziehen, zur Verfügung gestellt werden. Es gibt einige wenige Beispiele für Erweiterungen von Filtern der Bildbearbeitung zwecks Anwendung auf Sequenzen, wie z.B. den Mittelwertfilter von [73]. Ein weiteres Argument bei der Wahl des Ansatzes war, dass für Farbreduzierung und zusätzliche Abstraktionsschritte wie Konturenbildung deutlich mehr Literatur bezüglich Einzelbilder vorhanden ist.

Als Inputvideo werden, wie bei vielen anderen Videoeffekt-Systemen (z.B. [9], [49] oder [77]), einzelne Shots angenommen, also Videoclips, die frei von Schnitten, schnellen Kamerabewegungen, Überblendungen oder ähnlichen Techniken sind, sodass sich adjazente Frames in der Regel sehr ähnlich sehen. Es gibt verschiedene Methoden, wie Schnitterkennung, welche dazu eingesetzt werden können, eine Videosequenz in geeignete Ausschnitte zu zerlegen [42][97].

3.2.1 ColorReduction Videoeffekt

Der *ColorReduction Videoeffekt* zeigt eine Möglichkeit, wie versucht werden kann, Artefakte, die durch Frame-zu-Frame Anwendung des Bildeffekts entstehen, zu reduzieren. Der Einsatz ei-

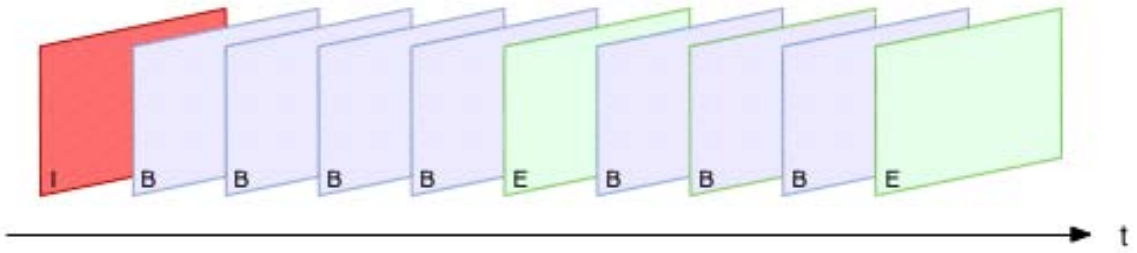

Abbildung 3.7: Mit dem ersten Frame (*rot, I*) wird der Videoeffekt initialisiert. Das erste Frame wird, zusätzlich zum Endframe des ersten Zyklus, dem Bildeffekt zur Verarbeitung übergeben. Für die Binnenframes (*blau, B*) werden die Farben der sie umgebenden Endframes (*E, grün bzw. I, rot*) interpoliert. Die Anzahl der Binnenframes kann bei aufeinander folgenden Zyklen variieren, da auch ihre Werte in den Keyframes interpoliert werden.

nes dynamischen Erneuerungszyklus soll einerseits die Geschwindigkeit des Videoeffekts etwas erhöhen, andererseits aber vor allem auch ermöglichen, mehr als nur den diskreten Zeitpunkt des aktuellen Frames in die Berechnungen einzubeziehen. Je nach der im Keyframe gesetzten Erneuerungsrate wird im Bildeffekt eine der in Abschnitt 3.1.1 beschriebenen Farbreduktionen angewandt, oder die Grundfarben der umgebenden farbreduzierten Frames werden interpoliert.

Wie bei einer Reihe anderer Videoeffekte, denen Bildeffekte als Basis dienen (u.a. [82] [28][34]), verarbeitet auch dieser Effekt im Regelfall das erste Frame anders als die kommenden. Datenstrukturen werden initialisiert und nicht nur das erste Frame, sondern anschließend auch das am Ende des Erneuerungszyklus liegende, werden mit den entsprechenden Effektoptionen durch den Bildeffekt stilisiert. Die übrigen Frames werden in folgende zwei Klassen geteilt (siehe Abbildung 3.7):

- Ein Einzelbild mit der Position zwischen zwei vom Bildeffekt bearbeiteten Frames gehört zur ersten Kategorie. Im Folgenden werden solche Bilder als *Binnenframes* bezeichnet.
- Zur zweiten Klasse, den *Endframes*, zählen Einzelbilder, die einen Zyklus beenden und die Grunddaten durch eine Anwendung der Farbreduktion im Bildeffekt erneuern. Sie begrenzen eine Reihe von Binnenframes.

Binnenframes werden anders als Endframes verarbeitet. Jeder Grundfarbe vom Endframe des vorhergehenden Erneuerungszyklus wird mindestens eine ähnliche oder gleiche Farbe des kommenden Endframes zugeordnet. Um das auch bei unterschiedlicher Anzahl an Farben zu gewährleisten, wird eine Distanzmatrix (siehe Abschnitt 3.1.2) verwendet.

Im ersten Durchlauf werden, ausgehend vom Frame mit weniger Farben, einmalige Farbpaare gebildet. Der Algorithmus sucht dabei in jeder Spalte nach dem Eintrag mit der geringsten Differenz (siehe Abbildung 3.8). Ist diese Reihe schon vergeben, ermittelt eine einfache Heu-

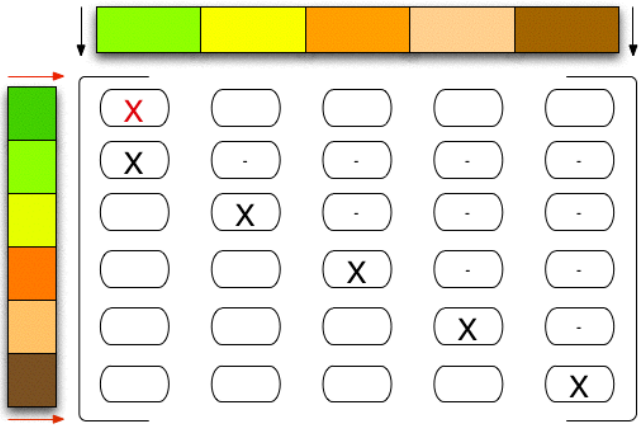

Abbildung 3.8: Diese Abbildung visualisiert wie mit Hilfe einer Distanzmatrix Farbpaare gebildet werden. Das Frame mit weniger Farben beginnt (*schwarzer Pfeil*). Pro Spalte wird der kleinste Eintrag einer noch nicht gewählten Reihe gesucht und markiert (*schwarzes X*). Diese Markierung weist auf ein gefundenes Farbpaar hin. Anschließend werden ausgehend von den übrig gebliebenen Reihen (im Frame mit mehr Farben, *roter Pfeil*) die ähnlichsten Farben gesucht, wobei eine gefundene Farbe auch zusätzlich einer weiteren Farbe zugeordnet sein kann (*rotes X*).

ristik, ob die bereits zugeordnete Farbe dieser Reihe eher der aktuellen oder der alten Farbe entspricht und tauscht sie gegebenenfalls aus, um die Farbzuordnung zu optimieren.

Der zweite Durchlauf geht vom Frame mit mehr Farben aus. Die übrig gebliebenen Reihen, in Abbildung 3.8 wäre es lediglich die mit den Differenzen von Dunkelgrün, wählen nun jeweils die Spalte mit der geringsten Differenz. Die Einschränkung der Eindeutigkeit der Wahl ist bei diesem Durchlauf aufgehoben, sodass auch Farben, die bereits zugeteilt wurden, gewählt werden können.

Die Gruppen einander entsprechender Farben werden anschließend durch ihr Mittel repräsentiert und nach der Nähe zu den Endframes gewichtet. Die Zu- bzw. Abnahme der Farben wird diesbezüglich unterschiedlich behandelt. Bei ersterer Situation vermischen sich jeweils zwei Farben, eine vorhergehende c_p und eine kommende c_n, zu einer Grundfarbe c_{new} des aktuellen Frames.

$$c_{new} = c_p x + c_n(x-1) \tag{3.13}$$

Die Faktoren x bzw. $(x-1)$ geben dabei prozentuell die Nähe zum vorhergehenden bzw. kommenden Binnenframe an und steuern die Farbmischung. Sie sollen Farbsprünge am Ende eines Zyklus verhindern.

Das Verhalten bei Abfall der Farbanzahl ist konträr, wenige Farben werden auf mehrere abgebildet. Die neue Grundfarbe c_{new} des aktuellen Frames ergibt sich also aus der Mischung einer vorhergehenden c_p und N kommender Grundfarben $c_{nx1}, c_{nx2}, \ldots c_{nxN}$. Um trotzdem eine

Gewichtung nach der Nähe der Endframes zu ermöglichen (x bzw. $(x-1)$), wird das arithmetische Mittel der entsprechenden Farben des neueren Binnenframes berechnet.

$$c_{new} = c_p x + \frac{c_{nx1} + c_{nx2} + ... + c_{nxN}}{N}(x-1) \tag{3.14}$$

Im letzten Schritt werden eine LUT konstruiert und Pixel-Label erstellt. Aufgrund der Farbähnlichkeit (euklidische Distanz) wird dabei für jedes Pixel eine Referenz auf die ähnlichste Grundfarbe gespeichert. Beides wird dem Bildeffekt übergeben, sodass er von dieser Farbreduktion ausgehend gegebenenfalls weitere Schritte des Effekts durchführen kann.

3.2.2 MultiCluster Videoeffekt

Videoeffekte wie Video Cubes [43], Wasserfarben-Effekt [46] oder Video Tooning [77] haben gezeigt, dass eine globalere Verarbeitung des Videomaterials Verbesserungen mit sich bringen kann. Aufgrund dieser Erkenntnis stellt der *MultiCluster Videoeffekt* durch das gemeinsame Verarbeiten von mehreren Frames einen zeitlichen Zusammenhang her. Auch dieser Videoeffekt hat das Ziel, Farbsprünge im Ergebnisvideo zu vermeiden.

Wie viele vorhergehende und nachfolgende Frames in die Verarbeitung einbezogen werden, bestimmen BenutzerInnen in der BenutzerInnenoberfläche durch Auswahl entsprechender Werte in Keyframes. Die Anzahl muss weder für alle Frames gleich, noch für das gesamte Video gesetzt werden, sondern wird genauso wie andere Werte für ein aktuelles Frame interpoliert. Die Extremeinstellung von null Frames davor und danach führt zum selben Ergebnis wie eine Frame-zu-Frame Anwendung des Bildeffekts.

Der Algorithmus des *MultiCluster Videoeffekts* ist relativ simpel. Er konstruiert ein Bild, das aus allen benötigten Frames besteht, und führt darauf eine der Farbreduktionen aus (siehe Abbildung 3.9). Für Anfangs- und Endframes wird eine Randbehandlung notwendig. Dazu kann eine von folgenden Methoden ausgewählt werden:

- Ist in einer Richtung auf der Zeitachse ausgehend vom aktuellen Frame nicht die spezifizierte Anzahl an Einzelbildern vorhanden, werden so viele wie möglich einbezogen. Zum Zeitpunkt Null werden also keine vorhergehenden Frames in die Farbreduktion involviert.

- Das Einbeziehen einer Richtung (vor oder nach dem aktuellen Zeitpunkt) erfolgt nur, wenn genug Frames vorhanden sind. Anfangs werden also so lange keine Vorgängerframes verarbeitet, bis die gewünschte Anzahl existiert.

Das Ergebnis der Farbreduktion (LUT sowie Label der Pixel) wird dem Bildeffekt übergeben, sodass dieser die weitere Verarbeitung übernehmen kann.

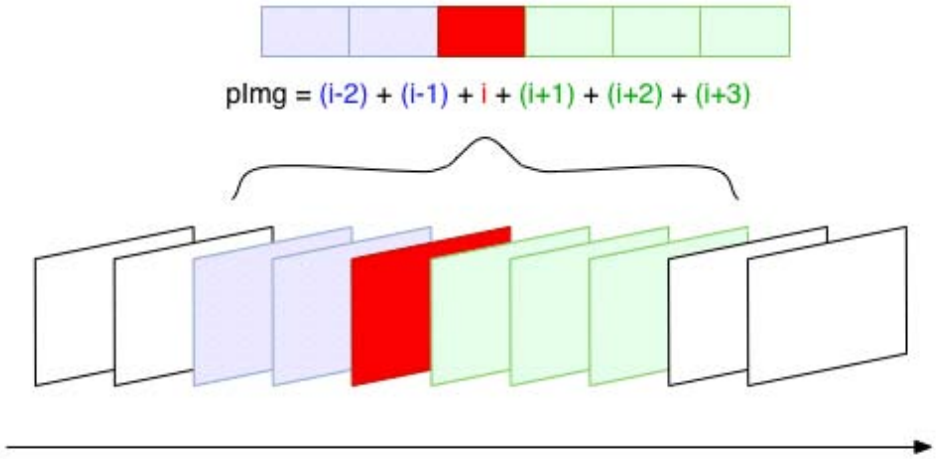

Abbildung 3.9: Zusätzlich zum aktuellen Frame (*rot*) wird auch eine beliebige Anzahl an vorhergehenden (*blau*) und nachfolgenden Frames (*grün*) verarbeitet. In einem Zwischenschritt werden die einzelnen Frames zu einem Bild (*pImg*) zusammengefügt.

3.2.3 FrameDifferencingColorReduction Videoeffekt

Der *FrameDifferencingColorReduction*, kurz *FrameDifferencing*, *Videoeffekt* bezieht sich auf Ansätze, welche Differenzen, sei es direkt durch Frame Differencing (siehe Kapitel 2.2.3) oder indirekt durch die Berechnung des Optical Flows (siehe Kapitel 2.2.2), und darauf basierend Entscheidungsfunktionen definieren. Der Effekt greift dieses allgemeine Konzept zur Verminderung flimmernder Effektprimitive auf und adaptiert es in eine speziell für Transformationen ausgehend vom *ColorReduction Bildeffekt* angepasste Variante. Ein wesentlicher Unterschied zwischen den eben erwähnten Algorithmen (siehe Kapitel 2.2.2 und 2.2.3) und dem in dieser Diplomarbeit präsentierten ist die Sichtweise. Während bereits vorhandene Ansätze das neue Effektbild erst aufgrund der gewonnenen Information aus Primitiven des alten Effektbilds erstellen oder es sukzessive erneuern, abstrahiert der *FrameDifferencing Videoeffekt* das aktuelle Frame zunächst unabhängig und verbessert es mit Hilfe seines Vorgängers.

Anfänglich wird das erste Frame mit dem herkömmlichen Bildeffektalgorithmus verarbeitet. Für zwei adjazente Frames (allgemein I) mit gleich vielen Farben werden sowohl in Effekt- als auch Originalsequenz pixelweise die Farbdifferenzen berechnet und bezüglich des gewählten Schwellenwerts τ als verändert oder gleichbleibend markiert (siehe Abbildung 3.10):

$$r(I_x, I_{x+1}) = I_x - I_{x+1} > \tau \tag{3.15}$$

Die Kombination des Verhaltens an der Position eines Pixels in Original- O und Effektsequenz E kann dazu verwendet werden, potentiell flimmernde Pixel zu erkennen. So weisen zum Beispiel im einfachsten Fall übereinstimmende Ergebnisse auf eine korrekte Farbwahl hin. In den

beiden anderen Fällen ist die Farbreduktion zu berücksichtigen. Es muss möglich sein, eine Veränderung im Original, nicht jedoch im Effektbild, zu erkennen, da im Effektbild die Differenzen höher ausfallen können. Die umgekehrte Kombination, gleiches Original- und verändertes Effektpixel, ist hingegen ein Indiz für Flimmern. Diese Bedingungen werden für ein potentiell nicht flimmerndes Pixel durch folgenden Ausdruck zusammengefasst:

$$R_1 = \neg r(O_x, O_{x+1}) \wedge r(E_x, E_{x+1}) \tag{3.16}$$

$\neg r(.)$ entspricht dabei dem Ergebnis aus Funktion 3.15, welches auf veränderte Pixel hinweist. Bei der Schreibweise $r(.)$ wird ein gleichbleibendes Pixel erwartet.

Um Artefakte wie sichtbare Ränder bewegter Regionen aus dem alten Effektbild, welche aufgrund voreilig getroffener Entscheidungen entstehen würden, zu verhindern, werden in einem weiteren Schritt das Verhalten von Nachbarpixel N in Original- und Effektbild (siehe Formel 3.17) sowie ihre photometrische Distanz zum aktuellen Pixel in die Entscheidung einbezogen (siehe Formel 3.18). Schon alleine der Zusatz, der Tendenz im Verhalten der Nachbarpixel zu folgen, führt zu einer sichtbaren Verbesserung im Ergebnis. Ist die Anzahl der veränderten Pixel in der Nachbarschaft kleiner als die Anzahl der gleichgebliebenen Pixel, wird das Pixel im Zentrum nicht als verändertes Pixel markiert:

$$\begin{aligned} R_2 \quad = \quad & \sum_N r(E_x, E_{x+1}) + (\sum_N r(O_x, O_{x+1}) < \\ & \sum_N \neg r(E_x, E_{x+1}) + \sum_N \neg r(O_x, O_{x+1}) \end{aligned} \tag{3.17}$$

Wird weiters die Ähnlichkeit der mittleren Farbe der Nachbarpixel im alten und neuen Effektbild zum aktuellen Pixel $d_N(.)$ eingebunden, werden die erwähnten Artefakte weitgehend eliminiert und Flimmern sichtbar reduziert. Eine Aktualisierung des Pixels erfolgt also nur, wenn seine neue Farbe im aktuellen Frame den Nachbarpixeln ähnlicher sieht als seine alte Farbe:

$$R_3 = d_N(E_x) > d_N(E_{x+1}) \tag{3.18}$$

Zusammen mit den bereits beschriebenen Beschränkungen resultiert dies in folgendem allgemeinen Test:

$$R(\vec{x}, \tau) = \begin{cases} E_x, & \text{wenn } R_1 \wedge R_2 \wedge R_3 \\ E_{x+1}, & \text{sonst} \end{cases} \tag{3.19}$$

Nur bei Verstärkung der anfänglich getroffenen Annahme R_1 durch die hinzugekommenen Bedingungen R_2 und R_3 wird Flimmern erkannt und das Pixel nicht aktualisiert, also der Farbwert des alten Effektbilds beibehalten.

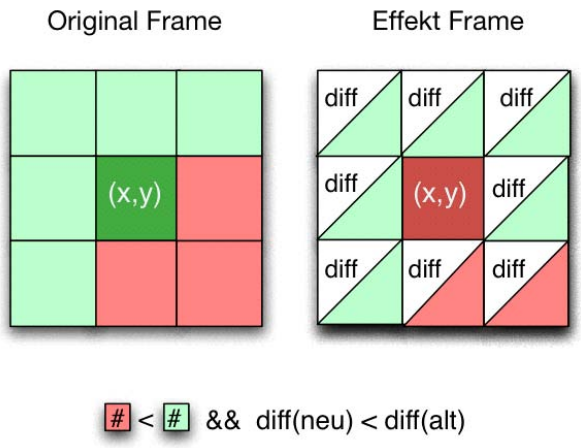

Abbildung 3.10: Diese Abbildung visualisiert die verschiedenen Einflussgrößen der Entscheidungsfunktion im *FrameDifferencing* Videoeffekt. Wird in adjazenten Effekt Frames eine Änderung (*rot*) bei gleichbleibendem (*grün*) Originalpixel erkannt, muss eine Entscheidung für oder gegen die Aktualisierung des aktuellen Pixels *(x,y)* getroffen werden. Dabei geben das Verhältnis von veränderten zu gleichbleibenden Nachbarpixel und ihre farbliche Differenz (*diff*) im neuen und alten Frame zum aktuellen Pixel den Ausschlag.

Auf Transparenz, deren Einsatz in [28] zu einem äußerst robusten Algorithmus führt, wird in diesem Ansatz verzichtet, da mit sehr kleinen Effektprimitiven, nämlich einzelnen Pixel, gearbeitet wird und bei diesem Algorithmus keine Löcher entstehen können, die langsam auszufüllen wären. Der *FrameDifferencing Videoeffekt* stellt nicht nur die Erweiterung der sequenziellen Anwendung des Bildeffekts dar, sondern kann auch in Kombination mit dem *ColorReduction Videoeffekt* angewandt werden und so gleichzeitig allgemeine sowie variierende Farbzuordnungen reduzieren (siehe Kapitel 4.3).

3.3 LabColorReduction Bildeffekt

Der *LABColorReduction Bildeffekt* adaptiert den in Kapitel 2.2.5 bzw. in [93] beschriebenen Videoeffekt als Bildeffekt. Gemäß dem Algorithmus der Cartoon Abstraktion werden für das cartoonartige Aussehen folgende Schritte ausgeführt: [93]

1. Der Algorithmus beginnt mit der einmaligen Anwendung eines Bilateralen Filters [66].
2. Darauf startet parallel die Kantenerkennung mittels DoG Operator [58] und
3. ein weiteres Filtern, gefolgt von der Reduktion der Helligkeitswerte.
4. Das farbreduzierte Bild wird mit dem Kantenbild zu einem Ergebnisbild kombiniert, welches optional durch *Image Based Warping* verbessert werden kann.

3.3.1 Abstraktion: Bilateraler Filter

Die für einen Cartoon typische reduzierte Farbpalette und große einfärbige Flächen entstehen durch das initiale Anwenden eines Bilateralen Filters auf die CIE L*a*b* Farbwerte der einzelnen Frames. Im Gegensatz zum Gauß Filter glättet dieser das gesamte Bild, ohne dabei Kanteninformation zu zerstören. Die Idee, die diesem Filter zu Grunde liegt, ist die Kombination zweier Filter, welche zusammen die Nähe zweier Pixel, gemessen durch räumliche und photometrische Differenz, zueinander berechnen können.

Der *Domain Filter* [84] misst die räumliche Nähe zwischen dem aktuellen Pixel $\vec{x}$ und seinen Nachbarpixeln ξ. Bei traditionellen Filtern wird das durch eine mit der Entfernung vom aktuellen Pixel sinkenden Gewichtung der einzelnen Nachbarpixel realisiert. Im Gauß'schen Fall erfolgt die Gewichtung entsprechend der symmetrischen Dichtefunktion. Dadurch ist der Domain Filter c folgendermaßen gegeben:

$$c(\xi, \vec{x}) = \exp(-\frac{1}{2}(\frac{d(\xi, \vec{x})}{\sigma_d})^2) \tag{3.20}$$

Dabei bezeichnet $d(\xi, \vec{x})$ die euklidische Distanz zwischen dem aktuellen Pixel $\vec{x}$ und seinen Nachbarpixeln ξ. Die Filtergröße des Domain Filters σ_d beeinflusst die Stärke der Glättung. Bei höheren Werten werden mehr Nachbarpixel in die Berechnung einbezogen.

Der *Range Filter* [84] misst die photometrische Nähe zwischen dem aktuellen Pixel $\vec{x}$ und seinen Nachbarpixeln ξ. Im Gauß'schen Fall wird seine Ähnlichkeitsfunktion s analog zu der des Domain Filters definiert:

$$s(\xi, \vec{x}) = \exp(-\frac{1}{2}(\frac{\delta(f(\xi), f(\vec{x}))}{\sigma_r})^2) \tag{3.21}$$

Der Ausdruck $\delta(f(\xi), f(\vec{x}))$ steht für die Ähnlichkeit zweier Farbwerte $f(.)$, wie beispielsweise ihre Differenz. Die Filtergröße des Range Filters σ_r beeinflusst die räumliche Ausbreitung der Farbmischung. Analog zum Domain Filter werden bei hohen Werten mehr Nachbarpixel hinsichtlich der Farbmischung berücksichtigt. Die räumliche Verteilung der Pixel wird vom Range Filter selbst vernachlässigt, sodass weit entfernte Pixel sowie direkte Nachbarn gleichberechtigt gemischt werden.

Die beiden Filter werden miteinander und mit der Farbe der Nachbarpixel multipliziert, sodass die Nachbarpixel sowohl räumlich als auch der Farbdifferenz entsprechend gewichtet werden. Diese Kombination kann für ein Pixel $\vec{x}$ nach [84] durch

$$h(\vec{x}) = k^{-1}(\vec{x}) * \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi) c(\xi, \vec{x}) s(f(\xi), f(\vec{x})) d\xi \tag{3.22}$$

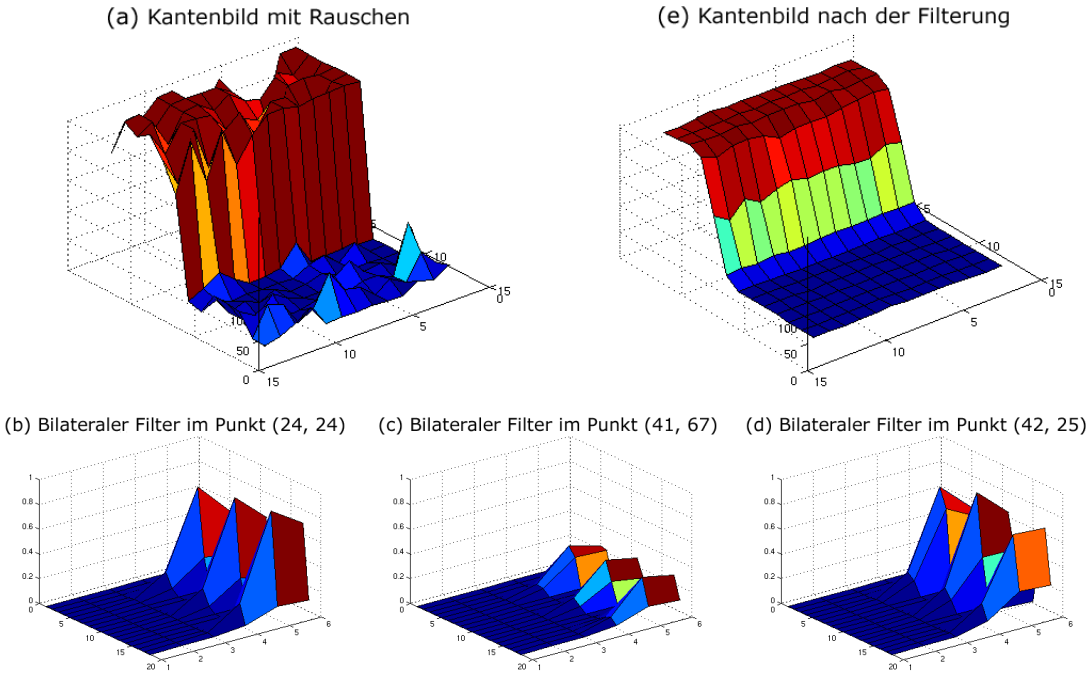

Abbildung 3.11: Diese Abbildung zeigt verschiedene mit MATLAB erzeugte *Surface Plots* der Bilateralen Filterung. *(a)* wurde aus einem Kantenbild mit zehn Prozent Gauß'schem Rauschen erzeugt. Das Gefälle der hellen und dunklen Pixel auf beiden Seiten der Kante ist zu sehen. *(e)* ist das Ergebnis der Anwendung des Bilateralen Filters mit $\sigma_r = 11.25$ (Radius von 23 Pixel) und $\sigma_d = 6$ (Radius von zwölf Pixel) auf das Kantenbild (a). In *(b), (c)* und *(d)* sind die kombinierten Ähnlichkeitswerte $c(\xi, \vec{x})s(f(\xi), f(\vec{x}))$ der Nachbarschaft des Bilateralen Filters an verschiedenen Positionen im Bild zu sehen. Das Zentrum der Pixel-Nachbarschaft ist in *(b)* ein helles, von Rauschen umgebenes Pixel, bei *(c)* ein helles Pixel in rauschfreierer Nachbarschaft und bei *(d)* ein dunkles Pixel unmittelbar neben der Kante.

beschrieben werden. Die Normalisierungsfunktion $k(\vec{x})$ ist dabei durch

$$k(\vec{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \vec{x}) s(f(\xi), f(\vec{x})) d\xi \tag{3.23}$$

gegeben. In der Implementierung werden die doppelten Integrale als Summe über die Nachbarschaft und die Eulersche Zahl realisiert.

