Freie Universität Berlin

Master These

Softwareentwicklung zur Vorverarbeitung von Videodaten auf der CineBox

Autor: Heiko Baumann
Matrikelnummer: 4425053

Betreuer der FU-Berlin:
Prof. Dr. Raúl Rojas
Betreuer des HHI:
Dipl. Ing. Christian Weissig

9. September 2015

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis oder ggf. in Fußnoten angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Unterschrift: _____

Datum: _____

Zusammenfassung

Softwareentwicklung zur Vorverarbeitung von Videodaten auf der CineBox

von Heiko Baumann

Am Fraunhofer Heinrich-Hertz-Institut wurde im Rahmen des TiMELab-Projektes ein immersiven Projektionssystem mit 180°-Leinwand entwickelt. Das Herzstück des Systems stellt die CineBox dar. Es handelt sich dabei um ein Linux-basiertes Ausgabe- und Hostsystem für die am Fraunhofer HHI entwickelte CineCard, welche eine Echtzeitbildverarbeitung vornimmt. Unter anderem ist es Aufgabe der CineBox eine Vorverarbeitung der zu projizierenden Videodaten vorzunehmen. Um den wachsenden Ansprüchen in Bezug auf die Auflösung der dargestellten Video- und Bilddaten und den damit verbundenen steigenden Datenmengen zu begegnen, wird derzeit eine Weiterentwicklung der CineCard durchgeführt. Für diese weiterentwickelte Version der CineCard wurde ein Programm entwickelt, dass die Vorverarbeitung der nötigen Daten realisiert.

Danksagung

Ich möchte mich ganz herzlich bei Herrn Prof. Dr. Raúl Rojas für die Betreuung meiner Abschlussarbeit vonseiten der Freien Universität Berlin bedanken. Ebenso möchte ich Herrn Dipl. Ing. Christian Weissig für die Möglichkeit, diese Arbeit am Fraunhofer Heinrich-Hertz-Institut durchzuführen, meinen Dank aussprechen. Außerdem möchte ich meinen Kollegen Fritjof Steinert und Arne Finn danken, welche mir immer mit gutem Rat zur Seite standen.
Auch möchte ich meinem Kommilitonen und guten Freund Severin Junker, für die Unterstützung bei dieser Arbeit aber auch während des gesamten Studiums, meinen herzlichsten Dank aussprechen.
Zu guter Letzt, gilt mein tiefster Dank meiner Freundin Katja Vieregge sowie meiner gesamten Familie , insbesondere meinen Eltern. Sie begleiteten und unterstützen mich während meines gesamten Studiums und ohne sie wäre es wohl nicht zu dieser Arbeit gekommen.

Vielen Dank !

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Diese Masterarbeit wurde am Fraunhofer-Institut für Nachrichtentechnik, dem Heinrich-Hertz-Institut (HHI), in der Abteilung Vision & Imaging Systems angefertigt. Die Projektgruppe Capture & Display Systems ist mit der Entwicklung von Soft- und Hardwarelösungen im Bereich der 2D und 3D Videoverarbeitung beauftragt. Genauer beschäftigt sich die Gruppe mit der Erstellung von 2D und 3D Panorama-Videoaufnahmen, deren Verarbeitung und Wiedergabe mit Hilfe eines Multiprojektionssystems geschieht. Die Aufnahme der Panorama-Videoaufnahmen wird durch die von der Gruppe entwickelte OmniCam realisiert. Für die Wiedergabe wurde im Heinrich-Hertz-Institut ein Panoramakino installiert, welches mit einer 180° Leinwand ausgestattet ist. Diese Leinwand ist konkav geformt, so dass beim Betrachter der Eindruck entsteht sich mitten im Geschehen zu befinden. Die Wiedergabe des Videomaterials erfolgt mit Hilfe der CineBox, welche mit mehreren CineCards bestückt ist. Die Vorverarbeitung der nötigen Daten wird mit Hilfe eines Matlab-Skripts erzielt. Um den wachsenden Ansprüchen, in Bezug auf die Auflösung der dargestellten Video- und Bilddaten und den damit verbundenen steigenden Datenmengen zu begegnen, wird derzeit eine Weiterentwicklung der CineCard durchgeführt. Um die neue Version der CineCard zu betreiben, ist ein Programm, welches die Vorverarbeitung der Daten entsprechend der Bedürfnisse der Neuauflage der CineCard vornimmt, zu entwickeln.

1.2 Aufgabenbeschreibung

Im Rahmen des TiMELab-Projektes des Fraunhofer HHI, eines immersiven Projektionssystem mit 180°-Leinwand, ist die Cinebox das Herzstück des Systems. Es handelt

sich dabei um ein Linux-basierten Playout-Server und Hostsystem für die am Fraunhofer HHI entwickelte CineCard, welche eine Echtzeitbildverarbeitung vornimmt. Unter anderem ist es Aufgabe der CineBox eine Vorverarbeitung der zu projizierenden Videodaten vorzunehmen. Für die in der Entwicklung befindlichen Neuauflage der CineCard, soll in dieser Masterarbeit ein Programm entwickelt werden, dass die Vorverarbeitung der nötigen Daten realisiert. Zu den Aufgaben des Programms gehört:

- Vorverarbeitung der Transformationskoordinaten, Alpha- und Blacklevel-Werte
- Bestimmung der Bildpunktadressen des Eingangsbildes
- Bestimmung der Interpolationskoeffizienten
- Erzeugen von Steuerkommandos
- Umgang mit verschiedenen Eingabeformaten für die Transformationskoordinaten
- Implementierung eines Simulationsmodus
- Steuerung des Programms per Konfigurationsdatei sowie per Parameterübergabe innerhalb eines Konsolenaufrufes
- Abwärtskompatibilität zur der sich bereits im Einsatz befindlichen CineCard-Version
- Verbesserung der Laufzeit
- Vorverarbeitung zur Erstellung einer Panoramavorschau der OmniCam

1.3 Gliederung

In Kapitel 2 wird das zum Verständnis dieser Arbeit notwendige Wissen vermittelt. Dazu wird ein Überblick über die mit dem Thema verbundenen Grundlagen gegeben und diese kurz erläutert.
Im folgenden 3. Kapitel wird auf das Projekt in dessen Rahmen diese Arbeit stattfindet beschrieben. Es werden die mit dem Projekt verbundenen und benötigten Hardware- und Softwarekomponenten erläutert.
Anschliessen werden in Kapitel 4 die Hintergründe und Vorgaben, auf deren Grundlage die Entscheidungen für das entwickelte Design getroffen wurden beschrieben.
Im 5. Kapitel wird die vorgenommene Implementierung zur Umsetzung der gestellten Aufgaben erläutert.
In Kapitel 6 wird eine Evaluierung des entwickelten Programms durchgeführt.

Innerhalb des Kapitels 7 wird eine Schlussfolgerung und abschließende Bewertung bezüglich der gestellten Aufgaben vorgenommen.
Abschliessend wird in Kapitel 8 ein mögliches weiteres Vorgehen skizziert

Kapitel 2

Grundlagen

2.1 Abbildungsfehler

Bei der Aufnahme von Bilddaten durch Kameras, wird nicht etwa eine perfekte Kopie des zu erfassenden Motivs erstellt, sondern lediglich eine verzerrte Abbildung des Motivs. Diese Verzerrung wird in der Optik als Abbildungsfehler oder Aberration bezeichnet und wird meist durch die in Kamerasystemen verwendeten Linsen und Blenden bzw. deren Aufbau verursacht. Eine weitere Ursache von solchen Abbildungsfehlern kann auch die Perspektive der Aufnahme sein. Eine spezielle Form dieser Abbildungsfehler wird Verzeichnung genannt. Dabei handelt es sich um Fehler die dadurch auftreten, das es bei anwachsender Bildhöhe zu einer zunehmenden Abweichung des Abbildungsmaßstabs zum realen (Soll-) Abbildungsmaßstabs kommt [1].

$$V = \frac{y'_{ist} - y'_{soll}}{y'_{soll}} \cdot 100\% \tag{2.1}$$

Mit der Gleichung 2.1, lässt sich die prozentuale Verzeichnung V bestimmen. Wobei y'_{ist} die von der Kamera aufgezeichnete (verzeichnete) Koordinate und y'_{soll} die eigentliche Koordinate (ohne Verzeichnung) im Bild darstellt. Somit kann man sagen, dass es sich bei der Verzeichnung um einen Lagefehler bestimmter Bildkoordinaten handelt. Der für V errechnete Wert, besitzt eine starke Aussagekraft darüber, welche Form der Verzeichnung vorliegt. Nimmt V einen positiven Wert an, so handelt es sich um eine kissenförmige Verzeichnung. Wird stattdessen für V ein negativer Wert bestimmt, so spricht man von einer tonnenförmigen Verzeichnung. Wie oben bereits erwähnt, hat die Anordnung der Linse und Blende in der Optik bzw. einem Objektiv einer Kamera Einfluss auf die Abbildung. Liegt die Blende vor der bildseitigen Hauptebene, d.h. zwischen Linse und Motiv, kommt es zu einer tonnenförmigen Verzeichnung. Dies ist in 2.1 dargestellt.

Die Linien eines aufgenommenen Gitternetzes werden hierbei zum Bildrand nach außen gebogen.

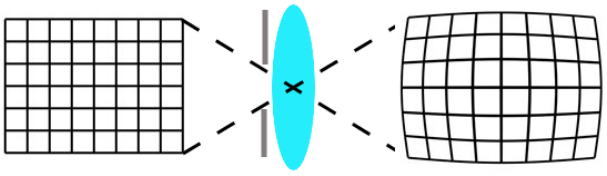


Abbildung 2.1: tonnenförmige Verzeichnung, Blende vor Linse

Im umgekehrten Fall, wenn die Blende sich hinter der Linse befindet, kommt es zu einer kissenförmigen Verzeichnung. Die Linien des Gitternetzes werden, wie in 2.2 zu sehen, zur Bildmitte hin nach innen gebogen.

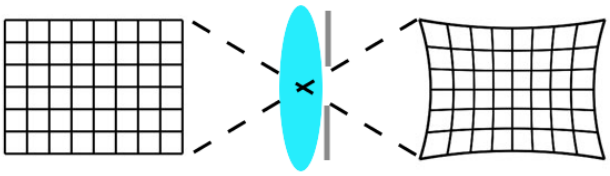


Abbildung 2.2: kissenförmige Verzeichnung, Linse vor Blende

Zu diesen Formen der Verzeichnung kommt noch die sogenannte wellenförmige Verzeichnung. Dabei handelt es sich um eine Kombination der beiden bereits erwähnten Verzeichnungen. Sie tritt bei viellinsigen Optiken, wie z.B. einem Weitwinkelobjektiv auf [2].

2.2 Aufbau eines digitalen Bildes

Mathematisch betrachtet, handelt es sich bei digitalen Bildern um Matrizen gefüllt mit numerischen Werten. Im Falle eines Graustufenbildes entspricht jeder Wert in der Matrix einem Punkt des Bildes. Wobei der Wert, die Information über die Intensität dieses Punktes darstellt. Je höher der Wert, desto heller stellt sich der Punkt im Bild da. Bei einem Grautonbild entspricht der Wert 0 einem absoluten Schwarz und der Wert 255 einem absoluten Weiss. Obwohl nur die Intensitätswerte numerisch in der Matrix enthalten sind, beinhaltet diese noch eine zusätzliche Information zu jedem Punkt, und zwar seine Position. So gibt die Zeile und die Spalte der Matrix, in der sich der Intensitätswert befindet, Auskunft über seine Lage im Bild. Üblicherweise wird diese Form der Adressierung von der oberen linken Bildecke begonnen, sodass mit Zeile 1, Spalte 1 auf den obersten linken Bildpunkt verwiesen wird. Anstelle von Zeile (row) und Spalte

(column) werden auch entsprechend die Koordinaten X und Y verwendet. Bei RGB-Farbbildern müssen drei Werte pro Zelle bzw. Kanal der Matrix gespeichert werden. So wird ein RGB-Farbbild, eigentlich durch eine dreidimensionale Matrix beschrieben, wobei jede Dimension der Matrix die Information für den R, G oder B -Kanal enthält. Bei den gespeicherten Werten in den einzelnen Kanälen, handelt es sich um die Intensitätswerte für die drei Grundfarben (auch Farbkanäle genannt) Rot, Grün und Blau. Die Größen der Intensitätswerte sind so normiert, dass sie zwischen 0 und 1 liegen. Durch die lineare Kombination dieser drei Farben, entsprechend ihrer Intensitäten, werden RGB-Farbbilder dargestellt.

2.2.1 Alphakanal

Im Alphakanal werden Informationen, welche Aussage über die Transparenz bzw. Opazität der eigentlichen Bilddaten treffen, gespeichert. Meist besitzt der Alphakanal die gleiche Bittiefe wie die entsprechend zugehörigen Farbkanäle des Bildes. Bei einem Grautonbild besitzt also der Alphakanal 256 Stufen, wobei ein Alphawert von 0 einer vollständigen Transparenz und ein Alphawert von 255 als absolut nicht transparent entspricht. Bei einer vollständigen Transparenz wird der zugehörige Grau- bzw. Farbwert als „unsichtbar“ deklariert und nicht dargestellt, wodurch an dieser Stelle des Bildes der Hintergrund sichtbar ist. Sind Bildpunkte als absolut nicht transparent(opak) deklariert, wird der Grau- bzw. Farbwert so dargestellt, dass der Hintergrund komplett verdeckt ist. Bei den Abstufungen zwischen den beiden Maximalwerten, wird eine entsprechende Überlagerung der Bildpunktwerte und des Hintergrundes durchgeführt. Dies wird in der Bild- und Videoverarbeitung genutzt, um ein sogenanntes „Alphablending“ durchzuführen. Hierbei werden verschiedene Bilder zu einem Gesamtbild überlagert. Um nun festzulegen, welches der zwei überlagerten Bilder stärker in die Darstellung des Gesamtbildes einfließt, wird der Alphawert genutzt.

2.3 Black Level

Der Helligkeitswert an der dunkelsten Stelle eines Bildes, wird auch als Black Level bezeichnet. Dabei handelt es sich um einen Wert zur Kalibrierung der Helligkeit, bei der Darstellung von Bildinhalten auf einem Display oder ähnlichem. Durch die Kalibrierung anhand des Black Level's soll sichergestellt werden, dass schwarze Bildinhalte bei der Ausgabe auch wirklich als schwarz dargestellt werden. So kann es bei falscher Helligkeitseinstellungen dazu kommen, dass beispielsweise schwarze Bildinhalte als Grau dargestellt werden, was den Gesamteindruck des Bildes minimiert. Ebenfalls kann es dazu kommen, dass dunkle Grauwerte als schwarz dargestellt werden und somit

in der Darstellung eventuell nicht mehr wahrnehmbar sind. Zur Ermittlung der richtigen Helligkeits- bzw. Black Level-Werte werden Musterbilder bzw. Testpanels verwendet.

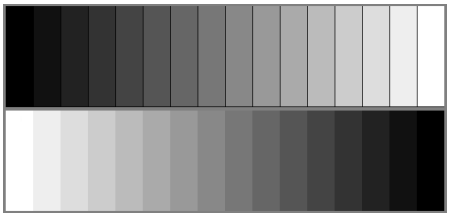


Abbildung 2.3: Graustufen-Testpattern

Anhand der der Darstellungungen, kann der Helligkeitswert so eingestellt werden, dass nur geplant sichtbare, dunkle Grautöne noch zu sehen sind und darunter liegen Werte als schwarz dargestellt werden.

2.4 Geometrische Transformation (Warping)

Geometrische Transformationen (engl. Warping bzw. Image Warping) nehmen keinen Einfluss auf die Intensitätswerte der Bildpunkte (auch Pixel genannt), sondern auf ihren Ort. Ganz allgemein kann man von einer Umsortierung der Pixel eines Eingangsbildes auf andere Positionen bzw. Koordinaten eines Zielbildes sprechen. Um die neuen Koordinaten für die Pixel im Zielbild zu bestimmen, wird ein mathematisches Modell in Form einer Transformationsfunktion (Abbildungsfunktion) herangezogen. Die Transformationsfunktion stellt dabei einen geometrischen Zusammenhang zwischen den Koordinaten x und y des Eingangsbildes G (auch Urbild) und den Koordinaten $x^{'}$ und $y^{'}$ des Zielbildes $G^{'}$ her. Bei der Transformationfunktion T handelt es sich um eine Matrix (Transformationsmatrix), welche mit der Pixelkoordinate, in Vektordarstellung multipliziert wird. Somit lässt sich der Zusammenhang von Pixelkoordinaten im Urbild und den Pixelkoordinaten im Zielbild wie folgt beschreiben.

$$\begin{pmatrix} x^{'} \\ y^{'} \end{pmatrix} = T \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.2}$$

2.4.1 Vorwärts- und Rückwärtstransformation

Ein Problem bei der geometrischen Transformation entsteht dadurch, dass die Raster vom Eingangsbild und Zielbild im Allgemeinen nicht übereinstimmen. Durch die Anwendung der Transformation kann es dazu kommen, dass die ganzzahligen Koordinaten des Pixels aus dem Eingangsbild, auf reelle Koordinaten in dem Zielbild abgebildet werden. So liegt der Pixel des Urbildes, neben dem Raster des Zielbildes,

weshalb eine Interpolation nötig wird um diese wieder in das Koordinatenraster zu überführen. Zudem können sich Eingangsbild und Zielbild in ihrer Auflösung, d.h. in ihrer Anzahl von Pixeln, unterscheiden. Bei einem angenommenen Verhältnis von 4:1, müsste eine Gruppe von 4 Pixeln des Urbilds auf 1 Pixel des Zielbildes überführt werden. Dies führt wiederum zu einer Notwendigkeit der Interpolation von Pixeln. Zudem würde bei der Vorwärtstransformation 2.2 unter Umständen nicht jedem Pixel des Zielbildes ein Wert zugewiesen werden können, was zu Lücken im Bild führt. Bei der Rückwärtstransformation wird vom Zielbild ausgegangen und für jedes Pixel dessen Ort im Eingangsbild bestimmt. Für diese Abbildung, wird die Inverse der Transformationsfunktion T^{-1} verwendet. Durch diese Herangehensweise können keine Bildlücken entstehen, weil für jedes Pixel im Zielbild, der entsprechende Ort im Urbild ermittelt wird.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = T^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.3}$$

Die Problematik der Interpolation bleibt jedoch bestehen, nur dass sie sich auf das Urbild verlagert.

2.4.2 Affine Transformationen

In manchen Fällen, lassen sich Abbildungsfehler bereits durch lineare Abbildungsgleichungen beschreiben. Diese Abbildungsgleichungen besitzen die Eigenschaft, zueinander parallele Geraden wieder in parallele Geraden zu überführen. So besteht zwischen dem Originalbild und dem transformiertem Bild noch eine gewisse *Verwandtschaft*. Aus diesem Grund werden lineare Abbildungsgleichungen auch als affine (verwandte) Transformationen bezeichnet [3].

2.4.2.1 Translation

Die Translation kann als gleichmäßige Verschiebung des Bildes beschrieben werden. Alle Bildpunkte werden um die gleiche Distanz, in die gleiche Richtung verschoben. Im Grunde handelt es sich um eine Addition des Verschiebungsvektors mit den Bildpunktkoordinaten. Jedoch kann durch die Einführung der homogenen Ebene und einer entsprechenden Translationmatrix die Translation mittels Multiplikation durchgeführt werden.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.4}$$

Durch 2.4 ergibt sich für $x' = x + dx$ und für $y' = y + dy$. Wobei die Pixelkoordinaten um dx auf der X-Achse und um dy auf der Y-Achse linear verschoben werden.

2.4.2.2 Skalierung

Durch die Skalierung wird eine Vergrösserung bzw. Verkleinerung des Eingangsbildes erreicht. Unter Zuhilfenahme des Skalierungsfaktors s werden ausgehend vom Koordinatenursprung $(0, 0)$ die neuen Pixelkoordinaten (x', y') abgebildet. Die Skalierung wird durch Multiplikation der Pixelkoordinaten mit der Transformationsmatrix durchgeführt.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.5}$$

So ergibt sich aus 2.5 für $x' = x \cdot sx$ und für $y' = y \cdot sy$. Werden die Skalierungsfaktoren mit dem gleichen Wert versehen $sx = sy$, so bezeichnet man dies als eine isotrophische Skalierung. Unterscheiden sich die Werte der Skalierungsfaktoren $sx \neq sy$, so spricht man von einer anisotrophischen Skalierung. Wird für die Skalierungsfaktoren ein Wert $s < 1$ gewählt, wird das Eingangsbild verkleinert und mit $s > 1$ wird das Bild vergrößert. Zu einer Sonderform der Skalierung, der Spiegelung, kommt es, wenn einer der beiden Skalierungsfaktoren sx, sy oder beide mit einem negativen Wert versehen werden.

2.4.2.3 Scherung

Bei der Scherung, handelt es sich um die Verschiebung eines Bildpunktes, beschränkt auf eine Dimension. So wird bei einer Scherung in X-Richtung nur die X-Koordinate des betreffenden Bildpunktes beeinflusst, die Y-Koordinate bleibt unverändert. Allgemein kann man sagen, dass die Scherung einen Pixel nur Zeilen-, oder Spaltenweise verschiebt. Die Verschiebung wird durch den Scherfaktor a angegeben. So kippt ein in X-Richtung geschertes Rechteck nach rechts und wird dabei zu einem Parallelogram [4].

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & ax & 0 \\ ay & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.6}$$

Will man eine Scherung in X-Richtung erreichen so wird in 2.6 $ay = 0$ gesetzt. Entsprechend wird $ax = 0$ gesetzt wenn eine Scherung in Y-Richtung erzielt werden soll.

2.4.2.4 Rotation

Bei der Rotation wird das Eingangsbild gedreht im Zielbild abgebildet. Diese Drehung bezieht sich auf den Koordinatenursprung $(0, 0)$. Der Winkel ϕ gibt an, um wie viel Grad der entsprechende Pixel um den Koordinatenursprung gedreht bzw. rotiert wird.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.7}$$

Wird für ϕ ein positiver Wert angenommen, so handelt es sich um eine Rotation gegen den Uhrzeigersinn. Entsprechend wird bei negativem ϕ im Uhrzeigersinn rotiert. Ist die Rotation mit einer Verschiebung (Translation) verbunden, so wird diese durch den Translationsvektor $\vec{v}$ definiert [5]. Es ergibt sich $T \cdot \vec{x} + \vec{v} = \vec{x}'$, wobei es sich bei $\vec{x}$ um die Bildpunktkoordinaten handelt. Durch die Addition des Verschiebungsvektors, wird nicht mehr der Koordinatenursprung als Bezugspunkt, sondern der durch $\vec{v}$ definierte Punkt, für die Rotation herangezogen. Die in $\vec{v}$ enthaltenden Koordinaten definieren also einen neuen Rotationspunkt um den der entsprechende Pixel $(\vec{x})$ rotiert wird.

