



Masterarbeit

**JENIFFER2: Ein RAW-Processor
mit wählbaren Demosaicing-Algorithmen
für das universelle RAW-Format DNG**

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Informationsdienste
Studiengang Medieninformatik
Eugen Ljavin, eugen.ljavin@student.uni-tuebingen.de, 2020

Bearbeitungszeitraum: 01.11.2019 - 04.06.2020

Betreuer/Gutachter: Prof. Dr. Thomas Walter, Universität Tübingen
Zweitgutachter: Prof. Dr. Andreas Schilling, Universität Tübingen

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Eugen Ljavin (Matrikelnummer 4220530), 28. Mai 2020

Kurzfassung

Digitale Aufnahmen werden von Bildsensoren aufgezeichnet und häufig in Form von unverarbeiteten Sensordaten gespeichert. Dazu existieren sogenannte RAW Formate, bei den es sich um proprietäre Dateiformate handelt. Eine Ausnahme bietet das offen spezifizierte Digital Negative (DNG), das ein nicht proprietäres RAW Format ist. Um aus den Sensordaten eine Aufnahme zu erhalten, müssen RAW Formate von RAW Konvertern verarbeitet werden und durchlaufen dazu eine Vielzahl von Verarbeitungsschritten. Von besonderer Bedeutung ist dabei ein Verarbeitungsschritt, der Demosaicing genannt wird. Dieser kann vom Java Extended NEF Image File Format Editor (JENIFFER), einem Open Source RAW Konverter, beeinflusst werden. JENIFFER unterstützt jedoch nur das Nikon Electronic Format (NEF), ein proprietäres RAW Format des Herstellers Nikon.

In der vorliegenden Arbeit wurde das DNG Format in Bezug auf die RAW Konvertierung analysiert und es wurde auf Basis von JENIFFER ein RAW Konverter für die Verarbeitung von DNG Dateien entwickelt. Dazu wurden zunächst in einem theoretischen Teil die Dateistruktur, die Verarbeitungsschritte und die Erzeugungsmöglichkeiten von DNG Dateien an einem laufenden Beispiel untersucht. Gezeigt wurde, dass verschiedene Dateistrukturen, unterschiedlich kodierte Sensordaten sowie diverse Verarbeitungsschritte beim Auslesen und Verarbeiten von DNG Dateien berücksichtigt werden müssen. Ferner wurden Charakteristiken des DNG Formats identifiziert, welche dessen herstellerübergreifende Nutzung ermöglichen. Auf Grundlage der Analyse konnten in einem darauffolgenden praktischen Teil Komponenten zum Auslesen und Verarbeiten von DNG Dateien einschließlich einer grafischen Benutzeroberfläche implementiert werden. Abschließend wurden mehrere Vergleiche durchgeführt, um die Auswirkungen des Demosaicings auf die Qualität einer digitalen Aufnahme zu veranschaulichen sowie die Ausführungszeiten unterschiedlicher Verarbeitungsschritte zu erörtern. Aus diesen geht hervor, dass die Pixelwiederholung der schnellste der vier implementierten Demosaicing Algorithmen ist, jedoch die schletesten Ergebnisse erzeugt. Deutlich länger braucht die bikubische Interpolation, erzielt jedoch die vergleichsweise besten Ergebnisse.

Inhaltsverzeichnis

Abbildungsverzeichnis	ix
Tabellenverzeichnis	xi
Quelltextverzeichnis	xiii
Abkürzungsverzeichnis	xv
1 Einleitung	1
1.1 Motivation	1
1.2 Ziele	2
1.3 Vorgehen	2
2 Grundlagen	5
2.1 Bildsensor	5
2.2 Bayer Filter	6
2.2.1 Aufbau und Funktionsweise	7
2.2.2 Demosaicing	9
2.3 Dateiformate digitaler Aufnahmen	10
2.3.1 Tagged Image File Format (TIFF)	11
2.3.2 Joint Photographic Experts Group (JPEG)	14
2.3.3 Exchangeable Image File Format (EXIF)	15
2.3.4 Rohdatenformate (RAW)	15
2.4 RAW Konvertierung	17
2.4.1 Verarbeitungsschritte	17
2.4.2 RAW Konverter	18
2.4.3 JENIFFER	20
3 Digital Negative (DNG)	21
3.1 Eigenschaften	21
3.2 Dateiformat	23
3.2.1 Bestandteile	24
3.2.2 Aufbau	25
3.3 Erzeugung von DNG Dateien	27
3.3.1 Kamerainterne Erzeugung	27
3.3.2 Kameraexterne Erzeugung	28

Inhaltsverzeichnis

3.4	RAW Konvertierung	30
3.4.1	Rohdatenzuordnung	32
3.4.2	Weißabgleich	34
3.4.3	Demosaicing	36
3.4.4	Bildzuschnitt	37
3.4.5	Farbraumtransformation	38
3.4.6	Weitere Verarbeitungsschritte	44
4	Konzeption	45
4.1	Anforderungen	45
4.1.1	Funktionale Anforderungen	45
4.1.2	Nicht-funktionale Anforderungen	48
4.2	Architektur	48
5	Implementierung	51
5.1	DNG-Reader	51
5.1.1	Aufbau	51
5.1.2	Auslesen einer DNG Datei	54
5.2	DNG-Processor	58
5.2.1	Aufbau	58
5.2.2	Verarbeitung von Rohdaten	61
5.3	Grafische Benutzeroberfläche	63
5.3.1	Unterteilung der Benutzeroberfläche	64
5.3.2	Aufbau und Funktionsweise	66
6	Vergleich	71
6.1	Demosaicing Algorithmen	71
6.2	Zeitverhalten der Verarbeitungsschritte	73
6.2.1	Messung	73
6.2.2	Auswertung	74
7	Fazit und Ausblick	79
7.1	Fazit	79
7.2	Ausblick	80
A	Ausführungszeiten	83
B	Installationsanleitung	87
B.1	Ausführen	87
B.2	Kompilieren	87
C	Rechtlicher Hinweis	89
	Literaturverzeichnis	91

Abbildungsverzeichnis

1.1	Implizite Verarbeitung einer proprietären RAW Datei durch JENIFFER2	2
2.1	Erfassungsprozess einer digitalen Aufnahme samt Komponenten, angelehnt an [Wal05, S. 78]	5
2.2	Beispiel eines Bayer Filters mit dem Muster GRBG	7
2.3	Die vier unterschiedlichen Permutationen eines Bayer Filters	7
2.4	Prinzip der Farbtrennung durch einen Bayer Filter nach [Luk09, S. 5]	8
2.5	Bildsensordaten einer digitalen Aufnahme mit einem Bayer Filter des Musters GBRG	8
2.6	Zusammenfassung von drei benachbarten Pixel verschiedener Farben durch die Pixelwiederholung nach [Wal05, S. 97]	9
2.7	Interpolierte Bildsensordaten durch die Pixelwiederholung, die zu Zipper Artefakten und verfälschten Farben führt	10
2.8	Die Dateistruktur einer TIFF Datei	12
2.9	Als Strips und Tiles gespeicherte Bilddaten in einer TIFF Datei	14
2.10	Bestandteile einer proprietären RAW Datei, angelehnt an [ABF06, S. 7]	16
2.11	Verarbeitungsschritte der RAW Konvertierung, angelehnt an [RSYD05, S. 3]	18
2.12	RAW Konvertierung einer proprietären RAW Datei durch einen RAW Konverter, angelehnt an [Pea10a]	19
2.13	Typische Benutzeroberfläche eines RAW Konverters am Beispiel von RawTherapee [Raw]	19
2.14	Interpolationsdialog des RAW Konverters JENIFFER	20
3.1	RAW Konvertierung einer DNG Datei durch einen RAW Konverter, angelehnt an [Pea10a]	22
3.2	Typische Dateistruktur einer DNG Datei	26
3.3	Kameraexterne Erzeugung einer DNG Datei durch einen DNG Konverter, angelehnt an [Pea10a]	28
3.4	Voreinstellungsmasken des Adobe DNG Converters [Ado18]	29
3.5	Verarbeitungsschritte einer DNG Datei	31
3.6	Zuordnung von Rohdaten zu linearen Referenzwerten	32
3.7	Rohdatenzuordnung im Vorher - Nachher Vergleich am laufenden Beispiel	34
3.8	Weißabgleich im Vorher - Nachher Vergleich am laufenden Beispiel	35
3.9	Auswirkung des Weißabgleichs auf die finale Aufnahme	36

Abbildungsverzeichnis

3.10 Demosaicing im Vorher - Nachher Vergleich am laufenden Beispiel	37
3.11 Beispielhafte Aufteilung der einzelnen Bereiche der Rohdaten einer DNG Datei	38
3.12 Farbraumtransformation im Vorher - Nachher Vergleich am laufenden Beispiel	43
4.1 UML Komponentendiagramm des RAW Konverters	49
5.1 Klassendiagramm des DNG-Readers	52
5.2 Beispielhaftes Befüllen eines Puffers mit Bytes einer DNG Datei	55
5.3 Beispielhafte Zusammenfassung zweier Bytes zu einem 16 Bit Rohdatenwert	56
5.4 Klassendiagramm des DNG-Processors	59
5.5 Tab "Bibliothek" des RAW Konverters	64
5.6 Hinweisdialog des RAW Konverters	64
5.7 Interpolationsdialog des RAW Konverters	65
5.8 Tab "Editor" des RAW Konverters	65
5.9 Das MVP Entwurfsmuster, angelehnt an [Pot96, S. 6]	66
5.10 Bestandteile zur Realisierung des MVP Entwurfsmusters mittels afterburner.fx	67
6.1 Gegenüberstellung verschiedener Demosaicing Algorithmen anhand einer markanten Stelle des laufenden Beispiels	72
6.2 Ausführungszeiten der Demosaicing Algorithmen	75
6.3 Ausführungszeiten der Verarbeitungsschritte (ohne Demosaicing)	75
6.4 Anteile der Ausführungszeiten je Demosaicing Algorithmus	76

Tabellenverzeichnis

2.1	Wichtige TIFF Tags mit ihrer Beschreibung	13
2.2	RAW Formate bekannter Hersteller [Har] [Ado20]	17
3.1	Zur Durchführung der Rohdatenzuordnung relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)	34
3.2	Zur Durchführung des Demosaicings relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)	36
3.3	Zur Durchführung des Bildzuschnitts relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)	38
3.4	Zur Durchführung der Farbraumtransformation relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)	44
6.1	Ausführungszeiten der einzelnen Verarbeitungsschritte für jeden Demosaicing Algorithmus	74
A.1	Ausführungszeiten der Verarbeitungsschritte mit der Pixelwiederholung als Demosaicing Algorithmus	83
A.2	Ausführungszeiten der Verarbeitungsschritte mit der bilinearen Interpolation als Demosaicing Algorithmus	84
A.3	Ausführungszeiten der Verarbeitungsschritte mit der Median Interpolation als Demosaicing Algorithmus	84
A.4	Ausführungszeiten der Verarbeitungsschritte mit der bikubischen Interpolation als Demosaicing Algorithmus	85

Quelltextverzeichnis

5.1	Methode zum Auslesen eines Bytes einer DNG Datei	55
5.2	Methode zum Dekodieren eines Strips oder eines Tiles	56
5.3	Methode zum Auslesen und Dekodieren von Strips	57
5.4	Methodensignatur des generischen Interfaces Processor	60
5.5	Die generische Klasse Pipeline	60
5.6	Die abstrakte Klasse PreProcessorOperation	61
5.7	Die abstrakte Klasse PreProcessor	62
5.8	Beispielhafte Konfiguration und Ausführung einer Verarbeitungspipeline am laufenden Beispiel	63
5.9	Beispielhafte XML Datei des Tabs "Bibliothek" zur Beschreibung einer grafischen Komponente	68
5.10	Beispielhafter Presenter des Tabs "Bibliothek"	68

Abkürzungsverzeichnis

A/D	Analog-Digital
ACR	Adobe Camera RAW
CCD	Charge-Coupled Device
CFA	Color Filter Array
CMOS	Complementary Metal Oxide Semiconductor
CSS	Cascading Style Sheets
DNG	Digital Negative
Exif	Exchangeable Image File Format
IFD	Image File Directory
IPTC	International Press Telecommunications Council
IQ	Image Quality
JENIFFER	Java Extended NEF Image File Format Editor
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
LUT	Lookup-Tabelle
MVP	Model–View–Presenter
NEF	Nikon Electronic Format
PNG	Portable Network Graphics
RIMM	Reference Input Medium Metric
TIFF	Tagged Image File Format
TIFF/EP	Tag Image File Format / Electronic Photography
UML	Unified Modeling Language
XML	Extensible Markup Language
XMP	Extensible Metadata Platform

1 Einleitung

Mit dem Ziel das visuelle Erscheinungsbild einer digitalen Aufnahme zu optimieren, geht eine aufwendige Ver- und Bearbeitung der Aufnahme für den professionellen Einsatz einher. Herausragende Ergebnisse lassen sich jedoch nur erzielen, wenn die digitale Aufnahme in Form von unverarbeiteten Sensordaten vorliegt. Zu diesem Zweck werden digitale Aufnahmen in einem von etlichen herstellerabhängigen Dateiformaten gespeichert, die in Summe als RAW oder Rohdatenformate bezeichnet werden. Mit Ausnahme vom Digital Negative (DNG) sind diese nicht offen spezifiziert und folglich proprietär.

Verarbeitet werden RAW Formate mittels RAW Konvertern - umfangreiche Anwendungsprogramme, die auf einer leistungsstarken Hardware ausgeführt werden. Innerhalb eines RAW Konverters durchlaufen die Sensordaten eine Vielzahl unterschiedlicher Verarbeitungsschritte. Von enormer Wichtigkeit ist dabei das sogenannte Demosaicing - ein komplexer Verarbeitungsschritt, welcher die Qualität einer digitalen Aufnahme maßgeblich mitbestimmt [Luk09, S. 13]. Aufgrund der meist festen Verankerung im RAW Konverter kann das Demosaicing im Vergleich zu anderen Verarbeitungsschritten häufig nicht beeinflusst werden. Ein Open Source RAW Konverter, der die Einflussnahme dennoch erlaubt, ist der Java Extended NEF Image File Format Editor (JENIFFER) [WGGK05, S. 438].

1.1 Motivation

Die Einflussnahme auf das Demosaicing wird durch JENIFFER zwar sichergestellt, jedoch kann JENIFFER nur das proprietäre RAW Format des Herstellers Nikon verarbeiten, das Nikon Electronic Format (NEF). Andere RAW Formate werden nicht unterstützt, womit JENIFFER auf einen kleinen Nutzerkreis eingeschränkt ist. Darüber hinaus werden von JENIFFER nur NEF Dateien älterer Nikon Kameras berücksichtigt. Die Verarbeitung von NEF Dateien neuer Kameramodelle ist damit nicht möglich.

Durch die Implementierung von JENIFFER für das nicht proprietäre RAW Format DNG, können auch andere proprietäre RAW Formate berücksichtigt werden: Neben der offengelegten Spezifikation von DNG stellt das Unternehmen Adobe einen DNG Konverter zur Verfügung, der zur Konvertierung einer beliebigen proprietären RAW Datei in eine DNG Datei dient. Folglich ist ein RAW Konverter, der DNG Dateien verarbeiten kann, implizit in der Lage, proprietäre RAW Dateien zu verarbeiten,

Kapitel 1. Einleitung

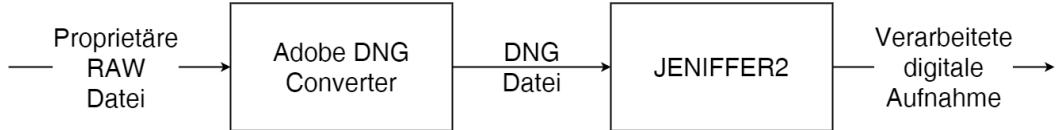


Abbildung 1.1: Implizite Verarbeitung einer proprietären RAW Datei durch JENIFFER2

sofern diese vorher in eine DNG Datei konvertiert wurden. Dieses Prinzip ist in Abbildung 1.1 illustriert und soll sich in einer Neuimplementierung von JENIFFER, dem JENIFFER2, zunutze gemacht werden.

1.2 Ziele

Das primäre Ziel der Arbeit ist die Entwicklung eines RAW Konverters für die Verarbeitung des universellen RAW Formates DNG auf Basis von JENIFFER. In Anlehnung an JENIFFER wird dieser JENIFFER2 genannt. Konkret ist das Auslesen und Verarbeiten von DNG Dateien nach der DNG Spezifikation zu realisieren. Eine moderne Benutzeroberfläche soll es außerdem ermöglichen, den Verarbeitungsprozess steuern und die verarbeitet Aufnahme begutachten zu können.

Damit das oben genannte Ziel erreicht werden kann, ist eine umfassende Analyse des DNG Formats notwendig. Dies stellt ein weiteres aber nicht weniger wichtiges Ziel der Arbeit dar. Unter anderem soll Aufschluss über den Aufbau einer DNG Datei gegeben sowie die einzelnen Verarbeitungsschritte verdeutlicht werden. Darüber hinaus sind die Eigenschaften einer DNG Datei, welche diese als nicht proprietäres RAW Format auszeichnen, herauszuarbeiten.

Ein abschließender Vergleich unterschiedlicher Demosaicing Algorithmen soll außerdem die Auswirkungen des Demosaicings auf die Qualität einer digitalen Aufnahme verdeutlichen. Des Weiteren soll das Laufzeitverhalten der einzelnen Verarbeitungsschritte untersucht werden.

1.3 Vorgehen

Zunächst werden in Kapitel 2 wichtige Begriffe, Komponenten und Verfahren erläutert, die zur Erfassung und Verarbeitung einer digitalen Aufnahme notwendig sind. Darüber hinaus werden bedeutende Dateiformate zur Speicherung digitaler Aufnahmen betrachtet.

Die Analyse des DNG Formats erfolgt in Kapitel 3. Dieses befasst sich zuerst mit den Charakteristiken einer DNG Datei und geht auf die beiden Erzeugungsmöglichkeiten von DNG Dateien ein. Im Fokus des Kapitels liegt die Dateistruktur sowie die

1.3. Vorgehen

Verarbeitungspipeline von DNG. Dabei wird zunächst der typische Aufbau einer DNG Datei einschließlich seiner Bestandteile erläutert. Abschließend werden die einzelnen Verarbeitungsschritte anhand eines laufenden Beispiels, das mit einer Kamera des Typs Leica SL2 erfasst wurde, veranschaulicht.

Das Kapitel 4 dient zur Konzeption des RAW Konverters. Darin werden funktionale und nicht-funktionale Anforderungen an den RAW Konverter identifiziert und definiert sowie die zugrundeliegende Architektur des RAW Konverters begründet. Auf der Grundlage dieses Konzepts wird im technisch ausgelegten Kapitel 5 die konkrete Implementierung des RAW Konverters durch ausgewählten Quelltext beschrieben.

Unter Zuhilfenahme des implementierten RAW Konverters wird die Arbeit durch das Kapitel 6 vervollständigt. Dieses befasst sich mit dem visuellen Vergleich unterschiedlicher Demosaicing Algorithmen am laufenden Beispiel. Ferner werden die Ausführungszeiten der einzelnen Verarbeitungsschritte gemessen und ausgewertet.

In Kapitel 7 wird die gesamte Arbeit abschließend zusammengefasst und in einem Fazit bewertet. Der Ausblick widmet sich offenen Fragen sowie der zukünftigen Verwendung der Arbeit.

2 Grundlagen

Das Erfassen einer digitalen Aufnahme ist ein komplexer Prozess, der den Einsatz unterschiedlicher Komponenten erforderlich macht. Dieser Prozess kann samt Komponenten gemäß Abbildung 2.1 dargestellt werden und lässt sich wie folgt zusammenfassen: Ein Bildsensor misst zunächst auf ihn einfallendes Licht, das anschließend unter Zuhilfenahme eines Analog-Digital (A/D) Wandlers digitalisiert wird. Die digitalisierten Informationen werden danach in einem Rechnersystem verarbeitet und abschließend in einem Dateiformat gespeichert, das für den weiteren Gebrauch der digitalen Aufnahme geeignet ist oder direkt auf dem Bildschirm angezeigt. [Wal05, S. 77] Die Verarbeitung der digitalisierten Informationen erfolgt durch eine Verarbeitungspipeline entweder als interner Kameraprozess oder kameraextern auf einem Computer [Luk09, S. 7].

Nachfolgend werden im Einzelnen die Komponenten näher erläutert, die zur Erfassung einer digitalen Aufnahme notwendig sind. Des Weiteren werden wichtige Dateiformate zur Speicherung digitaler Aufnahmen beschrieben.

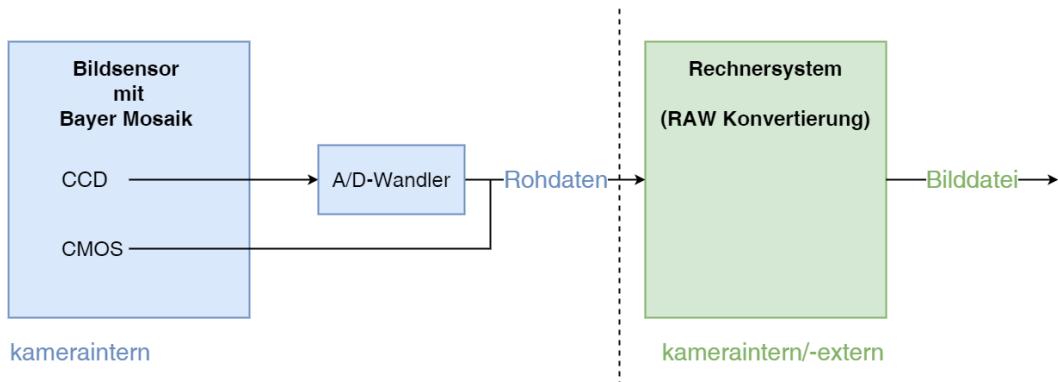


Abbildung 2.1: Erfassungsprozess einer digitalen Aufnahme samt Komponenten, angelehnt an [Wal05, S. 78]

2.1 Bildsensor

Die digitale Bilderfassung beginnt mit dem Aufzeichnen von Licht durch ein elektronisches Bauteil, dem Bildsensor. Konkret handelt es sich dabei um einen lichtempfindlichen Halbleiter, der einfallendes Licht in ein elektrisches Signal

Kapitel 2. Grundlagen

umwandelt [Mas04, S. 19] [Wal05, S. 81f.]. Bildsensoren können, abhängig von ihrer Beschaffenheit, einen weiten Spektralbereich des Lichts erfassen [Nak06, S. 54]. Für die vorliegende Arbeit sind jedoch nur Bildsensoren relevant, welche das für den Menschen wahrnehmbare Lichtspektrum aufzeichnen können. Die zwei verbreitetsten Typen solcher Bildsensoren sind der Charge-Coupled Device (CCD) sowie der Complementary Metal Oxide Semiconductor (CMOS) Sensor, die auf den gleichen Grundprinzipien basieren und sich im Aufbau ähneln.

Ein Bildsensor besteht i. d. R. aus Millionen von Fotodioden, die in einer zweidimensionalen Matrix angeordnet sind. Durch diese Anordnung entsteht eine rechteckige Fläche von Fotodioden, weshalb sie auch als Flächensensoren bezeichnet werden. Wird der Sensor dem Licht ausgesetzt - ein Vorgang, der Belichtung genannt wird - absorbieren die Fotodioden die auf sie einfallenden Photonen (Lichtteilchen) und setzen Elektronen frei, deren Anzahl proportional zur Lichtintensität ist. Dadurch entsteht eine elektrische Ladung, die mittels der Speicherung in einem Kondensator über die Belichtungszeit integriert wird. Nach Abschluss der Belichtung wird die elektrische Ladung bei einem CCD Sensor zeilenweise und sequenziell an jeder Fotodiode ausgelesen, abtransportiert und in Spannung umgewandelt. Die elektrische Spannung wird abschließend von einem A/D-Wandler in diskrete, digitale Werte mit einer Auflösung von 8 bis 16 Bit konvertiert. Bei einem CMOS Sensor hingegen erfolgt die Umwandlung der Ladung in elektrische Spannung durch einen Transistor direkt an der Fotodiode. Dadurch lässt sich die Spannung an jeder Fotodiode für die nachfolgende Digitalisierung parallel auslesen und übertragen, was zu einer deutlich schnelleren Auslesegeschwindigkeit als bei einem CCD Sensor führt. [Lit01, S. 154] [Nak06, S. 54ff.] [Wal05, S. 80ff., 87f.] [Mas04, S. 19] Die nun vorliegenden digitalen Daten werden als Rohdaten bzw. raw-Daten bezeichnet. Diese werden entweder direkt in Form einer Rohdatei (vgl. Kapitel 2.3.4) gespeichert oder durch die kamerainterne Verarbeitungspipeline zu einer fertigen digitalen Aufnahme verarbeitet [Luk09, S. 5].

Damit die Dauer der Belichtung gesteuert werden kann, ist der Bildsensor hinter einer mechanischen Konstruktion, dem sogenannten Kameraverschluss, angebracht. Diese lässt sich für die an der Kamera eingestellte Belichtungszeit öffnen, um Licht auf den Sensor fallen zu lassen [Wal05, S. 29]. Gängig sind heutzutage auch elektronische Kameraverschlüsse, welche die Dauer der Belichtung durch Ein- und Ausschalten des Bildsensors steuern und dadurch besonders leise sind [Nik16]. Ein weiterer Parameter zur Steuerung der Belichtung ist die Blende, mit der sich die Linsenöffnung des Objektivs verkleinern oder vergrößern lässt, um die Menge des einfallenden Lichtes zu regulieren [Mas04, S. 59].

2.2 Bayer Filter

Ein Bildsensor erfasst jegliches Licht, das innerhalb seines empfindlichen Wellenlängenbereiches liegt [Nak06, S. 62]. Aus diesem Grund handelt es sich bei

dem Bildsensor um einen monochromatischen Sensor, der farbenblind ist [Nak06, S. 62] [Luk09, S. 3]. Einzelne Farben, die zur Farbwahrnehmung zwingend benötigt werden, lassen sich somit vom Bildsensor nicht ohne Hilfsmittel erfassen. Um dennoch die notwendigen Farbinformationen zu erhalten, müssen die entsprechenden Farben während der Lichtaufzeichnung voneinander getrennt werden. Die gängigste Methode der Farbtrennung, die sich bis dato bewährt hat, ist der sogenannte Bayer Filter. Dieser ist nach seinem Erfinder Bryce E. Bayer benannt und wurde im Jahre 1976 für das Unternehmen Eastman Kodak Company patentiert [Bay76].

2.2.1 Aufbau und Funktionsweise

Der Bayer Filter besteht aus Farbfiltern der Grundfarben Rot, Grün und Blau, die in einem bestimmten Muster auf den einzelnen Fotodioden des Bildsensors aufgedampft werden, was in Abbildung 2.2 zu erkennen ist [Wal05, S. 85]. Durch die musterhafte Anordnung, die einem Mosaik ähnelt, wird der Bayer Filter auch Bayer Muster oder Bayer Mosaik genannt. Wie im schwarzen Kasten visualisiert werden jeweils ein roter, zwei grüne und ein blauer Farbfilter gruppiert, wobei die beiden grünen Farbfilter diagonal angeordnet werden. Daraus ergeben sich gemäß Abbildung 2.3 insgesamt vier unterschiedliche Permutationen zur Realisierung eines Bayer Filters. Das Muster wiederholt sich vertikal und horizontal über dem gesamten Bildsensor und führt dazu, dass sich in jeder Zeile und Spalte grüne und abwechselnd rote und blaue Farbfilter befinden.

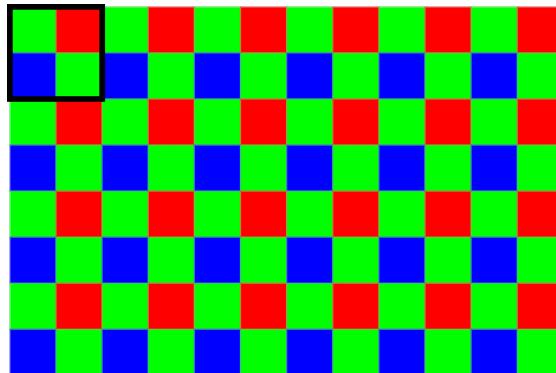


Abbildung 2.2: Beispiel eines Bayer Filters mit dem Muster GRBG

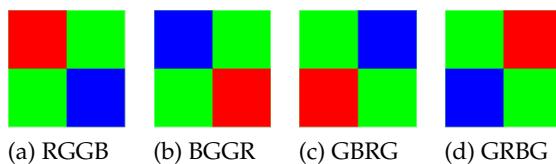


Abbildung 2.3: Die vier unterschiedlichen Permutationen eines Bayer Filters

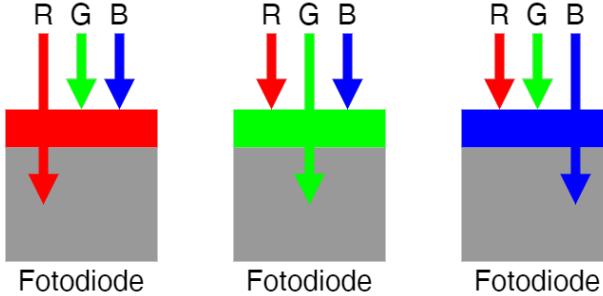
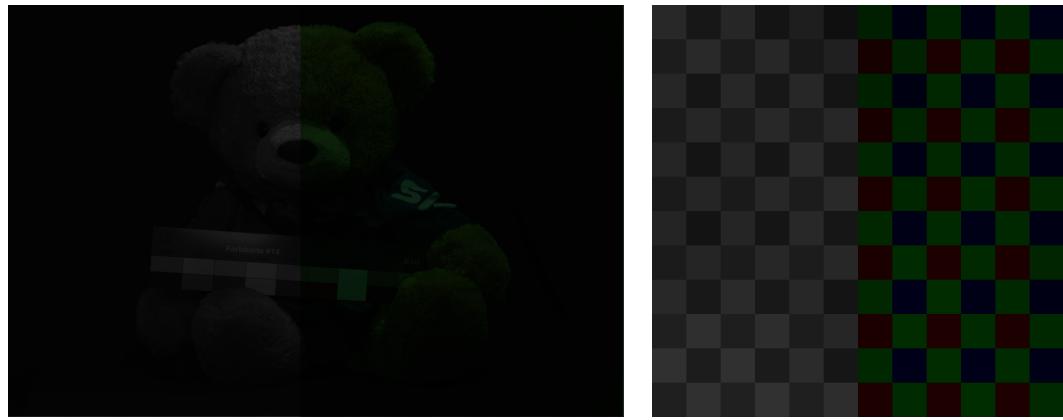


Abbildung 2.4: Prinzip der Farbtrennung durch einen Bayer Filter nach [Luk09, S. 5]

Durch die unterschiedlichen spektralen Empfindlichkeiten der Farbfilter ist der Bildsensor in der Lage, die Lichtmenge einzelner Farben zu erfassen, wobei ein Graustufenwert proportional zur Lichtmenge der Filterfarbe aufgezeichnet wird [Fra04, S. 2]. Abbildung 2.4 visualisiert dieses Prinzip. Konkret kann ein Bildsensor mit einem Bayer Filter 25 Prozent Rot, 25 Prozent Blau und 50 Prozent Grün wahrnehmen. Die höhere Wahrnehmung von Grün beruht auf der Funktionsweise des menschlichen visuellen Systems, das gegenüber Grün eine wesentlich höhere Empfindlichkeit hat als gegenüber Blau und Rot [Bay76].

