Universität des Saarlandes

Naturwissenschaftlich-Technische Fakultät I
Fachbereich Informatik
Ubiquitous Media Technology Lab

Bachelorarbeit

Toolkit zur verzerrungsfreien Projektion auf beliebigen Oberflächen mittels mobiler Projektoren und Tiefenkameras

Jonas Scheer

s9joscee@stud.uni-saarland.de

10.12.2013

Betreuer:	Markus Löchtefeld
Erstgutachter:	Prof. Dr. Antonio Krüger
Zweitgutachter:	Dr. Jürgen Steimle

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Zudem stimmt die vorliegende Arbeit mit der elektronischen Version überein.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Saarbrücken, den 10.12.2013 _____

Jonas Scheer

Kurzfassung

Projektoren werden immer kleiner und sind heute bereits so klein, dass sie Platz in einem Smartphone finden. Dasselbe gilt für Tiefensensoren, wie man sie beispielsweise in der Microsoft-Kinect findet. Die Projektion mittels Mobilgerät bringt jedoch den Nachteil mit sich, dass die Projektion verzerrt dargestellt wird, sobald Projektionsfläche und Projektor nicht perfekt zueinander ausgerichtet sind. Zudem ist nur das Projizieren auf flache Oberflächen möglich, da sonst ebenfalls die Projektion verzerrt wird.
Diese Arbeit befasst sich mit dem automatischen Entzerren einer Projektion mithilfe eines Tiefensensors, wobei die Oberfläche, auf die projiziert wird, eine völlig beliebige Form haben darf. Ändert sich während dem Projizieren die Oberfläche soll sich die Projektion automatisch anpassen, sodass stets ein unverzerrtes Bild entsteht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abbildungen

Tabellen

1 Einleitung und Motivation

In einem modernen Smartphone finden bereits heute eine Reihe von Sensoren und andere elektronische Bauteile Platz. So beinhaltet das iPhone 5 [3] einen 3-Achsen-Gyrosensor, einen Beschleunigungssensor, einen Annäherungssensor und zwei Kameras. Projektoren werden ebenfalls immer kleiner und können mittlerweile in kommerzielle Smartphones eingebaut werden, wie beispielsweise dem Samsung Galaxy Beam [19], das in Abb. 1.1 zu sehen ist. Ein weiteres Bauteil, das in Zukunft Platz

Abbildung 1.1: Smartphone mit eingebautem Projektor [**?**].

in einem Smartphone finden kann, ist der Tiefensensor, mit dessen Hilfe ein dreidimensionales Bild erstellt werden kann. Während eine gewöhnliche Kamera, wie sie derzeit in Smartphones verbaut wird, ein zweidimensionales Farbbild aufnimmt, wird mit einem Tiefensensor ein sogenanntes Distanzbild aufgenommen. Im Gegensatz zu einem Farbbild, bei dem jedem 2D-Bildpunkt ein Farbwert zugeordnet wird, wird hier jedem 2D-Bildpunkt ein Tiefenwert zugeordnet. Dieser gibt an, wie weit ein Bildpunkt vom Tiefensensor entfernt ist, was in Abb. 1.2 veranschaulicht ist. Objekte, die näher am Tiefensensor liegen und somit auch einen geringeren Tiefenwert haben, sind heller dargestellt. Objekte, die weiter entfernt sind und einen größeren Tiefenwert haben, werden dunkler dargestellt. Ein Tiefensensor ist beispielsweise in der Microsoft-Kinect [12] zu finden, was zeigt das diese auch sehr preiswert hergestellt werden können.

Abbildung 1.2: Tiefenwerte werden mit unterschiedlichen Grautönen dargestellt.

1.1 Anwendungsbeispiele für Smartphones mit Beamer

Durch die Erweiterung des kleinen Smartphonedisplays mit einem Projektor ergeben sich viele neue Anwendungsmöglichkeiten. Statt Bilder oder Filme auf einem kleinen Bildschirm zu betrachten, können diese in einer Größe bis zu 40 Zoll an die Wand geworfen werden. Somit können Bilder oder Filme, anstatt wie bei einem kleinen Smartphonedisplay, von wesentlich mehr Leuten gleichzeitig betrachtet werden. Es wird kein zusätzlicher großer Projektor benötigt, sondern man hat den Projektor ständig bei sich und kann ihn ungestört in der Hosentasche herumtragen. Somit sind Präsentationen zu jeder beliebigen Zeit und an jedem beliebigen Ort möglich. Lediglich eine flache Projektionsfläche wird benötigt. Neuere Anwendungsgebiete für Smartphones mit eingebautem Projektor wären Augmented-Reality Anwendungen. Dies sind Anwendungen, bei denen eine physikalische Umgebung mit digitalen Informationen erweitert wird. Bei der Smartphone-App Wikitude [21] wird beispielsweise das Kamerabild mit zusätzlichen Informationen versehen. Steht man zum Beispiel in einer Stadt, umringt von verschiedenen Sehenswürdigkeiten, erfasst Wikitude anhand der GPS-Daten die Position und mittels Kompass wird überprüft, in welche Richtung man schaut. Die App kann so herausfinden, auf welche Sehenswürdigkeit die Kamera gerichtet ist und blendet den entsprechenden Namen im Kamerabild ein. Gleiches funktioniert auch für Restaurants, Hotels oder andere Orte, wie in Abb. 1.3 zu sehen ist. Hier wird beispielsweise der Name eines Restaurants eingeblendet, das 0,1 km vom Benutzer entfernt in Blickrichtung der Kamera liegt. Durch den Einbau eines Beamers in ein Smartphone, lässt sich eine völlig neue Art von Augmented-Reality Anwendungen erstellen. Hier müssen die zusätzlichen Informationen nicht mehr über den Umweg des Smartphonedisplays eingeblendet werden,

Abbildung 1.3: Orte werden im Kamerabild eingeblendet [**?**]

sondern können direkt auf das Objekt selbst projiziert werden. Will man zum Beispiel während des Einkaufens in einem Geschäft zusätzliche Informationen über eine Produkt erhalten, können diese mittels Smartphone-Projektor auf das Produkt projiziert werden. Somit könnte auf ein Buch dessen Leserbewertung projiziert werden, wie es bei dem Projekt ShelfTorchlight [11] von Löchtefeld et. al. der Fall ist. Ein weiteres Anwendungsbeispiel wäre ein Verkaufsregal mit Wasserflaschen, auf die Ernährungsinformationen projiziert werden. Statt jede Flasche aus dem Regal nehmen zu müssen und nach dem klein gedruckten Magnesium-Gehalt zu suchen, könnte dieser in großen Zahlen auf alle Flaschen projiziert werden, sodass man schnell eine gute Übersicht bekommt. Das System könnte dabei so flexibel sein, dass man statt des Magnesium-Gehalts auch andere Ernährungsinformationen einblenden kann. Dies ist mit statischem Text, der auf die Flasche gedruckt ist nicht möglich, da jeder Benutzer andere Informationen bevorzugt. Augmented-Reality Anwendungen die einen Beamer benutzen, benötigen somit oft beliebig geformte Oberflächen statt flache Oberflächen. Neben Augmented-Reality Anwendungen, könnte ein Smartphone mit eingebautem Beamer auch einen stationären Desktop-PC ersetzen. Der Beamer ersetzt dabei den Monitor und eine kabellose Tastatur sowie Maus, die an das Smartphone gekoppelt sind, dienen als Eingabegeräte. Alternativ kann ein Smartphone auch mit zwei Projektoren ausgestattet werden. Einer wird dabei wie gewohnt als Desktop-Display verwendet, ein Zweiter projiziert eine Tastatur auf den Tisch, wie es bei dem Konzept-Smartphone Mozilla Seabird angedacht ist. Somit kann man seinen Daten und seine gewohnte Desktop-Umgebung überall hin mitnehmen.

1.2 Problem von mobilen Projektoren

Mobile Projektoren bringen das Problem mit sich, dass sie möglichst genau zur Oberfläche, auf die projiziert werden soll, ausgerichtet werden müssen. Ist dies nicht der Fall, wird die Projektion verzerrt und nimmt eine ungewollte Form an. Die optische Achse eines Projektors ist eine durch die Projektorlinse gehende, gedachte Linie, entlang der sich das Licht ausbreitet, das vom Projektor ausgesandt wird. Diese muss orthogonal zur Projektionsfläche stehen, damit eine Projektion unverzerrt dargestellt werden kann. Ist dies nicht der Fall und handelt es sich um eine flache Projektionsfläche, so stellt sich ein sogenannter Keystone-Effekt ein. Dabei werden parallel liegende Bildkanten unterschiedlich groß dargestellt, wie Abb. 1.4 (oben) zeigt. Hier ist die Oberkante des Bildes breiter, als dessen Unterkante. Dem Keystone-Effekt kann allerdings auch entgegen gewirkt werden, indem man das Ursprungsbild so verzerrt, dass die letztendliche Projektion unverzerrt dargestellt wird. Dieser Vorgang nennt sich Keystone-Korrektur und ist bereits heute in vielen Projektoren verbaut. Die Auswirkungen der Keystone-Korrektur sind in Abb. 1.4 (unten) dargestellt. Im Vergleich zum oberen Bild sind beide Bildkanten gleich breit. Der Nachteil vieler

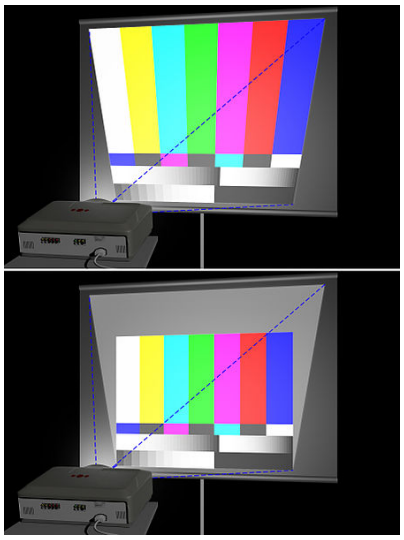

Abbildung 1.4: Beispiel von Keystone-Effekt und Keystone-Korrektur. [**?**]

System mit Keystone-Korrektur jedoch ist, dass diese von Hand eingestellt werden muss und somit etwas Zeit in Anspruch nimmt. Bei mobilen Projektoren kann es oft der Fall sein, dass sich die Position und Ausrichtung des Projektors, während die Anwendung läuft, ständig verändert. Bei Spotlight Navigation von Rapp [17] wird beispielsweise durch Schwenken des Projektors nach links, rechts, oben oder unten durch eine Karte navigiert. Ähnlich wie beim Bewegen einer Taschenlampe wird dabei immer ein anderer Teil der Karte sichtbar. Ein manuelles Anpassen der Projektion mittels Keystone-Korrektur ist somit nicht in allen Anwendungen

sinnvoll. Eine Keystone-Korrektur kann durch Anwenden einer Zentralprojektion erreicht werden. Dabei wird ein Homomorphismus auf das zu projizierende Bild angewandt, was letztendlich zu einer Matrixmultiplikation führt. Je nachdem wie stark die Keystone-Korrektur in horizontaler und vertikaler Richtung angewandt werden soll, ändern sich die Einträge dieser Matrix. Sollte es sich allerdings nicht um eine flache Projektionsfläche handeln, bringt eine Keystone-Korrektur nicht den gewünschten Effekt. Um beispielsweise auf eine kugelförmige Oberfläche zu projiziert muss eine wesentlich kompliziertere Entzerrungsprozedur angewandt werden.

1.3 Aufgabenstellung und Ziel der Arbeit

Ziel der Arbeit ist es ein Toolkit, bestehend aus einer Software-Hardware Kombination, zu entwickeln, mit dem eine Projektionsentzerrung für beliebige Oberflächen möglichst einfach erreicht werden kann. Dieses Toolkit soll auch für andere Entwickler so einfach wie möglich und mit minimalem Mehraufwand zu verwenden sein. Die Projektionsentzerrung soll idealerweise in Echtzeit ablaufen, sodass das System auf Änderungen an der Projektionsumgebung so schnell wie möglich reagieren kann. Für den gesamten Entzerrungsprozess soll low-cost Hardware verwendet werden und es soll untersucht werden, wie effizient eine Projektionsentzerrung damit gelingt.

1.4 Aufbau dieser Arbeit

In Kapitel 2 werden einige verwandte Arbeiten vorgestellt, die sich mit dem Thema automatische- oder halb automatisch Projektionsentzerrung befassen. Dabei werden einerseits flache Projektionsflächen betrachtet, auf denen eine Projektion unabhängig vom Projektionswinkel unverzerrt dargestellt wird, aber auch beliebige Oberflächen, auf denen eine sehr vereinfachte Form der Projektionsentzerrung stattfindet. In allen Anwendungen wird die Projektion automatisch angepasst, sobald sich die Position des Projektors oder die Projektionsfläche verändert. Zudem wird auf die Unterschiede der einzelnen Anwendungen zu dieser Arbeit eingegangen. Kapitel 3 befasst sich mit den mathematischen Grundlagen zur Projektionsentzerrung. Es wird erklärt, wie eine verzerrte Projektion entsteht, wie man dem entgegenwirken kann und welche mathematischen Modelle dazu verwenden werden können. In Kapitel 4 wird schließlich die Implementierung vorgestellt. Es wird auf die Hard- und Software eingegangen die verwendet wird und erklärt wie die in Kapitel 3 besprochenen mathematischen Modelle effizient implementiert werden können. Kapitel 5 fasst alle Ergebnisse noch einmal zusammen und gibt einen Ausblick, was mit der erstellten Software in Zukunft möglich wäre.

2 Verwandte Arbeiten

In diesem Kapitel werden einige verwandte Arbeiten vorgestellt, die sich mit dem Thema Projektionsentzerrung befassen und eine wichtige Grundlage für diese Arbeit bilden. Es werden zunächst die einzelnen Projekte vorgestellt und ihre Vor- und Nachteile besprochen. Anschließend werden die Projekte sowie diese Arbeit in einer Übersichtstabelle verglichen, um ihre Vor- und Nachteile hervorzuheben. Speziell werden die einzelnen arbeiten nach folgenden Kriterien kategorisiert:

- **beliebige Oberflächen**
  Es wird unterschieden, ob nur auf flache Oberflächen projiziert werden kann, oder eine Projektionsfläche eine beliebige Form haben darf.

- **Echtzeitentzerrung**
  Die Oberfläche auf die Projiziert wird darf sich während des Projektionsvorgangs verändern wobei die Projektion in Echtzeit angepasst wird.