2.5 Interpolation

Wie bereits in 2.4.1 erwähnt, kann durch die Transformation von Bildern nicht unbedingt jedem Bildpunkt im Zielbild, ein Bildpunkt des Urbildes zugewiesen werden, was als Lücken im Zielbild resultiert. Ebenfalls kommt es dazu, dass *durch die geometrische Abbildung T(bzw. T^{-1}) , diskrete Rasterpunkte im Allgemeinen nicht auf diskrete Bildpositionen im jeweils anderen Bild transformiert werden* [6]. Anders ausgedrückt, ergeben sich ausgehend von den Bildpunktkoordinaten (x, y), welche als ganze Zahlen vorliegen, durch die Transformation die Abbildungskoordinaten (x', y'), die als reelle Zahlen vorliegen. Um diesen Problemen zu begegnen, wurden verschiedene Interpolationsverfahren entwickelt. Allen Interpolationsverfahren ist gemein, dass zur Bestimmung

eines zu interpolierenden Bildpunktes, die nähere Umgebung des Bildpunktes betrachtet und einbezogen wird.

2.5.1 Nächster Nachbar Interpolation

Die wohl einfachste Form der Interpolation ist die Nächster Nachbar (engl. nearest neighbour) Interpolation, sie wird zum Teil auch als "zeroth-order interpolation" bezeichnet [7]. Das Verfahren setzt die Intensitätswerte des zu interpolierenden Bildpunktes, gleich denen des Bildpunktes auf den durch Runden der Koordinatenwerte verwiesen wird.

$$G^{'}(x^{'}, y^{'}) = G(round(x), round(y)) \tag{2.8}$$

Das Verfahren zeichnet sich durch seine Einfachheit aus und kann somit schnell berechnet werden. Jedoch hat diese Schlichtheit große Auswirkungen auf die Qualität des Ergebnisses. So sind im interpolierten Bild (siehe Abbildung 2.4) deutliche blockartige Strukturen wahrzunehmen, welche vor allem an geraden Linien als Stufen erkennbar werden [8].

Abbildung 2.4: vergrössertes Bild mit angewendeter Nächster Nachbar Interpolation

2.5.2 Bilineare Interpolation

"*Das Gegenstück zur linearen Interpolation im eindimensionalen Fall ist die sogenannte Bilineare Interpolation*" [6] im zweidimensionalen Fall. Die Bilineare Interpolation (auch "first-order interpolation" [7]) betrachtet dafür die nächsten vier Nachbarn des zu interpolierenden Bildpunktes $G(x^{'}, y^{'})$ und bezieht den Abstand zu ihnen als Gewichtung ein.

Die in der Darstellung 2.5 eingezeichneten Abstände dx und dy geben die Gewichtung an, mit welcher der Punkt $f(x_0, y_0)$ in die Berechnung zur Bestimmung des untersuchten

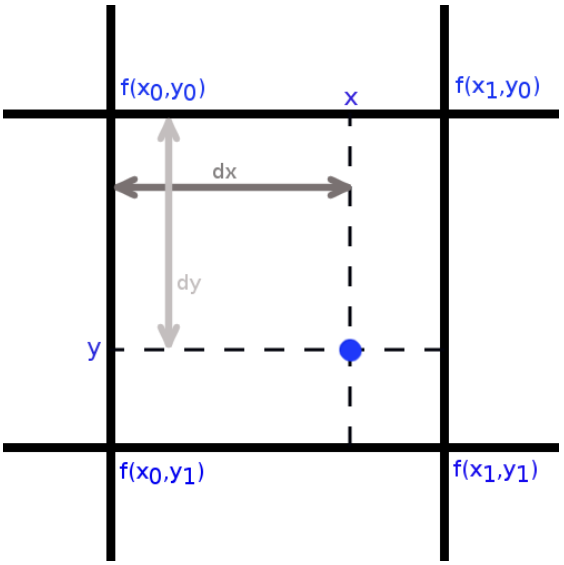


Abbildung 2.5: Darstellung der Bilinearen Interpolation

Bildpunktes einfließt. Nach diesem Prinzip sind die Abstände zu den verbleibenden Nachbarpunkten und somit ihre Gewichtung zu bestimmen. Der letztendliche Itensitätswert mit dem der untersuchte Punkt versehen wird, ergibt sich durch die gewichtete Summe der vier Nachbarpunkte.

$$\begin{aligned} G(x^{'}, y^{'}) = f(x_0, y_0) \cdot (1 - dx)(1 - dy) \\ + f(x_1, y_0) \cdot (dx)(1 - dy) \\ + f(x_0, y_1) \cdot (1 - dx)(dy) \\ + f(x_1, y_1) \cdot (dx \cdot dy) \end{aligned} \tag{2.9}$$

Abbildung 2.6: vergrössertes Bild mit angewendeter Bilinearer Interpolation

Das Verfahren der Bilinearen Interpolation stellt sich in der Durchführung der Berechnung komplexer als das des Nächsten Nachbarn dar. Dafür sind die groben blockartigen

Strukturen und die Stufen an geraden Linien kaum noch wahrnehmbar (siehe Abbildung 2.6) [8].

2.5.3 Bikubische Interpolation

Ähnlich wie bei der Bilinearen Interpolation, wo benachbarte Bildpunkte mit unterschiedlicher Gewichtung zur Bestimmung des untersuchten Pixelwertes einbezogen wurden, wird auch bei der Bikubischen Interpolation vorgegangen. Die Verfahren unterscheiden sich in der Anzahl der einbezogenen Nachbarpixel, welche für die Bestimmung des Ergebnisses verwendet werden. So wird die Menge, von den nächsten vier Nachbarpunkten der Bilinearen Interpolation, um zwölf Bildpunkte erweitert. Diese 12 Bildpunkte sind wiederum die nächsten benachbarten Punkte der Bilinearen Interpolation. Somit ergibt sich eine Gesamtmenge von 16 Bild- bzw. Nachbarpunkten, die zur Bestimmung des Ergebnisses der Bikubischen Interpolation mit einbezogen werden.

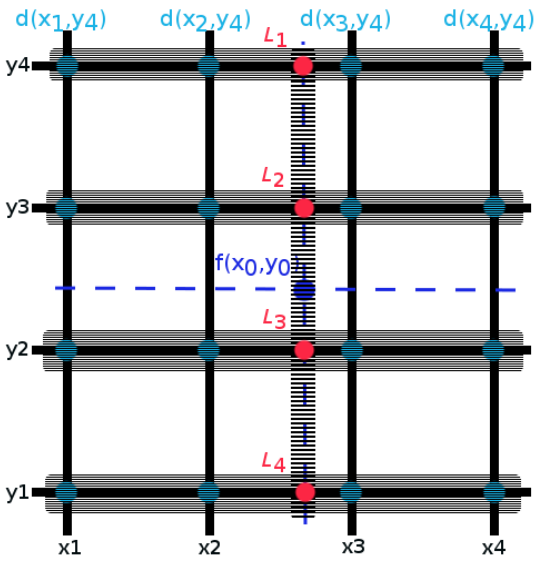


Abbildung 2.7: Darstellung der Bikubischen Interpoaltion

Mit Hinblick auf 2.7 ergibt sich die Gleichung 2.10 zur Berechnung der Bikubischen Interpolation [9].

für $m = \{1, 2, 3, 4\}$

$$
\begin{aligned}
l_m = {} & (d(x_4, y_m) - d(x_3, y_m) - d(x_1, y_m)) + d(x_2, y_m))(x_0 - x_2)^3 \\
& + (2d(x_1, y_m) - 2d(x_2, y_m) - d(x_4, y_m) + d(x_3, y_m))(x_0 - x_2)^2 \\
& + (d(x_3, y_m) - d(x_1, y_m))(x_0 - x_2) + d(x_2, y_m)
\end{aligned}
$$

$$
\begin{aligned}
f(x_0, y_0) = {} & (l_4 - l_3 - l_1 + l_2)(y_0 - y_2)^3 \\
& + (2l_1 - 2l_2 - l_4 + l_3)(y_0 - y_2)^2 \\
& + (l_3 - l_1)(y_0 - y_2) + l_2
\end{aligned}
\tag{2.10}
$$

Die Bikubische Interpolation ist in ihrer Berechnung bzw. Durchführung deutlich komplexer, als die vorangegangenen Verfahren. Allerdings erzielt sie auch eine deutlich höhere Qualität im Ergebnis (siehe Abbildung 2.8). Die groben blockartigen Strukturen und Stufen an Geraden, bekannt aus der Nächster Nachbar Interpolation, sind nicht mehr wahrzunehmen. Ebenfalls stellt sich das Ergebnis deutlich schärfer als das der Bilinearen Interpolation da [8].

Abbildung 2.8: vergrössertes Bild mit angewendeter Bikubischer Interpolation

2.6 Stitching

Unter Stitching (zu deutsch “annähen“) versteht man das Zusammenfügen von mehreren Einzelbildern zu einem großem Gesamtbild. Dadurch können z.B. große Objekte, die über den Erfassungsbereich einer einzelnen Aufnahme hinausragen, im Ganzen dargestellt werden. Dadurch werden hochauflösende Bilder erzeugt, die deutlich über die Auflösung der Einzelbilder hinaus gehen. Dieses Verfahren findet zumeist Anwendung in

der Erstellung von Foto- und Videopanoramen. Seit langem wird es auch in Videokameras zur Bildstabilisierung eingesetzt [10]. Zudem findet das Verfahren auch Anwendung in der Erstellung von Kartenmaterial und Satellitenaufnahmen. Ein weiteres wichtiges Einsatzgebiet ist die medizinische Bildgebung. So wird es hier zum Erstellen von zusammengesetzten panoramischen Röntgenbildern genutzt [11] und ist unverzichtbar bei der Bildgebung von z.B. der Computertomographie (CT) und der Magnetresonanztomographie (MRT). Bei diesen Verfahren kommt es zu Aufnahmen von Schichtbildern, bei denen mittels Stitching Artefakte (Störungen) entfernt werden. Es können darüber hinaus ganze 3D-Modelle aus den Schichtbildern erstellt werden [12]. Es ist zu beachten, dass es beim Stitching von Nöten ist, dass die Einzelbilder über einen Überlappungsbereich verfügen. D.h. es müssen Objekte bzw. Teile eines Objektes auf beiden Bildern vorhanden sein, um geeignete Bereiche für das Zusammenfügen der Bilder extrahieren zu können. Wie in [13] beschrieben, lässt sich das Stitching Verfahren in drei Hauptschritte unterteilen:

- Kalibrierung (Calibration)
- Registrierung (Registration)
- Blenden (Blending)

Der Schritt der **Kalibrierung** zielt darauf ab möglichst ähnliche Bilder, die zusammengefügt werden sollen, zu produzieren. Unterschiede in den Bildern können durch unterschiedliche Belichtung oder Verzeichnungen, welche von unterschiedlichen Kameraobjektiven erzeugt werden, entstehen. "*Um die 3D-Struktur einer Szene aus den Pixelkoordinaten der Bildpunkte zu rekonstruieren, werden innere und äußere Kameraparameter rekonstruiert*" [13]. Die **Registrierung** ist sozusagen der Kern des Stitching-Verfahrens. Hierbei wird der geometrische Zusammenhang zwischen den Einzelbildern hergestellt, wodurch sie verglichen werden können. Der geometrische Zusammenhang kann z.B. durch die in 2.4.2 erwähnten Affinen Transformationen, planare perspektivische Modelle und durch das Mapping auf nicht planare Oberflächen (z.B.. zylindrisch, sphärisch), beschrieben werden [10]. Somit lassen sich die Einzelbilder, welche aus unterschiedlichen Perspektiven aufgenommen wurden, zueinander ausrichten. Als Bildregistrierung wird der Prozess bezeichnet, in dem zwei oder mehrere Bilder zueinander ausgerichtet werden. Das **Blenden** wird an den "Nahtstellen", wo die Einzelbilder aneinander gefügt werden, durchgeführt. Es sorgt dafür, dass die Übergänge zwischen den Bildern und etwaige Helligkeitsunterschiede möglichst nicht mehr wahrzunehmen sind. Für das Stitching von Bildern gibt es zwei Hauptansätze, den Direkten Ansatz (Direct Technique) und den Merkmals basierten Ansatz (Feature based Technique).

2.6.1 Direkter Ansatz (Direct Technique)

Bei diesem Ansatz werden die Bilder zueinander verschoben bzw. “gewarped“. Bei jedem Verschiebungs- bzw. Warpingschritt werden die Intensitätswerte, der sich überlagernden Pixel, miteinander verglichen. Um nun die bestmögliche Übereinstimmung zu ermitteln, wird nach dem Minimum der *sum of squared differences* (SSD) -Funktion gesucht. Hierfür wird ein Bildausschnitt $I_0(x_i)$ mit den Pixelkoordinaten $x_i = (x_i, y_i)$ definiert. Nun soll bestimmt werden, wo sich dieser Ausschnitt im anderen Bild $I_1(x_i)$ befindet.

$$E_{SSD}(u) = \sum_i [I_1(x_i + u) - I_0(x_i)]^2 = \sum_i e_i^2 \tag{2.11}$$

Wobei in 2.11 $u = (u, v)$ den Versatz bzw. die Verschiebung zwischen den Bildern und e_i den Restfehler (residual error) angibt [10]. Die SSD-Funktion kann auch durch andere Kosten-Funktionen ersetzt werden. Der Vorteil des direkten Ansatzes ist, dass alle verfügbaren Informationen zum Ausrichten der Bilder genutzt werden. Der Nachteil ergibt sich durch den hohen Aufwand der betrieben werden muss. Um eine volle Suche durchzuführen, müssen alle möglichen Ausrichtungen der Bilder zueinander ausprobiert und untersucht werden. Um diesen Aufwand zu verringern, wurden verschiedene Techniken (z.B. coarse-to-fine) entwickelt, welche in [10] näher beschrieben werden.

2.6.2 Merkmals basierter Ansatz (Feature based Technique)

Bei dem Merkmals basiertem Ansatz werden ausgeprägte Merkmale aus den Einzelbildern extrahiert. Nachdem die Merkmale extrahiert sind, werden sie miteinander abgeglichen bzw. zugeordnet, um dann den geometrischen Zusammenhang zwischen den Bildern herstellen zu können. Um ausgeprägte Merkmale (auch “points of interest“ oder “feature points“) aus Bildern zu extrahieren, kommen sogenannte “feature detectors“ (auch “corner detectors“) zum Einsatz. Die meisten Verfahren zum Auffinden von Eckpunkten (corners) basieren auf der Grundlage, dass *“eine Kante in der Regel definiert wird als eine Stelle im Bild, an der der Gradient der Bildfunktion in einer bestimmten Richtung besonders hoch und normal dazu besonders niedrig ist, weist ein Eckpunkt einen starken Gradientwert in mehr als einer Richtung gleichzeitig auf“* [6].Es wurde bereits eine große Menge von Detektoren für Merkmale entwickelt, die verbreitetesten sind der Harris-Detektor (beschrieben in [14]) und der SIFT-Detektor (beschrieben in [15]). Der schon vor längerem entwickelte Harris-Detektor, ist trotz etwaiger Störungen wie Rauschen oder einer Rotation im Bild zuverlässig. Der jüngere SIFT-Detektor ist darüber hinaus auch zuverlässig bei einer vorliegenden Skalierung. Weitere Detektoren werden in [13] diskutiert. Wurden die Merkmale aus den Einzelbildern extrahiert,

müssen sie einander zugeordnet werden. Meist liegt bei der Erstellung eines Panoramas eine Translation zwischen den Koordinaten der gefundenen Merkmale zugrunde, so können in diesem Fall die direkten Umgebungen um die Merkmalskoordinaten herum mit der in 2.11 beschriebenen SSD-Funktion verglichen werden. Anhand des Ergebnisses des Merkmalsvergleichs, kann der geometrische Zusammenhang zwischen den Bildern hergestellt werden. Der Vorteil des Merkmal basierten Ansatzes liegt darin, dass er selbst bei auftretender Skalierung (SIFT) zuverlässig Merkmale extrahiert und somit der geometrische Zusammenhang hergestellt werden kann. Des Weiteren arbeitet dieser Ansatz potentiell schneller, weil nur die extrahierten Features bzw. ihre nähere Umgebung untersucht bzw. abgeglichen werden müssen und somit keine Vollsuche über das gesamte Bild notwendig ist.

2.6.3 Globale Registrierung

Bei Panoramen sollen meist mehr als zwei Bilder zusammengefügt werden. Um dieses umzusetzen, kann man schrittweise vorgehen und jedes weitere Bild zu dem vorherigen, bereits im Panorama registrierten, Bild ausrichten. Jedoch können sich Fehler akkumulieren und dies kann bei 360° Panoramen dazu führen, dass Lücken bzw. Überlagerungen an den beiden Enden des Panoramas auftreten. Diese unerwünschten Effekte lassen sich jedoch durch Dehnung bzw. Stauchung des Panoramas wieder entfernen. Ein anderer Ansatz stellt das “bundle adjustment“ dar. Hierbei werden alle zum Panorama gehörenden Bilder auf einmal betrachtet und in einer 3D Rekonstruktion zusammen gefügt. Dies soll dazu dienen, dass eine global konsistente Ausrichtung gefunden wird und somit die Fehlausrichtung zwischen den einzelnen Bildpaaren minimiert wird. Hierfür müssen initiale Schätzungen für die 3D Lokalisierung der Merkmale und für die Kameraposition vorgenommen werden. Im nächsten Schritt wird beim “bundle adjustment“ ein iterativer Algorithmus ausgeführt, der die Werte für die 3D Rekonstruktion der Szene und der Kamera optimiert. Dies geschieht durch die Minimierung der logarithmierten Likelihood-Funktion über alle Projektionsfehler der Merkmale unter Verwendung der Methode der kleinsten Quadrate (least-squares) [16]. Des Weiteren ist es nötig, die verwendete Kostenfunktion (z.B. 2.11) auf eine Globale Kostenfunktion zu erweitern [10].

2.6.4 Zusammensetzung (Compositing)

Das Zusammensetzen oder Compositing befasst sich damit, wie das fertige Panoramabild dargestellt werden soll. Die einfachste Form der Darstellung ist das sogenannte *flat* Panorama, wobei ein Bild des Panoramas als Referenz ausgewählt wird. Genauer

wird so das Referenzkoordinatensystem ausgewählt und im Anschluss werden alle anderen Bilder des Panoramas in dieses Koordinatensystem überführt bzw. “gewarped“. Problematisch wird diese Form der Darstellung wenn das Panorama einen zu großen sichtbaren Bereich (field of view) darstellt. Ab einem sichtbaren Bereich größer als 90°, bei einer flachen Darstellung, kommt es zu Dehnungen der Pixel nahe der Bildgrenzen. Um dies zu vermeiden, wählt man zur Darstellung von größeren sichtbaren Bereichen eine zylindrische oder sphärische Darstellung. Je nach gewählter Darstellungsform ist anschließend noch eine Transformation der Pixelkoordinaten notwendig, um die Pixel des Eingangsbildes auf die gewünschte Position des Ausgangsbildes abzubilden.

2.6.5 Blenden

Den Abschluss bildet das Blenden der Überlappungsbereiche. Wenn alle Einzelbilder perfekt zueinander ausgerichtet sind und die Bilder sich nicht in ihrer Helligkeit unterscheiden, ist dies ein einfacher Schritt. Ein simpler Ansatz wäre das Alphalevel und die Farbintensitäten der Bilder im Überlappungsbereich zu halbieren, sodass sich durch die Addition der Bilder wieder die allgemeinen Helligkeits- und Farbelevel einstellen. Sollten jedoch nicht perfekte Bedingungen vorliegen, ist es nötig eine etwas aufwändigere Technik anzuwenden. Eine dieser Techniken ist das “Laplace pyramid blending“. Hierfür werden zunächst die beiden Einzelbilder die zusammengefügt werden sollen, in eine Laplace-Pyramide überführt (L_A und L_B). Im Anschluss wird die ausgewählte Region, wo das Blending stattfinden soll, in eine Gauß-Pyramide (G_R) überführt. Nun werden die beiden Laplace-Pyramiden L_A und L_B zu einer neuen kombinierten Pyramide L_K vereinigt, dies geschieht unter Verwendung der Stützstellen von G_R als Gewichte.

$$L_K(i,j) = G_R(i,j) * L_A(i,j) + (1 - G_R(i,j)) * L_B(i,j) \tag{2.12}$$

Abschliessend werden alle Ebenen der neu gewonnenen, kombinierten Pyramide L_K interpoliert und anschliessend aufsummiert. Hierdurch wird als Ergebnis, das aus den beiden Einzelbildern bestehende fertig geblendete Bild erlangt. Neben dem Verfahren des “Laplace pyramid blending“ wurden noch weitere Verfahren entwickelt, wie z.B. das “Gradient domain blending“ oder das “Exposure compensation“-Verfahren.

Kapitel 3

Projektbeschreibung

3.1 OmniCam

Die OmniCam-360 ist ein am Fraunhofer HHI entwickeltes Kamerasystem, welches die Aufnahme von hochauflösenden 360° Panoramen ermöglicht. Das System besteht aus 10 kompakten Micro HD Kameras, welche jeweils mit einer Auflösung von 1920x1080 Bildpunkten aufzeichnen. Sie sind hochkant und kreisförmig um einen Sockel herum installiert. Die Kameras sind auf ein sich darüber befindliches, ebenfalls kreisförmig angeordnetes, Spiegelsystem ausgerichtet.

Abbildung 3.1: OmniCam im Profil

Abbildung 3.2: OmniCam mit Blick auf den Kameraring

Jedes Kamera/Spiegel-Paar erfasst einen Blickwinkel von 36° in der horizontalen Achse und einen Blickwinkel von circa 60° in der vertikalen Achse. Durch diesen Aufbau lässt sich die gesamte Panoramaauflösung von circa 10.000 x 2.000 Bildpunkten erzielen. Das Spiegelsystem ermöglicht es, die virtuellen Brennpunkte der einzelnen Spiegelbilder auf einen gemeinsamen Brennpunkt des gesamten Systems abzubilden (siehe Punkt E in Abbildung 3.3). Ebenfalls sind die Überlappungsbereiche der Kamera/Spiegel-Paare in der Abbildung dargestellt.