Im Allgemeinen wird der Bayer Filter auch Color Filter Array (CFA) bezeichnet. Das CFA ist nicht auf einzelne Farben oder Formen beschränkt. So gibt es Bildsensoren, die z. B. neben Rot, Grün und Blau zusätzlich die Farbe Emerald nutzen. Abhängig von der Bauweise des Bildsensors sind auch andere Formen und Anordnungen von Fotodioden möglich, die ein dementsprechendes CFA benötigen. [Wal05, S. 85f.] Das Bayer Mosaik ist somit eine konkrete Ausprägung eines CFA.



(a) Bildsensordaten einer digitalen Aufnahme mit einem Bayer Filter des Musters GBRG

(b) Ausschnitt der digitalen Aufnahme aus (a)

Abbildung 2.5: Bildsensordaten einer digitalen Aufnahme mit einem Bayer Filter des Musters GBRG

Die Bildsensordaten einer digitalen Aufnahme mit einem Bayer Filter des Musters GBRG sind in Abbildung 2.5 zu sehen. Die Abbildung 2.5 (a) zeigt die Aufnahme in voller Auflösung. Rechts davon ist in Abbildung 2.5 (b) ein Ausschnitt dargestellt, auf dem der Bayer Filter deutlich zu erkennen ist. Beide Abbildungen sind mittig geteilt, wobei die rechte Seite der Abbildung zur besseren Veranschaulichung des Bayer Musters jeweils koloriert ist.

2.2.2 Demosaicing

Nach dem Aufzeichnen der Lichtinformationen durch einen Bildsensor mit einem Bayer Filter liegen je Pixel Graustufenwerte einer einzelnen Farbe vor. Da bei einem RGB Bild je Pixel drei Farbkomponenten benötigt werden, müssen die Farbwerte der beiden fehlenden Farbkomponenten aus den benachbarten Pixeln interpoliert werden - ein Prozess, der Demosaicing genannt wird [CT06, S. 1]. Neben dem Wiederherstellen der Farbinformationen einer Aufnahme, ist das Rekonstruieren des strukturellen Inhaltes der Aufnahme ein weiteres wichtiges Ziel des Demosaicings [Luk09, S. 13].

Es existieren zahlreiche Demosaicing Algorithmen, die sich in ihrer Komplexität und dem Resultat stark unterscheiden. Nach [TAR02, S. 3ff.] lassen sich Demosaicing Algorithmen in nicht-adaptive und adaptive Algorithmen klassifizieren. Nicht-adaptive Algorithmen berechnen die fehlenden Farbkomponenten für jedes Pixel nach ein und derselben festgelegten Berechnungsregel. Sie zeichnen sich durch geringe Komplexität aus und erfordern wenig Rechenleistung. Repräsentanten nicht-adaptiver Algorithmen sind u. a. die Pixelwiederholung, die Median-Interpolation und die bilineare Interpolation. Bei adaptiven Demosaicing Algorithmen wird zur Ermittlung der fehlenden Farbkomponenten unter Beachtung der Bildeigenschaften für jedes Pixel eine eigene Berechnungsregel angewendet. Dies kann z. B. nach heuristischen Regeln erfolgen [LK04, S. 347]. Im Vergleich zu nicht-adaptiven Algorithmen sind adaptive Algorithmen deutlich komplexer und rechenintensiver, können jedoch bessere Ergebnisse erzielen. Ein Vertreter adaptiver Algorithmen ist beispielsweise die Interpolation durch Kantenerkennung. [TAR02, S. 3ff.]

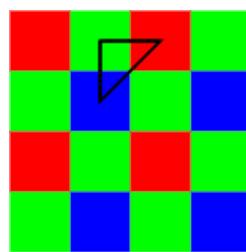


Abbildung 2.6: Zusammenfassung von drei benachbarten Pixel verschiedener Farben durch die Pixelwiederholung nach [Wal05, S. 97]

Kapitel 2. Grundlagen

Exemplarisch wird das Demosaicing anhand der Pixelwiederholung veranschaulicht, bei der es sich um einen sehr einfachen Demosaicing Algorithmus handelt. Bei der Pixelwiederholung werden zur Rekonstruktion aller drei RGB Werte eines Pixels drei benachbarte Pixel verschiedener Farben zusammengefasst, wie in Abbildung 2.6 gezeigt ist [Wal05, S. 97]. Die RGB Information des grünen Pixels, das im schwarzen Dreieck liegt, lässt sich durch das rechts davon liegende rote Pixel und darunter liegende blaue Pixel vollständig reproduzieren. Eine durchgeführte Pixelwiederholung ist in Abbildung 2.7 (a) zu sehen.

Das Demosaicing ist ein zentraler Bestandteil der Verarbeitungspipeline von Sensordaten und hat einen großen Einfluss auf die Qualität der Aufnahme, da es störende und unerwünschte Ausprägungen, sogenannte Artefakte, erzeugen kann [Luk09, S. 13f.]. Vor allem einfache Demosaicing Algorithmen wie z. B. die Pixelwiederholung führen zu Zipper Effekten und verfälschten Farben, die überwiegend an Kanten sichtbar sind [CT06, S. 2] [LK04, S. 347]. Diese sind beispielhaft in Abbildung 2.7 (b) anhand der Pixelwiederholung veranschaulicht. Auch andere Artefakte wie Unschärfe und Aliasing sind möglich [Luk09, S. 14]. Die Wahl eines geeigneten Demosaicing Algorithmus ist somit eine wichtige Entscheidung, auf die man meistens nur wenig Einfluss hat, da Demosaicing Algorithmen für gewöhnlich in der Verarbeitungspipeline fest verankert sind.



Abbildung 2.7: Interpolierte Bildsensordaten durch die Pixelwiederholung, die zu Zipper Artefakten und verfälschten Farben führt

2.3 Dateiformate digitaler Aufnahmen

Digitale Aufnahmen lassen sich zur Archivierung, zum Austausch und zur weiteren Ver- und Bearbeitung speichern. Dazu existieren viele, meist spezifizierte Datei-

formate, die für unterschiedliche Anwendungszwecke konzipiert sind. Im Bereich der digitalen Fotografie lässt sich zwischen Dateiformaten für die Speicherung von bereits verarbeiteten Aufnahmen und Dateiformaten für die Speicherung von unverarbeiteten Sensordaten differenzieren [Luk09, S. 351]. Üblicherweise werden verarbeitete Aufnahmen im Tagged Image File Format (TIFF) oder als ein Joint Photographic Experts Group (JPEG) basiertes Dateiformat gespeichert. Im Gegensatz dazu stehen die Rohdatenformate.

Die Wahl eines geeigneten Dateiformates hängt vom weiteren Verwendungszweck der Aufnahme ab und liegt im Ermessen des Nutzers. Sie muss deswegen mit Bedacht getroffen werden. Im Folgenden werden wichtige Dateiformate zur Speicherung digitaler Aufnahmen erläutert. Da in Kapitel 3 der Arbeit ein umfassendes Verständnis von TIFF vorausgesetzt wird, wird dieses näher beschrieben.

2.3.1 Tagged Image File Format (TIFF)

Das TIFF ist ein Dateiformat, das vorwiegend für die Speicherung und den Austausch von pixelbasierten Bilddaten entwickelt wurde und seit 1992 unverändert in der Version 6.0 spezifiziert ist. In einer TIFF Datei lassen sich Farbbilder sowie weitere Bildtypen wie Graustufenbilder in unterschiedlichen Farbräumen ablegen. Des Weiteren unterstützt TIFF eine Vielzahl verschiedener Kompressionsverfahren, wobei auch eine unkomprimierte Speicherung der Bilddaten möglich ist. [Ado92, S. 4ff.] Die unkomprimierte Speicherung von Bilddaten erlaubt eine Farbtiefe von bis zu 32 Bit je Farbkomponente [Ado92, S. 30]. TIFF Dateien können deswegen im Vergleich zu anderen Bildformaten sehr groß sein, ermöglichen jedoch durch die vielen Farbabstufungen eine präzisere Farbwiedergabe.

TIFF zeichnet sich durch eine besonders flexible und erweiterbare Dateistruktur aus, die in Abbildung 2.8 exemplarisch dargestellt ist. Die TIFF Spezifikation dient daher als Grundlage für diverse andere Bildformatspezifikationen wie beispielsweise das Exchangeable Image File Format (Exif) (vgl. Kapitel 2.3.3) oder das DNG (vgl. Kapitel 3). Eine TIFF Datei besteht aus den nachfolgend beschriebenen Komponenten.

Header

Eine TIFF Datei beginnt mit einem acht Byte großen Header. Dieser definiert in den ersten zwei Bytes die Byte-Reihenfolge der Datei und identifiziert die Datei in den nachfolgenden zwei Bytes durch die magische Zahl 42 als eine TIFF Datei. Die letzten vier Bytes des Headers verweisen auf ein sogenanntes Image File Directory (IFD). Die Verweise innerhalb einer TIFF Datei werden stets als Offset vom Dateibeginn aus angegeben. [Ado92, S. 13]

Kapitel 2. Grundlagen

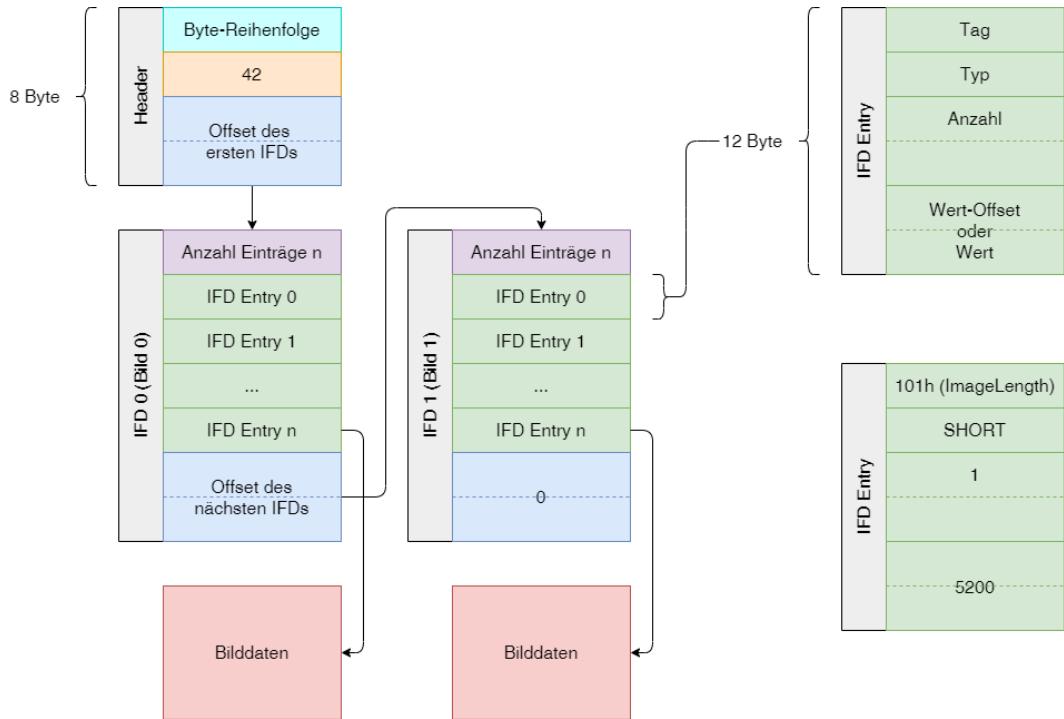


Abbildung 2.8: Die Dateistruktur einer TIFF Datei

Image File Directory

Ein IFD ist eine Liste von Einträgen, die Informationen über die gespeicherte Aufnahme enthalten. Ein solcher Eintrag wird IFD Entry oder IFD Field bezeichnet.

Die konkrete Anzahl der IFD Entries, die ein IFD beinhaltet, ist in den ersten zwei Bytes des IFDs festgelegt. Damit es möglich ist mehrere Bilder in einer TIFF Datei zu speichern, lassen sich IFDs verketten, indem die letzten vier Bytes eines IFDs auf ein anderes IFD verweisen. Folgt auf ein IFD kein weiteres IFD, ist in den letzten vier Bytes anstelle des Verweises der Wert 0 enthalten. [Ado92, S. 14]

Image File Directory Entry

Ein IFD Entry ist eine 12 Byte große Datenstruktur, die sogenannte TIFF Tags mit ihren zugehörigen Werten kapselt. Ein TIFF Tag ist im Grunde eine ID mit einer konkreten Bezeichnung, die den Inhalt eines IFD Entries beschreibt. Dabei kann es sich z. B. um die Bildhöhe, die Bildbreite oder um Verweise zu den Bilddaten handeln. Jedes Tag kann nur bestimmte Werte eines oder mehrerer Datentypen annehmen. Dazu spezifiziert TIFF insgesamt 12 unterschiedliche Datentypen.

Die ersten zwei Bytes eines IFD Entries beinhalten das TIFF Tag, dessen Datentyp in den nachfolgenden zwei Bytes definiert ist. Die nächsten vier Bytes enthalten die

2.3. Dateiformate digitaler Aufnahmen

Anzahl der Werte, die konkret in den letzten vier Bytes des IFD Entries aufgeführt werden. Können die Werte aufgrund ihres Datentyps oder aufgrund ihrer Anzahl nicht in den letzten vier Bytes platziert werden, ist darin stattdessen ein Verweis auf die Werte enthalten. [Ado92, S. 14ff.]

Wenn sich Daten nicht in der Struktur eines IFD Entries abbilden lassen, kann in einem IFD Entry ein Verweis auf eine beliebige Datenstruktur enthalten sein. [Ado92, S. 16] Dieses Prinzip wird beispielsweise von dem DNG Dateiformat genutzt, um komplexe und DNG-spezifische Datenstrukturen in einer DNG Datei einzubetten (vgl. Kapitel 3.2). Des Weiteren lassen sich dadurch sogenannte TIFF Trees realisieren. Bei TIFF Trees erfolgt die Verkettung von IFDs nicht über die letzten vier Byte eines IFDs, sondern über Verweise, die als IFD Entry durch das TIFF Tag *SubIFDs* angegeben sind [Ald93, S. 1f.].

Die konkreten TIFF Tags sind bis zur Nummer 32767 spezifiziert. Alle nachfolgenden Tags können frei gewählt werden und sind für den privaten Nutzen reserviert. [Ado92, S. 8] Beispielhaft sind einige wichtige Tags in Tabelle 2.1 mit ihrer Beschreibung gelistet.

Bilddaten

Die Bilddaten können sich an einer beliebigen Stelle in der TIFF Datei befinden und sind einem IFD zugeordnet. Die Zuordnung erfolgt über Verweise, die in einem IFD

Tag Name	Tag	Beschreibung
ImageWidth	256	Breite des Bildes in Pixel
ImageLength	257	Höhe des Bildes in Pixel
BitsPerSample	258	Farbtiefe je Komponente
Compression	259	Kompressionsmethode z. B. 1 für Unkomprimiert
PhotometricInterpretation	262	Farbraum z. B. 3 für RGB Bild, 5 für CMYK Bild
StripOffsets	273	Verweise zu allen Strips
SamplesPerPixel	277	Anzahl Komponenten je Pixel z. B. 3 bei einem RGB Bild
RowsPerStrip	278	Anzahl Zeilen je Strip
StripByteCounts	279	Anzahl Bytes je Strip (nach Komprimierung)
TileWidth	322	Anzahl Spalten je Tile
TileLength	323	Anzahl Zeilen je Tile
TileOffsets	324	Verweise zu allen Tiles
TileByteCounts	325	Anzahl Bytes je Tile (nach Komprimierung)

Tabelle 2.1: Wichtige TIFF Tags mit ihrer Beschreibung

Kapitel 2. Grundlagen

Entry hinterlegt sind. Bilddaten lassen sich in einer TIFF Datei entweder als Strips oder als Tiles ablegen. [Ado92, S. 19, 66] Abbildung 2.9 visualisiert beide Varianten.

Liegen die Bilddaten als Strips vor, enthält das zugehörige IFD ein IFD Entry mit dem Tag *StripOffsets*, das auf die einzelnen Strips verweist. Andernfalls liegen die Bilddaten als Tiles vor, die durch das Tag *TileOffsets* referenziert werden. Der Vorteil von Tiles gegenüber Strips liegt in der höheren Kompressionsrate sowie in einem effizienteren Bilddatenzugriff. Da Tiles immer die gleiche Größe haben müssen, werden Tiles, die sich am rechten und unteren Bildrand befinden und aus dem Bild hinausragen, mit Füllwerten erweitert. [Ado92, S. 66f.]

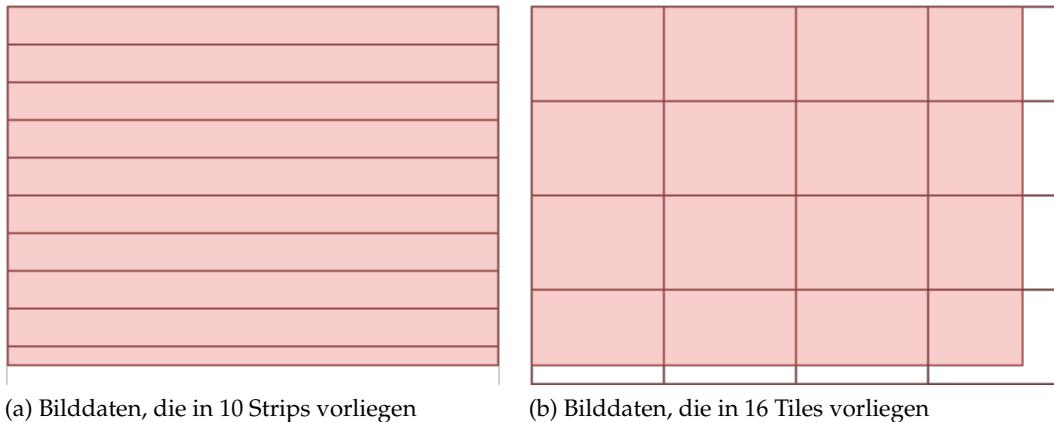


Abbildung 2.9: Als Strips und Tiles gespeicherte Bilddaten in einer TIFF Datei

2.3.2 Joint Photographic Experts Group (JPEG)

Das JPEG ist ein internationaler Standard, der Verfahren zur verlustbehafteten und verlustfreien Komprimierung von Bilddaten spezifiziert [CCI92, S. 1, 14]. JPEG ist das am häufigsten verwendete Dateiformat zur Speicherung digitaler Aufnahmen [Mia99, S. 35]. Es wird nach [W3T] auf 70.5 Prozent aller Webseiten genutzt. Digitale Aufnahmen im JPEG Format zeichnen sich vor allem durch ihre gute Bildqualität in Relation zur geringen Dateigröße aus [Wal05, S. 112]. Ein Vorteil von JPEG ist die konfigurierbare Kompressionsrate, die sich je nach gewünschter Bildqualität regulieren lässt [Wal91, S. 31]. Dadurch können digitale Aufnahmen um ein Vielfaches ihrer ursprünglichen Dateigröße verringert werden, was sie deutlich kleiner als TIFF Dateien macht. Aufgrund der hohen verlustbehafteten Kompression, die JPEG auf die Farbinformationen einer digitalen Aufnahme anwendet sowie der geringen Farbtiefe von 8 Bit je Farbkanal, sind JPEG Dateien für die Bearbeitung ungeeignet [Fra04, S. 3f.] [Mia99, S. 35].

Die JPEG Spezifikation trifft keine Aussage über das Dateiformat selbst. Es wird nicht beschrieben wodurch eine Datei für den Austausch zwischen Anwendungen charakterisiert sein muss, damit sie interpretiert werden kann. Aus diesem Grund

wurde das JPEG File Interchange Format (JFIF) Dateiformat entwickelt. [Mia99, S. 40] Das JFIF spezifiziert eine Syntax zum Austausch von JPEG komprimierten Bilddaten [Ham92, S. 1]. JPEG komprimierte Bilddaten werden deshalb häufig im JFIF Dateiformat gespeichert. Auch TIFF sowie das nachfolgend erläuterte Exif erlaubt das Speichern von JPEG komprimierten Bilddaten.

2.3.3 Exchangeable Image File Format (EXIF)

Das Exif ist ein Standard, welcher aktuell in der Version 2.32 vorliegt und Formate für digitale Aufnahmen sowie für Audio spezifiziert, die von digitalen Fotokameras aufgezeichnet wurden [Cam19, S. 4]. Da Audioformate für die vorliegende Arbeit keine Relevanz haben, werden nur die Bildformate erläutert. In einer Exif Datei lassen sich sowohl unkomprimierte Bilddaten gemäß der TIFF 6.0 Spezifikation (vgl. Kapitel 2.3.1) als auch JPEG komprimierte Bilddaten (vgl. Kapitel 2.3.2) speichern. In beiden Fällen wird die in der TIFF 6.0 Spezifikation definierte Dateistruktur verwendet. Darüber hinaus spezifiziert Exif eine Methode, um Vorschaubilder, sogenannte Thumbnails, in die Datei einzubetten. [Cam19, S. 8]

Die Exif Spezifikation erweitert die in der TIFF 6.0 Spezifikation definierten TIFF Tags durch kamerabezogene Exif Tags, die es ermöglichen aufnahmespezifische Daten in eine (Exif) Datei zu integrieren. Dabei kann es sich beispielsweise um die eingestellte Belichtungszeit oder die Blende handeln. Die kamerabezogenen Tags werden in einem gesonderten Exif IFD abgelegt, das die gleiche Struktur eines TIFF IFDs besitzt, jedoch keine Bilddaten referenzieren kann [Cam19, S. 8]. Exif Tags lassen sich somit auch in andere auf der TIFF Spezifikation basierende Dateiformate wie z. B. das DNG einbetten.

2.3.4 Rohdatenformate (RAW)

Die in den vorherigen Kapiteln vorgestellten Dateiformate dienen zur Speicherung von digitalen Aufnahmen, die bereits verarbeitet wurden. Auf den darin enthaltenen Bilddaten wurden somit destruktive Verarbeitungsschritte wie Demosaicing oder Kompression durchgeführt. Demgegenüber stehen Rohdatenformate, die im Nachfolgenden RAW Formate genannt werden.

RAW Formate sind proprietäre Dateiformate, welche die unverarbeiteten oder nur geringfügig verarbeiteten Sensordaten beinhalten [Ado19, S. 9]. Bevor der Inhalt einer RAW Datei als ein oben vorgestelltes Bildformat gespeichert oder auf dem Bildschirm angezeigt werden kann, muss dieser verarbeitet werden (vgl. Kapitel 2.4). Es ist beispielsweise notwendig ein Demosaicing durchzuführen, um die fehlenden Farbinformationen der darin enthaltenen Sensordaten zu rekonstruieren (vgl. Kapitel 2.2.2). Ein Vorteil der Verarbeitung liegt in der nachträglichen Steuerbarkeit von Aufnahmeparametern wie z. B. dem Weißabgleich [Fra04, S. 3]. Dadurch kann das visuelle Erscheinungsbild der digitalen Aufnahme optimiert werden. Neben

Kapitel 2. Grundlagen

den eigentlichen Sensordaten enthalten RAW Dateien deswegen weitere wichtige Daten. Diese lassen sich nach [ABF06, S. 7] wie in Abbildung 2.10 illustriert, in drei Kategorien einordnen:

- **Kameradaten:** Liegen meistens in Form von Exif Tags vor und können nicht geändert werden wie Belichtungszeit, Verschlusszeit, ISO (vgl. Kapitel 2.3.3)
- **Bilddaten:** Können während der RAW Konvertierung angepasst werden wie Weißabgleich, Schärfe, Rauschunterdrückung
- **Rohdaten:** Die eigentlichen nicht interpolierten Sensordaten

Im Gegensatz zu TIFF, JPEG/JFIF oder Exif sind proprietäre RAW Formate nicht offen spezifiziert. Wie Tabelle 2.2 zeigt existieren darüber hinaus unzählige RAW Formate verschiedener Hersteller, die sich nicht nur in ihrem Aufbau, sondern auch in den benötigten Verarbeitungsschritten unterscheiden [Luk09, S. 371]. Dies führt zu dem Problem der erschwerten herstellerübergreifenden Verarbeitung von RAW Dateien. Des Weiteren müssen Hersteller das Verarbeiten ihrer proprietären RAW Formate durch das Mitliefern einer entsprechenden Verarbeitungssoftware sicherstellen [Ado19, S. 9]. Bei Unverfügbarkeit der Software oder durch Obsoleszenz der proprietären RAW Datei kann die Verarbeitung jedoch unter Umständen nicht mehr garantiert werden [Ado19, S. 9f.] [Luk09, S. 371]. Proprietäre RAW Dateien sind deswegen für die Archivierung ungeeignet. Eine Sonderstellung nimmt das von Adobe spezifizierte und universelle RAW Format DNG ein, das im Fokus dieser Arbeit liegt und in Kapitel 3 genauer analysiert und erläutert wird.



Abbildung 2.10: Bestandteile einer proprietären RAW Datei, angelehnt an [ABF06, S. 7]

Hersteller	RAW Format
Adobe	DNG
Canon	CR2, CR3, CRW
Epson	ERF
FujiFilm	RAF
GoPro	GPR
Hasselblad	3FR
Kodak	DCR, K25, KDC
Leica	RWL, (DNG)
Minolta	MRW
Nikon	NEF, NRW
Olympus	ORF
Panasonic	RW2, RAW
Pentax	PEF, (DNG)
Phase One	IIQ
Samsung	SRW, (DNG)
Sony	ARW, ARQ, SR2, SRF

Tabelle 2.2: RAW Formate bekannter Hersteller [Har] [Ado20]

2.4 RAW Konvertierung

Bevor die Sensordaten, die von der Kamera aufgezeichnet wurden, als eine pixel-basierte Bilddatei gespeichert oder von einem Bildschirm wiedergegeben werden können, müssen sie einer Reihe von Verarbeitungsschritten unterzogen werden. Diese werden in Form einer Verarbeitungspipeline realisiert. Die Verarbeitung kann kameraintern während der Aufzeichnung der digitalen Aufnahme erfolgen z. B. wenn die Aufnahme direkt als ein JPEG basiertes Dateiformat gespeichert wird. Werden die Sensordaten hingegen als eine RAW Datei gespeichert, muss die Verarbeitung auf einen Computer ausgelagert werden. [Luk09, S. 7] Der Verarbeitungsprozess wird fortan als RAW Konvertierung bezeichnet.

Nachfolgend wird ein grober Überblick über die grundlegenden Verarbeitungsschritte der RAW Konvertierung gegeben. Diese werden in Kapitel 3.4 am Beispiel des DNG detailliert erläutert. Des Weiteren wird in diesem Kapitel der RAW Konverter beschrieben, der zur kameraexternen Verarbeitung von RAW Dateien dient.

2.4.1 Verarbeitungsschritte

Nach [RSYD05] müssen zur RAW Konvertierung die in Abbildung 2.11 visualisierten Verarbeitungsschritte durchgeführt werden, wobei das Pre- und Postprocessing mehrere Teilschritte umfassen kann. Die tatsächliche Reihenfolge kann sich jedoch von Hersteller zu Hersteller unterscheiden [RSYD05, S. 2]. Es ist beispielsweise

Kapitel 2. Grundlagen

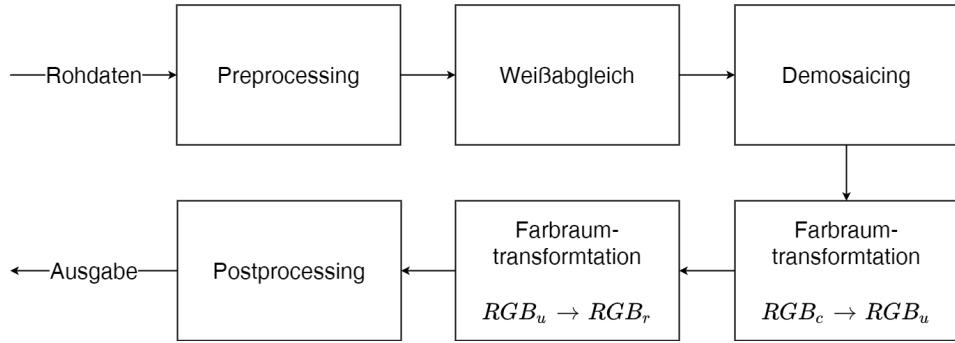


Abbildung 2.11: Verarbeitungsschritte der RAW Konvertierung, angelehnt an [RSYD05, S. 3]

auch möglich, den Weißabgleich und andere Verarbeitungsschritte erst nach dem Demosaicing durchzuführen [Luk09, S. 9f.].

Zu Beginn der Verarbeitung werden die Sensordaten während dem Preprocessing für die darauffolgenden Verarbeitungsschritte vorbereitet. Dabei werden Rauschen sowie andere Artefakte entfernt, welche durch den Bildsensor entstanden sind. Gängige Verarbeitungsschritte des Preprocessings sind u. a. die Korrektur defekter Pixel oder die Dunkelstromkompensation. [RSYD05, S. 8f.] Nach dem Preprocessing werden die vorverarbeiteten Sensordaten durch einen Weißabgleich angepasst, um von unterschiedlichen Lichtquellen verursachte Farbunterschiede zu kompensieren [CF06, S. 497]. Darauf folgend wird das Demosaicing als zentraler Verarbeitungsschritt der RAW Konvertierung durchgeführt, das in Kapitel 2.2.2 ausführlich erläutert wurde. Durch das Demosaicing liegen je Pixel drei Farbkomponenten vor, die sich jedoch in einem von der Kamera charakterisierten Farbraum befinden [RSYD05, S. 11]. Mittels mehrerer Farbraumtransformationen werden die kameraspezifischen Farbkomponenten in einen ausgabebasierten Farbraum wie beispielsweise sRGB überführt [RSYD05, S. 11ff.] [Luk09, S. 77f.]. Das abschließende Postprocessing befasst sich mit der Optimierung des Erscheinungsbildes der digitalen Aufnahme. Dazu gehört die Korrektur von Artefakten, die aufgrund vorhergehender Verarbeitungsschritte entstanden sind. [RSYD05, S. 13f.] Auch eine Tonwert- sowie eine Gammakorrektur kann während dem Postprocessing durchgeführt werden [Luk09, S. 78f.]. Die nun vorliegende digitale Aufnahme kann als Bilddatei gespeichert oder auf dem Bildschirm ausgegeben werden.

2.4.2 RAW Konverter

Ein RAW Konverter ist eine Anwendungssoftware, die zur kameraexternen Verarbeitung von RAW Dateien dient. Analog zur kamerainternen RAW Konvertierung führt ein RAW Konverter die im vorherigen Kapitel vorgestellten Verarbeitungsschritte durch. RAW Konverter bieten i. d. R. eine umfangreiche grafische Benutzeroberfläche.