- **oberflächenadaptiv**
  Einige der vorgestellten Arbeiten können zwar auf beliebige Oberflächen projizieren und die Projektion in Echtzeit anpassen, jedoch müssen hier zuerst alle Oberflächen, die für die Projektion zur Verfügung stehen, dem System beigebracht werden. Es wird somit ein Initialisierungsschritt benötigt, bei dem die Gesamte Umgebung auf die im weiteren Verlauf projiziert werden soll, dem System bekannt gemacht wird. Dieser Schritt wird bei Anwendungen, die das Kriterium oberflächenadaptiv erfüllen nicht benötigt. Es kann ohne weiteres eine Echtzeitentzerrung auf beliebige sich verändernden Flächen durchgeführt werden.

- **Mobilität**
  Mobile Systeme sind nicht an einen speziellen Raum bzw. an eine spezielle Umgebung gebunden und werden nicht fest installiert.

2.1 Automatic Projector Calibration with Embedded Light Sensors

Um eine Projektion auf einer flachen Oberfläche zu entzerren, kann die bereits erwähnte Keystone-Korrektur bzw. ein Homomorphismus angewandt werden. Lee et. al. [10] haben dazu 4 Lichtsensoren auf einer Projektionsfläche befestigt, sodass herausgefunden werden kann, wie die Projektionsfläche zum Projektor hin ausgerichtet

ist. Zunächst wird eine Sequenz von binären Kalibrierungsmuster projiziert, wobei jeder der 4 Lichtsensoren einen Teil des Kalibrierungsmusters erfassen können muss. Die Muster bestehen einmal aus horizontalen und einmal aus vertikalen schwarzen und weißen Streifen. Die Anfangsbilder enthalten lediglich einen breiten schwarzen und weißen Streifen, diese jedoch im Verlauf der Sequenz vervielfacht werden. Die Endbilder der Sequenz bestehen somit aus vielen sehr dünnen schwarzen und weißen Streifen. Nach Projektion der Kalibrierungsmuster-Sequenz kann die Position jedes Lichtsensors im Raum bestimmt werden. Da diese zudem an den Ecken der Projektionsfläche befestigt sind, kann herausgefunden werden, welche Position im Raum die Projektionsfläche hat, wie diese zum Projektor hin ausgerichtet ist und wie groß sie ist. Dies führt letztendlich zu einem Homomorphismus, der beschreibt, wie ein Bild verzerrt werden muss, damit die Projektion unverzerrt dargestellt wird. Dies funktioniert auch noch, wenn der Projektor auf dem Kopf steht, oder die Projektion über Spiegel umgeleitet wird. Lediglich alle Lichtsensoren müssen im Verlauf der Kalibrierung im Lichtkegel des Projektors liegen. In Abb. 2.1 oben ist zu sehen, wie eine Projektion unverändert dargestellt wird. In Abb. 2.1 unten wird die Projektion verändert, sodass nur die, mit Lichtsensoren versehene, Projektionsfläche genutzt wird. Laut Lee et. al. genügen für einen Projektor mit einer Auflösung von 1024×768 Pixel eine Sequenz von 20 Kalibrierungsmustern. Die Kalibrierung ist dadurch sehr schnell und in wenigen Sekunden durchführbar, muss allerdings nach jedem Ändern der Position der Projektionsfläche erneut durchgeführt werden.

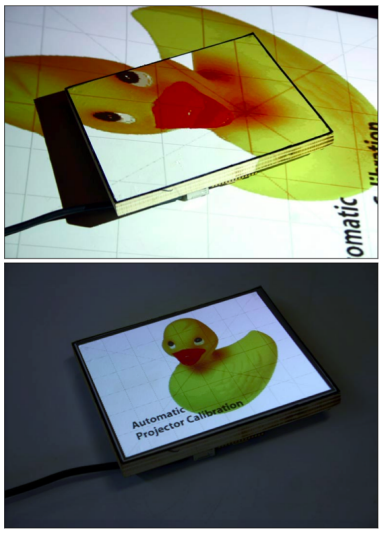

Abbildung 2.1: Lichtsensoren zur Projektionsentzerrung [**?**].

Die Vorteile eines solchen Systems sind die Projektion auf beliebig ausgerichtete, flache Oberflächen und die automatische Anpassung der Projektion nach Positionsänderung des Projektors oder der Projektionsoberfläche. Lediglich eine kurze Ka-

librierung, die zudem automatisch durchgeführt wird, ist nötig. Der Kalibrierungsvorgang muss jedoch auch als Nachteil aufgeführt werden, da nach jedem Abändern der Position der Projektionsfläche bzw. des Projektors eine erneute Kalibrierung durchgeführt werden muss. Die Kalibrierung nimmt einige Sekunden in Anspruch und ist somit nicht für mobile Projektoren nutzbar. Ein weiterer Nachteil ist die Limitierung auf flache Projektionsflächen. Das vorgestellte System bietet daher keine Echtzeitentzerrung, keine Projektion auf beliebige Oberflächen und somit auch keine Oberflächenadaptivität. Es ist auch nur beschränkt mobil, da es an eine Umgebung, die mit Lichtsensoren ausgestattet ist, gebunden ist und der Projektor nur in dieser bewegt werden darf.

2.2 Semi-Automatic Realtime Calibration

Wie im obigen Projekt von Lee et. al. führen auch Dao et. al. [5] eine Projektionsentzerrung auf flachen Oberflächen durch. Dabei wird jedoch nicht auf Lichtsensoren zurück gegriffen, sondern die Daten eines Accelerometers und eines Kompasses genutzt. Mithilfe dieser Daten kann die Ausrichtung des Projektors bezüglich der Projektionsfläche bestimmt werden und das zu projizierende Bild wird so verändert, dass die Projektion unverzerrt erscheint. Zunächst wird ein kurzer Kalibrierungsprozess durchgeführt. Dabei wird der Projektor so zur Projektionsfläche ausgerichtet, dass die Projektion unverzerrt erscheint. Durch die Betätigung eines Knopfes werden die Daten des Accelerometers und des Kompasses gespeichert. Beim verändern der Position des Projektors werden die gespeicherten Daten mit den Daten, die für die neue Position geliefert werden, verglichen. Somit können pitch- und yaw-Winkel des Projektors bezüglich der Projektionsfläche berechnet werden, und das Quellbild entsprechend verändert werden. In Abb. 2.2 (a) und (b) wird die beschriebene Technik genutzt und die Projektion wird unverzerrt dargestellt. Obwohl der Projektor in Abb. 2.2 (b) in einem sehr steilen Winkel zur Projektionsfläche gehalten wird, ist die Projektion dennoch nicht verzerrt. Abb. 2.2 (c) dagegen zeigt die Verzerrung, die entsteht, wenn die vorgestellte Technik nicht genutzt wird. Um Sensorrauschen und das minimale Wackeln der Hand auszugleichen, werden 10 aufeinanderfolgende Sensorwerte auf einmal betrachtet und deren Mittelwerte berechnet.
Der Vorteil dieser Methode besteht darin, dass die Kalibrierung nur ein einziges mal durchgeführt werden muss. Verändert sich die Position oder die Ausrichtung des Projektors wird dies in Echtzeit von Accelerometer und Kompass wahrgenommen, sodass dem entgegen gewirkt werden kann. Während in [10] die Projektionsfläche mit Lichtsensoren versehen wird, werden hier keine weiteren Anforderungen an die Projektionsfläche gestellt. Diese muss lediglich flach sein. Allerdings funktioniert das System nicht mehr, wenn die Position oder Ausrichtung der Projektionsfläche verändert wird. Das System ist also an eine feste, unveränderliche Projektionsfläche gebunden. Die Begrenzung auf flache Projektionsoberflächen ist ein weiterer Nachteil und schränkt die Anzahl der Anwendungen ein, für die dieses System von nutzen wäre. Da ein Kalibrierungsvorgang benötigt wird und nur auf flache Oberflächen

Abbildung 2.2: Entzerrung mit Accelerometer und Kompass [**?**].

projiziert werden kann, ist das System nicht oberflächenadaptiv. Allerdings handelt es sich um ein mobiles System, da es nicht fest installiert werden muss und der Projektor frei bewegt werden kann. Aufgrund der Eigenschaften des Bewegungssensors, können auch nur langsame Bewegungen durchgeführt werden. Sollte eine Bewegungsänderung nicht korrekt erfasst werden, oder sollte Sensorrauschen diese verhindern, so wird die Projektion nicht korrekt entzerrt, was sich nach einiger Zeit aufsummiert.

2.3 Everywhere-Display

Bei dem Everywhere-Display [15] von Pinhanez wird nicht nur die Fläche vor dem Beamer als Projektionsfläche genutzt, sondern gleich ein ganzer Raum. Mithilfe eines beweglichen Spiegelsystems kann die Projektion auf alle Wände, den Boden sowie auf Tische, Schränke und sonstige Möbel gelenkt werden, wie in Abb. 2.3 gezeigt wird. Zudem dient die Projektion als Touchscreen, sodass Benutzereingaben getätigt werden können und Wände, Boden und Möbel werden zu Eingabegeräten umfunktioniert. Damit eine Projektion unverzerrt durch das Everywhere-Display dargestellt werden kann, wird ein simples, virtuelles 3D-Modell des Raumes erstellt, mit dessen Hilfe die Projektion entzerrt wird. Soll auf eine bestimmte Fläche im physikalischen Raum projiziert werden, wird dieselbe Fläche im virtuellen Raum mit einer virtuellen Kamera betrachtet. Diese Kamera nimmt dieselbe Position und Ausrichtung im virtuellen Raum ein, wie der Projektor im physikalischen Raum einnimmt. Zudem werden der Kamera dieselben Linsenparameter übergeben, wie der Projektor hat. Das Kamerabild, das so entsteht, ist das inverse Bild zur verzerrten Projektion. Wird dieses in den physikalischen Raum projiziert, so erscheint die Projektion unverzerrt. Das virtuelle 3D-Modell erhält man, indem man die Größe, Rotation und Position für jede zu nutzende Projektionsfläche von Hand anpasst. Soll beispielsweise auf einen Tisch projiziert werden, wird das zu projizierende Bild zunächst mithilfe der Maus verkleinert, indem man es an den Ecken staucht. Danach wird es verschoben, bis die Projektion auf dem Tisch an der gewünschten Stelle ist und zuletzt deren Ro-

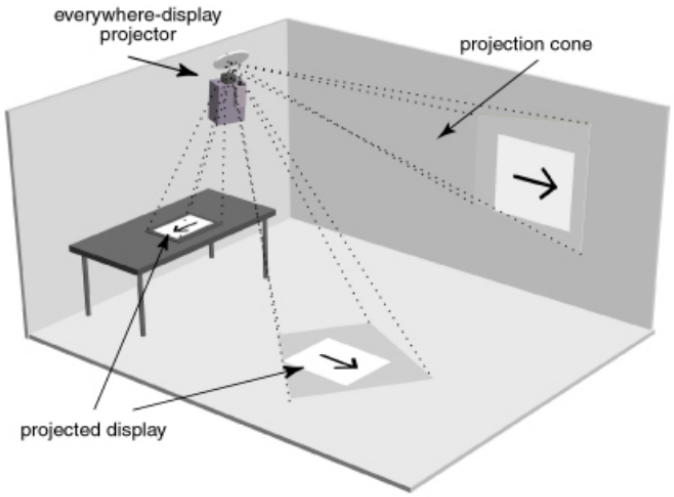

Abbildung 2.3: Der ganze Raum kann als Display genutzt werden [**?**].

tationswinkel angepasst. Diese 3 Anpassungsvorgänge werden mehrmals wiederholt, bis die Projektion die gewünschte Form und Position hat. Da dies für jede Oberfläche durchgeführt werden soll, die von dem Everywhere-Display genutzt werden soll, ist dies ein sehr langwieriger Prozess.
Dies ist auch der größte Nachteil des Systems. Laut Pinhanez dauert es ca. 10-20 Minuten, bis eine einzelne Oberfläche nutzbar ist. Zudem können nur flache Projektionsflächen benutzt werden, was jedoch in einer zukünftigen Version auf beliebige Oberflächen erweitert werden kann. Ein weiterer Nachteil ist, dass es vorkommen kann, dass ein Benutzer im Lichtkegel der Projektion steht. Bei dem vorherigen Projekt kann dies nicht passieren, da dort der Benutzer den Projektor in der Hand hält. Weiterhin handelt es sich um sein stationäres System. Ändert sich der Raum, indem beispielsweise ein Schrank verschoben wird, muss das System dementsprechend neu kalibriert werden, wodurch keine Oberflächenadaptivität gegeben ist. Gleiches gilt, wenn das System in einen neuen Raum integriert werden soll. Der Vorteil des Systems ist, dass es nach Kalibrierung und Erstellung des 3D-Modells sehr flexibel eingesetzt werden kann. Der ganze Raum ist als interaktives Display nutzbar und die Projektion wird in Echtzeit entzerrt. Während die vorherigen Systeme nur einen kleinen Bereich als Projektionsfläche genutzt haben, wird hier die Projektion als interaktive virtuelle Umgebung nahtlos in die physikalische Umgebung integriert.

2.4 Illuminating-Clay

Illuminating-Clay [16] von Piper et. al. ist ein System, mit dem Landschaftsmodelle manipuliert werden können. Dabei wird eine frei verformbare Tonoberfläche durch

einen Projektor angestrahlt, der je nach Topografie unterschiedliche Farben auf das Tonmodelle projiziert. Der Ton wird also so verformt, dass Berge und Täler entstehen, auf die unterschiedliche Farben projiziert werden. Dabei reagiert das System in Echtzeit auf Änderungen an der Ton-Topografie. Wird in die Mitte eines Ton-Berges ein Loch gedrückt, so ändert sich die Farbe in der Mitte des Berges entsprechend der Farbe für tiefer gelegene Landschaften. Damit das System Informationen über die derzeitige Ton-Landschaft erhält wird diese mithilfe eines Laserscanners erfasst. Abb. 2.4 zeigt, wie Illuminating-Clay benutzt werden kann. Hoch gelegene Landschaften werden beispielsweise hellblau angestrahlt, tiefer gelegene Landschaften werden gelb oder rot eingefärbt.

