UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Konstruktion eines Gonioreflektometers zum anschließenden Aufnehmen und Rendern von BRDFs

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Felix-León Schröder

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	M. Sc. Kevin Keul (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Juni 2019

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Zusammenfassung

Ein Gonioreflektometer ist ein Gerät zum Vermessen der Reflexionseigenschaften von Materialien. Ein solche Apparatur wird in dieser Arbeit mit handelsüblichen Bauteilen gebaut. Dafür werden drei Schrittmotoren und 809 Leuchtdioden mit einem Arduino-Mikrocontroller gesteuert. Als Reflexionsdaten werden RGB-Bilder mit einer industriellen Kamera aufgenommen. Zusätzlich wird eine Steuersoftware für verschiedene Aufnahmeprogramme sowie ein Renderer zum Anzeigen der vermessenen Materialien implementiert. Somit können komplette bidirektionale Reflektanz-Verteilungsfunktionen (BRDFs) aufgenommen und gerendert werden, wodurch selbst komplizierte anisotrope Materialeigenschaften repräsentierbar sind. Die Qualität der Ergebnisse ist aufgrund von Schattierungen zwar Artefakt-behaftet, jedoch können diese Artefakte durch entsprechende Algorithmen wie Inpainting weitestgehend behoben werden. Außerdem wurde das Gonioreflektometer auf andere Anwendungen übertragen. So sind ohne Veränderungen am Gerät auch 3D-Scans, Lichtfeldaufnahmen und Light-Staging möglich. Auch die Qualität der Ergebnisse dieser Aufnahmeverfahren entspricht den Erwartungen im positiven Sinne. Somit ist das in dieser Arbeit gebaute Gonioreflektometer im Vergleich zu anderen Publikationen eine breit anwendbare und kostengünstige Alternative.

Abstract

A gonioreflectometer is a device to measure the reflection properties of arbitrary materials. In this work, such an apparatus is being built from easily obtainable parts. Therefore three stepper-motors and 809 light-emitting diodes are controlled by an Arduino microcontroller. RGB-images are captured with an industrial camera which serve as refelction data. Furthermore, a control software with several capture programs and a renderer for displaying the measured materials are implemented. These allow capturing and rendering entire bidirectional reflection distribution functions (BRDFs) by which also complex anisotropic material properties can be represented. Although the quality of the results has some artifacts due to shadows of the camera, these artifacts can be largely removed by using special algorithms like inpainting. In addition, the goniorefelctometer is applied to other use cases. One can perform 3D scans, light field capturing and light staging without altering the construction. The quality of these processes also meet the expectations in a positive way. Thus, the gonioreflectometer built in this work can be seen as a widely applicable and economical alternative to other publications.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Grundlagen . . . 1
  - 1.2.1 BRDFs . . . 2
  - 1.2.2 Gonioreflektometer . . . 3
  - 1.2.3 Schrittmotoren . . . 6
  - 1.2.4 Leuchtdioden . . . 10
  - 1.2.5 Arduino . . . 11

2 Aufbau des Gonioreflektometers **13**
- 2.1 Konstruktion des Kameraarms . . . 14
- 2.2 Konstruktion der Lichtkuppel . . . 16
- 2.3 Anschließen der elektronischen Bauteile . . . 19
  - 2.3.1 Schrittmotoren . . . 20
  - 2.3.2 LED-Streifen . . . 21
  - 2.3.3 Kamera . . . 22

3 Aufnahme von BRDFs **23**
- 3.1 Programmierung des Arduino-Controllers . . . 23
  - 3.1.1 Programmierung der Motoransteuerung . . . 24
  - 3.1.2 Programmierung der LED-Ansteuerung . . . 25
- 3.2 Befehlssatz zur Ansteuerung über USB . . . 27
- 3.3 Konzeption der Ansteuerungs-Software . . . 28
- 3.4 Kalibrierung . . . 30
  - 3.4.1 Kameraarm . . . 30
  - 3.4.2 Kamera . . . 30
  - 3.4.3 Lichtquellen . . . 31
  - 3.4.4 Schatten-Maskierung . . . 34
- 3.5 Aufnahmeprogramme . . . 35
  - 3.5.1 Diffuse BRDF (1 DoF) . . . 36
  - 3.5.2 Rotationssymmetrische BRDF (2 DoF) . . . 38
  - 3.5.3 Halb-Isotrope BRDF (3 DoF) . . . 39
  - 3.5.4 Vollständige BRDF (4 DoF) . . . 39

4 Rendering der aufgenommenen BRDFs **39**
- 4.1 Framework . . . 40
- 4.2 Laden der Bilddaten . . . 42
- 4.3 Anzeigen der Bilddaten . . . 43
  - 4.3.1 Farbtextur . . . 44
  - 4.3.2 Grauwerttextur . . . 45
  - 4.3.3 Mittlere Farbe . . . 45
  - 4.3.4 Mittlere Helligkeit . . . 45

5 Weitere Aufnahmeverfahren **45**
5.1 3D-Scan 46
5.2 Lichtfeld-Scan 48
5.3 Light-Stage 48

6 Evaluation **49**
6.1 BRDF-Messung 49
6.2 Weitere Aufnahmenverfahren 53
6.2.1 3D-Scan 53
6.2.2 Lichtfelder 55
6.2.3 Light-Stage 55

7 Fazit **57**

8 Ausblick **57**

1 Einleitung

Diese Masterarbeit beschäftigt sich mit dem Aufbau eines Gonioreflektometers, der Aufnahme von realen Materialien und BRDFs, sowie dem Rendern der aufgenommenen Daten auf virtuellen Objekten.

Als Erstes werden die Grundlagen für den Aufbau und das Rendering gelegt. Anschließend wird der Aufbau und die Ansteuerungssoftware des Gonioreflektometers beschrieben. Weiterhin wird die Implementierung des Aufnahme- und des Renderprogramms erläutert. Abschließend werden weitere Aufnahmemethoden diskutiert und die Ergebnisse evaluiert.

1.1 Motivation

Materialien sind überall. Jedes Objekt in der Welt hat ein Material, welches durch bestimmte Farben und Reflexionseigenschaften charakterisiert ist. Eine wichtige Frage ist daher: Wie können Materialien möglichst detailgetreu und realitätsnah beschrieben werden?

Eine Lösung dafür bieten Gonioreflektometer. Ein solches Gerät kann Materialien, besonders dessen Reflexionseigenschaften, vermessen. In dieser Arbeit soll ein solches Gerät gebaut und gesteuert werden.

Doch was kann man mit einem vermessenen und digitalisierten Material anfangen? Eine mögliche Anwendung ist das realistische Rendern von virtuellen Objekten. Wird ein vermessenes Material beispielsweise auf ein 3D-Modell angewendet, so erhält man eine sehr realistische Darstellung des Materials und somit auch des Modells. Benötigt wird also auch ein entsprechender Renderer, um aufgenommene Materialien anzuzeigen.

Leider liegen die Kosten für ein solches Gerät üblicherweise im vier- bis fünfstelligen Bereich. Es soll daher auch erforscht werden, ob es möglich ist, ein qualitativ hochwertiges Gonioreflektometer mit handelsüblichen Bauteilen für unter 500€ zu bauen und zu betreiben.

Des Weiteren soll untersucht werden, für welche weitere Anwendungsbereiche ein solches Gerät außerdem verwendet werden kann. Entsprechend muss auch für diese Anwendungszwecke die jeweilige Software konzipiert und implementiert werden.

1.2 Grundlagen

Im Folgenden werden die Grundlagen für diese Arbeit gelegt. Dabei wird zum einen auf vorangegangene Arbeiten und Publikationen zu diesem Thema eingegangen. Zum anderen werden die theoretischen Grundlagen der verwendeten Mathematik sowie den benötigten Bauteilen für den Aufbau des Gonioreflektometers besprochen.

1.2.1 BRDFs

Eines der wichtigsten Themengebiete in der Computergrafik ist die Beschreibung von verschiedenen Materialien. Dabei möchte man in der Regel eine möglichst realistische Materialdefinition erreichen, die nicht nur Farbe und Helligkeit, sondern auch verschiedene Reflexionseigenschaften abbilden kann.

Üblicherweise werden hierfür mehrdimensionale, aber stark vereinfachte Funktionen wie das Blinn-Phong-Modell oder das Cook-Torrance-Modell verwendet, die auf der mathematischen Modellierung von Mikrofacetten basieren. Hiermit können viele einfache Materialien weitgehend realistisch dargestellt werden. Allerdings handelt es sich bei diesen Modellen stets um isotrope, also rotationssymmetrische Funktionen, weshalb besonders anisotrope Materialien mit diesen Modellen schwer zu beschreiben sind.