In gleichmäßigen Regionen sind sich nahe Nachbarpixel sehr ähnlich und die normalisierte Ähnlichkeitsfunktion des Range Filters $k^{-1}s$ ist ungefähr Eins. Der Bilaterale Filter arbeitet in diesem Fall wie ein gewöhnlicher Gauß Filter und glättet den Bereich. Beinhaltet die Nachbarschaft eine scharfe Kante (siehe Abbildung 3.11) und das aktuelle Pixel liegt auf der hellen Seite der Kante, ergibt die Ähnlichkeitsfunktion s für Pixel auf der selben Seite Werte um Eins und auf der dunklen Seite Werte nahe Null. Die dunklen Pixel werden ignoriert und das Mittel der hellen Pixelfarben gewählt. Wäre das aktuelle Pixel auf der dunklen Seite der Kan-

te, würden umgekehrt die hellen Pixel vernachlässigt und die Kante deshalb nicht verwischt werden.

Anstelle der wiederholten Anwendungen dieses Filters wie beim initialen Abstraktionsalgorithmus [93] wird der Filter in dieser Adaption nur einmalig ausgeführt. Der Grund dafür sind die erhöhten Laufzeiten der Implementierung dieses Filters nach [84] (siehe Kapitel 5.4).

3.3.2 Kantenerkennung: DoG Operator

Die für einen Cartoon typischen Konturen erhält das Ergebnisvideo durch das Ermitteln von Kanten mittels Kantenerkennung. Analog zu traditionellen Gradientenoperatoren (wie z.B. der Sobel Filter [85]), deren Faltungskerne die erste Ableitung approximieren, kann auch die zweite Ableitung durch Differenzen angenähert werden. Dabei identifizieren letztere Nulldurchgänge als Kanten. Eine Möglichkeit der Realisation ist, Intensitätsveränderungen in verschiedenen Auflösungen eines Bilds heranzuziehen. Der DoG (*Difference of Gaussians*) Operator glättet das Bild mit den Pixeln (x, y) deshalb mit einem Paar verschieden großer Gauß Filter und berechnet deren Differenz (siehe Abbildung 3.12). [85]

$$DoG(\sigma_e, \sigma_r, x, y) = \frac{1}{2\sqrt{\pi\sigma_e}} \exp(-\frac{x^2+y^2}{2\sigma_e^2}) - \frac{1}{2\sqrt{\pi\sigma_r}} \exp(-\frac{x^2+y^2}{2\sigma_r^2}) \tag{3.24}$$

Die zwei Gauß Filter mit unterschiedlichen Standardabweichungen σ_r und σ_e, also unterschiedlicher Größe, stehen ungefähr im Verhältnis 1:1.6. Um das Verhältnis zumindest ungefähr einzuhalten, wird in der Implementierung nur einer der beiden Filter gewählt und der andere automatisch berechnet. Dieser mit der *Primal Sketch Theorie* [58] in Zusammenhang stehende Faktor verweist auf das rezeptive Feld der Netzhaut des menschlichen Auges, in dem viele Sinneszellen wenigen Ganglien des Nervensystems zugeordnet sind.

Damit die Kanten etwas weicher erscheinen und im Videoeffekt die zeitliche Kohärenz verbessert wird, wurde die Kantenerkennung von [93] um einen fixen Schärfeparameter ϕ_e erweitert. Dieser glättet die durch einen ebenfalls fixen Schwellenwert τ gebildete Stufenfunktion $D(.)$, sodass schwächere Kanten oder Kantenränder heller erscheinen als starke. Als Faktor der Differenz beeinflusst er, ob der Unterschied der beiden Gauß Filter $S_{\sigma e}$ und $S_{\sigma r}$, also die Stärke der Kante, mehr oder weniger ins Gewicht fällt.

$$D(\vec{x}, \sigma, \tau, \phi_e) = \begin{cases} 1, & \text{wenn } (S_{\sigma e} - \tau S_{\sigma r}) > 0 \\ 1 + \tanh(\phi_e(S_{\sigma e} - \tau S_{\sigma r})), & \text{sonst} \end{cases} \tag{3.25}$$

Die modifizierte DoG Funktion $D(.)$ aus [93] ergibt also einen von Eins unterschiedlichen Wert, wenn das aktuelle Pixel $\vec{x}$ als Kantenpixel identifiziert wurde. Für große gewichtete Differenzen strebt der Tangens Hyperbolicus gegen -1 und der gesamte Ausdruck gegen Null, sodass eine

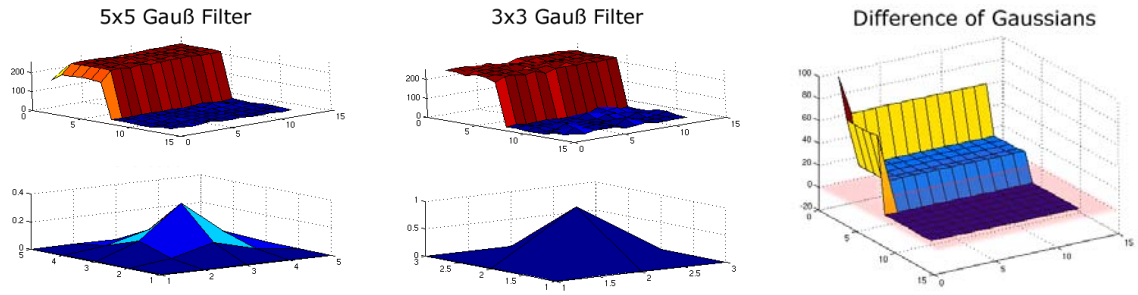

Abbildung 3.12: Diese Abbildung zeigt den mit MATLAB erzeugten *Surface Plot* der Anwendung eines DoG Operators mit $\sigma_e = 0.34$ (3x3) und $\sigma_r = 0.67$ (5x5). Die *vier linken Diagramme* zeigen (*oben*) zwei mit verschieden großen Gauß Filtern geglättete Kanten $S_{\sigma e}$ und $S_{\sigma r}$ sowie (*unten*) die dazugehörigen Filterkerne. Das *rechte Diagramm* zeigt die Differenz der beiden Bilder beim Schwellenwert $\tau = 0.91$, also $S_{\sigma e} - \tau S_{\sigma r}$. Die *rote Ebene* bzw. ihr Schnitt mit der Kurve zeigt den Nulldurchgang an.

spätere Multiplikation mit der initialen Luminanz einen schwarzen Pixel ergibt.

Für τ wird in der VEL Implementierung des Abstraktionsalgorithmus durchgehend 0.91 und im Original 0.98 verwendet. Generell gilt für $\tau \rightarrow 1$, dass der Filter an Stabilität verliert [93].

3.3.3 Farbreduktion: Reduktion der Helligkeitswerte

Der im *LABColorReduction Videoeffekt* obligatorische Schritt der Farbreduktion kann in [93] ausgelassen werden. Er ist der zweite Punkt, der die für einen Cartoon typische reduzierte Farbpalette und Kontraste hervorbringt.

Die Luminanz, die im Intervall [0, 100] liegt, wird bei der Quantifizierung zunächst in zwischen zwei und 32 Bins q der selben Größe Δ_q unterteilt. Das Bild wird Pixel für Pixel verarbeitet, wobei die Funktion $Q(.)$ mit dem Schärfeparameter ϕ_q und mit dem Bin $q_{nearest}$, das der Helligkeit $f(.)$ des jeweiligen Pixels $\vec{x}$ am ähnlichsten ist, den neuen Helligkeitswert bestimmt [93].

$$Q(\vec{x}, q, \phi_q) = q_{nearest} + \frac{\Delta_q}{2} \tanh(\phi_q(f(\vec{x}) - q_{nearest})) \tag{3.26}$$

Ähnlich dem Schärfeparameter in Abschnitt 3.3.2 gewichtet ϕ_q die Farbdifferenz, sodass mehr oder weniger kontrastreiche Bereiche entstehen. Zusammen mit dem Tangens Hyperbolicus glättet ϕ_q die Sprungstellen, und für BetrachterInnen entsteht bei einem großen ϕ_q eine kontinuierliche Funktion. Es besteht die Möglichkeit diesen Parameter fix zu wählen, sodass das gesamte Bild entweder eher starke Kontraste oder weiche Übergänge aufweist. In der Implementierung dieser Diplomarbeit ist der Schärfeparameter jedoch standardmäßig lokal durch ei-

ne Funktion der zuvor mittels Sobel Operator [85] ermittelten Gradienten G_x und G_y bestimmt.

$$\phi_q(\vec{x}, \tau, G_x, G_y) = \begin{cases} \frac{11\sqrt{G_x^2+G_y^2}}{\tau} + 3, & \text{wenn } \sqrt{G_x^2 + G_y^2} < \tau \\ 1, & \text{sonst} \end{cases} \tag{3.27}$$

Funktion 3.27 bewirkt dasselbe wie die von Winnemöller et al. [93] beschriebene Methode. Bei hohen Gradienten ($\phi_q = 1$) wird nach der Formel 3.26 die Differenz verstärkt und somit eine Luminanz nahe dem nächsten Binzentrum gewählt. Bei niedrigeren Gradienten ist mehr Spielraum geboten, so kann sich die neue Luminanz, entsprechend dem ersten Teil der Sprungfunktion 3.27, in dem, von [93] empfohlenen, Intervall [3, 14] vom nähesten Binzentrum wegbewegen. Diese Streuung der Helligkeitswerte resultiert in weicheren Übergängen.

3.3.4 Scharfzeichner: Image Based Warping

Als letzter Schliff kann die Kombination der Helligkeitsreduktion und Kantenerkennung mittels Image Based Warping (IBW) [4] im RGB Farbraum geschärft werden. Dieser Schritt bessert zusätzlich kleine Ungenauigkeiten bei der Positionierung der Kanten aus [93]. Der ursprünglich von Arad und Gotsman [4] entwickelte Scharfzeichner hat gegenüber einer *Unscharfmaskierung* [85] den Vorteil, dass er nicht zu dunklen oder hellen Linien und ähnlichen Nebenwirkungen führen kann. Beim IBW werden Pixel auf beiden Seiten einer unscharfen Kante zu dieser hinbewegt und der unscharfe Bereich somit verkleinert. Durch diese geringfügige geometrische Verformung wirkt das Bild, ohne neue Farben hinzuzufügen, insgesamt schärfer. Wie in [93] vorgeschlagen, verwendet der *LABColorReduction Effekt* eine einfachere IBW Implementierung von Loviscach [54], welche ursprünglich durch Photoshop Aktionen beschrieben wurde.

Das in vorhergehenden Schritten erzeugte Kantenbild wird mit einem Gauß Filter geglättet und außerdem mittels *Relief Filter* je ein Verschiebungsbild für horizontale h_x und vertikale h_y Verformungen erzeugt. Die Standardgröße der Filterkerne ist dabei 3x3:

$$h_x = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix} h_y = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{3.28}$$

Nach diesem Schritt sind im horizontalen bzw. vertikalen Verschiebungsbild $h(.)$ die Pixel neben einer Kante hell oder dunkel markiert (siehe Abbildung 3.13) und damit die Richtung, in die sie sich bewegen müssen, um sich ihrer jeweiligen Kante anzunähern, definiert.

Der für den *LabColorReduction Effekt* implementierte Filter skaliert, wie der GIMP [81] *Verschiebungsfilter*, die Reliefbilder auf die Größe des Anfangsbilds. Anschließend werden die

Abbildung 3.13: Das *erste Bild* zeigt *links* einen Ausschnitt aus dem Original. Das *zweite Bild* zeigt den entsprechenden Ausschnitt des mittels IBW geschärften Ergebnisses. Sowohl Verschiebungskoeffizient als auch Filterradien betragen jeweils drei Pixel. Die *beiden letzten Bilder* sind Zwischenergebnisse dieses Verfahrens. Das mit Gauß Filter geglättete Kantenbild (*drittes Bild*) wird horizontal und vertikal gefaltet, um ein Verschiebungsbild zu generieren. Das *letzte Bild* zeigt das horizontale Verschiebungsbild dieses Prozesses.

Grauwerte der beiden Bilder dazu verwendet, die X- bzw. Y- Verschiebung $D(.)$ in die vorgegebene Richtung zu berechnen:

$$D(\vec{x}, \phi_d) = \begin{cases} 0, & \text{wenn } h(\vec{x}) = 128 \\ n(h(\vec{x}))\phi_d, & \text{sonst} \end{cases} \tag{3.29}$$

Die normierte Intensität $n(h(\vec{x}))$ des Pixels $\vec{x}$ im Verschiebungsbild $h(.)$ ist dabei durch folgende Funktion gegeben:

$$n(h(\vec{x})) = \frac{h(\vec{x}) - 127.5}{127.5} \tag{3.30}$$

Die Differenz der Intensität, also des Grauwerts, eines Pixels im jeweiligen Verschiebungsbild wird vom mittleren Grauwert $(255/2 = 127.5)$ subtrahiert und durch die Division normiert. Das Ergebnis, die normierte Intensität, gibt somit die eigentliche Verschiebung (Richtung und initiale Stärke) an. Bei negativen $n(h(\vec{x}))$ werden Pixel in eine andere Richtung geschoben als bei positiven. Der Verschiebungskoeffizient ϕ_d wird in Funktion 3.29 mit der normalisierten Intensität multipliziert und verstärkt oder schwächt damit den bereits berechneten Wert. Zusammen mit der normierten Intensität gibt der Koeffizient an, wie stark ein unscharfer Bereich verkleinert werden soll. Bei großen ϕ_d wird das Pixel also weiter weg geschoben als bei kleineren.

3.4 LabColorReduction Videoeffekt

Der in der VEL als *LabColorReduction Videoeffekt* bezeichnete Videoeffekt ist eine auf der CPU laufende Adaption des bereits in Kapitel 2.2.5 skizzierten Videoeffekts. Wie beim Original handelt es sich auch hier lediglich um eine Frame-zu-Frame Anwendung des Bildeffekts. Winnemöller et al. weisen in ihrem Paper [93] darauf hin, dass aufgrund der Eigenschaften

der Abstraktion im Bildeffekt keine weiteren Transformationsschritte notwendig sind, um ein zeitlich kohärentes Ergebnis zu erzielen.

Kapitel 4

Experimentelle Ergebnisse

Dieses Kapitel hat eine umfassende Betrachtung der durch die, im Zuge dieser Diplomarbeit entwickelten, Effekte erzielten visuellen Ergebnisse zum Inhalt. Es bietet einen Rahmen für experimentelle Variationen von Parametern und mögliche weiterführende Modifikationen von Effekten.

4.1 Effektvariationen

In diesem Abschnitt werden zunächst zwei Effektvariationen vorgestellt. Die erste (Abschnitt 4.1.1) versucht, Optical Flow Daten in den Segmentierungsprozess einzubinden. Die zweite Variation (Abschnitt 4.1.2) geht der Frage nach, ob die Farbreduktion im HSB Raum visuell ansprechendere Ergebnisse als jene im RGB Raum erzeugt.

4.1.1 Einbinden von Optical Flow Daten

In Kapitel 2 wurde die Erkenntnis gewonnen, dass der Einsatz von Optical Flow bei der Transformation von Bild- zu Videoeffekt, trotz seiner Ungenauigkeit, hilfreich sein kann. Dieser Aspekt soll auch in einer experimentelleren Variation des *MultiCluster Videoeffekts* (siehe Kapitel 3.2.2) zur Anwendung kommen. Diese Variation bezieht ein zweites Frame nicht mehr durch die Erhöhung der Datenmenge (Rot, Grün, Blau - Datenpunkte), sondern durch das Hinzufügen zwei weiterer Dimensionen ($FlowX$, $FlowY$) oder einer weiteren Dimension, der euklidischen Norm des horizontalen und vertikalen Vektorfelds [23], ein. Der *FlowCluster Videoeffekt* gibt dazu die Freiheit, zwischen den Methoden der Farbreduktion wählen zu können, für eine erweiterte Definition von Regionen, im Bezug auf ihre Bewegungen von Frame zu Frame, auf. Unter der Annahme, dass sich die zu einem Objekt gehörenden Pixel in die selbe Richtung bewegen, wird die Bewegungsinformation als ein weiteres Homogenitätskriterium des k-Means

Clusterings hinzugefügt [23].

Die Optical Flow Daten werden dabei mit einem einfachen *Block Matching* Algorithmus [55] ermittelt. Die Bewegungsvektoren der Pixel eines Blocks b_1 zeigen auf einen gleich großen Block b_2 innerhalb des Suchfensters W, welcher den mittleren absoluten Luminanzunterschied (MAD) minimiert. Also wird für einen Block b_1 der Größe NxM mit dem Mittelpunkt (x, y) die Differenz (MAD) zu allen anderen Blöcken $(x + d_x,\ y + d_y)$ innerhalb des Suchfensters berechnet. Der Block mit dem geringsten Unterschied wird gewählt, um den Bewegungsvektor für alle Pixel von b_1 abzuleiten. Der Ausdruck $L(x, y)$ verweist dabei in folgender Funktion auf die Luminanz des Pixels an der Position (x, y).

$$\begin{aligned} MAD(b_1, b_2) &= \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} |L(x+i, y+j) - L(x+d_x+i, y+d_y+j)| \\ &= W(x, y)|L(x+i, y+j) - L(x+d_x+i, y+d_y+j)| \end{aligned} \tag{4.1}$$

Diese Schätzung ist zwar schnell und einfach zu implementieren, aber in vielen Fällen auch ungenau (siehe Abbildung 4.1). Zur Unschärfe, die sich durch die Verwendung der MAD ergibt, kommt die Approximation, die durch die Blockbildung entsteht, hinzu. Diese Artefakte sind auch im Ergebnis sichtbar (siehe Abbildung 4.2, *rechts*).

Abbildung 4.1: Diese Abbildung zeigt zwei aufeinander folgende Frames und das aus ihnen berechnete Optical Flow Vektorfeld. Die weißen Flächen im Vektorfeld (*rechtes Bild*) zeigen bewegungslose Bereiche an. Stark texturierte Regionen (z.B. Fenster) liefern bessere Ergebnisse als große einfärbige Flächen (z.B. Wand oder Boden).

Die Startzentren werden, wie gehabt, mit den im Bild am häufigsten vorkommenden Farben initialisiert (siehe Kapitel 3.1.1). Außerdem werden den Farben zusätzlich pseudo-zufällige Startvektoren zugewiesen. Bei gleichbleibender Anzahl der Farben ist die Farbe-Vektor Kombination identisch.

Abbildung 4.2 zeigt Ausschnitte einer verarbeiteten Videosequenz bei variierenden Parametern. Bei zunehmendem Einfluss der Optical Flow Daten steigt auch das Flimmern im

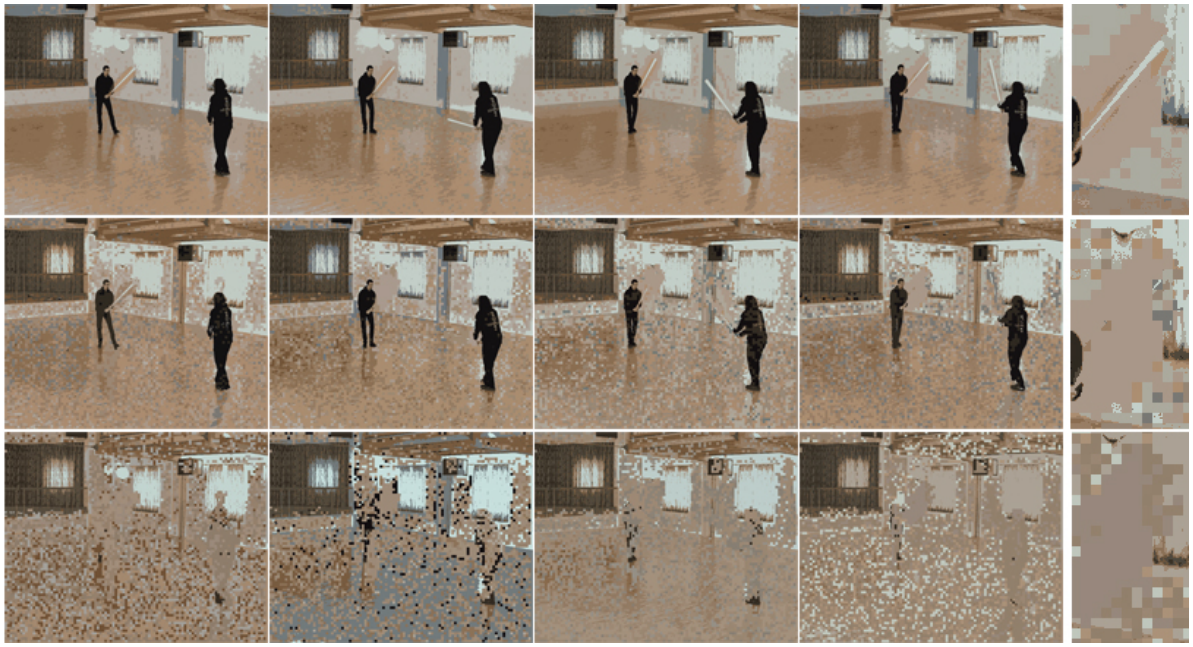

Abbildung 4.2: Diese Abbildung zeigt drei verschiedene Ergebnisse der Verarbeitung einer Bildsequenz mit dem *FlowCluster Videoeffekt*. Mit jeder Zeile wurden dabei die Dimensionen verschieden gewichtet. Die Anzahl der Artefakte in der *ersten Reihe* ist am geringsten, die Gewichtung Farb- zu Bewegungsdaten steht dabei im Verhältnis 1000:1. Die Gewichtung der Optical Flow Daten der zweiten Reihe entspricht einem Hundertstel der Farbdaten. In der *letzten Reihe* ist die Gewichtung der Farbkanäle doppelt so hoch wie jene der Bewegungsdaten. Das *letzte Bild jeder Zeile* zeigt jeweils einen Bildausschnitt, der weniger stark verkleinert ist als die übrigen Frames. Dadurch kann die visuelle Qualität der Ergebnisse besser evaluiert werden.

Ergebnisvideo. Die Größe der Artefakte entspricht dabei jener der Blöcke beim Block Matching. Im Vergleich zu den übrigen im Zuge dieser Diplomarbeit produzierten Videoeffekten kann durch das Einbinden der Bewegungsinformation, zumindest bei dieser experimentellen Vorgangsweise, keine Verbesserung erzielt werden.

4.1.2 Verarbeitung im HSB Farbraum

Eine weitere experimentelle Variante des *ColorReduction Bildeffekts* versucht, angelehnt an Effekte wie [93][11][77], die Verarbeitung nicht mehr im bisher verwendeten RGB Farbraum, sondern in einem der menschlichen Wahrnehmung etwas besser angepassten Farbraum durchzuführen. Das Ziel der praktischen Umsetzung ist, einer möglichen Verbesserung der Segmentierung und Darstellung sowohl im Bild- als auch im Videoeffekt nachzugehen.

Das Histogramm Binning aus Kapitel 3.1.1, welches ursprünglich mit einem Histogramm der RGB Daten arbeitet, verwendet hier ein HSB Histogramm. Die darin am häufigsten vorkommenden Farben werden analog durch Zählen der Farb-, Sättigungs- und Helligkeitswerte

ermittelt.

Das k-Means Clustering kann ebenfalls im HSB Farbraum durchgeführt werden. Darauf basierend wurden zwei weitere Farbreduktionen abgeleitet. Eine davon wählt die Startzentren aufgrund des HSB Histogramms, während die Zweite von RGB Startzentren ausgeht.

Abbildung 4.3: Diese Abbildung zeigt ein Bild nach der Anwendung des *ColorReduction Bildeffekts* mit verschiedenen Farbreduktionen. In der *ersten Reihe* wurden dabei unterschiedliche HSB-Varianten gewählt: *HSB Histogramm Binning* (*links*), *HSB Clustering* mit HSB Startzentren (*Mitte*) und *HSB Clustering* mit RGB Startzentren (*rechts*). In der *zweiten Reihe* befinden sich zum Vergleich die RGB Farbreduktionen: Histogramm Binning (*links*), Clustering (*Mitte*), Simple Farbreduktion (*rechts*). Bis auf die Art der Farbreduktion wurden die gleichen Parameter gewählt, so enthalten alle Effektbilder zehn Farben und leicht transparente (0.7) Kanten. Die HSB Entsprechungen machen zwar einen flacheren, cartoonartigeren Eindruck, produzieren aber schon im Bildeffekt sichtbare Artefakte.

Ein Ergebnis dieser alternativen Verarbeitung der Farbdaten kann in Abbildung 4.3 betrachtet werden. In diesem Beispiel weisen die HSB Farbreduktionen schon im Bildeffekt deutliche Artefakte auf, was auch auf eine gewisse Fehleranfälligkeit im Videoeffekt hinweist. Von diesem Problem abgesehen, werden bei den HSB Entsprechungen im Allgemeinen kräftigere Farben gewählt und das Bild anders abstrahiert. Oft bleiben, wie auch in Abbildung 4.3, bei gleich vielen Farben weniger Details erhalten, wodurch das Bild flacher wirkt und cartoonartigere Ergebnisse erzielt werden. Diese Tatsache macht zumindest die beiden HSB Clustering-Methoden für den Bildeffekt interessant. Ein Problem, das davor noch überwunden werden müsste, ist, dass einige Cluster leer bleiben. Dadurch kann es bei höherer Farbanzahl im Extremfall auch zu einfärbigen Effektbildern kommen.

4.2 Parametervariationen

Dieser Abschnitt zeigt beispielhaft Ergebnisse, die durch Variation der Effektparameter erzielt werden können, und ermöglicht ihre visuelle Evaluierung. Für die Parameter des *LabColorReduction Bildeffekts* (siehe Abschnitt 4.2.2) wird dabei nur kurz auf die verpflichteten Schritte eingegangen, da es sich um die Adaption eines bereits existierenden Effekts [93] handelt.

4.2.1 ColorReduction Effekte

Der Einfluss, den die Wahl der Farbreduktion auf das Ergebnis hat, wurde bereits in Abbildung 4.3 visualisiert. Dennoch soll an dieser Stelle ein weiteres, ausführlicheres Beispiel gebracht werden (siehe Abbildung 4.4).