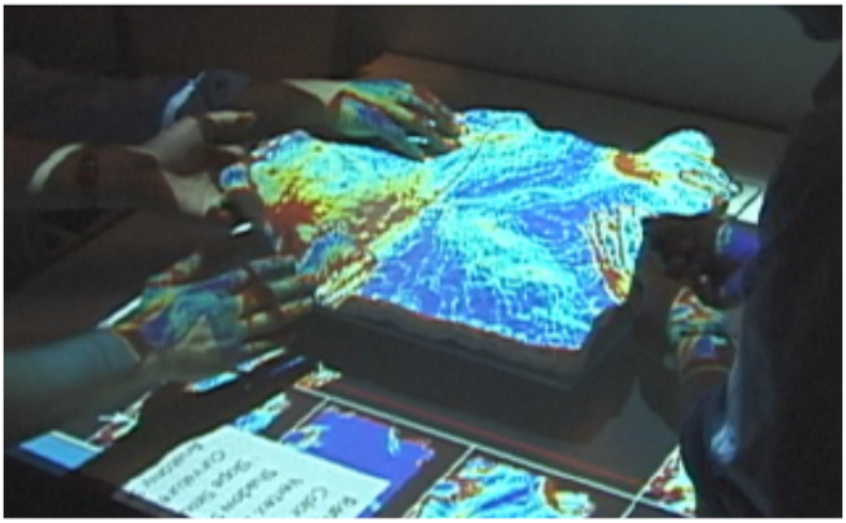

Abbildung 2.4: Illuminating-Clay im Einsatz [**?**].

Der Vorteil dieses Systemes ist eine vereinfachte Art der Projektionsentzerrung. Es wird keine komplexe Grafik entzerrt, sondern lediglich Pixel, entsprechend der Topografie des Tonmodells, eingefärbt. Dennoch ist das System nicht an flache Oberflächen gebunden und passt die Projektion in Echtzeit bei Veränderungen an. Es ist auch oberflächenadaptiv, da die Tonlandschaft beliebig abgeändert werden darf, ohne dass ein Initialisierungsprozess benötigt wird. Der Nachteil des Systemes ist, dass es sich um ein stationäres System handelt, bei dem Laserscanner und Projektor perfekt zueinander ausgerichtet sein müssen.

2.5 RFIG Lamps: Interacting with a Self-Describing World via Photosensing Wireless Tags and Projectors

Bei RFIG Lamps [18] von Raskar et. al. werden gleich mehrere Methoden zur Projektionsentzerrung vorgestellt. Eine ist dabei vergleichbar mit [10] und benutzt Licht-

sensoren zur Projektionsentzerrung. Zwei andere Methoden sind eher zur Projektionsstabilisierung gedacht, um beispielsweise das Wackeln der Hand oder minimale Bewegungen auszugleichen.

2.5.1 Formerkennung mit Lichtsensoren

Raskar et. al. benutzen ein tragbares Gerät, das neben einem Projektor und einer Kamera einen Inertialsensor, vier fest angebrachte Laserpointer sowie eine Infrarot-Einheit zur Kommunikation besitzt. Weiterhin werden elektronische Tags benutzt, die ebenfalls über eine Infrarot-Einheit zur Kommunikation verfügen und einen Lichtsensor, einen Mikroprozessor sowie Speicher besitzen.

Die Tags können nun an ein Objekt, beispielsweise an eine Box angebracht werden, und durch Projizieren einer binären Kalibrierungsmuster-Sequenz, wie in [10], kann die Position und Ausrichtung des Projektors bestimmt werden. Während bei [10] nur flache Oberflächen genutzt werden konnten, sind hier auch komplexere Oberflächen möglich. Dazu müssen die elektronischen Tags die Form des Objektes, auf dem sie angebracht sind, sowie ihre Position auf dem Objekt kennen. Das System ist darauf ausgerichtet, dass die Tags an den Ecken eines Objektes angebracht werden. Für eine quaderförmige Box werden somit 8 Tags an den jeweiligen Ecken angebracht. Mithilfe einer ersten Kalibrierungsmuster-Sequenz und einem Structure-From-Motion-Verfahren, ähnlich wie in [8] beschrieben, werden die Positionen der Tags auf dem Objekt bestimmt, die Tags zu einem 3D-Mesh verbunden und somit die Form des Objektes berechnet. Dieser Initialisierungsschritt muss für jede Oberfläche, auf die projiziert werden soll, einmalig durchgeführt werden. Die ermittelten Tag-Positionen und die abgeleitete Objektform werden anschließend in den Tags gespeichert und können auf Abruf an das System, an das der Projektor angeschlossen ist, übertragen werden. Dazu werden die Tags zunächst aktiviert, indem sie vom Projektor beleuchtet werden. Die Lichtsensoren der Tags registrieren dies, woraufhin sie die gespeicherten Daten, über die Infrarotschnittstelle, an das System senden. Mithilfe einer zweiten Kalibrierungsmuster-Sequenz kann nun die Position und die Ausrichtung des Projektors, unter Zuhilfenahme der Tag-Informationen, berechnet werden. Hat ein Objekt, über dessen Tags, dem System seine Form und die Tag-Positionen mitgeteilt, und ist die Ausrichtung und die Position des Projektors bekannt, wird ein zu projizierendes Bild so verzerrt, dass die Projektion unverzerrt auf dem Objekt dargestellt wird.
Der Vorteil des Systems ist, die Fähigkeit auf komplexe Oberflächen projizieren zu können, allerdings ist dafür Zusatzhardware in Form von elektronischen Tags notwendig. Da das System aus allen Tag-Positionen ein 3D-Mesh generiert, kann nicht auf gekrümmte Oberflächen projiziert werden. Hierzu müsste man sehr viele Tags an der Oberfläche anbringen, was aufgrund deren Größe nicht möglich ist. Wie in [10] ist dieses System nicht für Echtzeitentzerrung nutzbar, da das Entzerren mithilfe der Kalibrierungsmuster-Sequenz einige Sekunden in Anspruch nimmt. Somit bietet das

System auch keine Oberflächenadaptivität, jedoch handelt es sich um ein mobiles System.

2.5.2 Projektionsstabilisierung für mobile Projektoren

Bei den Projektionsstabilisierung Methoden wird eine, von Raskar et. al. sogenannte, *absolute Stabilisation* und eine *quasi Stabilisation* vorgestellt.

Bei der absoluten Stabilisation werden vier (oder mehr), physikalische Markierungspunkten auf einer flachen Oberfläche befestigt, beispielsweise in rechteckiger Anordnung. In der Ausgangsstellung befindet sich die Projektion innerhalb dieses Rechtecks. Wird nun der Projektor bewegt, so wird das zu projizierende Bild so verändert, dass die Projektion ihre Position und Form innerhalb der Markierungspunkte beibehält. Die Bewegungen des Projektors werden also herausgefiltert.
Der Vorteil dieser Methode ist, dass eine Projektion, unabhängig von der Position und Ausrichtung des Projektors, immer ihre Position und Form beibehält und in Echtzeit angepasst werden kann. Der Nachteil jedoch ist, dass nur auf flache Oberflächen projiziert werden kann und die Oberfläche mit Markierungspunkten vordefiniert werden muss.

Bei der quasi Stabilisation werden die vier Laserpointer genutzt. Da die Laserpunkte sehr hell sind, können diese im Kamerabild vom System erkannt werden. Weiterhin wird die Laserpointer-Konstruktion und die Kamera, ähnlich wie in [8] beschrieben, kalibriert. So lässt sich anhand der Position der Laserpunkte im Kamerabild sowie mithilfe des Inertialsensors, über den die Rotation um die Normale der Projektionsebenen berechnet wird, die Position und die Ausrichtung des Projektors bestimmen und die Projektion entsprechend anpassen.
Als Vorteil dieser Methode kann die Anpassung der Projektion in Echtzeit genannt werden. Allerdings kann hier wie bei der absoluten Stabilisation nur auf flache Oberflächen projiziert werden. Zudem muss auf Zusatz-Hardware in Form von Laserpointern und Inertialsensor zurückgegriffen werden. Quasi Stabilisation und absolute Stabilisation sind mobile Systeme. Da sie jedoch nur auf flache Oberflächen projizieren können, sind sie auch nicht oberflächenadaptiv.

2.6 Flexpad: Highly Flexible Bending Interactions for Projected Handheld Displays

Steimle et. al. haben mit FlexPad [20] ein System erstellt, das auf beliebige Oberflächen projizieren kann und zusätzlich auf Änderungen an der Oberfläche in Echtzeit reagiert. Abb. 2.5 zeigt Beispiele der FlexPad Anwendung, bei der auf ein gekrümmtes Blatt Papier projiziert wird. Das FlexPad System besteht dabei aus einem fest montierten Full HD Projektor und einer Kinect, die beide so kalibriert sind, dass sie im selben dreidimensionalen Koordinatensystem arbeiten. Zu einem Kinect-Pixel

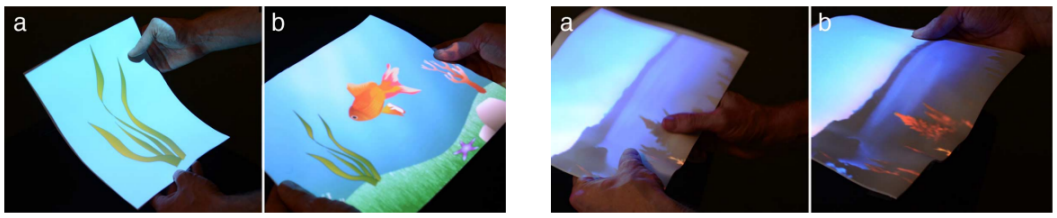

Abbildung 2.5: entzerrte Projektion auf gekrümmtem Papier [**?**].

kann somit das zugehörige Projektor-Pixel gefunden werden. Zudem beinhaltet das System einen Intel i7 Prozessor, eine AMD Radeon 6800 sowie 8GB RAM und beinhaltet damit leistungsstarke Hardware. Das System erkennt automatisch verformbare Oberflächen, beispielsweise ein Blatt Papier und kann gewünschte Inhalte auf diese projizieren. Die Oberflächen können sich zudem frei im Lichtkegel des Projektors bewegen, wobei sie automatisch getrackt werden und die Projektion entsprechend zur Oberfläche mitwandert. Damit das System eine frei verformbare Oberfläche erkennt, wird ein mathematisches *Verformungsmodell* erstellt, von dem, bei Ausführen der Anwendung, nur die entsprechenden Parameter gefunden werden müssen. Das Modell nimmt an, dass die verformbaren Oberflächen eine feste Größe haben. Steimle et. al. nutzen hier die Größe eines Din A4 Blattes. Intern wird eine Oberfläche mit 25×25 Knotenpunkten beschrieben. Weiterhin kann eine verformbare Oberfläche nur aus einer Kombination aus 8 Basisverformungen, einem Startpunkt dieser Basisverformungen sowie 3 Rotationen und 3 Translationen bestehen. Eine Oberfläche wird somit durch einen 15-dimensionalen Vektor beschrieben, dessen Einträge gefunden werden müssen. Soll nun bestimmt werden, welches Modell, mit den entsprechenden Parametern, für eine gekrümmtes Din A4 Blatt zutrifft, wird ein *Analyse durch Synthese Verfahren* bzw. *direktes Tracking Verfahren* auf das Tiefenbild der Kinect angewandt. Dabei werden Modelle mit verschiedenen Parametern erzeugt und für alle 25×25 Knotenpunkten eines Modells deren Tiefenwerte bestimmt. Zusätzlich werden für die Knotenpunkte diejenigen Pixel im Kinect-Tiefenbild bestimmt, auf welche die Knotenpunkte abgebildet werden würden. Die 25×25 Tiefenwerte dieser Kinect-Pixel werden nun mit den berechneten 25×25 Tiefenwerte des Modells vergleichen. Schließlich wird das Modell genommen, deren Tiefenwerte der Knotenpunkte im Mittel die geringste Abweichung bezüglich des Kinect-Tiefenbildes haben und die Projektion wird entsprechend angepasst.
Der Vorteil von FlexPad ist die Echtzeitentzerrung sowie das projizieren auf fast beliebige Oberflächen. Mit den oben beschriebenen 15 Parameter lassen sich nicht alle möglichen Oberflächen darstellen, jedoch bereits eine Vielzahl an gekrümmten Oberflächen. Zusätzlich können durch die Beschränkung auf 25×25 Knotenpunkte keine detailreichen Oberflächen approximiert werden. Da kein Initialisierungsprozess benötigt wird und ohne Weiteres auf eine fast beliebige Oberflächen in Echtzeit projiziert werden kann, ist das System beschränkt oberflächenadaptiv. Es handelt sich

allerdings nicht um ein mobiles System, da FlexPad fest in einem Raum installiert ist.

2.7 Interactive Environment-Aware Handheld Projectors for Pervasive Computing Spaces

Molyneaux et. al. [13] stelle 2 weitere Ansätze zur Projektionsentzerrung vor. Dabei wird einmal eine bestimmte Infrastruktur vorausgesetzt und einmal vollständig auf eine Infrastruktur verzichtet. Beide Methoden nutzen KinectFusion, das im folgenden erläutert wird.

2.7.1 KinectFusion

Bei dem Projekt KinectFusion [9] von Izadi et. al. wird eine physikalische Umgebung mithilfe der Kinect in Echtzeit gescannt und ein virtuelles 3D-Abbild davon erstellt. Zudem können zusätzliche, virtuelle Objekte in das erstellte 3D-Abbild eingebracht werden. Abb. 2.6 zeigt ein Beispiel einer gescannten 3D-Umgebung mit zusätzlichen virtuellen, gelben Kugeln. Um ein solches 3D-Modell zu erhalten, wird

Abbildung 2.6: gescannte Umgebung mit virtuellen Objekten [**?**].

nach und nach die Umgebung mithilfe der Kinect gescannt, sodass mit jedem neuen Tiefenbild das Modell verfeinert wird. Dies geschieht in 3 Schritten. Zunächst werden die Tiefendaten eines Frames in 3D-Koordinaten umgewandelt. Dabei liegt ein Koordinatensystem, welches die Kinect als Ursprung hat, zugrunde. Anschließend wird anhand der berechneten 3D-Koordinaten und dem *Iterative Closest Point Algorithmus* (ICP) die Lage und Ausrichtung der Kinect bestimmt. Intern besteht das

virtuelle 3D-Abbild aus einem *Voxel-Gitter*, das mittels *Volumetric integration* [4] in einem nächsten Schritt erzeugt wird. Um die erstelle virtuelle 3D-Welt schließlich auf einem Ausgabegerät darzustellen, wird ein Ausschnitt der Welt mithilfe von Raycasting erstellt, welcher letztendlich gerendert werden kann. Der Nachteil, des auf diese Weise erstellten 3D-Modells ist, dass es nur für statische Szenen, oder Szenen die sich nur bis zu einem gewissen Grad verändern, benutzt werden kann. Sollte sich eine Frame einer Szene von seinem vorherigen Frames grundlegend unterscheiden, kann der ICP-Algorithmus die Lage und Ausrichtung der Kinect nicht mehr genau bestimmen und bei dem bestehende interne Voxel-Gitter werden Tiefendaten von 2 verschieden Szenen zusammengemischt. Dadurch wird das bisher erstellte 3D-Abbild zerstört und auch alle folgenden Tiefendaten werden falsch eingeordnet. KinectFusion ist jedoch in der Lage, die interne 3D-Umgebung anzupassen, wenn sich nur ein kleiner Teil einer Szene ändert. So kann beispielsweise die Kanne in Abb. 2.6 bewegt werden, wobei das 3D-Modell entsprechend angepasst wird. KinectFusion erzeugt somit effizient und Echtzeit ein virtuelles Abbild einer physikalische Umgebung. Diese Technik wird beispielsweise in Abschnitt 2.7 zur Projektionsentzerrung genutzt.