Bidirektionale Reflexions-Verteilungsfunktionen (engl. *Bidirectional Reflection Distribution Function*, kurz *BRDF*) sind Teil der Rendering-Equation von James Kajiya aus dem Jahr 1986 [Kaj86]. Die Formel beschreibt das ausgehende Licht in eine bestimmte Richtung unter Einfluss des einfallenden Lichts aus allen Richtungen im vorderen Halbraum und den Materialeigenschaften. Sie ist der Grundstein der Computergrafik und Basis von nahezu allen realistischen Rendering-Anwendungen. Die Formel ist wie folgt definiert:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_\Omega f_r(x, \vec{\omega}', \vec{\omega}) \cdot L_i(x, \vec{\omega}') \cdot (\vec{\omega}' * \vec{n}) \, d\vec{\omega}'$$

Die einzelnen Komponenten der Formel beschreiben die folgenden Eigenschaften des Punktes x:

- $\vec{n}$: Die Normale am Punkt x
- L_e: Das vom Punkt selbst emittierte Licht in Richtung $\vec{\omega}$ (falls dieser wie z.B. eine Lampe selbst strahlt)
- $\int_\Omega$: Das Integral über alle Winkel der Hemisphäre über x um $\vec{n}$
- $f_r(x, \vec{\omega}', \vec{\omega})$: Die BRDF am Punkt x
- $L_i(x, \vec{\omega}')$: Das einfallende Licht
- $\vec{\omega}$: Die Richtung des Betrachters bzw. der Kamera
- $\vec{\omega}'$: Die Richtung des einfallenden Lichts

Das Material von x ist durch die BRDF definiert, welche abhängig von der Ein- und Austrittsrichtung des Lichts ($\vec{\omega}'$ und $\vec{\omega}$, daher „bidirektional") die Reflexionseigenschaften des Materials beschreibt.

Eine BRDF kann abhängig von den abgebildeten Materialeigenschaften unterschiedlich viele Freiheitsgrade (engl. *degrees of freedom*, kurz *DoF*) und somit verschiedene Formen bzw. Dimensionen haben. Diffuse Materialien sind nur vom Lichteinfallswinkel zur Normale abhängig und daher eindimensional.

BRDFs können aber auch beliebig komplex werden, wenn beispielsweise anisotrope (4 DoF) Materialien mit RGB-Textur (2 Textur- + 3 Farb-DoF) gespeichert werden, was insgesamt neun Freiheitsgrade ergibt.

In der Computergrafik wird die BRDF fast immer durch ein mathematisches Modell beschrieben, da dies deutlich speicherplatzeffizienter ist. Die Reflexionseigenschaften können dann zur Laufzeit mit Parametern eingestellt und im Shader berechnet werden. Die gängigsten Modelle sind das Blinn-Phong-Modell [Bli77], welches hauptsächlich abhängig von einer Glanzzahl ist, und das Cook-Torrance-Modell [CT81], welches ein Material über dessen „Rauheit" und „Metalligkeit" beschreibt. Alle diese Modelle sind isotrope Modelle und zudem stark vereinfacht, weshalb mit ihnen komplexere Materialeigenschaften wie Subsurface-Scattering oder Retroreflexion nicht darstellbar sind.

1.2.2 Gonioreflektometer

Eine Gerät zur Aufnahme von BRDFs ist das Gonioreflektometer. Die Bezeichnung setzt sich aus den Worten „Gonio" und „ Reflektometer" zusammen. „Gonio" stammt aus dem Griechischen und bedeutet Winkel, was auf die Aufnahme aus allen Richtungen hindeutet. „Reflektometer" bezeichnet die Eigenschaft, dass das Gerät die Reflektanz von Licht misst.

Ziel eines solchen Gerätes ist es, die Reflexionseigenschaften eines realen Materials zu vermessen. Eine Materialprobe kann somit „digitalisiert" und als Materialbeschreibung genutzt werden. Die dadurch erhaltene Materialdefinition kann deutlich mehr Lichteffekte beschreiben als mathematische Modelle.

In früheren Arbeiten wurden bereits verschiedene Arten von Gonioreflektometern konstruiert und evaluiert.

Eine der ersten Veröffentlichungen zu Gonioreflektometern ist der bereits 1980 veröffentlichte Artikel „Computer-controlled gonioreflectometer for the measurement of spectral reflection characteristics" [Erb80]. Hier wurde ein Gonioreflektometer entwickelt, mit dem aus allen Kamera- und Lichtwinkeln die Strahldichte gemessen werden kann. Dieses Gerät wurde speziell für die Kalibrierung von Reflexionsstandards in der Optik entwickelt. Ähnliche Geräte wurden auch in [WSJB+98] und [NMI04] vorgestellt.

In [Foo97] und [LFTW06] wurde ein Drei-Achsen-System gebaut. Hier kann die Lichtquelle um eine Achse und die Materialprobe um zwei Achsen rotiert werden. Der Sensor ist bei diesem Aufbau fest montiert und nimmt das Material über einen halb-transparenten Spiegel auf. Zusätzlich kann ein Prisma eingebaut werden, um auch spektrale Aufnahmen zu ermöglichen. Allerdings kann auch hier nur die Strahldichte gemessen werden, da keine Bilder aufgenommen werden. Das Gerät in [LFTW06] wurde speziell mit dem Ziel gebaut, den Photorealismus von computergenerierten Bildern zu erhöhen, indem Informationen über die Lichtstreuung der Oberflächen aufgenommen

wurden.

Neuere Publikationen wie [HGH06], [BNC09] und [WZ15] setzen vermehrt auf einen Roboterarm. Dabei wird die Materialprobe auf einem Fünf-Achsen-Roboterarm positioniert. Um diesen Arm kann die Lichtquelle um 360° waagerecht rotiert werden. Als Aufnahmegerät dient ein komplexes optisches System mit einem Monochromator und mehreren Sensoren für verschiedene Wellenlängenbereiche. Dieses Gonioreflektometer kann durch dem Roboterarm besonders viele Ein- und Ausfallswinkel messen und die Materialprobe sogar parallel zur Kameraebene drehen. Allerdings wird auch hier nur ein Wert pro Einstellung gemessen, also keine Textur- bzw. Bilddaten.

Einen etwas anderen Ansatz verfolgen Deschaintre et al. in [DAD+18]. Hier wird die BRDF mittels *Deep Learning* und neuronalen Netzen aus nur einem Bild abgeleitet. In [MSY09] wurde ein ellipsoider Spiegel zur Ausleuchtung der Materialprobe verwendet. Dadurch kann der Lichteinfallswinkel mit einem Projektor ohne Bewegung von Motoren und somit deutlich schneller eingestellt werden. Ben-Ezra et al. stellten in [BEWW+08] ein Gonioreflektometer vor, das komplett ohne bewegliche Teile auskommt. Es benutzt die in einer Halbkugel verbauten LEDs sowohl als Beleuchtung, als auch als Sensor.

Insgesamt lässt sich sagen, dass bisherige Publikationen zu Gonioreflektometern stets die reflektierte Strahldichte oder das reflektierte Spektrum messen. Es werden also nicht mehrere Bilder mit einer Kamera aufgenommen. Dabei verfügen die meisten vorgestellten Geräte über jeweils einen Sensor (bzw. Sensorgruppe) und eine Lichtquelle. Dies sind die beiden größten Unterschiede zu dem in dieser Arbeit vorgestellten Gonioreflektometer.

Funktionsweise Alle Gonioreflektometer arbeiten nach dem selben Prinzip: Sie nehmen ein Material bzw. eine Materialprobe aus möglichst vielen Betrachtungswinkeln und mit möglichst vielen verschiedenen Lichteinfallswinkeln auf. Je mehr Winkelkombinationen dabei abgedeckt werden, desto näher ist das virtuelle Material an der Realität.

Hauptkomponenten eines Gonioreflektometers sind also Kameras oder Helligkeitssensoren, Lichtquellen und das zu vermessende Material. Um die verschiedenen Reflexionseigenschaften des Materials zu messen, werden nun der Reihe nach jeweils ein Licht- und ein Kamerawinkel eingestellt und die Helligkeit bzw. das Bild unter diesen Einstellungen aufgenommen und gespeichert.

Die aufgenommenen Daten können dann zu einer Materialbeschreibung in Form einer BRDF genutzt werden. Hierbei können die abgespeicherten Werte entweder direkt als Look-Up-Table verwendet werden, oder es kann eine mathematische Funktion gesucht werden, die den Helligkeitsverlauf möglichst präzise annähert. Die aufgenommenen Daten können dabei Helligkeitswerte, spektrale Informationen oder Farbbilder sein.

Ein wichtiges Detail ist, dass die gesamte Aufnahme in einer Dunkelkam-

mer stattfinden sollte. Dies ist essenziell, da Umgebungslicht von anderen Lichtquellen außerhalb des Gonioreflektometers die Messwerte sehr stark verfälschen kann. Auch das Innere des Gonioreflektometers sollte möglichst schwarz bzw. nicht reflektiv sein, damit das Licht nicht innerhalb des Messraumes propagiert und die Materialprobe aus „mehreren Richtungen" beleuchtet wird.