2.4. RAW Konvertierung

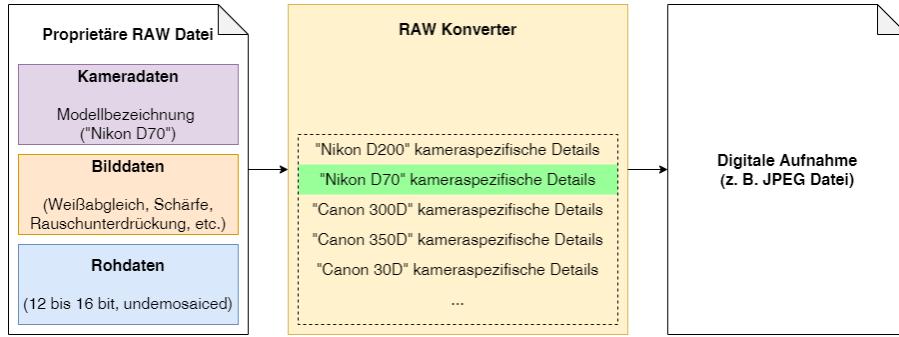


Abbildung 2.12: RAW Konvertierung einer proprietären RAW Datei durch einen RAW Konverter, angelehnt an [Pea10a]

che, die es durch Bedien- und Anzeigeelemente erlaubt, Aufnahmeeinstellungen wie beispielsweise den Weißabgleich zu regulieren und die Auswirkungen davon direkt zu betrachten. Im Vergleich zur kamerainternen RAW Konvertierung werden kameraexterne RAW Konverter auf einer leistungsstarken Hardware ausgeführt.

Damit ein RAW Konverter eine proprietäre RAW Datei verarbeiten und interpretieren kann, benötigt der RAW Konverter Detailinformationen über die Kamera, welche die Rohdaten aufgezeichnet hat. Diese kameraspezifischen Details sind im RAW Konverter integriert und können über die Modellbezeichnung der Kamera, die sich in den Metadaten der RAW Datei befindet, den Rohdaten zugeordnet werden. [Ado19, S. 10] [Pea10a] Dieser Prozess ist in Abbildung 2.12 dargestellt.



Abbildung 2.13: Typische Benutzeroberfläche eines RAW Konverters am Beispiel von RawTherapee [Raw]

Kapitel 2. Grundlagen

Beispielhaft ist in Abbildung 2.13 die grafische Benutzeroberfläche des freien RAW Konverters RawTherapee [Raw] veranschaulicht. Ein typischer RAW Konverter besteht i. d. R. aus einer live Vorschau (rot), einer Dateiverwaltung (blau), statischen Anzeigeelementen (grün) sowie den Bedienelementen (orange), die zur Anpassung der Aufnahmeparametern dienen.

2.4.3 JENIFFER

JENIFFER ist ein in der Programmiersprache Java implementierter und damit platformunabhängiger Open Source RAW Konverter [Wal05, S. 120]. Mit JENIFFER können RAW Dateien im NEF Format angezeigt und bearbeitet werden [WGGK05, S. 438].

Das Besondere an JENIFFER ist die Handhabung des Demosaicings, was es von anderen RAW Konverter unterscheidet. Während herkömmliche RAW Konverter einen oft unbekannten und nicht offengelegten Demosaicing Algorithmus verwenden, kann bei JENIFFER der Algorithmus vor dem Demosaicing ausgewählt werden [WGGK05, S. 438]. Es werden dabei sowohl nicht-adaptive als auch adaptive Demosaicing Algorithmen unterstützt (vgl. Kapitel 2.2.2). Der Dialog, der diese Auswahl erlaubt, ist in der Abbildung 2.14 aufgeführt.

JENIFFER bildet die Basis des im Rahmen der Arbeit entstehenden RAW Konverters JENIFFER2, wobei die nicht-adaptiven Demosaicing Algorithmen von JENIFFER übernommen wurden und die restliche Anwendung reimplementiert wurde.

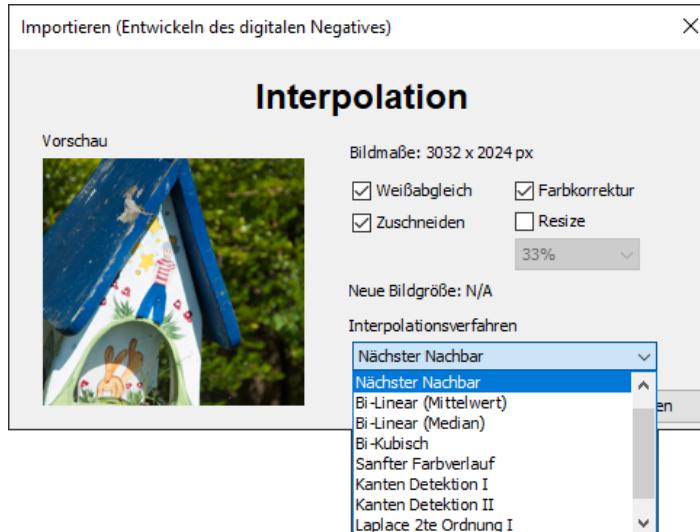


Abbildung 2.14: Interpolationsdialog des RAW Konverters JENIFFER

3 Digital Negative (DNG)

Das Kapitel 2.3.4 hat sich bereits mit RAW Formaten im Allgemeinen auseinander gesetzt und die Problematik proprietärer RAW Formate erläutert. Dieses Kapitel beschäftigt sich nun näher mit dem RAW Format DNG. Das DNG wurde von Adobe Inc. im September 2004 erstmals mit dem Ziel veröffentlicht, dem Problem eines fehlenden Standards für RAW Formate entgegenzuwirken und sich als nicht proprietäres RAW Format zu etablieren. Dementsprechend ist es, wie der Tabelle 2.2 entnommen werden kann, bei einigen bedeutenden Kameraherstellern im Einsatz. Zum Zeitpunkt der Arbeit befindet sich die DNG Spezifikation in der Version 1.5.0.0, die im Mai 2019 veröffentlicht wurde. Dies zeigt das Bestreben von Adobe, das DNG Format stetig zu verbessern und zu erweitern.

Zu Beginn des Kapitels werden wichtige Eigenschaften von DNG erläutert, welche das DNG als nicht proprietäres RAW Format charakterisieren. Anschließend werden die Bestandteile und der Aufbau einer typischen DNG Datei beschrieben sowie auf die Erzeugungsmöglichkeiten von DNG Dateien eingegangen. Zum Abschluss werden die Verarbeitungsschritte verdeutlicht, die zur Überführung einer DNG Datei in ein pixelbasiertes Bildformat durchgeführt werden müssen.

3.1 Eigenschaften

Proprietäre RAW Formate ähneln sich zwar inhaltlich, unterscheiden sich jedoch von Hersteller zu Hersteller in ihrem Aufbau, was den herstellerübergreifenden Umgang mit Rohdaten erschwert. Das DNG liegt der Idee zugrunde, die herstellerübergreifende Speicherung, Verarbeitung und Archivierung von Rohdaten durch ein spezifiziertes und flexibles RAW Format zu ermöglichen. Damit sich das DNG als ein solches nicht proprietäres RAW Format etablieren kann, besitzt es die nachfolgend identifizierten Eigenschaften.

TIFF basierend

Das DNG ist eine Erweiterung des im Kapitel 2.3.1 vorgestellten TIFF 6.0 Formats und mit dem Tag Image File Format / Electronic Photography (TIFF/EP) Standard kompatibel [Ado19, S. 10]. Der TIFF/EP Standard ist ebenfalls eine Erweiterung der TIFF 6.0 Spezifikation, die Tags zur Speicherung von Bilddaten spezifiziert, welche sich nicht über die TIFF Spezifikation abbilden lassen. Des Weiteren dokumentiert

Kapitel 3. Digital Negative (DNG)

die TIFF/EP Spezifikation eine Methode, um Vorschaubilder über eine Eltern-Kind Beziehung in eine TIFF Datei zu integrieren. [ISO00, S. IV] Dadurch lassen sich beispielsweise CFA basierte Rohdaten einschließlich Vorschaubildern speichern. Der Aufbau einer DNG Datei ist damit klar definiert.

In sich geschlossen

Eine DNG Datei ist ein in sich geschlossenes Konstrukt. Wie bereits in Kapitel 2.4.2 erläutert wurde, sind in einem RAW Konverter Detailinformationen verschiedener Kameras integriert, die zur Verarbeitung und zur Interpretation einer proprietären RAW Datei benötigt werden. Eine DNG Datei hingegen umfasst neben den typischen Bestandteilen einer RAW Datei (vgl. Kapitel 2.3.4) alle Informationen, die zur Verarbeitung der DNG Datei erforderlich sind. Die DNG Spezifikation definiert dazu eine Menge generischer und dadurch herstellerübergreifender Tags, die das Einbetten von kameraspezifischen Details in eine DNG Datei ermöglichen. Eines dieser Tags ist z. B. *ColorMatrix1*, das eine Transformationsmatrix enthält, welche einen kameraunabhängigen Farbraum in einen kamerahängigen Farbraum transformiert (vgl. Kapitel 3.4.5) [Ado19, S. 31]. Ein RAW Konverter kann diese auslesen und verarbeiten ohne interne Kenntnisse über die Kamera zu haben, welche die DNG Datei aufgezeichnet hat. [Ado19, S. 10] [Pea10a] Dieses Vorgehen ist in Abbildung 3.1 dargestellt.

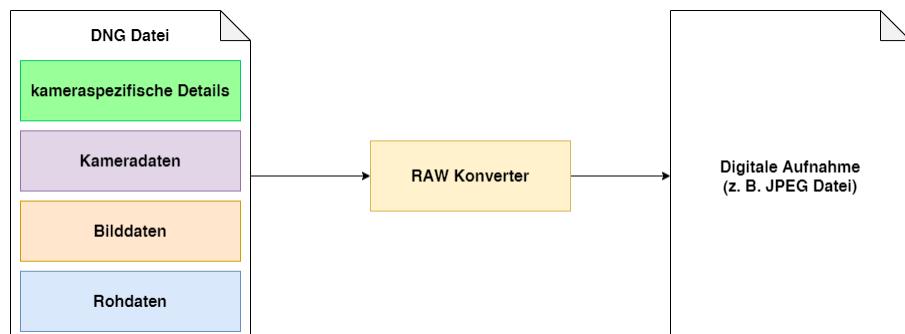


Abbildung 3.1: RAW Konvertierung einer DNG Datei durch einen RAW Konverter, angelehnt an [Pea10a]

Universell

In einer DNG Datei lassen sich die Rohdaten einer Vielzahl unterschiedlicher Bildsensoren mit einer Farbtiefe von bis zu 32 Bit speichern. Die DNG Spezifikation definiert dazu verschiedene CFA Farben und Anordnungen, die zur Abbildung einer spezifischen CFA Konfiguration miteinander kombiniert werden können. Des Weiteren ist das Speichern von Rohdaten möglich, bei denen das Bayer Muster

bereits interpoliert wurde oder die nicht auf einem CFA basieren, sondern alle Farbinformationen an einer Fotodiode erfassen [Ado19, S. 17ff.].

Metadatenorientiert

Das DNG Format sieht die Einbettung verschiedener Metadaten vor. Konkret lassen sich in einer DNG Datei TIFF bzw. TIFF/EP, Exif, Extensible Metadata Platform (XMP), das als International Press Telecommunications Council (IPTC) bekannte Information Interchange Model (IIM) und die DNG eigenen Metadaten integrieren [Ado19, S. 12]. Durch die Einbettung von XMP ist es z. B. möglich, Bearbeitungsschritte, die bei der RAW Konvertierung durchgeführt wurden, direkt in die DNG Datei zu schreiben, ohne sie als Filialdatei auslagern zu müssen [RA10, S. 157].

Komprimierbar

Die Rohdaten einer DNG Datei können verlustfrei komprimiert werden, um die Dateigröße gering zu halten. Dazu ist das DNG Format mit einer Menge unterschiedlicher Kompressionsverfahren kompatibel wie z. B. der verlustfreien JPEG Kompression. Es ist ebenfalls möglich, die Rohdaten in unkomprimierter Form zu speichern. Ferner unterstützt DNG die verlustbehaftete JPEG Kompression von Bilddaten, bei denen das Bayer Muster bereits interpoliert wurde. [Ado19, S. 18]

Archivierbar

Durch die offengelegte Spezifikation von DNG wird die langfristige Lesbarkeit von DNG Dateien berücksichtigt [Ado19, S. 10]. Ferner kann die Verarbeitung und Interpretierbarkeit von DNG Dateien sichergestellt werden, indem die DNG Datei alle zur Verarbeitung benötigten Daten wie kameraspezifische Details einbietet [Pea11]. Aufgrund der vorgesehenen Integration von XMP können außerdem Bearbeitungsschritte in einer DNG Datei mitgeführt werden, welche dadurch nicht verloren gehen. Darüber hinaus ist es möglich, eine proprietäre RAW Datei bei der kameraexternen Erzeugung einer DNG Datei (vgl. Kapitel 3.3.2) in diese einzubetten, um sie später ggfs. wiederherzustellen zu können [Ado19, S. 45]. Im Gegensatz zu proprietären RAW Dateien sind DNG Dateien somit für die Archivierung geeignet.

3.2 Dateiformat

Das DNG Dateiformat baut auf dem TIFF 6.0 Dateiformat auf und ähnelt diesem dadurch stark. Damit die Rohdaten einer DNG Datei ausreichend beschrieben werden können, erweitert die DNG Spezifikation die TIFF Spezifikation um rohdaten- und kameraspezifische Tags. Darüber hinaus können DNG Dateien neben den eigentlichen Rohdaten und Tags, weitere Daten wie z. B. Vorschaubilder beinhalten.

Kapitel 3. Digital Negative (DNG)

Die konkreten Bestandteile sowie der typische Aufbau einer DNG Datei werden im Nachfolgenden beschrieben.

3.2.1 Bestandteile

Entsprechend der TIFF 6.0 Dateistruktur (vgl. Kapitel 2.3.1) besteht eine DNG Datei aus einem Header, einer Menge von IFDs, die wiederum IFD Entries enthalten sowie aus den eigentlichen Rohdaten. Zusätzlich beinhaltet eine DNG Datei weitere optionale Datenstrukturen und Vorschaubilder. Konkret setzt sich eine DNG Datei folgendermaßen zusammen:

- Header
- RAW IFD mit Verweis auf die Rohdaten
- Thumbnail IFD mit Verweis auf ein primäres Vorschaubild (optional)
- Mehrere weitere Thumbnail IFDs, die je auf ein alternatives Vorschaubild verweisen (optional)
- Mehrere Opcode Listen innerhalb des RAW IFDs (optional)
- Private DNG Daten innerhalb des ersten IFDs (optional)
- Originale RAW Daten innerhalb des ersten IFDs (optional)
- Verweis innerhalb des ersten IFDs auf Exif Metadaten (optional)
- Verweis innerhalb des ersten IFDs auf XMP Metadaten (optional)
- Verweis innerhalb des ersten IFDs auf IPTC Metadaten (optional)
- Verweis innerhalb des ersten IFDs auf eine Liste von Kameraprofilen (optional)

Opcode Listen

Durch Opcodes können zusätzliche Verarbeitungsschritte wie z. B. Objektivkorrekturen für die spätere Verarbeitung in eine DNG Datei integriert sowie komplexe Verarbeitungsschritte von der Kamera auf einen DNG Reader ausgelagert werden [Ado19, S. 14]. Dazu werden Opcodes in Opcode Listen zusammengefasst, die zu unterschiedlichen Zeitpunkten der DNG Verarbeitung ausgeführt werden. Nach [Ado19, S. 66f.] kann eine DNG Datei drei solcher Listen beinhalten:

- *OpcodeList1*: Wird auf die Rohdaten unmittelbar nach dem Auslesen angewandt
- *OpcodeList2*: Wird auf die Rohdaten unmittelbar nach der Rohdatenzuordnung angewandt
- *OpcodeList3*: Wird auf die interpolierten Rohdaten unmittelbar nach dem Demosaicing angewandt

Jedes Opcode stellt eine eigene Datenstruktur dar, in der eine bestimmte Menge von Parametern kodiert ist. Wie diese Parameter zur Durchführung eines Opcodes genutzt werden müssen, ist für jedes Opcode gesondert spezifiziert. In der Regel sind zur Durchführung eines Opcodes komplexe und rechenintensive Berechnungen notwendig.

Private DNG Daten

Analog zu einer TIFF Datei können in einer DNG Datei proprietäre Daten in privaten Tags oder privaten IFDs platziert werden. Zur Sicherstellung, proprietäre Daten bei der Bearbeitung einer DNG Datei nicht zu überschreiben, empfiehlt DNG jedoch proprietäre Daten in dem extra dafür vorgesehenem Tag *DNGPrivateData* zu speichern. [Ado19, S. 13, 42]

Kameraprofile

Kameraprofile sind ein wichtiger Bestandteil einer DNG Datei. Sie enthalten einen Großteil der zur Verarbeitung einer DNG Datei benötigten Tags, von denen viele die bereits erwähnten kameraspezifischen Details sind. Jede DNG Datei beinhaltet ein primäres Kameraprofil, das sich als Ansammlung von Tags im ersten IFD einer DNG Datei befindet. Da Kameraprofile mit unterschiedlichen Referenzkameras erzeugt werden können, lassen sich zusätzlich beliebig viele weitere Kameraprofile in einer DNG Datei einbetten. Diese werden durch das Tag *ExtraCameraProfiles* referenziert. [Ado19, S. 13f., 52]

Originale RAW Daten

Bei der Konvertierung einer proprietären RAW Datei in eine DNG Datei (vgl. Kapitel 3.3.2) kann die proprietäre RAW Datei in die DNG Datei eingebettet und durch das Tag *OriginalRawFileData* referenziert werden [Ado19, S. 45]. Die originale RAW Datei lässt sich bei Bedarf extrahieren und wiederherstellen.

3.2.2 Aufbau

Die typische Dateistruktur einer DNG Datei ist in Abbildung 3.2 dargestellt. Zur Verkettung von IFDs nutzt DNG das Prinzip von TIFF Trees. Das erste IFD beinhaltet dazu das Tag *SubIFDs*, das beliebig viele weitere IFDs referenziert, die als SubIFD bezeichnet werden. Die Verkettung von IFDs über Verweise in den letzten vier Byte eines IFDs wird von DNG nicht unterstützt. [Ado19, S. 11]

Eine Empfehlung der DNG Spezifikation ist das Referenzieren eines niedrigaufgelösten, unkomprimierten und in Strips vorliegenden primären Vorschaubildes im ersten IFD, wie es die TIFF/EP Spezifikation nahelegt. Ein solches Bild lässt sich von jedem

Kapitel 3. Digital Negative (DNG)

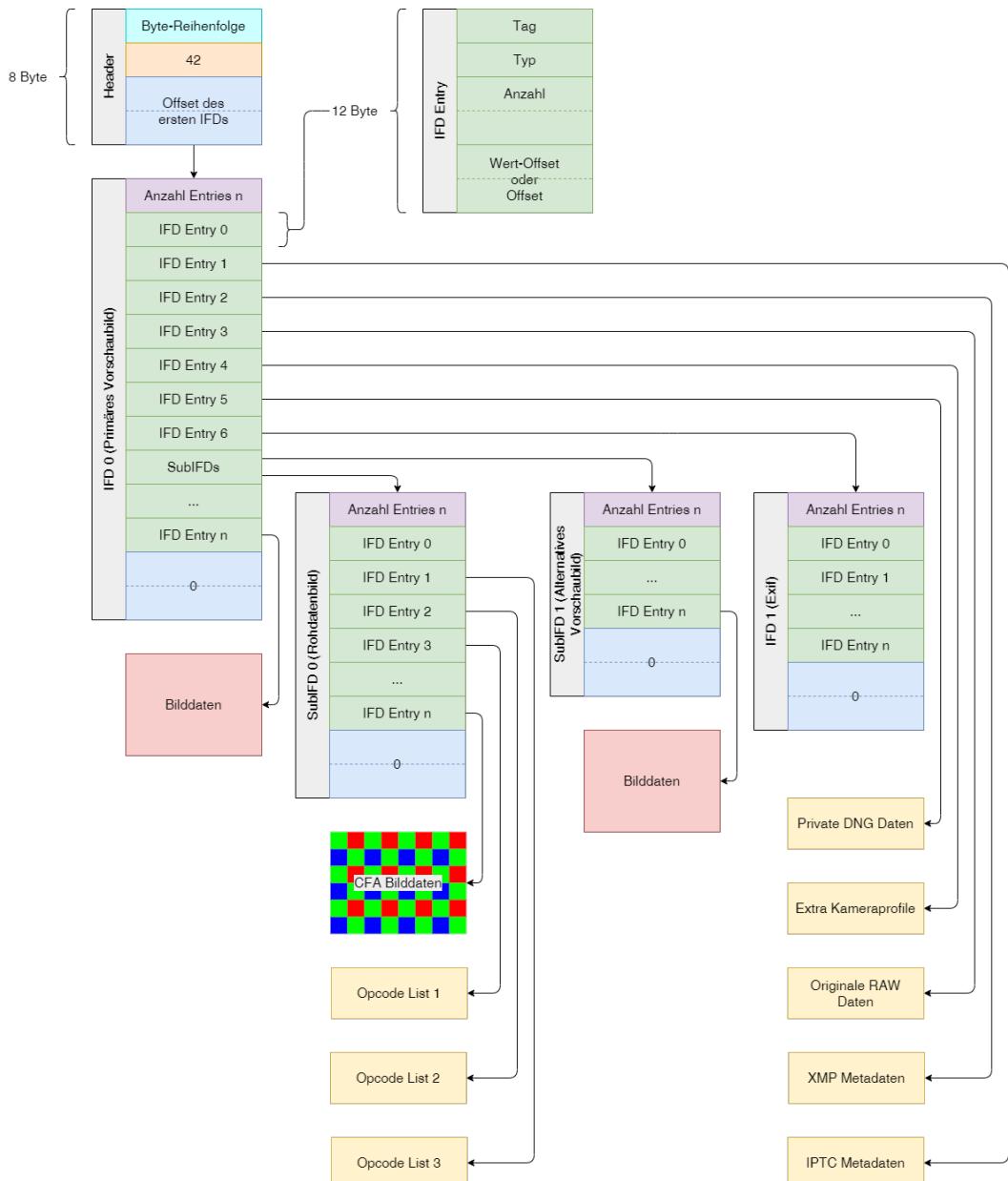


Abbildung 3.2: Typische Dateistruktur einer DNG Datei

TIFF Reader lesen und anzeigen. [Ado19, S. 11] [ISO00, S. 11] Dadurch kann bei unbekanntem Aufbau der DNG Datei zumindest das Anzeigen des Vorschaubildes garantiert werden.

Unter der Annahme, dass das erste IFD ein Vorschaubild enthält, befinden sich die Rohdaten in einem SubIFD, das vom ersten IFD referenziert wird. Neben Verweisen auf die Rohdaten, die als Strips oder Tiles vorliegen können (vgl. Kapitel 2.3.1),

enthält das SubIFD außerdem Tags zur Beschreibung der Rohdaten sowie drei optionale Opcode Listen. Ferner können weitere alternative Vorschaubilder als SubIFDs in der DNG Datei eingebettet sein [Ado19, S. 11].

Alle weiteren Daten werden stets vom ersten IFD referenziert. Bei diesen handelt es sich jeweils um eigene Datenstrukturen, die in der Abbildung 3.2 nur schematisch dargestellt sind (gelb) und aufgrund ihrer Komplexität nicht näher erläutert werden. Des Weiteren muss der in Abbildung 3.2 dargestellte Aufbau nicht verpflichtend eingehalten werden. Eine alternative DNG Datei kann beispielsweise die Rohdaten im ersten IFD referenzieren, das auf ein SubIFD mit einem Vorschaubild verweist. Ebenso kann eine DNG Datei valide sein, die keinerlei Vorschaubilder beinhaltet. Ein DNG-Reader muss demnach mit einer Vielzahl unterschiedlicher Dateistrukturen umgehen können.

3.3 Erzeugung von DNG Dateien

Zur Erzeugung von DNG Dateien existieren unterschiedliche Möglichkeiten. Im einfachsten Fall speichert eine Kamera die aufgezeichneten Sensordaten direkt als DNG Datei ab, sofern die Kamera das DNG Format unterstützt. Andernfalls kann eine DNG Datei mittels einer Computersoftware aus einer proprietären RAW Datei generiert werden. [Pea10a]

3.3.1 Kamerainterne Erzeugung

Bei der kamerainternen Erzeugung ist die Kamera für das Erzeugen der DNG Datei zuständig. Der Kamerahersteller nutzt dazu das DNG entweder als natives RAW Format oder als ein optionales RAW Format. Im zweiten Fall kann eine DNG Datei also alternativ zu einer proprietären RAW Datei erzeugt werden. [Pea10b]

Während der Erzeugung der DNG Datei schreibt die Kamerasoftware die typischen Bestandteile einer RAW Datei, bestehend aus den Kameradaten, den Bilddaten und den eigentlichen Rohdaten (vgl. Kapitel 2.3.4) entsprechend der DNG spezifizierten Dateistuktur in die DNG Datei. Zusätzlich werden die von DNG benötigten kameraspezifischen Details, die in der Kamera integriert sind, in die DNG Datei eingebettet [Pea10a]. Des Weiteren können Vorschaubilder sowie weitere Daten wie z. B. private DNG Daten in die DNG Datei integriert und die Rohdaten ggfs. zur Reduzierung der Dateigröße komprimiert werden. Einige Kameramodelle der Hersteller Leica, Casio, Ricoh, Samsung und Pentax sind in der Lage, DNG Dateien auf diese Weise zu erzeugen [Ado20].

3.3.2 Kameraexterne Erzeugung

Die kameraexterne Erzeugung einer DNG Datei erfolgt unter der Verwendung einer externen Computersoftware wie beispielsweise dem kostenfreien Adobe DNG Converter. Dieser ist unter Sicherstellung der Abwärtskompatibilität zu älteren RAW Formaten in der Lage, eine Vielzahl von (proprietären) RAW Dateien in eine DNG Datei zu konvertieren [Ado18].

Während der DNG Konvertierung wird eine RAW Datei vom DNG Konverter eingelesen und die darin enthaltenen Daten nach der DNG Spezifikation in eine DNG Datei geschrieben. Fernerbettet der DNG Konverter kameraspezifische Details in die DNG Datei ein, wie es in Abbildung 3.3 visualisiert ist. Dazu sind äquivalent zu einem RAW Konverter (vgl. Kapitel 2.4.2) im DNG Konverter kameraspezifische Details unterschiedlicher Kameramodelle integriert. Diese werden über die Modellbezeichnung der Kamera, die sich in den Metadaten der RAW Datei befindet, den Rohdaten zugeordnet [Pea10a].

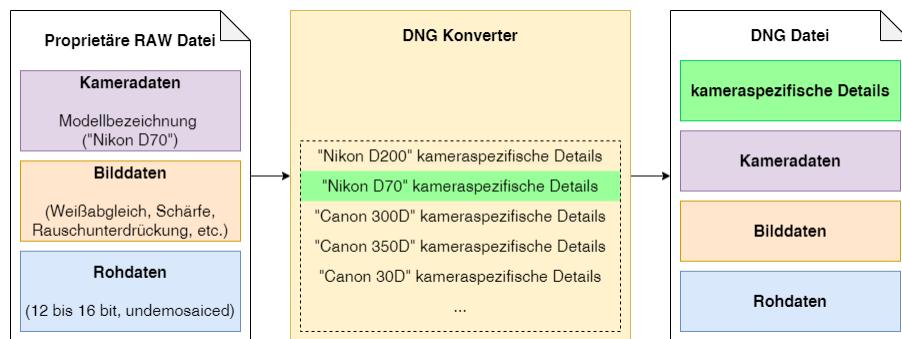


Abbildung 3.3: Kameraexterne Erzeugung einer DNG Datei durch einen DNG Konverter, angelehnt an [Pea10a]

Adobe DNG Converter

Unter Einbeziehung von Adobe Camera RAW (ACR), welches die Kernfunktionalität zur RAW Konvertierung von RAW Dateien implementiert, konvertiert der Adobe DNG Converter¹ (proprietäre) RAW Dateien in DNG Dateien. Ein wesentlicher Vorteil davon ist die Konfigurierbarkeit der zu erzeugenden DNG Datei. Beispielhaft sind in den Abbildungen 3.4 (a) und (b) die Voreinstellungsmasken des Adobe DNG Converters zu sehen. Diese bieten diverse Einstellungsmöglichkeiten, die vor der Konvertierung angepasst werden können. Konkret lassen sich folgende Einstellungen vornehmen:

- Kompatibilität: Diese bezieht sich auf die früheste ACR Version, welche die DNG Datei verarbeiten können soll. Bei der benutzerdefinierten Auswahl kann

¹Dieser ist für Windows sowie Mac OS verfügbar und kann unter dem folgenden Link heruntergeladen werden: <https://helpx.adobe.com/photoshop/using/adobe-dng-converter.html>

3.3. Erzeugung von DNG Dateien

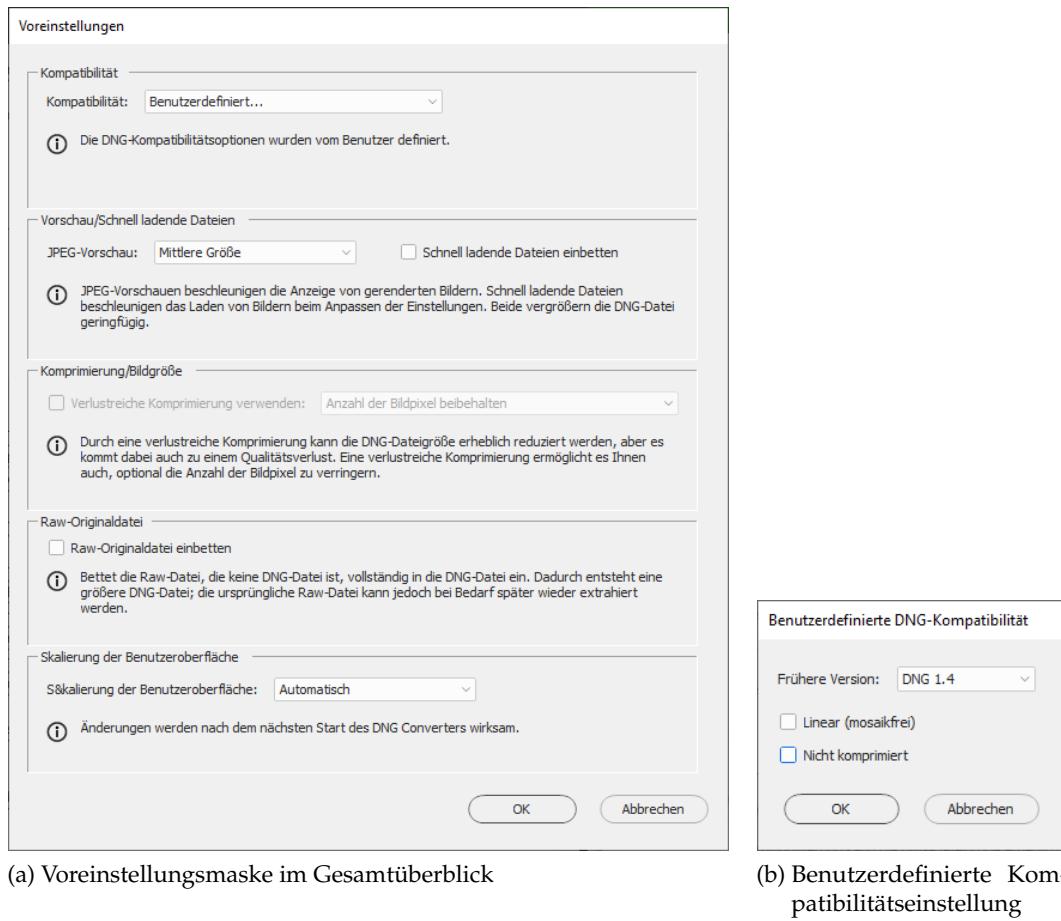


Abbildung 3.4: Voreinstellungsmasken des Adobe DNG Converters [Ado18]

aus der Einstellungsmaske der Abbildung 3.4 (b) die Versionsnummer der DNG Spezifikation gewählt werden, auf welcher die erzeugte DNG Datei basieren soll. Zusätzlich kann das Bayer Muster interpoliert und die verlustfreie Kompression abgewählt werden.