2.7.2 RoomProjector

Das Projekt RoomProjector nutzt eine Infrastruktur, um Informationen über die Projektionsumgebung zu erhalten. Es befasst sich ähnlich wie das Everywhere-Display mit dem Scannen eines ganzen Raumes, um darin an jeder möglichen Stelle eine Projektion zu erzeugen. Dabei werden 4 Kinect-Einheiten in jede Ecke an der Decke eines Raumes angebracht, um ein virtuelles Abbild des Raumes zu erzeugen. Dazu wird zunächst der Raum ohne Benutzer gescannt und daraus ein Hintergrund-Abbild erzeugt. Anschießend können Benutzer den Raum betreten. Dieser wird erneut gescannt, wobei mithilfe des Hintergrund-Abbildes die Position eines Benutzers bestimmt werden kann. Ein Beispiel eines Hintergrund-Abbildes ist in Abb. 2.7 (1) zu sehen. In (2) und (3) werden verschiedene Oberflächen-Rekonstruktionsalgorithmen angewandt, bei denen Hintergrundrauschen entfernt wird und aus den einzelnen Tiefendaten ein geglättetes 3D-Abbild erzeugt wird. Abb. 2.7 (4)-(6) zeigen, wie ein Benutzer im Raum geortet wird und (6) blendet zudem den Lichtkegel des Projektors ein. Um die Position eines Projektors herauszufinden, wird dieser mit stark reflektierendem Klebeband versehen, welches die Infrarotstrahlen der Kinect verstärkt. Dies ist auch im Tiefenbild der Kinect zu erkennen, sodass mit Hilfe von 3 Tiefenbildern von 3 unterschiedlichen Kinect-Einheiten und Triangulierung die Position des Projektors im Raum bestimmt werden kann. Mithilfe eines Inertialsensors (IMU) wird zudem die Ausrichtung des Projektors im Raum erfasst und roll-pitch-yaw Winkel des Projektors bestimmt.
Mithilfe der Ausrichtung und der Position des Projektors sowie den Umgebungsdaten, die durch die Kinect Einheiten geliefert werden, kann eine Projektion in Echtzeit angepasst werden, sodass sie unverzerrt dargestellt wird. Da das System allerdings nur das Hintergrunde-Abbild zur Projektionsentzerrung nutzt, kann das es ähnlich

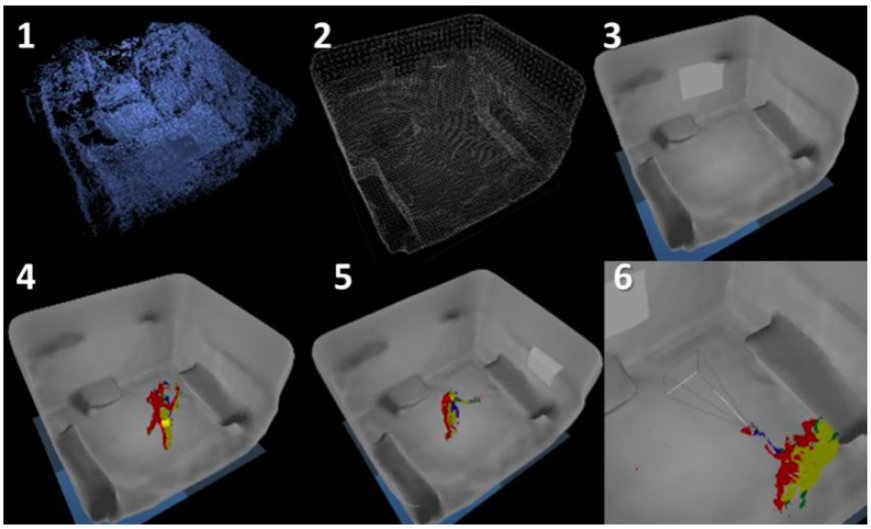

Abbildung 2.7: erstelltes 3D Abbild [**?**].

wie bei dem Everywhere-Display Projekt nicht auf Änderungen des Raumes reagieren. Das System ist somit nicht oberflächenadaptiv und aufgrund der benötigten Infrastruktur auch kein mobiles System.

2.7.3 SLAMProjector

Das zweite Projekt, SLAMProjector, versucht diesen Mängeln entgegenzuwirken. Hier wird vollkommen auf eine Infrastruktur verzichtet und ein sogenanntes SLAM-Verfahren (Simultaneous Localization and Mapping) eingesetzt, wie es von Cetto et. al. [2] beschrieben wird. Dabei wird eine Projektor mit einer Kinect zu einer mobilen Einheit gekoppelt (SLAM-Projektor) und mithilfe der KinectFusion-Technik ein Abbild der Umgebung in Echtzeit erstellt. Zusätzlich ermittelt KinectFusion die Position und die Ausrichtung des SLAM-Projektors bezüglich der bereits gescannten Umgebung. Mithilfe der Position und Ausrichtung des SLAM-Projektor sowie den Umgebungsdaten kann die Projektion in Echtzeit entsprechend angepasst werden, sodass sie auf beliebigen Oberflächen unverzerrt dargestellt werden kann.
Da das System auf der KinectFusion-Technik beruht, ist es nur begrenzt oberflächenadaptiv. Bei KinectFusion darf sich eine Umgebung nur bis zu einem gewissen Grad verändern, wie in Abschnitt 2.7.1 erläutert wird. Sollte außerdem der SLAM-Projektor sehr schnell im Raum bewegt werden, kann kein genaues 3D-Abbild erstellt werden, da die Tiefendaten zu großen Schwankungen unterliegen und die einzelnen Algorithmen von KinectFusion falsche Daten liefern. Wird jedoch von diesen Grenzsituationen abgesehen, kann der SLAM-Projektor auf jede beliebige Oberfläche in Echtzeit projizieren und kann auch seine intern erstellte 3D-Umgebung bei Veränderungen anpassen. Zudem handelt es sich um mobiles System, das flexibel eingesetzt

werden kann und an keinen bestimmten Ort oder an sonstiges Hardware gebunden ist.

2.8 Vergleich der Arbeiten

Tabelle 2.1 veranschaulicht die Vor- und Nachteile der vorgestellten Arbeiten und zeigt auf, wie sich diese Arbeit von ihnen abhebt. Es werden die Kriterien *beliebige Oberflächen*, *Echtzeitentzerrung*, *Mobil* und *Oberflächenadaptiv* miteinander verglichen. Dabei können die Werte *Ja (✓)*, *beschränkt (✳)* und *Nein (×)* vorkommen. Wird ein Kriterium nur beschränkt erfüllt, so müssen weitere Bedingungen und Einschränkungen gelten. Beispielsweise kann bei dem Entzerren mithilfe von elektronischen Tags [18] zwar auf beliebige Oberflächen projiziert werden, allerdings dürfen diese nur ein gewisses Maß an Komplexität haben (keine gekrümmten Oberflächen), wie in Kapitel 2.5 beschrieben. Zudem ist bei keiner der vorgestellten Arbeiten, die Software frei verfügbar.
Die Projekte werden folgendermaßen abgekürzt:

a) Automatic Projector Calibration with Embedded Light Sensors (Kapitel 2.1)

b) Semi-Automatic Realtime Calibration (Kapitel 2.2)

c) Everywhere-Display (Kapitel 2.3)

d) Illuminating-Clay (Kapitel 2.4)

e) RFIG Lamps: Entzerrung mit Tags (Kapitel 2.5)

f) RFIG Lamps: Projektionsstabilisierung (Kapitel 2.5)

g) Flexpad (Kapitel 2.6)

h) RoomProjector (Kapitel 2.7)

i) SLAMProjector (Kapitel 2.7)

j) diese Arbeit

	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)
beliebige Oberflächen	×	×	✓	✓	✳	×	✳	✓	✓	✓
Echtzeitentzerrung	×	✓	✓	✓	×	✓	✓	✓	✓	✓
Mobil	✳	✓	×	×	✓	✓	×	×	✓	✓
Oberflächenadaptiv	×	×	×	✓	×	×	✳	×	✳	✓

Tabelle 2.1: Vergleich der Arbeiten.

3 Mathematische und Technische Grundlagen

Dieses Kapitel befasst sich mit den theoretischen Grundlagen der Projektionsentzerrung. Um zu verstehen, wie eine verzerrte Projektion entsteht und wie dem entgegengewirkt werden kann, wird in Abschnitt 3.1 zuerst das Lochkameramodell beschrieben. In Abschnitt 3.2 wird anschließend auf die Einzelheiten des Zustandekommens einer verzerrten Projektion eingegangen. Mithilfe von diesem Wissen kann herausgefunden werden, wie ein zu projizierendes Bild verzerrt werden muss, damit die Projektion unverzerrt dargestellt wird. Um solch ein verzerrtes Bild zu erhalten, muss mithilfe der Kinect-Daten ein 3D-Modell der Oberfläche erstellt werden, auf die projiziert wird, was in Abschnitt 3.3 beschrieben wird. Abschnitt 3.4 erläutert schließlich, wie dieses 3D-Modell und Texture Mapping genutzt werden kann, um das gewünschte verzerrte Bild zu erhalten.

3.1 Lochkameramodell

Nimmt man einen Karton, sticht mit einer Nadel ein Loch in die Mitte einer Kartonseite und ersetzt die gegenüberliegende Kartonseite mit einer halb durchsichtigen Folie, so kann eine sogenannte Lochkamera nachgebaut werden. Hält man diese in einem abgedunkelten Raum vor eine Lichtquelle, beispielsweise eine angezündete Kerze, sodass das Loch zur Kerze hin ausgerichtet ist, so entsteht auf der halb durchsichtigen Folie ein Abbild der Kerze, was in Abb. 3.1 veranschaulicht wird. Dieses Abbild ist um 90° gedreht und steht somit auf dem Kopf. Diese Rotation kommt folgendermaßen zustande. Die Lichtstrahlen, die von der Kerze ausgehen, werden durch das Loch auf die Folie projiziert. Nimmt man die gedachte Gerade, die von einem Punkt der Kerze ausgeht und durch das Loch der Lochkamera verläuft, so wird dieser Punkt auf die Stelle der Folie projiziert, an der die gedachte Gerade die Folie schneidet. In Abb. 3.1 sind 3 solcher gedachten Linien eingezeichnet. Eine beginnt am oberen Ende der Kerzenflamme, eine am unteren Ende der Flamme und eine am unteren Ende der Kerze. Im Loch der Lochkamera schneiden sich alle diese Linien was letztendlich zur 90° Rotation führt. Im rein mathematischen Lochkameramodell wird oft ein Abbild ohne Rotation betrachtet, indem man eine virtuelle Bildebene zwischen Kameraloch und Objekt hinzunimmt. Diese virtuelle Bildebene liegt parallel zu der Ebene der Lochkamera, in der sich das Loch befindet. Zudem ist sie genau so weit vom Kameraloch entfernt, wie die halb durchsichtige Folie. Aufgrund der Punktsymmetrie entspricht das so entstandene, virtuelle Bild der 90° Rotation des Bildes, das auf der halb durchsichtigen Folie entsteht. Die-

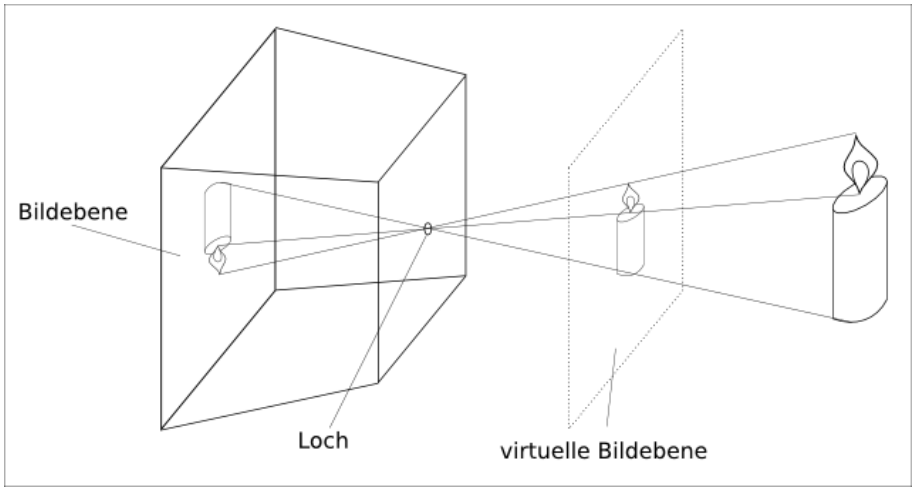

Abbildung 3.1: Entstehung eines Bildes mit einer Lochkamera.

ses Prinzip des Abbildens von Objekten des dreidimensionalen Raumes auf einer zweidimensionalen Ebene nennt sich Zentralprojektion. Mit ihr kann das Sehen mit dem menschlichen Auge beschrieben werden oder die Funktionsweise einer Kamera mit Linse. Die Zentralprojektion ist ein mathematisches Modell, das beschreibt, wie Punkte eines dreidimensionalen Raumes auf einer zweidimensionalen Ebene abgebildet werden können.