Bauweisen In ihrer Bauweise unterscheiden sich Gonioreflektometer hauptsächlich in der Anzahl an möglichen Freiheitsgraden und der Wahl zwischen beweglichen oder statischen Komponenten. Abbildung 1 zeigt links beispielhaft links eine Bauweise mit zwei Armen und rechts schematisch die Verwendung eines halb-transparenten Spiegels.

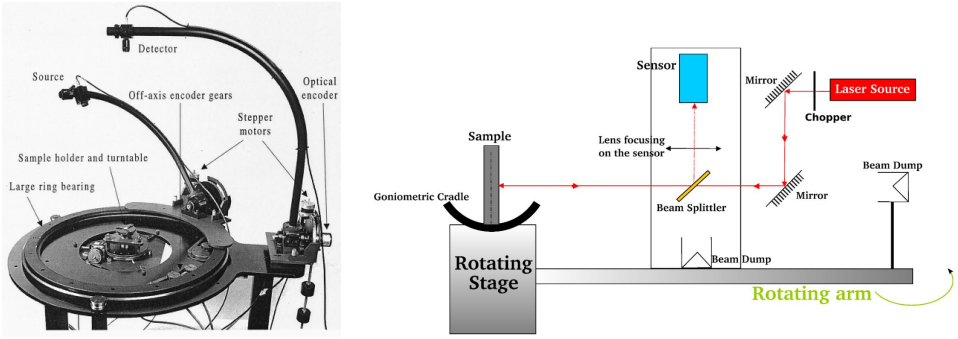


Abbildung 1: Links: Beispielhafter Aufbau eines Gonioreflektometers mit zwei beweglichen Armen für Lichtquelle und Kamera [WSJB+98], rechts: Schematische Darstellung der Funktionsweise eines Gonioreflektometers mit halb-transparentem Spiegel (engl. *Beam-Splitter*) [BPD+14]

Je nach Anzahl an Freiheitsgraden, die das Gerät messen können soll, kann beispielsweise die Kamera fest montiert werden. Auch für die Lichteinfallswinkel genügt für BRDFs mit nur einem Freiheitsgrad ein Halb- oder Viertelkreis. Je weniger Freiheitsgrade, desto weniger bewegliche Teile (und somit auch elektronische Bauteile) werden benötigt.

Die Wahl zwischen wenigen beweglichen oder vielen statischen Komponenten ist hingegen schwieriger. Sowohl für die Kamera als auch für die Lichtquellen gilt: Es kann entweder eine solche Komponente an einem beweglichen Arm montiert oder viele statisch in einer Halbkugel angebracht werden. Aus Kostengründen ist die Variante mit zwei beweglichen Armen für Kamera und Lichtquelle die am weitesten verbreitete Bauweise. Dies hat außerdem den Vorteil, dass die Winkelauflösung des Gonioreflektometers hier von der Präzision der Motoren abhängt und daher deutlich flexibler eingestellt werden kann, als bei statischen Bauweisen. Nachteilhaft ist hierbei jedoch, dass die Aufnahmen länger dauern, da das Bewegen der Komponenten mehr Zeit in Anspruch nimmt.

Ein weiterer Unterschied ist die Art der Bewegung der Materialprobe. Diese kann mittels einer Drehscheibe um sich selbst gedreht werden. Alternativ können jedoch auch die Kamera- bzw. Lichtarme um die vertikale Achse des Objekts herum rotiert werden.

Oft kommt zusätzlich zu den bisher genannten Bauteilen ein halb-durchlässiger Spiegel zum Einsatz. Dieser wird zwischen der Materialprobe und der Lichtquelle platziert. Die Kamera nimmt das Bild im Spiegel auf, während gleichzeitig Licht durch den Spiegel auf die Materialprobe fallen kann. Dies verhindert Verschattungsartefakte, die entstehen, wenn sich die Kamera zwischen Lichtquelle und Materialprobe befindet. Dieses Problem wird in Abschnitt 3.4.4 genauer untersucht.

Möchte man nicht nur die Reflexionseigenschaften, sonder auch die Transmissionseigenschaften eines Materials messen, so muss die gesamte Messapparatur nicht nur im oberen Halbraum, sondern in allen Richtungen um die Materialprobe herum messen. Kamera und Lichtquelle müssen also auch unterhalb der Materialprobe positioniert werden können, um hindurchgehendes Licht messen zu können.

1.2.3 Schrittmotoren

Schrittmotoren sind aufgrund ihrer sehr genauen Bewegungen ein wichtiger Bestandteil vieler Präzisionsgeräte. Diese präzisen Bewegungen können dabei sogar ohne zusätzliche Sensoren zur Positionsbestimmung ausgeführt werden. Im Folgenden wird diese Art von Motoren genauer beschrieben. Da in dieser Arbeit nur bipolare Schrittmotoren verbaut wurden, wird der Fokus auf diese spezielle Art vom Schrittmotoren gelegt.

Abbildung 2: Links: Schrittmotor mit transparenter Abdeckung, Mitte: Ausgebauter Rotor mit zwei gezahnten und magnetischen Weicheisenkränzen, rechts: Schematische Anordnung der Statoren und des Rotors [Sch]

Aufbau Ein Schrittmotor besteht im Allgemeinen aus einem beweglichen und magnetischen Rotor (die sich drehende Komponente), welcher von mehreren Spulen umgeben ist. Die Spulen dienen zur Erzeugung eines Magnetfeldes und werden Statoren genannt, da sie sich nicht bewegen. Anders als beim

„klassischen" Elektromotor bewegen sich die elektronischen Komponenten beim Schrittmotor also nicht, was zu einer erhöhten Lebensdauer führt.

Ein weiterer Unterschied zu normalen Elektromotoren ist, dass die Spulen einzeln oder in Gruppen ansteuerbar und gezahnt sind. Daher haben Schrittmotoren nicht nur einen Plus- und einen Minuspol, sondern in der Regel mehrere (bei bipolaren Schrittmotoren vier) Anschlüsse.

Der Rotor besteht aus einem Dauermagneten. Um die mögliche Schrittanzahl zu erhöhen, wird nicht nur ein Magnet, sondern sehr viele kleine stäbchenförmige Magneten in Form von zwei gezahnten Weicheisenkränzen verbaut, wie in Abbildung 2 (Mitte) zu sehen ist. Diese Bauweise nennt man auch Hybridschrittmotor, da es sich um eine Kombination aus Reluktanz-Schrittmotor und Permanentmagnet-Schrittmotor handelt.

Die üblicherweise acht Statoren werden in zwei Gruppen eingeteilt, woher sich auch die Bezeichnung „bipolar" ableitet. Dabei haben gegenüberliegende Statoren immer die gleiche Polung und somit auch die gleiche magnetische Richtung in Bezug zum Rotor. Die nicht gegenüberliegenden Statoren einer Gruppe haben folglich unterschiedliche Polungen. Des Weiteren sind die Statoren so angeordnet, dass niemals zwei Statoren der selben Gruppe nebeneinander liegen (siehe Abbildung 2 rechts).

Funktionsweise Bei einem Schrittmotor richtet sich der Rotor immer entsprechend dem von den Statoren erzeugten Magnetfeld aus. Der Rotor besteht wie oben beschrieben aus zwei gezahnten Weicheisenkränzen, von welchen einer den magnetischen Nord- und der andere den magnetischen Südpol darstellt. Die Zähne dieser Eisenkränze sind für zusätzliche Präzision um einen halben Schritt versetzt.

Die Anzahl der Statorzähne ist immer um zwei geringer als die der Rotorzähne. Üblicherweise hat der Rotor also 50 und die Statoren insgesamt 48 Zähne. Dadurch sind immer zwei Statoren an den Zähnen des Rotors ausgerichtet. Die anderen Statorzähne liegen entsprechend nur halb oder gar nicht über den Rotorzähnen. Abbildung 2 veranschaulicht die Lage der Komponenten.

Werden die Statoren der ersten Gruppe mit Strom versorgt, richtet sich der Rotor so aus, dass gleiche magnetische Pole sich abstoßen und unterschiedliche sich anziehen. Wird nun die zweite Gruppe der Statoren mit Strom versorgt und die erste abgeschaltet, so bewegt sich der Rotor um ein Viertel des Winkels eines Rotorzahns. Da der Rotor wie oben beschrieben üblicherweise 50 Zähne aufweist, erhält man eine Schrittweite von 1.8° bzw. 200 Schritten pro Umdrehung.

Durch abwechselnde Ansteuerung der beiden Statorgruppen wird der Rotor also in kleinen Schritten weiterbewegt. Die Drehrichtung hängt dabei von der Polarität ab, mit der die Statoren angesteuert werden. Je schneller die aktive Statorgruppe oszilliert, desto schneller bewegt sich der Rotor.