- **Vorschau/Schnell ladende Dateien:** Unter der JPEG Vorschau kann die Größe des primären Vorschaubildes gewählt werden, sofern ein Vorschaubild eingefügt werden soll. Durch schnell ladende Dateien werden mehrere alternative Vorschaubilder unterschiedlicher Auflösung in die DNG Datei eingebettet.
- **Komprimierung/Bildgröße:** Hierdurch können die Rohdaten verlustbehaftet komprimiert und optional in der Auflösung verringert werden. Das Bayer Muster wird dann interpoliert.
- **RAW-Originaldatei:** Ermöglicht das Einbetten der originalen RAW Datei in die erzeugte DNG Datei, die bei Bedarf wiederhergestellt werden kann.

Kapitel 3. Digital Negative (DNG)

Bereits erzeugte DNG Dateien lassen sich durch den Adobe DNG Converter gleichermaßen konfigurieren, indem sie unter den vorgenommenen Einstellungen zu einer neuen DNG Datei konvertiert werden. So können beispielsweise die unkomprimierten Rohdaten einer DNG Datei komprimiert oder ein Vorschaubild der DNG Datei hinzugefügt werden.

Zu beachten ist, dass der Adobe DNG Converter bei der Konvertierung einer (proprietären) RAW Datei in eine DNG Datei, jedes kodierte Pixel mit Nullen auf das nächste volle Byte auffüllt. Exemplarisch wird das laufende Beispiel betrachtet, das kameraintern erzeugt wurde. Darin wird jedes Pixel mit 14 Bit kodiert, was durch das Auslesen des Tags *BitsPerSample* ermittelt werden kann. Nachdem die DNG Datei mit dem Adobe DNG Converter zu einer neuen DNG Datei konvertiert wurde, ist jedes Pixel der neuen DNG Datei mit 16 Bit kodiert. Durch die zwei zusätzlichen Bits je Pixel steigt die Dateigröße zwar von ursprünglichen ~83MB auf ~91MB an, jedoch können die Pixel nach der Konvertierung byteweise ausgelesen und dekodiert werden, was zu einer einfacheren Implementierung und zu einer höheren Performance führen kann.

3.4 RAW Konvertierung

Wie eine proprietäre RAW Datei muss auch eine DNG Datei zunächst verarbeitet werden, bevor sie als pixelbasierte Bilddatei gespeichert oder auf dem Monitor wiedergegeben werden kann. Im Gegensatz zu einer proprietären RAW Datei sind alle Metadaten, die zur Verarbeitung der DNG Datei benötigt werden, in der DNG Datei integriert und offen spezifiziert. Viele der notwendigen Verarbeitungsschritte sind außerdem präzise beschrieben und deren Ausführungszeitpunkt definiert. Dennoch bietet die DNG Spezifikation Spielraum und schreibt nicht alle Verarbeitungsschritte vor. Die nachfolgend vorgestellte Verarbeitungspipeline ist deswegen eine von mehreren Möglichkeiten und nutzt als Referenz die in Kapitel 2.4.1 erläuterte Verarbeitungspipeline.

Abbildung 3.5 zeigt eine mögliche Pipeline zur Verarbeitung einer DNG Datei, die eine Vielzahl der spezifizierten DNG Tags berücksichtigt. Welche Verarbeitungsschritte tatsächlich durchgeführt werden müssen hängt von den vorliegenden Tags der DNG Datei ab. Am laufenden Beispiel müssen die in der Abbildung 3.5 grün gekennzeichneten Schritte durchlaufen werden, da für diese in der DNG Datei die benötigten Tags enthalten sind. Aufgrund der großen Menge an Verarbeitungsschritten werden in den nachfolgenden Kapiteln nur die grün markierten Schritte erläutert. Über die Anderen wird in Kapitel 3.4.6 ein kurzer Überblick gegeben.

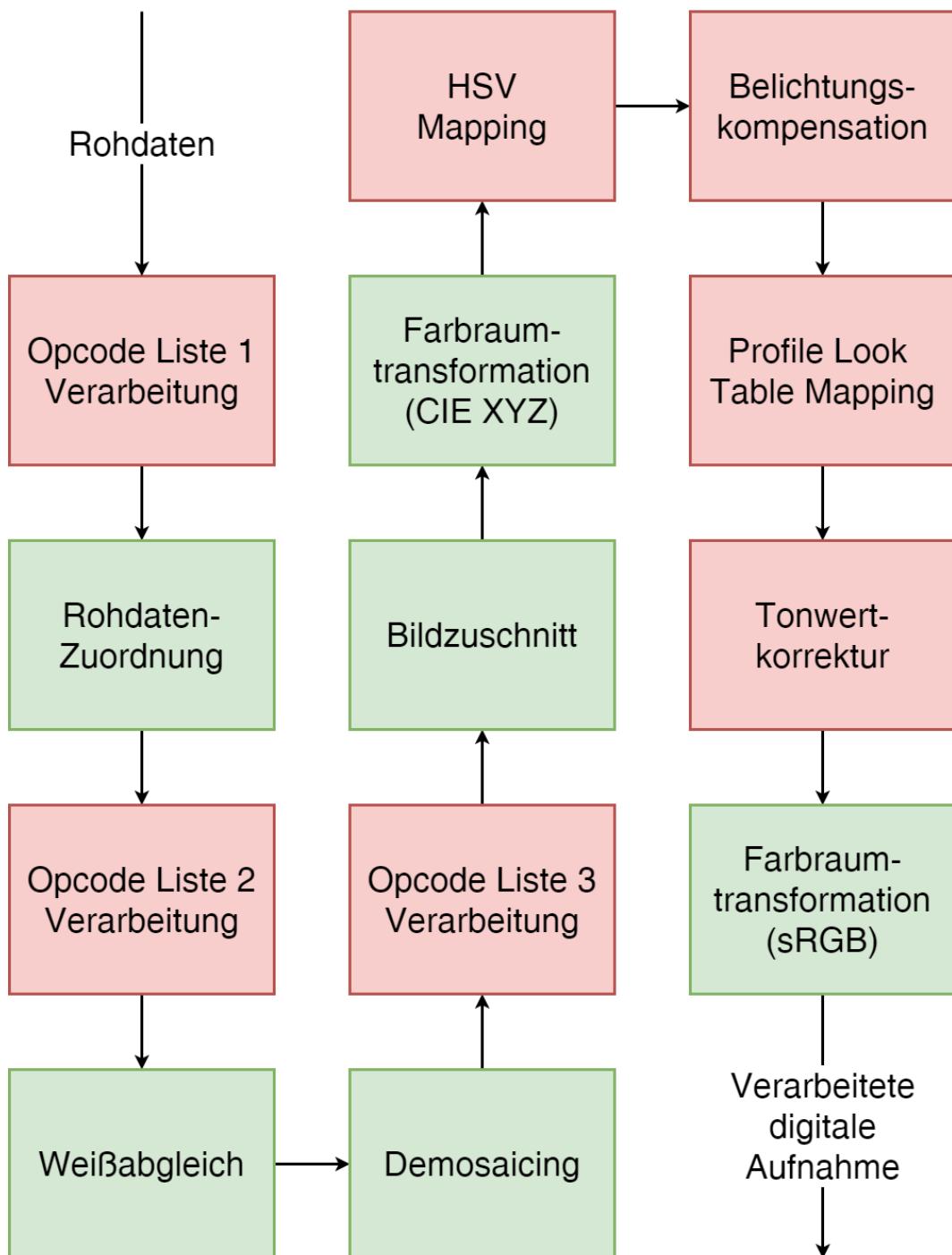


Abbildung 3.5: Verarbeitungsschritte einer DNG Datei

3.4.1 Rohdatenzuordnung

Die Rohdaten einer DNG Datei liegen zunächst in einem vom Dunkelstrom, von der Sättigung sowie vom Sensorrauschen abhängigen Wertebereich und wurden ggfs. zur Reduzierung der Dateigröße transformiert. Für die weitere Verarbeitung muss die Transformation ggfs. rückgängig gemacht und die Rohdaten auf einen fest definierten Wertebereich abgebildet werden. Dies erfolgt durch die Rohdatenzuordnung.

Bei der Rohdatenzuordnung werden die in der DNG Datei gespeicherten Rohdaten „linearen Referenzwerten“ [Ado19, S. 83] zugeordnet. Diese liegen im Wertebereich zwischen 0.0 (kein Licht) und 1.0 (maximale Sättigung). Die Rohdaten lassen sich gemäß Abbildung 3.6 durch die nachfolgenden vier Teilschritte auf lineare Referenzwerte abbilden. [Ado19, S. 83]

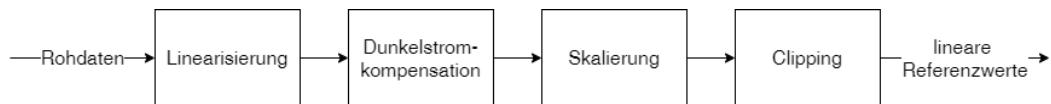


Abbildung 3.6: Zuordnung von Rohdaten zu linearen Referenzwerten

Linearisierung

Um für die Speicherung der Rohdaten höhere Kompressionsraten zu erreichen, können die Rohdaten auf einen kleineren, nicht-linearen Wertebereich abgebildet werden [Ado19, S. 25]. Betrachtet wird beispielsweise eine digitale Aufnahme mit 16 Bit Farbtiefe. Die Rohdaten dieser Aufnahme müssen mit 16 Bit kodiert werden, selbst wenn aus den $2^{16} = 65536$ möglichen Werten nur 256 Werte verwendet werden. Ein einfaches Runterskalieren der 16 Bit kodierten Werte auf $\log_2 256 = 8$ Bit kodierte Werte, würde bei eng beieinanderliegenden Werten zu einem Informationsverlust führen. Stattdessen könnten die tatsächlichen 16 Bit kodierten Werte in einer Lookup-Tabelle (LUT) verwaltet und von 8 Bit kodierten Werten referenziert werden. Dieses Prinzip setzt DNG durch das Tag *LinearizationTable* um.

Werden Rohdaten einer DNG Datei in einer LUT gespeichert, beinhaltet die DNG Datei das Tag *LinearizationTable*, welches die tatsächlichen Rohdatenwerte enthält. Die als Rohdaten gespeicherten Werte indexieren die tatsächlichen Rohdatenwerte innerhalb der *LinearizationTable*. Ist in der DNG Datei das Tag *LinearizationTable* nicht vorhanden, können die Rohdaten direkt ausgelesen werden.

Dunkelstromkompenstation

Unabhängig davon, ob Licht auf einen Bildsensor fällt, zeichnet ein Bildsensor an jeder Fotodiode einen sogenannten Dunkelstrom auf, der durch Wärme erzeugte Elektronen entsteht. Dunkelstrom spiegelt sich in einem Offset der Rohdaten wieder

und muss deshalb zunächst gemessen und anschließend von den Rohdaten subtrahiert werden. Dieses Verfahren wird als Dunkelstromkompensation bezeichnet. Eine gängige Methode zur Messung des Dunkelstroms ist das Abdunkeln eines Randbereichs des Bildsensors mit einer undurchsichtigen Maske. An dem maskierten Bereich wird lediglich der Dunkelstrom erfasst und für die Dunkelstromkompensation genutzt. [RSYD05, S. 9]

In einer DNG Datei kann der Dunkelstrom auf unterschiedliche Weise angegeben werden. Die direkte Angabe ist verpflichtend und erfolgt über das Tag *BlackLevel*, das zeilen- und spaltenweise mehrere Werte beinhalten kann, wobei die Anzahl an Zeilen und Spalten im Tag *BlackLevelRepeatDim* vermerkt ist. Dadurch ist es z. B. möglich, für jede Farbe eines Bayer Filters einen eigenen Dunkelstrom anzugeben. Des Weiteren kann der Dunkelstrom für eine ganze Bildzeile bzw. Bildspalte als *BlackLevelDeltaV* bzw. *BlackLevelDeltaH* Tag aufgeführt werden. Sofern maskierte Bereiche vor der Speicherung der Rohdaten nicht entfernt wurden, ist es zusätzlich möglich, eine Menge von maskierten Bereichen in dem Tag *MaskedAreas* anzugeben. Diese lassen sich zur manuellen Berechnung des Dunkelstroms heranziehen. Der aktive Bildsensorbereich, für den die Dunkelstromkompensation durchgeführt werden muss, wird außerdem in dem Tag *ActiveArea* definiert. Wurde von der Kamera bereits eine Dunkelstromkompensation durchgeführt, kann als Standardwert für das *BlackLevel* Tag der Wert 0 angenommen werden.

Skalierung

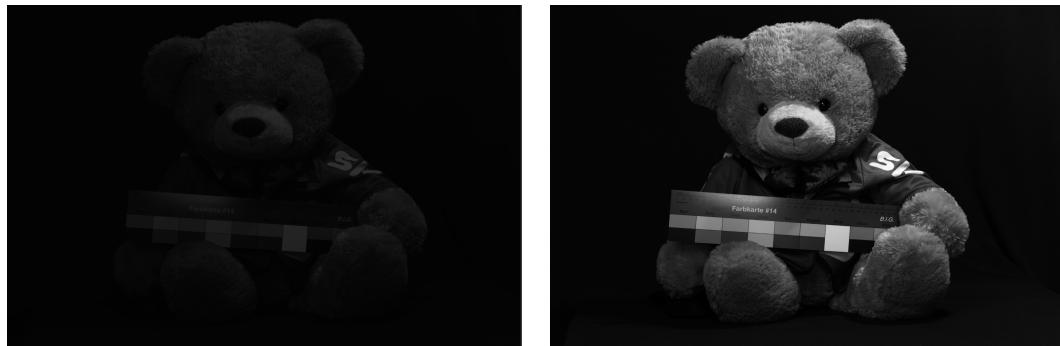
Nach der Dunkelstromkompensation müssen die Rohdaten auf den Wertebereich $[0, 1]$ skaliert werden [Ado19, S. 83]. Dazu wird die Sättigung benötigt, die den maximal möglichen Wert angibt. Diese befindet sich in dem Tag *WhiteLevel*. Zur Skalierung wird jeder Rohdatenwert mit einem Skalierungsfaktor c multipliziert, der sich nach [Ado19, S. 83] gemäß der Formel 3.1 berechnen lässt. Dabei gibt wl die Sättigung an und bl den Dunkelstrom, der wie oben beschrieben in Abhängigkeit seines Vorliegens mehrere Werte umfassen kann.

$$c = \frac{1}{wl - \max(bl_1, bl_2, \dots, bl_n)} \quad (3.1)$$

Clipping

Aufgrund von Sensorrauschen können die skalierten Rohdaten außerhalb des möglichen Wertebereichs von $[0, 1]$ liegen. Sie müssen deswegen auf diesen Wertebereich geclipped werden, bevor sie weiterverarbeitet werden können. [Sum14, S. 7]

Eine durchgeführte Rohdatenzuordnung ist unter Einbeziehung der in Tabelle 3.1 stehenden Tags in Abbildung 3.7 (b) zu sehen. Im Vergleich zur Abbildung 3.7 (a), das die Rohdaten vor der Rohdatenzuordnung zeigt, haben die Rohdaten nach der Rohdatenzuordnung aufgrund der Skalierung und Normalisierung auf den Wertebereich [0,1] einen deutlich höheren Kontrast.



(a) Direkt ausgelesene Rohdaten der DNG Datei (b) Durchgeführte Rohdatenzuordnung

Abbildung 3.7: Rohdatenzuordnung im Vorher - Nachher Vergleich am laufenden Beispiel

Tag	Wert
BlackLevelRepeatDim	2 2
BlackLevel	510 510 511 511
WhiteLevel	11500
ActiveArea	0 0 5632 8392

Tabelle 3.1: Zur Durchführung der Rohdatenzuordnung relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)

3.4.2 Weißabgleich

Das menschliche visuelle System ist durch die sogenannte chromatische Adaption in der Lage, Farbunterschiede verschiedener Lichtquellen zu kompensieren. Dadurch erscheint die Farbe eines Gegenstandes gleich, unabhängig von der Lichtquelle, mit welcher der Gegenstand beleuchtet wird. Diese Eigenschaft wird Farbkonstanz genannt. [Ber19, S. 21] Damit die Farbe eines Gegenstandes auch auf einer digitalen Aufnahme konstant wirkt, muss die chromatische Adaption auf die Kamera übertragen werden. Dies erfolgt durch den Weißabgleich [CF06, S. 497].

Jede Lichtquelle hat ihre eigene Farbcharakteristik, welche die Farbe der Aufnahme beeinflusst und durch die Farbtemperatur in der Einheit Kelvin (K) angegeben wird [Wal05, S. 193, 199]. Damit ein Weißabgleich durchgeführt werden kann, muss demnach zunächst die Lichtquelle der Szenenbeleuchtung und deren Farbtemperatur

bestimmt werden. Diese kann, sofern sie bekannt ist, entweder explizit ausgewählt oder implizit von der Kamera ermittelt werden. [Luk09, S. 280] Nach der Ermittlung der Farbtemperatur müssen die RGB Werte der Aufnahme zur Durchführung des Weißabgleichs gewichtet werden. Durch die Gewichtung wird das Verhältnis des Rot-, Grün- und Blauanteils an einem Pixel geändert, sodass die RGB Werte einer neutralen Stelle gleich sind [Büh04, S. 105]. Die Gewichtung der RGB Werte R , G und B erfolgt nach [Wal05, S. 199] gemäß der Formel 3.2 mit den Koeffizienten g_R , g_G und g_B , die von der Farbtemperatur abhängig sind. R_W , G_W sowie B_W entsprechen den RGB Werten nach dem Weißabgleich.

$$\begin{aligned} R_W &= g_R \cdot R \\ G_W &= g_G \cdot G \\ B_W &= g_B \cdot B \end{aligned} \quad (3.2)$$

In einer RAW Datei wird der Weißabgleich nicht direkt auf die Rohdaten angewendet. Stattdessen wird die ermittelte Farbtemperatur oder die ermittelten RGB Koeffizienten in den Metadaten der RAW Datei gespeichert. Dadurch kann der Weißabgleich zum Zeitpunkt der RAW Konvertierung durchgeführt und ggfs. optimiert werden. In einer DNG Datei sind die RGB Koeffizienten in dem Tag *AsShotNeutral* enthalten. Alternativ kann der Weißabgleich als xy Chromatizitätswerte in dem Tag *AsShotWhiteXY* angegeben werden. Diese werden in Kapitel 3.4.5 erläutert.

Abbildung 3.8 (b) zeigt den durchgeführten Weißabgleich am laufenden Beispiel mit den RGB Koeffizienten ~ 0.515 (R), ~ 1.0 (G) und ~ 0.656 (B). Im Vergleich zur Aufnahme in Abbildung 3.8 (a), welche die Rohdaten vor dem Weißabgleich zeigt, wirkt die Weißfläche der Farbkarte (rot eingerahmt) nahezu vollständig weiß. Aufgrund eines nicht exakt eingestellten Weißabgleichs kann es hier zu minimalen Abweichungen kommen. Die Abbildung 3.9 visualisiert außerdem die Auswirkung des Weißabgleichs auf die finale Aufnahme.

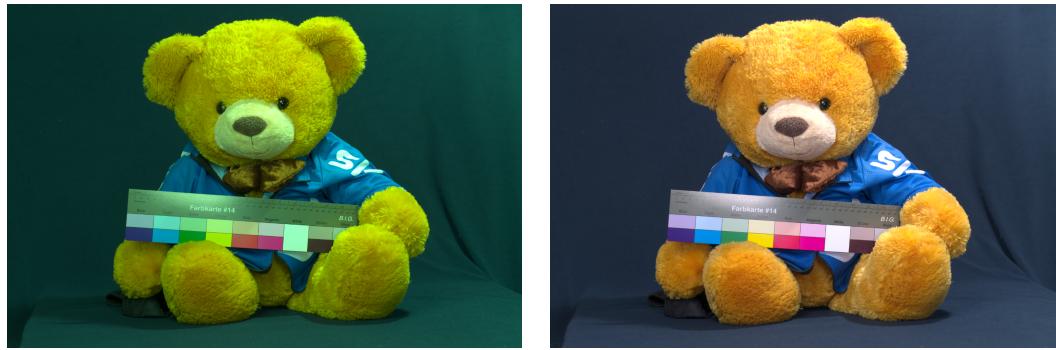


(a) Rohdaten nach der Rohdatenzuordnung, vor dem Weißabgleich



(b) Durchgeführter Weißabgleich

Abbildung 3.8: Weißabgleich im Vorher - Nachher Vergleich am laufenden Beispiel



(a) Finale Aufnahme ohne Durchführung eines Weißabgleichs
 (b) Finale Aufnahme mit Durchführung eines Weißabgleichs

Abbildung 3.9: Auswirkung des Weißabgleichs auf die finale Aufnahme

3.4.3 Demosaicing

Nach den vorhergehenden Verarbeitungsschritten liegt je Rohdatenpixel weiterhin lediglich eine Farbinformation vor. Durch das Demosaicing (vgl. Kapitel 2.2.2) werden die beiden fehlenden Farbinformationen wiederhergestellt und der strukturelle Inhalt der Aufnahme rekonstruiert [Luk09, S. 13].

Bevor das Demosaicing an einer DNG Datei durchgeführt werden kann, muss der Inhalt der DNG Datei geprüft werden. Enthält die DNG Datei auf einem CFA basierende Rohdaten, ist das Tag *PhotometricInterpretation* mit dem Wert 32803 (CFA) enthalten. Andernfalls wurde das Bayer Muster bereits interpoliert oder die Rohdaten wurden von einem Bildsensor erfasst, der alle Farbinformationen an einer Fotodiode aufzeichnet [Ado19, S. 19]. Im Falle von Rohdaten, die auf einem CFA basieren, muss das konkrete CFA Muster in dem Tag *CFAPattern* aufgeführt werden, wobei standardmäßig von einem Bayer Filter ausgegangen werden kann. Durch das Tag *CFAPatternRepeatDim* lässt sich dem *CFAPattern* Tag zusätzlich eine Dimension zuweisen. Sollten die Rohdaten nicht auf einem Bayer Filter beruhen, müssen über die Tags *CFAPlaneColor* sowie *CFALayout* die Farben der Filter und deren Anordnung aufgeführt werden. Für Bildsensoren, die keine quadratischen Fotodioden besitzen, kann außerdem durch das Tag *DefaultScale* eine Skalierung angegeben werden.

Tag	Wert
<i>PhotometricInterpretation</i>	32803 (CFA)
<i>CFALayout</i>	1 (Rechteckig)
<i>CFAPattern</i>	1 2 0 1 ([Grün, Blau][Rot, Grün])

Tabelle 3.2: Zur Durchführung des Demosaicings relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)



(a) Rohdaten nach dem Weißabgleich, vor dem Demosaicing

(b) Durchgeführtes Demosaicing durch die bikubische Interpolation

Abbildung 3.10: Demosaicing im Vorher - Nachher Vergleich am laufenden Beispiel

Abbildung 3.10 (b) zeigt ein durchgeführtes Demosaicing durch die bikubische Interpolation unter Zuhilfenahme der in Tabelle 3.2 aufgeführten Tags. Die fehlenden Farbinformationen wurden aus den Rohdaten der Abbildung 3.10 (a) interpoliert. Je Pixel liegen nun die vollen RGB Informationen vor, was zu einer Farbwiedergabe der Aufnahme führt.

3.4.4 Bildzuschnitt

Demosaicing Algorithmen interpolieren die Farbwerte fehlender Farbkomponenten aus den benachbarten Pixeln. Herausfordernd sind dabei die Bildränder. An diesen stehen nicht genügend Nachbarschaftspixel zur Verfügung, um die fehlenden Farbkomponenten eines Randpixels zuverlässig zu bestimmen. Um Artefakte am Bildrand zu verhindern, besitzt ein Bildsensor häufig aktive Dummypixel, welche den aktiven Sensorbereich erweitern und somit die fehlenden Nachbarschaftspixel bereitstellen. [Nak06, S. 87f.] Da die Dummypixel nach dem Demosaicing nicht mehr benötigt werden, müssen sie für die darauffolgenden Verarbeitungsschritte aus den interpolierten Bilddaten entfernt werden.

In einer DNG Datei wird der aktive Sensorbereich inklusive Dummypixel in dem Tag *ActiveArea* angegeben. Die ersten beiden Werte spezifizieren die Lage des aktiven Bereiches innerhalb der Aufnahme. Die nachfolgenden beiden Werte geben die Höhe sowie die Breite des aktiven Bereiches an. Relativ zum aktiven Bereich erfolgt über das Tag *DefaultCropOrigin* die Angabe einer Koordinate, welche die Startposition des auszuschneidenden Bereiches kennzeichnet. Die konkrete Breite sowie Höhe des Bereiches, der ausgeschnitten werden soll, ist in dem Tag *DefaultCropSize* enthalten.

Die Beziehung der einzelnen Bereiche ist in Abbildung 3.11 anhand der in Tabelle 3.3 aufgeführten Tags am laufenden Beispiel dargestellt.

Kapitel 3. Digital Negative (DNG)

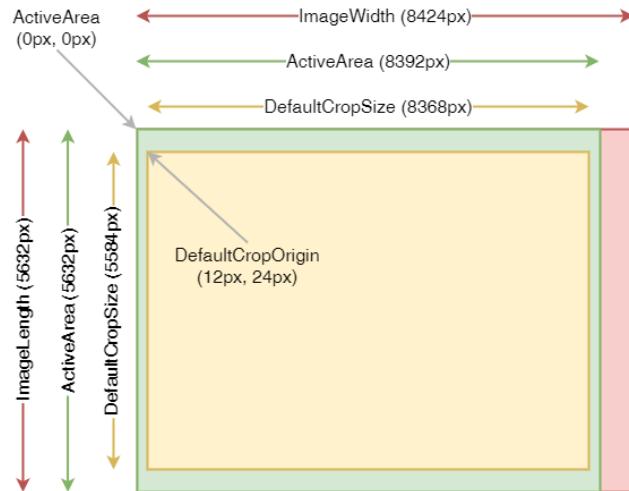


Abbildung 3.11: Beispielhafte Aufteilung der einzelnen Bereiche der Rohdaten einer DNG Datei

Tag	Wert
ImageWidth	8424
ImageLength	5632
ActiveArea	0 0 5632 8392
DefaultCropOrigin	12 24
DefaultCropSize	8368 5584

Tabelle 3.3: Zur Durchführung des Bildzuschnitts relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)

3.4.5 Farbraumtransformation

Die Rohdaten einer digitalen Aufnahme werden von einer Kamera in einem Farbraum aufgezeichnet, der durch die spektrale Empfindlichkeit des Bildsensors definiert ist [Luk09, S. 76]. Damit eine digitale Aufnahme von einem Ausgabegerät wie z. B. einem Monitor korrekt wiedergegeben kann, muss sie jedoch in einem vom Ausgabegerät geforderten Farbraum wie z. B. sRGB vorliegen. Die Rohdaten müssen deswegen durch eine lineare Transformation in einen solchen Farbraum überführt werden [Sum14, S. 3]. Dazu werden die RGB Werte des bildsensorabhängigen Kamerafarbraums i. d. R. zuerst in den CIE XYZ Farbraum und anschließend in einen ausgabebasierten Zielfarbraum transformiert [Luk09, S. 77f.].

Farbraumtransformation CIE XYZ

Der CIE XYZ Farbraum ist ein standardisierter Farbraum, welcher auf der menschlichen Farbwahrnehmung beruht. In diesem Farbraum lassen sich alle sichtbaren

Farben durch die drei Farbvalenzen X , Y und Z beschreiben, wobei Y ein Maß für die Luminanz (Helligkeit) ist. Mittels der XYZ Werte werden die Chromatizitätswerte x , y und z bestimmt, welche die Lage einer Farbe in einem zweidimensionalen Koordinatensystem angeben und dadurch die Farbart einer Farbe eindeutig kennzeichnen. Nach [BB16, S. 342] können die Chromatizitätswerte gemäß der Formeln 3.3 berechnet werden, wobei z durch die Bedingung $x + y + z = 1$ aus x und y ermittelt werden kann und damit nicht benötigt wird. [BB16, S. 341ff.]

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \\ z &= \frac{Z}{X+Y+Z} \end{aligned} \quad (3.3)$$

Ferner werden durch CIE sogenannte Normlichtarten spezifiziert, die durch eine feste Farbtemperatur charakterisiert sind. Die Normlichtart D50 hat beispielsweise eine Farbtemperatur von 5000K und entspricht dem natürlichen Sonnenlicht. Normlichtarten dienen u. a. zur Definition von Referenzweißpunkten, deren Bedeutung später erläutert wird. [BB16, S. 344f.]