3.1.1 Eigenschaften der Zentralprojektion

Um ein zweidimensionales Abbild von dreidimensionalen Objekten, bezüglich der Zentralprojektion zu erhalten, müssen sich alle Lichtstrahlen, die von dem Objekt ausgehen, in einem Punkt schneiden. Dieser Punkt wird *Brennpunkt* genannt. Weiterhin wird ein sogenannter Hauptpunkt definiert. Der *Hauptpunkt* ist ein spezieller Punkt der *Bildebene*, der in Abb. 3.1 dargestellten Ebene zwischen Brennpunkt und den abzubildenden Objekten. Die gedachte Linie, ausgehend vom Brennpunkt, durch den Hauptpunkt wird *optische Achse* genannt. Ein Punkt auf der Bildebene wird *Bildpunkt* genannt und der Abstand zwischen Brennpunkt und Hauptpunkt heißt *Brennweite*. Vergrößern oder Verkleinern der Brennweite kann mit dem Zoom einer Kamera verglichen werden. Wird die Brennweite vergrößert, so wird nur noch ein kleinerer Teil einer Szene auf einem vordefinierten Bereich der Bildebene, dem letztendlichen Bild, abgebildet. Allerdings wird dieser kleinere Teil vergrößert dargestellt. Wird die Brennweite verringert, so werden Objekte kleiner dargestellt und es passen zudem mehr Objekte ins Bild. Analog kann auch der Fall betrachtet werden, dass sich ein Objekt der Kamera nähert, oder sich von dieser entfernt. Entfernt sich das Objekt, so wird es kleiner im 2D-Bild dargestellt, nähert sich das Objekt, so wird es größer abgebildet. Durch Verschieben des Hauptpunktes bei einer

gleichbleibenden, statischen Szene, werden verschiedene Bereiche der Szene dargestellt. Wird beispielsweise der Hauptpunkt nach rechts verschoben, so wird auch der dargestellte Szenenausschnitt nach rechts verschoben. Die Parameter Brennweite, Hauptpunkt sowie auch die Bildgröße werden als *intrinsische Parameter* bezeichnet. Die Position der Kamera im Raum sowie deren Ausrichtung werden dagegen als *extrinsische Parameter* bezeichnet. Eine durch extrinsische und intrinsische Parameter definierte Zentralprojektion bildet somit einen Punkt der dreidimensionalen Welt auf eine zweidimensionale Ebene ab. Dazu müssen die Koordinaten eines 3D-Punktes in kameraspezifische 2D-Koordinaten, oft Pixel, umgewandelt werden. Für diese Abbildung kann die Formel von R. Hartley und A. Zisserman [8] (Seite 141; (5.4)) verwendet werden:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.1}$$

$$u = x \div z \tag{3.2}$$

$$v = y \div z \tag{3.3}$$

Dabei bezeichnet f die Brennweite, p_x die x-Koordinate des Hauptpunktes und p_y die y-Koordinate des Hauptpunktes. Somit können bei gegebenen 3D-Koordinaten X, Y und Z, die Bildkoordinaten u und v ermittelt werden, auf die der 3D-Punkt abgebildet wird. u und v erhält man durch Division von x und y durch z. Das Lochkameramodell nimmt dabei an, dass die Bildkoordinaten der Bildebene kartesische Koordinaten sind, bei denen beide Koordinatenachsen den selben Maßstab haben. Bei Digitalkameras mit CCD-Sensor kann es jedoch der Fall, dass die Pixel eines Bildes nicht quadratisch sind und die Koordinatenachsen somit nicht den gleichen Maßstab haben. Dazu kann (3.1) folgendermaßen abgewandelt werden ([8] Seite 143; (5.9)) :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.4}$$

Mit $\alpha_x = f * m_x$ und $\alpha_y = f * m_y$. Dabei bezeichnet f wie vorher die Brennweite der Kamera. m_x und m_y bezeichnen die Größe eines Kamerapixels in x- und y-Richtung, $x_0 = m_x * p_x$ und $y_0 = m_y * p_y$ den Hauptpunkt bezüglich der Pixelgröße. Neben der Koordinatenumwandlung von 3D nach 2D muss oft noch eine weitere Koordinatenumwandlung betrachtet werden. Es wird in der Regel zwischen Kamera-Koordinatensystem und Welt-Koordinatensystem unterschieden. Der Ursprung des Kamera-Koordinatensystems muss nicht mit dem Ursprung des Welt-Koordinatensystems übereinstimmen. Ebenso können beide Koordinatensysteme unterschiedliche Maßstäbe haben und ihre Achsenbezeichnungen können sich unter-

scheiden. (3.4) ist nur gültig, wenn die gegebenen 3D-Koordinaten im Kamera-Koordinatensystem vorliegen. Sollte dies nicht der Fall sein, so müssen die Welt-Koordinaten eines 3D-Punktes zuerst in Kamera-Koordinaten umgewandelt werden. Dazu müssen auf die abzubildenden 3D-Punkte die selbe Rotation und die selbe Translation angewandt werden, die auch auf die Kamera bezüglich des Weltkoordinatensystems angewendet werden. Hat die Kamera die Welt-Koordinaten X_K, Y_K und Z_K, welche sich durch eine Translation T_K und eine Rotation R_K, vom Welt-Koordinaten-Ursprung aus, ergeben, so muss diese Rotation und diese Translation auch auf jeden 3D-Punkt angewandt werden. (3.4) kann dazu folgendermaßen abgeändert werden (vgl. [8] Seite 142; (5.7)).

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} * R * \left[I| - \tilde{C} \right] * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.5}$$

Dabei wird ausgenutzt, dass sich eine Rotation durch eine Matrizenmultiplikation darstellen lässt und eine Translation durch eine Vektoraddition. In 3.5 ist R eine $3 \times 3 - Matrix$ und ist für die Rotation im Weltkoordinatensystem zuständig. $\left[I| - \tilde{C} \right]$ ist eine $4 \times 3 - Matrix$, die auf der linken Seite aus einer $3 \times 3 - Einheitsmatrix$ besteht und $-\tilde{C}$ der Spaltenvektor ist, der für die Translation verantwortlich ist. $\left[I| - \tilde{C} \right]$ ist somit gleichbedeutend mit einer Vektoraddition und bildet zusammen mit R die extrinsischen Parameter einer Kamera.

3.2 Entstehung einer verzerrten Projektion und Entzerrung

Während eine Kamera eine dreidimensionale Umgebung auf eine zweidimensionale Ebene abbildet, findet bei einem Projektor der umgekehrte Prozess statt. Hier wird ein zweidimensionales Bild auf eine dreidimensionale Umgebung projiziert. Analog zur Kamera können auch für einen Projektor dessen intrinsische und extrinsische Parameter definiert werden. Sollte die optische Achse orthogonal zu einer flachen Projektionsfläche liegen, so liegt ein vereinfachter Fall vor, bei dem die dritte Dimension (Tiefeninformationen) vernachlässigt werden kann. Ist dies jedoch nicht der Fall, entsteht eine verzerrte Projektion, die unerwünschte Effekte mit sich bringen kann. In Abschnitt 3.1.1 wurde erläutert, dass ein weiter entferntes Objekt kleiner im Kamerabild dargestellt wird als ein näher liegendes Objekt. Den umgekehrten Effekt kann bei einem Projektor beobachtet werden. Je weiter die Projektionsfläche vom Projektor entfernt ist, desto größer wird die Projektion. Je näher die Projektionsfläche herangerückt wird, desto kleiner wird die Projektion. Hat eine Projektionsfläche nun einen näher liegenden und einen weiter entfernteren Teil, so wird die Projektion im näher liegenden Teil kleiner dargestellt und im entfernteren Teil größer. Abb. 3.2 zeigt jeweils einen Projektor sowie 2 von ihm ausgehende Lichtstrahlen. Der obere Lichtstrahl ist dabei für die Oberkante der Projektion verantwortlich, der untere Teil für die Unterkante. Im linken Teil der Abbildung steht die optische Achse des

Projektors orthogonal zur Projektionsfläche, im rechten Teil ist dies nicht mehr der Fall. Es ist zu erkennen, dass in der rechten Abbildung der obere Lichtstrahl einen weiteren Weg zurücklegen muss als der untere. In der linken Abbildung dagegen legen beide Lichtstrahlen einen gleich langen Weg zurück. In der rechten Abbildung ist

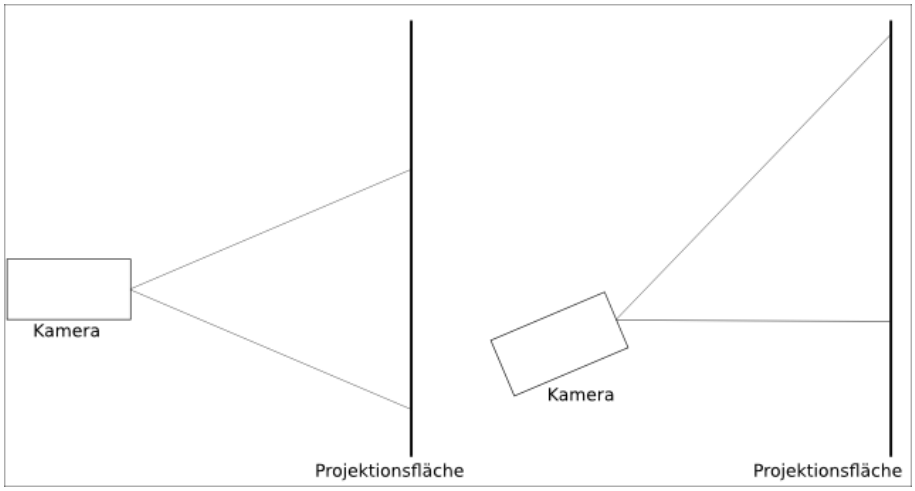

Abbildung 3.2: Projektionsänderung bei Kameraneigung

somit die obere Bildkante weiter vom Projektor entfernt als die untere Bildkante und aufgrund der in Abschnitt 3.1.1 beschriebenen Eigenschaften und der umgekehrten Funktionsweise zu einer Kamera, entsteht so ein Bild, dessen Oberkante breiter ist als dessen Unterkante. Soll dem nun entgegen gewirkt werden, so kann die Oberkante des Ursprungsbildes gestaucht und die Unterkante gestreckt werden. Im Ursprungsbild ist somit die Unterkante größer als die Oberkante. Wird dieses Bild projiziert, hebt sich die ursprüngliche Verzerrung auf und es entsteht eine rechteckige, unverzerrte Projektion. Abb. 3.3 zeigt im linken Bild eine Projektion die verzerrt wird, da der Projektor nicht orthogonal zur Projektionsfläche hin ausgerichtet ist. Im rechten Bild ist zu sehen, wie das Ursprungsbild angepasst werden muss, um eine unverzerrte Projektion zu erhalten. Im Allgemeinen gilt, dass zu einer verzerrten Projektion, deren inverses Bild erzeugt werden muss und dieses projiziert werden muss, damit sich eine Verzerrung aufhebt. Da sich eine Verzerrung lediglich auf die Form der Projektion, nicht aber auf die Farbdarstellung auswirkt, entspricht das *inverse Bild* zu einer verzerrten Projektion dem Ursprungsbild, wobei die Form so verändert wird, dass sich die Verzerrung beim Projizieren aufhebt. Entsteht beim Projizieren keine Verzerrung, so entspricht das inverse Bild dem unveränderten Ursprungsbild.

3.3 Erstellen eines 3D Modells

Um ein, wie in Abschnitt 3.2 beschriebenes, inverses Bild zu erzeugen, muss ein 3D-Modell der Oberfläche, auf die projiziert wird, erstellt werden. Da ein Projek-

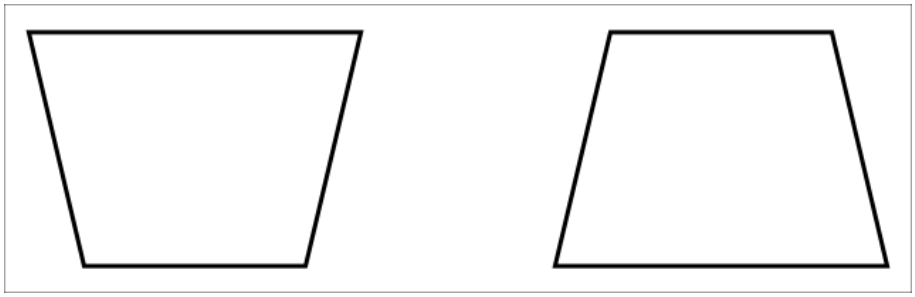

Abbildung 3.3: Entzerren durch Ändern des Ursprungsbildes.

tor keine Informationen über die Oberfläche hat, wird der Tiefensensor der Kinect genutzt, um die entsprechenden Informationen zu erhalten. Mit einer zusätzlichen Kalibrierung von Kinect und Projektor, wie in Abschnitt 4.3 beschrieben wird, kann die Projektionsfläche rekonstruiert werden. Durch die Kalibrierung von Kinect und Projektor werden deren intrinsische und extrinsische Parameter bestimmt, womit alle freien Parameter von (3.5) bestimmt sind. Die Rekonstruktion der Projektionsfläche geschieht nun in 3 Schritten. Zuerst wird zu jedem Kinect-Pixel ein Punkt im dreidimensionalen Raum zugeordnet. Anschließend wird jedem dieser Punkte ein Projektor-Pixel zugeordnet. Im letzten Schritt werden schließlich alle Punkte aussortiert, zu denen kein Projektor-Pixel gefunden werden kann. Die 3 Schritte werden im Folgenden genauer erläutert.