Weiterhin wirkt es sich günstig aus, dass eine Statorgruppe, solange sie mit Strom versorgt wird, den Rotor in seiner aktuellen Position hält. Der Rotor kann somit nur durch eine starke Krafteinwirkung von außen bewegt werden. Soll der Rotor in einer gewissen Position gehalten werden, ist dies mit Schrittmotoren, anders als mit herkömmlichen Elektromotoren, möglich.

Signal Wie oben beschrieben, müssen die Statorgruppen (A und B) immer abwechselnd mit Strom versorgt werden. Außerdem muss die Polung des Stroms pro Statorgruppe ebenfalls alternieren (+ und -). Ein Signalzyklus besteht also aus vier einzelnen Signalen: A+ → B- → A- → B+. Abbildung 3 zeigt im ersten Graphen den Signalverlauf.

Die Amplitude des Signals definiert das Dreh- und Haltemoment des Schrittmotors. Die Frequenz bestimmt hingegen die Bewegungsgeschwindigkeit des Rotors.

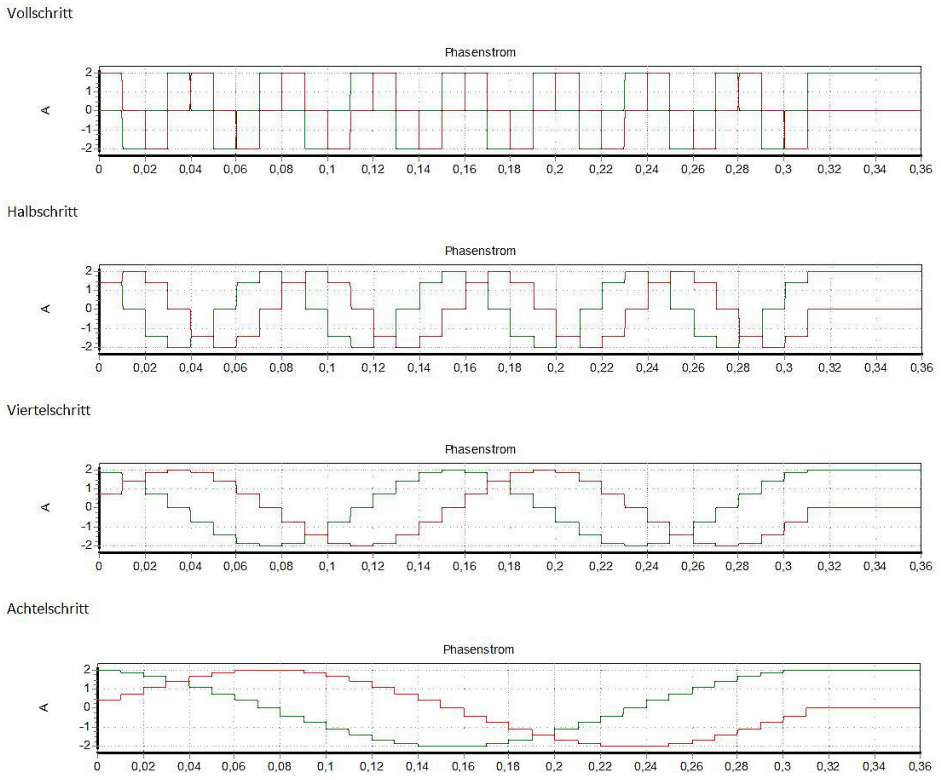


Abbildung 3: Signal zur Ansteuerung eines bipolaren Schrittmotors mit normalem Signal und Microstepping [Sch]

Microstepping In einigen Anwendungsfällen reicht die Auflösung von 1.8° pro Schritt nicht aus. Durch geschicktes Abändern des Signals kann ohne Veränderungen an der Hardware die Präzision um ein Vielfaches gesteigert

werden. Dieses Verfahren nennt sich Microstepping und kann in verschiedenem Ausmaß angewandt werden.

Beim Microstepping werden die Statoren nicht einfach nur an- und ausgeschaltet. Stattdessen wird ein Signal erzeugt, welches den Rotor auch auf eine Position zwischen zwei Schritten setzen kann. Dafür wird die Leistung der einen Statorgruppe schrittweise verringert, während die der anderen entsprechend erhöht wird (siehe Abbildung 3, Graphen 2 bis 4).

Mögliche Schrittunterteilungen sind in der Regel Halb-, Viertel-, Achtel- und Sechzehntel-Schritte. Mit der maximalen Schrittunterteilung von 1/16 können also pro Umdrehung 3200 Schritte bzw. 0.1125° pro Schritt erreicht werden.

Meistens wird für diese Technik „Pulse-Width-Modulation" (PWM) benutzt. Dies stellt zwar keine echte Leistungsverringerung dar, da prinzipiell der Strom nur gepulst wird, jedoch führt es in vielen Anwendungen zu den gleichen Ergebnissen. Diese Form der Leistungskontrolle wird benutzt, da dieses Signal von digitalen Bauteilen einfacher zu erzeugen ist, als eine tatsächliche Leistungsreduzierung.

Motor-Treiber Da der Arduino-Mikrocontroller nur eine maximale Ausgangsspannung von fünf Volt und einen maximalen Strom von 40 Milliampere pro Pin liefern kann und dies bei Weitem nicht ausreicht, um einen Motor zu bewegen, benötigt man einen Motor-Treiber. Dieses Bauteil dient als Interface zwischen Arduino und jeweils einem Schrittmotor. Abbildung 5A zeigt einen solchen Motor-Treiber.

Die Eingabe ist ein Signal des Arduinos, welches die oben beschriebene Form hat, jedoch sehr schwach bezüglich seiner Leistung ist. Der Motor-Treiber verstärkt nun dieses Signal um ein Vielfaches und gibt es an die Schrittmotoren weiter.

Um die Motoren ausreichend mit Strom zu versorgen, benötigen die Motor-Treiber eine zusätzliche Stromquelle von zwölf Volt. Aufgrund der hohen Leistungsaufnahme der Motoren und somit auch der Motor-Treiber entwickeln die Mikrocontroller der Treiber viel Hitze. Um diese Hitze abzuführen, sollten die Motor-Treiber mit Kühlkörpern und genügend Belüftung (z.B. durch einen PC-Lüfter) versehen werden.

Probleme mit Schrittmotoren Neben dem großen Vorteil der hohen Präzision haben Schrittmotoren allerdings auch einige Nachteile.

Der sogenannte Schrittverlust beschreibt das „Durchrutschen" des Rotors. Dies passiert, wenn der Motor nicht genügend Kraft erzeugen kann, um den Rotor zu bewegen oder zu halten und sich dieser durch äußere Krafteinwirkung unkontrolliert dreht. Problematisch ist dies vor allem, da die aktuelle Ausrichtung des Rotors nicht gemessen, sondern nur iterativ von der Startposition aus bestimmt wird. Somit ist nach einem Schrittverlust nicht mehr

garantiert, dass der Rotor sich in der gewünschten Position befindet und eine erneute Kalibrierung ist notwendig.

Problematisch ist außerdem, dass aufgrund der fehlenden Sensoren die Anfangsposition eines Schrittmotors nicht ohne zusätzliche Bauteile bestimmt werden kann. In der Regel wird dafür ein Motoranschlag verwendet, welcher jedoch getrennt verbaut, angeschlossen und ausgelesen werden muss.

Diese beiden Nachteile können von Servomotoren ausgeglichen werden, da sie über eine geschlossene Rückkopplungsschleife (engl. closed feedback loop) verfügen. Sie verfügen also über einen Lagesensor, über den immer die aktuelle Ausrichtung der Motorwelle ausgelesen werden kann. Servomotoren haben allerdings den Nachteil, dass sie aufgrund von Lastwechselreaktionen der verbauten Zahnräder keine so hohe Präzision wie Schrittmotoren aufweisen.

1.2.4 Leuchtdioden

Leuchtdioden (LEDs) sind heute aufgrund ihrer kleinen Bauweise und Energieeffizienz ein essenzieller Bestandteil in der Elektrotechnik. Im Folgenden werden die allgemeine Funktionsweise und die Besonderheiten der in dieser Arbeit verbauten LEDs beschrieben.

Funktionsweise Leuchtdioden bestehen neben der nötigen Verdrahtung und einem Glas- oder Plastikgehäuse aus einem Halbleiterkristall aus einer Galliumverbindung. Dieser Halbleiter besteht aus einer n- und einer p-Schicht. Wird eine Spannung angelegt, so wandern Elektronen von der n-Schicht über den p-n-Übergang zur p-Schicht und geben dabei Energie in Form von Photonen bzw. sichtbarem Licht ab. Die Farbe der LED wird dabei von den verwendeten Elementen (z.B. Aluminium oder Indium) in der Galliumverbindung oder einer zusätzlichen Lumineszenzschicht bestimmt.