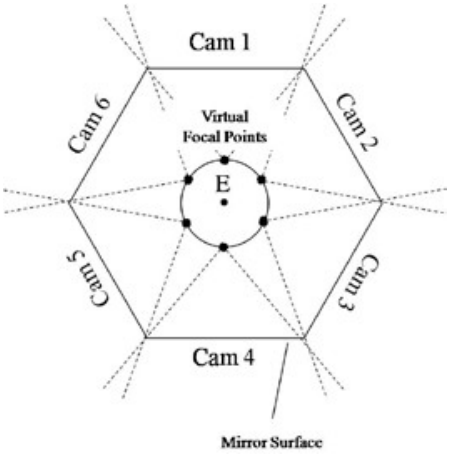


Abbildung 3.3: vereinfachte Darstellung OmniCam mit 6 Kamera/Spiegel-Paaren

Zudem sind die Kameraachsen auf der gleichen horizontalen Schnittebene. Somit lassen sich die Einzelbilder ohne Parallaxenfehler in einem Tiefenbereich von einem Meter bis unendlich zu einem Panorama zusammenfügen.

Neben der OmniCam-360 wurde auch ein 3D fähiges Aufnahmesystem am Fraunhofer HHI entwickelt, die OmniCam-3D. Die 3D-OmniCam verwendet zum jetzigen Zeitpunkt sechs Kameras, wobei jeweils ein Kamerapaar auf ein Spiegelsegment ausgerichtet ist. Es ergibt sich eine Gesamtauflösung von circa 3000 x 2000 Bildpunkten, wobei pro Segment ein Blickwinkel von 30° in der horizontalen Achse und ein Blickwinkel von 60° in der vertikalen Achse erzielt wird. Durch den modularen Aufbau des Systems, lässt sich die OmniCam-3D auf 12 Spiegelsegmente mit insgesamt 24 Kameras erweitern. Somit wäre die Aufnahme von 360° möglich und würde zu einer Gesamtauflösung von circa 12000 x 2000 Bildpunkten führen.

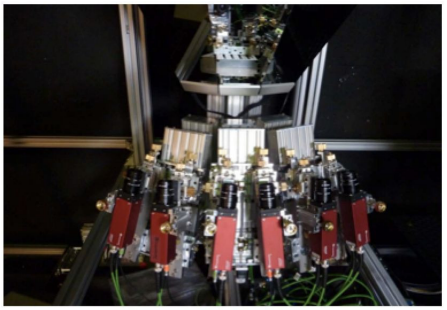


Abbildung 3.4: OmniCam-3D mit 3 Spiegelsegmenten und 6 Kameras

3.2 TiMELab

Das TiMELab (Tomorrow's immersive Media Experience Lab) ist eine Experimentaleinrichtung für immersive Medien des Heinrich-Hertz-Instituts. Es dient als Vorführraum im Bereich der 180°-Videoprojektion und der räumlichen Audiowiedergabe. Zeitgleich dient es natürlich auch der Weiterentwicklung von immersiven [1] Medien, wie z.B. der berührungslosen Interaktion. Das TiMELab befindet sich in einem circa 60 Quadratmeter großen Raum und ist ausgestattet mit einer 3,35 Meter hohen und 8 Meter langen konkav gekrümmten Leinwand. 14 HD-Projektoren sind im Hochkantformat mit einer variablen Aufhängung an der Decke montiert. Die variable Aufhängung erleichtert nicht nur die Wartung der Projektoren, sondern ermöglicht auch eine genaue Justierung.

Abbildung 3.5: TiMELab mit Projektoren und Leinwand

Abbildung 3.6: Projektoren und Spiegelsystem des TiMELab

Für den 2D Betrieb wird nur jeder zweite Projektor verwendet, wobei jeder Projektor eine Auflösung von 1080x1920 Bildpunkten (Hochkantbetrieb) liefert. Somit wäre eine Gesamtauflösung von 7560x1920 Bildpunkten möglich. Allerdings ist es für einen nahtlosen Übergang zwischen den Einzelbildern nötig, einen mit 96 Bildpunkten beanschlagten Überlappungsbereich vorzusehen, wodurch sich die Gesamtauflösung auf 6984x1920 Bildpunkte reduziert. Im Überlappungsbereich muss ein Alphablending der Bilddaten vorgenommen werden, damit es nicht zu einer Aufaddierung der Helligkeits bzw. Farbwerte kommt. Für die 3D Projektion sind alle 14 Projektoren in Betrieb, wobei zwei Projektoren auf die gleiche Fläche der Leinwand projizieren. Durch spezielle 3D Brillen mit Interferenzfiltertechnik, auch Infitec genannt, werden die unterschiedlichen Wellenlängen wieder voneinander getrennt und gelangen somit nur zu dem linken bzw. rechten Auge. Die Projektoren sind mit solchen Infitec-Filtern ausgestattet, welche die drei Grundfarben (RGB) mit unterschiedlichen Wellenlängen multiplexen (wave length division multiplexing). Um das immersive Erlebnis zu erhöhen, sind im TiMELab 120 Lautsprecher auf Ohrenhöhe montiert. Zu dem kommen noch drei Subwoofer,

[1] Immersion beschreibt den Eindruck, dass sich die Wahrnehmung der eigenen Person in der realen Welt vermindert und die Identifikation mit einer Person in der virtuellen Welt vergrößert.

die zusammen mit den Lautsprechern unter Verwendung der Wellenfeldsynthese für eine räumliche Soundwiedergabe sorgen. Die Wellenfeldsythese überlagert verschiedene Schallquellen wodurch ortsstabile Schallpunkte im Raum erzeugt werden. So kann man sich praktisch um ein Geräusch bzw. dessen Quelle herum bewegen. Hierdurch entseht ein eindeutiger Zusammenhang zwischen einer Geräuschquelle und der dazugehörigen Darstellung auf der Leinwand, was den immersiven Eindruck verstärkt.

3.2.1 Kalibrierung

Das TimeLab wurde mit Hilfe des an der Universität Hasselt entwickelten *auto calibration tool* kalibriert, welches in die CineBox integriert ist. Um die Kalibrierung durchzuführen wurde folgender Aufbau verwendet [17]:

- CineBox
- 14 HD-Projektoren der Firma Projectiondesign vom Typ F32 1080 single DLP
- Digitale Spiegelreflexkamera der Firma Canon vom Typ 5D 21MPix
- 8mm Fischaugen-Objektiv der Firma Sigma

Abbildung 3.7: TimeLab Aufnahme durch Fischaugen-Objektiv

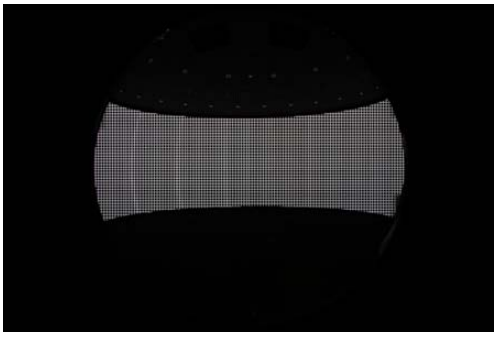


Abbildung 3.8: Aufnahme des projizierten Verifizierungsgitter

Zu Beginn der Kalibrierung werden die Projektoren grob zueinander ausgerichtet, sodass sichergestellt ist, das die benachbarten Projektionen über einen Überlappungsbereich verfügen. Die Kamera wird mittig zur Projektionsfläche positioniert, wobei sie in etwa auf Augenhöhe angebracht ist. Die Kamera wird auf den ISO-Wert 100 eingestellt um ein mögliches Rauschen im Bild niedrig zu halten. Um Farbartefakte zu vermeiden ist die Shutter-Geschwindigkeit an die Anzeigerate der Projektoren angeglichen. Das Fischaugen-Objektiv der Kamera wird auf die Projektionsfläche fokussiert. Nach diesem Einrichtungsvorgang wird die Kamera mit der CineBox verbunden, welche nun die Steuerung des Kalibrierungsvorgangs und somit auch die Steuerung über Kamera

und Projektoren übernimmt. Es werden die Shutter aller Projektoren bis auf einen, dessen Projektion sich gerade in Untersuchung befindet, geschlossen. Nun gibt die CineBox verschiedene Muster (pattern) über den Projektor aus. Bei jedem neuen Muster wird die Aufnahme der Kamera aktiviert. Die gewonnen Fotoaufnahmen werden an die CineBox übertragen und dort gespeichert. Dieser Vorgang wird mit jedem Projektor vollzogen. Wenn alle Aufnahmen getätigt wurden, werden die Bilder der Kamera an das auto calibration tool weiter gereicht. Dieses liefert daraus die entsprechenden Werte für eine Gegenverzerrung, um Bilder unverzerrt auf der Leinwand des TimeLab darzustellen. Des Weiteren werden auch Blendmasken und Gammawert-Tabellen durch das auto calibration tool ermittelt.

3.3 CineBox & CineCard

Bei der CineBox handelt es sich um ein Linux-basiertes(Gentoo) PC-System, welches als Aufnahme bzw. Hostsystem für die CineCard dient. Sie kann bis zu sieben CineCards aufnehmen und betreiben, wodurch die CineBox in der Lage ist, bis zu 14 HD-Projektoren zu bespielen. Bei Bedarf können mehrere CineBoxen kaskadiert werden. Mit der Hilfe der Cinebox werden die CineCards anhand von Skripten gesteuert und konfiguriert. Auf der Festplatte der CineBox sind die für Abspielung gedachten Videoströme gespeichert und werden per PCI-Schnittstelle an die CineCards übertragen [17].

Abbildung 3.9: CineBox bestückt mit 7 CineCards

Abbildung 3.10: CineCard

Die CineCard ist eine PC-Steckkarte mit PCI-Schnittstelle, mit der zwei Videoströme parallel verarbeitet werden können. Dabei werden die kodierten Videoströme gelesen, dekodiert und einer Signalverarbeitung auf einem FPGA, dem “CineController“, unterzogen. Darüber hinaus können auch unkodierte Datenströme über einen DVI-Eingang an die Karte angelegt werden.

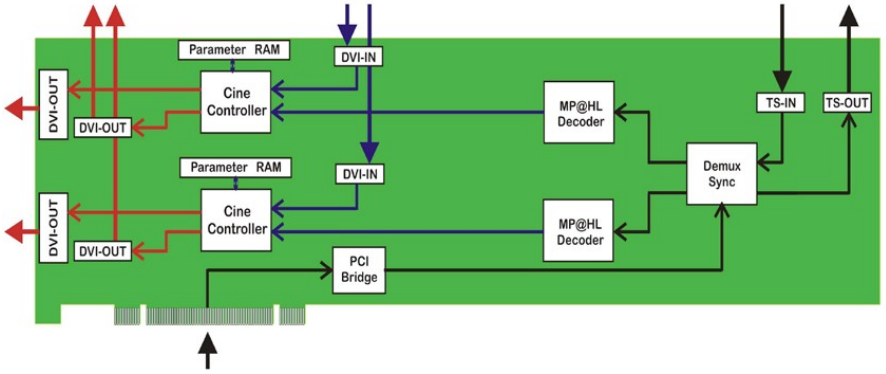


Abbildung 3.11: Übersichtsdarstellung der CineCard

Wie in 3.11 dargestellt ist, befinden sich zwei Datenpfade auf der CineCard, für jeden Video- oder Datenstrom einer. Das Herzstück jedes Datenpfades stellt der FPGA bzw. CineController da. Der verwendete FPGA ist ein *ECP33* der Firma Lattice Semiconductor. Ebenfalls ist pro Datenpfad ein MPEG-2-Dekoder vorhanden, der zur Dekodierung der Videoströme dient. Die dargestellten DVI-Eingänge dienen zum Übertragen von unverschlüsselten Datenströmen, so kann z.B. der Video-Ausgang eines weiteren Rechners direkt mit der Karte verbunden werden und so dessen Inhalt wiedergeben. Jeder der Datenpfade verfügt über zwei DVI-Ausgänge, wobei sich einer der Ausgänge auf einem Slotblech befindet und für die reguläre Ausgabe der Datenströme vorgesehen ist. Der zweite DVI-Ausgang ist als als PIN- bzw. Stiftleiste umgesetzt und dient mehr oder weniger zur Überwachung der ausgegebenen Daten. Zum Speichern von Parameterdaten befindet sich pro Datenpfad jeweils ein DDR-SDRAM-Speichermodul des Typs *Micron MT9VDDT6472H* mit jeweils 576MByte Speicherkapazität. Zu den Parameterdaten die im RAM gespeichert werden, gehören unter anderem die Bilddaten und die in dieser Arbeit erzeugten Adress- und Parameterdaten.

3.3.1 Arbeitsspeicher (RAM)

Die auf der CineCard verwendeten DDR-SDRAM-Speichermodule arbeiten mit der Double-Data-Rate-Technologie (DDR). Die Bezeichnung hat ihren Ursprung daher, das DDR-Speichermodule mit nahezu der doppelten Datenrate im Vergelich zu klassischen SDRAM-Modulen arbeiten. Dies wird dadurch erreicht, das DDR-SDRAM bei steigender, als auch bei fallender Taktflanke Daten überträgt, hingegen werden bei klassischen SDRAM-Modulen die Daten nur bei der aufsteigenden Taktflanke übertragen. Jedes der Speichermodule verfügt über ein sogenanntes Mode-Register, es dient zur Einstellungen bestimmter Parameter und definiert so, wie das Speichermodul arbeitet. Zu diesen Parametern gehört die Burstlänge (Burst Length) und der Bursttyp (Burst Type) [18].

Burtslänge

Lese- und Schreibezugriffe auf dem DDR-SDRAM sind burstorientiert, wobei die Burstlänge einstellbar ist. Die Burstlänge gibt an, auf wie viele Speicherzellen (Spalten) innerhalb einer Zeile für einen gewünschten Lese- oder Schreibbefehl zugegriffen werden kann. Es sind Burstlängen von 2, 4 oder 8 einstellbar. Es wird entsprechend der Burstlänge, die Anzahl der Speicherzellen zu einem Block zusammen gefasst. Jeder Zugriff innerhalb des Burst's findet nur in diesem Block statt. So wird beim Erreichen der Blockgrenzen wieder zum Anfang des Blockes zurück gesprungen.

Bursttyp

Der Bursttyp gibt an, in welcher Reihenfolge bei einem Lese- oder Schreibbefehl, auf die Speicherzellen zugegriffen wird. Es steht ein sequentieller (sequential) oder verschachtelter (interleaved) Modus zur Verfügung. Des Weiteren hat die Startadresse der Spalte ebenfalls Auswirkung auf die Reihenfolge des Zugriffs. So wird beispielsweise ausgehend von einer Spaltenstartadresse von 2, bei einer eingestellten Burstlänge von 8 und einem sequentiellen Bursttyp, auf die Speicherzellen (Spalten) in der Reihenfolge 2-3-4-5-6-7-0-1 zugegriffen. Im Gegensatz dazu wird bei gleicher Startadresse und gleicher Burstlänge unter Verwendung des verschachtelten Bursttyp's, auf die Speicherzellen in der Reihenfolge 2-3-0-1-6-7-4-5 zugegriffen.

3.4 CineCard v2

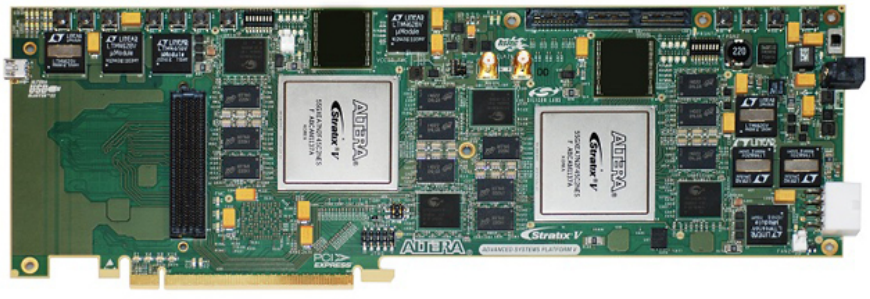


Abbildung 3.12: Stratix V Advanced Systems Development Board

Derzeit wird eine neue Version der CineCard, welche von hier an mit CineCard v2 bezeichnet wird, entwickelt. Um sie mit den entsprechenden Parameterdaten für die Darstellung von Bild- und Videomaterial im TiMELab zu versorgen, wurde das in dieser Arbeit beschriebene Programm entwickelt. Die Entwicklung der CineCard v2 basiert auf dem „Stratix V Advanced Systems Development Board". Dabei handelt es sich um eine PC-Steckkarte welche von der Firma Altera entwickelt und hergestellt

wurde. Wie der Name vermuten lässt, ist die Karte speziell für die Entwicklung von Prototypen konzeptioniert worden. Die Karte ist zur Integration in einem PC mit einer PCI-Express-Schnittstelle ausgestattet. Sie verfügt über zwei FPGA's vom Typ „Stratix 5SGXEA7N2F45C2N“, wobei jeder FPGA über zwei DDR3-SDRAM Module mit einer 64 Bit Schnittstelle und zwei weiteren DDR3-SDRAM Modulen mit einer 32 Bit Schnittstelle verfügt [19]. Dadurch erhöht sich die verfügbare Bandbreite erheblich. So wird es möglich sein, Videomaterial mit einer Bildwiederholungsrate von 60Hz darzustellen, was gegenüber der Bildwiederholungsrate der CineCard v1 von 30 Hz, eine signifikante Steigerung darstellt. Durch die „high-speed mezzanine card“ (HSMC) Steckverbinder ist sie modular und skalierbar. Über die HSMC-Verbindungungen kann die Karte zum Beispiel mit SDI- oder HDMI-Modulen erweitert werden. Dies ermöglicht es Videomaterial über diese Module direkt an den FPGA zur Verarbeitung zu übergeben. Das Entwicklungsziel sieht vor, dass auf einem FPGA vier Videoströme mit einer Bildwiederholungsrate von 60Hz verarbeitet werden. Somit könnte beispielsweise das TiME Lab, welches derzeit mit sieben CineCards v1 ausgestattet ist, mit nur zwei CineCards v2 betrieben werden um die 14 HD-Projektoren mit Daten zu versorgen. Im Unterschied zu dem auf der CineCard v1 verwendetem Speicher, siehe 3.3.1, werden beim DDR3-Speicher nur Burstlängen von 4 und 8 unterstützt, wobei bei einer gewählten Burstlänge von 4 es zur Anwendung des sogenannten “Burst Chop“ kommt. Bei einem “Burst Chop“ wird eigentlich eine Burstlänge von 8 verwendetet jedoch wird die zweite Hälfte des Burst's terminiert und nicht überrtragen. Dieser Vorgang ist in [20] genauer beschrieben.

Kapitel 4

Hintergründe zu Design-Entscheidungen

4.1 Geometrische Transformation

Um eine optisch ansprechende Darstellung des Bildmaterials im TiMELab zu erreichen, wird das durch die OmniCam produzierte Filmmaterial vorverarbeitet. Das heißt, es wird die Verzeichnung, ausgelöst durch die Objektive, vorab entfernt. So wird eine unverzerrte Darstellung des Bildmaterials, zum Beispiel auf einem PC erzielt. Um eine unverzerrte Darstellung auf der zylindrischen Leinwand im TimeLab zu erlangen, werden die durch die in 3.2.1 beschriebenen Kalibrierung erlangten Parameter genutzt. Sie dienen als Ausgangspunkt für das Programm, welches während der Durchführung dieser Arbeit entwickelt wurde. Um der in 2.4.1 beschriebenen Problematik mit entstehenden Bildlücken zu begegnen, wird die Transformation ausgehend von den Bildpunktkoordinaten des Ausgangsbildes durchgeführt. Es wird also eine Rückwärtstransformation angewandt. Für die richtige Abbildung muss das entsprechende Eingangsbild bei der Transformation schon vollständig vorhanden sein. So wird immer ein komplettes Eingangsbild im Arbeitsspeicher der CineCard gespeichert. Die entsprechenden Operationen für die geometrische Transformation der Bildpunktkoordinaten und die anschließende Interpolation, findet auf dem FPGA („CineController“) der CineCard statt. Die dafür nötige Vorverarbeitung der Daten, sowie die Ermittlung der richtigen Interpolationsparameter, wird durch das während dieser Arbeit entwickelte Programm umgesetzt. Eine komplette Verarbeitung der Daten auf der CineCard ist durch die begrenzte Hardware nicht möglich, dies würde zu einer allgemeinen Verringerung der maximalen Taktfrequenz, mit welcher der CineController betrieben wird führen, was wiederum in einem verringerten Datendurchsatz beim Warping der Bilddaten resultieren würde. Zudem

würden damit die begrenzten Hardwareressourcen auf der CineCard, wie DSP-Blöcke (Digital Signal Processing), in denen Vorgänge, wie das Umsetzen der Multiplikationen, für die Berechnung der Transformationsfunktion fehlen. Somit wären diese nicht mehr zur Durchführung der Interpolation verfügbar. Darüber hinaus, würde sich bei einer kompletten Verarbeitung auf der CineCard der Entwicklungsaufwand für das FPGA-Design deutlich erhöhen. Aus diesen Gründen wird der Ansatz der Vorverarbeitung der Daten in einem separaten Programm gewählt.

4.2 Interpolation

Während der geometrischen Transformation wird ein geeignetes Interpolationsverfahren benötigt. Wie in 2.5 beschrieben, bietet die Bikubische Interpolation eine hohe Qualität in der Darstellung und ist deshalb das angestrebte Ziel. Um die Bikubische Interpolation umzusetzen werden 16 Bildpunkte, genauer deren Koordinaten, im Eingangsbild benötigt. Im einfachsten Fall werden die benötigten Bildpunktkoordinaten bei Bedarf direkt aus dem Arbeitsspeicher der CineCard gelesen. Dies würde allerdings auf Grund der hohen Latenz eines Lesebefehls und der insgesamt nötigen Bandbreite nicht funktionieren. Um diesem Problem zu begegnen werden zwei Strategien angewendet.