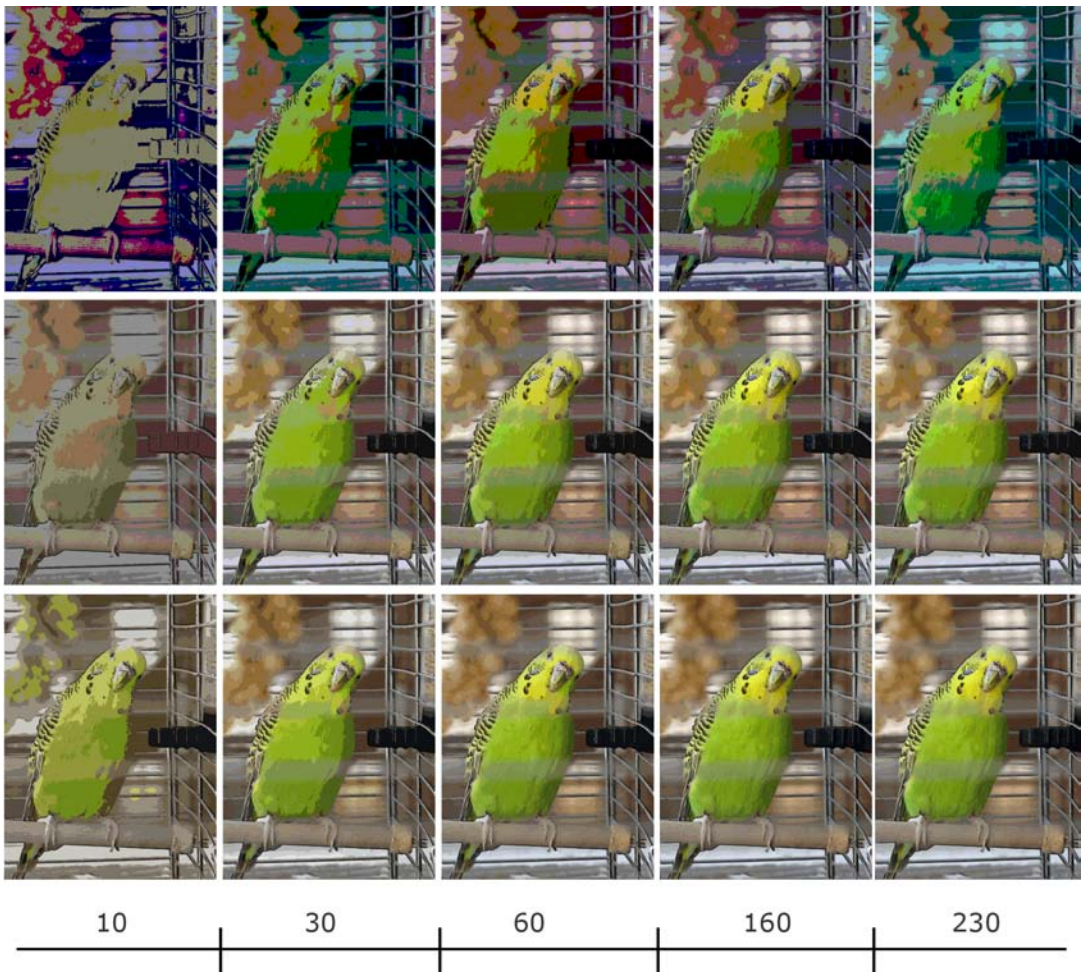

Abbildung 4.4: Die Abbildung zeigt eine Abstraktion mit variierender Farbanzahl bei sonst gleichbleibenden Einstellungen. Die *erste Reihe* stellt dabei die Ergebnisse der Simplen Farbreduktion dar und die *zweite* die des Histogramm Binnings. Die *dritte Reihe* ist die Folge von Clustering. Für die Abstraktion wurden jeweils zehn, 30, 60, 160 und 230 Farben gewählt.

Bei Erhöhung der Farbanzahl wird durchwegs eine Annäherung an das Originalbild erzielt. Die einzige Ausnahme ist die Simple Farbreduktion, welche besonders bei Primzahlen (siehe Abbildung 4.5), aber auch im Allgemeinen zu abweichenden Ergebnissen führt. In Abbildung 4.4 macht sich das durch eine vergleichsweise starke Änderung der Farben von zehn zu 30 oder 160 zu 230 bemerkbar. Die Farbwahl erscheint beim Clustering am optimalsten, wobei mit dem Histogramm Binning bei steigender Farbanzahl ähnlich gute Ergebnisse zu erreichen sind (Abbildung 4.4, ab 60 Farben). Bei Simpler Farbreduktion und Clustering können trotz unterschiedlicher Farbanzahl sich gleichende Abstraktionen entstehen, da aufgrund der Farbwahl nicht alle Farben benötigt werden (Abbildung 4.4, bei 30 wurden nur 19 und bei 230 nur 79 Farben verwendet) oder während der Verarbeitung einige Cluster leer bleiben (Abbildung 4.4, bei 160 gingen 40 Cluster und bei 230 ganze 121 Cluster leer aus).

Abbildung 4.5: Die Simple Farbreduktion liefert bei Farbanzahlen, die Primzahlen sind, wie beispielsweise bei elf (*erstes Bild*), 19 (*mittleres Bild*) und 21 (*rechtes Bild*), von ihren Nachbarwerten stark abweichende Ergebnisse. Aufgrund der Verteilung auf die verschiedenen Farbbänder (siehe Kapitel 3.1.1) wird beispielsweise bei der Wahl von elf Farben, Grün durch elf Bins und Rot sowie Blau jeweils nur durch ein Bin repräsentiert. Die Kombination dieser ungleichen Verteilung und der dürftigen Ausschöpfung der zur Wahl stehenden Farben kann zu minderwertigen Ergebnissen führen.

Nach der Abstraktion des Bilds können weitere Modifikationen angewandt und so beispielsweise ungesättigte Ergebnisse aus der Farbreduktion verbessert werden. Zusätzlich zur Sättigung kann, nach HSB Farbmodell, auch die Helligkeit und der Farbton verändert werden. Beispiele dazu und für Farbabbildungen, die bis jetzt nur theoretisch behandelt wurden (siehe Kapitel 3.1.2), können in Abbildung 4.6 betrachtet werden. Die Effektbilder der Farbabbildung liefern für gewöhnlich visuell ansprechende Ergebnisse, welche unter anderem an den im Film *Sin City* [80] verwendeten Stil erinnern.

Die Häufigkeit, die Deckkraft und der Berechnungszeitpunkt der Kanten können ebenfalls durch die Modifikation zweier Parameter im User Interface variiert werden. Wegen des Zusammenspiels von Kantendetektor, automatischer Schwellenwertextraktion und geglätteter Sprungfunktion (siehe Kapitel 3.1.3) resultiert die Wahl eines höheren Häufigkeitwerts, und damit einer geringfügigen Erhöhung des extrahierten Schwellenwerts, in breiteren, kräftigeren

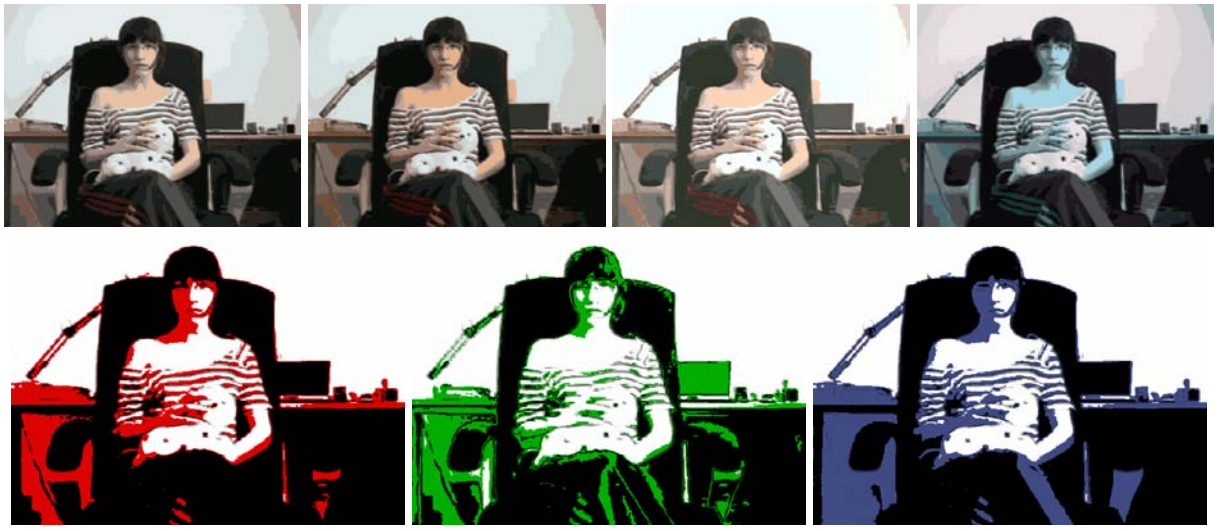

Abbildung 4.6: Diese Abbildung summiert die verschiedenen Farbmodifikationen. In der *ersten Reihe* folgt nach einem abstrahierten Bild mit Standardwerten eines mit etwas erhöhter Sättigung (*zweites Bild*). Veränderte Helligkeit (*drittes Bild*) und verschobener Farbton (*viertes Bild*) sind weitere Modifikationen. Die *zweite Reihe* zeigt Ergebnisse, die durch eine Farbabbildung, wie in Kapitel 3.1.2 beschrieben, erreicht werden können.

und zahlreicheren Kanten (siehe Abbildung 4.7). Niedrigere Häufigkeiten erlauben im Gegensatz dazu weniger Kantenpixel zu extrahieren, wodurch nur stärkere Kanten für die Stilisierung verwendet werden. Die entsprechenden Werte für den Schwellenwert und die Anzahl der Kantenpixel des Beispielbilds sind in Tabelle 4.1 zu finden. Eine Verminderung der Deckkraft be-

Tabelle 4.1: Diese Tabelle zeigt das Zusammenspiel von Filtergröße und automatischer Schwellenwertextraktion. Diese Werte beziehen sich auf das Testbild in Abbildung 4.4 mit einer Größe von 366 x 492 Pixel. Die Anzahl der Kantenpixel entspricht nicht exakt der zugehörigen Prozentzahl, da immer alle Pixel eines Bins übernommen werden (siehe Kapitel 3.1.3).

Häufigkeit	6%	9%	15%
Anzahl Kantenpixel	10804	16206	27010
Grauwert Schwellenwert	133	95	57

wirkt eine langsame Verschmelzung der Konturen mit dem Hintergrund, wobei durch das vorherige Glätten mit der Sprungfunktion schwächere Kanten zuerst verschwinden (siehe Abbildung 4.7, *rechte Bilder*). Ein weiterer kantenbezogener Parameter ist der Zeitpunkt der Berechnung der Kanten, so kann einerseits das Originalbild herangezogen werden, um auch Details hervorzuheben, oder andererseits das Effektbild als Quelle dienen, was sich im Ergebnis durch Linien zwischen Segmenten auswirken kann (siehe Abbildung 4.7).

Drei weitere Einstellungsmöglichkeiten dienen dem Hinzufügen (Effekttransparenz, Kon-

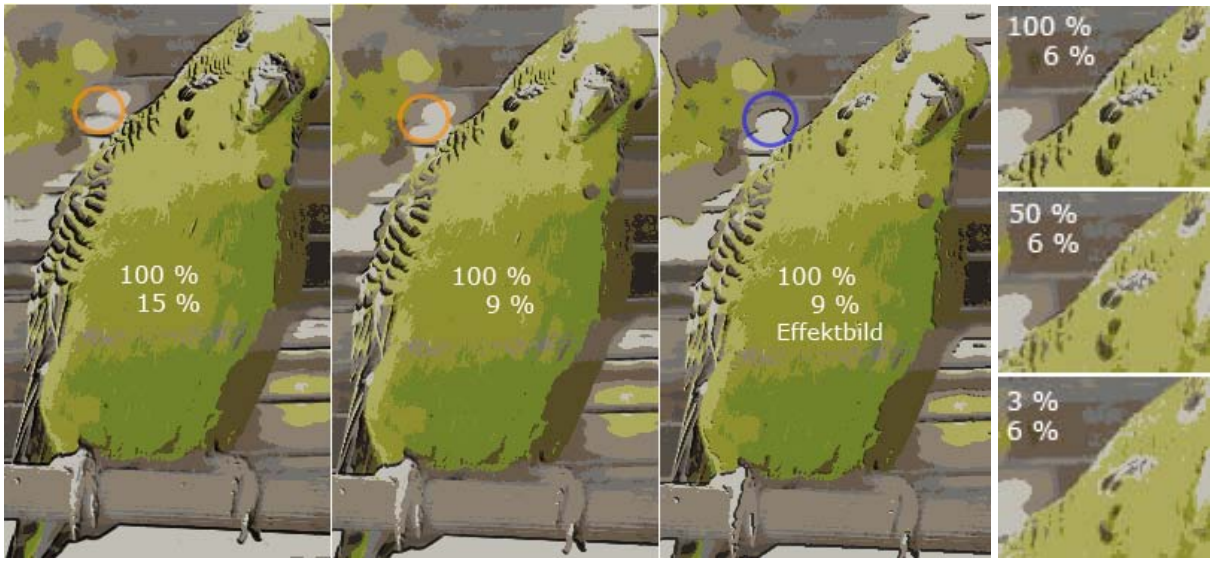

Abbildung 4.7: Diese Abbildung zeigt das Verhalten der Konturen bei verschiedenen Einstellungen. Der erste Prozentwert steht dabei jeweils für die Deckkraft und der zweite für die Häufigkeit der Kanten. Die *ersten beiden Bilder* zeigen den Effekt, den die Modifikation des Häufigkeitsparameters zur Folge hat. Größere Werte bewirken dunklere, breitere und mehr Kanten (siehe z.B. *orange Markierung* für eine verschwindende Kante). Das *dritte Bild* verdeutlicht die Berechnung der Konturen im Effektbild. Die Anzahl an Linien, die Details, wie zum Beispiel in diesem Fall die Musterung des Gefieders, hervorheben, verringert sich und die Anzahl an Linien zwischen Segmenten (siehe z.B. *blaue Markierung*) steigt. Bei den *vertikal angeordneten Bildern* wird bei gleichbleibender Häufigkeit (6%) die Transparenz (100%, 50%, 3%) verringert, wodurch die Konturen nach und nach, beginnend bei den Schwächeren, mit dem Hintergrund verschmelzen.

traststärkung) und Entfernen (Glätten) von Details. Bei einigen der Testbilder konnten durch Reduktion auf wenige Farben ($c \in [3, 5]$) und die transparente Anwendung des Bildeffekts ($t \in$ [20%, 40%]) gute Ergebnisse hinsichtlich der cartoonartigen Abstraktion erzielt werden (siehe Abbildung 4.8). Auch das Hinzufügen von Details in Form von Kontrasten kann das Ergebnis vorhergegangener Abstraktionsschritte dahingehend verbessern (siehe Abbildung 4.9). Die Farbvariation wird dabei minimal, für das menschliche Auge kaum sichtbar, erhöht. Zur Stärkung von Kontrasten und Streuung von Helligkeitswerten im Effektbild werden BenutzerInnen zwei Parameter geboten (siehe Kapitel 3.1.4). Die Größe des Streuungsintervalls hat direkten Einfluss auf die Kontraststärke und die maximale Größe von Farbverläufen. Der Schwellenwert steuert wo und wie weit sich diese Verläufe einer Kante nähern dürfen. Während niedrige Werte Verläufe schon bei kleinen Änderungen abschneiden und damit bereits an weicheren Kanten Kontraste stärken, erlauben hohe Werte großflächigere Verläufe. Geringe Frequenzänderungen spiegeln sich dabei durch leichte Helligkeitsänderungen wieder. Erst starke Kanten werden zusätzlich betont.

Abbildung 4.8: Das *linke Bild* zeigt ein Testbild nach der Anwendung des *ColorReduction Bildeffekts* mit nur drei Farben, um ein simples Ergebnis mit besonders großen einfärbigen Flächen zu produzieren. In der *Mitte* wurde zusätzlich die Effekttransparenz auf 30% erhöht, wodurch ein gewisser Anteil an Details und Farbe ergänzt wird. Das *rechte Bild* ist hingegen ein Beispiel für die Verminderung von Details. Hier wurde zusätzlich eine Glättung durchgeführt.

Abbildung 4.9: Das *linke Bild* zeigt ein Testbild nach der Anwendung des *ColorReduction Bildeffekts* mit acht Farben. Im *mittleren Bild* wurden bis zu einem Gradienten von 80 die HSB-Helligkeitswerte in einem Intervall von [0,10] gestreut. Das *rechte Bild* erlaubt eine Helligkeitsänderung von bis zu 20 Stufen, sodass mehr Details und Kontraste sichtbar werden. Die *vertikal angeordneten Bildausschnitte* zeigen jeweils vergrößerte Teile der drei vorhergegangenen Bilder, um den Effekt der Helligkeitsstreuung in hoch- und niedrigfrequenten Bereichen zu verdeutlichen.

4.2.2 LabColorReduction Effekte

Während der Videoeffekt von [93] in seiner ursprünglichen Variante eine mehrfache Anwendung des Bilateralen Filters vorsieht, ist seine Adaption im Zuge dieser Diplomarbeit (siehe Kapitel 2.2.5) auf eine einmalige Anwendung beschränkt. Diese Entscheidung wurde vor allem aufgrund der langen Verarbeitungszeiten des Filters (siehe Kapitel 5.4) getroffen. Abbildung 4.10 visualisiert den Unterschied, den eine weitere Anwendung des Filters mit sich bringen würde, und stellt die Auswirkungen seiner Eigenschaften auf das Ergebnis dar.

Neben dem Bilateralen Filter spielen auch die Anzahl der gewählten Lightness Bins (siehe Abbildung 4.11) und der Schwellenwert für die Steuerung von Verläufen (siehe Abbildung 4.12) eine entscheidende Rolle. Erstere beeinflusst den Grad der Abstraktion, die mit steigen-

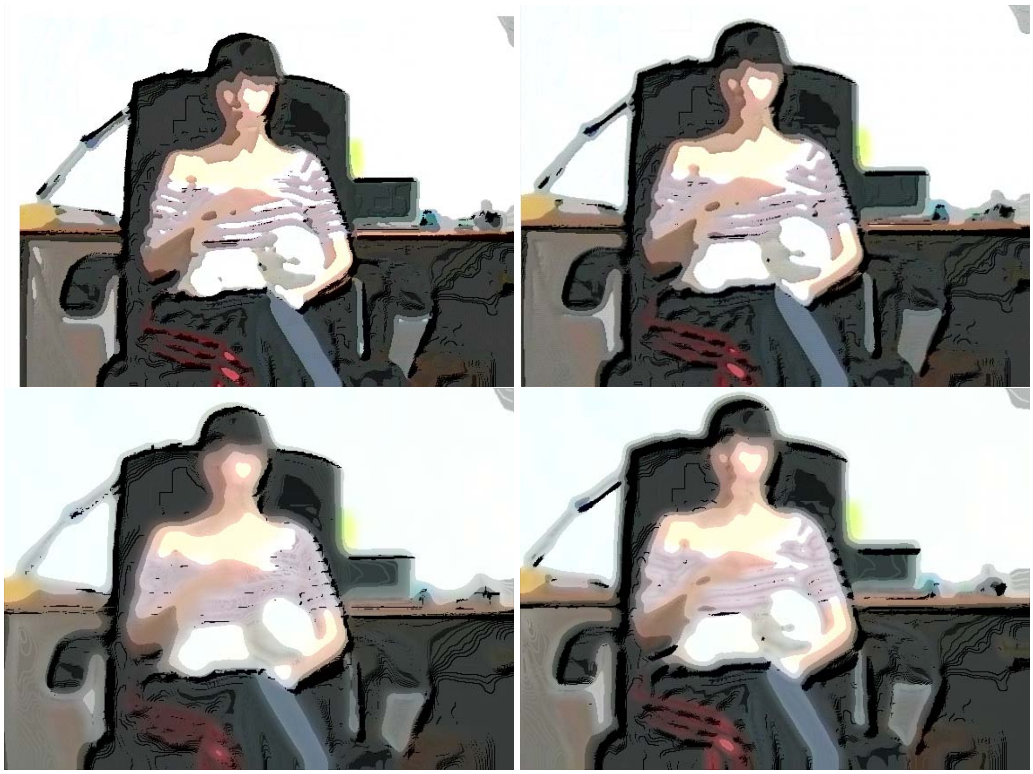

Abbildung 4.10: *Bilateraler Filter*: Domain Filter = 3.0, Range Filter = 4.25. *Farbreduktion:* Helligkeit wird auf vier Bins reduziert, *DoG-Kanten:* Die Filtergröße ist fünf bzw. sieben und der Schärfeparameter 3.75. Das *erste Bild* zeigt ein Beispiel, das mit diesen Standardeinstellungen des Effekts erstellt wurde. Beim *zweiten Bild der ersten Reihe* wurde vor der Farbreduktion der Bilaterale Filter ein zweites Mal angewandt, wie es bei der Abstraktion von [93] eigentlich vorgesehen ist. Die Bilder der zweiten Reihe zeigen die Änderung, die eine Verdoppelung der Größe des Domain- (*linkes Bild*) bzw. Range Filters (*rechtes Bild*) bei gleichbleibenden übrigen Parametern bewirkt.

der Anzahl der Bins dem Originalbild ähnlicher wird. Der Schwellenwert für die automatische Wahl des Schärfeparameters gibt dem, bereits zu diesem Zeitpunkt nahe am Cartoon stilisierten, Ergebnis den letzten Schliff. Bei den gewählten Testsequenzen wurden die besten Resultate bei acht Lightness Bins und niedrigen Schwellenwerten erzielt.

In Abbildung 4.13 wird das Verhalten eines weiteren Parameters, nämlich des Schärfeparameters der Konturen, dargestellt. Dieser bewirkt, dass Kanten zwischen farblich sehr unterschiedlichen Flächen (z.B. schwarz und weiß) stärker sind als Kanten zwischen Flächen mit ähnlichen Farben (z.B. rot und rosa).

Verglichen mit den Resultaten der *ColorReduction Effekte* kann durch diese Abfolge an Abstraktionsschritten ein visuell sehr ansprechendes Ergebnis erzielt werden. Die verschiedenen Stile, welche die jeweiligen Effekte hervorbringen, unterscheiden sich vor allem im Umgang mit Verläufen. Diese Verläufe sind ein wichtiger Bestandteil des *LABColorReduction Bild-*

Abbildung 4.11: Diese Abbildung zeigt drei Ergebnisse der Anwendung des *LabColor-Reduction Bildeffekts*. Die dabei aktiven Parameter unterscheiden sich nur in der Wahl der Lightness Bins und entsprechen sonst den Standardeinstellungen. Beim *linken Bild* wurden zwei, beim *mittleren Bild* vier und beim *rechten Bild* 32 Bins verarbeitet.

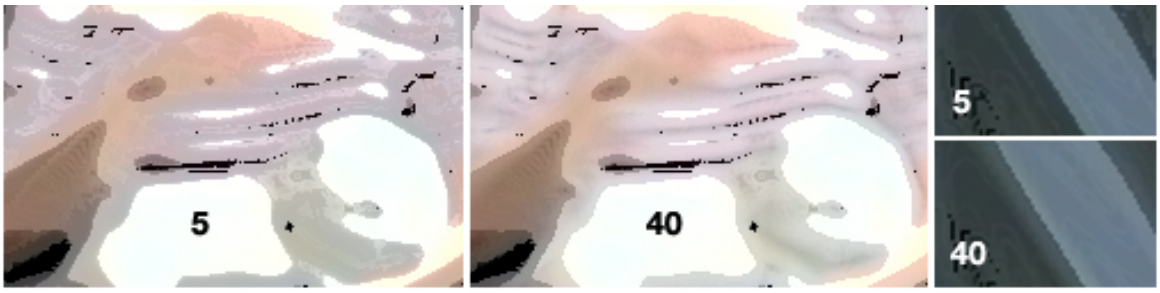

Abbildung 4.12: Der Schwellenwert für die automatische Wahl des Schärfeparameters, der zur Identifikation von hohen bzw. niedrigen Gradienten dient, beeinflusst die Streuung der Helligkeitswerte um ihre Bins. Die Bildausschnitte dieser Abbildung zeigen die Auswirkungen durch Änderungen des Werts bei fünf und 40. Bei höheren Schwellenwerten sind die Ergebnisse im Allgemeinen heller, was auf die größere Anzahl langsamer Farbverläufe zurückzuführen ist (*linke Bilder*). Niedrigere Schwellenwerte sind durch stärkere Übergänge geprägt, was sich in härteren Kanten äußert (*rechte Bilder*).

effekt, während sich der *ColorReduction Bildeffekt* mehr auf Kontraste konzentriert. Ein Vorteil, der sich aus der von der Cartoon Abstraktion vorgegebenen Farbreduktion ergibt, ist die Geschmeidigkeit der Objektränder. Der *ColorReduction Bildeffekt* hingegen kann bei langsamen Übergängen „körnige“ Objektgrenzen aufweisen.

4.3 Vergleichende Ergebnisse

Dieser Abschnitt stellt die implementierten Videoeffekte und die darin umgesetzten unterschiedlichen Ansätze für die Erhaltung von zeitlicher Kohärenz in abstrahierten Videosequenzen gegenüber. Ihre Effektivität wird dabei durch die Anzahl und Höhe von Farbsprüngen, sowie an der Änderung einzelner Pixel gemessen.

Abbildung 4.13: Die beiden Bildausschnitte zeigen die Auswirkung des Schärfeparameters der DoG-Kantenerkennung. Beim *linken Bild* ist dieser Wert 0.75, beim *rechten Bild* 4.75. Besonders das *linke Bild* macht den Helligkeitsunterschied zwischen Kanten mit hohen und niedrigeren Gradienten sichtbar. Die schwarzen Linien sind zwischen kontrastreichen Flächen dunkler und dicker als zwischen Flächen mit ähnlichen Farben. Durch Erhöhung des Schärfeparameters (*rechtes Bild*) sind auch diese vermeintlich schwachen Kanten vergleichsweise stark ausgeprägt.

4.3.1 Farbsprünge allgemein

Das Beibehalten der zeitlichen Kohärenz im Zuge der Stilisierung von Videosequenzen wurde aufgrund der Ergebnisse der Frame-zu-Frame Anwendung des *ColorReduction Bildeffekts* vor allem durch Strategien zur Minimierung von Farbsprüngen und Stabilisierung der Farbwahl allgemein, ohne Einfluss räumlicher Komponenten, adressiert. In diesem Zusammenhang sind sowohl die Anzahl, als auch die Höhe der Farbsprünge als immanente Kriterien zur Evaluierung der zeitlichen Kohärenz zu sehen. In diesem Abschnitt sollen unter Verwendung dieser beiden Referenzwerte die implementierten Videoeffekte verglichen werden. Um die dabei getroffenen Aussagen im Bezug auf erzielte Veränderung allgemeiner Farbsprünge zu unterstützen, wurden exemplarisch die Ergebnisse einer der verwendeten Testsequenzen (*car_crash2_lower*, siehe Abbildung 4.14) ausgewertet und visualisiert (siehe Abbildung 4.15 und 4.16 sowie Tabelle 4.2). In dieser Sequenz betritt ein neues Objekt eine simple Szene, was für die implementierten Videoeffektalgorithmen eine Herausforderung darstellt.