WS2812B In dieser Arbeit wurden spezielle LEDs vom Typ WS2812B verbaut. Die Spezifikationen dieser LEDs wurden aus [WS2] entnommen. Diese LEDs sind im platzsparenden SMD-Formfaktor (egl. „Surface-mounted device") gebaut und auf leitende Kupferstreifen gelötet. Jedes Pixel besteht aus drei Leuchtdioden in den Farben rot, grün und blau und einem zusätzlichen Mikrochip, wie in Abbildung 4 zu sehen ist. Dieser Mikrochip verarbeitet ein Eingangssignal, steuert in Abhängigkeit dieses Signals die drei LEDs und gibt das Signal anschließend an das nächste Pixel weiter. Somit kann jede LED einzeln angesteuert werden, unabhängig von der Anzahl der miteinander verbundenen SMD-LEDs.

Jedes Pixel verarbeitet dabei insgesamt 24 Bits (acht Bit pro Farbe). Der Mikrochip nimmt dabei immer die ersten 24 Bits eines Bitstreams und verarbeitet diese. Dabei werden diese Bits aus dem Bitstream herausgeschnitten. Die restlichen Bits werden unverändert an das nächste Pixel weitergegeben.

Aus diesem Grund ist die korrekte Verkabelung der SMD-LEDs wichtig: Jedes Pixel hat einen Dateneingang und einen Datenausgang. Aufgrund dieser kaskadierenden Datenverarbeitung können beliebig viele WS2812B-LEDs miteinander verbunden und angesteuert werden.

Das Signal besteht aus drei Komponenten: einem 1-Bit-Signal, einem 0-Bit-Signal und einem Reset-Code (siehe Abbildung 4). Hat man beispielsweise zehn Pixel miteinander verbunden, so muss der benötigte Bitstream zur Steuerung der LEDs mindestens $24 \cdot 10 = 240$ Bit lang sein. Auf diesen Bitstream muss eine mindestens 50 Mikrosekunden lange Pause als Reset-Code folgen, bis das nächste Signal gesendet werden kann.

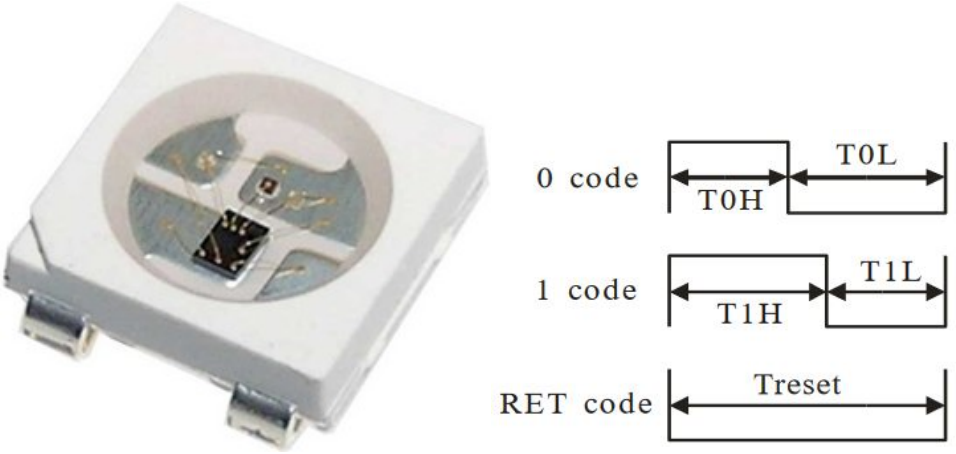


Abbildung 4: Links: Nahaufnahme einer WS2812B-LED im SMD-Formfaktor, rechts: Signal zur Steuerung des Mikrochips [WS2]

1.2.5 Arduino

In dieser Arbeit wurde ein Arduino-Uno-Mikrocontroller zum Ansteuern der elektronischen Bauteile verwendet. Dieser wurde gewählt, da er über genügend Output-Pins verfügt, leicht erweiterbar ist, einfach programmiert und angesteuert werden kann, und preisgünstig erhältlich ist. Außerdem ist die gesamte Arduino-Produktfamilie (Soft- und Hardware) vollständig Open-Source.

Aufbau Das Herzstück des Arduinos ist ein Atmel ATmega328P Mikrocontroller. Dieser ist auf eine Schaltplatine gelötet. Er kann über USB an einen PC angeschlossen werden und verfügt zusätzlich über einen separaten Stromanschluss. Weiterhin hat der Arduino fünf analoge Eingänge und 13 digitale Ports, von denen sechs ein PWM-Signal ausgeben können. In dieser Arbeit werden lediglich die digitalen Ports verwendet, da diese sowohl als Ein-, als auch als Ausgang genutzt werden können und für diese Anwendung vollkommen ausreichen. Abbildung 5C zeigt den Arduino ohne weitere

angeschlossene Bauteile.

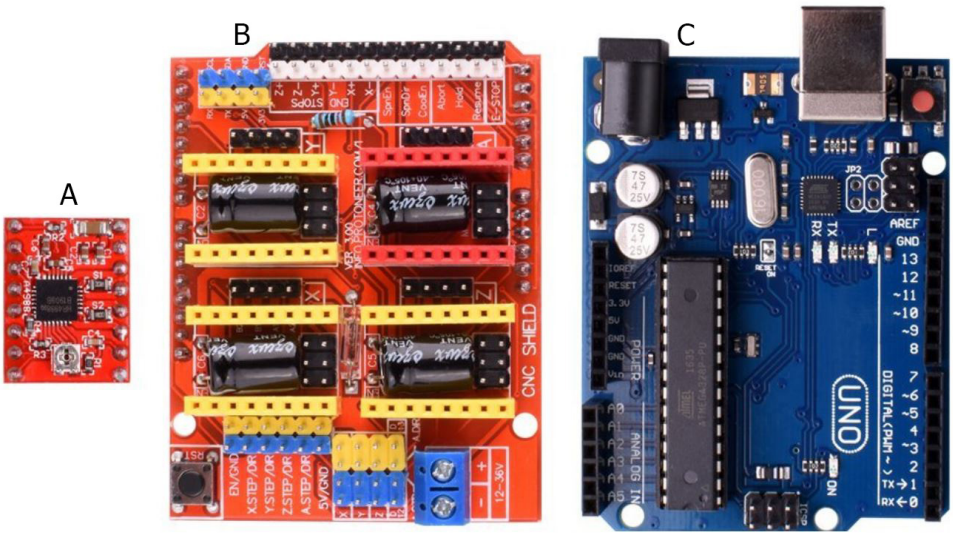


Abbildung 5: A: Motor-Treiber, B: Schrittmotor-Shield, C: Arduino Uno

Programmierung Programmiert wird der Arduino in einer C++-ähnlichen Programmiersprache, welche stetig als Open-Source-Projekt weiterentwickelt wird. Der Code wird dabei am PC geschrieben, kompiliert und anschließend über USB auf den Mikrocontroller des Arduinos geflasht. Da der Arduino nur über 32 Kilobyte Speicherplatz verfügt, muss beim Programmieren auf Speichereffizienz geachtet werden.

Prinzipiell werden über den Arduino lediglich die IO-Pins an- und ausgeschaltet oder ausgelesen. Dazu wird als erstes definiert, ob ein Pin als Ein- oder Ausgabe-Pin verwendet werden soll. Anschließend kann mit dem entsprechenden Befehl der Status des Pins auf An bzw. Aus gesetzt werden oder der aktuelle Status (also ob Strom anliegt) abgefragt werden. Die Code-Beispiele 1 und 2 zeigen dieses Vorgehen.

Code-Ausschnitt 1: Schreiben des Status von Pin 13

```
void setup() {
  pinMode(13, OUTPUT); // setzt Pin 13 als Output
  digitalWrite(13, HIGH); // schaltet Pin 13 an
}
```

Code-Ausschnitt 2: Lesen des Status von Pin 12

```
void setup() {
  pinMode(12, INPUT); // setzt Pin 12 als Input
  int val = digitalRead(12); // schreibt Pin 12 in val
}
```

Zur Laufzeit können Daten über eine serielle USB-Schnittstelle zwischen dem Arduino und dem PC ausgetauscht werden. Wichtig dabei ist, dass beide Geräte auf dieselbe Baud-Rate eingestellt sind (z.B. 115200). Sobald eine Verbindung hergestellt wurde, können Nachrichten (z.B. Textbefehle) gesendet und verarbeitet werden. Der Code-Ausschnitt 3 zeigt eine beispielhafte Textsendung über die serielle Schnittstelle des Arduinos.

Code-Ausschnitt 3: Verwendung der seriellen Schnittstelle

```
void setup() {
  Serial.begin(115200); // serielle Schnittstelle initialisieren
  String in = Serial.readString(); // eingehende Nachricht lesen
  Serial.println("Message received"); // Nachricht senden
}
```

Erweiterungen („Shields") Der Arduino-Mikrocontroller kann sehr einfach durch sogenannte „Shields" erweitert werden. Ein Shield ist prinzipiell eine weitere Platine im gleichen Formfaktor wie der Arduino und wird auf den Arduino aufgesteckt. Auf dieser Platine sind die Ein- und Ausgangspins des Arduinos je nach Anwendung passend mit weiteren Bauteilen verdrahtet.