1. Bei der CineCard v1 werden die Parameter für die Bildpunkte (Koordinaten, Alpha-Werte und Blacklevel-Werte) von insgesamt drei Bildpunkten in eine der 72 Bit großen Speicherzellen des Arbeitsspeichers abgelegt. Da durch die Verwendung einer Burstlänge von 2 immer zwei Speicherzellen zugleich ausgelesen werden, können pro Takt des CineController's die Daten von sechs Bildpunkten anstatt von nur einem verarbeitet werden. Dieses Vorgehen verringert zwar deutlich die Anzahl der nötigen Operationen auf dem Arbeitsspeicher, jedoch gestaltet sich dadurch die Interpolation aufwendiger. Es können nun nicht mehr einzelne Daten für die Bildpunkte adressiert werden, sondern nur noch in Blöcken von sechs Bildpunkten. Bei der CineCard v2 werden die Daten von einem Bildpunkt in eine der 64 Bit großen Speicherzellen des Arbeitsspeichers abgelegt. Durch die verwendete Burstlänge von 8 sind jedoch auch hier die Bildpunkte nicht mehr einzeln adressierbar, sondern nur noch in Gruppen von acht Bildpunkten. So muss das zu entwickelnde Programm, abhängig von der Version der CineCard, entsprechende Gruppen von Pixeln (Pixelblöcke) erstellen, um für diese die Adressierung zu ermöglichen.

2. Die Pufferung der Bildpunktdaten in einem Cache. Hierfür steht ein Zeilenspeicher, von insgesamt vier Zeilen, im Embedded Block Ram (EBR) des FPGA's

zur Verfügung. Dies deckt sich damit, dass für die Bikubische Interpolation vier Zeilen um die Koordinate des zu interpolierenden Bildpunktes benötigt werden. Das Puffern der Bildzeilen bringt den Vorteil mit sich, das die Inhalte des Pufferspeichers, ohne einen erneuten Zugriff auf den Arbeitsspeicher, wiederverwendet werden können. Die mögliche Wiederverwendung bei benachbarten transformierten Koordinaten ist zwar nicht zwangsläufig gegeben, hat jedoch eine hohe Wahrscheinlichkeit. So ergibt sich ein geringerer Bandbreitenbedarf, weil die Daten dieser Bildpunkte nicht wieder aus dem Arbeitsspeicher geladen werden müssen.

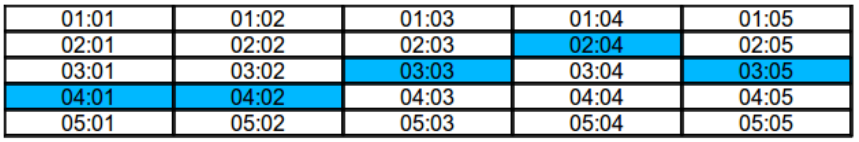


Abbildung 4.1: Beispielhafter Verlauf Bildausgangszeile

In 4.1 ist durch die farbliche Markierung ein beispielhafter Verlauf der benötigten Bildpunktdaten für die Hauptgewichtung einer Bildausgangszeile im Raster des Eingangsbildes dargestellt. Wie der Darstellung zu entnehmen ist, werden hier drei verschiedene Zeilen des Eingangsbildes als Quelle für das Transformationsergebnis einer Ausgangszeile benötigt. Bei einer Rotation von 45°, was für die Darstellung im TimeLab als „worst case“ angesehen wird, würden sechs verschiedene Zeilen des Eingangsbildes als Hauptgewichtung für das Ausgangsbild benötigt. Durch das in 4.2 erwähnte Speichern von vier Bildzeilen in einem Pufferspeicher, müssen nicht alle benötigten Zeilen neu aus dem Arbeitsspeicher abgerufen werden, sondern liegen einige, wenn nicht sogar alle, nötigen Bildzeilen im Pufferspeicher vor. Zudem wird die Interpolation angepasst, wenn nicht alle nötigen Bildpunkte im Pufferspeicher vorhanden sind. So müsste die Bikubische Interpolation bzw. die Gewichtung selbiger so angepasst werden, dass aus der ursprünglichen 4×4 Interpolation teilweise z.B. eine 3×4 Interpolation wird. Darüber hinaus sollte, falls nötig, auf eine Bilineare Interpolation oder auf eine Nächster Nachbar Interpolation zurückgegriffen werden. Die fehlenden Zeilen für die Interpolation werden durch die begrenzte Bandbreite zum Arbeitsspeicher nicht extra nachgeladen. Um während der Bestimmung der Interpolationskoeffizienten immer einen Überblick über die im Zeilenspeicher enthaltenen Daten zu haben, wird für das zu entwickelnde Programm der Ansatz gewählt eine Abbildung des Zeilenspeichers in der Software zu schaffen.

Die CineCard führt die entsprechende Interpolation nacheinander, getrennt in X-Richtung und Y-Richtung durch, wobei die vertikale Interpolation (Y-Interpolation) zuerst durchgeführt wird. Würde die Interpolation in X-Richtung zuerst durchgeführt werden, würde

das Problem auftreten, dass benötigte Pixel nicht vorhanden wären. Dieses Problem tritt an den Grenzen zwischen zwei Pixelblöcken auf.

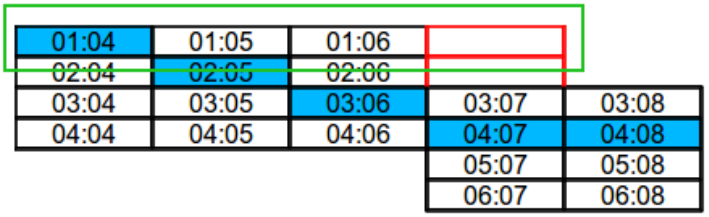


Abbildung 4.2: X- vor Y-Interpolation

Wie in Abbildung 4.2 dargestellt, kann es an den Grenzen zwischen zwei Pixelblöcken zu dem Problem von nicht vorhanden Pixeln kommen. Beispielsweise würde der grüne Rahmen die benötigten Pixel für eine Bikubische Interpolation markieren. Wie zu sehen ist, fehlt der mit rot markierte Pixel hierfür. Unabhängig vom beschriebenen Problem, wäre es nötig um die Interpolation in X-Richtung als erstes durchzuführen, zwei 4 x 6 Pixelstrukturen bereit zuhalten. Diese Pixelstrukturen könnten zwar geladen werden, jedoch belastet dies unnötig die verfügbare Bandbreite zum Arbeitsspeicher. Wird hingegen die Interpolation in Y-Richtung als erstes durchgeführt, tritt dieses Problem nicht auf (siehe Abbildung 4.3), weil durch den Zeilenspeicher immer vier Pixel pro Spalte für die Interpolation verfügbar sind. Zudem ist es nicht nötig zwei 4 x 6 Pixelstrukturen, sonder nur eine 4 x 6 Pixelstruktur bereit zu halten. Dementsprechend wird auch bei dem zu entwickelnden Programm die vertikale Interpolation als erstes durchgeführt.

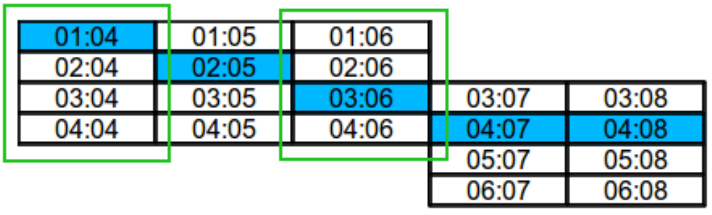


Abbildung 4.3: Y- vor X-Interpolation

Für die Interpolation werden immer aus allen vier Zeilenspeicheren gleichzeitig ein Pixelblock ausgelesen. Dabei muss sichergestellt werden, das die Pixelblöcke immer in der richtigen Reihenfolge im Zeilenspeicher abgelegt und verfügbar sind. Würden die Pixelblöcke in einer falschen Reihenfolge abgelegt sein, würde das die Implementierung der Hardware nicht registrieren und die Interpolation der Pixel fortsetzen. Das würde dazu führen, das die Gewichtungsfaktoren bzw. Koeffizienten für die Interpolation mit den

falschen Bildpunkten verarbeitet werden würde und somit eine fehlerhafte Interpolation durchgeführt werden würde. Durch die in 4.3.1 beschriebene unabhängige Beschreibung der Spalten des Zeilenspeichers, befinden sich in den verschiedenen Spalten Pixelblöcke mit den Daten für unterschiedliche Ausgangszeilen. Durch den geführten Index, welcher den „obersten“ Zeilenspeicher der aktuellen Spalte markiert, kann sichergestellt werden dass die Pixelblöcke in der korrekten Reihenfolge an das Modul zur Interpolation auf der CineCard weitergereicht werden.

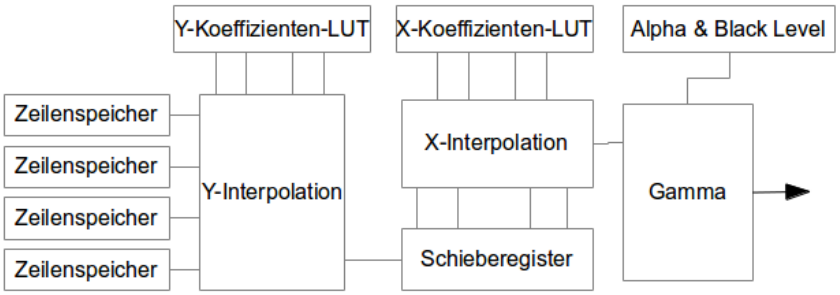


Abbildung 4.4: Datenpfad der Interpolation

Die Abbildung 4.4 stellt den implementierten Datenpfad auf der CineCard für Interpolation da. Ausgehend vom Zeilenspeicher werden die Pixeldaten an das Modul zur Interpolation in Y-Richtung weitergereicht. Durch den geführten Index, welcher die Reihenfolge der Zeilenspeicher für die aktuelle Spaltenposition führt, wird die vertikale Reihenfolge der Pixel sichergestellt. Die aus den Parameterdaten gewonnenen Y-Koeffizienten werden in die Y-Koeffizienten-LUT geladen. Anhand der Koeffizienten führt das Modul zur Y-Interpolation eine Multiplikationen mit den Pixeldaten durch und gewinnt so das vertikale Interpolationsergebnis. Das Ergebnis der vertikalen Interpolation wird an ein Schieberegister übergeben. Aus dem Schieberegister werden die Interpolationsergebnisse anhand ihrer Reihenfolge gelesen und an das Modul zur horizontalen Interpoaltion (X-Interpolation) weitergereicht. Aus den erstellten Parameterdaten werden die entsprechenden Koeffizienten zur horizontalen Interpolation ausgelesen und in die X-Koeffizienten-LUT geschrieben. Nun wird im Modul zur horizontalen Interpolation eine Multiplikationen von den gegebenen Gewichtungsfaktoren und den Ergebnissen der vertikalen Interpolation durchgeführt. Die so erhaltenden Daten, stellt das Ergebnis der vertikal und horizontal interpolierten Pixelwerte aus dem Eingangsbild da. Im Anschluss daran befindet sich das Modul zur Gammakorrektur. Hier werden aus den erstellten Parameterdaten die entsprechenden Alpha- und Black Level-Werte ausgelesen und anhand derer eine Gammakorrektur der Bilddaten durchgeführt.

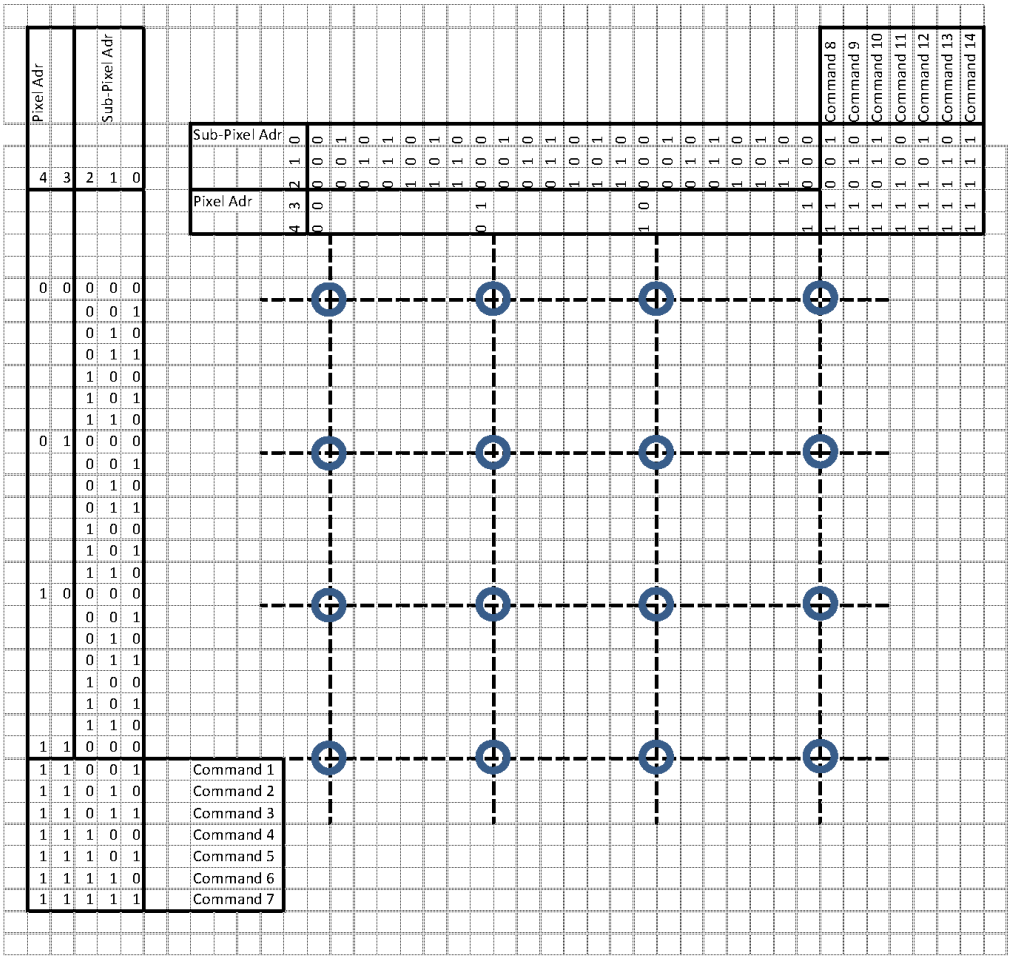


Abbildung 4.5: Speichermatrix Interpolation

Die Abbildung 4.5 stellt die Speichermatrix, welche bei der Interpolation genutzt wird da. Die mit blauen Kreisen markierten Positionen repräsentieren die Bildpunktkoordinaten des Eingangsbildes. Die Werte der Koeffizienten für die vertikale bzw. horizontale Interpolation werden in Pixeladresse und Sub-Pixeladresse unterteilt. Die obersten beiden Bit der Werte des Koeffizienten stellen die Pixeladresse da und die untersten drei Bit stellen die Sub-Pixeladresse da. Für die Nächste Nachbar Interpolation wird nur die Pixeladresse, das heißt die oberen beiden Bit des Koeffizienten benötigt. Für die Bilineare oder Bikubische Interpolation werden zu dem noch die Sub-Pixeladressen bzw. die untersten drei Bit des Koeffizienten benötigt. Liegt zum Beispiel für die vertikale Interpolation die Zeilenkoordinate, welche auf das Eingangsbild verweist, zwischen der ersten und zweiten Pixelposition und es soll eine Nächster Nachbar Interpolation durchgeführt werden, so muss der Y-Koeffizient so erstellt werden, das je nachdem an welcher Zeile die zu interpolierende Zeilenkoordinate näher liegt, entweder auf die Pixeladresse „00“ oder

„01“ verwiesen wird. Hierfür müssen die Positionen der Bildpunktdaten innerhalb des Zeilenspeichers bekannt sein und als Pixeladresse genutzt werden. Für beispielsweise die Bilineare Interpolation, muss die Nachkommastelle der Zeilenkoordinate entsprechend auf die verfügbaren Sub-Pixeladressen „001“ bis „110“ aufgeteilt werden, um somit die Gewichtung der bei der Interpolation einfließenden Pixel anzugeben. Für die vertikale und horizontal Interpolation existieren jeweils sieben Bit-Kombinationen, welche als Interpolationskommandos genutzt werden können (siehe 4.1).

Tabelle 4.1: Interpolationskommandos

Binär	**Dezimal**	**Name**	**Beschreibung**
11001	25	BLACK_VALUE	Ausgabe eines schwarzen Pixels
11010	26	NO_VALUE	kein Interpolationsparameter verfügbar, benutze vorherigen Parameter
11011	27	STOP_X_INTERPOLATION	führe X-Interpolation aus und stoppe Y-Interpolation
11100	28	SHIFT_X_INTERPOLATION	stoppe X-Interpolation und führe Y-Interpolation aus
11110	30	NO_ACTION	führe keine Operation durch
11111	31	END_OF_LINE	signalisiert das Ende der Zeile

4.3 Speicherstruktur

4.3.1 Zeilenspeicher

Auf der CineCard sind vier Pufferspeicher zur Verarbeitung der Interpolation vorgesehen, sie werden auch als Zeilenspeicher bezeichnet. In ihnen werden die zur Interpolation benötigten Pixelblöcke gespeichert. Anhand der Adressdaten werden hierfür die entsprechenden Pixelblöcke aus dem Arbeitsspeicher gelesen. In welche Spalte die Pixelblöcke geschrieben werden, wird Anhand des „RowOffset“, welcher in den Adressdaten enthalten ist, bestimmt. Ein neu geladener Pixelblock stellt den untersten der vier benötigten Pixelblöcke da und wird in den untersten der vier Zeilenspeicher geschrieben. Der Speicherinhalt an entsprechender Spaltenpostion dieses Zeilenspeichers, wird in den darüber liegen Zeilenspeicher verschoben, der Speicherinhalt des obersten Zeilenspeichers wird verworfen.

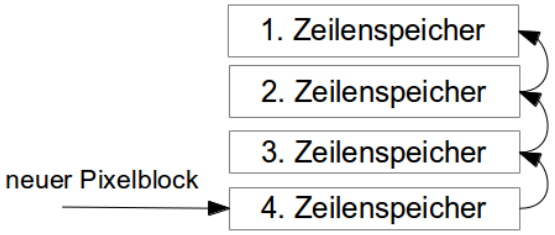


Abbildung 4.6: Modellhafte Darstellung des Zeilenspeichers

Dabei handelt es sich jedoch um eine Modellvorstellung. In der Hardware werden die Speicherinhalte nicht einfach verschoben, vielmehr wird das Verschieben durch eine Um- bzw. Neuadressierung erreicht, wobei der Inhalt des „obersten“ Zeilenspeichers, bzw. der mit den ältesten Daten an entsprechender Spaltenposition, überschrieben wird. Dies macht es nötig einen Index für Zeilenspeicher zu führen, sodass anhand des Index immer ermittelt werden kann, welcher Zeilenspeicher für den Inhalt an entsprechender Spaltenposition den „obersten“ Zeilenspeicher darstellt.

4.3.2 Arbeitsspeicher

Zeile	**Bank 0**	**Bank 0**	**Bank 1**	**Bank 1**	**Bank 2**	**Bank3**
	Spalte 0	*Spalte 1024*	*Spalte 0*	*Spalte 1024*	*Spalte 0*	*Spalte 0*
Zeile 0	Bildzeile 0	Bildzeile 1	Bildzeile 2	Bildzeile 3	Parameter	Adressen
Zeile 1	Bildzeile 4	Bildzeile 5	Bildzeile 6	Bildzeile 7	Parameter	Adressen
Zeile 2	Bildzeile 8	Bildzeile 9	Bildzeile 10	Bildzeile 11	Parameter	Adressen
...	...	...	...	...	...	...

Abbildung 4.7: Speicherstruktur der CineCard v1

Die Speicherstruktur der CineCard v1 ist in Abbildung 4.7 dargestellt. Alle Speicherbereiche beginnen mit der ersten Speicherzelle, der jeweiligen Zeile. In den ersten beiden Speicherbänken sind Bilddaten des darzustellenden Eingangsbild gespeichert, wobei immer zwei Zeilen des darzustellenden Bildes in einer Speicherbank pro Speicherzeile gespeichert werden. Es sind pro Speicherzeile 2048 Speicherzellen mit einer Größe von 72 Bit verfügbar. Bei einer Bildauflösung von 1920 x 1080 Bildpunkten sind 1920 Bildpunkte pro Zeile zur Darstellung nötig. Dadurch das immer drei Bildpunkte zusammen in einer Speicherzelle untergebracht sind (siehe 4.2), ergibt sich bei einer HD-Auflösung eine Notwendigkeit von 640 Speicherzellen / Spalten zum Speichern einer Bildzeile. Im Speicher sind für jede Bildzeile 1024 Speicherzellen reserviert und bieten somit jeder Bildzeile genügend Speicherplatz. In Bank 2 werden die Parameterdaten und in Bank 3 die Adressdaten gespeichert.

Im Falle der CineCard v2 werden vier Speicherbänke zum Speichern der Bilddaten verwendet, wobei jede Speicherzeile über 1024 Speicherzellen / Spalten verfügt. Jede dieser Speicherzellen hat eine Größe von 64 Bit. Pro Speicherzelle sind die Daten für

Zeile	Bank 0	Bank 1	Bank 2	Bank 3	Bank 4
Zeile 0	Bildzeile 0.1	Bildzeile 1.1	Bildzeile 2.1	Bildzeile 3.1	Parameter
Zeile 1	Bildzeile 0.2	Bildzeile 1.2	Bildzeile 2.2	Bildzeile 3.2	Parameter
Zeile 2	Bildzeile 4.1	Bildzeile 5.1	Bildzeile 6.1	Bildzeile 7.1	Parameter
...	...	...	...	...	...
Zeile 2944	Bildzeile X.1	Bildzeile X+1.1	Bildzeile X+2.1	Bildzeile X+3.1	Adressen
Zeile 2945	Bildzeile X.2	Bildzeile X+1.2	Bildzeile X+2.2	Bildzeile X+3.2	Adressen
...	...	...	...	...	...

Abbildung 4.8: Speicherstruktur der CineCard v2

einen Bildpunkt abgelegt. Somit sind zum Speichern einer Bildzeile bei einer angenommen Auflösung von 1920 x 1080 zwei Zeilen im Speicher notwendig. Wie der Abbildung 4.8 zu entnehmen ist, wird beispielsweise für die Bildzeile 0 die erste Hälfte der Bildzeile in Speicherbank 0 und Speicherzeile 0 abgelegt. Um nicht auf unterschiedliche Speicherbänke zum Abrufen einer gesamten Bildzeile zugreifen zu müssen, wird die zweite Hälfte der Bildzeile 0 in die Speicherzeile 1 der Speicherbank 0 abgelegt. Die sich ergebenen Lücken zwischen dem Ende der zweiten Bildzeile und dem Ende der Speicherzeile bleiben vorerst ungenutzt bzw. bieten Raum für etwaige größere Auflösungen des zu speichernden Bildes. Die Parameterdaten und Adressdaten sind zusammen in der Speicherbank 4 untergebracht. Um genügend Speicherplatz für die Daten sicherzustellen, wird ein Zeilenoffset eingeführt. So werden die Parameterdaten ab Speicherzeile 0 fortlaufend in die Speicherbank 4 abgelegt. Hingegen werden die Adressdaten beginnend ab Speicherzeile 2944 der Speicherbank 4 fortlaufend gespeichert. Dementsprechend muss bei der Erstellung der Pixeladressen die Version der CineCard und die damit verbundene Speicherstruktur beachtet werden.