Nach [Luk09, S. 77] können die RGB Werte des Kamerafarbraums in den CIE XYZ Farbraum durch die Multiplikation mit der Transformationsmatrix $A_{XYZ|Cam}$ entsprechend der nachfolgenden Formel transformiert werden:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} R_{Cam} \\ G_{Cam} \\ B_{Cam} \end{pmatrix} = A_{XYZ|Cam} \cdot \begin{pmatrix} R_{Cam} \\ G_{Cam} \\ B_{Cam} \end{pmatrix} \quad (3.4)$$

Zur Bestimmung der Transformationsmatrix $A_{XYZ|Cam}$ muss für eine DNG Datei zuerst die Transformationsmatrix $A_{Cam|XYZ}$ ermittelt werden, die XYZ Werte des CIE XYZ Farbraums in den Kamerafarbraum überführt. Wie Formel 3.5 zeigt bildet $A_{Cam|XYZ}$ somit das Inverse der Transformationsmatrix $A_{XYZ|Cam}$. Die Transformationsmatrix $A_{Cam|XYZ}$ steht dabei nicht unmittelbar zur Verfügung, sondern muss aus einer Menge unterschiedlicher Matrizen bestimmt werden, welche sich als Tags im primären Kameraprofil oder zusätzlich in beliebig vielen weiteren Kameraprofilen (vgl. Kapitel 3.2.1) der DNG Datei befinden.

$$A_{XYZ|Cam} = A_{Cam|XYZ}^{-1} \quad (3.5)$$

Die benötigten Matrizen können in zwei verschiedenen Mengen vorliegen, die jeweils für eine andere Normlichtart optimiert sind [Ado19, S. 85]. Eine Menge besteht dabei aus den nachfolgenden Tags, die für die erste Menge das Suffix 1 und für die zweite Menge das Suffix 2 besitzen:

Kapitel 3. Digital Negative (DNG)

- *ColorMatrix*: Eine Transformationsmatrix, die XYZ Werte in den Kamerafarbraum einer Referenzkamera transformiert
- *CameraCalibration*: Eine Transformationsmatrix, die RGB Werte des Kamerafarbraums einer Referenzkamera in den Kamerafarbraum einer individuellen Kamera transformiert
- *CalibrationIlluminant*: Die Normlichtart, für welche die Matrizen optimiert sind

Welche Menge zur Berechnung der Transformationsmatrix $A_{Cam|XYZ}$ am besten geeignet ist, muss in Abhängigkeit des eingestellten Weißabgleichs bestimmt werden. Dafür wird zunächst die Farbtemperatur des Weißabgleichs, der entweder als RGB Koeffizienten im Tag *AsShotNeutral* oder als *xy Chromatizitätswerte* im Tag *AsShotWhiteXY* vorliegt sowie die Farbtemperatur der beiden Normlichtarten, ermittelt. [Ado19, S. 85] Da die Farbtemperatur der Normlichtarten standardisiert ist, müssen die entsprechenden Farbtemperaturen lediglich den beiden Normlichtarten zugeordnet werden. Zur Farbtemperaturbestimmung des Weißabgleichs existieren hingegen unterschiedliche mathematische Methoden. Eine einfache Approximation ist bei bekannten *xy Chromatizitätswerten* beispielsweise nach [McC92, S. 142f.] durch die Formel 3.6 möglich, wobei T der Farbtemperatur entspricht. Nach der Ermittlung aller Farbtemperaturen wird geprüft, ob die Farbtemperatur des Weißabgleichs zwischen den Farbtemperaturen der beiden Normlichtarten liegt. Ist dies der Fall, werden die zusammengehörigen Matrizen beider Mengen mit einem durch die Kehrwerte aller Farbtemperaturen bestimmten Gewichtungsfaktor linear interpoliert. Andernfalls wird die Menge der Matrizen derjenigen Normlichtart verwendet, die zur Farbtemperatur des Weißabgleichs am nächsten liegt. [Ado19, S. 85]

$$T = 449n^3 + 3525n^2 + 6823.3n + 5520.33 \quad (3.6)$$

wobei gilt: $n = \frac{x - 0.3320}{y - 0.2858}$

Zusätzlich muss geprüft werden, ob die Transformationsmatrizen, die sich in den Tags *CameraCalibration1* und -2 befinden, zur Berechnung der Transformationsmatrix $A_{Cam|XYZ}$ genutzt werden können. Dazu werden die Strings der Tags *ProfileCalibrationSignature* sowie *CameraCalibrationSignature* miteinander verglichen. Sind diese identisch, wurden die *CameraCalibration* Matrizen sowie das Kameraprofil mit der gleichen Referenzkamera ermittelt und können zur Berechnung von $A_{Cam|XYZ}$ genutzt werden. Andernfalls wird für jede *CameraCalibration* Matrix eine Identitätsmatrix angenommen. [Ado19, S. 85]

Nach der Bestimmung aller benötigten Matrizen lässt sich die Transformationsmatrix $A_{Cam|XYZ}$ nach [Ado19, S. 86] gemäß der Formel 3.7 berechnen. D steht dabei für eine Diagonalmatrix, die in dem Tag *AnalogBalance* enthalten ist, $A_{ICam|RCam}$ für die (interpolierte) Transformationsmatrix *CameraCalibration* und $A_{RCam|XYZ}$ für

die (interpolierte) Transformationsmatrix *ColorMatrix*. Durch die Formel 3.5 kann schließlich $A_{XYZ|Cam}$ ermittelt werden.

$$A_{Cam|XYZ} = D \cdot A_{ICam|RCam} \cdot A_{RCam|XYZ} \quad (3.7)$$

Chromatische Adaptionstransformation

Die Angabe von XYZ Werten im CIE XYZ Farbraum erfolgt in Bezug auf einem Weißpunkt, der von der Farbtemperatur abhängig ist. Somit führen XYZ Werte, die sich auf unterschiedliche Weißpunkte beziehen, zu einer anderen Farbwahrnehmung. [BB16, S. 355] Die XYZ Werte, welche durch die Transformationsmatrix $A_{XYZ|Cam}$ berechnet werden, beziehen sich auf den Weißpunkt, der durch die Farbtemperatur des eingestellten Weißabgleichs bestimmt ist. Zur Durchführung mehrerer Verarbeitungsschritte erfordert die DNG Spezifikation jedoch das Vorliegen der XYZ Werte in Bezug auf den Weißpunkt, der durch die Normlichtart D50 spezifiziert ist.

Durch die Formel 3.8 kann nach [Ado19, S. 87] die Matrix $A_{XYZ50|Cam}$ bestimmt werden, die RGB Werte im Kamerafarbraum in den CIE XYZ Farbraum transformiert, sodass sich die XYZ Werte auf den Weißpunkt D50 beziehen. Die Matrix C wird im Allgemeinen als chromatische Adoptionsmatrix bezeichnet und kann durch den linearen Bradford Algorithmus ermittelt werden, der hier nicht näher erläutert wird [Ado19, S. 87]. Konkret transformiert die chromatische Adoptionsmatrix $C_{50|Wb}$ auf dem eingestellten Weißabgleich basierte XYZ Werte in D50 basierte XYZ Werte.

$$A_{XYZ50|Cam} = C_{50|Wb} \cdot A_{XYZ|Cam} \quad (3.8)$$

$$\begin{pmatrix} X_{50} \\ Y_{50} \\ Z_{50} \end{pmatrix} = A_{XYZ50|Cam} \cdot \begin{pmatrix} R_{Cam} \\ G_{Cam} \\ B_{Cam} \end{pmatrix} \quad (3.9)$$

Farbraumtransformation sRGB

In welchen Farbraum die XYZ (D50) Werte weiter transformiert werden müssen, hängt von den übrigen Verarbeitungsschritten der Verarbeitungspipeline sowie vom finalen Zielfarbraum ab. Für die Durchführung des Hue/Saturation/Value Mappings müssen die XYZ (D50) Werte beispielsweise in den Reference Input Medium Metric (RIMM) RGB Farbraum überführt werden [Ado19, S. 88]. Da dieser Verarbeitungsschritt am laufenden Beispiel nicht benötigt wird, können die XYZ (D50) Werte direkt für die Anzeige auf einem Monitor in den finalen Zielfarbraum transformiert werden. Als Zielfarbraum wird sRGB gewählt, der zwar einen geringen Farbumfang besitzt, allerdings besonders stark verbreitet ist [Wal05, S. 167f.].

Kapitel 3. Digital Negative (DNG)

Wie Formel 3.9 zeigt überführt die Transformationsmatrix $A_{XYZ50|Cam}$ die *RGB* Werte des Kamerafarbraums in den CIE XYZ Farbraum in Bezug auf den Weißpunkt D50. Der sRGB Farbraum bezieht sich jedoch auf den Weißpunkt, welcher durch die CIE Normlichtart D65 definiert ist [AMCS96, S. 201]. Folglich muss erneut eine chromatische Adaptionstransformation der Formel 3.10 durchgeführt werden, bevor die XYZ Werte in *RGB* Werte des sRGB Farbraums transformiert werden können. Die chromatische Adoptionsmatrix $C_{65|50}$ transformiert dabei D50 basierte XYZ Werte in D65 basierte XYZ Werte.

$$\begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix} = C_{65|50} \cdot \begin{pmatrix} X_{50} \\ Y_{50} \\ Z_{50} \end{pmatrix} \quad (3.10)$$

Liegen die neuen XYZ (D65) Werte vor, können sie nach [AMCS96, S. 202] gemäß der Formel 3.11 zu *RGB* Werten im sRGB Farbraum transformiert werden.

$$\begin{pmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{pmatrix} = \begin{pmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{pmatrix} \cdot \begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix} = A_{sRGB|XYZ65} \cdot \begin{pmatrix} X_{65} \\ Y_{65} \\ Z_{65} \end{pmatrix} \quad (3.11)$$

Durch das Einsetzen der Formel 3.9 in 3.10 und 3.10 in 3.11 lässt sich die gesamte Transformation der *RGB* Werte des Kamerafarbraums in den sRGB Farbraum als eine einzige Matrixmultiplikation mit der Matrix $A_{sRGB|Cam}$ formulieren:

$$\begin{pmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{pmatrix} = A_{sRGB|XYZ65} \cdot C_{65|50} \cdot A_{XYZ50|Cam} \cdot \begin{pmatrix} R_{Cam} \\ G_{Cam} \\ B_{Cam} \end{pmatrix} = A_{sRGB|Cam} \cdot \begin{pmatrix} R_{Cam} \\ G_{Cam} \\ B_{Cam} \end{pmatrix} \quad (3.12)$$

Gammakorrektur

Ein Monitor gibt Lichtsignale nicht linear wieder, sondern in Abhängigkeit eines Exponenten, der mit dem griechischen Buchstaben Gamma gekennzeichnet wird [Too16, S. 227f.]. Dies führt zu einer falschen Helligkeitswiedergabe der Aufnahme, die korrigiert werden muss. Die Korrektur erfolgt über die Kompensation der Nichtlinearität des Monitors und wird als Gammakorrektur bezeichnet [Too16, S. 238f.].

Bei der Gammakorrektur werden die *RGB* Werte einer Aufnahme i. d. R. mit dem Kehrwert des Gammas potenziert, wobei für das Gamma ein Wert von 2.2 angenommen werden kann [Too16, S. 241] [AMCS96, S. 198f.]. Für eine Aufnahme im Farbraum sRGB wird die Gammakorrektur nach [AMCS96, S. 202] durch die Formel 3.13 durchgeführt, die ein Gamma von 2.2 annähert. R'_{sRGB} , G'_{sRGB} sowie B'_{sRGB} stehen dabei für die gammakorrigierten *RGB* Werte im sRGB Farbraum.

$$\begin{aligned}
 R'_{sRGB} &= \begin{cases} 12.92 \cdot R_{sRGB} & \text{für } R_{sRGB} \leq 0.00304 \\ 1.055 \cdot R_{sRGB}^{(1.0/2.4)} - 0.005 & \text{anderenfalls} \end{cases} \\
 G'_{sRGB} &= \begin{cases} 12.92 \cdot G_{sRGB} & \text{für } G_{sRGB} \leq 0.00304 \\ 1.055 \cdot G_{sRGB}^{(1.0/2.4)} - 0.005 & \text{anderenfalls} \end{cases} \\
 B'_{sRGB} &= \begin{cases} 12.92 \cdot B_{sRGB} & \text{für } B_{sRGB} \leq 0.00304 \\ 1.055 \cdot B_{sRGB}^{(1.0/2.4)} - 0.005 & \text{anderenfalls} \end{cases}
 \end{aligned} \tag{3.13}$$



Die Abbildung 3.12 (b) zeigt die digitale Aufnahme nach der Farbraumtransformation vom Kamerafarbraum, in dem sich die Abbildung 3.12 (a) befindet, in den sRGB Farbraum unter Einbeziehung der in Tabelle 3.4 stehenden Tags. Für einen besseren Vergleich wurde in Abbildung 3.12 (b) noch keine Gammakorrektur durchgeführt. In der Abbildung 3.12 (c) ist schließlich die digitale Aufnahme nach der Durchführung aller Verarbeitungsschritte einschließlich der Gammakorrektur zu sehen.



(a) Digitale Aufnahme im Kamerafarbraum nach dem Demosaicing



(b) Durchgeführte Farbraumtransformation ohne Gammakorrektur



(c) Digitale Aufnahme nach der Durchführung aller Verarbeitungsschritte

Abbildung 3.12: Farbraumtransformation im Vorher - Nachher Vergleich am laufenden Beispiel

Tag	Wert		
CalibrationIlluminant1	17 (Standard Light A)		
CalibrationIlluminant2	23 (D50)		
ColorMatrix1	1.2587	-0.5232	-0.1496
	-0.361	1.0841	0.0277
	-0.0911	0.1674	0.207
ColorMatrix2	0.8389	-0.3198	-0.1019
	-0.3834	1.0222	0.0661
	-0.1122	0.19	0.3415
AsShotNeutral	0.5151	1	0.6564

Tabelle 3.4: Zur Durchführung der Farbraumtransformation relevante Tags mit Werten aus dem laufenden Beispiel (Leica SL2)

3.4.6 Weitere Verarbeitungsschritte

In Abhängigkeit der enthaltenen Tags einer DNG Datei, müssen zu den oben vorgestellten Verarbeitungsschritten ggfs. Weitere durchgeführt werden, die teils als optional gekennzeichnet sind. Die Wichtigsten dieser Verarbeitungsschritte sind nachfolgend kurz zusammengefasst:

- Opcodes: Werden zu unterschiedlichen Verarbeitungszeitpunkten ausgeführt und beinhalten u. a. Verarbeitungsschritte wie die Objektivkorrektur oder die Korrektur defekter Pixel (vgl. Kapitel 3.2.1). Ob ein Opcode ausgeführt werden muss, ist für jede DNG Datei im Opcode kodiert
- Hue/Saturation/Value Mapping: Passt den Farbwert (Hue), die Farbsättigung (Saturation) sowie die Helligkeit (Value) eines Pixels an
- Belichtungskompensation: Passt die Belichtung der Aufnahme an
- Profile Look Table Mapping: Ein optional durchzuführender Schritt, der analog zum Hue/Saturation/Value Mapping den Farbwert, die Farbsättigung sowie die Helligkeit eines Pixels anpasst. Dieser wird jedoch zwischen der Belichtungskompensation und der Tonwertkorrektur durchgeführt
- Tonwertkorrektur: Ein optional durchzuführender Verarbeitungsschritt, der den Kontrast der digitalen Aufnahme anpasst

4 Konzeption

Der RAW Konverter, der in dieser Arbeit entsteht, soll sich insbesondere durch Erweiterbarkeit und Wartbarkeit auszeichnen - Zwei wichtige Qualitätsmerkmale, die eine gute Software auszeichnen. Diese sind nur zu erreichen, indem bereits vor der Implementierung Anforderungen identifiziert und definiert werden sowie die zugrunde liegende Architektur einschließlich der benötigten Komponenten festgelegt wird. Nachfolgend werden die Anforderungen an den RAW Konverter sowie dessen Architektur vorgestellt.

4.1 Anforderungen

Üblicherweise werden Anforderungen an eine Software in funktionale und nicht-funktionale Anforderungen unterteilt. Funktionale Anforderungen beschreiben im Einzelnen die Funktionalitäten, welche eine Software bereitstellen soll. Im Gegensatz dazu betrachten nicht-funktionale Anforderungen eine Software im Gesamten und definieren ihre Eigenschaften sowie Einschränkungen. [Som11, S. 85ff.] Die nachfolgend identifizierten und definierten Anforderungen sind diesem Schema entsprechend klassifiziert.

4.1.1 Funktionale Anforderungen

Durch die Vielzahl an funktionalen Anforderungen, die in einer begrenzten Zeit realisiert werden müssen, ist eine Priorisierung der Anforderungen erforderlich. Die Priorisierung erfolgt gemäß der MoSCoW Methode, welche Anforderungen nach [Int09, S. 102] in die vier genannten Kategorien unterteilt:

1. Must: Hochpriorisierte Anforderungen, die zur Erreichung des Ziels umgesetzt werden müssen
2. Should: Hochpriorisierte Anforderungen, die umgesetzt werden sollten aber zur Erreichung des Ziels nicht zwingend erforderlich sind
3. Could: Erwünschte Anforderung, die nur umgesetzt werden, sofern genügend Ressourcen zur Verfügung stehen
4. Won't: Identifizierte Anforderungen, die nicht in der aktuellen Version umgesetzt werden

Funktionale Anforderungen der Kategorie "Must" definieren die grundlegenden Funktionalitäten des RAW Konverters und gewährleisten seine Nutzbarkeit. Sie stellen u. a. sicher, DNG Dateien auslesen und verarbeiten zu können, welche durch den Adobe DNG Converter erzeugt wurden. Die funktionalen Anforderungen der Kategorie "Should" erweitern den Funktionsumfang des RAW Konverters und ermöglichen das Einlesen und Verarbeiten von DNG Dateien, die kameraintern erzeugt wurden sowie mit dem verlustfreien JPEG Verfahren komprimiert sind. In der Kategorie "Won't" identifizierte Anforderungen haben für diesen Arbeit keine Relevanz, können jedoch bei der Weiterführung des RAW Konverters realisiert werden.

Must

- Der RAW Konverter muss alle zur Verarbeitung der DNG Datei benötigten IFDs und Tags auslesen können. Diese umfassen:
 - Das vom Header referenzierte (erste) IFD
 - Das RAW IFD
 - Das Exif IFD
- Der RAW Konverter muss DNG Dateien auslesen können, deren Rohdaten mit dem Vielfachen eines Bytes kodiert sind. Andernfalls ist eine aussagekräftige Fehlermeldung auszugeben.
- Der RAW Konverter muss DNG Dateien auslesen können, deren Rohdaten auf einem CFA basieren. Andernfalls ist eine aussagekräftige Fehlermeldung auszugeben.
- Der RAW Konverter muss DNG Dateien auslesen können, deren Rohdaten unkomprimiert gespeichert sind. Andernfalls ist eine aussagekräftige Fehlermeldung auszugeben.
- Der RAW Konverter muss DNG Dateien auslesen können, deren Rohdaten als Strips vorliegen. Andernfalls ist eine aussagekräftige Fehlermeldung auszugeben.
- Der RAW Konverter muss das primäre Vorschaubild einer DNG Datei auslesen können, sofern eines vorhanden ist.
- Der RAW Konverter muss DNG Dateien verarbeiten können, deren Rohdaten auf einem Bayer CFA basieren. Andernfalls ist eine aussagekräftige Fehlermeldung auszugeben.
- Der Nutzer muss aus einer Auswahl unterschiedlicher Demosaicing Algorithmen auswählen können.

- Der RAW Konverter muss eine zeitgemäße Benutzeroberfläche bereitstellen, welche folgende Funktionalitäten implementiert:
 - Die Navigation im Dateisystem erlaubt
 - Eine Übersicht der verfügbaren DNG Dateien bereitstellt
 - Die verarbeitete DNG Datei anzeigen kann
 - Fehler und Ausnahmezustände an den Endnutzer kommuniziert
- Der Nutzer muss die digitale Aufnahme in einem geeigneten Dateiformat speichern können. Konkret müssen folgende Dateiformate unterstützt werden:
 - TIFF (8 und 16 Bit Farbtiefe)
 - JPEG (8 Bit Farbtiefe)

Should

- Der RAW Konverter sollte zusätzlich DNG Dateien auslesen können, deren Rohdaten nicht mit dem Vielfachen eines Bytes kodiert sind.
- Der RAW Konverter sollte zusätzlich DNG Dateien auslesen können, deren Rohdaten verlustfrei nach dem verlustfreien JPEG Verfahren komprimiert sind.
- Der RAW Konverter sollte zusätzlich DNG Dateien auslesen können, deren Rohdaten als Tiles vorliegen.

Could

- Der Nutzer sollte die digitale Aufnahme zusätzlich im Portable Network Graphics (PNG) Dateiformat (8 oder 16 Bit Farbtiefe) speichern können.

Won't

- Der RAW Konverter sollte DNG Dateien auslesen können, deren Rohdaten mit anderen von DNG unterstützten Kompressionsverfahren komprimiert sind.
- Der RAW Konverter sollte DNG Dateien auslesen können, deren Rohdaten nicht auf einem CFA basieren.
- Der RAW Konverter sollte DNG Dateien verarbeiten können, deren Rohdaten nicht auf einem Bayer CFA basieren (Monochrom Sensoren).
- Der RAW Konverter sollte andere Bestandteile einer DNG Datei wie beispielsweise XMP Metadaten auslesen können

Kapitel 4. Konzeption

- Der RAW Konverter sollte bildverarbeitende Verfahren wie z. B. Schärfung implementieren
- Der RAW Konverter sollte Opcode Listen ausführen können

4.1.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen tragen gleichermaßen zur Qualität des RAW Konverters bei und sind deshalb nicht priorisiert. Alle nicht-funktionalen Anforderungen sind grundsätzlich einzuhalten.

- Der Programmcode des RAW Konverters muss erweiterbar sein. Die Erweiterbarkeit schließt dabei die folgenden Punkte ein:
 - Es müssen mit geringem Aufwand weitere Demosaicing Algorithmen hinzugefügt werden können
 - Es müssen mit geringem Aufwand weitere Verarbeitungsschritte hinzugefügt werden können
 - Es müssen mit geringem Aufwand weitere Dekomprimierungsverfahren hinzugefügt werden können
 - Es müssen mit geringem Aufwand neue Funktionalitäten hinzugefügt werden können
- Der Programmcode des RAW Konverters muss wartbar sein. Die Wartbarkeit schließt dabei die folgenden Punkte ein:
 - Die Änderung einer Funktionalität darf keine Auswirkungen auf andere Funktionalitäten haben
 - Einzelne Komponenten wie z. B. die Benutzeroberfläche müssen austauschbar sein
- Öffentliche Methoden des Programmcodes müssen ausreichend und verständlich kommentiert sein
- Die grafische Benutzeroberfläche muss modern, überschaubar und intuitiv zu bedienen sein

4.2 Architektur

Eine gute Softwarearchitektur leistet einen großen Beitrag zur Gewährleistung der Softwarequalität und ist deswegen von wesentlicher Bedeutung [BCK13, S. 26f.]. Unter Berücksichtigung der Anforderungen wird zur Festlegung der RAW Konverter Architektur ein komponentenbasierter Ansatz gewählt. Dieser besteht aus

voneinander entkoppelten Komponenten, die miteinander kommunizieren. Konkret wird der RAW Konverter in die drei folgenden Komponenten aufgeteilt:

- DNG-Reader: Ist für das Auslesen einer DNG Datei zuständig und stellt die Rohdaten, das primäre Vorschaubild, die IFDs sowie alle weiteren Daten bereit, die sich in einer DNG Datei befinden und für das Verarbeiten der Rohdaten notwendig sind
- DNG-Processor: Ist für das Verarbeiten und Interpretieren der Rohdaten zuständig, die vom DNG-Reader bereitgestellt werden
- JENIFFER2: Stellt die grafische Benutzeroberfläche zur Verfügung, die zur Steuerung des DNG-Readers und des DNG-Processors dient

Die Abbildung 4.1 zeigt die Architektur des RAW Konverters als Unified Modeling Language (UML) Komponentendiagramm, welches die einzelnen Komponenten und deren Abhängigkeiten zueinander veranschaulicht. Der DNG-Reader und der DNG-Processor bilden eine logische Einheit und sind deswegen in einer gemeinsamen Komponente gekapselt. Da der DNG-Processor vom DNG-Reader die Rohdaten und andere Bestandteile einer DNG Datei übergeben bekommt, aber der DNG-Reader keine Informationen vom DNG-Processor benötigt, findet eine unidirektionale Kommunikation zwischen diesen beiden statt.

Die grafische Benutzeroberfläche ist ebenfalls in einer eigenen Komponente ausgelagert. Sie kommuniziert unidirektional mit dem DNG-Processor und dem DNG-Reader über eine sogenannte Fassade. Dabei handelt es sich um ein Entwurfsmuster (konkret um ein Strukturmuster), das in der Softwareentwicklung als zentrale Schnittstelle zur Kommunikation mit verschiedenen Unterkomponenten eingesetzt wird [JGVH95, S. 185].

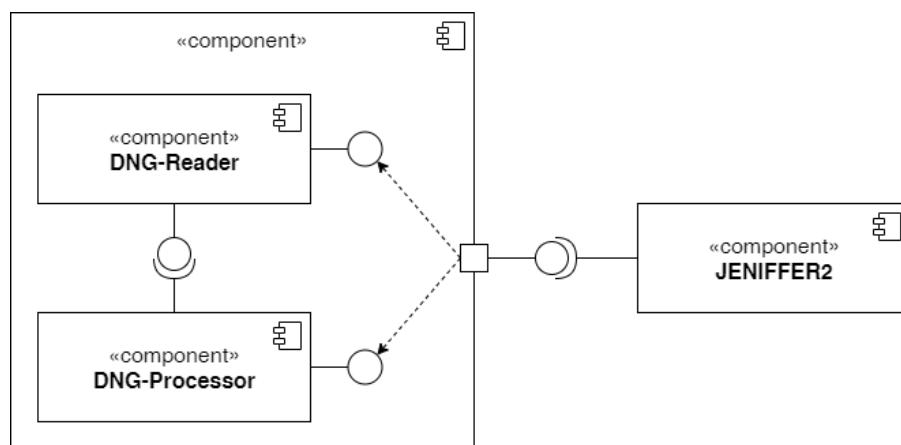


Abbildung 4.1: UML Komponentendiagramm des RAW Konverters

Kapitel 4. Konzeption

Die Unterteilung des RAW Konverters in verschiedene Komponenten, mit Kommunikation über eine Fassade, impliziert entscheidende Vorteile dieser Architektur: Die Fassade reduziert die Menge an Abhängigkeiten zwischen den Komponenten und damit auch die Gesamtkomplexität der Anwendung [JGVH95, S. 185]. Werden beispielsweise Änderungen am DNG-Processor vorgenommen, muss höchstens die Fassade angepasst werden, die Benutzeroberfläche bleibt jedoch unangetastet. Da die Fassade in jedem Fall bestehen bleibt, können außerdem Komponenten leichter ausgetauscht werden. Insgesamt realisiert diese Architektur eine lose Kopplung zwischen den Komponenten.

5 Implementierung

In Kapitel 4 wurde das zugrunde liegende Konzept des in dieser Arbeit entstehenden RAW Konverters erläutert. Auf der Grundlage dieses Konzepts wird nun die Implementierung vorgestellt. Dabei wird jede Komponente individuell betrachtet. Es wird zunächst dessen Rolle im RAW Konverter konkretisiert und der Aufbau der Komponente zusammen mit den wichtigsten Bestandteilen beschrieben. Abschließend wird die grundlegende Funktionsweise der Komponente anhand von ausgewähltem Quelltext beispielhaft erläutert.

5.1 DNG-Reader

Der DNG-Reader bildet das Fundament des RAW Konverters und ist für das Auslesen und Dekodieren einer DNG Datei zuständig. Er stellt Methoden zur Verfügung, die den Zugriff auf die Rohdaten, das primäre Vorschaubild sowie auf die IFDs ermöglichen. Durch den DNG-Reader können sowohl als Strips als auch als Tiles vorliegende Rohdaten sowie kameraintern und kameraextern erzeugte DNG-Dateien ausgelesen werden.

Gemäß der funktionalen Anforderungen (vgl. Kapitel 4.1.1) ist das Auslesen einer DNG Datei unter den Bedingungen möglich, dass die Rohdaten auf einem CFA basieren und nicht komprimiert sind. Verlustfrei komprimierte Rohdaten können nur ausgelesen und dekodiert werden, wenn als Kompressionsverfahren das verlustfreie JPEG verwendet wurde. Eine DNG Datei, die diesen Bedingungen genügt, ist durch die folgenden Tags mit den zugehörigen Werten charakterisiert:

- *PhotometricInterpretation*: 32803 (CFA)
- *PlanarConfiguration*: 1 (Chunky)
- *SamplesPerPixel*: 1
- *Compression*: 1 (Uncompressed) oder 7 (Lossless JPEG)

5.1.1 Aufbau

Die Abbildung 5.1 veranschaulicht die Struktur des DNG-Readers als UML Klassendiagramm. Dieses ist nicht vollständig, zeigt jedoch die wichtigsten Klassen und Interfaces mit ihren Attributen und Methoden sowie deren Beziehungen

Kapitel 5. Implementierung

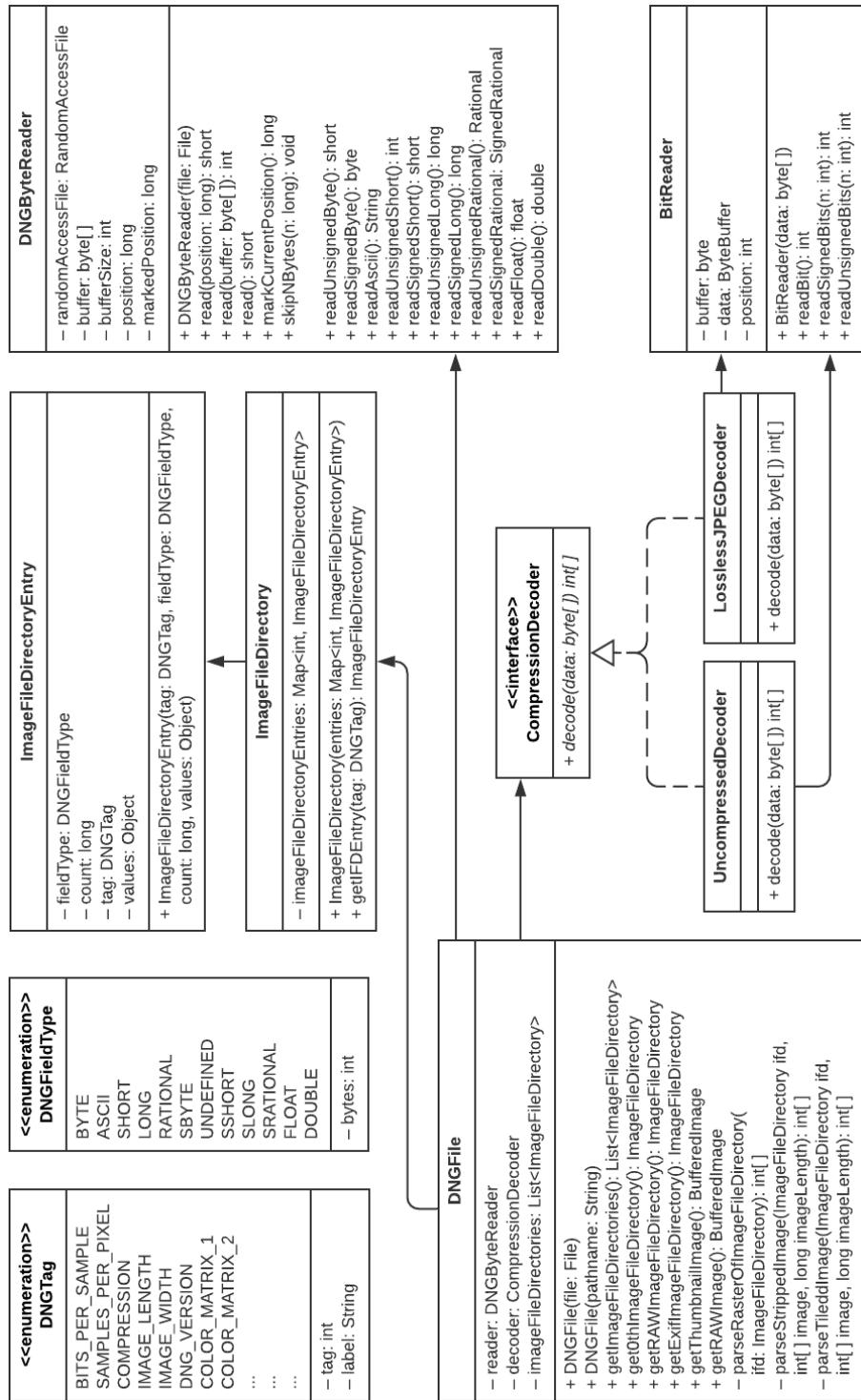


Abbildung 5.1: Klassendiagramm des DNG-Readers

zueinander. Durch die Klassen `DNGByteReader`, `BitReader` sowie das Interface `CompressionDecoder` liegt ein besonderer Fokus auf der Erweiterbarkeit des DNG-Readers. Diese implementieren Funktionalitäten, welche das Auslesen und Dekodieren weiterer Bestandteile einer DNG Datei wie z. B. XMP Metadaten sowie die Implementierung anderer Kompressionsverfahren ermöglichen. Nachfolgend werden die wichtigsten Klassen und Interfaces genauer erläutert.