Schon bei der Frame-zu-Frame Anwendung des Bildeffekts werden die Unterschiede angesichts der Wahl der Farbreduktion, welche bereits den ersten Schritt Richtung Stabilisierung der Farbwerte darstellt, deutlich. Während das Histogramm Binning im Allgemeinen wenig Farbänderungen aufweist, ist dieser Nebeneffekt bei den anderen beiden Varianten öfter zu beobachten (siehe Abbildung 4.15). Diese Tatsache ist auf die Beschaffenheit der verschiedenen Ansätze zurückzuführen. Die Simple Farbreduktion basiert auf einer beliebigen Teilung der

Abbildung 4.14: Diese Abbildung zeigt vier markante Frames der Testsequenz *car_crash2_lower*. Knapp zusammengefasst betritt in ihr ein Objekt eine simple Szene, wodurch sich die Farbzusammensetzung ändert und die Stabilisierung der Farbwahl erschwert wird.

Farbkanäle und das Clustering auf einer Veränderung der Farbwerte bezüglich des Bildinhalts, wodurch Unsicherheiten bei der Frame-zu-Frame Konsistenz hinzukommen. Betrachtet man hingegen die durchschnittliche Höhe der Farbsprünge (siehe Abbildung 4.16), fallen beim Histogramm Binning Ausreißer stärker ins Gewicht und können deshalb größere Farbdifferenzen liefern. Verändern sich die k am häufigsten im Bild vorkommenden Farben, wie beispielsweise bei der Testsequenz *car_crash2_lower*, durch das Eintreten eines neuen Objekts in die Szene, kann ein Farbsprung hervorgerufen werden. Sieht man von diesen Einzelfällen ab, werden auch mit diesem Kriterium die guten Ergebnisse der Farbkonstanz des Histogramm Binnings bei einer Frame-zu-Frame Anwendung unterstrichen. Die Cluster Farbreduktion liefert ebenfalls relativ gute Ergebnisse. Im Allgemeinen wird sie mit mehr Iterationen etwas präziser, so brachte die als Beispiel herangezogene Testsequenz bei zwei Iterationen 36 und bei drei 33 Änderungen hervor. Da die Anzahl der Farbsprünge (mit dem Maximalwert von 33) sowie ihre durchschnittliche Höhe (mit dem Maximalwert bei 236) bei einer kleineren Farbanzahl (zwischen zwei und fünf) noch verhältnismäßig hoch ist, versuchen die implementierten Videoeffekte sie zu minimieren und im Zuge dessen auch an kritischen Stellen, wie Zeitpunkten, an welchen sich der Farbparameter, die Helligkeit oder die Szene selbst ändert, die Farbkonstanz zu erhöhen.

Der an globaler Verarbeitung von Bilddaten orientierte *MultiCluster Videoeffekt* verhält sich in vielen Fällen wie die Frame-zu-Frame Anwendung, kann aber leichte Verbesserungen bezüglich Anzahl und durchschnittlicher Höhe der Farbsprünge erzielen. Im Vergleich zum *ColorReduction Videoeffekt* ist er hinsichtlich Variationen der Farbanzahl stabiler. Beim Spezialfall eines in eine Szene eintretenden Objekts wird ein lokales Maximum, das bei den anderen beiden Videoeffekten und der Frame-zu-Frame Anwendung des *ColorReduction Bildeffekts* aufgrund der neuen Farbhäufung im aktuellen Frame entsteht, durch ein globales Maximum bezüglich Vor- und Nachfolgeframe ersetzt und damit das Ergebnis geglättet. Bei der Farbreduktion des Clusterings konnte zwar eine leichte Zunahme bezüglich der Anzahl der Änderungen, aber gemäß der durchschnittlichen Differenzen durchwegs die Tendenz zur Verbesserung der Werte aus der Frame-zu-Frame Anwendung festgestellt werden.

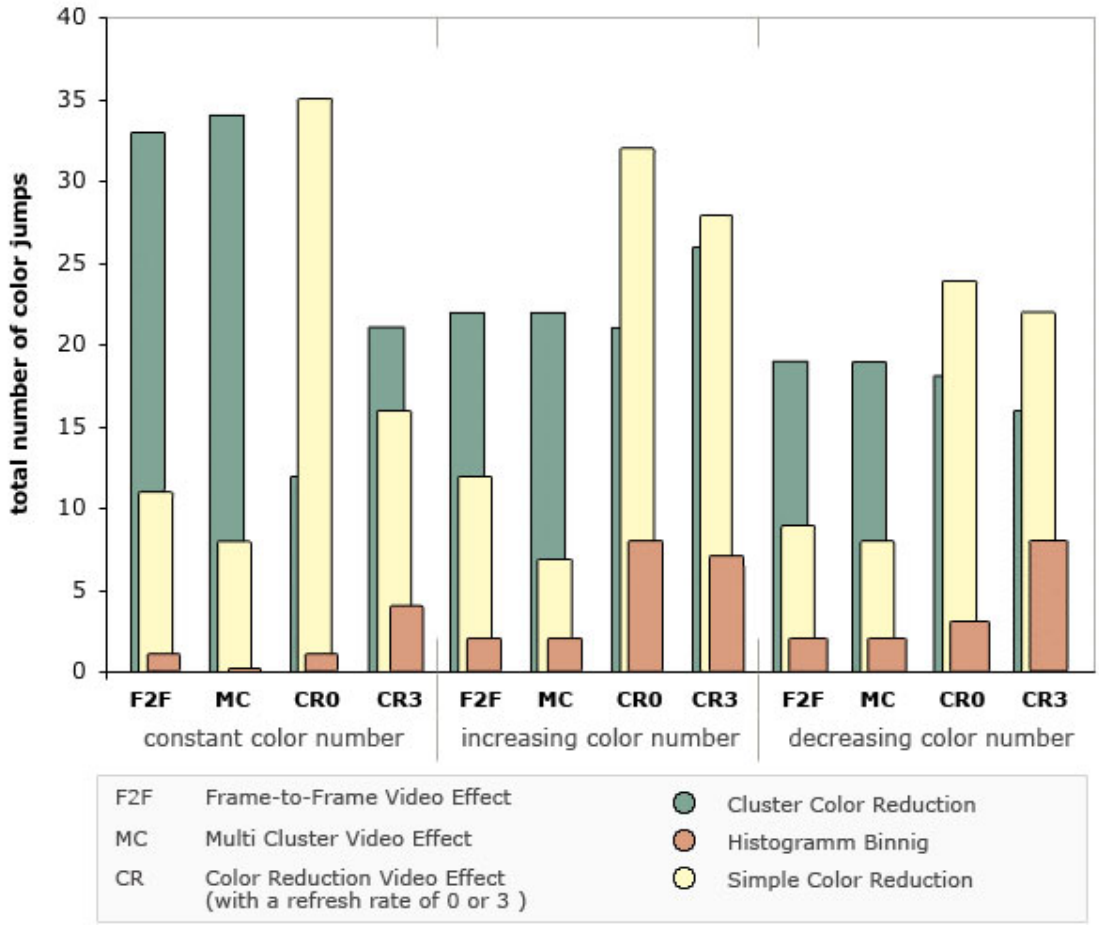

Abbildung 4.15: Dieses Säulendiagramm fasst die Anzahl der Farbsprünge der Testsequenz *car_crash2_lower* bei konstanter (fünf), steigender (von zwei zu fünf) und sinkender (von fünf zu zwei) Farbanzahl unter der Verwendung verschiedener Farbreduktionen zusammen.

Tabelle 4.2: Diese Tabelle fasst die durchschnittliche Höhe der Farbsprünge der Testsequenz *car_crash2_lower* bei konstanter (fünf), steigender (fünf zu zwei) und sinkender (zwei zu fünf) Farbanzahl unter der Verwendung verschiedener Farbreduktionen (Clustering *C*, Histogramm Binning (*H*) und Simple Farbreduktion (*S*)) zusammen. Die Abkürzungen beziehen sich auf die unterschiedlichen Videoeffekte, siehe Legende in Abbildung 4.16.

	konstante Farbanzahl				steigende Farbanzahl				sinkende Farbanzahl			
	F2F	MC	CR0	CR3	F2F	MC	CR0	CR3	F2F	MC	CR0	CR3
C	9,6	7,2	1,8	15,0	14,8	13,6	11	10.6	4,2	4,2	8,8	10,5
H	236,0	0,0	148,0	69,5	0,0	0,0	9,9	14,6	0,0	0,0	24,0	15,0
S	134,6	172,5	35,4	34,1	113,6	73,0	37,9	20,0	108,8	111,8	26,7	20,5

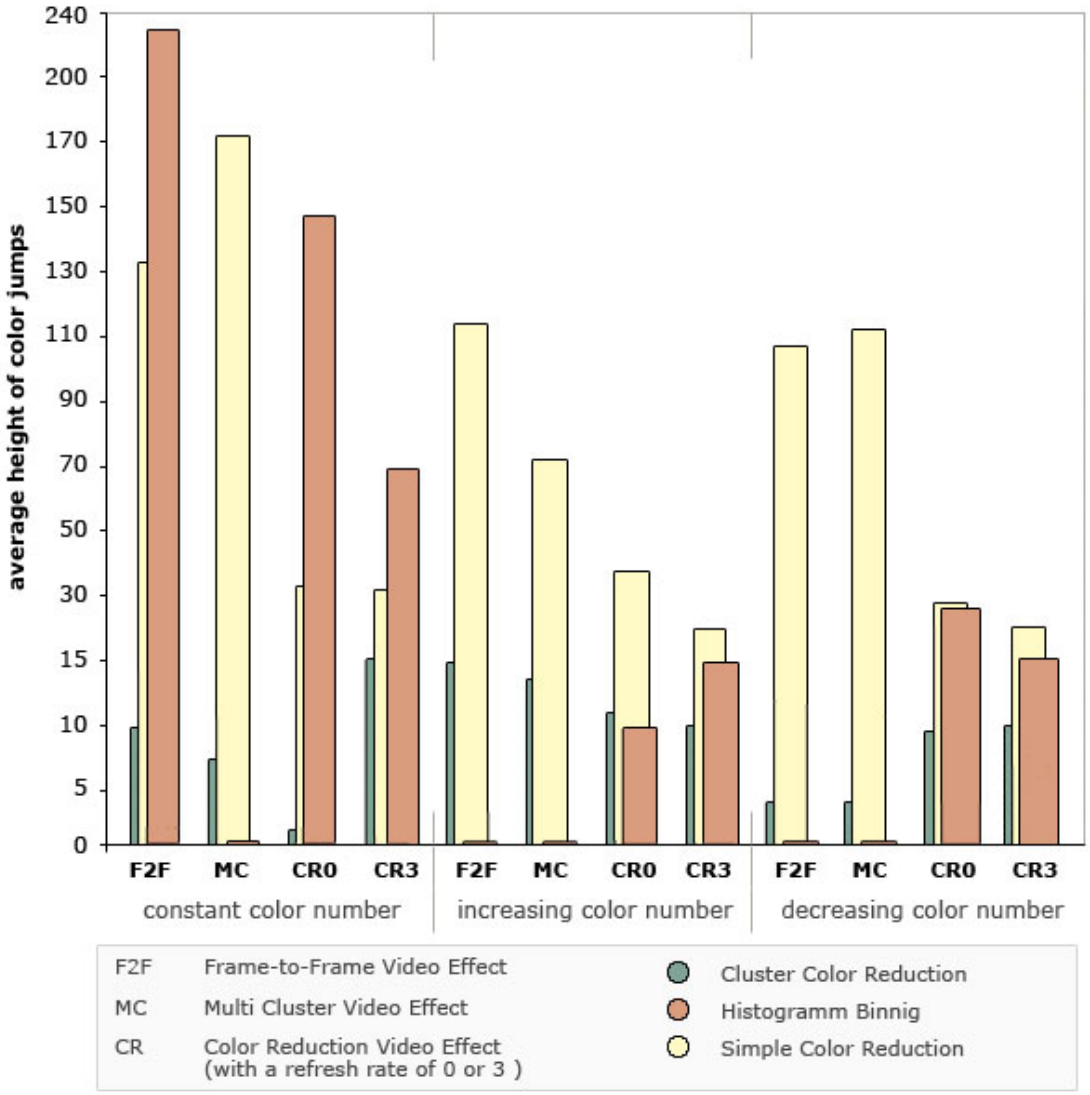

Abbildung 4.16: Dieses Säulendiagramm fasst die durchschnittliche Höhe der Farbsprünge der Testsequenz *car_crash2_lower* bei konstanter (fünf), steigender (von zwei zu fünf) und sinkender (von fünf zu zwei) Farbanzahl unter der Verwendung verschiedener Farbreduktionen zusammen und ist in Bezug auf Abbildung 4.15 zu interpretieren.

Der *ColorReduction Videoeffekt* wurde in zwei Varianten, einerseits bei einer direkten Mittelung der Farben benachbarter Frames und andererseits mit Erneuerungszyklus von drei Binnenframes, untersucht. Erstere funktioniert am besten bei konstanter Farbanzahl und kann für alle drei Reduktionen die Farbkonstanz, insbesondere im Bezug auf die Farbdifferenzen, erhöhen. Mit variierendem Parameter können jedoch nur bedingt Verbesserungen erzielt werden, so ist bei der Cluster Farbreduktion durchwegs und bei steigender Farbanzahl teilweise eine Verminderung von Sprunganzahl bzw. -höhe zu erkennen. Trotz dieses Teilerfolgs sind immer

wieder höhere Ergebnisse als bei der Frame-zu-Frame Anwendung des Bildeffekts zu finden. Das ist bei einer Testsequenz beispielsweise auch beim Histogramm Binning (siehe Abbildung 4.16) der Fall.

Die zweite Variante des *ColorReduction Videoeffekts* verhält sich ähnlich. In der als Beispiel herangezogenen Testsequenz wurden nur unter Verwendung der Farbreduktion via Clustering und mit konstanter sowie sinkender Farbanzahl Verbesserungen bezüglich der Anzahl an Farbsprüngen festgestellt. Die durchschnittliche Höhe der Farbsprünge konnte jedoch in den meisten Fällen abgeflacht werden (siehe Tabelle 4.2), sodass es nicht mehr so abrupt wie in der Frame-zu-Frame Anwendung des Bildeffekts zu diesen Änderungen kommt. Insgesamt, also auch bei Betrachtung der Ergebnisse weiterer Testsequenzen, kann der *ColorReduction Videoeffekt* keine signifikante Minderung der Sprunganzahl erzielen, führt aber dennoch zu einer Glättung der Höhen. Dies ist eine Folge der Mittelung der Farben mit oder ohne Erneuerungszyklus. Der lokale Fehler Farbdifferenz wird durch diesen Effekt von einem auf mehrere Frames verteilt, was in Erhöhung der Sprunganzahl und Verkleinerung der durchschnittlichen Differenz resultiert. Das Hinzufügen von Binnenframes kann diesen Effekt noch weiterführen und entweder die Anzahl der Fehler auf Kosten ihres Ausschlags oder umgekehrt die Höhe der Differenzen bei gleichzeitiger Steigerung ihrer Anzahl minimieren. Allgemein ist zu erwähnen, dass bei höherer Anzahl an Farben weniger sichtbare Farbsprünge vorhanden sind.

4.3.2 Farbsprünge räumlich

Die zeitliche Kohärenz bezieht sich neben der allgemeinen Betrachtung von Farbsprüngen auch auf deren räumliche Verteilung, welche in diesem Unterkapitel behandelt werden soll. Der Begriff „räumliche Farbsprünge“ fasst auch Artefakte, wie das Flimmern einzelner Pixel oder den Wechsel von Farbzuordnungen ganzer Regionen, zusammen.

Die Verteilung der Farben im Raum und die Form der Segmente hängt bei *ColorReduction-* und *MultiCluster Videoeffekt* sowie der Frame-zu-Frame Anwendung des Bildeffekts in erster Linie von der gewählten Farbreduktion und damit der Qualität der Abbildung der realen Szene in den gewählten Farbraum ab. Innerhalb von nicht texturierten Regionen ist bei konstanter Farbanzahl und normalem Verlauf die Farbzuordnung der einzelnen Pixel stabil, wodurch sich bewegende Objekte einer Szene von BetrachterInnen als Animation wahrgenommen werden können. An Rändern und in stark texturierten oder verlaufenden Regionen ist die Wahrscheinlichkeit, flackernde Ergebnisse zu bekommen, höher (siehe Abbildung 4.17).

Da die Simple Farbreduktion oder die Reduktion mittels Clustering in der Regel kontrastreichere Abstraktionen als das Histogramm Binning erzeugen und mehr Details darstellen können, sind auch flimmernde Pixel häufiger und sichtbarer (siehe Tabelle 4.3). Erhöht man die Anzahl der Iterationen und damit die Genauigkeit der Segmentierung, können dahingehend

Abbildung 4.17: Diese Abbildungen zeigen jeweils einen vergrößerten Ausschnitt zweier Nachbarframes einer abstrahierten Videosequenz. Die *ersten beiden Nachbarn* einer Sequenz wurden dabei mit Cluster Color Reduction abstrahiert und die *zwei mittleren Ausschnitte* mit Histogramm Binnig. *Rechts* ist ein Beispiel der räumlichen Verteilung einer Farbreduktion des *LAB-ColorReduction Effekts*. Die *roten Markierungen* weisen auf kleine Abweichungen der räumlichen Farbzuordnung in den benachbarten Frames hin.

Tabelle 4.3: Diese Tabelle beinhaltet in adjazenten Frames gezählte, durchschnittliche räumliche Änderungen. Die Anzahl der geänderten Pixel wurde dabei über die auf vier Frames verkürzten und mit Frame-zu-Frame Anwendung des Bildeffekts abstrahierten Testsequenzen gemittelt. Die Zahlen beziehen sich dabei nicht nur auf flackernde Pixel, sondern beinhalten auch Änderungen, die sich durch die Bewegung in der Szene ergeben.

Testsequenz	Farben	Cluster	Histogramm Binning	Simple Farbreduktion
car_crash2_lower	8	46947	13160	36151
car_crash2_lower	30	61284	41545	30335
lightsaber	8	43391	40317	33077
lightsaber	30	91571	62506	33077
desk_lower	8	46821	49173	18804
desk_lower	30	69547	62660	40681

und bezüglich der Vermeidung gravierender Änderungen bei Farbsprüngen Verbesserungen erzielt werden. Ränder starker Kanten oder Objektkonturen sind im Vergleich zu Farbverläufen weniger problematisch (siehe Abbildung 4.17 und Abbildung 4.18). Bei Anwendung der Videoeffekte ohne Hinzufügen von Konturen, welche vor der Abstraktion ermittelt wurden, können zwar leichte Änderungen beobachtet werden, die aber weniger einen fehlerhaften oder störenden als handgezeichneten Eindruck vermitteln.

Eine Abweichung der Segmentbildung, wie sie in der Frame-zu-Frame Anwendung des Bildeffekts entsteht, ist nur bei der Anwendung des *MultiCluster Videoeffekts*, des *ColorReduction Videoeffekts* mit der Wahl eines Erneuerungszyklus größer als Null oder des *Frame-Differencing Videoeffekts* möglich, da die Farbe sonst lediglich gemittelt und keine Änderung

Tabelle 4.4: Diese Tabelle beinhaltet in adjazenten Frames gezählte, durchschnittliche räumliche Änderungen, die im Ergebnis der Videoeffekte ermittelt wurden. Die Anzahl der geänderten Pixel wurde dabei über die auf vier Frames verkürzte *lightsaber*-Testsequenz gemittelt. Die Markierung (*1*, *0*) neben dem Namen der Videoeffekte steht für die Anzahl verwendeter Binnenframes. Bei der mit *x* geschriebenen Variante des *FrameDifferencing Videoeffekts* wurde auf das Mitteln der Farben gänzlich verzichtet.

Videoeffekt	Farben	Cluster	Histogramm Binning	Simple Farbreduktion
ColorReduction 1	8	44574	41181	267119
ColorReduction 1	30	96926	66028	321104
ColorReduction 0	8	42963	40048	32771
ColorReduction 0	30	91644	61290	49234
MultiCluster	8	51916	41783	34712
MultiCluster	30	100923	62468	52496
FrameDifferencing x	8	30484	30320	204702
FrameDifferencing x	30	63983	47117	35916
FrameDifferencing 0	8	30516	30320	21982
FrameDifferencing 0	30	30516	47000	46853
FrameDifferencing 1	8	30808	30190	170625
FrameDifferencing 1	30	66085	49955	260273
Videoeffekt				
LABColorReduction		76076		

am Labeling vorgenommen wird. Die Auswertung von verschiedenen, durch den *MultiCluster Videoeffekt* abstrahierten, Testsequenzen ergab visuell keine signifikante allgemeine Erhöhung oder Verminderung des Flimmerns. In einigen Bereichen wurde die räumliche Verteilung zwar stabiler, aber gleichzeitig in anderen unruhiger (siehe Abbildung 4.18). Das Zählen der in adjazenten Frames veränderten Pixel (siehe Tabelle 4.3 und Tabelle 4.4) machte einen leichten Trend zu verstärktem Flimmern sichtbar. Im Vergleich zum Labeling, welches bei sonst gleichen Einstellungen aus einer Frame-zu-Frame Anwendung resultiert, können bei Farbreduktion mittels Clustering in den Testsequenzen wenige Unterschiede wahrgenommen werden. Die Wahl des Histogramm Binnings resultiert hingegen schon bei niedriger Farbanzahl ($k \in [4, 10]$) in einem sichtbar von der Frame-zu-Frame Anwendung abweichenden Labeling, was auf die engere Kopplung von Farbwahl und räumlicher Verteilung zurückzuführen ist. Da dieser Faktor stark von der verwendeten Szene abhängt, kann auch hier keine allgemein gültige Aussage getroffen werden. Bei den betrachteten Testsequenzen war im Effektframe nur teilweise weniger Flimmern zu erkennen als in der Frame-zu-Frame Anwendung (siehe Tabelle 4.3 und Tabelle 4.4). Die Simple Farbreduktion erzeugt genauso unruhige Ergebnisse wie bei ihrer Frame-zu-Frame Anwendung und ist schon wegen der, auch bei stabilen Parametern und Szenen, wechselnden Farben weniger zweckdienlich.

Abbildung 4.18: Die Differenzbilder wurden jeweils aus zwei adjazenten Frames einer bereits abstrahierten Videosequenz berechnet. Die *roten Markierungen* weisen darin auf Unterschiede hin. Das *erste Bild* ist aus der Basis einer Anwendung des *ColorReduction Videoeffekts* ohne Binnenframes entstanden, das *zweite Bild* mit drei Binnenframes. Die Farbreduktion wurde dabei mittels Clustering durchgeführt. Das *dritte Bild* zeigt die Differenzen, die sich bei Anwendung des *MultiCluster Videoeffekts* ergeben. Im *letzten Bild* wurden die Unterschiede des *LABColorReduction Videoeffekts* bei sechs Helligkeitsbins visualisiert.

Der *ColorReduction Videoeffekt* weicht nur innerhalb des Erneuerungszyklus vom Frame-zu-Frame Labeling ab. Da das Histogramm Binning (siehe Abschnitt 4.3.1) wenig Farbsprünge und damit auch Änderungen der räumlichen Verteilung als Nebeneffekt hat, ergibt das Mitteln der Farben und die Neuberechnung der Labels kaum variierende Ergebnisse. Erfolgt die Verarbeitung mittels Clustering, wird durch die Erhöhung der Binnenframes ebenfalls keine hinreichende Verbesserung erzielt. In der abstrahierten *lightsaber*-Testsequenz wurde beispielsweise bei unterschiedlichen Farbparametern (siehe Abbildung 4.18 für fünf Farben und Tabelle 4.4) jeweils erhöhtes Flimmern festgestellt.

Der *FrameDifferencing Videoeffekt* hingegen modifiziert das Ergebnis der sequenziellen Anwendung des Bildeffekts oder des *ColorReduction Videoeffekts* direkt auf Basis der Differenzen zwischen benachbarten Frames. Er hat das Ziel räumliche Farbsprünge zu reduzieren und kann es bei der Wahl eines geeigneten Schwellenwerts auch erreichen (siehe Tabelle 4.4). In der Testsequenz fielen beispielsweise mit einem niedrigen Schwellenwert (*13*) und bei konstanter Farbanzahl nicht nur Ungenauigkeiten an Rändern, sondern auch textur- oder verlaufbedingte Ausreißer geringer aus (siehe Abbildung 4.19 im Vergleich zu Abbildung 4.18). Ergänzt der gewählte Schwellenwert die Bildsequenz und ihre Abstraktion nicht, kann dies zu weniger oder keiner Verminderung flimmernder Pixel oder im schlimmsten Fall zu Artefakten im Ergebnis führen (siehe Abbildung 4.19, *rechts*).

Die Videoabstraktion von [93] oder vielmehr die im Zuge dieser Diplomarbeit adaptierte Variante erzeugt bei Farbverläufen zwar keine „körnigen“ Segmentgrenzen, aber größere abgerundete Artefakte (siehe Abbildung 4.18 und Abbildung 4.17), welche die zeitliche Kohärenz ebenfalls negativ beeinflussen können. In Tabelle 4.4 ist ein Beispiel mit vergleichsweise vielen räumlichen Farbsprüngen aufgeführt. Beim *LabColorReduction Videoeffekt* ist die Anzahl der veränderten Pixel jedoch mit Vorsicht zu betrachten, da auch kleine, für das menschliche Auge

Abbildung 4.19: Die ersten beiden Differenzbilder wurden jeweils aus zwei benachbarten Frames einer bereits abstrahierten Videosequenz berechnet. Für die Stilisierung wurde der *FrameDifferencing Videoeffekt* mit zusätzlicher *ColorReduction Videoeffekt* Funktionalität benutzt. Beim *rechten Bild* beträgt der Schwellenwert *80* und beim *mittleren 13*. Das *linke Bild* zeigt Artefakte auf, welche durch die Wahl eines weniger geeigneten Schwellenwerts (in diesem konkreten Beispiel *44*) entstehen können.

kaum oder gar nicht sichtbare Änderungen mitgezählt wurden.

Das Interpolieren der Parameter über die Zeit allgemein, aber insbesonders des Parameters zur Steuerung der Farbanzahl, beeinflusst ebenfalls den Fluss des Ergebnisvideos. Das Erhöhen oder Vermindern der Details in der abstrahierten Szene ist besonders bei der Simplen Farbreduktion mit niedriger Farbanzahl sichtbar, da diese zu ganz anderen räumlichen Farbzusammenstellungen und somit starken Artefakten führen kann. Die Anwendung der Videoeffekte kann nur leichte Verbesserungen dieser Situation herbeiführen. Histogramm Binning und Clustering verhalten sich deutlich stabiler, wobei auch bei ihnen Farbänderungen möglich sind. Farben, welche vor der Erhöhung des Parameters noch nicht vorhanden waren und gegebenenfalls für ganze Objekte immanent sind, können ein Grund dafür sein. Objekte, die zuvor eher im Hintergrund standen, können nach der Änderung hervorstechen bzw. umgekehrt. Dieses Verhalten kann durch den *MultiCluster Videoeffekt* nicht verhindert werden.