3.3.1 Schritt 1: Kinect → Welt

Schritt 1 nutzt ausschließlich die intrinsischen und extrinsischen Parameter der Kinect.
Mithilfe der Kinect Tiefendaten und der Eigenschaften der Zentralprojektion kann jedem Kinect-Pixel ein Punkt in der dreidimensionalen Welt zugeordnet werden. Dazu wird (3.5) so umgestellt, dass anstatt Pixel-Koordinaten aus Welt-Koordinaten, Welt-Koordinaten aus Pixel-Koordinaten berechnet werden. Da jedoch durch (3.2) und (3.3) bei der Welt-Pixel Transformation die Tiefendaten verloren gehen, kann nicht ohne Weiteres aus einem Pixel die entsprechende Welt-Koordinate bestimmt werden. Stattdessen wird jedem Pixel eine Gerade zugeordnet, deren Punkte alle auf das entsprechende Pixel bei der Welt-Pixel Transformation abgebildet werden würden. Nimmt man schließlich die Tiefendaten eines Kinect-Pixels hinzu, kann zu einem Pixel die entsprechende Welt-Koordinate eindeutig bestimmt werden. Diese liegt auf der entsprechenden Gerade mit einem Abstand zum Kinect-Ursprung entsprechend der Tiefendaten des jeweiligen Pixels. Um die Welt-Koordinaten X, Y, Z zu einem Kinect-Pixel (u, v) zu erhalten, wird (3.5) in mehreren Schritten abgeän-

dert. Zunächst wird (3.5) folgendermaßen vereinfacht:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} * \Big[R|T\Big] * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.6}$$

Wobei der linke Teil entsprechend (3.2) und (3.3) angepasst wurde und Rotations- und Translationsmatrix zu $\Big[R|T\Big]$ zusammengefasst wurden. (3.6) wird nun aufgelöst sodass X, Y und Z zu einem festen Kinect-Pixel (u, v) berechnet werden können:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = sR^{-1} * \begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} * \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} - R^{-1} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \tag{3.7}$$

Schließlich wird (3.7) vereinfacht zu:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = s * Q - J \tag{3.8}$$

Mit:

$$Q = R^{-1} * \begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} * \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{3.9}$$

$$J = R^{-1} * \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \tag{3.10}$$

Für ein gegebenes Kinect-Pixel (u, v), ist dessen Tiefenwert bekannt und damit auch Z. Für s gilt somit:

$$\begin{aligned} & Z = s * q_3 - j_3 \\ \leftrightarrow \quad & s = \tfrac{Z - j_3}{q_3} \end{aligned} \tag{3.11}$$

Wobei q_3 der entsprechende Eintrag der Matrix Q ist, j_3 der entsprechende Eintrag der Matrix J. Durch Einsetzen von (3.11) in (3.8) können schließlich X und Y berechnet werden:

$$\begin{aligned} X &= \tfrac{Z - j_3}{q_3} * q_1 - j_1 \\ Y &= \tfrac{Z - j_3}{q_3} * q_2 - j_2 \end{aligned} \tag{3.12}$$

Mit Einträgen q_1 und q_2 aus Q, j_1 und j_2 aus J. Für jedes Kinect-Pixel (u, v) kann somit die Weltkoordinate der Oberfläche bestimmt werden, die auf das entsprechende Kinect-Pixel abgebildet wird.

3.3.2 Schritt 2: Welt → Projektor

Schritt 2 nutzt ausschließlich die intrinsischen und extrinsischen Parameter des Projektors.
In Schritt 1 wurden Weltkoordinaten zu jedem Kinect-Pixel berechnet. Um jedoch das gewünschte 3D-Modell zu erhalten, müssen Weltkoordinaten für Projektor-Pixel bestimmt werden. Dazu werden die in Schritt 1 berechneten Weltkoordinaten bezüglich (3.5), (3.2) und (3.3) auf eine Ebene projiziert. Da hier für (3.5) die intrinsischen und extrinsischen Parameter des Projektors verwendet werden, entspricht die Ebene, auf die projiziert wird, genau der Bildebene des Projektors. Es genügt nicht, aus den in Schritt 1 erzeugten Weltkoordinaten ein 3D-Modell zu erstellen. Dies wäre nur der Fall, wenn Projektor und Kinect dieselben intrinsischen und extrinsischen Parameter besitzen würden. Da jedoch zwei Geräte benutzt werden, die aufgrund ihrer physikalischen Gegebenheiten, nicht dieselben extrinsischen Parameter haben, müssen die aufgeführten Schritte durchgeführt werden. Ein Kinect-Pixel $p_k = (\tilde{u}, \tilde{v})$ kann beispielsweise der Weltkoordinate $(\tilde{X}, \tilde{Y}, \tilde{Z})$ zugeordnet werden. Aufgrund unterschiedlicher Projektor-Parameter wird diese Weltkoordinate jedoch nicht dem Projektor-Pixel $p_p = (\tilde{u}, \tilde{v})$ zugeordnet, sondern $p_k = (\tilde{u}', \tilde{v}')$. Schritt 2 berechnet somit, aus den Weltkoordinaten aus Schritt 1, die entsprechenden Projektorkoordinaten. Dadurch lässt sich auch jedem Kinect-Pixel ein Projektor-Pixel zuordnen.

3.3.3 Schritt 3: Aussortieren von Pixel und Interpolation

Für die Brennweiten von Kinect und Projektor muss gelten, dass die Brennweite des Projektors größer als die Brennweite der Kinect sein muss. Sollte dies nicht der Fall sein, so ist das projizierte Bild größer als der Ausschnitt, der mit der Kinect erfasst wird. Den Projektor-Pixeln die nicht im Kinect-Ausschnitt liegen können somit keine Weltkoordinaten zugeordnet werden und im zu erstellenden 3D-Modell fehlen Daten um eine Projektion korrekt zu entzerren. Damit eine Projektionsentzerrung jederzeit möglich ist, müssen sich die Brennweiten von Kinect und Projektor so unterscheiden, dass das projizierte Bild jederzeit von der Kinect aufgenommen werden kann. In der Regel gibt es somit Kinect-Pixel, zu denen kein Projektor-Pixel gefunden werden kann, da diesen Kinect-Pixel Weltkoordinaten zugeordnet werden, die bereits außerhalb der Projektion liegen. Diese werden zur Projektionsentzerrung nicht benötigt und werden herausgefiltert. Je nach Auflösung von Kinect und Projektor kann es passieren, dass nicht jedem Projektorpixel ein Kinect-Pixel mit zugehöriger Weltkoordinate zugeordnet werden kann. Um dennoch ein Lückenloses 3D-Modell zu erstellen, werden die fehlenden Informationen durch Interpolation hinzugerechnet. Kann beispielsweise den Projektor-Pixel $(4, 5)$ und $(6, 5)$ eine Weltkoordinate zugeordnet werden, $(5, 5)$ jedoch nicht, so wird die Weltkoordinate für $(5, 5)$ aus den beiden anderen berechnet. Letztendlich erhält man so zu jedem Projektor-Pixel eine Weltkoordinate, sodass sich ein lückenloses, virtuelles Abbild der Oberfläche, auf die projiziert wird, ergibt.

3.4 Entzerren mittels Texture Mapping

Wie in Abschnitt 3.2 beschrieben, muss zum Entzerren einer Projektion deren inverses Bild projiziert werden. Dies wird mithilfe eines 3D-Modells erzeugt, das mit den in Abschnitt 3.3 dargestellten Schritten erzeugt werden kann. Dieses 3D-Modell ist ein virtuelles Abbild der Oberfläche, auf die projiziert werden soll. Um nun das gewünschte inverse Bild zu erzeugen, wird ausgenutzt dass eine Kamera und ein Projektor inverse Operationen durchführen. Während eine Kamera die Farben von Objekten der dreidimensionalen Welt auf die Bildebene projiziert, werden bei einem Projektor die Bildpunkte der Bildebene auf die dreidimensionale Welt projiziert. Da die extrinsischen und intrinsischen Parameter des Projektors bekannt sind, kann im virtuellen Abbild die Translation und Rotation des Projektors der physikalischen Welt simuliert werden. Die Ausrichtung des Projektors und seine Position stimmen somit in der physikalischen sowie virtuellen Welt überein. Der virtuelle Projektor würde daher in der virtuellen Welt dieselbe Projektion erzeugen, wie der physikalische Projektor in der physikalischen Welt. Das 3D-Modell kann nun mit einer beliebigen Grafik texturiert werden und der Projektor in der virtuellen Welt kann mit einer Kamera ersetzt werden. Dieser werden die extrinsischen und intrinsischen Parameter des Projektors übergeben. Die Kamera nimmt somit ein Bild der texturierten Oberfläche auf und aufgrund der in Abschnitt 3.2 beschriebenen, inversen Eigenschaften von Kamera und Projektor, wird so das gewünschte inverse Bild erzeugt. Diese Vorgehensweise wird auch in [15] und [2] verwendet. Allerdings werden dort die 3D-Modelle anders erzeugt.

4 Implementierung

In diesem Kapitel wird auf die Implementierung der Entzerrungssoftware eingegangen. In Abschnitt 4.1 wird die benuzte Hardware beschrieben und ein kleiner Einblick in das XNA-Framework sowie auf openCV gegeben. Abschnitt 4.2 befasst sich mit der Implementierung des Entzerrungsprozesses und Abschnitt 4.3 veranschaulicht die Kalibrierung von Kinect und Projektor mit der Hilfe von OpenCV, um die extrinsischen und intrinsischen Parameter für die Kinect und den Projektor zu erhalten. Schließlich wird in Abschnitt 4.4 demonstriert, wie die Software die Projektion einer Beispielanwendung entzerrt.

4.1 Verwendete Hard-und Software

4.1.1 Tiefensensor: Kinect

Die Kinect von Microsoft ist ein Gerät zur Steuerung von Xbox 360 Videospielen, wobei mittlerweile auch Gerätetreiber für PC zur Verfügung stehen. Mithilfe der Kinect können Bewegungen eines Benutzers erkannt werden und in entsprechende Steuersignale für die Xbox umgewandelt werden. Somit entfällt der typische Controller zur Spielsteuerung und der Benutzer selbst wird zum Eingabegerät. Dazu ist in der Kinect eine RGB-Kamera sowie ein Tiefensensor verbaut, deren Daten zur Bewegungserkennung genutzt werden. Der Tiefensensor besteht aus einer Lichtquelle, die ein Infrarotmuster aussendet und einer speziellen Kamera, die dieses Muster erkennt. Mit der *Structured-Light* Methode können Tiefendaten aus dem Infrarotmuster abgeleitet werden und eine Tiefenbild wie in 1.2 wird erzeugt. Anhand dieses Tiefenbildes und mit Methoden des maschinellen Lernens, leitet die Kinect die Position und die Haltung einer Person ab, womit entsprechende Kommandos zur Spielsteuerung abgeleitet werden. Zur Projektionsentzerrung wird der eingebaute Tiefensensor genutzt, um Informationen über die Oberfläche zu erhalten, auf die projiziert werden soll. Die Kinect liefert dazu 30 Tiefenbilder pro Sekunde, was ausreichend für eine Echtzeitentzerrung ist.

4.1.2 Prototyp

Der verwendete Prototyp, der in Abb. 4.1 zu sehen ist, besteht aus einer Microsoft Kinect und einem Axaa L1 v2 Laser-Projektor [1]. Der Projektor hat eine Auflösung von 800×600 und kann Bilder mit einer Diagonalen von bis zu 50 Zoll projizieren. Der Vorteil eines Laserprojektors ist, dass keine Fokussierung notwendig ist. So wird stets jeder Teil der Projektion scharf dargestellt, auch wenn ein Teil näher

am Projektor liegt als ein Anderer. Die Entzerrung wird mit einem PC berechnet, der einen AMD Athlon(tm) II X2 255 Prozessor und eine Nvidia GeForce 8800 GT mit 512 MB RAM verwendet. Wie der SLAM-Projektor aus Abschnitt 2.7.3 werden auch hier Projektor und Kinect so fixiert, dass sich der Abstand und die Ausrichtung der beiden zueinander nicht verändern kann. Die gesamte Kinect-Projektor-Einheit bleibt jedoch mobil benutzbar.

Abbildung 4.1: Prototyp aus Kinect und Mini-Projektor.

4.1.3 XNA

XNA ist eine Programmierschnittstelle, die in erster Linie zur Spieleprogrammierung geschaffen wurde. Mit ihr können Spiele auf DirectX-Basis erstellt werden und somit Windows PC-Programme (Windows XP - Windows 8), Windows Phone und XBox Programme. XNA Programme werden mit der objektorientierten Programmiersprache C# erstellt. XNA wird als Basis der Entzerrungssoftware genutzt, sodass die oben genannten Programmtypen Gebrauch davon machen können. Im Folgenden werden XNA-spezifische Klassen und Funktionen vorgestellt, die für den Entzerrungsprozess genutzt werden. Wie über diese der Entzerrungsprozess in XNA integriert wird, ist in Abschnitt 4.2.1 erläutert.

- **Game-Klasse**
  Die Klasse *Microsoft.Xna.Framework.Game* dient als Einstiegspunkt für jedes XNA-Programm. Da XNA in erster Linie auf die Spieleentwicklung ausgerichtet ist, werden entsprechende Funktionen in der Game-Klasse zur Verfügung gestellt. Somit kann beispielsweise die Funktion *Initialize()* implementiert werden, um entsprechende spielespezifische Initialisierungsschritte durchzuführen. Die folgenden Funktionen der Game-Klasse dürfen nicht überschrieben werden, da über sie das Entzerrungstoolkit in den Rendering-Prozess eingebunden wird. Alternativ werden entsprechenden Funktionen bereitgestellt,

die mit dem Suffix *Undistorted* enden. So wird beispielsweise für die Funktion *Update(GameTime gameTime)* die Alternativfunktion *UpdateUndistorted(GameTime gameTime)* zur Verfügung gestellt.

- **Initialize()**
  Die Initialisierungsmethode übernimmt Initialisierungsaufgaben für die Projektionsentzerrung. Hier wird eine Verbindung zur Kinect hergestellt und Ressourcen zum späteren Rendern reserviert. Beispielsweise werden entsprechende Buffer initialisiert oder der virtuellen Kamera aus Abschnitt 3.4 die entsprechend intrinsischen und extrinsischen Parameter übergeben.

- **Update(GameTime gameTime)**
  Die Funktion *Update()* wird für gewöhnlich dazu genutzt, um einen Spieleschritt zu simulieren. So wird beispielsweise die neue Position einer Spielfigur berechnet, entsprechend den Eingaben, die ein Benutzer über einen Controller getätigt hat. Die Funktion wird automatisch so lange aufgerufen, bis das Programm beendet oder pausiert wird.

- **Draw(GameTime gameTime)**
  Über die *Draw(GameTime gameTime)* Methode werden Inhalten gerendert und sie wird, wie die Update-Methode, so lange aufgerufen, bis das Programm gestoppt oder pausiert wird.

- **RenderTarget2D Klasse**
  Um Bilder in XNA rendern zu können, muss ein Ziel angegeben werden, in das gerendert werden soll. Solche Ziele können mit der *RenderTarget2D* Klasse realisiert werden. Soll beispielsweise ein Primitiv gerendert werden, so muss zuerst ein Ziel aktiviert werden und mit einem *DrawPrimitives()* Aufruf wird das Primitiv in das aktivierte Ziel gerendert. Ein spezielles Render-Ziel ist der *BackBuffer*. Wird in diesen gerendert, so erscheint das gerenderte Bild auf dem Bildschirm.