DNGFile

Die Klasse `DNGFile` dient als zentrale Schnittstelle und verwaltet die Kommunikation zu den anderen Klassen des DNG-Readers. Unter Zuhilfenahme der Klasse `DNGByteReader` sowie einer Implementierung des Interfaces `CompressionDecoder` ist sie für das Interpretieren der dekodierten Daten zuständig und bestimmt dadurch die einzelnen Bestandteile einer DNG Datei. Über sie können die Rohdaten, das primäre Vorschaubild sowie die einzelnen IFDs abgefragt werden. Ferner prüft sie die Validität einer DNG Datei und die Einhaltung der oben genannten Bedingungen.

DNGByteReader

Das eigentliche Auslesen und Dekodieren einer DNG Datei erfolgt über die Klasse `DNGByteReader`. Mit Hilfe der Klasse `RandomAccessFile`, die den wahlfreien Zugriff auf Dateien erlaubt, greift sie lesend auf eine angegebene DNG Datei im Dateisystem zu und liefert die dezimale Repräsentation für n gelesene Bytes. Da lesende Zugriffe auf das Dateisystem kostspielig sind, verfügt der `DNGByteReader` über einen Puffer mit konfigurierbarer Größe, der mehrere gelesene Bytes zwischenspeichern kann.

BitReader

Nicht immer ist das byteweise Auslesen von Rohdaten durch den `DNGByteReader` möglich, z. B. wenn für die Kodierung eines Rohdatenwerts kein Vielfaches eines Bytes verwendet wurde (vgl. Kapitel 3.3.2) oder die Rohdaten verlustfrei JPEG komprimiert sind. In diesem Fall müssen die Rohdaten bitweise ausgelesen werden. Diese Funktionalität implementiert die Klasse `BitReader`, die für n gelesene Bits die dezimale Repräsentation ermittelt.

CompressionDecoder

Das Interface `CompressionDecoder` definiert eine Methodensignatur zur Dekomprimierung und Dekodierung eines Strips oder eines Tiles. Eine Klasse, die dieses Interface implementiert, muss ein entsprechendes Dekomprimierungsverfahren realisieren.

Die Dekodierung unkomprimierter Strips oder Tiles erfolgt durch die Klasse `UncompressedDecoder`. Verlustfrei JPEG komprimierte Strips oder Tiles hingegen können durch die Klasse `LosslessJPEGDecoder` dekomprimiert und dekodiert werden.

ImageFileDirectoryEntry und ImageFileDirectory

Die einzelnen Tags einer DNG Datei werden mit ihren Werten und dem zugehörigen Datentypen in der Klasse `ImageFileDirectoryEntry` gekapselt. Jedes `ImageFileDirectoryEntry` ist einer Instanz der Klasse `ImageFileDirectory` zugeordnet und wird von dieser verwaltet. Sofern vorhanden, liefert ein `ImageFileDirectory` das entsprechende `ImageFileDirectoryEntry` für ein gegebenes Tag zurück. Beide werden von der Klasse `DNGfile` erzeugt.

5.1.2 Auslesen einer DNG Datei

Das Auslesen einer DNG Datei beginnt mit der Instanziierung der Klasse `DNGfile`, die den Pfadnamen zur DNG Datei erwartet. Mit Hilfe einer Instanz der Klasse `DNGByteReader` liest sie den Header der DNG Datei aus, interpretiert diesen und führt eine Validierung durch. Bei der Validierung wird geprüft, ob der Header die Byte Reihenfolge festlegt und ob dieser die magische Zahl 42 beinhaltet (vgl. Kapitel 3.2.2). Falls die DNG Datei valide ist, können über die `DNGfile` Instanz die Rohdaten, das primäre Vorschaubild sowie die einzelnen IFDs abgefragt werden. Die erste Abfrage führt zum einmaligen Auslesen der genannten Bestandteile, die für alle nachfolgenden Abfragen zwischengespeichert werden.

Das Auslesen und Dekodieren der Bestandteile einer DNG Datei sind komplexe Programmabläufe. Diese sind am besten anhand der Abfrage von Rohdaten zu demonstrieren, die als unkomprimierte und 16 Bit kodierte Strips vorliegen.

Auslesen eines Bytes einer DNG Datei

Mit Ausnahme von Rohdaten, die nicht mit dem Vielfachen eines Bytes kodiert sind, erfolgt das Auslesen und Dekodieren einer DNG Datei byteweise. Quelltext 5.1 zeigt die Methode der Klasse `DNGByteReader`, welche das Auslesen eines Bytes einer DNG Datei implementiert. Um die Anzahl an lesenden Zugriffen auf das Dateisystem zu minimieren, nutzt sie einen Puffer, in dem mehrere gleichzeitig gelesene Bytes zwischengespeichert werden.

Ein Byte wird an einer Position gelesen, die in der Variable `position` festgelegt ist. Zuerst wird in Zeile 11 geprüft, ob das Byte dem Puffer entnommen werden kann oder dieser vorher befüllt werden muss. Der letztere Fall ist in Abbildung 5.2 für einen Puffer der Größe 8 Byte exemplarisch veranschaulicht und führt zur Ausführung der Zeile 12. Grün dargestellt sind die Bytes, die bereits gelesen wurden

und sich im Puffer befinden und orange die Bytes, mit welchen der Puffer befüllt werden muss. Dazu wird in Zeile 12 die neue Startposition ermittelt, die immer ein Vielfaches der Puffergröße ist. Zu dieser wird durch eine Instanz der Klasse RandomAccessFile, die den lesenden Zugriff auf die DNG Datei tätigt, in Zeile 13 navigiert. Die darauffolgende Zeile liest den Bereich der DNG Datei aus (orange) und schreibt ihn in den Puffer. Abschließend wird die Zeile 17 in jedem Fall ausgeführt, die den Byte an der gewünschten Position zurückliefert und diese inkrementiert.

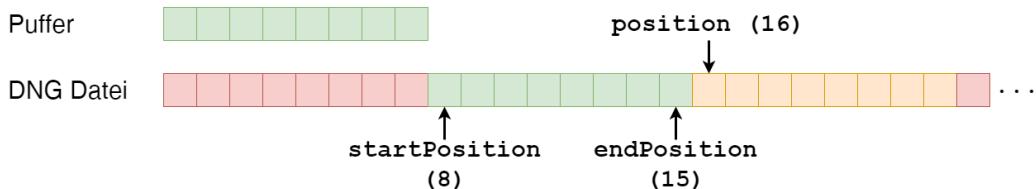
```

1  public class DNGByteReader {
2
3      private RandomAccessFile randomAccessFile;
4      private byte fileBuffer[];
5      private int bufferSize = 8192;
6      private long startPosition = -1;
7      private long endPosition = -1;
8      private long position;
9
10     ...
11     public byte read() throws IOException {
12         if (position < startPosition || position > endPosition) {
13             startPosition = (position / bufferSize) * bufferSize;
14             randomAccessFile.seek(startPosition);
15             int numBytesRead = randomAccessFile.read(fileBuffer);
16             endPosition = startPosition + numBytesRead - 1;
17         }
18         return fileBuffer[(int) (position++ - startPosition)];
19     }
}

```

Quelltext 5.1: Methode zum Auslesen eines Bytes einer DNG Datei

Vor dem Befüllen (Zeile 11)



Nach dem Befüllen (Zeile 12 bis 15)

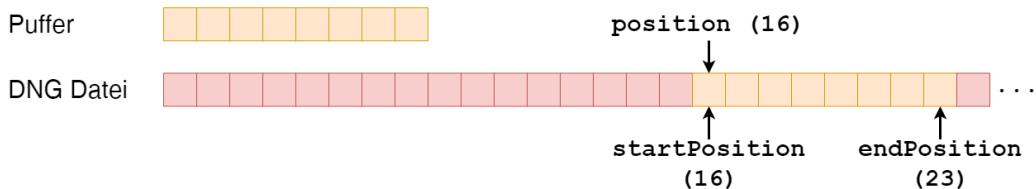


Abbildung 5.2: Beispielhaftes Befüllen eines Puffers mit Bytes einer DNG Datei



Dekodieren eines Strips oder eines Tiles

Liegen die Rohdaten als unkomprimierte Strips oder Tiles in Byte Arrays vor, können sie über die Klasse UncompressedDecoder dekodiert werden. Quelltext 5.2 zeigt einen Ausschnitt des Programmcodes, der für das Dekodieren von 16 Bit kodierten Strips und Tiles zuständig ist. Die Klasse UncompressedDecoder implementiert das Interface CompressionDecoder und die darin definierte Methode. Diese bekommt, wie in Zeile 4 zu sehen ist, das Strip oder Tile als Byte Array übergeben und gibt das dekodierte Strip oder Tile als Integer Array zurück. Da 16 Bit zur Bildung von 2 Bytes benötigt werden, müssen zwei benachbarte Bytes des Byte Arrays zur Dekodierung eines Rohdatenwerts zusammengefasst werden. In Zeile 6 wird deswegen ein Integer Array erzeugt, das halb so viele Werte umfasst wie das übergebene Byte Array.

Die Dekodierung zweier benachbarter Bytes erfolgt in den Zeilen 8 bis 10 und ist für ein besseres Verständnis in der Abbildung 5.3 beispielhaft illustriert, wobei die zugehörigen Dezimalwerte jeweils in Klammern angegeben sind. Um eine vorzeichenlose Zahl zu erhalten, müssen beide Bytes in Zeile 8 und 9 zunächst durch eine bitweise UND-Verknüpfung mit der hexadezimalen Zahl FF in einen Integer

```

1 public class UncompressedDecoder implements CompressionDecoder {
2     ...
3     @Override
4     public int[] decode(byte[] stripOrTile) {
5         ...
6         int[] decodedStripOrTile = new int[stripOrTile.length / 2];
7         for (int i = 0; i < stripOrTile.length; i = i + 2) {
8             int stripOrTile1AsInt = (stripOrTile[i] & 0xFF) << 8;
9             int stripOrTile2AsInt = stripOrTile[i + 1] & 0xFF;
10            decodedStripOrTile[i] = stripOrTile1AsInt | stripOrTile2AsInt;
11        }
12        return decodedStripOrTile;
13    }
14}
```

Quelltext 5.2: Methode zum Dekodieren eines Strips oder eines Tiles

$ \begin{aligned} & 10001001 \\ & \& 000000000000000000000000000011111111 \\ & = 000000000000000000000000000010001001 \\ & = 00000000000000000000000000001000100010000000 \end{aligned} $	$ \begin{aligned} & stripOrTile[i] (-119) \\ & 0xFF \\ & (137) \\ & stripOrTile1AsInt (35072) \end{aligned} $
$ \begin{aligned} & 01001000 \\ & \& 000000000000000000000000000011111111 \\ & = 00000000000000000000000000001001000 \end{aligned} $	$ \begin{aligned} & stripOrTile[i+1] (72) \\ & 0xFF \\ & stripOrTile2AsInt (72) \end{aligned} $
$ \begin{aligned} & 00000000000000001000100100000000 \\ & \mid 0000000000000000000000001001000 \\ & = 000000000000000010001001001000 \end{aligned} $	$ \begin{aligned} & stripOrTile1AsInt (35072) \\ & stripOrTile2AsInt (72) \\ & decodedStripOrTile[i] (35144) \end{aligned} $

Abbildung 5.3: Beispielhafte Zusammenfassung zweier Bytes zu einem 16 Bit Rohdatenwert

Datentypen konvertiert werden. Dies ist notwendig, da der Datentyp Byte sowie andere primitive Datentypen in Java, eine Zahl im Zweierkomplement darstellen. Das höchstwertige Bit gibt damit das Vorzeichen an, womit im Falle des Datentyps Byte ein maximaler Wert von $2^{8-1} = 128$ möglich ist. Zusätzlich muss durch den Schiebeoperator << die Integer Repräsentation des ersten Bytes um 8 Bit nach links geschoben werden (Zeile 8). Der dekodierte 16 Bit Rohdatenwert ergibt sich schließlich in Zeile 10 durch eine bitweise ODER-Verknüpfung beider Integer Werte.

Abfragen von Rohdaten

Die Rohdaten können über die Methode `getRawImage()` der Klasse `DNGFile` abgefragt werden. Diese ermittelt das RAW IFD, welches Verweise auf die Rohdaten enthält und prüft über das Tag `StripOffsets`, ob die Rohdaten als Strips vorliegen. In diesem Fall wird die in Quelltext 5.3 angegebene Methode aufgerufen. Als Eingabewerte bekommt sie das ermittelte RAW IFD sowie eine Referenz auf ein Array übergeben, in welches die gelesenen Rohdaten geschrieben werden sollen. Durch die Zeilen 7 und 8 werden dem RAW IFD die Verweise auf die Strips, die als Offset vom Dateibeginn aus angegeben werden sowie die Anzahl an Bytes, welche ausgelesen werden müssen, entnommen. Jedes Strip wird anschließend in den Zeilen 10 bis 15 vom `DNGByteReader` ausgelesen und in der Zeile 16 vom `CompressionDecoder` dekodiert. Alle ausgelesenen und dekodierten Strips werden abschließend in den Zeilen 17 bis 19 zu einem eindimensionalen Array zusammengefasst und in das übergebene Array geschrieben.

```

1 public class DNGFile {
2
3     private DNGByteReader reader;
4     private CompressionDecoder decoder;
5     ...
6     private int[] parseStrippedImage(ImageFileDirectory ifd, int[] image) {
7         long[] stripOffsets = ifd.getStripOffsets();
8         long[] stripByteCounts = ifd.getStripByteCounts();
9         for (int i = 0; i < stripOffsets.length; i++) {
10             long stripOffset = stripOffsets[i];
11             long stripByteCount = stripByteCounts[i];
12             byte[] strip = new byte[(int) stripByteCount];
13             reader.reset();
14             reader.skipNBytes(stripOffset);
15             reader.read(strip);
16             int[] decodedStrip = decoder.decode(strip);
17             int offset = i * decodedStrip.length;
18             int remainingSpace = image.length - offset;
19             System.arraycopy(decodedStrip, 0, image, offset, Math.min(
20                 remainingSpace, decodedStrip.length));
21         }
22         return image;
23     }
}

```

Quelltext 5.3: Methode zum Auslesen und Dekodieren von Strips



Kapitel 5. Implementierung

Die Methode des `DNGByteReaders`, welche in Zeile 15 aufgerufen wird, implementiert dieselbe Funktionalität, die in Quelltext 5.1 illustriert und erläutert wurde. Sie liest jedoch mehrere Bytes gleichzeitig aus und speichert diese in dem übergebenen Byte Array.

5.2 DNG-Processor

Als unverzichtbarer Bestandteil des RAW Konverters ist der DNG-Processor für das Verarbeiten der Rohdaten einer DNG Datei zuständig und realisiert dazu die in Kapitel 3.5 vorgestellte Pipeline. Er stellt unterschiedliche Demosaicing Algorithmen zur Verfügung, wobei vor der Verarbeitung eine Auswahl getroffen werden kann. Durch seinen modularen Aufbau kann der DNG-Processor mit geringem Aufwand um weitere Verarbeitungsschritte und Demosaicing Algorithmen ergänzt werden. Damit erfüllt er die Anforderung der Erweiterbarkeit.

Die Verarbeitung von Rohdaten durch den DNG-Processor ist auf jene beschränkt, die von einem Bildsensor mit einem Bayer Filter erfasst wurden. Eine DNG Datei, die solche Rohdaten beinhaltet, zeichnet sich durch die folgenden Tags mit den angegebenen Werten aus:

- *CFAPlaneColor*: 0, 1, 2 (Rot, Grün, Blau)
- *CFALayout*: 1 (Rechteckig)
- *CFARepeatPatternDim*: 2, 2
- *DefaultScale*: 1.0, 1.0

5.2.1 Aufbau

Die Abbildung 5.4 gibt einen Überblick der wichtigsten Klassen, Interfaces, Attribute und Methoden des DNG-Processors in Form eines UML Klassendiagramms. Von zentraler Bedeutung sind das generische Interface `Processor<T>` und die generische Klasse `Pipeline<T>`, welche die Modularität und Erweiterbarkeit des DNG-Processors sicherstellen. Der Typ `T` wird generischer Typparameter genannt und erlaubt die Implementierung einer Klasse unabhängig eines konkreten Typs. Im Nachfolgenden werden diese sowie weitere relevante Klassen vorgestellt.

Processor

Das Interface `Processor<T>` definiert eine Methodensignatur zur Durchführung eines beliebigen Vorgangs wie z. B. eines Verarbeitungsschrittes. Wie Quelltext 5.4 verdeutlicht erwartet die Methode einen Eingabewert vom Typ `T` und liefert einen Rückgabewert vom gleichen Typ zurück. Dadurch ist es möglich, mehrere

5.2. DNG-Processor

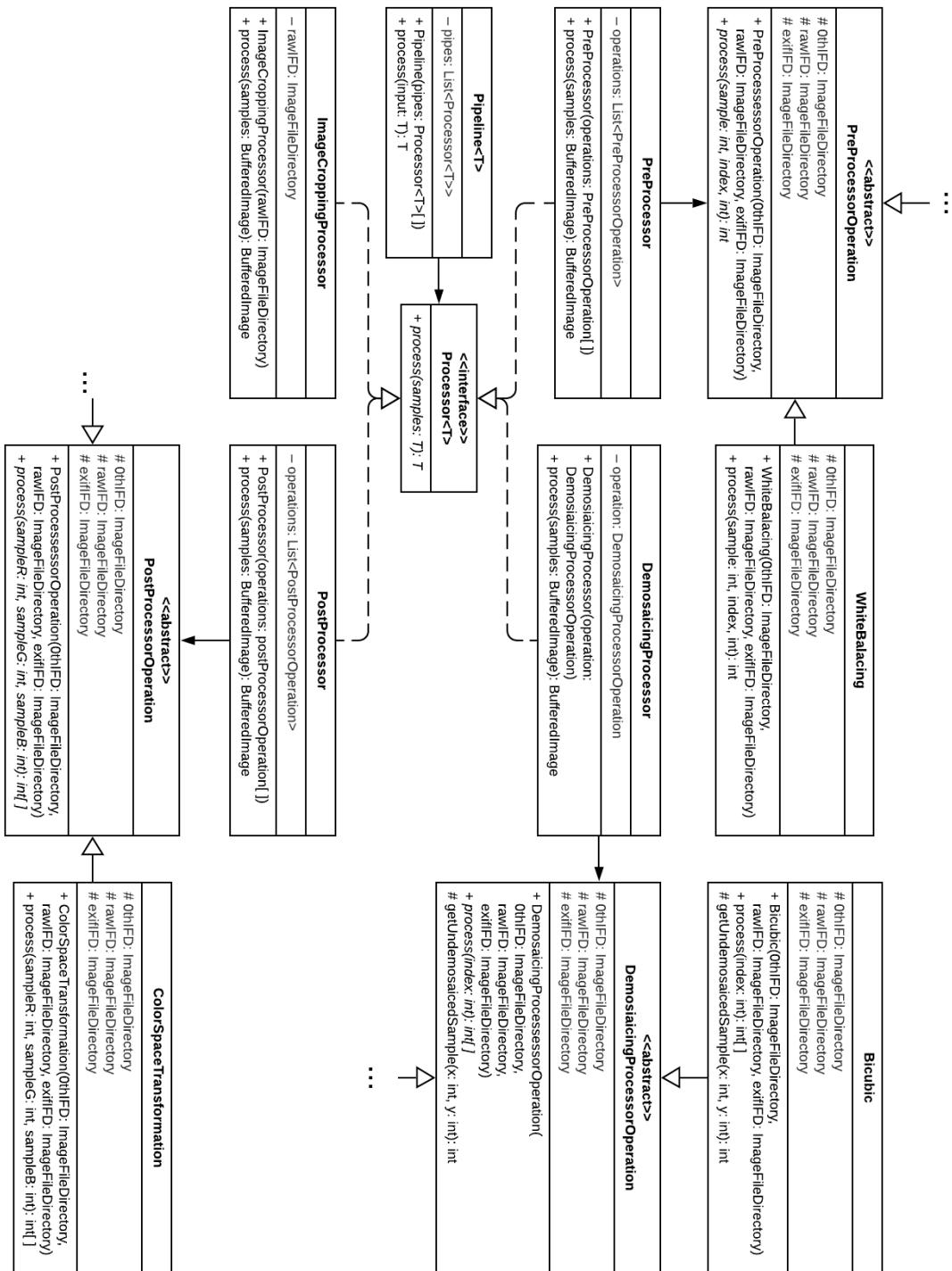


Abbildung 5.4: Klassendiagramm des DNG-Processors

Kapitel 5. Implementierung

Vorgänge sequenziell auszuführen, wobei der Rückgabewert eines Vorgangs als Eingabewert eines darauffolgenden Vorgangs genutzt werden kann. Dies setzt die Klasse Pipeline<T> um.

```
1 public interface Processor<T> {  
2     T process(T samples);  
3 }
```

Quelltext 5.4: Methodensignatur des generischen Interfaces Processor



Pipeline

Alle Vorgänge, die im Einzelnen das Interface Processor<T> implementieren, werden von der Klasse Pipeline<T> sequenziell ausgeführt. Diese ist durch den Typparameter T ebenfalls typisiert und stellt damit sicher, dass nur Vorgänge ausgeführt werden können, die den gleichen Typparameter besitzen.

Die Funktionalität zur Ausführung einzelner Vorgänge ist in Quelltext 5.5 verdeutlicht. Als Eingabewert erwartet die darin aufgeführte Methode einen Typ T und liefert den gleichen Typ als Rückgabewert zurück. Sie führt in den Zeilen 7 bis 9 alle Vorgänge sequenziell aus, wobei der Rückgabewert eines Vorgangs als Eingabewert eines darauffolgenden Vorgangs genutzt wird.

```
1 public class Pipeline<T> {  
2     private final List<Processor<T>> pipes;  
3     ...  
4     public T process(T input) {  
5         T processed = input;  
6         for (Processor<T> pipe : pipes) {  
7             processed = pipe.process(processed);  
8         }  
9     }  
10    return processed;  
11 }  
12 }
```

Quelltext 5.5: Die generische Klasse Pipeline

Pre-, Demosaicing- und PostProcessor

Viele der notwendigen Verarbeitungsschritte haben die Gemeinsamkeit, für jeden Rohdatenwert durchgeführt werden zu müssen, womit für jeden dieser Schritte das Iterieren über alle Rohdatenwerte erforderlich ist. Um dies zu vermeiden, sind die Iteriervorgänge in die drei Klassen PreProcess, DemosaicingProcessor und PostProcessor ausgelagert, die das Interface Processor<T> implementieren. Diese iterieren einmalig über die übergebenen Rohdaten und führen für jeden Rohdatenwert sequenziell die notwendigen Verarbeitungsschritte aus.

Pre-, Demosaicing- und PostProcessorOperation

Die konkrete Implementierung eines Verarbeitungsschrittes erfolgt in einer von der abstrakten Klassen PreProcessorOperation, DemosaicingProcessorOperation oder PostProcessorOperation ableitenden Klasse. Eine Klasse, die beispielsweise von PreProcessorOperation ableitet, implementiert die Rohdatenzuordnung oder den Weißabgleich. Eine von DemosaicingProcessorOperation abgeleitete Klasse implementiert hingegen einen konkreten Demosaicing Algorithmus.

5.2.2 Verarbeitung von Rohdaten

Die Rohdaten einer DNG Datei werden durch eine Instanz der generischen Klasse Pipeline<T> verarbeitet, indem die Iterationsvorgänge von dieser ausgeführt werden, welche wiederum die einzelnen Verarbeitungsschritte ausführen. Damit die Verarbeitungsschritte mit der in Kapitel 3.4 vorgestellten Pipeline übereinstimmen, muss die Klasse Pipeline<T> konfiguriert werden. Zur Konfiguration sind mehrere Bestandteile notwendig:

- Implementierungen der einzelnen Verarbeitungsschritte in Form von Klassen, die von PreProcessorOperation, DemosaicingProcessorOperation und PostProcessorOperation ableiten
- Vorgänge, welche das Interface Process<T> implementieren und die einzelnen Verarbeitungsschritte ausführen

Implementierung eines Verarbeitungsschrittes

Davon abhängig, ob ein Verarbeitungsschritt vor oder nach dem Demosaicing durchgeführt werden muss (vgl. Kapitel 3.4), erfolgt die Implementierung des Verarbeitungsschrittes in einer von PreProcessorOperation oder PostProcessorOperation ableitenden Klasse. Die abstrakte Klasse DemosicingProcessorOperation ist wiederum für die Implementierung eines konkreten Demosaicing Algorithmus vorgesehen. Der Quelltext 5.6 verdeutlicht den Aufbau einer solchen Klasse anhand der Klasse PreProcessorOperation. Diese definiert eine abstrakte Methode, welche einen Rohdatenwert und einen Index erwartet. Eine von ihr abgeleitete Klasse wie z. B. WhiteBalancing implementiert die abstrakte Methode und liefert den verarbeiteten Rohdatenwert zurück. Der Index wird benötigt, um beispielsweise zu ermitteln, ob der Rohdatenwert von einem Rot-, Grün- oder Blaufilter erfasst wurde.

```

1 public abstract class PreProcessorOperation {
2     ...
3     public abstract int process(int sample, int index);
4 }
```

Quelltext 5.6: Die abstrakte Klasse PreProcessorOperation

Ausführung eines Verarbeitungsschrittes

Die einzelnen Verarbeitungsschritte werden für jeden Rohdatenwert von einer der Klassen PreProcessor, DemosaicingProcessor oder PostProcessor ausgeführt. Wie Quelltext 5.7 beispielhaft zeigt, implementiert jede dieser Klassen unter Angabe des Typparameters BufferedImage das generische Interface Processor<BufferedImage>. Die zu implementierende Methode der Zeile 6 erwartet damit eine Instanz der Klasse BufferedImage und liefert eine Rückgabewert des gleichen Typs zurück. Die Methode iteriert in Zeile 8 über alle Rohdatenwerte, die sich im BufferedImage Objekt befinden und führt in den Zeilen 9 und 10 die einzelnen Verarbeitungsschritte sequenziell aus.

```
1 public class PreProcessor implements Processor<BufferedImage> {
2
3     private List<PreProcessorOperation> operations;
4     ...
5     @Override
6     public BufferedImage process(BufferedImage bufferedImage) {
7         short[] samples = ((DataBufferUShort) bufferedImage.getRaster().
8             getDataBuffer()).getData();
9         for (int i = 0; i < samples.length; i++) {
10             for (PreProcessorOperation operation : operations) {
11                 samples[i] = (short) operation.process(samples[i] & 0xFFFF, i);
12             }
13         }
14     }
15 }
```

Quelltext 5.7: Die abstrakte Klasse PreProcessor



Konfiguration und Ausführung der Pipeline

Die Konfiguration der Verarbeitungspipeline erfolgt durch die Instanziierung der oben beschriebenen Klassen und ist in Quelltext 5.8 exemplarisch für das laufende Beispiel abgebildet. Über eine Instanz der Klasse DNGFile werden in den Zeilen 1 bis 5 alle notwendigen IFDs und Rohdaten abgefragt. Durch die Übergabe der entsprechenden Verarbeitungsschritte an die jeweiligen Processor<BufferedImage> Implementierungen, werden diese in den Zeilen 7 bis 16 konfiguriert. Die Verarbeitungsschritte werden dabei in der übergebenen Reihenfolge ausgeführt. Die vollständige Verarbeitungspipeline wird abschließend in den Zeilen 18 bis 23 analog zu den Zeilen 7 bis 16 gebildet. Der Methodenaufruf in Zeile 25 reicht aus, um die gesamte Verarbeitungspipeline auszuführen.

Anhand dieses Beispiels wird die Stärke der Implementierung deutlich: Soll z. B. ein neuer Demosaicing Algorithmus realisiert werden, reicht es aus, eine neue Klasse zu erzeugen, die von DemosaicingProcessorOperation ableitet und die darin definierte Methode implementiert. Der neue Demosaicing Algorithmus kann, wie in Zeile 12

demonstriert, der Verarbeitungspipeline hinzugefügt werden. Dasselbe Vorgehen ist für weitere Verarbeitungsschritte möglich.

```

1 DNGFile dngFile = new DNGFile("C:\\running_example.dng");
2 ImageFileDirectory ifd = dngFile.get0thImageFileDirectory();
3 ImageFileDirectory rawIFD = dngFile.getRAWImageFileDirectory();
4 ImageFileDirectory exifIFD = dngFile.getExifImageFileDirectory();
5 BufferedImage rawImage = dngFile.getRAWImage();
6
7 PreProcessor preProcessor = new PreProcessor(
8     new RawMapping(ifd, rawIFD, exifIFD),
9     new WhiteBalancing(ifd, rawIFD, exifIFD)
10 );
11 DemosaicingProcessor demosaicingProcessor = new DemosaicingProcessor(
12     new BiCubic(ifd, rawIFD, exifIFD)
13 );
14 PostProcessor postProcessor = new PostProcessor(
15     new ColorSpaceTransformation(ifd, rawIFD, exifIFD)
16 );
17
18 Pipeline<BufferedImage> pipeline = new Pipeline<>(
19     preProcessor,
20     demosaicingProcessor,
21     new ImageCroppingProcessor(rawIFD),
22     postProcessor
23 );
24
25 BufferedImage processedImage = pipeline.process(rawImage);

```

Quelltext 5.8: Beispielhafte Konfiguration und Ausführung einer Verarbeitungspipeline am laufenden Beispiel

5.3 Grafische Benutzeroberfläche

Der DNG-Reader und der DNG-Processor stellen Funktionalitäten zur Verfügung, die das Einlesen und Verarbeiten einer DNG Datei ermöglichen. Gesteuert werden diese Funktionalitäten über die grafische Benutzeroberfläche, die sich aus verschiedenen Bedien- und Anzeigeelementen zusammensetzt.