Die Entscheidungsfunktion des *FrameDifferencing Videoeffekts* wird an der Stelle der konkreten Veränderung des Parameters unterdrückt, um keine „veralteten“, möglicherweise abweichenden Farben sichtbar als Artefakte im Ergebnisframe zu erhalten. Da, zumindest zu diesem Zeitpunkt, kein alternativer Transformationsschritt definiert ist, können die Übergänge auch durch diesen Effekt nicht abgeschwächt werden.

Im *ColorReduction Videoeffekt* sind die mit einer Änderung der Farbanzahl einhergehenden Veränderungen durch die Mittelung nicht mehr derart abrupt und neue Farben besser an bereits vorhandene angepasst. Zusammen mit der stabilen Farbwahl des Histogramm Binnings (siehe auch Abschnitt 4.3.1) kann eine bessere Kohärenz erreicht werden. Das Hinzukommen einer neuen Farbe wird als solches wahrgenommen und nicht wie beim Extremfall, der Simplen Farbreduktion, als eine neue Zusammenstellung von Farben und Segmenten. Die Farbreduktion

mittels Clustering liefert ebenfalls gute Ergebnisse. Während bei der sequenziellen Anwendung des Bildeffekts das Hinzufügen einer Farbe noch zu starken Kontrasten führen kann, werden diese im *ColorReduction Videoeffekt* geglättet und die Übergänge damit weniger abrupt (siehe auch Abschnitt 4.3.1).

Neben dem Flackern von einzelnen, in adjazenten Frames unterschiedlich zugeordneten Pixel sind springende Farbzuordnungen ganzer Regionen und stark variierende Farbzusammenstellungen Faktoren, die bei der Frame-zu-Frame Anwendung des Bildeffekts die zeitliche Kohärenz negativ beeinflussen. Diese Effekte konnten vor allem bei niedriger Farbanzahl und abrupten Änderungen der Szene durch z.B. Lichtsprünge beobachtet werden (siehe Abbildung 4.20). Da die vom *ColorReduction Bildeffekt* abgeleiteten Videoeffekte auf der Annahme basieren, dass sich die Farbe zwischen benachbarten Frames nicht oder nur geringfügig ändert, reagieren diese mit je nach Stärke der Helligkeitsänderung mehr oder weniger vom Vorgängerframe abweichenden Ergebnissen. Es werden neue Farben eingeführt und andere verworfen (Histogramm Binning), Regionen andere, bereits vorhanden gewesene Farben zugeordnet (Clustering) oder unterschiedliche Farben oder Segmente dargestellt (Simple Farbreduktion). Die Videoeffekte erzielen zwar eine Verbesserung des Verhaltens der Frame-zu-Frame Anwendung des Bildeffekts, können aber mit Ausnahme des *ColorReduction Videoeffekts* und einer gezielten Platzierung der Farbsprünge zwischen zwei Endframes diesen Effekt nicht eliminieren. Auch die Wiederverwendung von Farben alter Pixel vor der Helligkeitsänderung im *Frame-Differencing Videoeffekt* ist unter Umständen (der Schwellenwert erkennt keine Änderung im Original, aber im abstrahierten Bild, siehe Kapitel 3.2.3) im Ergebnis sichtbar.

Die Abstraktion, wie sie von [93] präsentiert wurde, ist in diesem Fall stabiler (siehe Abbildung 4.20) und adaptiert starke Helligkeitsänderungen durch Aufhellen entsprechender Bereiche oder der gesamten Abstraktion. Dieses Verhalten erlaubt die für emotionale Gestaltung von Szenen wichtigen Reizwechsel beizubehalten [63] und auf neu eintretende Objekte besser zu reagieren.

4.3.3 Kanten im zeitlichen Verlauf

Ein weiterer Faktor, der die zeitliche Kohärenz beeinflusst, sind die (optionalen) Kanten bzw. Detailzeichnungen, auf welche in diesem Abschnitt kurz eingegangen wird. Besonders starke Kanten sind im allgemeinen sehr stabil (siehe Abbildung 4.21). Gegebenenfalls auftretende, leichte Änderungen der Grauwerte an den Seiten breiter Linien werden weniger als Flimmern, sondern als handgezeichnet bzw. als ein weiteres Merkmal von Zeichentrickfilmen wahrgenommen.

Im Gegensatz dazu wirken sich texturierte Bereiche, insbesondere mit starken Kontrasten, negativ auf die zeitliche Kohärenz aus. Kleinere Detailzeichnungen, wie sie beispielsweise in

Abbildung 4.20: Diese Abbildung zeigt exemplarisch das Verhalten der Videoeffekte bei variierenden Lichtverhältnissen. Das erste Bild zeigt dabei jeweils ein Frame vor dem Farbsprung und das zweite ein Frame zum Zeitpunkt des Farbsprungs.

Abbildung 4.21 durch die Musterung der Vorhänge entstehen, können in adjazenten Frames geringfügig in Deckkraft und Länge variieren. Durch allgemeines Erhöhen der Deckkraft oder Vermindern der Kantendichte wird damit zusammenhängendes Flimmern verringert. Darüber hinaus hat der Einsatz des *FrameDifferencing Videoeffekts* Einfluss auf Kantenpixel. Das beschriebene Verhalten der Kanten im zeitlichen Verlauf ist auch bei der im Zuge dieser Diplomarbeit entstandenen Implementierung der Video Abstraktion von [93] zu beobachten.

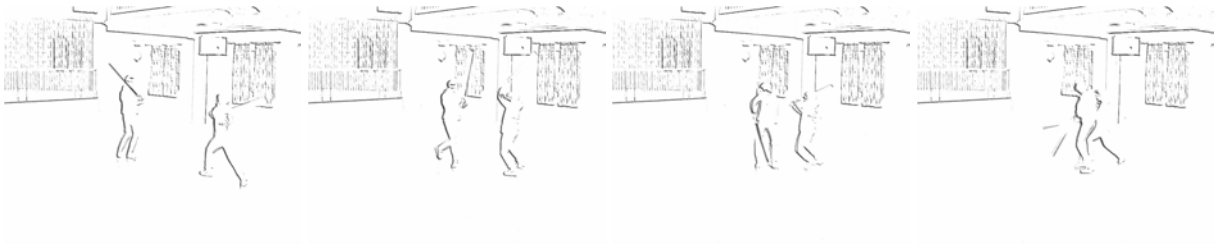

Abbildung 4.21: Diese vier Frames aus der Testsequenz *lightsaber* beinhalten Kanten mit der Deckkraft von *0.6* und dem Dichtewert von sechs. Es ist sichtbar, dass starke Kanten (u.a. im oberen Bereich der Einzelbilder) deutlich stabiler sind als die Musterung im Hintergrund.

Kapitel 5

Implementierung

Dieses Kapitel beschreibt detailliert die im Rahmen der Diplomarbeit durchgeführte Implementierung der Effektalgorithmen. Zuerst werden allgemeine Punkte wie etwa die Entwicklungsumgebung geklärt. Danach wird speziell auf das Framework der Effekte, die *Video Effect Library* (VEL), eingegangen. Anschließend werden zwecks Verständnis wichtige Klassen dokumentiert. Methoden, die ausschließlich zu Visualisierungs- und Testzwecken oder zur Datenverarbeitung (Einlesen und Speichern) dienen, werden nicht näher behandelt. Schlussendlich stellt dieses Kapitel Schätzungen der Laufzeit wichtiger Effektalgorithmen zur Verfügung.

5.1 Allgemeine Hinweise

Die Algorithmen der Effekte wurden in Java auf einem PowerBookG4 mit Mac OSX als Teil der VEL implementiert. Da das Framework selbst eine Java Library ist, stellte sich während der Konzeptionsphase dieser Diplomarbeit nicht die Frage nach einer für die Videoverarbeitung besonders gut geeigneten Programmiersprache. Aus didaktischer Perspektive sprechen mehrere Gründe für die Wahl. An der Technischen Universität Wien wird eine Großzahl an Lehrveranstaltungen, wie auch die „Einführung in das Programmieren“, der Informatikstudiengänge in Java abgehalten, weshalb bei StudentInnen im Masterstudium gute Kenntnisse vorausgesetzt werden können. Ein zweiter wichtiger Punkt, der für Java spricht, ist seine Plattformunabhängigkeit, welche organisatorischen Aufwand mindert.

Für das Clustering wird die k-Means Implementierung aus [67] verwendet. Die Realisierung des Bilateralen Filters entspricht der in [84] beschriebenen Implementierung und ist eine Adaption der in [13] zur Verfügung gestellten Realisierung. Die Verarbeitungsschritte des *LABColorReduction Effekt* entsprechen fast vollständig der Beschreibung der Video Abstraktion in den zugehörigen Papers [93][94].

5.2 Video Effect Library

Die Video Effect Library ist eine Sammlung von Bild- und Videoverarbeitungsalgorithmen, die im Studienjahr 2007/2008 in der Laborübung „Videoverarbeitung“ verwendet wurde. Ihre Kernfunktionalität findet sich in den Klassen des Packages *standard*. Dazu gehören einerseits Strukturen, zum Beispiel in Form von Interfaces für Bild- und Videoeffekte, andererseits aber auch verschiedene *Listener*, welche Kommunikation zwischen Komponenten ermöglichen. Dieser Abschnitt gibt eine Einführung in die wichtigsten Datenstrukturen und die grundlegende Funktionsweise der VEL (siehe Abbildung 5.1).

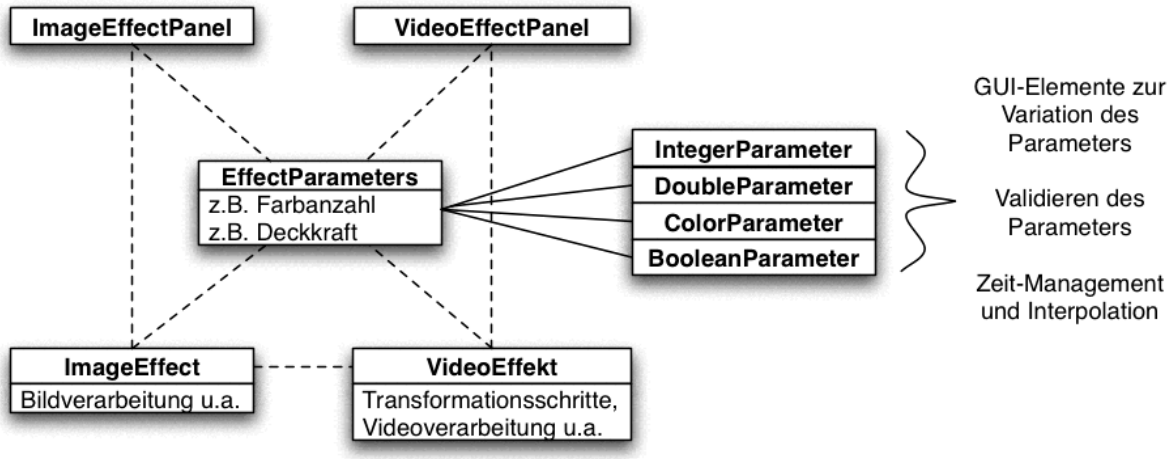

Abbildung 5.1: Dieses Diagramm stellt die wichtigsten Komponenten der VEL und ihren Zusammenhang vereinfacht dar. Die Parameter von Bild- und Videoeffekten werden in speziellen Containern, *Parameterklassen*, gespeichert, welche je nach Typ unterschiedliche GUI Elemente liefern (z.B. *JSlider* oder *JCombobox*), für das Zeitmanagement verantwortlich sind und auch gewählte Werte auf ihre Richtigkeit überprüfen.

5.2.1 Struktur von Bildeffekten

Das Interface *ImageEffect* gibt die Richtlinien für die Implementierung weiterer Bildeffekte vor. So muss ein Originalbild, das später vom Effekt bearbeitet und als Effektbild retourniert wird, gesetzt werden können. Der Rendervorgang des Bildeffekts soll als Thread und durch einen herkömmlichen Funktionsaufruf, **render**(), realisiert werden. Die Ausführung als Thread läuft mittels der Methode **run**(), die das *Runnable* Interface implementiert. Mit **abort()** wird ein Abbruch-Flag gesetzt, welches während des laufenden Prozesses regelmäßig überprüft wird. Durch die Klasse *ThreadEventManager*, die vom *ImageEffect* implementiert wird, können sich andere Komponenten als Listener beim Bildeffekt eintragen. Dies nützt zum Beispiel das User Interface, um den Status der Verarbeitung als Prozentzahl anzugeben.

5.2.2 Struktur von Videoeffekten

In der VEL implementierte Videoeffekte erben, analog zu den Bildeffekten, Struktur und Funktionen des Interfaces *VideoEffect*. Dazu gehören globale Variablen für die Bildsequenz und die aktuelle Position auf der Zeitachse bzw. in der Sequenz. Sie enthalten die Hauptschleife, in der Frames einer Sequenz aufgerufen, verarbeitet und die Transformationsschritte für die Erstellung eines Videoeffekts implementiert werden können (**process**()). Der *ThreadEventManager* ermöglicht es auch dem Videoeffekt Listener anzufügen.

5.2.3 Struktur von Keyframes

Parameterklassen dienen als Container für Eigenschaften von Bild- und Videoeffekten. Die abstrakte Klasse *EffectParameters* verwaltet die Parameter mit vier verschiedenen *Hashtables*, eine pro derzeit unterstütztem Typ (siehe Abschnitt 5.2.4), und stellt die Schnittstelle zwischen spezialisierten, pro Effekt selbst definierten Parametern und allgemeiner Verwaltung von Eigenschaften dar. Sie ermöglicht das Aufrufen von Methoden zum Setzen und Erhalten der Parameterwerte und ihrer GUI-Elemente (siehe Abbildung 5.1).

Während beim Bildeffekt nur Werte für einen Zeitpunkt von Nöten sind, werden bei Videoeffekten unter Umständen zwei oder mehr Keyframes entlang einer Zeitleiste gesetzt. Jede Klasse, welche *EffectParameters* erweitert, erbt zu diesem Zweck zusätzliche zeitabhängige **get-** und **set-**Methoden, sodass zu jedem Punkt auf der Zeitachse die zu dieser Zeit aktuellen Einstellungen zur Verfügung stehen. Jeder Bild- und Videoeffekt sollte einer Erweiterung der Klasse *EffectParameters* zugeordnet sein und seine Parameter dort speichern.

5.2.4 Struktur von Parametern

In der aktuellen Implementierung gibt es vier verschiedene Parameterklassen, welche nicht nur jeweils einen Wert des entsprechenden primitiven Datentyps beinhalten und verwalten, sondern auch ein passendes GUI-Element (für Bild- und Videoeffekt) und seinen *ActionListener* implementieren sowie mit dem passenden *InterpolationContainer* zwischen den gesetzten Werten linear interpolieren.

- *DoubleParameter* für Gleitpunktzahlen
- *IntegerParameter* für ganze Zahlen
- *BooleanParameter* für die logischen Werte *true* und *false*
- *ColorParameter* für Farbwerte

5.3 Klassenübersicht

Neben der Implementierung der Kernfunktionalität sowie der Bild- und Videoapplikation beinhaltet die Library noch weitere Klassen, welche je nach Aufgabengebiet in unterschiedliche Packages gegliedert sind (siehe Abbildung 5.2).

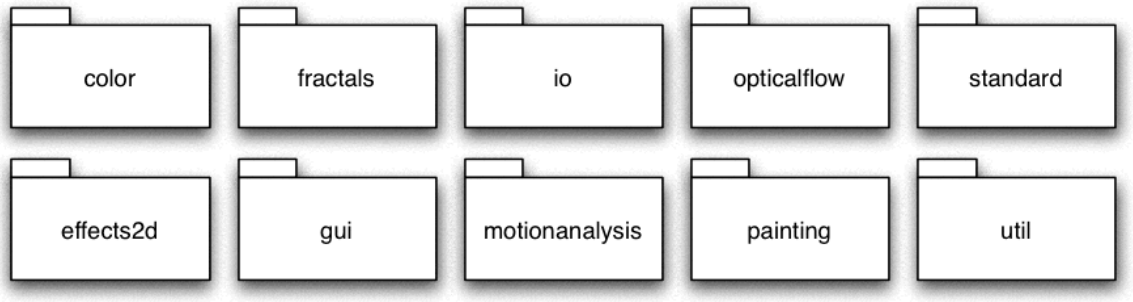

Abbildung 5.2: Neben dem Package *standard*, welches die Kernfunktionalität der Video Effect Library beinhaltet, besteht die Bibliothek aus neun weiteren für Effekt- und Parameterklassen (*color*, *fractals*, *motionanalysis*, *painting*, *effects2d*), Optical Flow Berechnungen (*opticalflow*), User Interface (*gui*), Basisfunktionen der Bild- und Videobearbeitung (*util*) und Einlesen von Konfigurationsdaten (*io*).

Im weiteren Verlauf dieses Kapitels werden die Implementierung der im Zuge dieser Diplomarbeit entstandenen Effekte und die dafür erstellten Hilfsklassen verschiedener Packages genauer beschrieben.

5.3.1 Effekte mit Farbmodifikation

Das Package *color* beinhaltet Effekte, ihnen zugeordnete Parameter- sowie speziellere Hilfsklassen, die hauptsächlich mit Farbänderungen arbeiten, wie die im Zuge dieser Diplomarbeit implementierten Bild- und Videoeffekte. Darüber hinaus verfügt die VEL über ähnliche Packages wie *effects2d* oder *painting*, in denen weitere Effektalgorithmen zusammengetragen werden. Letzteres beinhaltet beispielsweise eine Variation des in Abschnitt 2.2.2 vorgestellten *Painterly Renderings* von [34].

ColorReduction Bildeffekt

Die Bildeffektklasse *ColorReductionEffect* wandelt das Originalbild zu einem farbreduzierten bzw. cartoonartigen Effektbild. Wie in Abschnitt 5.2.1 schon angedeutet, ist die **render()**-Methode das Herzstück dieser Klasse. In ihr werden je nach Eigenschaften der zugeordneten Parameterklasse unterschiedliche Algorithmen und Parameterwerte gewählt. Der Bildeffekt macht

dazu von verschiedenen Klassen Gebrauch, welche in diesem Kapitel noch genauer beschrieben werden.

Es gibt drei Möglichkeiten, die jeweils in einer der Farbreduktionsklassen implementiert sind, wie die Farbreduktion erfolgen kann (siehe Abschnitt 3.1.1). Für eine zusätzliche Abbildung der ermittelten Farben auf Schwarz, Weiß und eine weitere beliebige Farbe verwendet die Bildeffektklasse ein *DistanceMatrix*-Objekt. Weitere Farbmodifikationen (Sättigung, Helligkeit und Farbton) werden mit dem Package *util.color* durchgeführt. Die Kantenerkennung, welche je nach Parametern Linienzeichnungen ergänzen kann, erfolgt durch die Klasse *SmoothEdges* auf die in diesem Abschnitt näher eingegangen wird. Eine zusätzliche Kontraststärkung ist in der Klasse *HSBBrightnessEnhancement* implementiert.

Die Methoden zum Auslesen der Pixeldaten sind in der Klasse *PixelData* zu finden. Die Parameter des *ColorReduction Bildeffekts* und der ihm zugeordneten Videoeffekte werden in *ColorReductionParameters* gespeichert.

Farbreduktionsklassen

Alle Farbreduktionsklassen implementieren das Interface *IColorReduction*. Die somit vorgegebene Struktur sieht vor, dass Methoden für das Setzen eines Originalbilds, die Reduktion der Farbdaten sowie das Retournieren der Ergebnisse zu implementieren sind (siehe Abbildung 5.3). Das Ergebnis der Farbreduktion kann in zwei verschiedenen Formen vorliegen, einerseits als farbreduziertes Bild (*BufferedImage*), andererseits als Kombination einer LUT (in Form eines Farbarrays) und auf diese verweisende Label der Pixel, die ebenfalls in einem Array gespeichert sind. Für Farbreduktionseffekte wird ausschließlich letztere Möglichkeit verwendet, um weitere Farbänderungen effizienter durchführen zu können.

Eine der Farbreduktionsklassen, *ClusterColorReduction*, löst die Aufgabe durch Clustering der Pixelfarben. Dieser Vorgang ist mit dem Package *util.statistic* realisiert, wobei die Wahl der Startzentren (implementiert z.B. in *util.statistic.MostPopularDataPoints*) mit Hilfe eines Farbhistogramms (*util.BinnedColorHistogram*) durchgeführt wird. Davon abgeleitet, bezieht der Spezialfall *FlowClusterColorReduction* zusätzlich Optical Flow Daten, welche zuvor gesetzt und gewichtet werden, in das Clustering mit ein.

Die Klasse *FastColorReduction* vereinfacht diesen Prozess und bildet mit dem vereinfachten Farbhistogramm *util.BinnedColorHistogram* die Farben eines Bilds auf die am häufigsten in ihm vorkommenden Farben ab.

SimpleColorReduction vernachlässigt die Häufigkeit der Farben im Bild und konzentriert sich auf die uniforme Quantifizierung der Farbpalette. Die Farbreduzierung selbst ist eine statische Methode in *util.image.ImageTransformations*. Durch *java.awt.image.LookupTable* wird dieser Vorgang möglichst effizient durchgeführt.

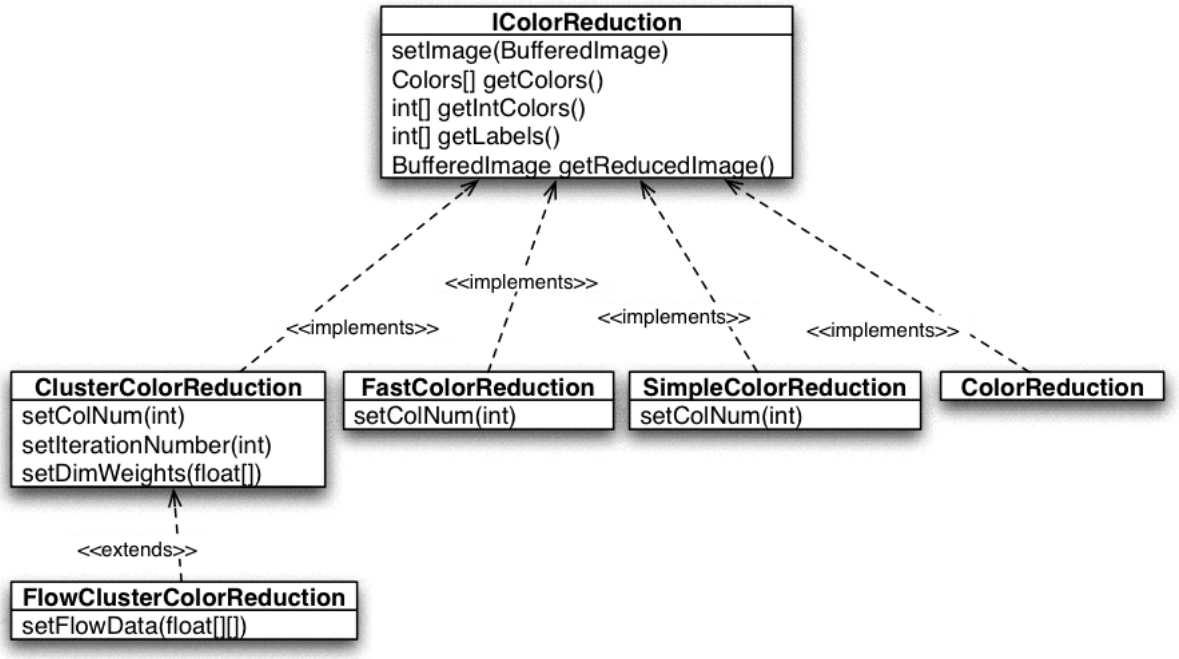

Abbildung 5.3: Dieses UML Diagramm bietet eine Übersicht der vom *ColorReductionEffect* verwendeten Farbreduktionsklassen. Sie implementieren die Funktionen des Interfaces *IColorReduction* und können, wie *ClusterColorReduction*, darüber hinaus noch weitere Parameter definieren. Die Modellierung mit Interfaces erlaubt einfaches Hinzufügen von weiteren Ansätzen zur Farbreduktion.

Die Klasse *ColorReduction* ist keine der im Bildeffekt zur Auswahl stehenden Farbreduktionen, sondern wird für die Interpolation von Farben über ein Intervall im *ColorReductionVideoEffect* benötigt. Aus diesem Grund ist auch keine **set**-Methode für die Anzahl der Farben notwendig. Stattdessen wird ein vorhandenes Farbarray gesetzt, und die Farben der Bildpixel werden aufgrund der Ähnlichkeit auf dieses abgebildet.

Dank der Modellierung als Interface besteht die Möglichkeit, den Effekt ohne viel Aufwand um weitere Ansätze zur Farbreduktion zu ergänzen und bereits vorhandene Algorithmen um beispielsweise genauere Segmentierungen zu erweitern.

Zusätzlich zu den Implementierungen von *IColorReduction* existiert eine mit diesen verwandte, speziell für die *LabColorReduction Effekte* ausgerichtete, Klasse *LabColorReduction*. Sie iteriert über das gesamte Bild und reduziert dabei Helligkeitswerte durch Binning und eine geglättete Stufenfunktion (siehe Kapitel 3 und [93]). Mit den Methoden **setBinNumber**(*int*), **setSharpness**(*float*) sowie **isFixed**(*boolean*) können die Parameter der Farbreduktion gesetzt werden. Neben **get**-Methoden zum Ermitteln der Anzahl an Bins sowie deren Breite, wird mit **getColorReducedImage**(*BufferedImage*) und **getColorReducedData**(*int[][]*) eine Quantisierung durchgeführt.

Distanzberechnungen

Aufwendige Distanzberechnungen photometrischer Differenzen, wie das Erstellen von Farbabbildungen, werden mit Hilfe der Klasse *DistanceMatrix* realisiert. Ihre Instanzen berechnen Differenzen zwischen Wertepaaren und ermitteln das Paar mit dem kleinsten oder größten Abstand zu einem der Werte (siehe Kapitel 3.1.2). Diese Datenstruktur kann im wesentlichen beim eindimensionalen Fall als Matrix und bei höherer Dimensionalität als Matrix mit mehreren Ebenen betrachtet werden. Ihre **get-**Methoden erlauben es, Informationen über das konkrete Objekt des Typs zu erlangen. Dazu gehören Routinen, die Distanzmatrizen pro Dimension oder über alle Dimensionen hinweg berechnen und retournieren oder weitergehend die kleinste Differenz für einen Eintrag ermitteln. **computeDistances**() berechnet beispielsweise die *Euklidische Distanz* [85] pro Dimension der Werte. Enthalten die Wertearrays der Datenstruktur z.B. jeweils drei Arrays, so wird daher für jede dieser drei Dimensionen paarweise die Differenz (*iValues[dim][i] - iValues[dim][j]*) berechnet. Einige Methoden dieser Klasse benötigen einen Sortieralgorithmus. Dafür wird ein modifizierter *BubbleSort* [89] verwendet.