In dieser Arbeit wurde ein Schrittmotor-Shield verwendet. Dieses wird einfach auf den Arduino gesteckt und bietet dann insgesamt vier Sockel, in welche die Motor-Treiber-Chips gesteckt werden können. Weiterhin befinden sich auf dem Shield mehrere Pins, die zum Anschließen der Schrittmotoren und anderen Bauteilen sowie zur externen Stromversorgung dienen. Auch die Auflösung des Microsteppings wird auf dieser Platine mittels Jumper eingestellt. Abbildung 5B zeigt das Shield ohne eingesetzte Motor-Treiber.

2 Aufbau des Gonioreflektometers

Beim Aufbau der Hardware für das Gonioreflektometer wurde vor allem darauf geachtet, dass es sich bei allen Bauteilen um handelsübliche Komponenten handelt, die beispielsweise im Baumarkt oder im Internet zu erschwinglichen Preisen erhältlich sind. Wurden keine passenden Teile für einen bestimmte Komponente gefunden, wurde versucht, das Ziel durch Kombination alternativer Bauteile zu erreichen (wie z.B. die Lichtkuppel).

Abbildung 6 zeigt den fertigen Aufbau des Gonioreflektometers. Im folgenden Abschnitt wird die Konstruktion der einzelnen Komponenten genauer beleuchtet.

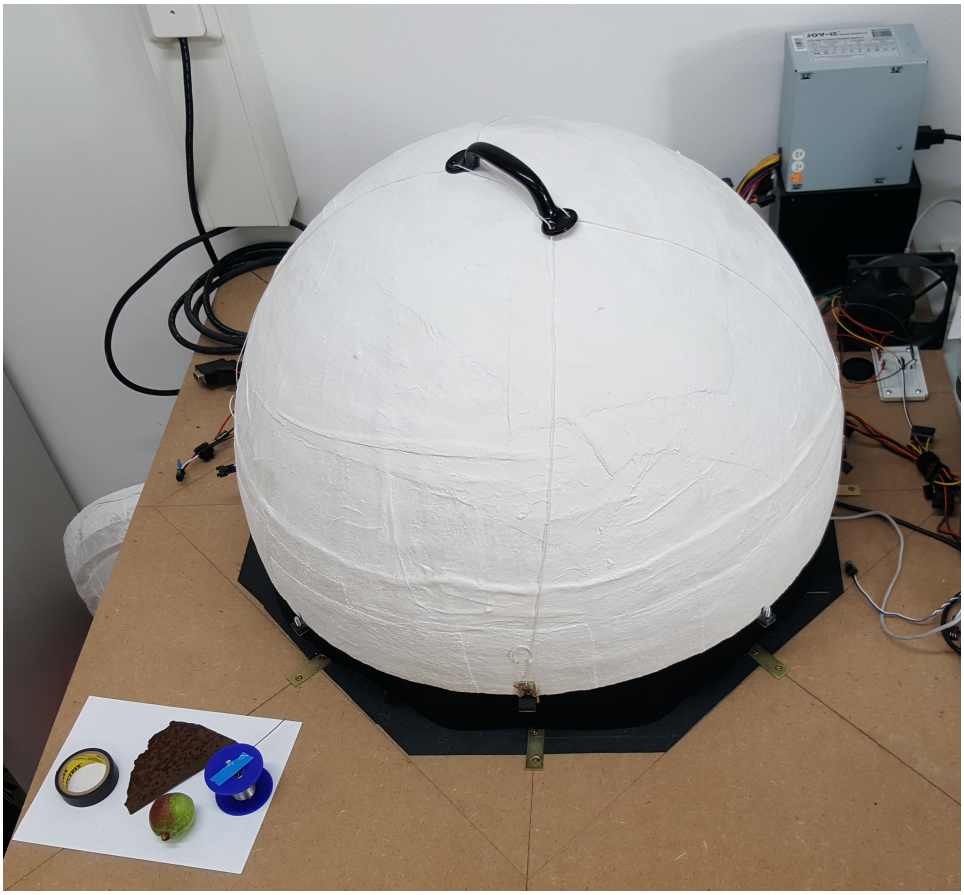


Abbildung 6: Finaler Aufbau des Gonioreflektometers von außen. Unten links sind einige 3D-Scan-Objekte und Materialproben zu sehen. Die gesamte Ansteuerungselektronik befindet sich im Bild oben rechts.

2.1 Konstruktion des Kameraarms

Für die Konstruktion des Kameraarms wurde aufgrund von Stabilitäts- und Gewichtsanforderungen hauptsächlich Aluminium verwendet. Die Lochplatte, auf welcher die Motoren angebracht wurden, besteht aus einer Zinklegierung. Für die Bewegung des Kameraarms wurden zwei bipolare Nema-17 Schrittmotoren mit jeweils 59Ncm Drehmoment verbaut. Die Drehscheibe wird vom einem ähnlichen aber kleineren Schrittmotor mit einem Drehmoment von 26Ncm bewegt. Um ein Überdrehen der Motoren zu verhindern wurden außerdem zwei Taster als Motoranschlag verbaut. Abbildung 7 zeigt Fotos von einigen Schritten des Aufbaus.

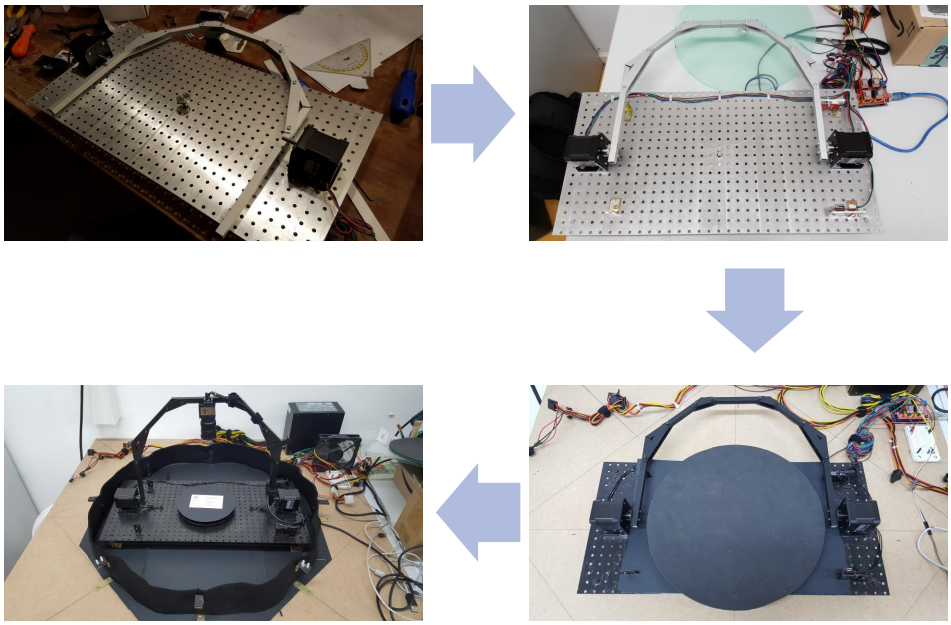


Abbildung 7: Aufbau des Kameraarms

Im ersten Arbeitsschritt wurde der U-förmige Arm konstruiert. Dazu wurde ein Vierkant-Aluminium-Rohr exakt ausgemessen. Anschließend wurden die vier Knickstellen markiert und eingeschnitten, um das Rohr jeweils um 45 Grad zu biegen. Da die Knickstellen nicht die nötigen Stabilität bieten, um eine Kamera damit zu heben, wurden die entsprechenden Stellen durch jeweils zwei Querstreben (ebenfalls aus Aluminium) verstärkt.

Im zweiten Schritt wurden die Motoren an die Trägerplatte montiert. Für bessere Stabilität und aufgrund der besseren räumlichen Aufteilung wurde der Drehscheibenmotor unter der Platte befestigt. Die beiden Armmotoren wurden mit passenden L-Winkeln oberhalb der Metallplatte festgeschraubt.

Anschließend wurde der Arm mittels zwei Flanschkupplungen an den Motoren befestigt. An den Drehscheibenmotor wurde ebenfalls eine Flanschkupplung montiert, auf welcher später die Drehscheibe geklebt wird. In diesem Schritt wurden auch die Motoranschläge auf einer passenden Höhe montiert.

Da sich herausstellte, dass die Motoren nicht genügend Drehmoment erzeugen können, um den Arm mitsamt Kamera zu heben, wurden zwischen den Armenden und der Metallplatte mehrere Federn gespannt, um die Motoren beim Heben zu unterstützen. Dies war sogar so wirkungsvoll, dass die Leistung der Motoren reduziert werden konnte.