- **Texture2D**
  Ein *Texture2D* Objekt stellt eine Textur in XNA dar. Die Texture2D-Klasse bietet eine einfache Möglichkeit Bilddaten in eine Textur zu schreiben und diese wieder auszulesen.

Soll nun die Entzerrungssoftware in einem anderen XNA-basierenden Programm genutzt werden, so muss dessen Game-Klasse die Klasse *UndistortedGame* erweitern und die Methoden *InitializeUndistorted*, *UpdateUndistorted* und *DrawUndistorted* müssen implementiert werden.

4.1.4 OpenCV

OpenCV [7] ist eine Programmbibliothek die Unterstützung für maschinelles Sehen und Bildverarbeitung bietet. Mithilfe von OpenCV wird der in Abschnitt 4.3 be-

schriebene Kalibrierungsprozess von Kinect und Projektor realisiert. Weiterhin bietet OpenCV eine einfache Möglichkeit, um den Entzerrungsprozess zu parallelisieren. Dazu nutzt openCV die CUDA-Plattform [14] von Nvidia, sodass Berechnungen auf der Grafikkarte ausgeführt werden können. Damit openCV unter C# genutzt werden kann, wird die Wrapper-Bibliothek Emgu CV [6] genutzt. Auf die genutzten openCV-Funktionen wird an den entsprechenden Stellen eingegangen.

4.2 Projektionsentzerrung

Dieser Abschnitt befasst sich mit der eigentlichen Implementierung der Projektionsentzerrung und erläutert, wie die in Abschnitt 3 beschriebenen Methoden praktisch realisiert werden. Abschnitt 4.2.1 zeigt, wie auf Basis von XNA ein inverses Bild erzeugt wird. Abschnitt 4.2.2 beschreibt, wie die Berechnungen aus Abschnitt 3.3 eingebunden werden um ein 3D-Modell zu erzeugen. In Abschnitt 4.2.3 wird veranschaulicht wie die Berechnungen parallelisiert werden können und in Abschnitt 4.2.4 wird genauer auf den Interpolierungsprozess eingegangen.

4.2.1 Erzeugen eines inversen Bildes

In Abschnitt 3.3 wurde beschrieben, wie ein 3D-Modell berechnet werden kann, das die physische Projektionsfläche widerspiegelt und mit dessen Hilfe ein inverses Bild erzeugt werden kann. Das 3D-Modell, das zur Entzerrung genutzt wird, baut sich aus einem sogenannten Mesh auf. Ein Mesh besteht aus Knotenpunkten, die miteinander zu einem Gitternetz verbunden sind. Im Hier benutzten Mesh werden immer jeweils 3 Knotenpunkte zu einem Dreieck zusammengefasst. Dabei können Knotenpunkte zu mehr als einem Dreieck gehören, um Speicherplatz zu sparen. Ein Beispiel eines solchen 3D-Meshes ist in Abb. 4.2 zu finden. Für jedes Projektorpixel

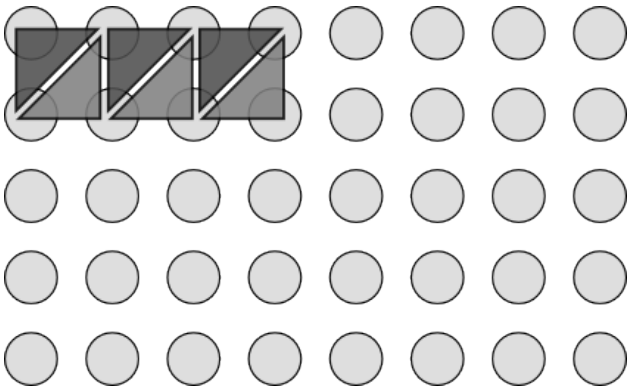

Abbildung 4.2: Mesh aus Knotenpunkten und Dreiecken.

wird ein Mesh-Knotenpunkt angelegt und mit den Knotenpunkten der benachbarten Pixel verbunden. Zum Entzerren werden die 3D-Koordinaten der Knotenpunkte

entsprechende der 3D-Weltkoordinate, auf die das zugehörige Projektorpixel projiziert, verändert. Das Mesh bildet somit die Oberfläche, auf die projiziert werden soll, virtuell ab. Soll nun ein inverses Bild erzeugt werden, das die Verzerrung beim Projizieren aufhebt, müssen die Bilddaten in einer Textur gespeichert werden. Mit dieser wird, das Mesh texturiert, wodurch das inverse Bild, wie in Abschnitt 3.4 beschrieben, erzeugt wird.

Das Mesh wird in der Methode Initialize() des XNA-Frameworks erzeugt. Dazu wird ein Array von *VertexPositionColorTexture* Objekten erstellt, dessen Länge der Auflösung des verwendeten Projektors entspricht. Für jeden Mesh-Knotenpunkt wird somit ein VertexPositionColorTexture Objekt angelegt. In der Update() Methode werden die Werte der Mesh-Knotenpunkte angepasst, indem die berechneten Weltkoordinaten den entsprechenden VertexPositionColorTexture Objekten übergeben werden. Anschließend werden in der Draw() Methode die Daten aus dem aktivierten RenderTarget2D Objekt in ein Texture2D Objekt kopiert. Dieses kann nun mit einem Aufruf von *GraphicsDevice.DrawUserIndexedPrimitives()* auf das Mesh angewandt werden. Soll beispielsweise das Bild einer XNA basierenden Animation entzerrt werden, so kann die Animation ihre Daten einfach in den BackBuffer rendern, das Entzerrungstoolkit ließt diese Daten automatisch aus und schreibt sie in ein Texture2D Objekt, das auf das Mesh angewandt wird. Die Update() Methode ändert somit bei jedem Aufruf das Mesh ab und passt es, mithilfe der Tiefendaten der Kinect, an die derzeitige Oberfläche an. Mit diesem veränderte Mesh wird in der Draw() Methode das gewünschte inverse Bild erzeugt.

4.2.2 Anpassen des 3D-Meshes

Das 3D-Mesh wird in der Update() Methode des XNA Frameworks angepasst. Dazu werden die Berechnungen aus Abschnitt 3.3 in mehrere, separate Teile untergliedert. Zudem wird der Entzerrungsprozess auf mehrere Frames verteilt. Dadurch wird auf jeden einzelnen Frame lediglich ein Teil des Entzerrungsprozesses angewandt. Die gesamte Projektionsentzerrung erstreckt sich somit über mehrere Frames. Der Vorteil dieser Vorgehensweise ist, dass bei einem aufwendigeren Spiel oder einer Animation die einen großen Rechenaufwand benötigt, noch genügend FPS (Frames per Second) erzeugt werden können, damit das Spiel bzw. die Animation flüssig erscheint. Für jeden Frame wird die Rechenzeit, die durch die Projektionsentzerrung zusätzlich beansprucht wird, nur geringfügig erhöht. Würde der Entzerrungsprozess auf jeden Frame vollständig angewandt werden, so würde die Berechnung eines Frames wesentlich mehr Zeit in Anspruch nehmen. Der Entzerrungsprozess ist in folgende Teile untergliedert, wobei manche Teil selbst wieder in bis zu 3 Teile untergliedert sind. Jede dieser Feinunterteilungen wird im Zuge eines Frames berechnet.

- **Initialisierung des Entzerrungsprozesses**
  Hier werden den Matrizen, die im weiteren Verlauf der Projektionsentzerrung genutzt werden, mit Null-Werten initialisiert. Damit werden die Daten eines

vorherigen Entzerrungsprozesses gelöscht, um falsche Ergebnissen zu vermeiden. Weiterhin werden die Tiefendaten der Kinect auf die Grafikkarte geladen.
Feinunterteilungen: 1
beanspruchte Zeit: 9ms

- **Erstellen des 3D-Modells**
  Hier finden die Berechnungen des 3D-Modells statt, die in Abschnitt 3.3 beschrieben sind. Zu jedem Kinectpixel wird eine Weltkoordinate berechnet sowie ein Projektorpixel.
  Feinunterteilungen: 2
  beanspruchte Zeit: 8+8=16ms

- **Zuordnen von Weltkoordinaten zu Projektorpixeln**
  Den Mesh-Knotenpunkten werden hier die Weltkoordinaten zugeordnet, die im vorherigen Teil, für das zugehörige Projektorpixel berechnet wurden.
  Feinunterteilungen: 1
  beanspruchte Zeit: 9ms

- **Interpolation**
  Hier werden Weltkoordinaten für die Pixel berechnet, für die bisher kein Wert gefunden wurde. Die genaue Vorgehensweise ist in 4.2.4 beschrieben.
  Feinunterteilungen: 3
  beanspruchte Zeit: 7+9+5=21ms

Der gesamte Entzerrungsprozess beansprucht somit, auf der Hardware des Prototyps, 55ms und wird auf 7 Frames aufgeteilt.

4.2.3 Parallelisierung

Die Berechnungen zur Projektionsentzerrung erfolgen wie in Abschnitt 3.3 beschrieben, jedoch werden die einzelnen Schritte parallelisiert. So muss die Berechnung der Weltkoordinaten im 3D-Modell nicht sequenziell durchgeführt werden und läuft um ein Vielfaches schneller ab. Die Parallelisierung wird mit der openCV Bibliothek realisiert, die Funktionen zum Berechnen von Matrizen bereitstellt. Beispielsweise soll wie in (3.11) dargestellt, ein *s-Wert* berechnet werden. Dieser ist Abhängig von der intrinsischen Matrix, der extrinsischen Matrix, vom jeweiligen Kinectpixel zu dem der s-Wert gehört und von den Tiefendaten dieses Pixels. Da intrinsische- und extrinsische Matrix für ein Kinectpixel fix sind, können diese Daten für jedes Pixel vorberechnet werden. Diese Werte werden in einer $1 \times m * n$ Matrix K für jedes Pixel gespeichert, wobei m die horizontale Kinect-Auflösung und n die vertikale Kinect-Auflösung ist. Während die Projektorpixel einer zweidimensionalen Anordnung folgen, wird diese auf eine eindimensionale Anordnung heruntergebrochen, indem alle Zeilen hintereinandergereiht werden. Werden nun die Tiefendaten auch in einer $1 \times m * n$ Matrix L hinterlegt, kann der s-Wert durch komponentenweise Division $s = L/K$ bzw. $s_i = L_i/K_i$ für jedes Kinect-Pixel $pixel_i$ errechnet werden. Die Daten der Matrix K an der Stelle i müssen also zu demselben Pixel gehören

wie die Daten der Matrix L an der Stelle i. Eine solche Komponentenweise Division erfolgt für jedes Pixel unabhängig von den Restlichen und kann durch die Emgu-Funktion GpuInvoke.Divide(L, K, s, 1, IntPtr.Zero) realisiert werden. Diese führt eine parallele, komponentenweise Division auf der Grafikkarte durch und speichert das Ergebnis in einer $1 \times m * n$ Matrix s. Komponentenweise Addition, Subtraktion oder Multiplikation können analog durchgeführt werden.

Soll eine nicht komponentenweise Matrizenmultiplikation angewandt werden, kann diese auch für jedes Pixel unabhängig von den Restlichen berechnet werden. Beispielsweise wird in (3.7) die inverse der intrinsischen Matrix I^{-1} mit einem Pixel $(u, v, 1)^T$ multipliziert. Soll dies für alle Pixel parallel durchgeführt werden, so wird eine $3 \times m * n$ Matrix P erstellt, wobei wieder eine eindimensionale Pixelanordnung genutzt wird. Die Spalte $[(j * horizontaleAuflösung) + i]$ mit $i < m$ und $j < n$ enthält somit die Koordinaten $(i, j, 1)$ des Pixels $p_{i,j}$, wobei die 1 durch die Verwendung von homogenen Koordinaten zustande kommt. Mit der openCV-Funktion *GpuInvoke.Gemm()* kann die gewünschte Multiplikation für jedes Pixel durchgeführt werden. Es ergibt sich:

$$P = \begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} * \begin{pmatrix} 0 & 1 & ... & (m-1) & 0 & ... & i & ... & (m-1) \\ 0 & 0 & ... & 0 & 1 & ... & j & ... & (n-1) \\ 1 & 1 & ... & 1 & 1 & ... & 1 & ... & 1 \end{pmatrix} \tag{4.1}$$

Mithilfe solcher Matrizenberechnungen wird das 3D-Modell aus Abschnitt 3.3 parallelisiert. Aus Effizienzgründen werden möglichst viele Matrizen und Zwischenwerte vorberechnet. Zusammen mit den Ressourcen, die der Interpolierungsprozess beansprucht, sollte eine Grafikkarte mit 512MB RAM oder mehr eingesetzt werden.

4.2.4 Interpolation

Wie bereits in Abschnitt 3.3.3 beschrieben, gibt es Projektorpixel, zu denen keine Weltkoordinate gefunden werden kann (invalide Stellen). Die Interpolation findet auf der Grafikkarte statt. Als Eingabe liegt eine $m \times n$ Matrix vor, bei der an der Stelle (i, j) die Weltkoordinaten des Projektorpixels $pixel_{i,j}$ stehen, auf die das Pixel projizieren wird. Für invalide Stellen enthält die Matrix Nullwerte: $pixel_{i,j} = 0$. X, Y und Z Werte einer Weltkoordinate werden separat betrachtet, sodass letztendlich 3 Matrizen vorliegen. $Matrix_X$ enthält die X-Werte der Weltkoordinaten, $Matrix_Y$ die Y-Werte und $Matrix_Z$ die Z-Werte für die entsprechenden Projektorpixel. Auf diese Matrizen werden folgende Operationen angewandt:

- **Zählen von validen Stellen**
  Zähle die Stellen in der Umgebung eines Pixels, die valide Einträge enthalten. Das Ergebnis für Pixel $pixel_{i,j}$ wird in der $m \times n$ Matrix $M1$ an der Stelle (i, j) gespeichert. Zu einer $m \times n$-Umgebung eines Pixels $p_{i,j}$ gehören alle Pixel $p'_{x,y}$ mit $|x - i| <= m$ und $|y - j| <= n$. Die Entzerrungssoftware nutzt standardmäßig eine 2×2-Umgebung.

- **Summieren von validen Stellen**
  Addiere alle Pixel, die zu einer Umgebung gehören und speichere das Ergebnis in der $m \times n$ Matrix M_{sum}. Diese enthält an der Stelle i die Summe der Werte der Umgebungspixel.