Kantenerkennung

Die Klasse *SmoothEdges* dient dazu, Kanten eines Bilds zu erkennen und spezifizierten Parametern entsprechend in visuell ansprechenden, geglätteteren Versionen darzustellen (siehe Kapitel 3.1.3). Ähnlich zu [93] kann neben den für Kantenoperatoren spezifischen Parametern ihre Schärfe, Deckkraft und Häufigkeit (**setSharpness**(*float*), **setOpacity**(*float*), **setDensity**(*int*)) beeinflusst werden.

Für die eigentliche Kantenerkennung stehen derzeit zwei Filter zur Verfügung, die jeweils durch Faltung, implementiert mit *java.awt.image.ConvolveOp*, des entsprechenden Filterkerns mit dem spezifizierten Bild realisiert werden.

- **getSobelFilteredImage**(*BufferedImage ...*)
- **getRobertsFilteredImage**(*BufferedImage ...*)

Die anschließende Extraktion der darzustellenden Kanten kann wahlweise durch einen fix gewählten oder vom Bild abhängigen und mit *util.BinnedGrayHistogram* ermittelten Schwellenwert durchgeführt werden.

Kontraststärkung

Die Klasse *HSBBrightnessEnhancement* streut Helligkeitswerte im Effektbild entsprechend Gradienten im Originalbild. Die private Methode **improveBrightnessData**(*float[][], float[][]*) iteriert zu diesem Zweck über die HSB-Helligkeitswerte aller Pixel. Im Zuge dessen wird der

Gradient und aus ihm die Modifikation des Farbwerts berechnet (siehe Kapitel 3.1.4). Diese Funktionalität ist durch **getEnhancedBrightnessImage**(*BufferedImage, BufferedImage*) gegeben und kann mit zwei Parametern beeinflusst werden.

- **setBrightnessRange**(*int*)
- **setGradientThreshold**(*float*)

ColorReduction Videoeffekt

Eine Instanz des *ColorReductionVideoEffects* verfolgt die durch Verarbeitung einzelner Frames eines Videos entstandenen Farben (siehe Abschnitt 3.2.1). Einander entsprechende Farbpaare werden dabei mit Hilfe der Klasse *DistanceMatrix* ermittelt und anschließend gemittelt. Wird die Farbreduktion nicht auf jedes Frame, sondern nur auf regelmäßig auftretende Endframes angewandt, generiert die Klasse *ColorReduction* Pixellabel für aktuelle Frames. Der Videoeffekt selbst ist der Bildeffektklasse *ColorReductionEffect* zugeordnet und ändert deren Bild, Parameter, Farb- und Labelarrays aufgrund der aktuellen Position in der Videosequenz.

MultiCluster Videoeffekt

Der *MultiClusterVideoEffect* erweitert ebenfalls den *ColorReductionEffect*. Anstelle des aktuellen Frames wird ein Bild, welches auch Nachbarframes beinhalten kann, konstruiert und farbreduziert. Das entstandene Labelarray und die dazugehörigen Pixelfarben werden mit der aktuellen Instanz der Parameterklasse im Bildeffekt gesetzt, um dort weitere Stilisierungen vorzunehmen.

ColorReductionFrameDifferencing Videoeffekt

Der in der Klasse *ColorReductionFrameDifferencingVideoEffect* implementierte und in Kapitel 3.2.3 beschriebene Videoeffekt führt eine Entscheidungsfunktion basierend auf Differenzen adjazenter Frames ein, um auf Pixelebene Flimmern im Ergebnis zu verringern. Die Funktionalität für die Differenzbildung zwischen (bereits abstrahierten) benachbarten Frames und Schwellenwertverfahren konnte dabei aus dem Package *util* übernommen werden.

Je nach Einstellung wird in der Implementierung von **process**() eine von zwei unterschiedlichen Hauptschleifen gewählt. Eine davon realisiert die Frame-zu-Frame Anwendung des Bildeffekts mit anschließender Differenzbildung. Die Zweite Hauptschleife kombiniert diese mit dem Ergebnis des *ColorReduction Videoeffekts*.

FlowBasedColorReduction Videoeffekt

Der in *FlowBasedColorReductionVideoEffect* umgesetzte Videoeffekt berechnet mit Hilfe des *BlockMatchingAlgorithm* aus dem Package *opticalflow* ein vertikales und ein horizontales Vektorfeld. Die Bewegungsdaten beeinflussen weiters die spezielle Farbreduktion *FlowClusterColorReduction*. Durch das Setzen von Labelarray, Farben und aktuellen Parameterwerten im Bildeffekt können weitere Abstraktionen durchgeführt werden.

LabColorReduction Effekte

LabColorReductionEffect und *LabColorReductionVideoEffect* implementieren weitgehend die Schritte der Video Abstraktion [93]. Die Stilisierung wird dabei immer vom Bildeffekt ausgeführt, da keine Zwischenschritte zur Erhaltung zeitlicher Kohärenz vorgesehen sind. Der Videoeffekt kümmert sich lediglich um das Zeitmanagement, sodass der Bildeffekt zu jedem Zeitpunkt mit den richtigen Parametern ausgeführt wird.

In der Hauptschleife werden L*a*b* Pixeldaten gewonnen (*PixelData*). Anschließend wird auf jeden Farbkanal ein *util.image.BilateralFilter* angewandt. Die Helligkeitswerte werden von der Klasse *LabColorReduction* sowie gegebenenfalls dem *util.image.DogEdgeDetector* bearbeitet. Die Implementierung des Bilateralen Filters entspricht nicht der von Winnemöller et. al. [93] gewählten Approximation, sondern der Implementierung entsprechend Tomasi und Manduchi [84].

Optional kann ein, an das Vorhandensein der Kanten gebundenes, Image Based Warping durchgeführt werden. Der Bildeffekt realisiert dieses durch einen *util.image.DisplacementFilter*.

5.3.2 Basisfunktionen der Bildbearbeitung

Das Package *util* enthält (statische) Basisfunktionen der Bildbearbeitung, dazu gehören unterschiedliche Klassen wie die Implementierungen des Interfaces *IColorHistogram*. Darüber hinaus gibt es noch eine Reihe an Subpackages mit weiteren hilfreichen Klassen. Das Package *colors* enthält Implementierungen für verschiedene Farbmodifikationen wie etwa das Verschieben der Farbtöne oder das Modifizieren der Sättigung. In *image* findet man Klassen, die Bildtransformationen und Filter realisieren. Das Subpackage *statistics* kapselt statistische Routinen.

Histogramme

Eine wichtige Klasse im Package *util* ist das Interface *IColorHistogram*, welches Schnittstellen für die häufigsten in einem spezifizierten Bild vorkommenden Farben für die derzeit drei existierenden Histogrammklassen deklariert (siehe Abbildung 5.4).

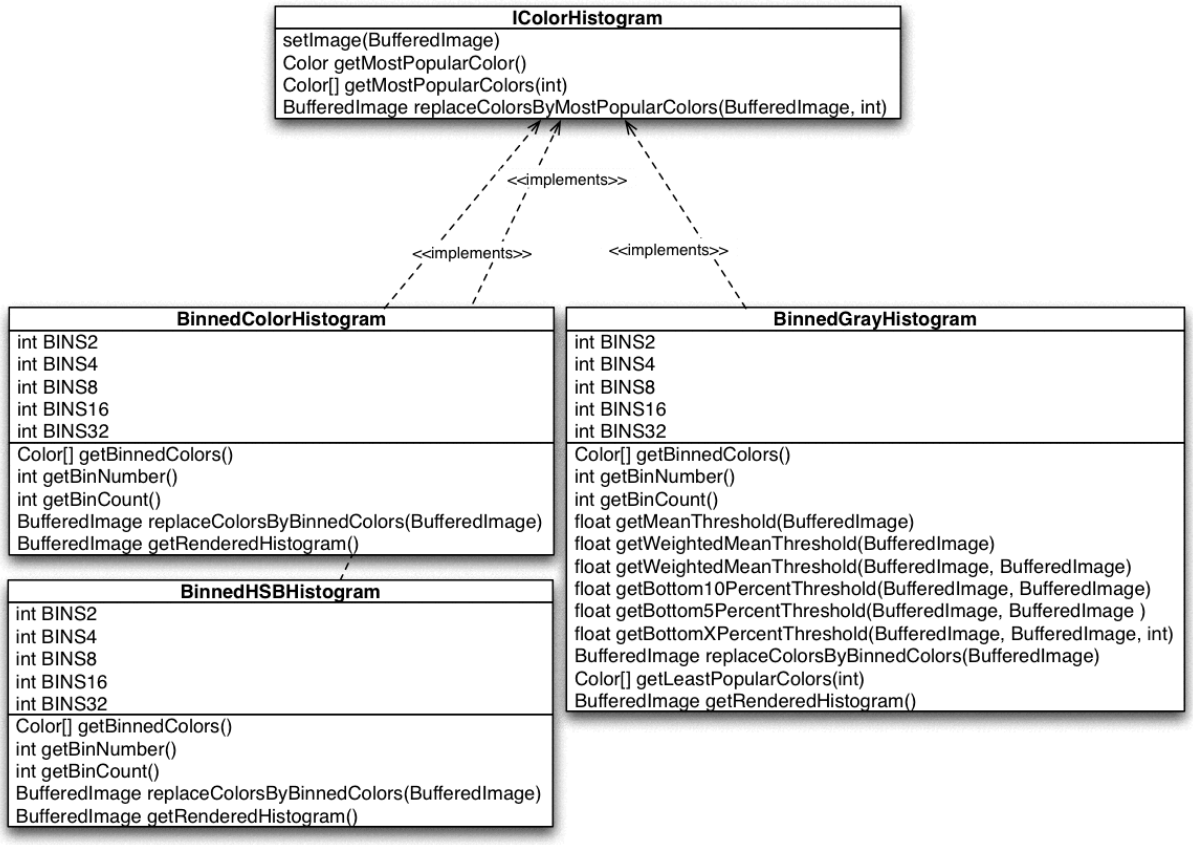

Abbildung 5.4: Im Package *util* gibt es drei verschiedene Histogramme. Sie sind Implementierungen vom Interface *IColorHistogram* und verfügen zusätzlich zu den vom Interface vorgeschriebenen noch über weitere Methoden.

Die Klasse *BinnedColorHistogram* setzt das Interface als Farbhistogramm um. Standardmäßig wird jedes Farbband (Rot, Grün und Blau) in acht gleich große Intervalle unterteilt, um den Arbeits- und Speicheraufwand zu reduzieren. Darüber hinaus gibt es weitere statische Konstanten, welche das Setzen davon unterschiedlicher Werte ermöglichen (siehe Abbildung 5.4). Der eigentliche Zählvorgang ist durch eine Schleife über alle Pixel realisiert. Sie speichert für jede erlaubte Farbkombination eine Bin-ID, den repräsentativen Mittelpunkt und die Häufigkeit global. Damit bietet sie die Grundlage für weitere Funktionen, wie diverse **get-** und **set-**Methoden. Mit **replaceColorsByBinnedColors**(*BufferedImage*) retourniert sie ein farbreduziertes Bild. Eine Instanz des Farbhistogramms kann mit **getRenderedHistogram**() durch farbige Linien mit Binhäufigkeit variierender Länge visualisiert werden. Analog zu diesem RGB Histogramm steht eine HSB Entsprechung, das *BinnedHSBHistogramm*, zur Verfügung.

Die dritte Implementierung, *BinnedGrayHistogram*, stellt die Verteilung der Grauwerte eines Bilds dar und implementiert die selben Methoden wie das Farbhistogramm. Darüber hinaus realisiert sie einfache Berechnungen über die Histogrammdaten, welche zur Analyse der

Verteilung dienen und Schwellenwerte (z.B. für Kantenerkennung) ermitteln:

getMeanThreshold ermittelt den Mittelpunkt des übrigen Grauwertintervalls.

getWeightedMeanThreshold berechnet das gewichtete arithmetische Mittel.

getMedianThreshold benutzt den Median.

getBottomXPercentThreshold und ähnliche Methoden ermitteln die unteren x Prozent im Histogramm. Die Mitte des Grenzbins wird dabei als Schwellenwert herangezogen.

getLeastPopularColors ermittelt die Intensitäten, die kaum oder gar nicht im Histogramm vorkommen.

Bilateraler Filter

Die Klasse *BilateralerFilter* implementiert den in [84] beschriebenen Bilateralen Filter und basiert auf dem von [13] zur Verfügung gestellten Code. Sie bietet überladene Routinen für die Anwendung auf Bilder (RGB, L*a*b*) und Datenarrays an, welche über spezifizierte Daten iterieren, um die benötigten Berechnungen durchzuführen (siehe Kapitel 3.3.1). Der Grad der Abstraktion kann mit **setSigmas**(*double*, *double*), einer Methode zum Setzen der Größe zweier global gespeicherter Filterkerne, beeinflusst werden.

DoG Kantenerkennung

Die Implementierung des in [93] beschriebenen DoG Operators kann, aufgrund der Differenz zweier mittels Gauß Filter berechneter Werte, Kanten erkennen. Dazu müssen die Filtergröße (**setFilterSize**(*int*)) und ein Parameter für die Genauigkeit (**setSharpness**(*float*)) gesetzt werden. Neben den **get**-Methoden für diese Parameter gibt es auch mehrere Methoden für die Berechnung der Kanteninformation von unterschiedlichen Grunddaten (*BufferedImage*, Datenarray). Nach der Initialisierung zweier Gauß Kerne faltet ein *DoGEdgeDetector*-Objekt in zwei geschachtelten Schleifen über die spezifizierten Daten und berechnet so punktweise Kanten.

Relieffilter

Die Klasse *EmbosFilter* beinhaltet zwei statische Methoden für das Erstellen von Reliefbildern und öffentliche statische Konstanten (*EDGE_DIR_HOR, EDGE_DIR_VER, EDGE_DIR_BOTH*), welche als Richtungsparameter verwendet werden können. Wie einige andere Filter (z.B. die DoG Kantenerkennung) arbeitet sich auch dieser mit zwei geschachtelten Schleifen durch ein *BufferedImage* und führt dabei Nachbarschaftsberechnungen aus.

Verschiebungsfilter

Eine Instanz von *DisplacementFilter* verschiebt Pixel entsprechend zumindest einer, mit *BufferedImage* spezifizierten, Verschiebungskarte im Zuge zweier Schleifen. Durch verschiedene **set**-Methoden kann die Berechnung des Verschiebungskoeffizienten beeinflusst werden.

setScaleFactor (*float*) spezifiziert einen Faktor, der mit der Verschiebung (bestimmt durch die Verschiebungskarten) multipliziert wird.

setMapHandling (*int*) setzt ein Flag, welches bei nicht übereinstimmender Dimension von Bild und Verschiebungskarte entscheidet, welches der beiden skaliert werden soll.

setBackHandling (*int*) definiert, ob das Ergebnis über das Originalbild oder in ein neues Bild gezeichnet werden soll.

Weitere Methoden (**getSharpenedImage**(...)) kombinieren diese Funktionalität mit jenen der Klasse *EmbosFilter*. Sie erstellen mit dem Relieffilter eine Verschiebungskarte aus einem spezifizierten Kantenbild und wenden anschließend den Verschiebungsfilter an.

5.3.3 Statistische Klassen

Das Subpackage *util.statistics* enthält die Implementierung des k-Mean Algorithmus aus [67]. Dazu gehören Datenstrukturen zur Verarbeitung der Clusterzentren (*ClusterCenters*) und der Daten (*Data*, *DataDimension*) sowie eine weitere Klasse für die Anwendung des Algorithmus (*KMeans*).

Das Interface *IInitialClusterCenterCreator* deklariert Heuristiken für die Wahl der Startzentren. Neben der Klasse *RandomDataPoints*, welche zufällige Werte aus den Datenelementen wählt, sind im Zuge dieser Diplomarbeit *MostPopularDataPoints*, *MostPopularHSBPoints*, *MostPopularRGBDataPoints* und *MostPopularAndRandomDataPoints* hinzugekommen. Diese Klassen nutzen alle ein *BinnedColorHistogram*-Objekt, um die Startzentren basierend auf den am häufigsten vorkommenden Datenpunkten zu ermitteln. Letztere Heuristik weist den Farbdaten, abhängig von der Anzahl der Farben, zusätzlich pseudo-zufällige *float* Werte zu.

5.3.4 Benutzeroberfläche

Die Funktionalität und die Definition der Benutzeroberfläche werden großteils im Package *gui* geregelt. Es beinhaltet *Frames*, *Panels* und *Handler* für die Bild- und Videoeffektapplikation, welche die Effekt-bezogenen *Panels* aus den Subpackages *imageeffectpanels* und *videoeffectpanels* einbinden. Das Subpackage *gui.util* beinhaltet dabei die Standard GUI-Elemente, wie

Implementierungen des *AbstractInterpolationPanels* oder des *AbstractTimePanels*, welche für die Videoapplikation von den Parameterklassen (siehe Kapitel 5.2.4) initialisiert werden.

Ein *Panel* für die Wahl der Eigenschaften eines Effekts kann GUI-Elemente und ihre *ActionListener* von Parameterklassen verwenden. Diese Aggregation erlaubt die von BenutzerInnen veränderten Eigenschaften direkt in die Parameterklasse zu übernehmen. Zusätzlich bestimmt ein Effektpanel mit **get**-Methoden die Position des Effekts im Menü und einen Bild- oder Videoeffekt, dem vor seiner Ausführung als Schnittstelle eine Spezialisierung des *Effekt-Parameter*-Objekts angefügt wird.

5.4 Geschwindigkeitstests

Dieser Abschnitt beinhaltet eine umfassende Betrachtung der Ergebnisse von empirischen Laufzeiten der wichtigsten Abstraktionsschritte der implementierten Effekte. Die Zeitangaben wurden auf einem PowerBookG4 mit PowerPC G4, einer CPU-Geschwindigkeit von 1.33 GHz und 768 MB Arbeitsspeicher ermittelt und sind durchgehend in Millisekunden gegeben.

5.4.1 Farbreduktion

Dieser Abschnitt beschäftigt sich mit Statistiken zum einzigen obligatorischen Schritt der *Color-Reduction Effekte*, der Farbreduktion. Die präsentierten Zahlen beziehen sich nicht nur auf die reine Verarbeitungszeit der Pixelinformation, sondern auch auf das Erstellen von LUT und Label sowie das Zeichnen des Bilds. Für die Geschwindigkeitstests wurde ein Testbild der Größe 344 x 470 zehn mal von der jeweiligen Methode verarbeitet und neben der Dauer einer Iteration auch Durchschnitt (Tabelle 5.1), Median (Tabelle 5.2), Varianz (Tabelle 5.5) und Standardabweichung (Tabelle 5.4) aller Durchläufe ermittelt. Das Clustering wurde während der Messungen mit einem Standardwert von drei Iterationen angewandt.

Tabelle 5.1: Folgende Tabelle zeigt die durchschnittlichen Laufzeiten der verschiedenen Methoden der Farbreduktion. Die Werte sind in Millisekunden angegeben.

Farbanzahl	8	48	100	150	200	250
Simple Farbredukt.	432	574,40	517	488	545	652
HistogrammBinning	633	998	1445	1870	2267	2690
Clustering	3125	15021	36520	56133	79535	101923

Die Mittelwerte und Mediane veranschaulichen, wie rechenintensiv die Farbreduktionen sind, und führen die Unterschiede der Laufzeiten vor Augen. Schon das Verhältnis zwischen Simpler Farbreduktion und Histogramm Binning beträgt ungefähr 2:1 und wird mit steigender

Tabelle 5.2: Diese Tabelle beinhaltet die Mediane der Geschwindigkeitstests. Die Werte sind in Millisekunden angegeben.

Farbanzahl	8	48	100	150	200	250
Simple Farbredukt.	341	394	453	417	483	564
HistogrammBinning	579	946,00	1394	1823	2202	2605
Clustering	3028	14841	35757	55864	79333	101732

Farbanzahl größer. Das Gefälle zwischen Histogramm Binning und Clustering ist noch deutlicher zu erkennen, so schlägt es sich durchschnittlich mit einem Faktor von 25 nieder. Auch hier wird der Unterschied mit wachsender Farbanzahl größer und ist bereits beim Schritt von acht zu 48 Farben verdreifacht.

Die hohen Zahlen beim Clustering werden durch die Kombination des Histogramm Binnings mit dem ebenfalls rechenintensiven Clustervorgang (siehe Kapitel 3.1.1) hervorgerufen. Die Startzentren sind die im Bild am häufigsten vorkommenden Farben. Bei einer zufälligen Wahl aus der Datenmenge würde diese Methode schneller laufen (siehe Tabelle 5.3), dies würde sich aber negativ auf die Qualität der Ergebnisse auswirken.

Tabelle 5.3: Die Farbreduktion mittels Clustering hat bei zufälliger Wahl der Startzentren eine geringere Laufzeit als bei der Wahl mittels Histogramm Binning. Diese Tabelle zeigt Statistiken dazu. Als Testbild wurde dasselbe Bild wie bei den anderen Tests gewählt. Das Bild wurde auf acht Farben reduziert. Die Zahlen sind dabei in Millisekunden angegeben.

/	Mittelwert	Median	Varianz	Standardabweichung
Random Clustering	1409	1366	56707	238

Während die Simple Farbreduktion für eine variierende Farbanzahl relativ gleichmäßige Laufzeiten von einer halben Sekunde aufweist (siehe Tabelle 5.1 und 5.2), steigen die der beiden anderen Methoden stärker an. Das Clustering beginnt bei drei Sekunden und kann bei der Wahl von 250 Farben einen Maximalwert von 1,7 Minuten pro Bild erreichen. Damit ist es nicht nur die langsamste der drei Methoden, sondern bezüglich der Verarbeitungsgeschwindigkeit auch sehr von ihren Parametern abhängig.

Die Standardabweichungen (siehe Tabelle 5.4) und das Streuungsmaß (siehe Tabelle 5.5) zeigen, dass die Laufzeiten bei zehn Iterationen schwanken können. Besonders beim Clustern weichen die Werte deutlich vom angegebenen Mittelwert ab, so entspricht die Differenz zwischen maximaler und minimaler Laufzeit bei 250 Farben 9538 Millisekunden.

Die Verarbeitungszeit ist nicht nur von den gewählten Parametern, sondern auch von der Wahl des zu verarbeitenden Bilds abhängig. Je größer das Bild und damit die Datenmenge ist,

Tabelle 5.4: Folgende Tabelle fasst die Standardabweichungen der einzelnen Methoden zusammen. Die Zahlen sind dabei in Millisekunden angegeben.

Farbanzahl	8	48	100	150	200	250
Simple Farbreduktion	322	642	219	227	221	281
Histogramm Binning	150	168	194	175	208	215
Clustering	361	819	2338	1262	924	3763

Tabelle 5.5: Diese Tabelle enthält die gemessenen Varianzen von Simpler Farbreduktion (SFR), Histogramm Binning (HB) und Clustering (C) bei unterschiedlichen Farbanzahlen. Die Werte sind dabei in Millisekunden gegeben.

Farben	8	48	100	150	200	250
SFR	103383	412392	47809	51617	49061	78839
HB	22532	28340	37740	30576	43246	46422
C	130421	671100	5463989	1593444	853788	14159415

desto länger dauert die Farbreduktion (siehe Tabelle 5.6). Auch hier ist die Wachstumsrate des Clusterings am höchsten.

Tabelle 5.6: Diese Tabelle beinhaltet die mittleren Verarbeitungszeiten für fünf Bilder verschiedener Dimensionen. Sie alle wurden dabei auf acht Farben reduziert. Die Werte sind dabei in Millisekunden gegeben.

Dimensionen	204 x 308	344 x 470	614 x 461	900 x 675	1548 x 1161
Simple Farbreduktion	305	432	574	985	2670
Histogramm Binning	291	633	1030	3324	5510
Clustering	1478	3125	5217	10799	31158

Der *LabColorReduction Bildeffekt* weist mit gleich vielen Bins bei der Helligkeitsreduktion wegen des rechenaufwendigen bilateralen Filters längere Laufzeiten auf (siehe Tabelle 5.7). Aufgrund dieser Tatsache wurde die mehrmalige Anwendung des Filters vom obligatorischen zum optionalen Schritt refaktorisiert. Wird gänzlich auf das Filtern verzichtet, so sind die Laufzeiten bei bis zu acht Lightness Bins mit knapp unter drei Sekunden um etwa eine halbe Sekunde schneller als bei der Farbreduktion mittels Clustering. Bei vorhergehendem Filtern ist die Laufzeit schon mit niedrigeren Parametern etwas höher, bleibt aber bei den Extremwerten von 32 Bins und einem Filterkern vom Radius 20 ($\sigma_d = 9.25, \sigma_r = 10.00$) unter dem Ergebnis des Clusterings. Es ist zu beachten, dass sich die präsentierten Werte nur auf die bei dieser Diplomarbeit entstandene Implementierung beziehen. Die auf der GPU laufende Video

Abstraktion [93] kann in Echtzeit durchgeführt werden.

Tabelle 5.7: Die Laufzeiten des *LabColorReduction Bildeffekts* wurden mit einem Testbild der Größe 344 x 470 ermittelt. Es wird sowohl auf Konturen als auch auf nachträgliches Scharfzeichnen verzichtet. Die Werte sind in Millisekunden gegeben.

Filter	Mittel	Median	Varianz	Standardabweichung
8 Lightness Bins				
$\sigma_d = 9.25, \sigma_r = 10.00$	59915	58186	22250096	4717
$\sigma_d = 5.50, \sigma_r = 7.00$	29366	29337	121108	348
$\sigma_d = 3.00, \sigma_r = 4.25$	13761	13550	247151	497
$\sigma_d = 2.02, \sigma_r = 2.25$	6209	6148	46250	215
kein Bilateraler Filter	2789	2721	84222	290
32 Lightness Bins				
$\sigma_d = 9.25, \sigma_r = 10.00$	59356	57472	48532694	6967
$\sigma_d = 5.50, \sigma_r = 7.00$	30453	30037	2645613	1627
$\sigma_d = 3.00, \sigma_r = 4.25$	14442	14415	502451	709
$\sigma_d = 2.02, \sigma_r = 2.25$	6146	6047	58817	243
kein Bilateraler Filter	2497	2438	16673	129

5.4.2 Kantenerkennung

In diesem Abschnitt wird kurz auf die Laufzeiten der Gewinnung und Stilisierung von Detailzeichnungen in Form von Linien eingegangen. Der Verarbeitungsprozess bestehend aus Erkennen, Histogrammanalyse, Stilisierung und Zeichnen der Kanten in den *ColorReduction Effekten* variiert nicht mit seinen Parametern, sondern bleibt diesbezüglich relativ konstant. Die Laufzeiten dieses Vorgangs hängen aber von der Dimension des Eingangsbilds ab (siehe Tabelle 5.8).