Nachdem alle bisher verbauten Teile an den Arduino-Mikrocontroller angeschlossen und die Kabel an der Metallplatte fixiert wurden, wurden alle

Bauteile mit Polyacryllack matt-schwarz gestrichen, um später Lichtreflexionen zu vermeiden. Zusätzlich wurde die Metallplatte und der Mikrocontroller auf einer mitteldichte Holzfaserplatte (MDF) montiert um eine versehentliche Verschiebung oder Beschädigung der Komponenten zu vermeiden.

Im letzten Schritt wurde die Kamera an den Arm montiert und das Kamerakabel mit Kabelbindern am Kameraarm entlang befestigt. Weiterhin wurde die Drehscheibe auf dem dafür vorgesehenen Motor montiert. Außerdem wurde aus Lichtschutzgründen die Fläche unterhalb der Metallplatte mit schwarzer Pappe ausgelegt und ein schwarzer Filzrand um die Konstruktion gespannt.

2.2 Konstruktion der Lichtkuppel

Der Grundaufbau der Lichtkuppel folgte zwei wichtigen Kriterien: Der Durchmesser sollte 60 bis 70 Zentimeter betragen und die Innenseite der Konstruktion sollte eine möglichst runde Form aufweisen. Leider stellte es sich als schwierig bzw. sehr kostspielig heraus, eine Kuppel mit denn genannten Kriterien in fertiger Form zu erwerben. Die meisten runden Kuppeln (z.B. aus Styropor oder Kunststoff) werden nur bis zu einem Durchmesser von 50 Zentimetern zu erschwinglichen Preisen angeboten, größere Ausführungen sind nur in Form von teuren Maßanfertigungen erhältlich.

Aus diesem Grund musste die Lichtkuppel aus anderen handelsüblichen Materialien von Hand gebaut werden. Nach langem Überlegen fiel die Entscheidung letztendlich auf eine etwas unübliche Bauweise: Eine Papierlampen-Gymnastikball-Gips-Konstruktion, wie in Abbildung 8 gezeigt.

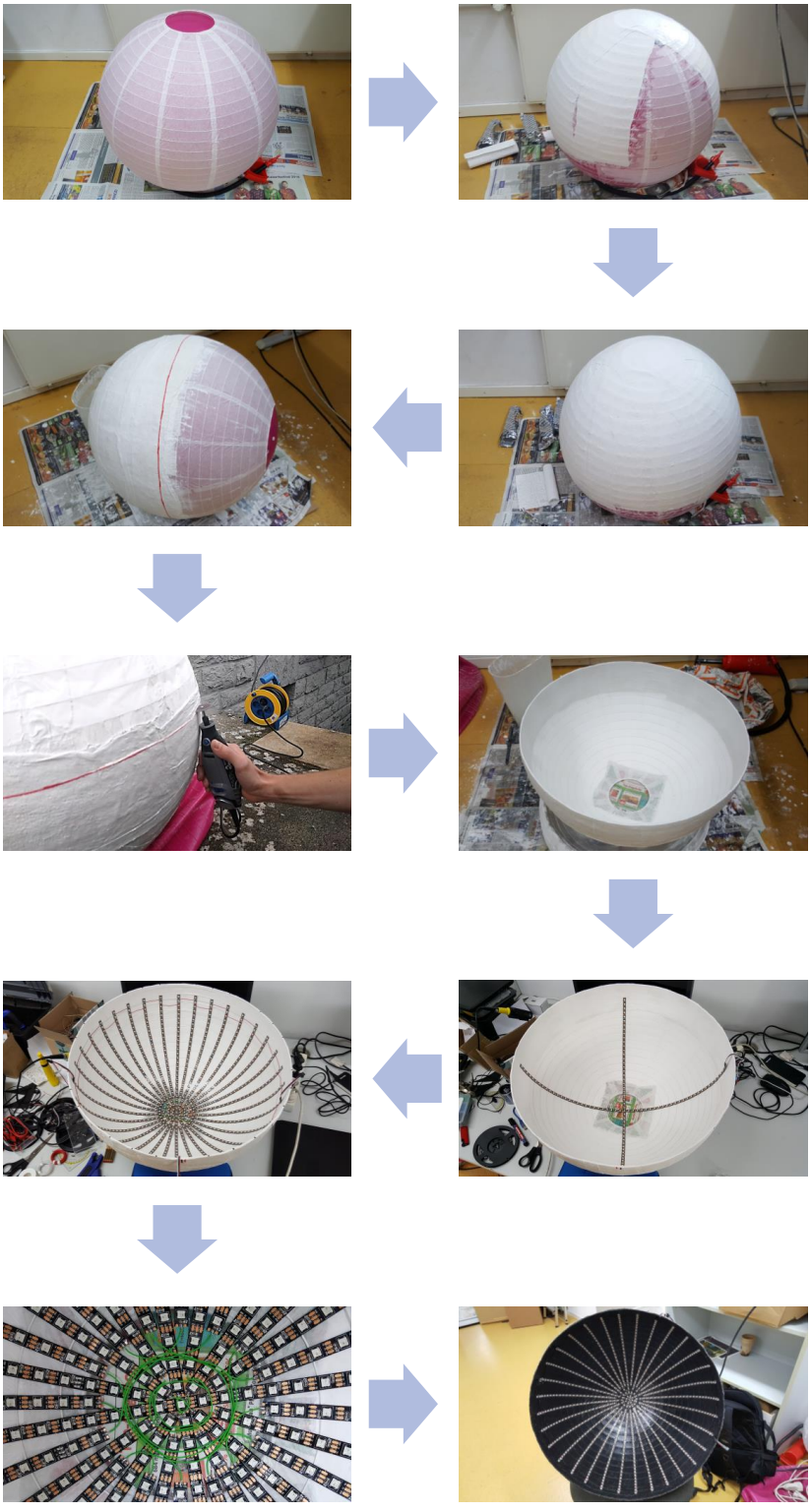


Abbildung 8: Aufbau der Lichtkuppel

Zuerst wurde ein Reispapierlampenschirm mit ca. 60cm Durchmesser besorgt. Anschließend wurde in diesem kugelförmigen Lampenschirm ein Gymnastikball mit einem Durchmesser von ebenfalls ca. 60cm aufgepumpt. Der Gymnastikball dient dazu, die Form der Reispapierleuchte zu stabilisieren, da das Material der Leuchte trotz der verstärkenden Metalldrähte nicht sehr formbeständig ist. Durch den Druck des Balls von innen gegen das Papier wurde dieses gestrafft und in eine gleichmäßige, runde Form gebracht, was für den späteren Anwendungszweck der Lichtkuppel entscheidend ist.

Im nächsten Schritt wurde eine Hälfte der Kugel mit handelsüblichen Gipsbinden eingekleidet. Um eine gute Stabilität zu erreichen, wurden zwei Lagen Gips auf der Reispapierlampe angebracht. Die Gipsbinden wurden hierfür etwa zehn bis 20 Sekunden in Wasser gehalten, dann auf die Kugel gelegt und abschließend mit den Händen glattgestrichen.

Nach dem Trocknen des Gipses wurde die Schnittkante in der Mitte der Kugel sorgfältig ausgemessen und markiert. Außerdem wurde der Gymnastikball durch die untere Öffnung der Reispapierlampe entfernt, indem die Luft abgelassen wurde. Nun konnte die Kugel mit einem Dremel entlang der Markierung aufgeschnitten werden. Um die Kante zu stabilisieren, wurden zwei zusätzliche Schichten Gips um diese herum angebracht.

Nach dem Trocknen und dem Reinigen der Innenseite konnten die schwarzen LED-Streifen nach und nach eingeklebt und mit Kupferdrähten verbunden werden. Insgesamt wurden 809 einzeln ansteuerbare RGB-LEDs vom Typ WS2812B in longitudinal äquidistanten Abständen verbaut. So hat die Lichtkuppel eine Auflösung von 32 Schritten in der Länge (horizontal) und 28 Schritten in der Breite (vertikal).

Um Artefakte aufgrund von Reflexionen zu vermeiden, wurden die Zwischenräume zwischen den einzelnen Streifen mit matt-schwarzem Polyacryllack bestrichen. Im Nachhinein wäre es vermutlich einfacher und zeitsparender gewesen, die Innenseite der Kuppel vor dem Einkleben der LED-Streifen schwarz anzumalen. Dann hätte nicht so präzise und sorgfältig gearbeitet werden müssen, um die LEDs nicht zu beschädigen. Das Endresultat ist jedoch trotzdem sehr zufriedenstellend.

Damit die Lichtkuppel sicher über dem Kameraarm fixiert werden konnte, wurden acht Füße aus L-Winkeln gebaut, welche die Kuppel jeweils mittels zwei langer Schrauben in Position halten (siehe Abbildung 9). Außerdem wurde ein Griff an der Oberseite der Kuppel angebracht, damit diese einfacher angehoben werden kann.