- **Bilden des Mittelwertes**
  Speichere den Mittelwert der Umgebungspixel in der $m \times n$ Matrix M_{avg}. $M_{avg} = M_{sum}/M_1$, wobei eine Komponentenweise Division durchgeführt wird.

- **Herausfiltern von validen Stellen.**
  Erstelle eine $m \times n$ Matrix M_{filter} mit: $M_{filter}(i, j) = 1$ für alle invaliden Stellen und $M_{filter}(i, j) = 0$ für alle validen Stellen. Multipliziere M_{filter} mit M_{avg} um die Mittelwerte an den validen Stellen herauszufiltern. Das Ergebnis wird in M_{filter} gespeichert.

- **Auffüllen der Mesh Lücken**
  Durch Addition von M_{filter} zur ursprünglichen Matrix ($Matrix_X$, $Matrix_Y$ oder $Matrix_Z$) werden die invaliden Stellen mit den berechneten Mittelwerten aufgefüllt.

Zum Herausfiltern von validen bzw. invaliden Stellen bietet openCV eine *Threshold()* Funktion an. Zum Aufsummieren von Werten in einer Umgebung kann die openCV-Funktion *Filter2D()* verwendet werden, die einen Filterkernel benutzt, der überall aus Einsen besteht. Diese kann auch zum Zählen von validen bzw. invaliden Stellen in einer Umgebung genutzt werden. Dazu wird zunächst, mithilfe von Threshold, eine Matrix erstellt die nur Einsen und Nullen enthält. Auf diese wird anschließend die Filter2D()-Funktion angewandt.

4.3 Kalibrierung von Projektor und Kinect

Dieser Vorgang muss einmalig durchgeführt werden, solange sich Position und Ausrichtung von Kinect und Projektor zueinander nicht verändern. Die Kalibrierung erfolgt in zwei Schritten. Zunächst wird eine Kamerakalibrierung durchgeführt. Diese bestimmt die extrinsischen und intrinsischen Parameter der Kinect und kann aktiviert werden, indem die Variable *calibrateCamera*, der UndistortedGame-Klasse auf *true* gesetzt wird. Anschließend wird der Projektor kalibriert, indem die Variable *calibrateProjector* auf *true* gesetzt wird. Im Folgenden wird die Durchführung der Kalibrierung genauer besprochen und auf Implementationsdetails eingegangen.

4.3.1 Kamerakalibrierung

Zum Kalibrieren der Kamera muss ein Blatt mit einem Schachbrettmuster vor die Kinect gehalten werden. Das Blatt darf dabei nicht gebogen oder gefaltet sein und wird idealerweise auf eine nicht flexible Platte geklebt. Es werden nun mehrere Bilder des Schachbrettmusters wie folgt analysiert.

Das Kamerabild der Kinect wird in schwarz-weiß gerendert. Die Knotenpunkte des Schachbrettmusters, an denen 4 Quadrate zusammenlaufen, werden im Kamerabild sichtbar gemacht, sobald openCV diese erkennt. Dazu wird die OpenCV-Funktion *findChessboardCorners()* genutzt. Alle erkannten 2D-Positionen der Knotenpunkte im i-ten Kinectbild werden in einem Array A_1 an der Stelle i gespeichert. $A_1[i]$ enthält somit alle Knotenpunkte des i-te, erkannten Schachbrettmusters. Zusätzlich zu diesen Knotenpunkten werden zu jedem erkannten Schachbrettmuster sogenannte Objektpunkte in einem Array A_2 gespeichert. Ein Objektpunkt stellt die 3D-Position im Raum zu einem Knotenpunkt dar. $A_2[i]$ enthält also die Objektpunkte zum i-te erkannten Schachbrettmuster. Weiterhin gehört der j-te Knotenpunkt des i-ten Schachbrettmusters zum j-ten Objektpunkt des i-ten Schachbrettmusters: $A_1[i][j]$ gehört zu $A_2[i][j]$. Diese Daten werden für mehrere Schachbrettmuster gespeichert, die sich an verschiedenen Positionen im Raum befinden und deren Ausrichtung möglichst verschieden sein sollte. Sobald die Knotenpunkte im Kamerabild sichtbar werden, kann das Schachbrettmuster verschoben werden. Wurden genügend Daten gesammelt, wird schließlich mit der openCV-Funktion *calibrateCamera()* die intrinsische Matrix berechnet. Hierzu müssen der Funktion die gespeicherten Knotenpunkte und die zugehörigen Objektpunkte übergeben werden. Für den ersten Objektpunkt (oben links) wird dabei angenommen, dass sich dieser im Ursprung des dreidimensionalen Raumes befindet. Für einen Objektpunkt $O_{i,j}$, der zum Knotenpunkt $K_{i,j}$ gehört, ergibt sich somit: $O_{i,j}(X, Y, Z) = (i * a, j * a, 0)$, wobei a die physikalische Breite und Höhe eines Quadrates des Schachbrettmusters ist. Alle Objektpunkte befinden sich somit in der X-Y-Ebene.

4.3.2 Projektorkalibrierung

Zur Projektorkalibrierung müssen ebenfalls mehrere Bilder analysiert werden und die Position und die Ausrichtung eines Schachbrettmusters verändert werden. Dazu wird ähnlich wie bei der Kamerakalibrierung vorgegangen und die openCV-Funktion calibrateCamera() genutzt um die intrinsischen Parameter des Projektors zu bestimmen. Diese nutzt die Objektpunkte und Knotenpunkte eines Schachbrettmusters, das auf eine nicht flexible Platte projiziert wird. Die Knotenpunkte werden nun nicht mehr mithilfe des Kinectbildes erstellt, sondern werden mit findChessboardCorners() aus dem zu projizierenden Schachbrettmuster berechnet. Um die 3D-Koordinaten eines Objektpunktes zu erhalten, werden folgende Schritte abwechselnd durchgeführt, bis genügend Daten gesammelt wurden:

- **Schritt 1: Bestimmen der extrinsischen Kameraparameter**
  Auf die Platte, auf die in Schritt 2 projiziert wird, wird das Schachbrettmuster aus Abschnitt 4.3.1 gehalten. Mithilfe der openCV-Funktion findChessboard-Corners() werden wie in 4.3.1 Knotenpunkte und Objektpunkte zu einem Kinectbild erstellt. Diese können nun der Emgu-Funktion *FindExtrinsicCameraParams2()* übergeben werden, die mithilfe der intrinsischen Matrix, die in 4.3.1 berechnet wurde, die extrinsischen Parameter der Kinect bestimmt.

- **Schritt 2: Berechnen der Objektkoordinaten**
  Hier wird ein Schachbrettmuster auf dieselbe Platte, die in Schritt 1 verwendet wird, projiziert. Mithilfe von openCV werden dessen Knotenpunkte im Kinectbild bestimmt. Die extrinsische Matrix der Kinect wurde so erstellt, dass sich alle projizierten Objektpunkte in der X-Y-Ebene befinden. Da die intrinsische und extrinsische Kinect-Matrix sowie die Z-Koordinate eines Objektpunktes ($Z = 0$) bekannt sind, können die Gleichungen (3.11) und (3.12) aus Abschnitt 3.3 so umgestellt werden, dass die Y- und X-Koordinaten eines Objektpunktes zu einem Knotenpunkt berechnet werden können.

4.4 Beispiel

Abb. 4.3 zeigt, wie eine Projektion während des Bewegens der Kinect-Projektor-Einheit entzerrt wird. Im linken Teil ist die Projektion in der Ausgangsposition zu sehen. Im mittleren Bild wurde die Kinect-Projektor-Einheit in horizontaler Richtung nach rechts und in vertikaler Richtung nach oben gedreht. Dadurch wird die rechte Bildkante größer als die Linke und die obere Bildkante größer als die Untere dargestellt. Abb. 4.3 rechts zeigt die Projektion, die automatisch durch die Software angepasst wurde. Die gesamte Projektion ist weiter nach rechts oben gerückt und die gegenüberliegenden Bildkanten sind wieder gleich groß. Die Projektion ist jedoch nicht vollständig rechteckig. Dies ist auf Bildrauschen der Kinect zurückzuführen und hängt auch mit der Qualität der Kalibrierung von Kinect und Projektor zusammen.

Abbildung 4.3: Beispiel der Projektionsentzerrung

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Es wurde eine Software entwickelt, mit der eine automatische Projektionsentzerrung realisiert werden kann, sodass ein Projektor nicht mehr orthogonal zur Projektionsfläche hin ausgerichtet werden muss. Weiterhin ist es nicht nur möglich auf flache Oberflächen zu projizieren, sondern auch auf beliebig förmige Oberflächen. Die Oberfläche darf sich dabei uneingeschränkt verändern, wobei die Projektion automatisch entsprechend angepasst wird. Diese Funktionalitäten ermöglichen eine Echtzeitentzerrung, die insbesondere bei mobilen Projektoren von Bedeutung ist. Der verwendete Prototyp ist dabei weder an einen bestimmten Raum noch an sonstige vordefinierte Oberflächen gebunden und kann mobil eingesetzt werden. Zum Entzerren einer Projektion wurde die Oberfläche, auf die projiziert werden soll, mithilfe einer Kinect gescannt und aus den Kinect -Tiefendaten wird ein 3D-Modell erstellt. Anschließend wird das zu projizierende Bild, mittels Texturierung des 3D-Modells, so verändert, dass die Projektion unverzerrt erscheint.

5.2 Ausblick

Das Entzerrungstoolkit kann als Basis für alle Anwendungen benutzt werden, die eine entzerrungsfreie, mobile Projektion benötigen. Wie in ShelfTorchlight [11] von Löchtefeld et. al. können Produktinformationen zu einem Artikel eingeblendet werden, wobei die Projektion automatisch angepasst wird. Weiterhin könnten Algorithmen zur Objekterkennung verwendet werden, um ein Objekt von dessen Hintergrund zu segmentieren, sodass nur auf das Objekt projiziert wird. Mithilfe des 3D-Modells kann eine Anwendung Informationen über die dreidimensionale Umgebung erhalten und entsprechende Aktionen ausführen. Beispielsweise kann eine Spielfigur, entsprechend den Tiefendaten im 3D-Modell, mit unterschiedlicher Geschwindigkeit laufen. Als Erweiterung können auch Daten eines Accelerometers verwendet werden, um den Entzerrungsprozess bei minimalen Änderungen zu beschleunigen oder um Energie zu sparen. Um das Zittern der Hand auszugleichen, muss so nicht der rechenintensivere Entzerrungsprozess mittels Kinect durchgeführt werden.

Literaturverzeichnis

[1] aaxa Technologies (Online; letzter Zugriff: September 2013). axaa l1 v2 laser pico projector. URL http://www.aaxatech.com/products/l1_laser_pico_projector.htm.

[2] Andrade-Cetto, J. & Sanfeliu, A. (2006). *Environment Learning for Indoor Mobile Robots*. Springer Tracts in Advanced Robotics. Springer.

[3] Apple (2012). iphone 5 specifications. URL http://www.apple.com/de/iphone/specs.html.

[4] Curless, B. & Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96 (pp. 303–312). New York, NY, USA: ACM.

[5] Dao, V. N., Hosoi, K., & Sugimoto, M. (2007). A semi-automatic realtime calibration technique for a handheld projector. URL http://dl.acm.org/citation.cfm?id=1315189.

[6] Emgu (2012). Emgu cv. URL http://www.emgu.com/wiki/index.php/Version_History#Emgu.CV-2.4.2.

[7] Garage, W. & Intel (2013 version 2.4.6). opencv. URL http://opencv.org.

[8] Hartley, R. & Zisserman, A. (2002). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition.

[9] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., & Fitzgibbon, A. (2011). Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. URL http://research.microsoft.com/pubs/155416/kinectfusion-uist-comp.pdf.

[10] Lee, J. C., Dietz, P. H., Maynes-Aminzade, D., Raskar, R., & Hudson, S. E. (2004). Automatic projector calibration with embedded light sensors. URL http://johnnylee.net/academic/proj4.pdf.

[11] Löchtefeld, M., Gehring, S., Schöning, J., & Krüger, A. (2010). Shelftorchlight: Augmenting a shelf using a camera projector unit. In *Adjunct Proceedings of the Eighth International Conference on Pervasive Computing*: Springer Lecture Notes in Computer Science.

[12] Microsoft (Online; letzter Zugriff: September 2013). Kinect for windows sensor components and specifications. URL `http://msdn.microsoft.com/en-us/library/jj131033.aspx`.

[13] Molyneaux, D., Izadi, S., Kim, D., Hilliges, O., Hodges, S., Cao, X., Butler, A., & Gellersen, H. (2012). Interactive environment-aware handheld projectors for pervasive computing spaces. URL `http://research.microsoft.com/pubs/171707/Interactive%20Environment-aware%20Handheld%20Projectors%20for%20Pervasive%20Computing%20Spaces%20Pervasive2012.pdf`.

[14] Nvidia (2012). Cuda toolkit 5.0. URL `https://developer.nvidia.com/cuda-toolkit-50-archive`.

[15] Pinhanez, C. (2001). The everywhere displays projector: A device to create ubiquitous graphical interfaces. URL `http://www.research.ibm.com/people/p/pinhanez/publications/ubicomp01.pdf`.

[16] Piper, B., Ratti, C., & Ishii, H. (2002). Illuminating clay: A 3-d tangible interface for landscape analysis. URL `http://tmg-trackr.media.mit.edu:8020/SuperContainer/RawData/Papers/266-Illuminating%20Clay%20A%203/Published/PDF`.

[17] Rapp, S. (2010). Spotlight navigation: a pioneering user interface for mobile projection. URL `http://eis.comp.lancs.ac.uk/workshops/ubiproject2010/pdf/rapp_ubiprojection2010.pdf`.

[18] Raskar, R., Beardsley, P., van Baar, J., Wang, Y., Dietz, P., Lee, J., Leigh, D., & Willwacher, T. (2006). Rfig lamps: interacting with a self-describing world via photosensing wireless tags and projectors. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06 New York, NY, USA: ACM.

[19] Samsung (2010). Samsung galaxy beam specifications. URL `http://www.samsung.com/global/microsite/galaxybeam/spec.html`.

[20] Steimle, J., Jordt, A., & Maes, P. (2013). Flexpad: highly flexible bending interactions for projected handheld displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13 (pp. 237–246). New York, NY, USA: ACM.

[21] Wikitude (2008). Wikitude world browser. URL `http://www.wikitude.com/app/`.