Tabelle 5.8: Diese Tabelle fasst die Laufzeiten der Kantenerkennung, wie sie in *ColorReduction Effekten* verwendet wird, zusammen. Der Geschwindigkeitstest wurde jeweils auf Bildern verschiedener Größe durchgeführt. Die Werte sind dabei in Millisekunden gegeben.

Mittelwert	Median	Varianz	Standardabweichung
204 x 308			
635	549	119696	346
344 x 470			
1367	1258	102306	320

Die Verarbeitungsdauer der Anwendung des modifizierten DoG Filters aus [93], in welchem das Zeichnen der Kanten nicht Teil der Arbeitsschritte ist, entspricht bei Wahl einer durch-

schnittlichen Filtergröße (siehe Tabelle 5.9, Filtergröße acht) den Laufzeiten der Kantenerkennung der *ColorReduction Effekte* (siehe Tabelle 5.8). Kleine Filtergrößen, wie beispielsweise zwei, sind weniger rechenintensiv und bewirken eine deutlich schnellere Verarbeitung.

Tabelle 5.9: Diese Tabelle beinhaltet die empirisch ermittelten Laufzeiten der Kantenerkennung mittels DoG-Operator und der anschließenden Stilisierung nach [93] für unterschiedliche Bild- und Filtergrößen. Die Werte sind dabei in Millisekunden gegeben.

Filtergröße	Mittelwert	Median	Varianz	Standardabweichung
204 x 308				
8	594	543	13564	116
2	192	179	517	23
344 x 470				
8	1498	1410	46222	215
2	478	470	5864	77

Kapitel 6

Einsatz in der Lehre

In diesem Kapitel soll zunächst eine kurze Einordnung der thematisierten Lehrveranstaltung erfolgen. Lehrinhalte, Zielgruppen und strukturelle Aspekte des Lehrprozesses werden geklärt (siehe Abschnitt 6.1). Der Hauptteil des Kapitels (siehe Abschnitt 6.2) beschäftigt sich mit den Rahmenbedingungen, die durch den Einsatz der *ColorReduction Effekte* in der Übung entstehen könnten, und ob unter ihnen die in der Gesamtkonzeption intendierten Ziele umsetzbar sind. Die Evaluierung erfolgt dabei aufgrund der in [90] und [36] präsentierten Kriterienkataloge. Für eine praxisbezogene Beurteilung bedarf es darüber hinaus dem Einsatz von sozialwissenschaftlichen Instrumenten wie Beobachtung oder Fragebögen [36], um allgemeine Probleme von Kriterienkatalogen wie Unvollständigkeit [87] zu umgehen.

6.1 Fachdidaktische Aspekte

6.1.1 Einordnung und Lehrinhalt

Die Laborübung Videoverarbeitung ist für HörerInnen verschiedener Magisterstudien der Informatik mit grundlegenden Bildverarbeitungs- und Programmierkenntnissen konzipiert und formuliert ihre Lehrinhalte und Ziele dahingehend. Die dazugehörige Vorlesung vermittelt theoretisches Wissen zur Bild- und Videoverarbeitung, das in der Laborübung praktisch vertieft werden kann. In der Übung sollen Probleme der Videoverarbeitung selbstständig identifiziert und Strategien zur Vermeidung dieser entwickelt werden.

6.1.2 Strukturierung des Lehrprozesses

Die Laborübung wird in Kleingruppen absolviert und strukturiert sich in drei Teile, die den Phasen des Problemlösens (Problemstellung und Problemanalyse - Lösungskonzept und Ausführung

des Konzepts - Reflexion und Evaluation [38]) entsprechen. Die VEL agiert dabei als *tutee* („Programm in lernender Position“ [36]). Im ersten Übungsteil, der Problemstellung und Problemanalyse, wird ein bereits in diese Library implementierter Bildeffekt gewählt, anhand anleitender Fragen analysiert und ein neuer Parameter für ihn konzipiert. Dabei wird auch auf die Problematik seiner Frame-zu-Frame Anwendung auf eine Videosequenz und Einsatzmöglichkeiten des Optical Flows eingegangen (siehe Kapitel 2). Im Zuge dessen setzen sich StudentInnen mit dem Framework auseinander und machen sich mit, für spätere Phasen grundlegenden, Funktionen vertraut. Aufbauend auf dem gewählten Bildeffekt soll, im zweiten Teil, ein Videoeffekt, welcher über eine Frame-zu-Frame Anwendung des Bildeffekts hinausgeht, konzipiert und in weiterer Folge implementiert werden. Abgesehen von einem beliebigen Parameter können die übrigen Variablen für das gesamte Video als unverändert angenommen werden. Die Analyse- und die Konzeptionsphase sollen dabei den Fachinhalt motivieren. Während der Implementierung werden Hilfestellungen in Form von Treffen, E-Mail Kontakt, betreuten Diskussionsforen und Dokumentationen des bereits vorhandenen Codes angeboten. Als Abschluss der Lehrveranstaltung präsentiert jede Gruppe ihre Ergebnisse, wodurch ein Austausch ermöglicht wird. Die Beurteilung erfolgt aufgrund der Ergebnisse der einzelnen Teile und den darauf folgenden Abgabegesprächen.

Durch diese Strukturierung werden verschiedene didaktische Vorgehensweisen zur Unterrichtsgestaltung kombiniert. Mit dem Hinweisen auf allgemeine Probleme durch konkrete Beispiele, nämlich den Bildeffekten und ihrer Frame-zu-Frame Anwendung bzw. dem Herstellen einer raum-zeitlichen Beziehung zwischen Frames als Problemlösung, wird eine induktive Vorgehensweise gewählt. Gleichzeitig wird auch deduktiv vorgegangen, indem aus der Vorlesung bekannte Techniken bei der Implementierung eines Videoeffekts konkretisiert werden können. Es wird die Möglichkeit offen gelassen, dass sich StudentInnen an bereits implementierten Effekten orientieren können und somit den Umgang mit der VEL imitierend lernen. Die Klassen dieser Effekte dienen als Vorbild für das Umsetzen der eigenen Konzepte. Beim Implementieren kann das Lernen durch Versuch und Irrtum eine Rolle spielen. [38]

6.2 Beurteilung des Einsatzes in der Lehre

In diesem Abschnitt soll evaluiert werden, ob der Einsatz des *ColorReduction Bildeffekts* als weiterer in der Übung zur Auswahl stehender Effekt denkbar ist. Der Bildeffekt und die davon abgeleiteten Videoeffekte sollen bei der problemorientierten Auseinandersetzung mit dem Thema Videoverarbeitung und im Speziellen der Transformation von Bild- zu Videoeffekt verwendet werden. Im Gegensatz zur rein technischen Evaluierung einer Applikation oder einer Library gibt es für die didaktische Beurteilung keine allgemein anerkannten Kriterien. Das Problem dabei sind der variierende Kontext und der Verwendungszweck, in dem diese durchgeführt

werden können [87]. Trotz dieser Schwierigkeiten wagen Stamelos et al. in [90] den Versuch, eine Reihe an Kriterien zu formulieren, welche jedoch bei der Evaluierung dem spezifischen Kontext entsprechend angepasst werden müssen (siehe Abbildung 6.1). In diesem Abschnitt wird versucht, diese Attribute aufzugreifen und die zur Auswahl stehenden Effekte dementsprechend zu untersuchen. Die Evaluierung der didaktischen Aspekte erfolgt dabei auf

- der didaktisch-technischen Ebene und
- auf der inhaltsbezogenen Ebene.

Diese Unterteilung, die beispielsweise auch in [36] zu finden ist, bezieht sich oft auf die Unterscheidung zwischen dem eingespeisten Inhalt und einem E-Learning oder Multimedia System selbst sowie dessen Umgang mit der bereitgestellten Information.

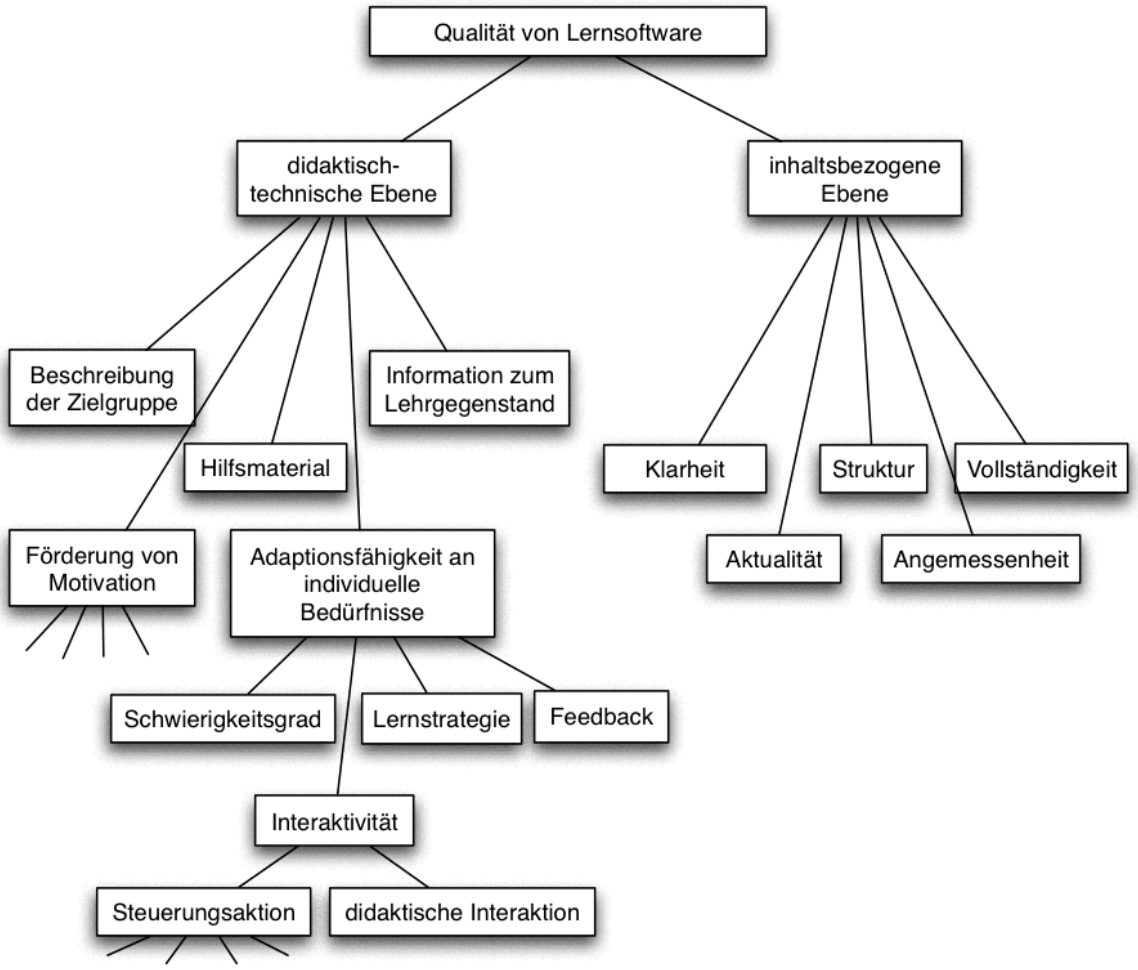

Abbildung 6.1: Dieser Baum visualisiert hierarchisch die für die Betrachtung der Videoeffekte herangezogenen Attribute.

Didaktisch-technische Ebene

Die erste Ebene betrachtet Eigenschaften wie die Angemessenheit des Evaluationsgegenstands für einen bestimmten Zweck. Da die VEL nicht ausschließlich als Unterrichtsmaterial der Übung Videoverarbeitung konzipiert wurde und im Allgemeinen nicht als Lernsoftware, sondern Library für Bild- und Videoeffekte zu sehen ist, sind didaktische Definitionen der Zielgruppe nicht Teil der Programmbeschreibung. Rein für die Anwendung der VEL-eigenen Applikationen zur Bearbeitung von Bild- oder Videomaterial ist kein besonderes Vorwissen notwendig. Wird jedoch auf der Entwicklungsebene gearbeitet, sind zumindest grundlegende Programmier- und Bildverarbeitungskenntnisse nötig. Voraussetzungen, Lehrinhalte und -ziele ergeben sich durch die in der Übung geforderten Aufgaben bzw. Modifikationen der Library. Sie werden allein von der Übungsleitung vorgegeben (siehe Abschnitt 6.1.1), so konzentriert sich auch der von der Applikation intendierte Themenbereich „Bild- und Videoverarbeitung“ in der Übung auf die „zeitlich kohärente Videoverarbeitung“ (mit und ohne Einsatz des Optical Flows). Dieses globale Themengebiet kann von StudentInnen durch die Wahl eines Bildeffekts (siehe Abbildung 6.2) genauer spezifiziert werden, so liegt der Schwerpunkt der *ColorReduction Effekte* bei der Erstellung zeitlich kohärenter Abstraktionen und der des *PainterlyRendering* bei der pinselstrichbasierten Verarbeitung. Das Hauptaugenmerk beim *Puzzle Bildeffekt* ist die Generierung und Verarbeitung von Puzzleteilen, während der *Alchemy Bildeffekt* mit einer gewählten Anzahl an Texturen das Originalbild modifiziert.

Abbildung 6.2: Diese Abbildung bietet eine Übersicht zu den in der Übung Videoverarbeitung verwendeten Effekten, in folgender Reihenfolge (von links nach rechts): *Alchemy Effekt*, *Painterly Rendering*, *Puzzle Effekt* und *ColorReduction Effekt*. Bei der Anwendung des *Alchemy Bildeffekt* auf eine Bildsequenz, erzeugen die fix positionierten Texturen Artefakte. *Painterly Rendering* (siehe Kapitel 2.2.2) sowie der *Puzzle Bildeffekt* flackern bei einer Frame-zu-Frame Anwendung. Die pro Bild zufällig erzeugten Puzzle Teile liefern ein nicht kohärentes Ergebnis. Die sequenzielle Anwendung des *ColorReduction Bildeffekts* auf ein Video resultiert in Farbsprüngen und Flackern im Ergebnis (siehe Kapitel 3).

Der Hauptunterschied zwischen den drei bereits in der Übung eingesetzten Bildeffekten zu jenem dieser Diplomarbeit ist die Definition von Regionen. Während sie auf Effektprimitive (Textur, Puzzleteil, Pinselstrich) an verschiedenen Positionen zugreifen (und sie beispielsweise wie in [52] oder [34], siehe Kapitel 2, verarbeiten), ist dies beim *ColorReduction Bildeffekt* nur

mit Mehraufwand möglich. Die Erweiterung des Effekts dahingehend wäre jedoch durchaus interessant und als zukünftige Entwicklung denkbar. Ein weiterer wichtiger Unterschied zu den bereits eingesetzten Bildeffekten ist die Bildverarbeitungslastigkeit der *ColorReduction Effekte*. Mit Weichzeichnen- und Kantenoperatoren, Histogrammen, Farbmodellen sowie der Segmentierung (siehe Kapitel 3.1) wird der Zielgruppe mehr Wissen zu diesem Thema abverlangt als bei *Alchemy Bildeffekt*, *Painterly Rendering* oder *Puzzle Bildeffekt*, welches nicht zwingend zum Qualifikationsprofil von AbsolventInnen eines Bachelorstudiums der Informatik gehört [92].

Die Themengebiete werden in erster Linie visuell präsentiert, genauere Information zur Funktionsweise kann in der Dokumentation gelesen oder aus dem Programmcode abgeleitet werden (Problemstellung und Problemanalyse, siehe Abschnitt 6.1.2). Dokumentation und Programmcode dienen als Hilfestellung für HörerInnen sowie Lehrpersonal und enthalten Information zur generellen Funktionsweise und Kommentare zur Implementierung. So können beide Parteien die Library erweitern, adaptieren und die Bild- und Videoverarbeitungsalgorithmen schneller nachvollziehen. Eine „Hilfe“ in Bild- und Videoapplikation selbst gibt es derzeit nicht. Die bereits implementierten Videoeffekte können als Art strategische Hilfestellungen zum Lösen der Aufgaben betrachtet werden. Nicht vom Programm, aber im Zuge der Übung, werden zusätzlich ein Diskussionsforum und Treffen angeboten, um Fragen bezüglich der VEL und ihrer Effekte zu klären.

Ein drittes Kriterium für die Qualität von in der Lehre eingesetzter Software ist ihre Adaptionsfähigkeit an individuelle Bedürfnisse, welche durch passendes Feedback, Unterstützung verschiedener Lernstrategien, Schwierigkeitsgrade und Interaktivität gekennzeichnet ist [90] [65]. Die VEL gibt einerseits Feedback durch Fehlermeldungen, andererseits auch in visueller Form. Die *ColorReduction Effekte* versuchen im User Interface fehlerhafte Eingaben vorweg auszuschließen und, sollte das nicht gelingen, einen Dialog mit aussagekräftiger Meldung anzuzeigen (siehe Abbildung 6.3). Beim Anpassen oder Erweitern der Library muss auf die Rückmeldungen der *Java Virtual Machine* zurückgegriffen werden, was für InformatikstudentInnen durchaus adäquat ist. Lauffähige Ergebnisse können visuell mit denen der Frame-zu-Frame Anwendung des erweiterten Bildeffekts verglichen werden, wodurch eine Selbstbeurteilung und Beobachtung der eigenen Fortschritte ermöglicht wird.

Handlungsorientiertes Lernen ist für Hochschul- [86] und insbesondere informatische Bildung [38] von großer Bedeutung. Handlungskompetenzen können erst durch Interaktivität, welche beim Lernenden Denk- und Handlungsprozesse aktiviert, erworben werden [86], weshalb sie eine wichtige Rolle in der didaktischen Evaluierung von multimedialem Arbeitsmaterial spielt. In diesem Zusammenhang unterscheiden Strzebkowski und Kleeberg in [88] zwischen

- Steuerungsaktion, welche sich auf die Navigation und Systemfunktionen der Lernsoftware bezieht und
- didaktischer Interaktion, welche beim Gewinnen von Erkenntnis unterstützen soll.

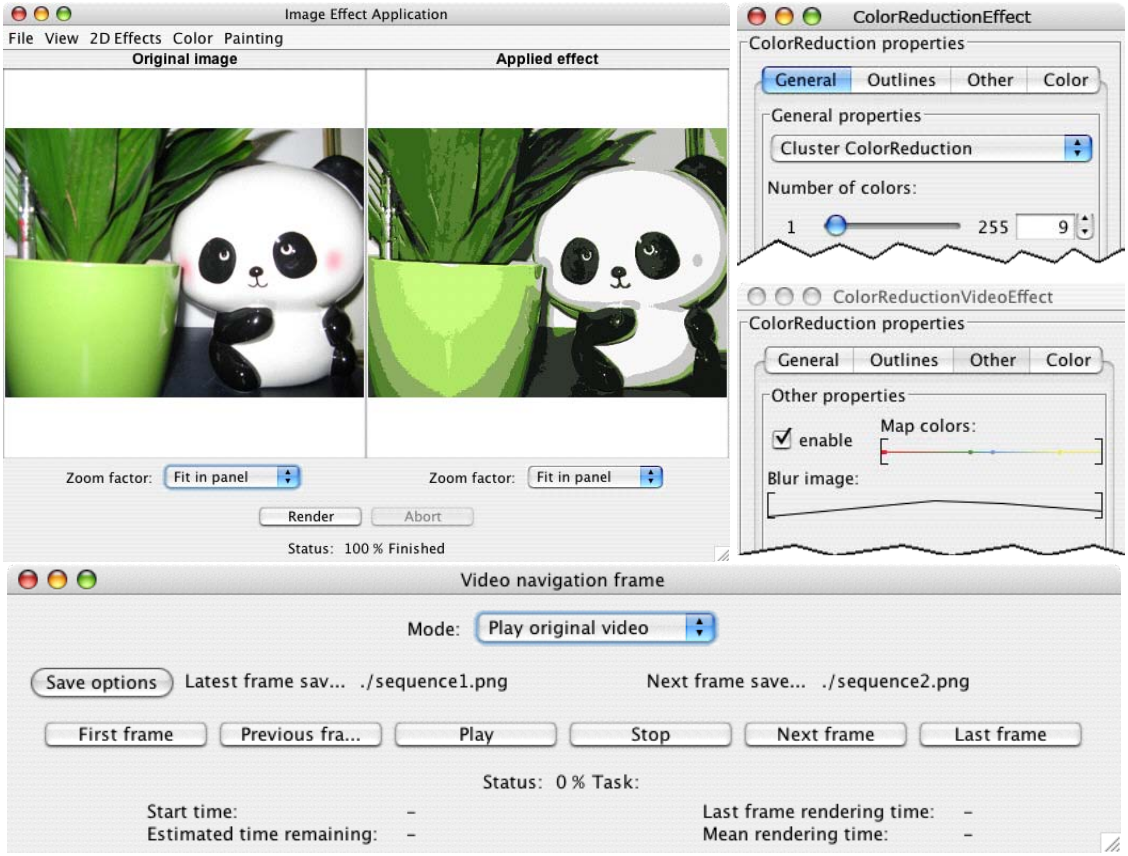

Abbildung 6.3: Diese Abbildung zeigt Screenshots verschiedener Komponenten des User Interfaces der aktuellen Implementierung der Bild- und Videoapplikation. Das Bild *oben, links* zeigt die Hauptkomponente mit Menü für die Auswahl des Effekts, Originalbild und Ergebnis sowie Start- und Abbruch-Button. Der Arbeitsbereich sieht für Bild- und Videoapplikation im Großen und Ganzen gleich aus und unterstützt *affordance* [36], er legt also mit seinem Design den Nutzen der Applikationen nahe. Die Komponenten im *rechten Screenshot* dienen der Bearbeitung der Parameter eines Effekts. In der Videoeffektapplikation ist diese durch dynamische Zeitleisten geprägt, welche per Klick einen weiteren Auswahldialog für Zeit und Parameter sichtbar machen. Außerdem kommt zusätzlich eine Navigationsleiste (*unteres Bild*) hinzu, welche neben grundlegenden Steuerungselementen für Videos auch eine Speicherfunktion bereitstellt.

Die Steuerungsaktion betreffend bieten die *ColorReduction Effekte* und das Framework die in Abbildung 6.3 zu sehenden User Interface Komponenten, welche unterschiedliche Elemente nach ergonomischen Gesichtspunkten gruppieren. Insgesamt ist die Gestaltung mit einem Werkzeugfenster (Steuerbereich [36]) und Bildfenster (Arbeitsbereich [36]) sowie einem Menü (Orientierungsbereich [36]) an verschiedene Bild- oder Videobearbeitungsapplikationen wie beispielsweise GIMP [81] angelehnt und legt somit den Mehrwert der Applikation und die Bedienbarkeit nahe (*affordance* [36]). Die Steuerungselemente sind auf einem Blick sichtbar und können zum Teil auch ausgeblendet werden (*visibility* [36]). Effekte und Bilder können gewählt und Ergebnisse sowie zeitbasierte Inhalte gesteuert werden. Einfache, aussagekräftige Beschrif-

tungen, sparsamer Einsatz von Farben, die Steuerung der Einflussstärke der Parameter und insgesamt ein konsistentes „*look and feel*“ erleichtern es BenutzerInnen, die einzelnen Eigenschaften des Effekts genauer zu erkunden. Die Definition von Standard-GUI-Elementen (siehe Kapitel 5.2.4) ermöglicht selbst beim Erstellen einer Werkzeugbox für den eigenen Videoeffekt automatisch einen gewissen Grad an Konsistenz beizubehalten. Auch wenn das User Interface noch benutzerfreundlicher gestaltet werden könnte (Ikonen, visuelle Metaphern, Tooltips), stellt seine aktuelle Implementierung eine deutliche Verbesserung zu früheren Versionen dar, in denen StudentInnen Parameter im Programmcode ändern mussten. Die gegebenenfalls längeren Laufzeiten der Effekte (siehe Kapitel 5.4) können unter Umständen die Interaktion behindern und in weiterer Folge die Motivation negativ beeinflussen. Eine vollständige Analyse der Gebrauchstauglichkeit (*usability*) nach der ISO Norm 9241 [21] liefert genauere Ergebnisse diesbezüglich [36], würde jedoch den Rahmen dieser Arbeit sprengen. Darüber hinaus wird die VEL in der Übung hauptsächlich als Library und nicht nur als Applikation verwendet.

Bezüglich der didaktischen Interaktion, welche nicht passive Rezeption sondern direkt den Erkenntnisprozess unterstützen soll, haben Library und Effekt noch mehr zu bieten. Durch den uneingeschränkten Zugriff zum Programmcode können vorhandene Daten modifiziert, neue erzeugt, visuelles Feedback verändert und komplexe Fragestellungen implementiert werden. Diese Freiheit lässt kreative und expressive Tätigkeiten zu und führt zu bedeutungsvollem, entdeckendem Lernen [88].

Bei der Implementierung des Videoeffekts im Rahmen des Bildeffekts impliziert die Adaptionsfähigkeit auch die Definition eines individuellen Schwierigkeitsgrads und keine Festlegung auf einen bestimmten Lerntyp, so können so genannte Konvergierer aktiv experimentieren, Akkommodierer intuitiv und nach Versuch und Irrtum Verfahren arbeiten, Assimilierer analytische Konzepte entwickeln und Divergierer aufbauend auf den visuellen Beobachtungen kreative Ansätze einbringen [44]. Die Handlungsorientierung unterstützt bei jedem der Bildeffekte aber vor allem Lerntypen, die praktischen Erfahrungsaufbau bevorzugen.

Die Motivation von StudentInnen kann auf verschiedene Weise beeinflusst werden. Sie wird durch das Definieren von klaren Zielen, Gruppenarbeiten und Präsentationen der finalen Videoeffekte, Abgabegespräche oder ähnliche Methoden von der Übungsleitung (extrinsisch) erhöht. Frustration wird durch weitere angebotene Hilfestellungen vermindert. Der Einsatz eines Frameworks in der Übung hat allgemein den Vorteil, dass bereits implementierte Methoden wiederverwendbar sind und damit Arbeitsaufwand gespart werden kann. So erlaubt es die VEL, Ideen für Bildeffekte oder Transformationen zu Videoeffekten auszuprobieren und dabei schnell zu visuellen Ergebnissen zu kommen, was durchaus motivationsfördernd wirken kann. Andere, zum Teil schon besprochene, Faktoren wie

- Neugierde und Explorationsdrang (aktive Problemlösung)
- Abwechslung (keine monotone Tätigkeit)

- Kreativität (Konzipieren und Implementieren eines eigenen Effekts)
- Feedback (visuelles Feedback liefert Antworten und fördert die Neugierde)
- Lernzielklarheit und -erreichbarkeit (Auswahl eigener Schwierigkeitsstufen, Hilfestellungen)
- zusätzliche Übungsmöglichkeit für interessierte Benutzer (z.B. eigene Bild- oder Videoeffekte implementieren)

sind ein wesentlicher Grad der Anregung. Sie deuten auf ein hohes Motivationspotential hin [36]. Die tatsächliche Messung von Motivationszuwachs wird üblicherweise mit Erhebungsinstrumenten, wie Fragebögen, vor und nach Einsatz des Hilfsmittels gemessen [36]. Auf diese Weise werden auch wichtige Kriterien wie etwa Erwartungshaltung und affektive Komponenten berücksichtigt.