Die grafische Benutzeroberfläche ist mittels JavaFX realisiert, das zur Implementierung von grafischen Java Anwendungen eingesetzt wird. Über diese kann im Dateisystem navigiert und eine DNG Datei selektiert werden, welche durch die anschließende Wahl eines gewünschten Demosaicing Algorithmus verarbeitet und angezeigt wird. Sollte eine DNG Datei nicht ausgelesen oder verarbeitet werden können, informiert die grafische Benutzeroberfläche den Nutzer durch eine entsprechende Fehlermeldung über diesen Ausnahmestatus. Mit Hinsicht auf ihre Erweiterbarkeit ist die Benutzeroberfläche modular aufgebaut.

5.3.1 Unterteilung der Benutzeroberfläche

Die Benutzeroberfläche ist in verschiedene Tabs unterteilt, die jeweils einen anderen Anwendungsfall berücksichtigen. Der Tab "Bibliothek", welcher den Einstiegspunkt der Anwendung bildet, ist in Abbildung 5.5 zu sehen. Dieser dient zur Navigation im Dateisystem und zur Vorschau der verfügbaren DNG Dateien. Durch die Wahl eines Verzeichnisses im oberen linken Seitenpanel, werden in der zentralen Übersicht die primären Vorschaubilder aller im Verzeichnis enthaltenen DNG Dateien angezeigt. Diese werden asynchron geladen, um das Blockieren der Benutzeroberfläche zu vermeiden. Ein Klick auf ein Vorschaubild lädt die in der DNG Datei enthaltenen IFDs nach  den Tags und listet sie im unteren linken Seitenpanel auf. Bei einem Doppelklick hingegen sind zwei Szenarien möglich: Die DNG Datei kann nicht ausgelesen oder verarbeitet werden und es öffnet sich der in Abbildung 5.6 gezeigte Hinweisdialog mit einer entsprechenden Meldung. Andernfalls öffnet sich der in Abbildung 5.7 visualisierte Interpolationsdialog, welcher das Vorschaubild in voller Auflösung wiedergibt und die Wahl eines Demosaicing Algorithmus erlaubt.

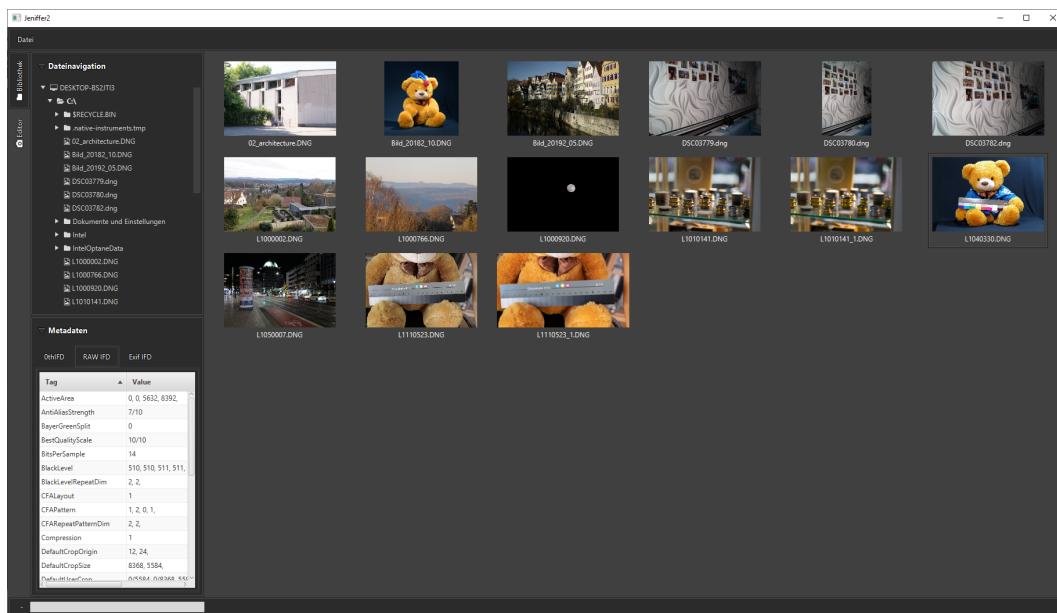


Abbildung 5.5: Tab "Bibliothek" des RAW Konverters

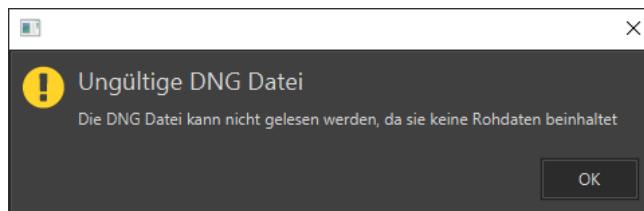


Abbildung 5.6: Hinweisdialog des RAW Konverters

5.3. Grafische Benutzeroberfläche

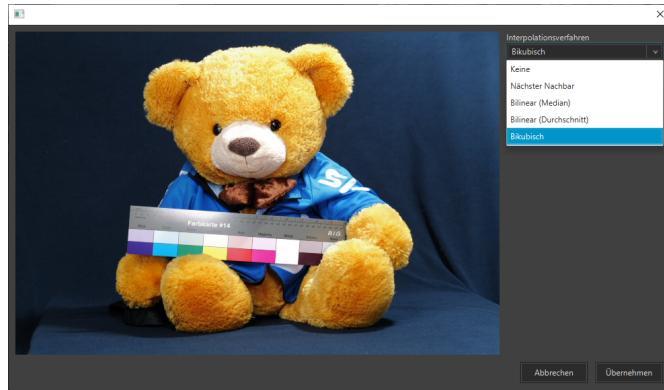


Abbildung 5.7: Interpolationsdialog des RAW Konverters

Mit der Übernahme des ausgewählten Demosaicing Algorithmus beginnt die Verarbeitung der DNG Datei, was zu einem Wechsel in den Tab "Editor" führt. Dieser ist in Abbildung 5.8 dargestellt und für das Anzeigen und Editieren der verarbeiteten Aufnahme vorgesehen. Über das Mausrad oder die dafür vorgesehenen Buttons kann die Anzeige der Aufnahme vergrößert, verkleiner oder auf dem maximal verfügbaren Platz skaliert werden. Außerdem lassen sich die auf den Wertebereich $[0,1]$ normierten RGB Werte der Aufnahme im oberen linken Seitenpanel anzeigen, indem der Mauszeiger über die Aufnahme gehovert wird. Ferner werden im unteren linken Seitenpanel die IFDs und Tags der verarbeiteten Aufnahme gelistet. Da das Editieren einer Aufnahme nicht im Fokus der Arbeit liegt, sind dafür keine Funktionalitäten implementiert.

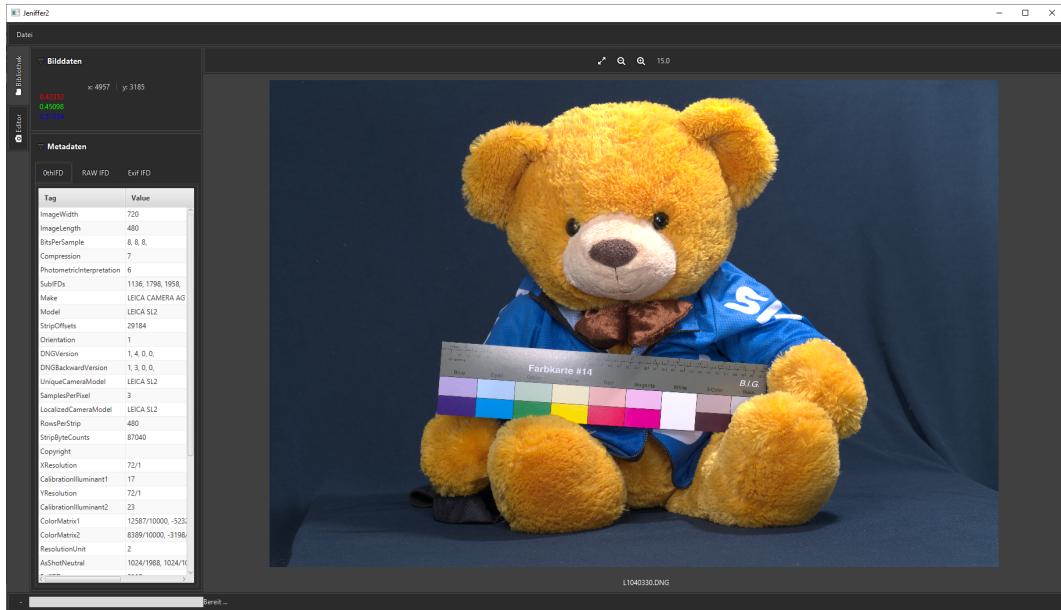


Abbildung 5.8: Tab "Editor" des RAW Konverters

5.3.2 Aufbau und Funktionsweise

Zur Realisierung grafischer Benutzeroberflächen existieren unterschiedliche Ansätze. Um die Erweiterbarkeit von JENIFFER2 gewährleisten zu können, ist ein Ansatz notwendig, der es einfach ermöglicht, der Benutzeroberfläche grafische Elemente und Features hinzuzufügen. Ein bekanntes Entwurfsmuster, das diese Anforderung realisiert und häufig genutzt wird, ist das Model–View–Presenter (MVP). Es unterteilt nach [Pot96] die einzelnen Bestandteile einer grafischen Benutzeroberfläche in drei Komponenten:

- Model: Beinhaltet das Datenmodell
- View: Ist für die Darstellung des Models zuständig und stellt Bedien- und Anzeigeelemente zur Verfügung - Enthält aber keinerlei Logik
- Presenter: Ist für das Interpretieren von Ereignissen zuständig (z. B. Klick auf einen Button) und Implementiert die Geschäftslogik, um das Model ggfs. zu manipulieren und anzuzeigen

Die Beziehungen der einzelnen Komponenten ist in Abbildung 5.9 visualisiert. Die View delegiert Ereignisse an den Presenter, welcher die Ereignisse interpretiert, Geschäftslogik ausführt, das Model ggfs. anpasst und die View aktualisiert. Der Vorteil dieser Architektur liegt in der klaren Trennung zwischen dem Model und der View, die dadurch beliebig ausgetauscht und wiederverwendet werden können [Pot96, S. 13f.].

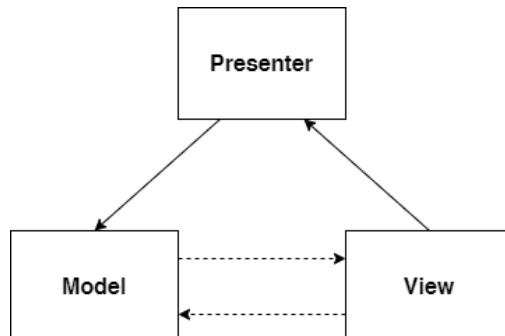


Abbildung 5.9: Das MVP Entwurfsmuster, angelehnt an [Pot96, S. 6]

Ein leichtgewichtiges Framework, das eine einfache Umsetzung des MVP Entwurfsmusters in einer JavaFX Anwendung ermöglicht, ist das Open Source Framework afterburner.fx. Es basiert auf dem Ansatz "Konvention vor Konfiguration" und kommt damit ohne Konfiguration aus, sofern die Konventionen eingehalten werden. [Bie] Als Konvention legt afterburner.fx fest, dass sich sowohl die View als auch der Presenter in einem gemeinsamen Java Package befinden und als Namenspräfix den Namen des Packages tragen.

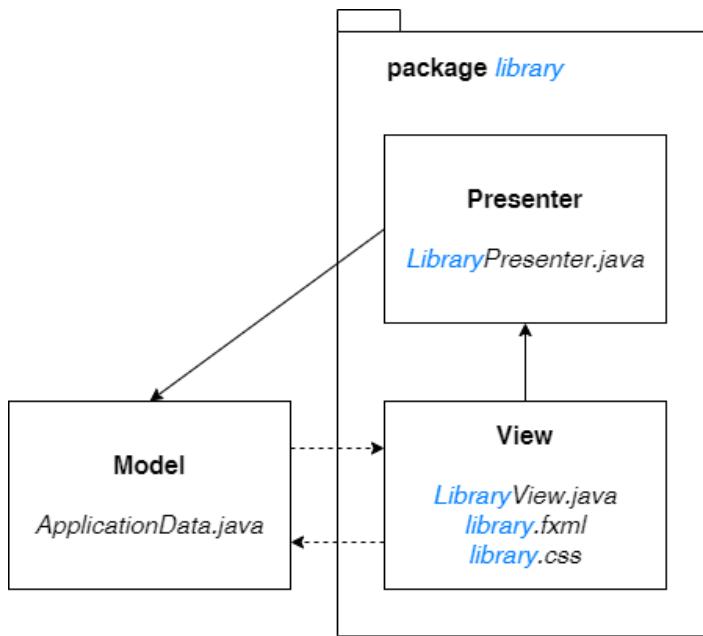


Abbildung 5.10: Bestandteile zur Realisierung des MVP Entwurfsmusters mittels afterburner.fx

Exemplarisch wird die Realisierung des MVP Entwurfsmusters im RAW Konverter mittels afterburner.fx anhand des Tabs "Bibliothek" erläutert. Als Hilfestellung soll die Abbildung 5.10 dienen, welche die einzelnen Bestandteile, die zur Umsetzung des Tabs benötigt werden, veranschaulicht. Darüber hinaus zeigt sie deren Zuordnung zu den Komponenten im MVP Entwurfsmuster und illustriert die festgelegten Konventionen.

View

Die View wird aus der Klasse `LibraryView`, der Extensible Markup Language (XML) Datei `library.fxml` sowie der Cascading Style Sheets (CSS) Datei `library.css` gebildet. Wie Quelltext 5.9 zeigt, beinhaltet die Datei `library.fxml` die Beschreibung der grafischen Benutzeroberfläche in Form der Auszeichnungssprache XML. Sie stellt die eigentliche View dar und kann über die CSS Datei gestaltet werden. Als einziger Bestandteil ist in Zeile 9 eine `SplitPane` angegeben, welche den Tab in mehrere vertikale Bereiche unterteilt, die vom Presenter dynamisch geladen werden. Die Beziehung zwischen der View und dem Presenter wird in der Zeile 7 hergestellt, indem der Pfad zum Presenter angegeben wird. Erzeugt wird die View von der Klasse `LibraryView`, die keine Implementierung enthält aber von der `afterburner.fx` Klasse `FXMLView` erbt.

Kapitel 5. Implementierung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?import javafx.scene.control.SplitPane?>
3 <?import javafx.scene.layout.BorderPane?>
4
5 <BorderPane fx:id="borderPane" styleClass="main"
6         xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/
7             fxml/1"
8         fx:controller="de.unituebingen.jeniffer2.main.tab.library.
9             LibraryPresenter">
10    <center>
11        <SplitPane fx:id="center"/>
12    </center>
13 </BorderPane>
```

Quelltext 5.9: Beispielhafte XML Datei des Tabs "Bibliothek" zur Beschreibung einer grafischen Komponente

Presenter

Der Presenter wird von der Klasse LibraryPresenter repräsentiert und ist in Quelltext 5.10 dargestellt. Wie in den Zeilen 9 und 10 gezeigt, erzeugt er weitere grafische Bestandteile wie z. B. das linke Seitenpanel, das selbst wiederum aus einem Model, einer View und einem Presenter besteht. Durch das Lauschen auf Ereignisse kann der Presenter auf diese reagieren und ggfs. Geschäftslogik ausführen sowie das Model aktualisieren. In diesem konkreten Beispiel lauscht der Presenter in Zeile 13 auf den Doppelklick eines Vorschaubildes und aktualisiert in der darauffolgenden Zeile das Model. Durch die Zeile 15 ist er außerdem für das Öffnen des Interpolationsdialogs zuständig.