4.4 Burst Count

Der FPGA (CineController) der CineCard führt unter Verwendung des Parameters „Burst Count“ einen Befehl n-mal (n = 1...32 bzw 1...64) aus. Bei einem Lese- oder Schreibbefehl werden allerdings mit jeder Befehlsausführung nicht immer die gleichen Speicherzellen ausgelesen bzw. beschrieben. Es werden vielmehr die aufeinander folgenden Speicherzellen (Spalten) innerhalb der aktiven Zeile des Speichers angesprochen. Dies geschieht durch die Verwendung eines internen Adresszählers [21], welcher mit der Startadresse der entsprechende Speicherzelle initialisiert wird. Nach der Durchführung des Befehls, wird vor der nochmaligen Ausführung der Adresszähler entsprechend erhöht und somit der Befehl auf die folgenden Speicherzellen ausgeführt. Wird bei dieser sequentiellen Abarbeitung der Befehle das Ende der Speicherzeile erreicht, wird zu deren Anfang zurückgekehrt und die Abarbeitung der noch ausstehenden Befehle fortgesetzt.

Abbildung 4.9: Darstellung von Lesebfehl mit Burstlänge 2 und Burst Count 4

So werden z.B. , wie in Abbildung 4.9 dargestellt, bei einem Lesebefehl unter Verwendung eines Burst Count's von 4 und einer Burstlänge von 2 insgesamt 4*2 = 8 Speicherzellen ausgelesen. Durch die Verwendung eines Bust Count's größer als 1, werden die Daten direkt hintereinander ohne Lücken verarbeitet. Diese Vorgehensweise ist effizienter als nur mit einzelnen Befehlen (Burst Count = 1) zu arbeiten. Dementsprechend soll das zu entwickelnde Programm durch Verwendung des Burst Count's den Auslesevorgang des Arbeitsspeichers optimieren. Da diese Technik nur angewendet werden kann, wenn sich die zu lesenden Speicherzellen in der selben Speicherbank und Speicherzeile befinden, ist dies bei der Erstellung zu prüfen.

4.5 Stitching

Um bei Aufnahmen mit der OmniCam eine bessere Vorstellung des aktuell aufgenommenen Panoramas zu erlangen, soll eine Vorschau aller zehn aufgenommenen Kamerabilder ermöglicht werden. Dafür sollen die zehn Einzelbilder aneinander gefügt bzw. „gestitcht" werden. Um das erhaltende Gesamtpanorama auf einem Display anzeigen zu können, ist es notwendig die Einzelbilder entsprechend zu skalieren. Die hierfür benötigten bzw. verfügbaren Daten sind für jedes Kamerabild in drei separaten Dateien abgelegt. Zum Einen sind die Transformationskoordinaten in einer Binärdatei mit der Endung „.pt2D" gespeichert. Die aufgenommenen Bilder der OmniCam liegen im Landscape-Format vor, wo hingegen die Transformationskoordinaten im Portrait-Format vorliegen. Dies ist begründet in der Tatsache dass die Kameras der OmniCam hochkant angebracht sind (3.1) und das Bildmaterial hochkant dargestellt werden soll. In einer Klartextdatei mit der Endung „.par" sind die Bildparameter gespeichert. Dazu gehören:

- Bildgröße
- Bild-Offset

Die Werte für die Bildgröße geben die Anzahl der Bildzeilen und Bildspalten an. Die Werte des Bild-Offset geben den Offset der Bildzeilen und Bildspalten an. Dieser Offset

beschreibt den für jedes Einzelbild gegebenen Abstand, zu einem angenommenen Koordinatenursprung. Ähnlich wie in 2.2 beschrieben, befindet sich der Koordinatenursprung in der linken, oberen Ecke des Gesamtpanoramas. In einer Binärdatei mit der Endung „.bin“ liegen die Alpha-Werte des jeweiligen Bildes vor.

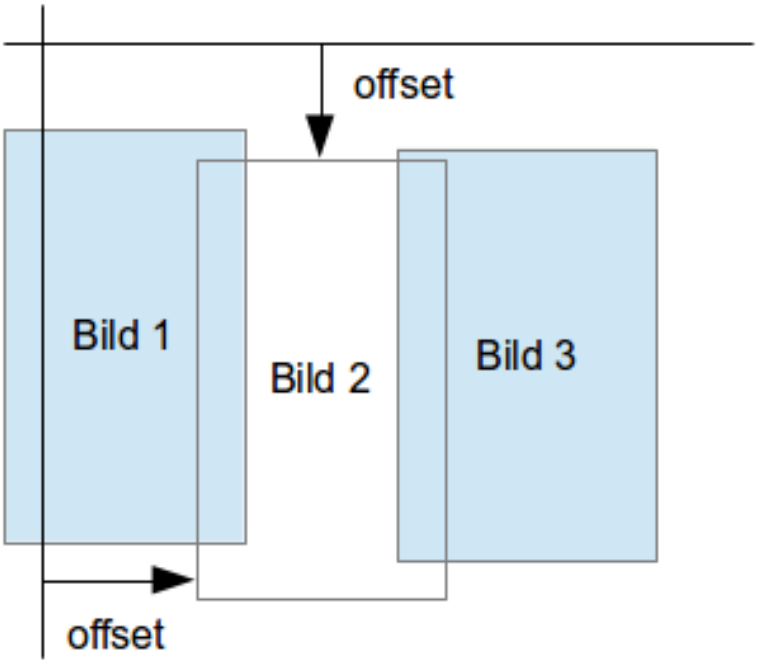


Abbildung 4.10: Modell Stitching

Die Abbildung 4.10 soll die Erstellung des Gesamtpanoramas veranschaulichen, wobei nur ein Ausschnitt des Gesamtpanoramas dargestellt ist. Das Gesamtpanorama beinhaltet, wie bereits erwähnt, zehn Einzelbilder, wobei sich das zehnte Einzelbild zur Schließung des Kreises, wieder mit Bild 1 überschneidet. Wie zu entnehmen ist, wird durch den Offset und die Größe eines Einzelbildes dessen Position und die von ihm abgedeckte Fläche innerhalb des Gesamtpanoramas definiert. Die Größe der Einzelbilder (Zeilen- und Spaltenanzahl) unterscheiden sich zueinander. Durch die Hochkant-Darstellung werden Bildzeilen zu Hochkant-Spalten und Bildspalten zu Hochkant-Zeilen. Diese Bezeichnungen werden im Folgenden zur klareren Beschreibung verwendet. Neben den Bildgrößen, unterscheiden sich ebenfalls die angegebenen Offset-Werte. Da das ausgegebene Panorama eine einheitliche obere und untere Grenze besitzen soll, muss bei der Implementierung diese Grenze ermittelt bzw. festgelegt werden. So wäre es sinnvoll, zunächst die Bildparameter für alle Einzelbilder auszuwerten. Dadurch könnte festgestellt werden, welches Einzelbild den größten Abstand zur X-Achse des angenommenen Koordinatensystems besitzt. Dieser Wert wird als obere Grenze festgelegt. Ebenfalls könnte durch den Offset zur X-Achse und der angegebenen Anzahl von Hochkant-Zeilen ermittelt

werden, welches Einzelbild mit dem geringsten Abstand zur X-Achse endet, dieses wird dann als untere Grenze festgelegt. Somit ist sichergestellt, dass über alle Einzelbilder hinweg, zwischen den ermittelten Grenzen, Bilddaten zur Darstellung vorhanden sind. Durch die angegebene Anzahl von Hochkant-Spalten und dem zugehörigen Offset, der den Abstand zur Y-Achse definiert, ergibt sich der Bereich in dem sich die Einzelbilder überlappen. Damit auf der CineCard das entsprechende „Blending“ durchgeführt werden kann, müssen diese Überlappungsbereiche markiert bzw. signalisiert werden. Um, wie bereits erwähnt, eine Darstellung des Gesamtpanoramas auf einem Display zu ermöglichen, müssen die Einzelbilder entsprechend skaliert werden, sodass sich beispielsweise für das Gesamtpanorama eine HD-Auflösung von 1920 x 1080 ergibt. Des Weiteren ist eine Anpassung der Alpha-Daten nötig. So sollten die Alpha-Daten entsprechend der bestimmten Grenzen und der durchgeführten Skalierung angepasst werden.

Kapitel 5

Implementierung

5.1 Verwendete Software / Bibliotheken

5.1.1 C++

Um die gestellten Aufgaben umzusetzen, wurde als Programmiersprache C++ gewählt. C++ gehört zu den weit verbreitetesten Programmiersprachen und ist somit den Meisten geläufig. So sollte eine spätere Weiterführung des entwickelten Programms ohne größere Probleme möglich sein. Zudem bietet C++ den Vorteil, dass auf den meisten Systemen entsprechende C++-Compiler bereits vorhanden sind, sodass sich die Verwendung auf einem neuen System mit einem relativ geringen Aufwand darstellt. Des Weiteren sind bereits sehr viele nützliche Bibliotheken für C++ verfügbar, welche die Implementierung deutlich vereinfachen.

5.1.2 Eclipse

Eclipse ist eine freie Entwicklungsumgebung zum Erstellen von Softwarelösungen. Ursprünglich wurde Eclipse für die Entwicklung in der Programmiersprache Java geschrieben. Durch das Eclipse CDT Projekt (C/C++ Development Tool) ist es aber mittlerweile möglich, unter Verwendung der Programmiersprachen C und C++ innerhalb von Eclipse Software zu entwickeln. Eclipse bietet neben dem üblichen Editor mit Syntaxhervorhebung, die Möglichkeit verwendete Bibliotheken komfortabel einzubinden und den geschriebenen Code per Knopfdruck zu kompilieren und auszuführen. Des Weiteren verfügt Eclipse über einen umfangreichen Debug-Modus, welcher es erlaubt, schrittweise die Ausführung des geschriebenen Quellcodes zu verfolgen. So können Speicherinhalte von Variablen zum aktiven Zeitpunkt angezeigt und der Aufruf von Funktionen genau

verfolgt werden. Da das Zielsystem (die CineBox), auf dem das entwickelte Programm laufen soll, auf Linux basiert, war es sinnvoll die Entwicklung bereits unter Linux durchzuführen. So fielen andere Entwicklungsumgebungen wie z.B. Visual Studio aus dem Kreis der verfügbaren Entwicklungsumgebungen heraus, was letztendlich zur Wahl von Eclipse beitrug.

5.1.3 OpenCV

Bei OpenCV handelt es sich um eine freie Programmbibliothek unter den Bedingungen der BSD-Lizenz, welche mit Hinblick auf Bildverarbeitung und maschinelles Sehen entwickelt wurde. Sie ist für die Programmiersprachen C und C++ geschrieben. Durch OpenCV können Bilder auf einfachste Weise eingelesen und die Daten direkt verarbeitet werden. Bilder werden in einem „Mat“-Objekt abgelegt und liegen dort in Matrixform vor. Diese Form der Speicherung der Bilddaten bzw. der Zugriff auf selbige, ist sehr intuitiv und bereits aus der Programmiersprache Matlab bekannt. Ein weiterer Vorteil besteht darin, dass sich die Mat-Klasse von OpenCV selbstständig um die Allokierung und Freigabe von Speicher kümmert. Dadurch ist der Nutzer nicht mehr gezwungen dieses manuell vorzunehmen, dennoch besteht die Möglichkeit dies zu tun.

5.1.4 Boost

Boost ist eine freie C++-Bibliothek, welche unter der „Boost Software License“ veröffentlicht wurde. Diese Lizenz ähnelt sehr stark der BSD-Lizenz. Boost ist eine sehr mächtige Bibliothek, die aus einer Vielzahl von portablen Unterbibliotheken besteht. Die Funktionalität von Boost umfasst Bibliotheken zur Verarbeitung von Zeichenketten, mathematische Bibliotheken, Bibliotheken zum Parsen von Daten und Bibliotheken für die Bildverarbeitung, um nur einige zu nennen. Im Fraunhofer HHI wird durch die Verwendung von Boost-Build und der Boost-Bibliothek angestrebt, entwickelte Software systemunabhängig zu gestalten.

5.1.5 Matlab

Matlab ist ein Programm zur Lösung von mathematischen Problemen, entwickelt von der Firma „The MathWorks“. Es ist primär für numerische Berechnungen mithilfe von Matrizen ausgelegt. Bei der Durchführung dieser Arbeit wurde Matlab zur Überprüfung der ermittelten Daten genutzt. Hierfür wurden in C++ Textdateien erstellt, welche mit Matlab eingelesen wurden. Durch die Matrizen-Darstellung von Matlab, wurde eine manuelle Überprüfung der Richtigkeit von Matrizen, die unter C++ erstellt wurden, stark

vereinfacht. Darüber hinaus wurde Matlab zur Entwicklung von Funktionsprototypen genutzt, welche dann anschliessend in C++ umgesetzt wurden.

5.2 Umsetzung des Programms

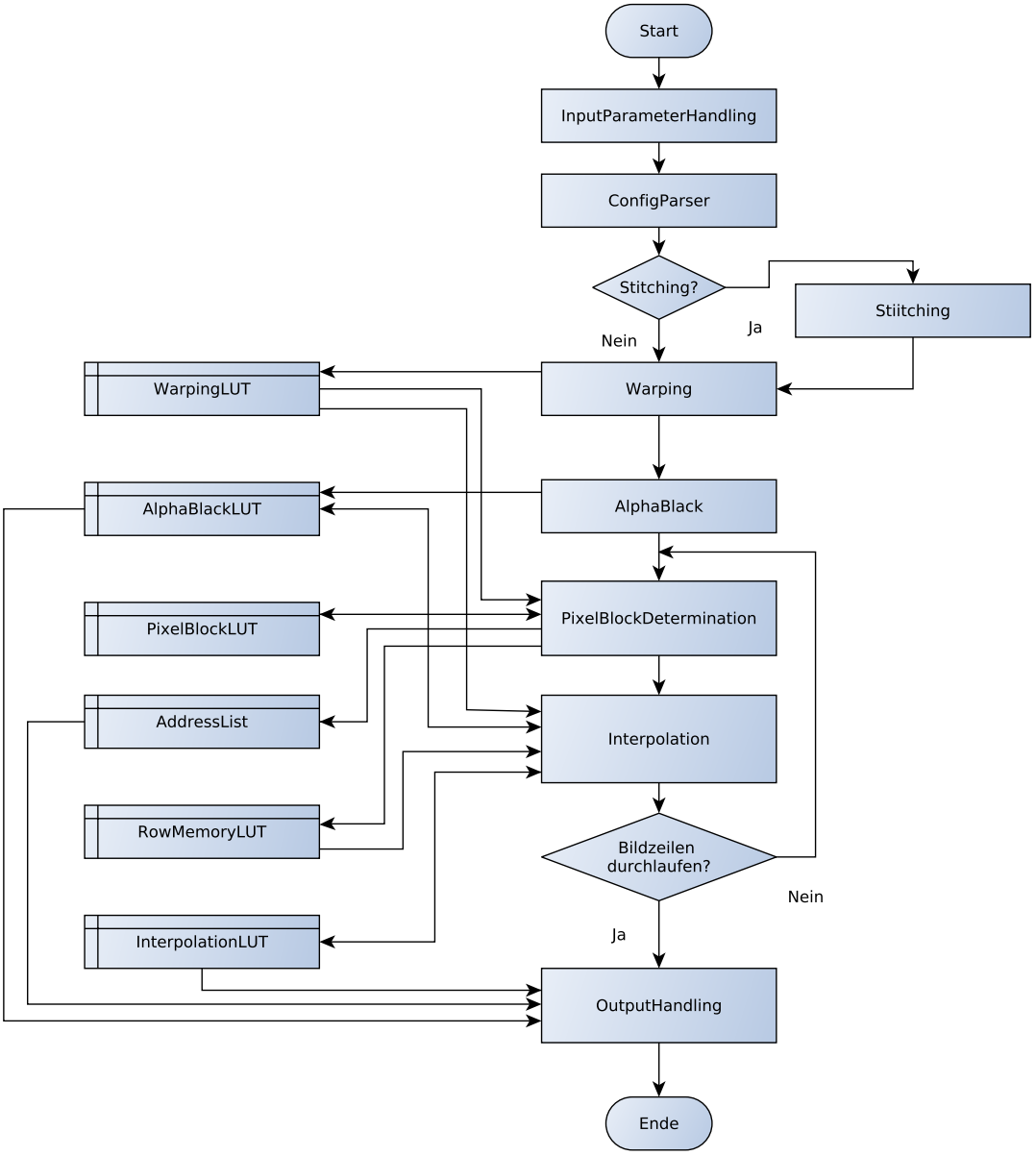


Abbildung 5.1: Modulübersicht

5.2.1 InputParameterHandling

Die Klasse „InputParameterHandling“ ist die erste Anlaufstelle des Programms und ist für die Analyse der vom Nutzer getätigten Eingabe zuständig. Eine der gestellten

Anforderung an das Programm ist es, die Initialisierung des Programms, über die Eingabe von einzelnen Parametern beim Programmaufruf, über die Konsole zu ermöglichen. Eine weitere geforderte Form der Initialisierung besteht darin, eine gegebene Konfigurationsdatei auszuwerten. Die Funktion „analyzeInputArguments“ unterscheidet anhand der Anzahl von Parametern bei der Eingabe, wie zu verfahren ist. Wird beim Programmaufruf kein Parameter mit übergeben, so wird die Funktion „useDefaultConfig“ aufgerufen. Anhand des übergebenen String, welcher den Pfad zur standardmäßig zu verwendenden Konfigurationsdatei enthält, wird zunächst einmal geprüft, ob die angegebene Datei überhaupt vorhanden ist. Der Pfad, welcher auf die standardmäßig zu verwendende Konfigurationsdatei verweist, wird in der „CineboxWarping.cpp“-Datei, in der sich die Main-Funktion befindet, gesetzt. Dadurch soll für eine eventuelle spätere Anpassung des Pfades, ein unnötiges Suchen nach der Stelle, an der dieser Pfad gesetzt wird, vermieden werden. Die Prüfung, ob die durch den Pfad angegebene Datei vorhanden ist, wird durch die Funktion „fileExists“ durchgeführt. Ihr wird der String mit dem sich darin enthaltenden Pfad übergeben. Die Funktion versucht mittels der Input-File-Stream-Klasse, welche zu den Standardbibliotheken von C++ gehört, die durch den Pfad definierte Datei zu öffnen. Mit Hilfe der „isgood“-Funktion wird überprüft, ob sich die Datei ohne auftretende Probleme öffnen lässt. Ist dies der Fall, so liefert die „fileExist“-Funktion ein *true* zurück und es wird ein Text mit dem Pfad der verwendeten Konfigurationsdatei ausgegeben. Andernfalls wird ein *false* zurückgegeben und das Programm mit einem Fehler und der dazu gehörigen Ausgabe beendet. Bei der Rückkehr in die „analyzeInputArguments“-Funktion, wird die boolesche Variable „singleArgumentsInit“ auf *false* gesetzt, um den „configParser“ zu signalisieren, dass eine Konfigurationsdatei zum Initialisieren des Programms verfügbar ist. Wird ein einzelner Parameter beim Aufruf des Programms übergeben, wird geprüft, ob es sich bei dem Parameter um den Text „`--help`“ handelt. Trifft dies zu, wird die Funktion „useSingleArguments“ aufgerufen und die beim Start des Programms angegebenen Parameter mit übergeben. Die Funktion verwendet die Boost Bibliothek „program_options“. Mit Hilfe der Bibliothek, lassen sich auf komfortable Weise, Eingabeparameter definieren. Es kann eine Liste mit den erlaubten bzw. unterstützen Argumenten definiert werden, wobei jedem Argument eine Beschreibung für die Ausgabe zugewiesen wird. Des Weiteren lässt sich eine Ausgabe im Falle eines „`--help`“-Aufrufes definieren, welche die Benutzung des Programms erläutert. So wird im Falle des „`--help`“-Aufrufs die Benutzung des Programms, mit der Reihenfolge der erwarteten Parameter erläutert und in der Konsole ausgegeben. Die Ausgabe wird vervollständigt, indem jedem Parameter eine Beschreibung folgt. Nach der Ausgabe der Hilfebeschreibung wird das Programm beendet. Werden beim Programmstart alle benötigten Parameter einzeln übergeben, wird ebenfalls die bereits erwähnte „useSingleArguments“-Funktion aufgerufen. Es werden alle Argumente anhand ihrer

Reihenfolge den entsprechenden Variablen zugewiesen. Sollte der Datentyp eines Arguments nicht mit dem erwarteten Datentyp übereinstimmen, kommt es zu einem Fehler. Bei der Übergabe der aus den Argumenten extrahierten Werte an die Mitgliedervariablen der Klasse, werden diese zur Bestätigung für den Nutzer auf der Konsole ausgegeben. Beim Rückkehren in die „analyzeInputArguments“-Funktion wird die boolesche Variable „singleArgumentsInit“ auf true gesetzt, um den „configParser“ zu signalisieren, das einzeln angegebene Argumente zum Initialisieren des Programms verwendet werden.