Inhaltsbezogene Ebene

Die zweite Ebene beschäftigt sich mit inhaltsbezogenen Eigenschaften. Während E-Learning Umgebungen oder Lernplattformen Inhalt ganz klar in Form von Text darstellen, wird er in diesem Kontext als effekt- bzw. programmcodebezogener interpretiert [90]. Die Qualität des Inhalts ist aufgrund von Klarheit, Vollständigkeit und Aktualität zu beurteilen. Obwohl viele Bild- und Videoverarbeitungsalgorithmen in der VEL zusammengetragen wurden, erhebt sie keinerlei Anspruch auf Vollständigkeit. Es gibt unzählige Versuche, eine kohärente Transformation zum Videoeffekt zu modellieren (siehe Kapitel 2.2), welche als Basis oft speziell für ihren Bildeffekt entwickelt wurden. Eine Integration dieser Spezialfälle und ihrer teilweise sehr komplexen Algorithmen würde sowohl HörerInnen einer einstündigen Lehrveranstaltung zum Thema Videoverarbeitung als auch Lehrpersonal (organisatorisch) überfordern. Darüber hinaus soll, auch wenn Basisliteratur (z.B. Artikel von [34] und [52]) und bereits implementierte Hilfestellungen (Parameter Interpolation und Zeitmanagement, Effekt Primitive, Frame Differencing, Optical Flow Berechnung u.v.m.) bereits vorhanden sind, nicht vergessen werden, dass die selbstständige Entwicklung und Implementierung einer Transformation ein wichtiges Ziel der Übung ist.

Die Klarheit der Stoffpräsentation in der Implementierung wird nicht mittels Satzlänge, Ausdruck oder Ähnlichem [65], sondern durch Namenskonversionen, Kommentare sowie einem lesbaren Programmierstil erzielt. Die Implementierung der *ColorReduction Effekte* behält deshalb die bereits im Framework angewandten Programmierrichtlinien sowie Form der Kommentare (siehe folgenden Codeauschnitt) bei.

Listing 6.1: Beispiel für Kommentare im Code

```
1 //1. make some initial checks
```

```
2 if (m_pOrigImage == null) {
3         fireInterruptedEvent("No original image was set.");
4         return;
5 }
6
7 //2. fetch basic parameters and color values
8 int iWidth = m_pOrigImage.getWidth();
9 int iHeight = m_pOrigImage.getHeight();
10 int iColNum = m_pColorReductionParameters.getIntegerParameterValue(
11                                         PARAM_INT_COLOR_NUMBER);
```

Neben der üblichen Großschreibung der Klassennamen [61] werden diese nach Verwendung und Art des programmiersprachlichen Elements kodiert, so beginnen beispielsweise Interfaces mit dem Präfix „*I*“ oder enden Parameterklassen mit dem Suffix „*Parameters*“. Abgesehen von Laufvariablen sind auch diese mit einem Präfix, diesmal bezogen auf ihren Typ („*i*“ für *integer*, „*f*“ für *float* usw.) und ihre Erreichbarkeit („*m_*“ für globale Variablen), markiert und je nach ihrer Verwendung benannt. Während herkömmliche Variablen üblicherweise mit einem Kleinbuchstaben beginnen, werden Konstanten ausschließlich mit Großbuchstaben geschrieben [61]. Die Implementierung beinhaltet Kommentare für die Generierung einer Dokumentation (JavaDoc) und das leichtere Verständnis von Routinen.

Ein damit verwandtes, inhaltsbezogenes Kriterium ist die Angemessenheit der Präsentation des Inhalts für die Zielgruppe. Im Gegensatz zur bereits diskutierten Wahl der Themengebiete (siehe didaktisch-technische Ebene) wird hier auf Struktur und Lesbarkeit der implementierten Effekte Bezug genommen. Aufgrund zahlreicher Grundlehrveranstaltungen der Informatikstudien können, das Programmieren betreffend, bei der Zielgruppe fundierte Kenntnisse vorausgesetzt werden [92]. Speziell Java, die Programmiersprache, in welcher die VEL entwickelt wird, hat den Vorteil, dass sie auch bei einer Vielzahl an Lehrveranstaltungen im Grundstudium eingesetzt wird. Diese beiden Gründe sprechen für die Angemessenheit der Inhaltspräsentation.

Sieht man über die logische Strukturierung des Inhalts in Packages und Klassen hinaus, organisiert das Framework auch die Entwicklung selbst in zwei Module. So folgt dem Bildeffekt mittels der Überleitung durch seine Frame-zu-Frame Anwendung die Implementierung des Videoeffekts. Die Struktur wird für die Organisation der Übung (siehe Abschnitt 6.1.2) aufgegriffen und betont, um den Lernprozess zu erleichtern. StudentInnen können sich so jeweils auf einen Teil des Problems konzentrieren und erlangen durch die vorgeschlagene Reihenfolge einen besseren Überblick.

Gesamtheitliche Betrachtung

Zusammengefasst kann die zusätzliche Verwendung der *ColorReduction Effekte* in der Lehre, über das Verschieben von Effektprimitiven entsprechend eines Bewegungsfelds hinaus, weitere Aspekte und grundsätzliche Ansätze zur Erstellung zeitlich kohärenter Videoeffekte motivieren und exemplarisch darlegen. Die Effekte setzen jedoch mehr Vorwissen im Bereich Bildverarbeitung voraus und weichen deshalb etwas von der intendierten Zielgruppe ab.

Das Framework selbst bietet gute Voraussetzungen für den Lehreinsatz und beeinflusst allgemein wichtige didaktisch-technische sowie inhaltsbezogene Elemente für alle darin implementierten Effekte. Hilfestellungen zur reinen Anwendung der Bild- und Videoeffektapplikationen oder den Effekten zu Grunde liegende Literatur sind nicht Teil der Library und müssen momentan unabhängig davon zur Verfügung gestellt werden. Ein gegebenenfalls auftretendes Problem sind längere Verarbeitungszeiten, welche die Motivation negativ beeinflussen können. Die neu implementierten Effekte halten strukturelle sowie formale Konventionen der VEL ein, um das Arbeiten im System und an den Effekten zu erleichtern.

Kapitel 7

Fazit und Ausblick

Eine Vielzahl an Publikationen bietet verschiedene Lösungen, nicht-photorealistische und zeitlich kohärente Animationen ausgehend vom Bildraum zu erzeugen. Doch ist ihnen gemein, dass aufeinander folgende Frames durch das Verschieben einfacher Effektprimitive oder Videoobjekte mit semantischer Bedeutung entsprechend der im Videomaterial stattgefundenen Bewegung konstruiert werden. Die Qualität ist dabei entscheidend von den Ergebnissen des eingesetzten Verfahrens zur Bewegungsschätzung abhängig.

Diese Diplomarbeit implementiert verschiedene Ansätze zum Erhalt zeitlicher Kohärenz. Eine vorhandene *Video Effect Library* wurde um Bild- und Videoeffekte, deren Verarbeitungsweisen bereits in der Bibliothek realisierte Strategien inhaltlich ergänzen, erweitert.

Die Diplomarbeit präsentiert einen neu implementierten cartoonartigen Bildeffekt, der wichtige Bildinformationen beibehält und gleichzeitig Details vernachlässigt. Farbreduktion durch stabilisiertes k-Means Clustering, Binning des Farbhistogramms oder nicht-uniforme Teilung der Farbbänder sind Optionen für den ersten Schritt der Abstraktion in Richtung Stil eines Cartoons. Wie visuell ansprechend das Ergebnis ist, hängt dabei von der gewählten Farbreduktion und einem zu spezifizierenden Detailgrad ab. Linienzeichnungen, realisiert durch stilisierte Kanten, sind als weiteres künstlerisches Merkmal implementiert. Gradientengesteuerte Streuung von Helligkeiten verstärkt Kontraste und verbessert das Ergebnis. Durch Variationsmöglichkeiten unterschiedlicher Parameter kann der entwickelte Stil des Effekts von BenutzerInnen deutlich modifiziert werden.

Auf diesem Bildeffekt bauen drei unterschiedliche Videoeffekte auf, welche ohne zusätzliche BenutzerInnenintervention animiert wirkende Sequenzen erzeugen. Einer davon verarbeitet mehrere Frames gleichzeitig und knüpft an global konzipierte Ansätze an. Die erwartete Verbesserung trat nur in Form einer Abflachung der Farbdifferenzen ein. Flimmern und die Anzahl an Farbsprüngen blieben etwa auf einem Niveau mit der Frame-zu-Frame Anwendung des Bildeffekts oder stiegen in Extremfällen sogar an. Ein weiterer Videoeffekt mittelt die Farben

über ein spezifiziertes Intervall oder zwischen adjazenten Frames. Durch ihn konnten allgemeine Farbsprünge vermindert werden. Eine signifikante Verbesserung der räumlichen Änderungen wurde jedoch nicht festgestellt. Erst ein dritter Videoeffekt, welcher Differenzen benachbarter Frames berechnet, konnte Flimmern deutlich reduzieren.

Ein zweites Bild- und Videoeffektpaar realisiert einen bereits existierenden Effekt, der verspricht, ohne zusätzliche Transformationsschritte flüssige Animationen zu erzeugen. Seine Ergebnisse sind visuell sehr ansprechend, jedoch erwies sich, dass er teils mehr fälschlicherweise veränderte Pixel erzeugt als die im Zuge der Diplomarbeit entstandenen Videoeffekte.

Obwohl nicht mit allen implementierten Videoeffekten das gewünschte Ergebnis erzielt werden konnte, gelang es, aufschlussreiche Erkenntnisse zu gewinnen und kohärente Animationen zu erzeugen. Es wurde gezeigt, dass globalere Verarbeitung alleine nicht ausreicht, um Inkonsistenzen zuverlässig zu reduzieren, sondern es darüber hinaus weiterer Methoden bedarf. Die Diplomarbeit präsentiert eine Heuristik, die k-Means Clustering für die sequenzielle Anwendung auf Videos stabilisiert. Als eine mögliche Alternative zum Mean Shift Algorithmus wird dabei mehr Freiraum bezüglich der Wahl der Farbanzahl geboten. Die *Video Effect Library* wurde um Keyframeinterpolation und die erwähnten Effekte erweitert, sodass auch für den künftigen Einsatz der Bibliothek in der Lehre ein Mehrwert entstand. Bild- und Videoeffekte bieten darüber hinaus Raum für weitere kreative, an ihnen orientierte Stile und Effekte.

Eine interessante zukünftige Entwicklung für in dieser Diplomarbeit entstandene Transformationen von Bild- zu Videoeffekt wäre das zusätzliche Erzeugen einer raum-zeitlichen Beschreibung von Videosequenzen durch interne Darstellungen und weitere Analysen gleichfarbiger, zusammenhängender Bereiche. Deren Erweiterung zu Objekten mit semantischer Bedeutung würde die Möglichkeit bieten, gelungene Abbildungen von originalen auf abstrahierte Bilder zu erlangen, stabile Objektränder zu erzeugen und Bewegungslinien zu berechnen. Zusätzliche Erweiterungen der Strategien zur Vermeidung räumlicher Anomalien wären denkbar. Beispielsweise könnte ein Alternativverhalten zum Zeitpunkt abrupter Änderungen der Farbzusammenstellung im Input-Video (z.B. Helligkeitsänderung) definiert werden. Auch bei großen Änderungen der Keyframewerte in kurzen Zeiträumen wäre ein zusätzlicher Transformationsschritt vorteilhaft. Eine weitere Verbesserung der Bibliothek würde mit dem Ermöglichen von Funktionalität hinsichtlich der Erkennung von Schnitten erreicht werden. So wäre sie in der Lage, nicht nur einzelne Szenen, sondern gesamte Videoclips zu verarbeiten. Neben anderen Maßnahmen zur Erhöhung der visuellen Qualität könnte außerdem die Laufzeit der Algorithmen optimiert oder die Bibliothek in einer effizienteren Programmiersprache, etwa C++, implementiert werden.

Danksagung

An dieser Stelle möchte ich mich herzlich bei allen bedanken, die mich während des Studiums und der Erstellung dieser Diplomarbeit unterstützt haben.

Im Besonderen gilt das für Stefan Siegl, der mir immer zur Seite gestanden ist und während des ganzen Studiums den Rücken gestärkt hat. Seine Hilfe bei vielen Formulierungen und der Korrektur der Diplomarbeit war unbezahlbar.

Mein herzlicher Dank gilt außerdem Melanie Fraunschiel, die mich nicht nur ein gutes Stück durch das Studium begleitet hat, sondern auch als Korrekturleserin zur Verfügung gestanden ist.

Ein weiteres Dankeschön geht an meine weniger technik-affinen FreundInnen, die mich moralisch stets unterstützt haben, obwohl ich sie immer wieder mit fachspezifischen Problemen gelangweilt habe. Martin Pamphlett, der von London aus mein Abstract unter die Lupe genommen hat, sei an dieser Stelle namentlich erwähnt.

Nicht zuletzt möchte ich meinen Eltern danken, die mir finanziell unter die Arme gegriffen und somit das Studium überhaupt erst ermöglicht haben.

Schließlich gilt mein Dank natürlich den Initiatoren und Förderern dieser Diplomarbeit Mag. DI Dr. Prof. Margrit Gelautz und DI Jürgen Platzer, der darüber hinaus seine Video Effect Library zur Verfügung gestellt hat.

Literaturverzeichnis

[1] Aseem Agarwala. SnakeToonz: A semi-automatic approach to creating cel animation from video. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*, pages 139–163, 2002.

[2] Jue Wang, Pravin Bhat, R. Alex Colburn, Maneesh Agrawala and Michael F. Cohen. Interactive video cutout. *ACM Transactions on Graphics*, 24(3):585–594, 2005.

[3] Jue Wang, Steven M. Drucker, Maneesh Agrawala and Michael F. Cohen. The cartoon animation filter. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1169–1173, 2006.

[4] Nur Arad and Craig Gotsman. Enhancement by image-dependent warping. *IEEE Transactions on Image Processing*, 8(8):1063–1074, 1999.

[5] Bruce G. Baumgart. Winged edge polyhedron representation. Technical report, 1972.

[6] Yaar Schnitman, Yaron Caspi, Daniel Cohen-Or and Dani Lischinski. Inducing semantic segmentation from an example. In *ACCV: 7th Asian Conference on Computer Vision*, volume 3852, pages 373–384, 2006.

[7] John P. Collomosse. *Higher level techniques for the artistic rendering of images and video*. PhD thesis, University of Bath, 2004.

[8] John P. Collomosse and Peter M. Hall. A mid-level description of video, with application to non-photorealistic animation. In *BMVC '04: Proceedings 15th British Machine Vision Conference*, 2004.

[9] John P. Collomosse and Peter M. Hall. Video Paintbox: The fine art of video painting. *Computers & Graphics*, 29(6):862–870, 2005.

[10] Dorin Comaniciu and Peter Meer. Mean Shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[11] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 769–776, 2002.

[12] Daniel Dementhon and Megret Remi. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *CVPR '02:IEEE International Conference on Computer Vision and Pattern Recognition*, pages 142–151, 2002.

[13] Mathias Eitz. Bilateral filtering. Webseite, 2007. `http://user.cs.tu-berlin.de/~eitz/bilateral_filtering/` zuletzt besucht am 26.03.2008.

[14] Jean-Daniel Fekete, Érick Bizouarn, Éric Cournarie, Thierry Galas and Frédéric Taillefer. TicTacToon: A paperless system for professional 2D animation. In *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 79–90, 1995.

[15] Adam Finkelstein and Marisa Range. Image mosaics. In *EP '98/RIDT '98: Proceedings of the 7th International Conference on Electronic Publishing, Held Jointly with the 4th International Conference on Raster Imaging and Digital Typography*, pages 11–22, 1998.

[16] Allison W. Klein, Peter-Pike J. Sloan, Adam Finkelstein and Michael F. Cohen. Stylized video cubes. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 15–22, 2002.

[17] Allison W. Klein, Tyler Grant, Adam Finkelstein and Michael F. Cohen. Video mosaics. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*, pages 21–28, 2002.

[18] Charles E. Jacobs, Adam Finkelstein and David H. Salesin. Fast multiresolution image querying. In *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 277–286, 1995.

[19] John L. Barron, David J. Fleet and Steven S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[20] Max Fleischer. Method of producing moving picture cartoons. US Patent, 1917. Patent No. 1,242,674.

[21] International Organization for Standardization. Ergonomics of human-system interaction. Webseite, 2008. `http://www.iso.org` zuletzt besucht am 11.06.2009.

[22] Keinosuke Fukunaga and Larry D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[23] Saša Galic and Sven Lončarić. Spatio-temporal image segmentation using optical flow and clustering algorithm. In *IWISPA '00: Proceedings of the First International Workshop on Image and Signal Processing and Analysis*, pages 63–68, 2000.

[24] Christopher M. Christoudias, Bogdan Georgescu and Peter Meer. Synergism in low level vision. In *ICPR '02: Proceedings of the 16th International Conference on Pattern Recognition*, pages 150–155, 2002.

[25] J. Hall, Darrel Greenhill and Graeme A. Jones. Segmenting film sequences using active surfaces. In *ICIP '97: Proceedings of the 1997 International Conference on Image Processing*, page 751, 1997.

[26] Paul Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, 1990.

[27] Adam Lake, Carl Marshall, Mark Harris and Marc Blackstein. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR '00: Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, pages 13–20, 2000.

[28] James Hays and Irfan Essa. Image and video based painterly animation. In *NPAR '04: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering*, pages 113–120, 2004.

[29] James Hays and Irfan Essa. Image and video-based painterly animation. Webseite, 2004. `http://www-static.cc.gatech.edu/gvu/perception/projects/artstyling/` zuletzt besucht am 26.03.2008.

[30] Donald Hearn and M. Pauline Baker. *Computer Graphics*. Pearson Education, Upper Saddle River, USA, third edition, 2004.

[31] Peter V. Henstock and David M. Chelberg. Automatic gradient threshold determination for edge detection. *IEEE Transactions on Image Processing*, 5(5):784–787, 1996.

[32] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 453–460, 1998.

[33] Aaron Hertzmann. Fast paint texture. In *NPAR '02: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*, pages 91–96, 2002.

[34] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *NPAR '00: Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, pages 7–12, 2000.

[35] Michael Hoch and Peter Litwinowicz. A semi-automatic system for edge tracking with snakes. *The Visual Computer*, 12(2):75–83, 1996.

[36] Andreas Holzinger. Beurteilungskriterien für Lernsoftware. Technical report, Universität Graz, 2003.

[37] Peng Huang. Video special effects. Master's thesis, University of Surrey, 2006.

[38] Ludger Humbert. *Didaktik der Informatik*. Teubner Verlag, Wiesbaden, 2. edition, 2006.

[39] Alper Yilmaz, Omar Javed and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):13, 2006.

[40] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55, 1997.

[41] Noah Snavely, Charles Lawrence Zitnick, Sing Bing Kang and Michael Cohen. Stylizing 2.5-D video. In *NPAR '06: Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, pages 63–69, 2006.

[42] HongJiang Zhang, Atreyi Kankanhalli and Stephen W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28, 1993.

[43] Michael F. Cohen, Alex Colburn, Adam Finkelstein, Allison W. Klein and Peter-Pike J. Sloan. Video cubism. Technical Report MSR-TR-2001-45, Microsoft Research (MSR), 2001.

[44] Alice Y. Kolb and David A. Kolb. The kolb learning style inventory. Technical report, Case Western Reserve Universität, 2005.

[45] Alexander Kolliopoulos. Image segmentation for stylized non-photorealistic rendering and animation. Master's thesis, University of Toronto, 2005.

[46] Levente Kovács and Tamás Szirányi. Creating animations combining stochastic paintbrush transformation and motion detection. In *ICPR '02: International Conference on Pattern Recognition*, volume 2, pages 1090–1093, 2002.

[47] Levente Kovács and Tamás Szirányi. Coding of stroke-based animations. In *WSCG '04: Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 81–84, 2004.

[48] John Lansdown and Simon Schofield. Expressive rendering: A review of nonphotorealistic techniques. *IEEE Computer Graphics and Applications*, 15(3):29–37, 1995.

[49] Sung-Soo Hong, Jong-Chul Yoon, In-Kwon Lee and Siwoo Byun. Interactive system for efficient video cartooning. In *Mirage '07: Model-based Imaging, Rendering, Image Analysis and Graphical Special Effects*, pages 161–172, 2007.

[50] Richard Linklater. Walking life. Cinema movie, 2001.

[51] Richard Linklater. A scanner darkly. Cinema movie, 2006.

[52] Peter Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 407–414, 1997.

[53] Kaleigh Smith, Yunjun Liu and Allison Klein. Animosaics. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer Animation*, pages 201–208, 2005.

[54] Jörn Loviscach. Klare Sicht. Bilder durch Verformen schärfen. *c't Zeitschrift für Computertechnik*, 99(22):236–ff, 1999.

[55] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *JCAI '81: Joint Conference on Aritificial Intelligence*, pages 674–679, 1981.

[56] Huitao Luo and Alexandros Eleftheriadis. Spatial temporal active contour interpolation for semi-automatic video object generation. In *ICIP '99: International Conference on Image Processing*, pages 944–948, 1999.

[57] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[58] David Marr and Ellen C. Hildreth. Theory of edge detection. Technical report, Massachusetts Institute of Technology, 1979.

[59] Scott McCloud. *Making Comics: Storytelling Secrets of Comics, Manga and Graphic Novels*. Harper Paperbacks, New York, USA, 2006.

[60] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 477–484, 1996.

[61] Sun Microsystems. Code conventions for the java programming language. Webseite, 1999. `http://java.sun.com/docs/codeconv/index.html` zuletzt besucht am 11.06.2009.

[62] Tomáš Mikolov. Color reduction using k-means clustering. Technical report, Brno University of Technology, 2007.

[63] Christian Mikunda. *Kino spüren. Strategien der emotionalen Filmgestaltung*. Facultas, Wien, 2002.

[64] Conrad George Mueller and Mae Rudolph. *Light and Vision*. Time Life, Michigan, USA, 1966.

[65] Claudia Pfleger. *Die Didaktik des Fernunterrichts in Formalerschließung unter besonderer Berücksichtigung von Lernprogrammen*. PhD thesis, Humboldt-Universität, 2002.

[66] Tuan Q. Pham and Lucas J. van Vliet. Separable bilateral filtering for fast video preprocessing. In *ICME '05: IEEE International Conference on Multimedia and Expo*, pages 454–457, 2005.

[67] Jürgen Platzer. Integrating statistical basefunctionality in interactive visual data analysis. Master's thesis, Technische Universität Wien, 2007.

[68] Luise F. Pusch. *Die Frau ist nicht der Rede Wert*. Suhrkamp Verlag, Frankfurt am Main, 1999.

[69] Dorin Comaniciu, Visvanathan Ramesh and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR '00: IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 142–149 vol.2, 2000.

[70] John P. Collomosse, David Rowntree and Peter M. Hall. Rendering cartoon-style motion cues in post-production video. *Graphical Models*, 67(6):549–564, 2005.

[71] Aseem Agarwala, Aaron Hertzmann, David H. Salesin and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics*, 23(3):584–591, 2004.

[72] Volker Scholz. *New editing techniques for video post-processing*. PhD thesis, Universität des Saarlandes, 2007.

[73] Mehmet K. Ozkan, Ibrahim M. Sezan and Murat A. Tekalp. Adaptive motion-compensated filtering of noisy image sequences. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(4):277–290, 1993.

[74] Mehmet Sezgin and Bulent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–168, 2004.

[75] Jonathan Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In *WACG '96: 1st Workshop on Applied Computational Geometry: Towards Geometric Engineering*, 1996.

[76] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR '94.: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[77] Jue Wang, Yingqing Xu, Heung-Yeung Shum and Michael F. Cohen. Video tooning. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 574–583, New York, NY, USA, 2004.

[78] Tamás Szirányi and Zoltán Tóth. Optimization of paintbrush rendering of images by dynamic MCMC methods. In *EMMCVPR '01: Proceedings of the Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 201–215, 2001.

[79] Tamás Szirányi and Zoltán Tóth. Random paintbrush transformation. In *ICPR '00: International Conference on Pattern Recognition*, volume 3, pages 151–154, 2000.

[80] Frank Miller, Robert Rodriguez, Quentin Tarantino. Sin City. Cinema movie, 2005.

[81] The GIMP Documentation Team. GNU image manipulation program user manual. Webseite, 2008. `http://www.gimp.org/docs/` zuletzt besucht am 16.04.2008.

[82] Adrien Bousseau, Fabrice Neyret, Joëlle Thollot and David Salesin. Video watercolorization using bidirectional texture advection. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Papers*, number 104, 2007.

[83] Adrien Bousseau, Matt Kaplan, Joëlle Thollot and François X. Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR '06: Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, pages 141–149, 2006.

[84] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, pages 839–846, 1998.

[85] Klaus D. Tönnies. *Grundlagen der Bildverarbeitung*. Pearson, München, 2005.

[86] Andreas Auinger und Christian Stary. *Didaktikgeleiteter Wissenstransfer*. Vieweg und Teubner, Wiesbaden, 2005.

[87] Peter Baumgartner, Hartmut Häfele und Kornelia Maier-Häfele. *E-Learning Praxishandbuch. Auswahl von Lernplattformen.* Studien Verlag, Innsbruck-Wien, 2002.

[88] Robert Strzebkowski und Nicole Kleeberg. Interaktivität und Präsentation als Komponenten multimedialer Lernanwendungen. In *Information und Lernen mit Multimedia und Internet*, pages 229–246. 2002.

[89] Thomas Ottmann und Peter Widmayer. *Algorithmen und Datenstrukturen*. Spektrum, Berlin, 4. edition, 2002.

[90] Ioannis Stamelos, Ioannis Refanidis, Panagiotis Katsaros, Alexis Tsoukias, Ioannis Vlahavas and Andreas Pombortsis. An adaptable framework for educational software evaluation. In *Decision Making: Recent Developments and Worldwide Applications*, pages 317–329. 2000.

[91] Ernst A. Weber. *Sehen, gestalten und fotografieren*. Birkhäuser, Basel, 1990.

[92] Technischen Universität Wien. Studienpläne für Bachelor- und Masterstudien der Studienrichtung Informatik an der Technischen Universität Wien. Verordnungstext, 2007.

[93] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1221–1226, 2006.

[94] Sven C. Olsen, Holger Winnemöller and Bruce Gooch. Implementing real-time video abstraction. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 133, 2006.

[95] Michael Kass, Andrew Witkin and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[96] Jue Wang, Bo Thiesson, Yingqing Xu and Michael Cohen. Image and video segmentation by anisotropic kernel mean shift. In *ECCV '04: European Conference on Computer Vision*, pages 238–249, 2004.

[97] Borko Furth, HongJiang Zhang and Stephen W. Smoliar. *Video and Image Processing in Multimedia Systems*. Kluwer Academic Publishers, Norwell, USA, 1995.