```
1 public class LibraryPresenter implements Initializable {
2
3     @FXML private SplitPane center;
4     @Inject private WorkflowManager workflowManager;
5     @Inject private ApplicationData applicationData;
6
7     @Override
8     public void initialize(URL url, ResourceBundle resourceBundle) {
9         AsideView asideView = new AsideView();
10        WorkspaceView workspaceView = new WorkspaceView();
11        center.getItems().addAll(asideView.getView(), workspaceView.getView());
12
13        workflowManager.doubleClickedImageProperty().addListener((observable,
14            oldValue, newValue) -> {
15            applicationData.setTiffReader(newValue);
16            InterpolationDialogView dialogView = new InterpolationDialogView();
17            ...
18        });
19    }
}
```

Quelltext 5.10: Beispielhafter Presenter des Tabs "Bibliothek"

Model

Das Model ist durch die Klasse `ApplicationData` festgelegt und wird zwischen unterschiedlichen Presentern der grafischen Benutzeroberfläche geteilt. Diese beinhaltet anwendungsweite Daten wie z. B. die DNG Datei, welche verarbeitet wurde oder werden soll. Innerhalb dieser Klasse wird außerdem die Kommunikation mit dem DNG-Reader und dem DNG-Processor über die in Kapitel 4.2 vorgestellte Fassade realisiert.

6 Vergleich

Wie bereits in Kapitel 2.2.2 erläutert wurde, hat das Demosaicing einen wesentlichen Einfluss auf die Qualität einer digitalen Aufnahme [Luk09, S. 13]. Außerdem wirkt es sich, abhängig von der Komplexität des gewählten Demosaicing Algorithmus, auf das Laufzeitverhalten der Verarbeitungspipeline aus.

Durch den visuellen Vergleich unterschiedlicher Demosaicing Algorithmen wird in diesem Kapitel der Einfluss des Demosaicings auf die Image Quality (IQ) einer digitalen Aufnahme am laufenden Beispiel verdeutlicht. Der RAW Konverter realisiert dazu die erforderlichen Funktionalitäten, da er unterschiedliche Demosaicing Algorithmen implementiert und die Wahl eines Algorithmus erlaubt. Ferner wird die Auswirkung des Demosaicings auf das Laufzeitverhalten der Verarbeitungspipeline betrachtet und es werden die Ausführungszeiten der einzelnen Verarbeitungsschritte miteinander verglichen.

6.1 Demosaicing Algorithmen

Der Vergleich wird an den vier Demosaicing Algorithmen durchgeführt, die vom RAW Konverter implementiert werden. Dabei handelt es sich um nicht-adaptive Algorithmen, die nachfolgend gemäß [Wal05, S. 97ff.] kurz zusammengefasst sind:

- Pixelwiederholung: Fasst drei benachbarte Pixel verschiedener Farben zu einem Pixel zusammen (vgl. Kapitel 2.2.2)
- Bilineare Interpolation: Berechnet den Mittelwert von vier benachbarten Pixeln einer Farbe und fasst diese zu einem Pixel zusammen
- Median Interpolation: Ermittelt den Median von vier benachbarten Pixeln einer Farbe und fasst diese zu einem Pixel zusammen
- Bikubische Interpolation: Ermittelt die fehlenden RGB Werte aus 16 benachbarten Pixeln unter Zuhilfenahme eines Polynoms dritten Grades

Zur Durchführung des Vergleichs wird das laufende Beispiel zunächst mit den vier Demosaicing Algorithmen verarbeitet. Anschließend wird ein markanter Bereich der verarbeiteten Aufnahmen ausgewählt und einander gegenübergestellt. Durch die Gegenüberstellung werden auffällige Unterschiede, welche durch die verschiedenen Demosaicing Algorithmen verursacht werden und die IQ beeinflussen, hervorgehoben.

Kapitel 6. Vergleich

Ein für die Gegenüberstellung besonders geeigneter Bereich ist aus Betrachtersicht das linke Auge des Bären aus dem laufenden Beispiel. An diesem markanten Bereich befinden sich vereinzelt viele helle und feine Strukturen (Haare) auf einem vergleichsweise dunklen und gleichmäßigen Hintergrund (Auge).

Abbildung 6.1 zeigt für jeden Demosaicing Algorithmus den markanten Bereich in 600 facher Vergrößerung. Deutlich hervor sticht die Pixelwiederholung in Abbildung 6.1 (a). Diese schneidet im Vergleich zu den anderen Demosaicing Algorithmen am

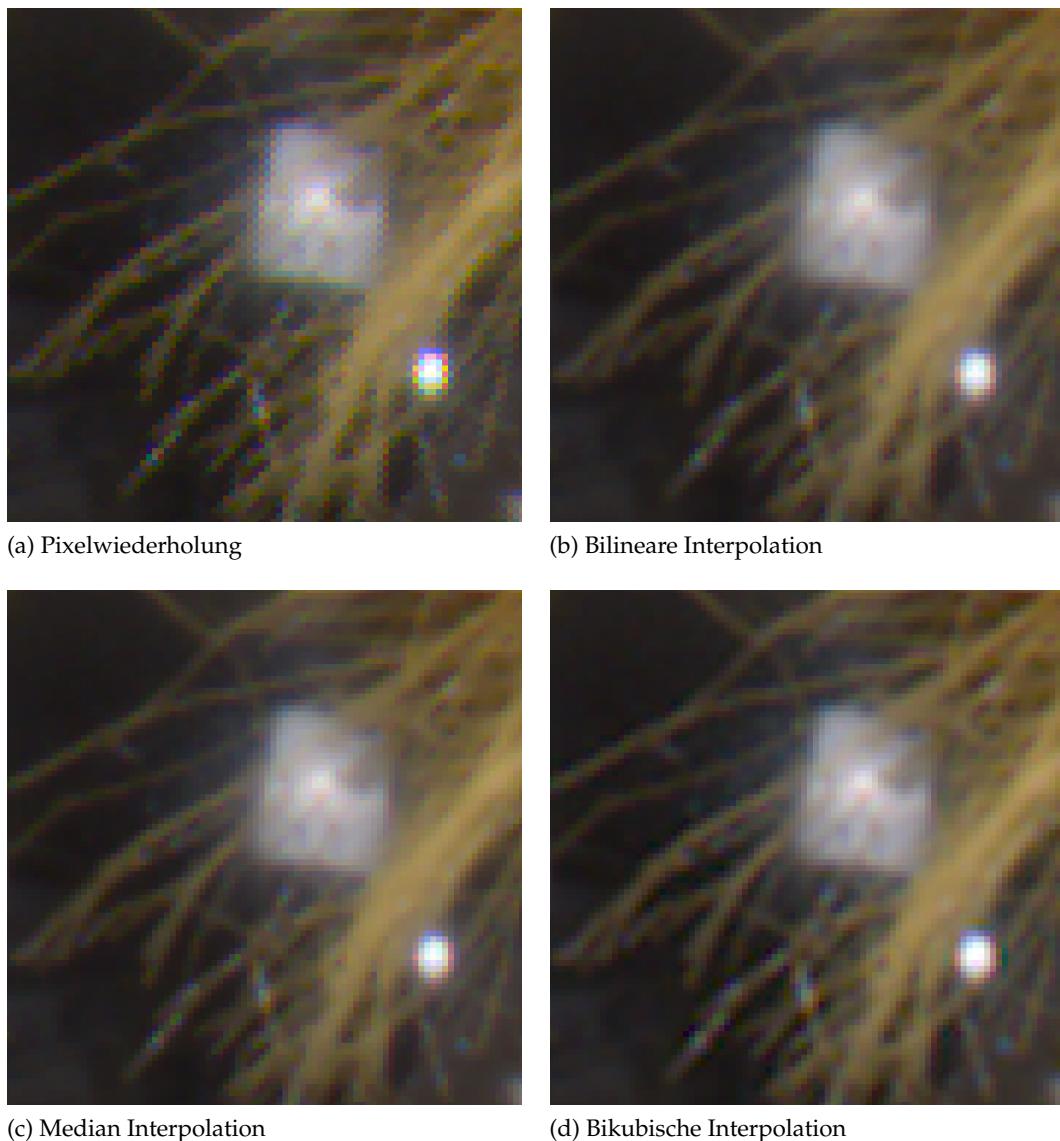


Abbildung 6.1: Gegenüberstellung verschiedener Demosaicing Algorithmen anhand einer markanten Stelle des laufenden Beispiels

6.2. Zeitverhalten der Verarbeitungsschritte

schlechtesten ab, ist aber auch am einfachsten zu realisieren. Besonders auffällig sind hierbei an Kanten treppenförmige Artefakte, die zu einem hohen Qualitätsverlust führen, da sie die Aufnahme pixelig und dadurch unscharf erscheinen lassen. Auch im dunklen Hintergrund sind ähnliche Unterschiede wahrnehmbar.

Zwischen der bilinearen Interpolation in Abbildung 6.1 (b) und der Median Interpolation in Abbildung 6.1 (c) sind im laufenden Beispiel keine nennenswerten Unterschiede erkennbar. Existieren in einer digitalen Aufnahme jedoch Ausreißer z. B. aufgrund defekter Pixel, werden diese durch die Median Interpolation herausgefiltert. Die bilineare Interpolation hingegen nutzt die Ausreißer zur Rekonstruktion fehlender Farbkomponenten umliegender Pixel, was sich negativ auf das Ergebnis auswirkt. Beide sind in der Umsetzung nur minimal komplexer als die Pixelwiederholung, liefern aber sichtbar bessere Ergebnisse.

Die bikubische Interpolation in Abbildung 6.1 (d) ist der Komplexeste der vier Demosaicing Algorithmen und bringt auf den ersten Blick ähnliche Ergebnisse hervor wie die bilineare und die Median Interpolation. Betrachtet man jedoch die Kante eines einzelnen Haars, sind wichtige Unterschiede erkennbar: Bei der bikubischen Interpolation weisen die Kanten einen höheren Kontrast auf, was den subjektiven Schärfeeindruck der Aufnahme verbessert. Demnach erzeugt die bikubische Interpolation damit für dieses Beispiel das beste Ergebnis, gefolgt von der Median Interpolation und der bilinearen Interpolation.

6.2 Zeitverhalten der Verarbeitungsschritte

Damit das Zeitverhalten aller Verarbeitungsschritte verglichen werden kann, müssen die Ausführungszeiten der einzelnen Verarbeitungsschritte zunächst gemessen werden. Die Messung erfolgt, indem die Differenz der Systemzeit nach und vor der Ausführung eines Verarbeitungsschrittes bestimmt wird. Gemessen und verglichen werden die Ausführungszeiten aller Verarbeitungsschritte, die am laufenden Beispiel durchgeführt werden müssen. Für das Demosaicing erfolgt die Messung für alle vier Demosaicing Algorithmen des RAW Konverters.

Von der Messung ausgenommen werden Zeiten, welche durch die Instanziierung der Klassen sowie das Einlesen der DNG Datei entstehen. Darüber hinaus ist zu beachten, dass die Ausführungszeiten von der Anzahl  Pixel sowie von den Systemeigenschaften abhängig sind.

6.2.1 Messung

Zur Messung der Ausführungszeiten wird die gesamte Verarbeitungspipeline unter gleichen Systemeigenschaften für jeden Demosaicing Algorithmus 10 mal ausgeführt. Anschließend wird der Mittelwert bestimmt. Daraus ergeben sich jeweils 10 Messungen für die vier unterschiedlichen Demosaicing Algorithmen und jeweils

Kapitel 6. Vergleich

	Rohdaten-zuordnung [ms]	Weiß-abgleich [ms]	Demo-saicing [ms]	Farbraum-transformation (davon Gamma-korrektur) 	Summe [ms]
Pixel-wiederholung	1375,3	525,3	761,7	4876,1 (4238,7)	7538,4
Bilineare Interpolation	1407,1	525	1117,7	4755,4 (4154,6)	7805,2
Median Interpolation	1373	521,8	2309,6	5153,2 (4549,8)	9357,6
Bikubische Interpolation	1361,9	520,4	3288,5	5137 (4525,9)	10307,8
Mittelwert	1381,1	523,1	-	4980,4 (4367,3)	-

Tabelle 6.1: Ausführungszeiten der einzelnen Verarbeitungsschritte für jeden Demosaicing Algorithmus

40 Messungen für alle anderen Verarbeitungsschritte. Gemessen wird bei einer Bildauflösung von 8.424 mal 5.632 Pixel (= 47.443.968 Pixel) unter den folgenden Systemeigenschaften:

- Windows 10 - 64-Bit Betriebssystem
- Intel Core i5-8265U CPU (4 Rechenkerne, 8 Threads), 1,60GHz (1,80GHz mit Turbo Boost)
- 16GB Arbeitsspeicher, davon 15,8GB verwendbar

Die Tabelle 6.1 zeigt den Mittelwert der Messungen für jeden Demosaicing Algorithmus. Die vollständigen Messungen können dem Anhang A entnommen werden. Der Bildzuschnitt ist nicht angegeben, da dieser aus einem einzigen Methodenaufruf besteht, der keine Ausführungszeit beansprucht. Für die Rohdatenzuordnung, den Weißabgleich und die Farbraumtransformation ist außerdem in der letzten Zeile unabhängig vom Demosaicing Algorithmus der Mittelwert aller 40 Messungen angegeben.

6.2.2 Auswertung

Betrachtet werden zunächst die Ausführungszeiten der verschiedenen Demosaicing Algorithmen, die in Abbildung 6.2 in Form eines Balkendiagramms visualisiert sind. Wie erwartet benötigt die Pixelwiederholung mit 761,7ms die geringste Ausführungszeit und die bikubische Interpolation mit 3288,5ms die höchste Ausführungszeit, da diese das komplexeste Verfahren ist und die meisten Rechenoperationen durchführen muss. Aus dem Diagramm geht außerdem deutlich hervor, dass die Ausführungs-

6.2. Zeitverhalten der Verarbeitungsschritte

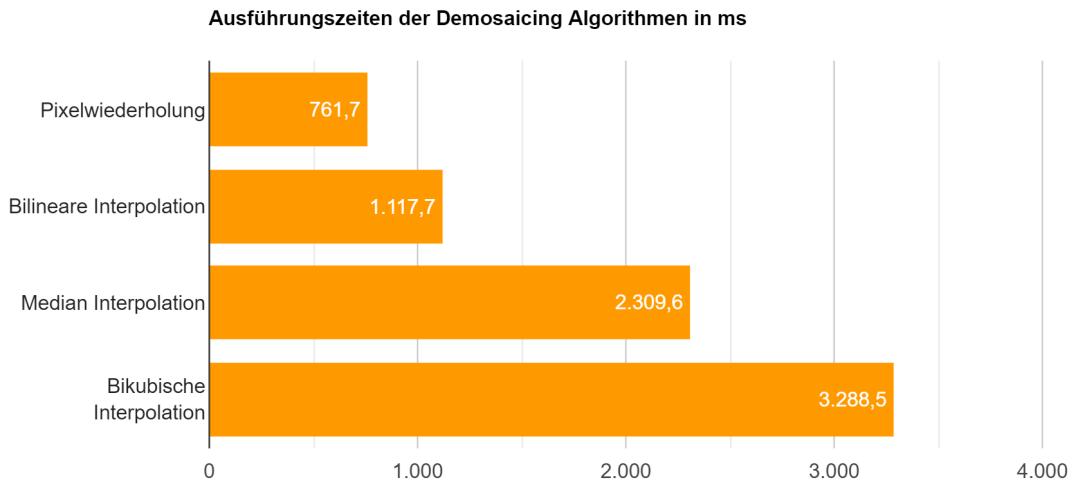


Abbildung 6.2: Ausführungszeiten der Demosaicing Algorithmen

zeit steigt umso besser das erzeugte Ergebnis eines Demosaicing Algorithmus ist (vgl. Kapitel 6.1). Interessant ist, dass die Median Interpolation etwa doppelt so lange benötigt wie die bilineare Interpolation. Dies ist vermutlich auf eine unperformante Implementierung der Median Interpolation zurückzuführen, die zur Bestimmung des Medians zunächst alle Elemente sortiert. Hier besteht demnach Optimierungsbedarf.

Werden die Ausführungszeiten aller anderen Verarbeitungsschritte miteinander verglichen, die in Abbildung 6.3 ebenfalls als Balkendiagramm veranschaulicht sind, fällt auf, dass die Farbraumtransformation mit 4980,4ms die längste Ausführungszeit einnimmt. Diese besteht zwar zur Transformation des Kamerafarbraums in den sRGB Farbraum aus einer einfachen Matrixmultiplikation, jedoch ist zusätzlich eine rechenintensive Gammakorrektur notwendig (vgl. Kapitel 3.4.5). Wie aus der Tabelle 6.1 sowie aus der Abbildung 6.3 hervorgeht, beansprucht die Gammakorrektur

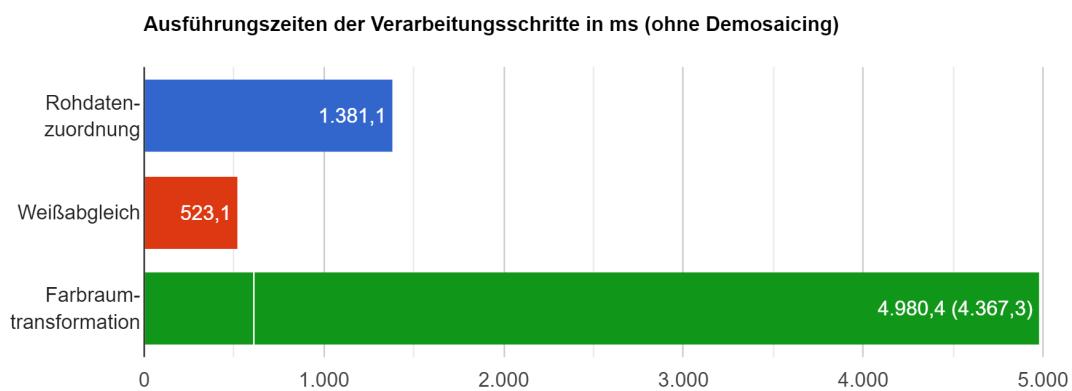


Abbildung 6.3: Ausführungszeiten der Verarbeitungsschritte (ohne Demosaicing)

Kapitel 6. Vergleich

4367,3ms der 4980,4ms. Dies sind rund 87,7 Prozent der gesamten Farbraumtransformation. Wird die bikubische Interpolation als Demosaicing Algorithmus verwendet, übersteigt die Gammakorrektur sogar das Demosaicing um das 1,3 fache. Abgesehen vom Bildzuschnitt benötigt der Weißabgleich, der lediglich aus einer Multiplikation besteht, mit 523,1ms die geringste Ausführungszeit. Zu erwarten wäre hier ebenfalls eine geringere Ausführungszeit, jedoch muss zusätzlich für jeden Pixel ermittelt werden, ob dieser von einem Rot-, Grün- oder Blaufilter erfasst wurde, um den richtigen Koeffizienten auswählen zu können. Etwa 2,6 mal so lange braucht die Rohdatenzuordnung, für die mehrere einzelne Teilschritte durchgeführt werden müssen.

Abbildung 6.4 (a) bis (d) veranschaulicht außerdem die Anteile der Ausführungszeiten jedes Verarbeitungsschrittes in der gesamten Verarbeitungspipeline je Demosaicing Algorithmus. Während die Pixelwiederholung mit 10,1 Prozent einen vergleichsweise geringen Anteil der Gesamtausführungszeit einnimmt, benötigt die bikubische Interpolation mit 31,9 Prozent einen deutlich höheren Anteil. Ferner wird

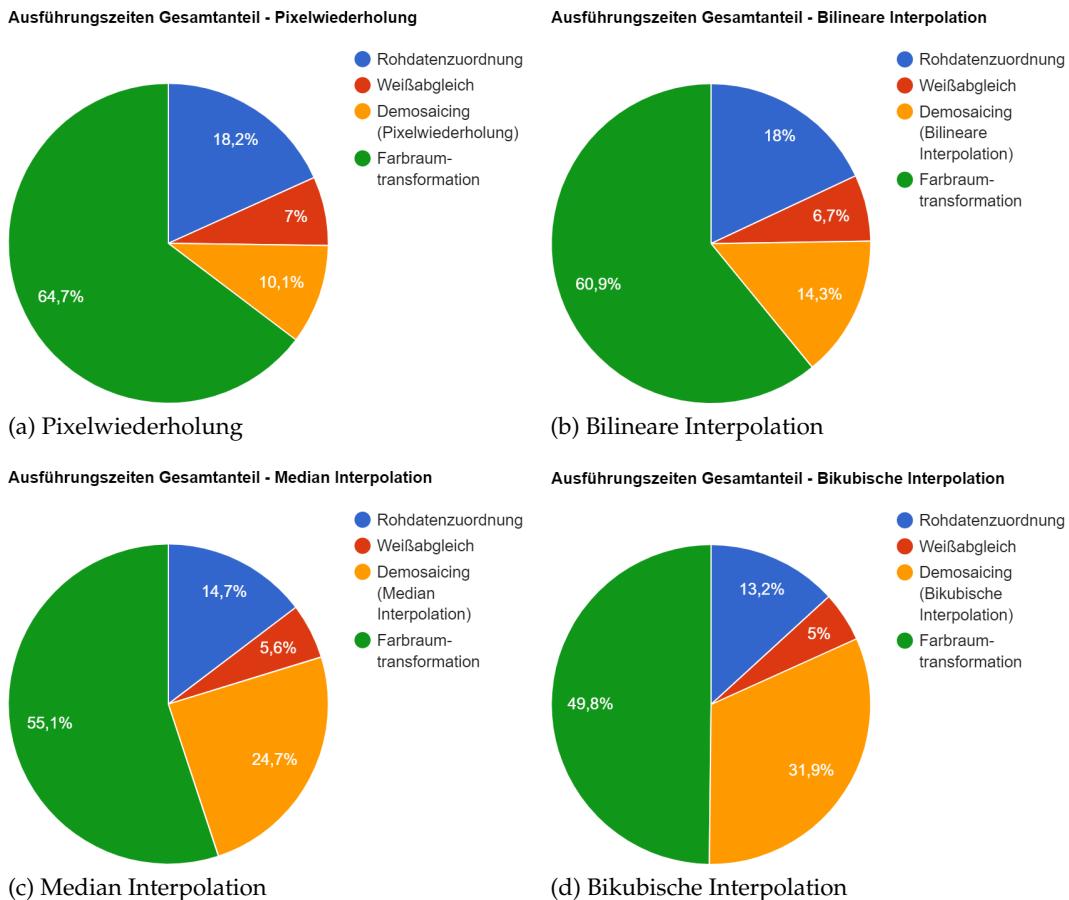


Abbildung 6.4: Anteile der Ausführungszeiten je Demosaicing Algorithmus

6.2. Zeitverhalten der Verarbeitungsschritte

hier erneut die Auswirkung der Farbraumtransformation einschließlich der Gammakorrektur auf die Gesamtausführungszeit deutlich: Selbst mit der bikubischen Interpolation als Demosaicing Algorithmus beansprucht die Farbraumtransformation nahezu 50 Prozent der Gesamtausführungszeit.

Im Allgemeinen sind die Ausführungszeiten relativ hoch. Die Verarbeitung der Rohdaten benötigt z. B. mit der bikubischen Interpolation durchschnittlich etwa 10,3s. Dies liegt daran, dass keine Nebenläufigkeit implementiert wurde. Stehen also mehrere Rechenkerne zur Verfügung wird in der aktuellen Implementierung trotzdem nur ein einzelner Rechenkern genutzt. Durch Nebenläufigkeit, welche in Java mittels Threads realisiert werden kann, lassen sich die Ausführungszeiten ggfs. verbessern. Ferner ist es in Java möglich, Berechnungen auf den Grafikprozessor auszulagern. Aufgrund der hohen Datenparallelität, welche durch Grafikprozessoren erreichbar ist, könnten die Ausführungszeiten weiter verkürzt werden



7 Fazit und Ausblick

Durch die Analyse des DNG Formats sowie die Implementierung des RAW Konverters JENIFFER2 konnten wichtige Erkenntnisse in Erfahrung gebracht werden. Diese werden abschließend zusammengefasst und offen gebliebene Fragen sowie der Ausblick der Arbeit erläutert.

7.1 Fazit

Durch die Analyse des DNG Formats konnten wichtige Fragen in Hinblick auf die RAW Konvertierung von DNG Dateien geklärt werden, die zur Implementierung eines RAW Konverters notwendig sind. Es wurde gezeigt, dass sich DNG Dateien sowohl von Kameras als auch von DNG Konvertern erzeugen lassen, wobei Unterschiede in der Kodierung der Rohdaten zwischen den erzeugten DNG Dateien existieren können. Diese sowie verschiedene Dateistrukturen müssen beim Auslesen von DNG Dateien berücksichtigt werden. Ferner hat sich ergeben, dass eine Vielzahl von nicht trivialen Verarbeitungsschritten erforderlich ist, um die Rohdaten einer DNG Datei zu einer digitalen Aufnahme zu verarbeiten. Welche Verarbeitungsschritte tatsächlich durchgeführt werden müssen, hängt dabei von den enthaltenen Tags einer DNG Datei ab. Die Verarbeitungsschritte konnten zum Teil auf einer theoretischen Ebene erläutert und durch die Implementierung des RAW Konverters anhand eines laufenden Beispiels praktisch veranschaulicht werden. Zusätzlich wurde verdeutlicht, dass verschiedene Eigenschaften das DNG Format charakterisieren. Neben der offengelegten Spezifikation sind es hauptsächlich die Eigenschaften, welche die herstellerübergreifende und universelle Nutzung des DNG Formats ermöglichen.

Das Ziel der Arbeit, einen RAW Konverter für das DNG Format auf Basis von JENIFFER zu entwickeln, konnte auf Grundlage der Analyse für ausgewählte DNG Dateien erfüllt werden. Dazu wurden alle vorher definierten funktionalen Anforderungen der Kategorien "Must", "Should" sowie "Could" realisiert. Die Korrektheit der Funktionalitäten für beliebige DNG Dateien kann nur durch eine hohe Testabdeckung geprüft werden. Diese lässt sich durch automatisierte Unitests erwirken, welche aufgrund der eingeschränkten Bearbeitungszeit im Rahmen dieser Arbeit nicht realisiert werden konnten und für die Weiterführung der Arbeit vorbehalten sind. In Hinsicht auf die geforderte Erweiterbarkeit und Wartbarkeit, ist eine klare ~~Auflage~~  über deren Einhaltung nur schwierig zu treffen. Es konnte jedoch

Kapitel 7. Fazit und Ausblick

begründet und verdeutlicht werden, dass die Einhaltung durch eine komponentenbasierte Architektur sowie eine generische und abstrakte Implementierung des RAW Konverters angestrebt wurde.

Unter Zuhilfenahme von JENIFFER2 konnten in einem abschließenden Vergleich von unterschiedlichen Demosaicing Algorithmen wichtige Erkenntnisse gewonnen werden, welche die Annahme bezüglich der Komplexität bestätigen. So machte der visuelle Vergleich des laufenden Beispiels, das mit unterschiedlichen Demosaicing Algorithmen verarbeitet wurde, die Auswirkungen des Demosaicings auf die Qualität einer digitalen Aufnahme deutlich. Konkret wurde gezeigt, dass die Pixelwiederholung unbefriedigende Ergebnisse erzeugt, aber am schnellsten durchgeführt werden kann. Die bikubische Interpolation hingegen ist am langsamsten, führt aber zu der schärfsten Aufnahme. Da nur einfache Demosaicing Algorithmen verglichen wurden, bleibt weiterhin die Frage offen, welche Resultate weitaus komplexere Demosaicing Algorithmen erzielen können. Darüber hinaus hat der Vergleich der Ausführungszeiten unterschiedlicher Verarbeitungsschritte ergeben, dass die Farbraumtransformation aufgrund der notwendigen Gammakorrektur einen erheblichen Anteil der gesamten Ausführungszeit beansprucht.

7.2 Ausblick

In Anbetracht der begrenzten Zeit sowie der Vielzahl an Tags, welche die DNG Spezifikation definiert, konnten lediglich die grundlegenden Verarbeitungsschritte berücksichtigt und implementiert werden. Es existieren jedoch weitere Verarbeitungsschritte wie z. B. Opcodes, welche für das Endresultat einer digitalen Aufnahme wichtig sind. Des Weiteren wurden von JENIFFER ausschließlich die einfachen, nicht-adaptiven Demosaicing Algorithmen übernommen. Komplexere Demosaicing Algorithmen könnten die Qualität einer digitalen Aufnahme signifikant steigern. In diesem Zusammenhang sind auch Methoden des maschinellen Lernens für das Demosaicing denkbar, an denen seit kurzem geforscht wird. Durch die erweiterbare Implementierung des RAW Konverters konnte eine gute Basis für dessen Weiterführung realisiert werden. Es bietet sich demnach an, weitere Verarbeitungsschritte sowie Demosaicing Algorithmen zu implementieren.

Neben dem Visualisieren von verarbeiteten Aufnahmen stellen RAW Konverter komplexe Funktionalitäten zur Verfügung, welche es ermöglichen, Aufnahmen zu editieren und zu optimieren. Ein nachträgliches Schärfen kann beispielsweise den subjektiven Schärfeeindruck einer Aufnahme verbessern. Ferner ist es häufig notwendig, den Weißabgleich oder die Belichtung nach der Erfassung der Aufnahme zu korrigieren. Solche Funktionalitäten sind grundlegende Bestandteile von RAW Konvertern, die jedoch nicht im Fokus der Arbeit lagen. Die Implementierung von JENIFFER2 berücksichtigt bereits die Integration solcher Funktionalitäten, für die eine Weiterführung der Arbeit vorstellbar wäre.

7.2. Ausblick

Obwohl die Mehrheit von Rohdaten auf einem Bayer CFA basieren, können in einer DNG Datei auch Rohdaten gespeichert werden, die von Bildsensoren mit anderen CFA Konfigurationen erfasst wurden. Diese können vom RAW Konverter bereits gelesen, jedoch nicht verarbeitet werden. Interessant wäre die Modifizierung des RAW Konverters zur Verarbeitung von Rohdaten, die nicht auf einem Bayer CFA basieren.

Abschließend kann die Aussage getroffen werden, dass das DNG ein fortschrittliches Format ist, das stetig erweitert und verbessert wird. Dies ist nicht zuletzt an der erst kürzlich aktualisierten DNG Spezifikation aus dem Mai 2019 zu erkennen. Es lohnt sich daher auch in naher Zukunft, dem DNG Beachtung zu schenken.

A Ausführungszeiten

Im Folgenden sind die gemessenen Ausführungszeiten, welche im Kapitel 6.2 ausgewertet wurden, aufgeführt.

	Rohdaten-zuordnung [ms]	Weiß-abgleich [ms]	Demo-saicing [ms]	Farbraum-transformation (davon Gamma-korrektur) [ms]	Summe [ms]
1	1338	501	758	4519 (3884)	7116
2	1304	518	747	5315 (4683)	7884
3	1391	525	779	4537 (3895)	7232
4	1365	515	759	5314 (4683)	7953
5	1365	527	766	5381 (4716)	8039
6	1422	531	772	3888 (3247)	6613
7	1386	531	735	5366 (4726)	8018
8	1461	547	787	5309 (4681)	8104
9	1379	530	772	4647 (4003)	7328
10	1342	528	742	4485 (3869)	7097
Mittelwert	1375,3	525,3	761,7	4876,1 (4238,7)	7538,4

Tabelle A.1: Ausführungszeiten der Verarbeitungsschritte mit der Pixelwiederholung als Demosaicing Algorithmus

Anhang A. Ausführungszeiten

	Rohdaten-zuordnung [ms]	Weiß-abgleich [ms]	Demosaicing [ms]	Farbraum-transformation (davon Gamma-korrektur) [ms]	Summe [ms]
1	1359	517	1104	5256 (4661)	8236
2	1347	523	1079	5305 (4714)	8254
3	1345	525	1127	5254 (4661)	8251
4	1351	512	1090	3817 (3226)	6770
5	1350	524	1103	3835(3238)	6812
6	1428	544	1136	4681 (4059)	7789
7	1453	531	1125	5400 (4797)	8509
8	1329	509	1110	4827 (4229)	7775
9	1430	538	1152	4552 (3936)	7672
10	1679	527	1151	4627 (4025)	7984
Mittelwert	1407,1	525	1117,7	4755,4 (4154,6)	7805,2

Tabelle A.2: Ausführungszeiten der Verarbeitungsschritte mit der bilinearen Interpolation als Demosaicing Algorithmus

	Rohdaten-zuordnung [ms]	Weiß-abgleich [ms]	Demosaicing [ms]	Farbraum-transformation (davon Gamma-korrektur) [ms]	Summe [ms]
1	1341	512	2328	5228 (4620)	9409
2	1344	514	2269	5290 (4679)	9417
3	1348	526	2294	5461 (4858)	9629
4	1402	515	2333	4634 (4038)	8884
5	1340	524	2282	5263 (4655)	9409
6	1427	528	2282	5341 (4737)	9578
7	1376	524	2275	5294 (4689)	9469
8	1352	518	2351	5284 (4687)	9505
9	1430	545	2398	5232 (4625)	9605
10	1370	512	2284	4505 (3910)	8671
Mittelwert	1373	521,8	2309,6	5153,2 (4549,8)	9357,6

Tabelle A.3: Ausführungszeiten der Verarbeitungsschritte mit der Median Interpolation als Demosaicing Algorithmus

	Rohdaten-zuordnung [ms]	Weiß-abgleich [ms]	Demo-saicing [ms]	Farbraum-transformation (davon Gamma-korrektur) [ms]	Summe [ms]
1	1415	534	3361	5439 (4760)	10749
2	1340	513	3297	5280 (4679)	10430
3	1382	518	3272	5263 (4659)	10435
4	1370	520	3296	5279 (4674)	10465
5	1323	514	3266	4497 (3879)	9600
6	1455	536	3347	5260 (4658)	10598
7	1328	514	3254	4481 (3887)	9577
8	1328	521	3264	5293 (4678)	10406
9	1326	510	3266	5322 (4725)	10424
10	1352	524	3262	5256 (4660)	10394
Mittelwert	1361,9	520,4	3288,5	5137 (4525,9)	10307,8

Tabelle A.4: Ausführungszeiten der Verarbeitungsschritte mit der bikubischen Interpolation als Demosaicing Algorithmus

B Installationsanleitung

Eine ausführbare Version von JENIFFER2 befindet sich auf der beigelegten CD im Verzeichnis /JENIFFER2/exec. Wie die darin enthaltene Datei ausgeführt werden kann, wird in Kapitel B.1 beschrieben.

Der gesamte Quelltext von JENIFFER2 befindet sich auf der beigelegten CD im Verzeichnis /JENIFFER2/code. Dieses ist in die beiden Unterverzeichnisse dng und ui unterteilt. In /JENIFFER2/code/dng befindet sich die Implementierung des DNG-Readers/-Processors. In /JENIFFER2/code/ui ist die grafische Benutzeroberfläche enthalten. Ferner ist der Quelltext auf einem Git Repository¹ des ZDV verfügbar. Wie der Code zu einer ausführbaren Datei kompiliert werden kann, wird in Kapitel B.2 beschrieben.

B.1 Ausführen

Um JENIFFER2 ausführen zu können, muss auf dem Rechner das **Java SE Runtime Environment 8 oder höher** oder das **Java SE Development Kit 12.0.2 oder höher** installiert sein. JENIFFER2 kann unter dieser Voraussetzung wie folgt ausgeführt werden:

1. Navigieren Sie in das Verzeichnis: /JENIFFER2/exec
2. Öffnen Sie in diesem Verzeichnis eine neue Konsole und führen Sie den folgenden Befehl aus: `java -jar JENIFFER2.jar`

Über die Angabe der optionalen Parameter `-Xms` und `-Xmx`, kann manuell der initial und maximal zur Verfügung stehende Arbeitsspeicher angegeben werden. Der Befehl `java -jar -Xms1024M -Xmx2048M JENIFFER2.jar` würde JENIFFER2 initial 1024MB und maximal 2048MB Arbeitsspeicher zuweisen.

B.2 Kompilieren

Um JENIFFER2 kompilieren zu können, muss auf dem Rechner das **Java SE Development Kit 12.0.2 oder höher** sowie **Apache Maven 3.6.2 oder höher** installiert sein. Da sich der DNG-Reader/-Processor sowie die grafische Benutzeroberfläche in

¹Erreichbar unter <https://gitlab.zdv.uni-tuebingen.de/groups/zdv-systeme/jeniffer2>

Anhang B. Installationsanleitung

unterschiedlichen Maven Modulen befinden, müssen sie separat kompiliert werden. Dazu wird wie folgt vorgegangen:

1. Navigieren Sie in das Verzeichnis: /JENIFFER2/code/dng
2. Öffnen Sie in diesem Verzeichnis eine neue Konsole und führen Sie den folgenden Befehl aus: mvn clean install
3. Navigieren Sie in das Verzeichnis: /JENIFFER2/code/ui
4. Öffnen Sie in diesem Verzeichnis eine neue Konsole und führen Sie den folgenden Befehl aus: mvn clean install
5. In dem Verzeichnis /JENIFFER2/code/ui/target befindet sich nun die Datei Jeniffer2-1.0-jar-with-dependencies.jar, welche beliebig umbenannt und wie in Kapitel B.1 beschrieben, ausgeführt werden kann

C Rechtlicher Hinweis

Dieses Produkt enthält die bei Adobe lizenzierte DNG-Technologie.

This product includes DNG technology under license by Adobe.

Literaturverzeichnis

- [ABF06] ANDREWS, PHILLIP, YVONNE J. BUTLER und JOE FARACE: *Raw workflow from capture to archives: A complete digital photographer's guide to raw imaging*. Focal Press, Amsterdam, 2006.
- [Ado92] ADOBE DEVELOPERS ASSOCIATION: *TIFF Revision 6.0 Final — June 3, 1992*. Technischer Bericht, 1992.
- [Ado18] ADOBE INC.: *Adobe Digital Negative Converter*. <https://helpx.adobe.com/de/photoshop/using/adobe-dng-converter.html>, 2018. Zugriff: 21.03.2020.
- [Ado19] ADOBE INC.: *Digital Negative (DNG) Specification - Version 1.5.0.0*. Technischer Bericht, San Jose, CA, 2019.
- [Ado20] ADOBE INC.: *Digital Negative (DNG) - The public archival format for digital camera raw data*. <https://helpx.adobe.com/de/photoshop/digital-negative.html>, 2020. Zugriff: 21.03.2020.
- [Ald93] ALDUS CORPORATION: *TIFF Technical Note 1: TIFF Trees*. Technischer Bericht, 1993.
- [AMCS96] ANDERSON, MATTHEW, RICARDO MOTTA, SRINIVASAN CHANDRASEKAR und MICHAEL STOKES: *Proposal for a Standard Default Color Space for the Internet - sRGB*. In: *Color Imaging Conference*, 1996.
- [Bay76] BAYER, BRYCE E.: *Color imaging array*, U.S. Patent 3 971 065, Juli 1976.
- [BB16] BURGER, WILHELM und MARK J. BURGE: *Digital Image Processing: An Algorithmic Introduction Using Java*. Texts in computer science. Springer, Berlin u.a., 2. Auflage, 2016.
- [BCK13] BASS, LEN, PAUL CLEMENTS und RICK KAZMAN: *Software Architecture in Practice*. Always learning. Addison-Wesley, Upper Saddle River, NJ, 3. Auflage, 2013.
- [Ber19] BURNS, ROY S.: *Billmeyer and Saltzman's Principles of Color Technology*. Wiley, Hoboken, NJ, 4. Auflage, 2019.
- [Bie] BIEN, ADAM: *afterburner.fx*. <http://afterburner.adam-bien.com/>. Zugriff: 20.04.2020.

Literaturverzeichnis

- [Büh04] BÜHLER, PETER: *MediaFarbe - analog und digital: Farbe in der Medienproduktion*. X.media.press. Springer, Berlin u.a., 2. Auflage, 2004.
- [Cam19] CAMERA AND IMAGING PRODUCTS ASSOCIATION: *CIPA DC-008-Translation-2019 - Exchangeable image file format for digital still cameras: Exif Version 2.32*. Technischer Bericht, 2019.
- [CCI92] CCIT: *T.81 Information Technology – Digital Compression And Coding Of Continuous-Tone Still Images–Requirements And Guidelines*. Technischer Bericht, 1992.
- [CF06] CHIKANE, VARSHA und CHIOU-SHANN FUH: *Automatic White Balance for Digital Still Cameras*. Journal for Information Science and Engineering, 22:497–509, 2006.
- [CT06] CHANG, LANLAN und YAP-PENG TAN: *Hybrid color filter array demosaicing for effective artifact suppression*. Journal of Electronic Imaging, 15(1):1 – 17, 2006.
- [Fra04] FRASER, BRUCE: *Understanding Digital Raw Capture*. Technischer Bericht, Adobe Systems Incorporated, San Jose, CA, 2004.
- [Ham92] HAMILTON, ERIC: *JPEG File Interchange Format Version 1.02*. Technischer Bericht, Milpitas, CA, 1992.
- [Har] HARVEY, PHIL: *ExifTool by Phil Harvey*. <https://exiftool.org/>. Zugriff: 25.03.2020.
- [Int09] INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS: *A Guide to the Business Analysis Body of Knowledge (BABOK Guide)*. IIBA, 2. Auflage, 2009.
- [ISO00] ISO/TC 42: *Photography — Electronic still picture imaging — Removable memory — Part 2: Image data format — TIFF/EP (ISO/DIS 12234-2)*. Technischer Bericht, 2000.
- [JGVH95] JOHNSON, RALPH, ERICH GAMMA, JOHN VLASSIDES und RICHARD HELM: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston u.a., 1995.
- [Lit01] LITWILLER, DAVE: *CCD vs. CMOS: facts and fiction*. Photonics Spectra, 35:154–158, Januar 2001.
- [LK04] LUKIN, ALEXEY und DENIS KUBASOV: *High-Quality Algorithm for Bayer Pattern Interpolation*. Programming and Computer Software, 30(6):347–358, 2004.
- [Luk09] LUKAC, RASTISLAV (Herausgeber): *Single-sensor imaging: Methods and Applications for Digital Cameras*, Band 9 der Reihe *Image processing series*. CRC Press, Boca Raton, Fla., 2009.

- [Mas04] MASCHKE, THOMAS: *Digitale Kameratechnik: Technik digitaler Kameras in Theorie und Praxis*. X.media.press. Springer, Berlin u.a., 2004.
- [McC92] McCAMY, C. S.: *Correlated color temperature as an explicit function of chromaticity coordinates*. Color Research & Application, 17(2):142–144, 1992.
- [Mia99] MIANO, JOHN: *Compressed Image File Formats - JPEG, PNG, GIF, XBM, BMP*. Addison-Wesley, Reading, Mass., 1. Auflage, 1999.
- [Nak06] NAKAMURA, JUNICHI (Herausgeber): *Image sensors and signal processing for digital still cameras*. Taylor & Francis, Boca Raton, Fla., 2006.
- [Nik16] NIKON: *Unterschiede zwischen mechanischen und elektronischen Verschlüssen*. https://www.nikonimgsupport.com/eu/BV_article?articleNo=000006443&configured=1&lang=de, 2016. Zugriff: 11.05.2020.
- [Pea10a] PEARSON, BARRY: *Camera details embedded in DNG*. <http://www.barrypearson.co.uk/articles/dng/profiles.htm>, 2010. Zugriff: 01.03.2020.
- [Pea10b] PEARSON, BARRY: *DNG and camera innovation*. <http://www.barrypearson.co.uk/articles/dng/innovation.htm#choices>, 2010. Zugriff: 20.03.2020.
- [Pea11] PEARSON, BARRY: *Benefits of DNG*. <http://www.barrypearson.co.uk/articles/dng/benefits.htm#archiving>, 2011. Zugriff: 18.03.2020.
- [Pot96] POTEL, MIKE: *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java*. <https://www.wildcrest.com/Potel/Portfolio/mvp.pdf>, 1996. Zugriff: 18.04.2020.
- [RA10] RUSSOTTI, PATRICIA und RICHARD ANDERSON: *Digital Photography Best Practices and Workflow Handbook: A Guide to Staying Ahead of the Workflow Curve*. Focal Press, Burlington, MA, 2010.
- [Raw] RAWTHERAPEE. <https://rawtherapee.com/>. Zugriff: 05.03.2020.
- [RSYD05] RAMANATH, RAJEEV, WESLEY E. SNYDER, YOUNGJUN YOO und MARK S. DREW: *Color image processing pipeline*. IEEE Signal Processing Magazine, 22(1):34–43, 2005.
- [Som11] SOMMERVILLE, IAN: *Software Engineering*. Pearson, München, 9. Auflage, 2011.
- [Sum14] SUMNER, ROB: *Processing RAW Images in MATLAB*. <https://www.cnba.it/contenuti/uploads/2016/03/Processing-RAW-Images-in-MATLAB-Sumner.pdf>, 2014. Zugriff: 10.04.2020.

Literaturverzeichnis

- [TAR02] TSAI, PING-SING, TINKU ACHARYA und AJAY RAY: *Adaptive fuzzy color interpolation*. Journal of Electronic Imaging, 11(3):293–305, 2002.
- [Too16] TOOMS, MICHAEL S.: *Colour Reproduction in Electronic Imaging Systems: Photography, Television, Cinematography*. Wiley, New York, 2016.
- [W3T] W3TECHS: *Usage statistics of JPEG for websites*. <https://w3techs.com/technologies/details/im-jpeg>. Zugriff: 21.03.2020.
- [Wal91] WALLACE, GREGORY K.: *The JPEG Still Picture Compression Standard*. Commun. ACM, 34(4):30–44, 1991.
- [Wal05] WALTER, THOMAS: *MediaFotografie - analog und digital: Begriffe, Techniken, Web*. X.media.press. Springer, Berlin u.a., 1. Auflage, 2005.
- [WGGK05] WALTER, THOMAS, CLAUDIA GROSCH, JOACHIM GROSS und MICHAEL KESSLER: *Mit JENIFFER zum digitalen Highend-Bild*. Informatik Spektrum, 28(5):437 – 438, 2005.