5.2.2 configParser

Die Klasse „configParser“ ist dafür zuständig, eine gegebene Konfigurationsdatei zu parsen und so aus ihr die angegebenen Werte für die einzelnen Argumente zu gewinnen. Der „configParser“ wird nachdem die Eingabe vom „InputParameterHandling“ analysiert wurde aufgerufen. Anhand der booleschen Variable „singleArgumentsInit“ kann er bestimmen, ob eine Konfigurationsdatei eingelesen werden muss oder ob die Argumente bereits beim Programmstart einzeln übergeben wurden. Hat die boolesche Variable den Wert *true*, so greift der „configParser“ mit Hilfe der entsprechenden „get“-Funktionen auf die Mitgliedervariablen des „InputParameterHandling“ zu und weist deren Werte, seinen eigenen Mitgliedervariablen zu. Im Anschluss daran wird die Methode „matchWarpingFile2WarpingMethod“ aufgerufen. Innerhalb der Methode, wird aus dem übergebenen Pfad zur Parameterdatei (enthält Koordinaten welche auf das Eingangsbild verweisen) die Dateiendung der Datei extrahiert. Die Dateiendung wird mit der gewählten bzw. übergebenen Verzerrungsform (distortionForm) abgeglichen. Dieser Abgleich basiert auf der Grundlage, dass für jede Verzerrungsform andere Dateiformate, zur Speicherung der Koordinaten für die Transformation, genutzt werden. So sind beispielsweise die Transformationskoordinaten für das TiMELab farbkodiert in einer PNG-Datei (Portable Network Graphics) gespeichert. Sollten die gewählte Verzerrungsform und das Dateiformat der Koordinaten nicht zu einander passen, wird eine entsprechende Fehlermeldung in der Konsole ausgegeben und das Programm mit einem Fehler beendet. Besitzt die boolesche Variable „singleArgumentsInit“ hingegen den Wert *false*, wird versucht die gegebene Konfigurationsdatei auszuwerten. Als Konfigurationsdatei, wird eine Datei im XML-Format erwartet bzw. genutzt. Zum Parsen der XML-Datei werden die Boost Bibliotheken „archive“ und „serialization“ genutzt. Mit Hilfe der „archive“-Bibliothek wird die XML-Datei geöffnet und eingelesen. Unter Verwendung der „serilization“-Bibliothek werden Name-Wert-Paare gebildet. Anhand der XML-Elementnamen wird der entsprechende zugehörige Wert ausgelesen und den Mitgliedervariablen des „configParser“ zugewiesen. Nachdem die Konfigurationsdatei eingelesen wurde, wird mit Hilfe der „matchWarpingFile2WarpingMethod“-Funktion überprüft,

ob die angegebene Verzerrungsform zum Dateiformat der Parameterdatei passt. Mit der Funktion „save_config“ lassen sich die aktuell gewählten Werte für die Eingabeparameter in eine XML-Konfigurationsdatei speichern, diese wird jedoch im Moment nicht weiter genutzt. Nachdem die Eingabeparameter auf die eine oder andere Weise verfügbar sind, setzt der „configParser“ ausgewählte Variablen innerhalb der „Global“-Klasse, um diese global im ganzen Programm verfügbar zu machen. Die durch das „InputParameterHandling“ oder den „configParser“ erfassten Eingabeparameter, sollen im Folgenden kurz erläutert werden.

pathParamFile

Gibt den Pfad zur Parameterdatei an, in der die Koordinaten für die Transformation gespeichert sind.

pathBlackLevelImage

Gibt den Pfad zur Datei, welche die Daten für das Black Level enthält an.

pathAlphaImage

Gibt den Pfad zur Datei, welche die Daten für das Alphablending enthält an.

pathOutputData

Gibt den Pfad an, wo die erstellten Dateien mit Adress- und Parameterdaten sowie die für die Simulation erstellten Dateien gespeichert werden sollen.

pathStitchingData

Gibt den Pfad zum Stammverzeichnis, der zum Stitching zugehörigen Daten an.

interpolationForm

Gibt an, welche Art der Interpolation durchgeführt werden soll. Es wird noch entschieden in wie weit dieser Parameter sinnvoll ist und ob er nicht entfernt wird. Im Moment wird eine Bilineare Interpolation durchgeführt und wenn notwendig auf eine Nächste Nachbar Interpolation zurückgegriffen. Wenn die Bikubische Interpolation implementiert wurde, ist es sinvoll bei Bedarf auf die Bilineare Interpolation zurück zufallen. Ein Erzwingen einer geringeren Interpolation von vornherein, erscheint zu diesem Zeitpunkt nicht mehr sinnvoll.

distortionForm

Gibt an, für welche Anwendung die Daten gedacht sind bzw. wie mit den entsprechenden Eingabedateien umgegangen wird. Mögliche Einstellungen sind hier „timelab“, „rtt“ oder „stitching“.

alphaForm

Gibt an, ob für das Alphablending eine entsprechende Datei oder ob Standard-Werte verwendet werden sollen.

memoryName

Enthält den Namen des verwendeten Speichers auf der CineCard, wird zum Erstellen der Simulationsdaten benötigt.

exchangeAlphaBlackLevel

Ist derzeit nicht implementiert bzw. wird nicht genutzt. Anhand des Parameters soll entschieden werden, ob die ermittelten Daten für die Interpolation für eine Wiederverwendung gespeichert werden. Bei einer Wiederverwendung sollen die Alpha- und Black-Level-Werte durch neu übergebene Werte ersetzt werden, um einen kompletten Durchlauf des Programms zu vermeiden.

simulationActive

Gibt an, ob eine Simulation durchgeführt und entsprechend Daten erstellt werden sollen.

simulationRows

Im Falle einer Simulation, können hier die zu bearbeitenden Bildzeilen angegeben werden, sodass beispielsweise nur ein halbes Bild simuliert werden kann.

imgFormatRows

Gibt die Anzahl der Zeilen des zu transformierenden Eingangsbildes an.

imgFormatCols

Gibt die Anzahl der Spalten des zu transformierenden Eingangsbildes an.

pixelBlockSize

Definiert die Größe der verwendeten Pixel-Blöcke, sodass bei einer Änderung der verwendeten Burstlänge im Speicher keine Quellcodeanpassungen notwendig sind.

cineCardVersion

Gibt an, für welche Version der CineCard die entsprechenden Daten erstellt werden sollen.

memStartAddress

Definiert die Speicherstartadresse, welche für die Erstellung der Simulationsdaten benötigt wird.

5.2.3 Warping / WarpingLUT

Nachdem die vom Nutzer übergebenen Argumente eingelesen und verfügbar gemacht worden sind, wird nun die erstellte Warping- und die zugehörige WarpingLUT-Klasse aufgerufen. Während die Warping-Klasse für die nötigen Operationen zuständig ist, dient

die WarpingLUT-Klasse als Speicherobjekt der ermittelten Daten. Zu Beginn wird die „init“-Funktion des erstellten Warping-Objektes aufgerufen. Ihr werden die vom Nutzer definierten Parameter pathParamFile, distortionForm, imgFormatRow und ImageFormatCols übergeben. Anhand des Parameters distortionForm wird ermittelt, wie die Transformationskoordinaten gespeichert worden sind und übergeben werden. Entspricht der Wert von distortionForm der Zeichenkette „timelab“, so wird davon ausgegangen, das Koordinaten für die Verzerrung farbkodiert in einer PNG-Datei abgelegt sind. Anhand des gegebenen Pfades wird versucht die Datei bzw. das Bild mit Hilfe der OpenCV-Bibliothek, unter Verwendung des Befehls „imread“, einzulesen. Das übergebene Bildformat wird genutzt, um die entsprechende Mitgliedervariable, wobei es sich um ein OpenCV-Mat-Objekt handelt, zu initialisieren. Gelingt das Einlesen des Bildes, wird es in dem Mat-Objekt abgelegt. Im Falle von „rtt“ als Verzerrrungsform handelt es sich bei der Datei, in der die Transformationskoordinaten enthalten sind, um eine Textdatei, wobei die Zeilenkoordinate und die Spaltenkoordinate, welche zusammen eine Pixelposition im Eingangsbild beschreiben, hintereinander durch ein Komma getrennt abgelegt sind. Darauf folgt, wieder durch ein Komma getrennt, das nächste Koordinatenpaar. Die Abbildung 5.2 soll dies verdeutlichen.

Zeile Pixel1,	SpaltePixel1,	Zeile Pixel2,	Spalte Pixel2,	...
Zeile PixelX,	Spalte PixelX,	Zeile PixelX+1,	Spalte PixelX+1,	...
...	...	...	...	...

Abbildung 5.2: Datenstruktur der rtt-Eingabedaten

Das übergebene Bildformat wird genutzt, um ein zweidimensionales Mat-Objekt zu intialiseren. Mit Hilfe eines C++-Filestreams wird nun auf die Textdatei zugegriffen und diese Zeile für Zeile eingelesen. Die einzelnen Zeilen werden in einem String abgelegt, in welchem mit Hilfe der „find_first_of“-Funktion die Positionen der Kommazeichen bestimmt werden. Die so extrahierten Zeilen- und Spaltenkoordinaten werden in das Mat-Objekt abgelegt. Die Zeilenkoordinaten werden hierbei in die erste, und die Spaltenkoordinaten in die zweite Dimension der Matrix geschrieben. So liegen die Koordinatenpaare, in unterschiedlichen Dimensionen, an den entsprechenden Pixelpostionen des Ausgangsbildes, wodurch eine spätere Zuordnung erleichtert wird. Um die entsprechenden Pixelpositionen zu ermitteln, wird jeweils ein Zähler für die Zeile und ein Zähler für die Spalte geführt. Für das Zusammenführen (stitching), wird das Einlesen und Verarbeiten der Daten separat in der dafür vorgesehenen Stiching-Klasse vorgenommen, auf welche später noch eingegangen wird. Somit wird in diesem Fall keine Operation innerhalb der Warping-Klasse vorgenommen. Entspricht die gewählte Verzerrungsform der Zeichenkette „noWarping“, so soll das Eingangsbild 1:1 ausgegeben werden, somit sind auch keine Transformationskoordinaten aus einer entsprechenden Datei einzulesen. Ist eine

unvorhergesehen Zeichenkette angegeben, wird das Programm mit einer Fehlermeldung beendet. Nachdem die Datei mit den Koordinaten eingelesen wurde, muss nun eine weitere Verarbeitung bzw. Aufbereitung der Daten vorgenommen werden. Die WarpingLUT, in welcher die letztendlich ermittelten Transformationskoordinaten gespeichert werden, ist durch einen Vektor bestehend aus Struktureinträgen verwirklicht. Die Struktureinträge verfügen über einen Zeilen- und Spaltenwert. So beinhaltet jeder Struktureintrag die Koordinaten für einen Pixel, welcher auf das Eingangsbild verweist. Ist kein Warping erwünscht, so wird eine 1:1 Zuweisung anhand von Zählern für Zeilen und Spalten vorgenommen. Der jeweilige Zählerstand wird den Einträgen der WarpingLUT zugewiesen. Soll ein „timelab"-Warping durchgeführt werden, so müssen die farbkodierten Bildpunktkoordinaten zunächst dekodiert werden.

Hierfür wird auf das zuvor beim Einlesen erstellte Mat-Objekt zugegriffen. Es handelt sich dabei um eine vierdimensionale Matrix. An dieser Stelle ist zu erwähnen, das OpenCV beim Einlesen von Bildern, beispielsweise unter Verwendung des Befehls „imread", diese nicht in der gewohnten RGB-Reihenfolge, sondern in einer BGR-Reihenfolge ablegt. Pixelweise werden alle vier Dimension der Matrix ausgelesen und zwischengespeichert. Zum Dekodieren der Farbkodierung wurde folgende Vorgehensweise, von den Erstellern des in 3.2.1 beschriebenen *auto calibration tool*, verfügbar gemacht.

$$\begin{aligned} u_{Int} &= r * 256 + g \\ v_{Int} &= b * 256 + a \\ u_{norm} &= (u_{Int}\text{–}16384)/32767 \\ v_{norm} &= (v_{Int}\text{–}16384)/32767 \\ u &= u_{norm} * imageCols\text{–}0.5 \\ v &= v_{norm} * imageRows\text{–}0.5 \end{aligned} \tag{5.1}$$

Die Gleichung 5.1 beschreibt die nötige Vorgehensweise zum Dekodieren der Daten, wobei *r,g,b* und *a* für die einzelnen Kanäle bzw. Dimensionen des Bildes stehen. Das erhaltene Ergebnis für u entspricht der Spaltenkoordinate und v der Zeilenkoordinate des Bildpunktes im Eingangsbildes. Im Anschluss werden die erhaltenen Koordinaten überprüft, ob sie sich innerhalb des Bildrasters befinden. Liegen sie außerhalb des Bildrasters, wird dem WarpingLUT-Objekt an entsprechender Postion der Wert -1 (BLACK_VALUE_SRC) zugewiesen. Andernfalls werden die ermittelte Zeilen- und Spaltenkoordinate dem entsprechenden Struktureintrag, innerhalb der WarpingLUT, zugewiesen. Bei gewählten „rtt"-Warping werden pixelweise die Koordinaten aus dem zuvor erwähnten zweidimensionalen Mat-Objekt ausgelesen. Liegen die Werte der Koordinaten

zwischen 0 und -0.5 so werden sie glatt auf Null gerundet. Im Anschluss wird wiederum, wie schon beim „timelab“-Warping, geprüft, ob sich die Koordinaten im Raster des Bildes befinden. Liegen sie außerhalb, wird dem WarpingLUT-Objekt an entsprechender Position der Wert -1 (BLACK_VALUE_SRC) zugewiesen. Andernfalls werden die Zeilen- und Spaltenkoordinate dem WarpingLUT-Objekt übergeben. Somit sind die Koordinaten, welche auf das Eingangsbild verweisen, nun in der WarpingLUT verfügbar.

5.2.4 AlphaBlack / AlphaBlackLUT

Wie schon bei der Warping- bzw. WarpingLUT-Klasse ist die AlphaBlack-Klasse für die notwendigen Operationen zuständig, während die AlphaBlackLUT-Klasse zur Speicherung der ermittelten Werte vorgesehen ist. Die AlphaBlackLUT besitzt als Mitgliedervariable ein cv::Mat-Objekt. Es wird eine dreidimensionale Matrix anhand der gegebenen Bildauflösung initialisiert. Die drei Dimensionen begründen sich darauf, dass für jeden Farbkanal des auszugebenden Bildes ein entsprechender Alpha- bzw. BlackLevel-Wert verfügbar ist. Zur Zeit werden allerdings nur monochrome Daten verwendet, d.h. die Werte in allen drei Dimensionen bzw. Kanälen einer Pixelposition sind gleich. Zum Einlesen der Daten wird in der AlphaBlack-Klasse die Funktion „init“ verwendet. Die Funktion liest, mit Hilfe der „imread“-Funktion, nacheinander die gegebenen Bilder für Alpha und BlackLevel ein und speichert diese in Mitgliedervariablen vom Typ cv::mat. Sollte das Einlesen der Bilder nicht gelingen, wird eine Fehlermeldung ausgegeben. An dieser Stelle ist zu erwähnen, dass das AlphaBlackLUT-Objekt zur Speicherung der Alpha- sowie der BlackLevel-Werte mit der doppelten Zeilenanzahl des angegebenen Bildformates initialisiert wird. Dies geschieht, um einen Index für die Spalten mit abspeichern zu können. Der Grund hierfür war es, ein debuggen bzw. die Fehlersuche zu vereinfachen. Dementsprechend wird beim Verarbeiten der Alpha- und BlackLevel-Werte, der genannte Index in das AlphaBlackLUT-Objekt geschrieben. Im Anschluss wird die Bittiefe des übergebenen Alphabildes überprüft. Dies geschieht, weil zum Teil Alphabilder mit unterschiedlichen Bittiefen verwendet werden. So werden bei einer zu geringen Bittiefe die Alphawerte durch Bit-Shiften und durch das damit verbundene Anhängen von Nullen auf die gewünschte Bitlänge von 16 Bit gebracht. Um bei der späteren Ausgabe der Daten eine einheitliche Verarbeitung zu gewährleisten, wird dieser Sonderfall an dieser Stelle bereits abgefangen. Folgend darauf, werden die Alpha- und BlackLevel-Werte dem jeweiligen AlphaBlackLUT-Objekt zugewiesen. Sollten für die Verarbeitung kein Alpha- und BlackLevel-Bild vorhanden sein, werden den entsprechenden LUT-Objekten Standardwerte zugewiesen. So wird für jede Pixelposition der BlackLevel-LUT der Wert 0 und der Alpha-LUT der Wert „4096“, was einem gesetzten 13. Bit entspricht, zugewiesen. Darüber hinaus verfügt die AlphaBlack-Klasse über eine Funktion, welche die erstellten

LUT-Objekte pro Zeile um eine Spalte erweitert und eine Markierung einfügt. Diese Markierung besagt, dass das Ende der Zeile erreicht wurde und ist zur Vereinfachung der Verarbeitung in der Hardware (genauer dem FPGA) bestimmt.

5.2.5 PixelblockDetermination / PixelblockDataLUT

Wie aus den vorherigen Modul- bzw. Klassenbeschreibungen bekannt ist, ist die Klasse „PixelblockDetermination“ für die vorzunehmenden Operationen zuständig. Die Klasse „PixelblockDataLUT“ wird zur Speicherung der ermittelten Daten genutzt. Mit der Funktion „getImageOrientation“ wird die Bildorientierung ermittelt. Genauer wird ermittelt, ob die eingetragenen Koordinaten des übergebenen WarpingLUT-Objektes aufsteigend oder absteigend vorliegen. Hierfür werden die Maximalwerte für Zeilen- und Spaltenkoordinate und deren Position im Ausgangsbild ermittelt. Zudem wird das Auftreten von Schwarzwerten (-1, BLACK_VALUE_SRC) innerhalb einer Zeile und innerhalb einer Spalte gezählt. Durch das Zählen der Schwarzwerte kann nun das geringste Vorkommen selbiger und deren Position bestimmt werden. Das Ermitteln der Schwarzwerte ist zur Behandlung eines Sonderfalls vorgesehen. Wäre z.B. eine Hälfte des Bildes (obere oder untere, linke oder rechte) schwarz, so könnte man nicht anhand der Position die auftretenden Maximalwerte der Koordinaten, die Bildorientierung ermitteln. Wäre die gesamte linke Bildhälfte schwarz, so würde der Maximalwert für die Spaltenkoordinaten definitiv in der rechten Bildhälfte liegen und somit wäre dennoch nicht zu klären, ob die Spaltenkoordinaten in der nicht schwarzen Bildhälfte aufsteigend oder absteigend vorliegen. Durch das Ermitteln des geringsten Vorkommen der Schwarzwerte, lässt sich ein eventueller Schwarzbereich vom Gesamtbild subtrahieren. Das Ergebnis ist der nicht schwarze Bereich des Bildes. Somit lässt sich dann auch ermitteln, in welcher Hälfte, des nicht schwarzen Bereiches des Bildes, die Maximalwerte der Koordinaten auftreten. Treten diese in der ersten Hälfte auf, so sind die Zeilenkoordinaten bzw. Spaltenkoordinaten absteigend und umgekehrt. Diese Erkenntnis wird im späteren Verlauf des Programms, bei der Erstellung der Pixelblöcke und der Ermittlung der Interpolationsdaten, noch verwendet.

Die folgenden Funktionen der hier beschriebenen Klasse, sowie die folgenden Funktionen der Klasse „Interpolation“ werden innerhalb einer „for“-Schleife, welche über alle Zeilen der übergebenen Bildpunktkoordinaten iteriert aufgerufen. Somit ist die folgende Bearbeitung der Daten ein zeilenweises Vorgehen.

Die Funktion „doBlockCount“ betrachtet nacheinander alle Spalten des WarpingLUT-Objektes der aktuellen Zeile. Es werden die Zeilen- und Spaltenkoordinate temporär gesichert. Anhand der Spaltenkoordinate wird nun die Pixelblockadresse ermittelt. Wie

in 3.3.1 und in 4.2 beschrieben, wird für die CineCard v1 eine Burstlänge von 2 verwendet, zudem werden die Daten von drei Pixeln innerhalb einer Speicherzelle abgelegt. So werden mit jedem Lese- oder Schreibbefehl die Daten von sechs Pixeln erfasst bzw. adressiert. Somit ergibt sich für die CineCard v1 eine Pixelblockgröße von sechs. Bei einer angenommenen Auflösung von 1920 x 1080 werden somit insgesamt 320 Pixelblöcke benötigt, um eine komplette Zeile zu speichern. Um die Pixelblockadresse zu bestimmen, wird die Spaltenkoordinate zunächst abgerundet und dann durch die Pixelblockgröße dividiert. Das Ergebnis wird wiederum abgerundet und nun mit der Pixelblockgröße multipliziert. Folgendes Beispiel soll diesen Vorgang verdeutlichen.

$$Pixelbockadresse = floor(floor(Spaltenkoordinate)/Pixelblockgroesse) * Pixelblockgroesse \tag{5.2}$$

Das folgende, aus der Anwendung entnommene, Beispiel soll der Verdeutlichung dienen:

$$36 = abgerundet(abgerundet(39,403561)/6) * 6 \tag{5.3}$$

Somit gehört der Pixel mit der Spaltenkoordinate 39,403561 zum Pixelblock mit der Pixelblockadresse 36. Der nächste Pixelblock ist folglich mit der Pixelblockadresse 42 versehen. Um eventuell auftretende Schwarzwerte im Bild zu behandeln, wenn die Spaltenkoordinate dem Wert -1 (BLACK_VALUE_SRC) entspricht, wird in einem solchen Fall die Spaltenkoordinate des vorherigen Durchlaufs genutzt, wobei diese dann aufgerundet wird um die nächste höhere bzw. folgende Spaltenkoordinate zu erhalten. Für die CineCard v2 wird das gleiche Verfahren zur Ermittlung der Pixelblockadressen verwendet. Durch die verwendete Burstlänge von 8 und dem Umstand, dass die Daten für jeden Pixel einzeln in einer Speicherzelle abgelegt sind, ergibt sich hierbei eine Pixelblockgröße von 8. Es ergeben sich, um eine komplette Zeile bei einer Auflösung von 1920 x 1080 zu speichern, 240 notwendige Pixelblöcke. Um zu ermitteln wie häufig ein Pixelblock zum Darstellen der in ihm enthaltenen Pixel benötigt wird, wird mit jedem Durchlauf temporär die ermittelte Pixelblockadresse sowie die Zeilenkoordinate gesichert. Auf diese Werte wird dann geprüft, ob es sich bei dem aktuellen Durchlauf um den gleichen Pixelblock wie zuvor handelt. Handelt es sich um den gleichen Pixelblock, so wird ein entsprechender Zähler inkrementiert. Es folgt die Beschreibung der „doBlockDetermination“-Funktion. Zum Speichern der in dieser Funktion ermittelten Daten wird ein weiteres Objekt vom Typ „PixelBlockDataLUT“ verwendet. Zur Vereinfachung wird dieses fortan als „PelsubArray“ bezeichnet. In der Funktion werden die Werte der Pixelblockaddresse und

der Zeilenkoordinate aus dem zuvor erstellten PixelBlockDataLUT-Objekt ausgelesen und temporär gespeichert. Diese Werte werden dahingehend geprüft, ob sie dem Wert -1 (BLACK_VALUE_SRC) entsprechen. Ist dies der Fall, werden diese Daten nicht in das PelsubArray abgelegt. Des Weiteren wird geprüft, ob sich die Pixelblockadresse von der des vorherigen Durchlaufs unterscheidet. Sollte dies nicht der Fall sein, wird angenommen, dass die Daten bereits vorhanden sind und somit nicht noch einmal gesichert werden müssen. Nach der Prüfung wird abhängig von der Bildorientierung (wie zuvor beschrieben), die Zeilenkoordinate aufgerundet und abgerundet zwischengespeichert. Das Auf- und Abrunden der Zeilenkoordinate geschieht, um die für die Bilineare Interpolation aufeinander folgenden Zeilen zu erhalten. Die Bikubische Interpolation wurde aus Zeitgründen noch nicht implementiert, so muss später zu diesem Zweck das Verhalten an dieser Stelle angepasst werden.
Es wird kontrolliert, ob sich die gerundeten Werte für die Zeilenkoordinate innerhalb der Bildgrenzen befinden. Entsprechend werden sowohl die Werte der Pixelblockadresse, des Blockcount, als auch die gerundete Zeilenkoordinate in zwei aufeinander folgenden Zeilen des „PelsubArray“ an entsprechender Spaltenposition abgelegt. Sollte bei der anfangs getätigten Überprüfung der Pixelblockadresse auf die des vorherigen Durchlaufs eine Übereinstimmung festgestellt worden sein, so wird auf eine Übereinstimmung der gerundeten Zeilenkoordinate auf die gerundete des vorherigen Durchlaufs geprüft. Sollte diese Kontrolle keine Übereinstimmung ergeben, wird mit einer weiteren Prüfung fortgefahren. Diese zusätzliche Überprüfung ist notwendig, weil bis zu diesem Zeitpunkt nur auf die Daten des vorherigen Durchlaufes geprüft wurde. Dieses Vorgehen, ist für den häufigst auftretenden Fall, in dem sich die Zeilenkoordinate und die Pixelblockadresse nicht ändern bzw. unterscheiden, ausreichend. Treten jedoch durch die Transformation bedingte „Zeilensprünge“ auf, kann es dazu kommen, dass nach besagtem Zeilensprung wieder auf den gleichen Pixelblock mit der gleichen zuvor benötigten Zeilenkoordinate zugegriffen wird. Um diese Pixeldaten nicht wiederholt in das „PelsubArray“ zu schreiben, wird nun das „PelsubArray“ durchlaufen und geprüft, ob der untersuchte Pixelblock (Pixelblockadresse mit zugehöriger Zeilenkoordinate) sich bereits im „PelsubArray“ befindet. Ist er bereits im „PelsubArray“ vorhanden, so werden die aktuellen Daten nicht noch einmal in das „PelsubArray“ geschrieben. Andernfalls werden die Daten im „PelsubArray“ abgespeichert. Die zuvor getätigte Prüfung für den häufig auftretenden „Standard“-Fall wird getätigt, um ein unnötig häufiges Durchlaufen des gesamten „PelsubArray“ zu vermeiden und somit die Laufzeit nicht unnötig zu erhöhen. Sind alle Spalten der aktuellen Zeile des in der „doBlockCount“-Funktion erstellten PixelBlockDataLUT-Objektes durchlaufen, wird das PelsubArray mit Hilfe eines „Bubblesort“-Sortierverfahren anhand des Blockcounts sortiert. Das „Bubblesort“-Verfahren wurde zunächst wegen der unkomplizierten Implementierung genutzt. Weil

die Entwicklung der CineCard v2 zeitgleich zu dieser Arbeit stattfindet, war es erforderlich immer wieder zeitnah Teilergebnisse zur Verfügung zu stellen. Somit wurden zum Teil Entscheidungen getroffen, mit dem Grundgedanken einer zeitnahen Implementierung, um den Fertigungsprozess nicht unnötig zu blockieren. Es ist angedacht zu einem späteren Zeitpunkt ein effizienteres Sortierverfahren, wie z.B. „Mergesort“, zu verwenden. Der eigentliche Gedanke hinter der Sortierung nach der Häufigkeit der Pixelblöcke ist, sicherzustellen das Pixelblöcke welche für die meisten darzustellenden Pixel benötigt werden, am Anfang einer Zeile und somit definitiv aus dem Speicher geladen werden können. Zwischen dem Darstellen von zwei Zeilen, in der sogenannten Austastlücke, steht am meisten Bandbreite zur Verfügung. Ein Nachteil dieses Vorgehens könnte darin bestehen, dass die als erstes geladenen und somit ältesten Pixelblöcke als erstes, bei ungenügend Speicherplatz im Zeilenspeicher, wieder überschrieben werden. Dieses Problem tritt jedoch in der Praxis durch das vorhanden sein von genügend Speicherplatz nicht auf.

In der Funktion „doAddressCalculation“ werden die Adressen der Pixelblöcke für die Hardware, so wie sie tatsächlich im Speicher abgelegt sind, bestimmt. Zu dem wird ein Objekt der Klasse „RowMemoryLUT“ erstellt und mit den entsprechenden Daten befüllt. Die Klasse „RowMemoryLUT“ dient als Speicherklasse um ein Abbild des Zeilenspeichers, welcher in 4.3.1 beschrieben wurde, der CineCard zur Verfügung zu stellen. Dieses Abbild wird für die spätere Interpolation benötigt. Das eigentliche Bestimmen der Adressen wird durch die ausgelagerte Funktion „calcCmdListEntry“ realisiert. Der Funktion werden die Werte für die Pixelblockadresse und der Zeilenkoordinate aus dem „PelsubArray“ übergeben. Die Pixelblockadresse wird noch zuvor durch die Pixel-Blockgrösse dividiert, um die Speicherspalte, wo der entsprechende Pixelblock untergebracht ist, zu bestimmen. Wie in 4.3.2 beschrieben, ist die Speicherstruktur der beiden CineCard Version unterschiedlich. Somit ist auch eine unterschiedliche Behandlung zur Ermittlung der Adressen nötig. Es wird die entsprechende Bank, Zeile und Spalte des Speichers ermittelt. Die bestimmten Ergebnisse werden in ein Objekt der „AddressList“-Klasse abgelegt, wobei diese Werte durch Bitshiften zusammengefasst noch einmal gesondert unter dem Namen „MemAddress“ gespeichert werden. Diese Klasse verfügt über eine Struktur für die einzelnen Werte und dient als Speicherklasse. Der in 4.3.1 beschriebene RowOffset, wird anhand eines Zählers ermittelt und ebenfalls in dem AddressList-Objekt abgelegt. Zudem wird ein „StoppFlag“, was das Ende einer Zeile angibt, mit abgespeichert. Der Zustand des „StoppFlag“ wird ebenfalls durch Bitshiften dem Wert „MemAddress“ beigefügt, wobei hierfür das oberste Bit getoggelt wird. Weitere Werte die in dem Objekt gespeichert werden, dienen ausschließlich zur besseren Debug-Möglichkeit und sollen deshalb hier nicht näher beschrieben werden. Der in 4.3.1 beschriebene nötige

Index, der angibt welcher Zeilenspeicher für die entsprechende Spaltenposition den obersten darstellt, wird innerhalb der Funktion „doAddressCalculation“ erstellt und genutzt um die Einträge für die Speicherspaltenadresse und Zeilenkoordinate in die entsprechende Zeile des „RowMemoryLUT“-Objektes zu schreiben. Am Ende der Zeile wird geprüft, ob das Auftreten von nicht-Schwarzwerten (-1, BLACK_VALUE_SRC) zum ersten Mal mit der aktiven Zeile vor kam. Ist das der Fall, wird eine temporäre Kopie der bis dahin getätigten AddressList-Einträge erstellt. Diese Einträge sollten aus „memAddress“-Einträgen (siehe 4.3.2) bestehen, in denen das oberste Bit getogglet ist, also ein Ende der Zeile signalisiert wird. Die Anzahl der Einträge ist deckungsgleich mit der Anzahl der vorausgegangenen Zeilen, die nur Schwarzwerte enthalten haben. Die „MemAddress“-Einträge der aktiven Zeile, werden an den Anfang der AddressList geschrieben und die temporär gespeicherten Einträge wieder angefügt. Dies geschieht aus dem Grund, dass immer nur eine Bildzeile auf einmal aus dem Arbeitsspeicher in den Zeilenspeicher geschrieben werden kann. Um bei der Bilinearen Interpolation die benötigten zwei Zeilen zur Verfügung zu haben, wird entsprechend die erste Zeile mit nicht-Schwarzwerten an den Anfang der AddressListe geschrieben, damit diese sich bereits im Zeilenspeicher befindet. Zum Zeitpunkt der Interpolation kann die zweite Zeile falls nötig geladen werden. Somit kommt es nicht dazu, dass gerade bei Bildanfängen zwei Zeilen auf einmal geladen werden müssten. Für die Bikubische Interpolation muss das Verfahren entsprechend auf vier Zeilen erweitert bzw. angepasst werden. Im Anschluss wird die Funktion „sortRowMemoryLUT“ aufgerufen. Sie sortiert in Abhängigkeit des oben genannten Index, die Zeilen des „RowMemoryLUT“-Objektes, sodass der durch den Index angegebene oberste Zeilenspeicher, der entsprechenden Spaltenposition, auch wirklich die oberste Zeile innerhalb der Matrix ist. Dies vereinfacht das Vorgehen bei der folgenden Interpolation. So wird vermieden, immer auf den Indexwert zu prüfen, sondern man kann nativ von einer absteigenden Zeilenreihenfolge ausgehen. Des Weiteren wird für jeden Pixelblock bestimmt, welche Spaltenkoordinaten durch die in ihm enthalten Pixel abgedeckt werden. Es erhält somit jeder Pixel seinen gesonderten Eintrag und ist nicht mehr als Block zusammengefasst. Dies ist nötig um für die Interpolation eine pixelweise und nicht blockweise Auflösung zur Verfügung zu haben. Ist die oben erwähnte Schleife über alle Zeilen der Bildpunktkoordinaten, welche sich im „WarpingLUT“-Objekt befinden, durchgelaufen, ist somit das vollständige „AddressList“-Objekt ermittelt bzw. erstellt worden. So kommt es zum Aufruf der „reduceAddressListWithBurstCount“-Funktion. Es ist zu erwähnen, das zum Ablauf der Schleife über alle Zeilen noch die Ermittlung der Interpolationsdaten, welche noch im weiteren Verlauf beschrieben werden, gehört. Die „reduceAddressListWithBurstCount“-Funktion geht schrittweise durch die Einträge des „AddressList“-Objektes und prüft ob sich ermittelte Speicherbank- und Speicherzeilenadresse vom vorherigen Eintrag unterscheiden. Ist dies nicht der Fall, können die Daten innerhalb eines Burst's , wie in 4.4 beschrieben, zusammengefasst werden, wobei

die maximale Länge eines Burst's beschränkt ist. Bei der CineCard v1 darf ein Burst 32 Speicherzellen, bei der CineCard v2 64 Speicherzellen umfassen. Dementsprechend werden die Einträge des „AddresssList"-Objektes reduziert. So wird für Daten, die sich auf der gleichen Bank- und in der gleichen Zeile des Speichers befinden, die maximale Länge auf einen Eintrag reduziert. Dieser gibt sozusagen die Startadresse an und der ermittelte Burst Count definiert die Länge des Burst's in dem die Daten sequentiell eingelesen werden. Die Verwendung des Burst Count steigert die Effizienz und entlastet die verfügbare Bandbreite zum Arbeitsspeicher.

5.2.6 Interpolation / InterpolationLUT

Die Klasse „InterpolationLUT" dient zur Speicherung der ermittelten Koeffizienten für X- und Y-Richtung bzw. der in 4.1 erwähnten Interpolationskommandos. Zudem wird in ihr ein Index abgelegt, der zur besseren Debug-Möglichkeit dient. Nachdem zuvor die Adressen der zu ladenden Pixelblöcke bestimmt wurden, werden in der Funktion „doY-Interppolation" die Y-Koeffizienten bzw. Interpolationskommandos einer Zeile ermittelt. Zu Begin wird für jeden Pixel der jeweiligen Spalte der Index in das „InterpolationLUT"-Objekt geschrieben. Es werden die Spaltenkoordinaten, über alle vier Zeilen der sortierten „RowMemoryLUT" , der aktuellen Spalte ermittelt. Diese werden dann überprüft, ob sie dem Schwarzwert (-1) entsprechen. Werden ausschließlich Schwarzwerte festgestellt, so wird als Interpolationsparameter (bzw. Kommando) der Wert BLACK_VALUE an die Position der aktuellen Spalte in das InterpolationsLUT-Objekt geschrieben. Somit wird dem FPGA der CineCard signalisiert, das keine Interpolation stattfinden muss und einfach ein schwarzer Pixel ausgegeben werden kann. Ist ein nicht-Schwarzwert als Spaltenkoordinate ausgelesen worden, so wird nun die entsprechende Zeile des WarpingLUT-Objekt untersucht. Wird der im Zeilenspeicher verfügbare Pixel, zur Darstellung des benötigten Pixels in der WarpingLUT gebraucht, so wird der Spaltenindex des Pixels in der WarpingLUT gespeichert. Ergibt die Untersuchung der WarpingLUT-Zeile dass der verfügbare Pixel nicht zur Darstellung gebraucht wird, so wird der Wert „NO_VALUE" als Interpolationskommando in die InterpolationsLUT abgelegt. Anhand des Spaltenindexes wird die zugehörige Zeilenkoordinate des Pixels aus dem WarpingLUT-Objekt bestimmt. Es wird überprüft, ob der verfügbare Pixel im Zeilenspeicher die gleiche Zeilenkoordinate besitzt. Stimmen Zeilenkoordinaten überein, so wird anhand der Spaltenkoordinate des benötigten Pixels und der Zeilenindexposition innerhalb des Zeilenspeichers des verfügbaren Pixels, der Interpolationskoeffizient für den verfügbaren Pixel bestimmt. Dafür wird die Funktion „detYParam" aufgerufen. Wie in 4.2 beschrieben,

wird die Zeilenposition des Zeilenspeichers zur Bestimmung der Pixeladdresse innerhalb der Interpolationsmatrix 4.5 verwendet. Die Zeilenkoordinate bzw. deren Nachkommastellen werden zur Ermittlung der Sub-Pixeladdresse genutzt. Der im Zeilenspeicher verfügbare und zur Bestimmung des Y-Koeffizienten verwendete Pixel , genauer dessen Zeilen- und Spaltenkoordinate, wird in einem Array (XValues) gespeichert und somit der X-Interpolation übergeben. Stimmen Zeilenkoordinate des im Zeilenspeicher (RowMemoryLUT) verfügbaren Pixels und die des benötigten (WarpingLUT) nicht überein, wird die der Zeilenkoordinate am nächsten liegende Zeile, durch Runden, im Zeilenspeicher gesucht. Wird diese gefunden, so wird die Zeilenposition des Pixels innerhalb des Zeilenspeichers als Interpolationsparameter für eine Nächster Nachbar Interpolation genutzt. Ist auch dies nicht der Fall, wird die Zeilenkoordinate auf einen Schwarzwert geprüft, je nach Ergebnis wird als Interpolationskommando der Wert BLACK_VALUE oder NO_VALUE übergeben.

Nachdem die Y-Parameter für eine Zeile bestimmt wurden, werden die X-Parameter in der Funktion „doXInterpolation“ ermittelt. Zur Speicherung der X-Parameter wird ein separates InterpolationLUT-Objekt verwendet. Zunächst werden aus dem WarpingLUT-Objekt die Daten des zur Darstellung des Ausgangsbildes benötigte Pixel ermittelt. Im Folgenden wird das Array XValues, welches die Y-Interpolierten Pixeldaten enthält, betrachtet. Das Array repräsentiert das Schieberegister der CineCard, siehe 4.4. Die verfügbaren Pixel im Array werden durch einen Index(XIndex) adressiert, er wird genutzt, um die zu betrachtenden Pixel für die X-Interpolation aus dem Array XValues zu extrahieren, wobei immer vier Pixel extrahiert werden. Dieser Index setzt sich zusammen aus dem aktuellen Schleifendurchlauf und dem Umstand, ob zuvor ein Stopp-Kommando (STOP_X_INTERPOLATION) eingefügt wurde. Trifft dies zu, darf auf der CineCard zeitgleich keine Y-Interpolation durchgeführt werden, weil sonst für die folgende X-Interpolation, nach dem Stopp-Kommando, nicht mehr die passenden Daten vorhanden wären. Ein neuer Y-Interpolierter Pixel würde im Schieberegister vorhanden sein. Ebenso darf der folgende Y-Parameter, nach dem Stopp-Kommando, keine Interpolation steuern, da sonst ein benötigter Pixel verloren gehen würde. Deswegen wird in einem solchen Fall, für beide Y-Interpolationsparameter das Kommando NO_ACTION eingefügt. Dementsprechend wird nach einem Stopp-Kommando der XIndex angepasst. Es wird überprüft, ob die Zeilenkoordinate des benötigten Pixels (WarpingLUT) einem Schwarzwert entspricht. Trifft dies zu, wird das Kommando BLACK_VALUE eingefügt. Andernfalls wird ermittelt, ob sich der benötigte Pixel, genauer die Spaltenkoordinate, unter den extrahierten verfügbaren Pixeln befindet und dessen Spaltenindex ermittelt. Anhand des Spaltenindex und dem Wert der Spaltenkoordinate des benötigten Pixels, wird nun der X-Koeffizient zur Interpolation bestimmt und abgelegt. Wird festgestellt, dass der benachbarte Pixel ein Schwarzwert aufweist, wird der X-Koeffizient für eine

Nächster Nachbar Interpolation angepasst. Wurde bis hier nicht der benötigte Pixel unter den extrahierten ausfindig gemacht, so wird kontrolliert, ob sich dieser dennoch im Array XValues (den verfügbaren Y-Interpolierten Pixeln) befindet, aber nicht extrahiert wurde. Trifft dies zu, wird das Shift-Kommando (SHIFT_X_INTERPOLATION) eingefügt. Hierdurch wird auf der CineCard die X-Interpolation gestoppt und die Y-Interpolation ausgeführt, wodurch der benötigte Pixel im Schieberegister verfügbar wird. Sollte der benötigte Pixel noch immer nicht aufgefunden werden, so wird das Kommando NO_VALUE gespeichert. Am Ende einer Zeile wird das Kommando END_OF_LINE eingefügt. Zudem wird überprüft, ob die InterpolationsLUT-Objekte der X- und Y-Interpolation über die selbe Anzahl von Einträgen verfügen. Ist dies nicht der Fall, so wird das kleinere Objekt mit NO_ACTION-Kommandos aufgefüllt, um eine gleiche Anzahl von X- bzw. Y-Parameter pro Zeile zu erhalten. Der Vollständigkeit halber ist hier zu erwähnen, dass die LUT-Objekte der Alpha- und BlackLevel-Werte um NO_ACTION-Kommandos erweitert wurden, wenn in die X- bzw. Y-InterpolationLUT Interpolationskommandos eingefügt wurden. Dies ist nötig um einen direkten Zusammenhang zu den verarbeiteten Pixeln zu bewahren.

5.2.7 OutputHandling

Die Klasse „OutputHandling“ ist dafür zuständig, die ermittelten Daten in das für die CineCard erwartet Format zu bringen. Zur Vereinfachung der Verarbeitung werden die Daten aus den LUT-Objekten, wo sie in Matrixform gespeichert sind, in eine Listenform gebracht, wobei jede Zeile dieser Listenform über den Inhalt einer Matrix-Zelle verfügt. Hierfür wird die Funktion „getParameterRows“ verwendet. Sie durchläuft die LUT-Objekte zeilen- und spaltenweise und speichert die Daten der einzelnen Matrixeinträge in einer Liste ab. Somit kann die Liste, Element für Element, abgearbeitet werden. Als Eingabeformat erwartet die CineCard v1 die Daten in einem PNG-Format. Hierfür sind die Funktionen „addressList2PNG“ und „parameter2PNG“ zuständig. Aus der Tatsache heraus, dass jede Speicherzelle 72 Bit groß ist und mit jedem Lesebefehl immer zwei Speicherzellen ausgelesen werden, werden die Addressdaten in 144 Bit lange Vektoren geschrieben, wobei immer Speicheradresse (Speicher-Bank, Zeile und Spalte) und der in 4.3.1 beschriebene RowMemOffset zusammengehörig abgelegt werden. So ergibt sich für die Datenspeicherung eines Pixels folgende Struktur.

Abbildung 5.3: Datennstruktur der Adressdaten für CineCard v1

Wie der Abbildung 5.3 zu entnehmen ist, wird zur Speicherung des RowMemOffset 9 Bits verwendet. Das „O“ in der Grafik steht für Offset. Das „S“ steht stellvertretend für das „StoppFlag“ (siehe 5.2.5) und ist mit einem Bit reserviert. Ist das Ende einer Zeile erreicht, wird dieses Bit auf „1“ gesetzt. Es folgen 11 Bits (R = Row) zur Speicherung der Speicherzeile. Danach werden 2 Bits (B = Bank) genutzt, um die verwendete Speicherbank zu adressieren. Den Abschluss stellt die Speicherspaltenadresse (C = Column) dar und ist mit 10 Bits veranschlagt. So werden zur Speicherung des Pixeldaten insgesamt 33 Bits verwendet. In den 144 Bit langen Vektor, werden die Pixeldaten von drei Pixeln zusammen abgelegt. Die übrigen, nicht verwendeten Bits (44 Bits), des Vektors werden als „Freebit's“ bezeichnet, welche auf „0“ gesetzt werden. Sie dienen zum Trennen der einzelnen Pixeldaten, als auch zum Auffüllen des Restvektors. Das Schreiben der Daten an die entsprechenden Bit-Positionen wird durch Bit-Maskierung, Bit-Shifting und Addition erreicht. Zum Speichern der Parameterdaten eines Pixels werden insgesamt 64 Bit benötigt. Diese teilen sich wie folgt auf. Zum Speichern der X- sowie der Y-Parameter werden jeweils 5 Bits verwendet. Jeder Alphakanal wird mit 13 Bits veranschlagt, was bei drei Alphakanälen zu insgesamt 39 Bits zum Speichern der Alphawerte führt. Für jeden BlackLevel-Kanal werden 5 Bits und somit insgesamt 15 Bits benötigt. Die Übrigen 8 Bits der Speicherzelle bzw. des Vektors werden wiederum als die bereits erwähnten „Freebit's“ verwendet. Die so erhaltenden Vektoren, wie auch die der Adressdaten, werden in ein OpenCV-Mat-Objekt gespeichert und anschließend mit Hilfe des OpenCV-Befehls „imwrite“ als PNG-Datei gespeichert.

Die CineCard v2 erwartet die Daten in einem Binärformat mit der Dateiendung „.bin“. Im Unterschied zur CineCard v1 sind die Speicherzellen 64 Bit groß, so dass die Adressdaten für jeden Pixel einzeln in einer Speicherzelle abgelegt werden.

Abbildung 5.4: Datennstruktur der Adressdaten für CineCard v2

Wie der Abbildung 5.4 entnommen werden kann, wird bei der CineCard v2 das oberste Bit (64) zur Speicherung des „StoppFlag“ verwendet. Absteigend gefolgt von insgesamt 21 „Freebit's“, welche Platz bieten für eventuelle spätere Erweiterungen. Die mit „b“ markierten 9 Bit beinhalten den Wert des in 5.2.5 ermittelten Burst Count. Darauf folgen 9 Bit für den „RowMemOffset“ (O = Offset) und 14 Bit für die Speicherzeilenadresse (R = Row). Für die Adressierung der Speicherbank (B = Bank) stehen hier 3 Bit zur Verfügung. Die Speicherspaltenadresse wird mit 7 Bit veranschlagt. Die Unterschiede in den benötigten Bitlängen für die Speicher-Zeile, Bank und Spalte liegt begründet, in der unterschiedlichen Speicherstruktur (4.3) bzw. dem verwendeten Arbeitsspeicher. Das Schreiben der Daten an die entsprechenden Bit-Positionen wird

wieder durch Bit-Maskierung, Bit-Shifting und Addition erreicht. Die Speicherung der Parameterdaten unterliegt der gleichen Struktur wie sie bereits bei der CineCard v1 geschildert wurde. Für die Namensgebung der Adress- bzw. Parameterdatei wird die vom Nutzer gewählte Eingabe, bezüglich Nutzung von Alpha- und BlackLevel-Dateien und der Nummerierung der genutzten Koordinatendatei ausgewertet und einbezogen. Mit Hilfe der Boost-Bibliothek werden gegebenenfalls die Verzeichnisse erstellt, die vom Nutzer im Ausgabepfad (pathOutputData) definiert wurden. Unter Verwendung eines C++-OutputFileStreams, werden die Dateien mit der Option „std::ios::binary“ als Binärdateien gespeichert. Für die Erstellung der Simulationsdaten für die CineCard v2 werden die Daten in der gleichen Struktur wie oben beschrieben abgelegt. Jedoch wird als Ausgabeformat eine Klartext-Datei erstellt. Zudem werden die Datenvektoren in Hexadezimalwerte umgewandelt. Der Anfang der Dateien wird mit einem Header versehen, welcher Auskunft über den Typ des verwendeten Arbeitsspeichers, sowie über den Speicherbereich der verwendet wird gibt. Damit in der Simulation der Speicher an den richtigen Positionen mit den Datenvektoren, welche nun als Hexadezimalwert vorliegen, initialisiert werden kann, ist es nötig vor jeden Vektor die entsprechende Speicheradresse zu schreiben. Hierfür wird der vom Nutzer übergebene Wert in „memStartAddress“ genutzt. Ausgehend von diesem Startwert wird die entsprechende Speicheradresse fortlaufend bestimmt und vor den entsprechenden Vektor in der Simulationsdatei geschrieben. Hierfür werden die Funktionen „increaseAddress“ und „calcMemAddress“ genutzt.

5.2.8 Stitching

Die „Stitching“-Klasse ist für die Umsetzung der Vorschau des Panoramas zuständig. Wird vom Nutzer die Eingabe „stitching“ als „distortionForm“-Parameter übergeben, so wird diese Klasse aufgerufen und ausgeführt. Zunächst wird mit Hilfe der Funktion „readWarpIndexFile“, die im Binär-Format vorliegenden Transformationskoordinaten eingelesen und in einer Matrix gespeichert. Vor dem Speichern der Koordinaten werden diese noch auf ihrer Werte geprüft, genauer, ob sie innerhalb des Bildrasters liegen. Ist dies nicht der Fall, wird für die Koordinate der Wert BLACK_VALUE_SRC gespeichert. Die Datei mit den Alpha-Werten, wird durch den Aufruf der „loadAlphaFile“-Funktion eingelesen. Auch sie liegt im Binär-Format vor. Die Alpha-Werte werden in eine separate Matrix gespeichert. Das Einlesen der Parameter-Dateien, welche die Werte für Bildgröße und Offset enthalten, werden von der Funktion „readStitchingWarpParameters“ eingelesen. Hierfür wird der vom Nutzer übergebene Pfad zum „Stitching“-Stammverzeichniss (pathStitchingData) genutzt. Mittels einer Schleife werden nacheinander für alle Einzelbilder des Gesamtpanoramas die zugehörigen Parameter-Daten eingelesen. Die dabei gewonnenen Werte für die Bildgröße und den Offset werden in Vektoren abgelegt.

Während des Einlesens, werden die obere und untere Grenze des Panoramas ermittelt. Zur Festlegung der oberen Grenze wird untersucht, welches Einzelbild den größten, vertikalen Offset-Wert besitzt. Dieser stellt dann die obere Grenze des Panoramas da. Für die Bestimmung der unteren Grenze wird jeder vertikale Offset und die Anzahl der Hochkant-Zeilen jedes Einzelbildes addiert. Der so erhaltene niedrigste Wert, aller Bilder, stellt die untere Grenze des Panoramas da. Somit ist sichergestellt, das zwischen den Grenzen für jedes Einzelbild die entsprechenden Bilddaten verfügbar sind. Darauf folgend, wird die Funktion „resizeIdxAndAlpha“ genutzt, um die Matrix der Transformationskoordinaten und die der Alpha-Werte auf die Panoramagrenzen zu beschränken. Hierfür werden die ermittelten Werte für die obere sowie untere Grenze als auch die Bildparameter übergeben. Anhand der oberen Grenze und des zugehörigen vertikalen Offset des Einzelbildes wird die erste, zur Extraktion der Daten benötigte, Hochkant-Zeile der Transformationkoordinaten und der Alphawerte bestimmt. Folgend werden alle Daten extrahiert und in eine, der Panoramagröße angepasste, neue Matrix geschrieben. Dies geschieht bis alle Hochkant-Zeilen bis zur unteren Grenze durchlaufen wurden. Durch die Tatsache, das die im Landscape-Format vorliegenden Eingangsbilder für die Vorschau des Panoramas beim Einlesen durch die CineCard rotiert werden, ist ebenfalls eine Rotation der Transformationskoordinaten notwendig. Dies wird durch die Funktion „rotateCoordinates“ verwirklicht.

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{5.4}$$

Durch Verwendung der Rotationsmatrix 5.4, werden die Transformationskoordinaten um 90 Grad im Uhrzeigersinn rotiert. Anschliessend findet durch die Addition eines Vektors eine Translation statt, um die Koordinaten wieder in den positiven Wertebereich des Koordinatensystems zu verschieben. Ist die Vorverarbeitung für die Vorschau des Panoramas abgeschlossen so werden die Transformationskoordinaten und Alpha-Werte den entsprechenden Speicherklassen übergeben. Mit diesen wird dann die Bestimmung der Adress- und Parameter-Daten, wie vorangehend beschrieben, durchgeführt.

Kapitel 6

Evaluierung

Eine wichtige Anforderung an das zu entwickelnde Programm ist es, im Vergleich mit dem für die CineCard v1 bisher genutzten Matlab-Skripts, eine kürzere Laufzeit zu erzielen. Dementsprechend wird im Folgenden die Laufzeit des Matlab-Skripts und die des entwickelten C++-Programms untersucht.

Zunächst wird unter Verwendung eines zufällig ausgewählten Datensatzes die durchschnittliche Laufzeit ermittelt. Es wurden insgesamt je 10 Durchläufe mit dem gleichen Datensatz durchgeführt.

Tabelle 6.1: Laufzeit von 10 Durchläufen mit gleichem Datensatz

Durchlauf	**Matlab Laufzeit in s**	**C++ Laufzeit in s**
1	1055,88	159,53
2	1047,34	159,66
3	1038,25	159,82
4	1051,71	159,15
5	1055,88	159,93
6	1052,11	158,87
7	1054,94	159,29
8	1046,83	158,9
9	1054,37	169,32
10	1055,23	159,72

Die Testläufe wurden alle auf dem gleichem System durchgeführt. Das Systemspezifikationen sind

- Betriebssystem: Ubuntu 12.04 64-Bit
- Kernel: Linux 3.9.0.-44-generic
- Arbeitsspeicher: 48 GB

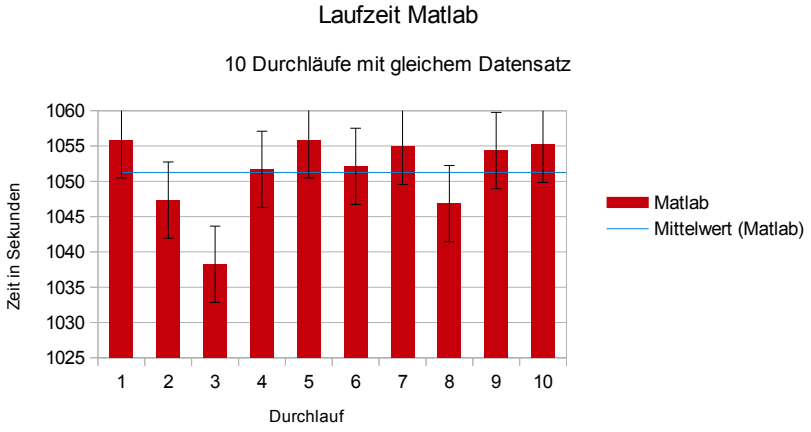
- Prozessor: Intel Xenon(R) CPU E5620 @ 2,40 GHz x 8

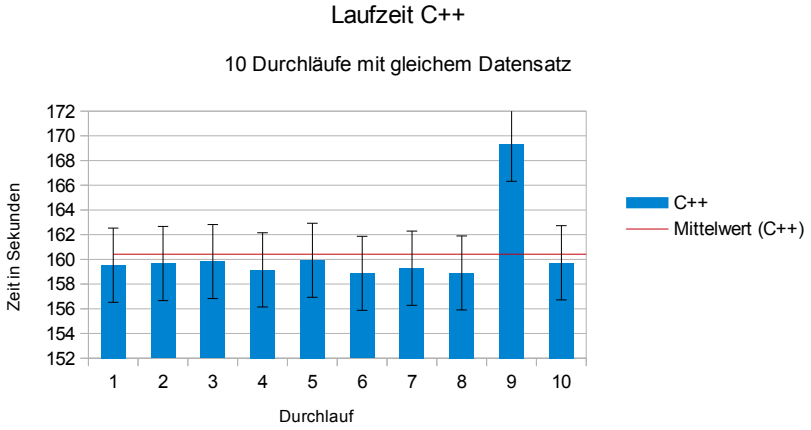
Aus den ermittelten Werten (siehe Tabelle 6.1) wird nun der Durchschnitt bzw. das arithmetische Mittel bestimmt. Ebenso werden die Varianz und die Standardabweichung ermittelt und in 6.2 dargestellt.

TABELLE 6.2: Durchschnitt, Varianz und Standardabweichung der Laufzeiten

	∅ **Laufzeit in s**	**Varianz in s**	**Standardabweichung in s**
Matlab	1051,254	28,684024	5,35574682
C++	160,419	8,927249	2,987850231

Die ermittelten Werte für die Laufzeit, sowie die durchschnittliche Laufzeit und die Standardabweichung werden folgend in Diagrammen dargestellt.

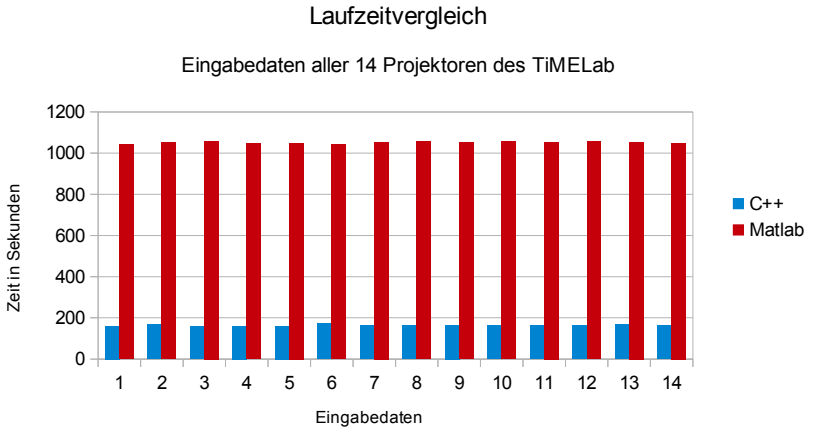




Wie aus den vorangegangen Diagrammen zu entnehmen ist, kommt es bei den Testdurchläufen für Matlab sowie für C++ zu Ausreißern. Diese lassen sich durch unterschiedliche Last des Testsystems zu den jeweiligen Zeitpunkten erklären. So sind die zu testenden Programme nicht die einzige Anwendungen, die Zeitgleich vom Betriebssystem ausgeführt werden. Systemprozesse, andere Programme, sowie Dateizugriffe die im Hintergrund ablaufen, können die Systemlast beeinflussen. Diese unterschiedlichen Systemlasten kommen auch im normalen Betrieb vor. So werden die Ausreißer in die Bestimmung der durchschnittlichen Laufzeit, sowie der Standardabweichung mit einbezogen, um somit ein realitätsnahes Ergebnis zu erlangen. Im Folgenden wurde je ein Durchlauf für 14 unterschiedliche Datensätze durchgeführt. Die Datensätze entsprechen den zur Bespielung des TiMELAB nötigen Daten für alle 14 Projektoren.

TABELLE 6.3: Laufzeit über 14 unterschiedliche Datensätze

Datensatz	**Matlab Laufzeit in s**	**C++ Laufzeit in s**
1	1042,87	160,3
2	1052,47	167,79
3	1059,12	158,7
4	1046,77	159,56
5	1050,21	159,47
6	1043,91	171,94
7	1054,23	161,44
8	1057,44	163,79
9	1051,31	165,51
10	1056,23	162,86
11	1053,97	162,45
12	1058,25	162,39
13	1054,96	169,07
14	1050,01	162,93



Kapitel 7

Fazit

Das entwickelte Programm ist in der Lage die spezifizierten Aufgaben umzusetzen. So ist es möglich, verschiedenste Formate für die Transformationskoordinaten zu behandeln. Das Programm ermittelt anhand der Transforamtionskoordinaten die nötigen Bildpunktadressen des Eingangsbildes, sowie die nötigen Interpolationskoeffizienten. Die Alpha- und Blacklevel-Werte werden entsprechend verarbeitet und zusammen mit den Interpolationskoeffizienten um die nötigen Steuerkommandos erweitert. Es wurde ein Simulationsmodus verwirklicht, mit Hilfe dessen, für eine gegebene Anzahl von Bildzeilen, die ermittelten Adress- und Parameterdaten in einem speziellem Simulationsformat ausgegeben werden. Durch die Angabe der jeweiligen Version der CineCard ist es möglich, die Daten für die bereits entwickelte bzw. die sich in der Entwicklung befindliche Version der CineCard zu erstellen. Somit ist eine Abwärtskompatibilität gegeben. Die erzeugten Daten wurden manuell auf ihre Richtigkeit geprüft. Zudem wurden die erzeugten Daten für die CineCard v1 mit den Daten, welche durch das Matlab-Skript erzeugt wurden verglichen. Auf diese Weise konnte eine korrekte Ermittlung der Daten bestätigt werden. Darüber hinaus wurden die vom Programm erzeugten Daten genutzt, um damit die CineCard v1 zu versorgen und im TiMELAB eine Ausgabe von Video- und Bildmaterial durchzuführen. Bei der Begutachtung des Video- und Bildmaterials traten keine sichtbaren Fehler auf. Durch den Umstand, das die CineCard v2 sich noch in der Entwicklung befindet, konnten die Daten nur manuell und in einer VHDL-Simulations geprüft werden. Diese Prüfungen ließen keinen Schluß auf fehlerhafte Daten zu. Die Steuerung des Programms ist mit Hilfe einer Konfigurationsdatei, sowie durch die Übergabe von einzelnen Parametern beim Aufruf des Programms durch die Konsole möglich. Wie in Kapitel 6 beschrieben, ist die Verbesserung der Laufzeit im Vergleich zum Matlab-Skript erzielt worden. Das entwickelte Programm benötigt in etwa nur 15-16% der Zeit zur Erstellung der Daten für die CineCard v1, als das zuvor genutzte Matlab-Skript. Ein weiterer Vorteil ist, dass durch den Einsatz von C++ nun nicht mehr die Installation von Matlab auf

der CineBox notwendig ist. Die Erstellung der Panoramavorschau für die OmniCam ist zum jetzigen Zeitpunkt nur teilweise implementiert. Aus zeitlichen Gründen konnte die Umsetzung noch nicht vollständig abgeschlossen werden. So fehlt im Moment noch die Bestimmung der horizontalen Überlappungsbereiche (Blending) und deren Markierung in den Ausgabedaten. Des Weiteren ist noch eine horizontale Skalierung der Einzelbilder vorzunehmen damit sich das Panorama auf einem einzelnen Display darstellen lässt.

Kapitel 8

Ausblick

Der Entwicklungsprozess an dem Programm ist noch nicht vollständig abgeschlossen. So ist noch eine Vervollständigung der Implementation für die Panoramavorschau der OmniCam vorgesehen. Hierfür steht zum jetzigen Zeitpunkt noch die Bestimmung der Überlappungsbereiche (Blending) und deren Markierung in den Ausgabedaten aus. Des Weiteren ist noch eine horizontale Skalierung der Einzelbilder vorzunehmen, damit sich das Panorama auf einem einzelnen Display darstellen lässt. Um eine weitere Verbesserung der Laufzeit zu erzielen ist angedacht, das zur Zeit genutzten Bubblesort-Verfahren zur Sortierung von Daten durch ein effizienteres Verfahren, wie zum Beispiel das Mergesort-Verfahren, zu ersetzen. Die Interpolation soll zukünftig durch das Verfahren der Bikubischen Interpolation erweitert werden, um somit die Qualität des dargestellten Video- und Bildmaterials zu verbessern.

Literaturverzeichnis

[1] Thomas Thöniß. Abbildungsfehler und Abbildungsleistung optischer Systeme. Technical report, Fachhochschule für angewandte Wissenschaft und Kunst (HAWK), 2004. URL `http://www.winlens.de/fileadmin/user_upload/Dateien/Technical_papers/Abbildungsfehler.pdf`.

[2] H. Nasse B. Hönlinger. Verzeichnung. Technical report, Zeiss, 2009. URL `http://www.zeiss.com/content/dam/Photography/new/pdf/de/cln_archiv/cln33_de_web_special_distortion.pdf`.

[3] Burkhard Neumann. *Bildverarbeitung für Einsteiger*. Springer Verlag Berlin Heidelberg, 2005. ISBN 3-540-21888-2.

[4] Rolf Socher. *Mathematik für Informatiker*. Fachbuchverlag Leipzig, 2011. ISBN 978-3-446-42254-4.

[5] Peter Haberäcker Gudrun Socher Alfred Nischwitz, Max Fischer. *Computergrafik und Bildverarbeitung, Band 2 Bildverarbeitung, 3. Auflage*. Vieweg Teubner Verlag, 2011. ISBN 978-3-8348-1712-9.

[6] Mark James Burge Wilhelm Burger. *Digitale Bildverarbeitung, Eine algorithmische Einführung mit Java, 3. Auflage*. Springer Verlag Berlin Heidelberg, 2015. ISBN 978-3-642-04603-2.

[7] Thomas B. Moeslund. *Introduction to Video and Image Processing, Building Real Systems and Applications*. Springer Verlag London, 2012. ISBN 978-1-4471-2502-0.

[8] Kenneth Dawson-Howe. *A Practical Introduction to Computer Vision with OpenCV*. John Wiley and Sons Ltd, 2014. ISBN 978-1-1188-4845-6.

[9] Alexander Adrian Rosiuta. Minimierung der Stützstellen zur Interpolation in dreidimensionalen Texturen. Technical report, Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart, 2003. URL `http://elib.uni-stuttgart.de/opus/volltexte/2003/1451/`.

[10] Richard Szeliski. Image alignment and stitching:a tutorial. Technical report, Microsoft Research, 2006. URL `http://research.microsoft.com/pubs/75695/Szeliski-FnT06.pdf`.

[11] H.Handels T.Tolxdorff H.-P.Meinzer, T.M.Deserno. *Bildverarbeitung für die Medizin 2009, Algortihmen Systeme Anwendungen*. Springer Verlag Berlin Heidelberg New York, 2009. ISBN 978-3-540-93859-0.

[12] Dr. Jürgen Braun. Vorlesungsskript: Bildverarbeitung für die Medizin. Technical report, Institut für Informatik FU-Berlin, Institut für Medizinische Informatik Charite.

[13] Hazem Elbakry Ebtsam Adel, Mohammed Elmogy. Image stitching based on feature extraction techniques: A survey. Technical report, Mansoura University , Egypt, 2014.

[14] Mike Stephens Chris Harris. A combined corner and edge detector. Technical report, Plessey Research Roke Manor, United Kingdom, 1988.

[15] David G. Lowe Matthew Brown. Automatic panoramic image stitching using invariant features. Technical report, Department of Computer Science, University of British Columbia, 2007.

[16] Allan Jaenicke Philip F. McLaucchlan. Image mosaicing using sequential bundle adjustment. Technical report, School of EE, IT and Mathematics, University of Surrey, 2002.

[17] Christian Weissig. 2020 3d media spatial sound and vision, d6.6 immersive display for home environments. Technical report, Fraunhofer Heinrich-Hertz-Institut, 2012.

[18] *DDR SDRAM SMALL-OUTLINE DIMM*. Micron, 2003. URL `http://www.digchip.com/datasheets/parts/datasheet/297/MT9VDDT6472H-pdf.php`.

[19] *Stratix V Advanced Systems Development Board, Reference Manual*. Altera, 2014. URL `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_svgx_as_dev_board.pdf`.

[20] David T. Wang Bruce Jacob, Spencer W. NG. *Memory Systems, Cache, DRAM, Disk*. Elsevier Inc., 2008. ISBN 978-0-12-379751-3.

[21] *DDR & DDR2 SDRAM Controller IP Cores User's Guide*. Lattice Semiconductor, 2012. URL `http://www.latticesemi.com/~/media/LatticeSemi/Documents/UserManuals/1D/DDRDDR2SDRAMControllerPipelinedUsersGuide.PDF`.