Abbildung 9: Stabilisierungsfuß für die Lichtkuppel (links) und Haltegriff zum Heben der Kuppel (rechts)

2.3 Anschließen der elektronischen Bauteile

Für die Stromzufuhr der Motoren und Lichtquellen wurde ein handelsübliches PC-Netzteil verwendet. Dies hat neben der guten und preisgünstigen Verfügbarkeit den Vorteil, eine hohe Leistungsaufnahme durch die elektronischen Bauteile bedienen zu können. Außerdem hat es sowohl einen 12 Volt Ausgang (für die Motoren), als auch einen 5 Volt Ausgang (für die LEDs). Abbildung 10 zeigt eine Nahaufnahme der Verdrahtung der einzelnen Bauteile.

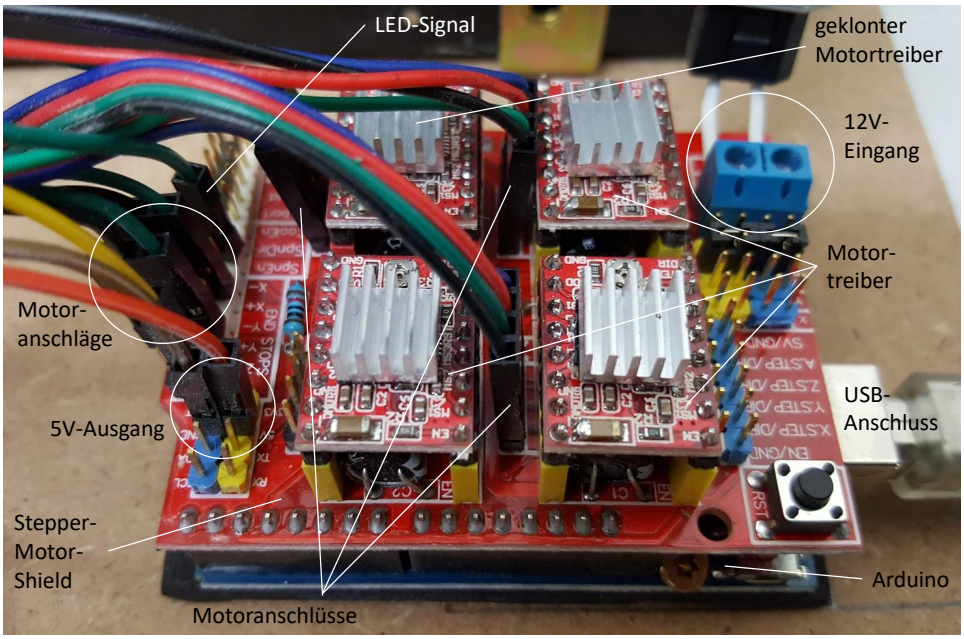


Abbildung 10: Verdrahtung des Arduino-Mikrocontrollers und des aufgesetzten Stepper-Motor-Shields

2.3.1 Schrittmotoren

Zum Ansteuern der Schrittmotoren wurde ein Stepper-Motor-Shield auf den Arduino-Mikrocontroller gesetzt und an 12 Volt angeschlossen, da Schrittmotoren ein besonderes Signal zum Bewegen benötigen (s. Grundlagen 1.2.3). Dieses kann drei separate Schrittmotoren und einen geklonten (also mit gleichem Signal) ansteuern. Pro Motor wird zur Erzeugung des Motor-Signals ein A4988-Schrittmotortreiber verwendet, welcher das Steuersignal des Arduinos in ein von den Schrittmotoren verwendbares Signal umwandelt.

Besonders der geklonte Motortreiber ist für diese Anwendung sehr praktisch, da sich die beiden Motoren zum Bewegen des Arms genau entgegengesetzt bewegen müssen. Dieser Treiber wird also so eingestellt, dass er das gleiche Signal wie Treiber Nummer drei (einer der Armmotoren) ausgibt. Der zugehörige Motor wird dann einfach „verkehrt-herum" angeschlossen und bewegt sich folglich genau entgegengesetzt.

Um eine feinere Schrittunterteilung zu ermöglichen, wurde das Microstepping (s. Grundlagen 1.2.3) der Motortreiber auf den höchsten Wert (1/16 Schritte) eingestellt, wodurch eine Auflösung von 3200 Schritten pro Umdrehung erreicht wird. Außerdem wurde die Leistung der Motoren so eingestellt, dass Sie den Kameraarm problemlos heben können. Da die Motortreiber dabei

sehr warm werden, wurden kleine Kühlkörper aufgeklebt und ein PC-Lüfter so angebracht, dass kühle Luft darüber strömt, um die Hitze abzuführen.

Des weiteren wurden die Taster für die Motoranschläge an die dafür vorgesehenen Signal-Pins des Arduinos und die Stromversorgung angeschlossen. Zusätzlich wurde ein Taster zum Zurücksetzen der Motoren und Lichtquellen eingebaut. Abbildung 11 zeigt die Schaltskizze für diesen Taster.

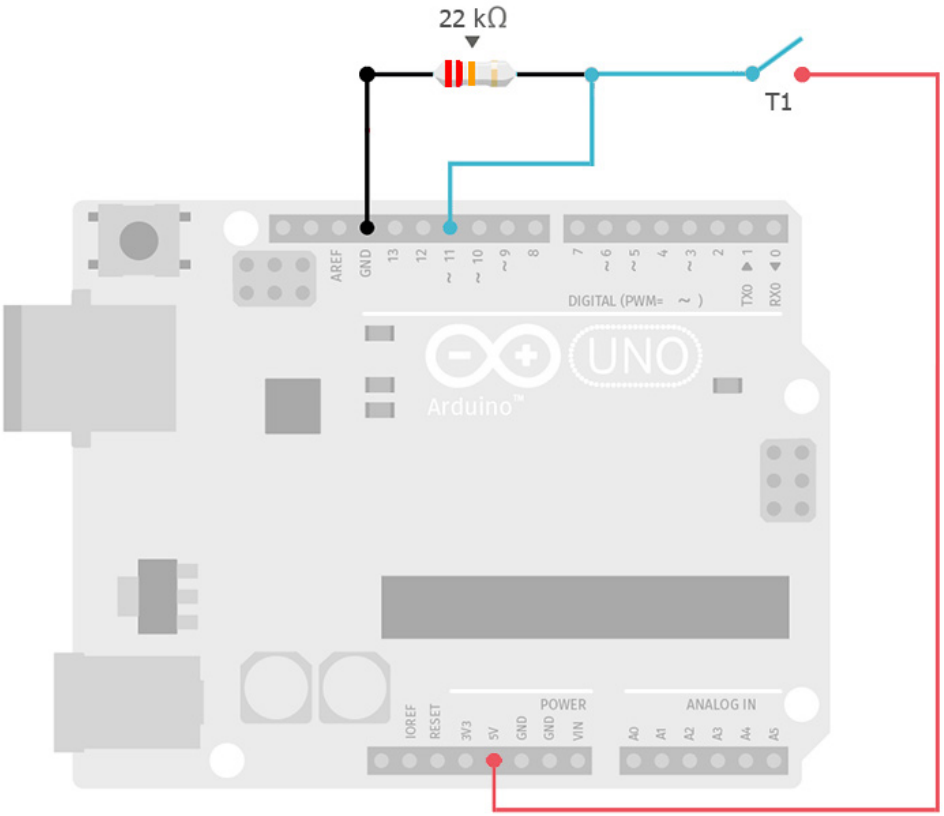


Abbildung 11: Schaltskizze für den Reset-Taster (T1)

2.3.2 LED-Streifen

Die LED-Streifen benötigen zur Stromversorgung 5 Volt. Da alle Streifen dieselbe Spannung benötigen, werden alle Streifen parallel geschaltet, was bedeutet, dass alle Minuspole miteinander verbunden werden und ebenso alle Pluspole. Um eine Beschädigung der LEDs durch Spannungsspitzen zu verhindern, werden zwischen Plus- und Minuspol drei Kondensatoren mit einer Kapazität vom jeweils 1000uF gelötet, was eine Gesamtkapazität von 3000uF ergibt.

Beim Verbinden der Signal-Pins der LEDs ist darauf zu achten, dass jede LED und somit auch jeder LED-Streifen das Signal in einer gewissen

Richtung verarbeitet. Jede LED hat also einen Din- und einen Dout-Pin. Die LEDs müssen immer so verbunden werden, dass auf einen Dout-Pin ein Din-Pin folgt. An der ersten LED wird der Din-Pin über einen 470 Ohm Widerstand mit einem Out-Pin des Arduinos verbunden. Abbildung 12 zeigt die Schaltskizze für die Ansteuerung der Lichtquellen.

Damit die Lichtkuppel leichter angeschlossen werden kann, wurden Steckverbinder eingelötet, die einfach mit dem Stromkabel bzw. dem Signalkabel verbunden werden können.

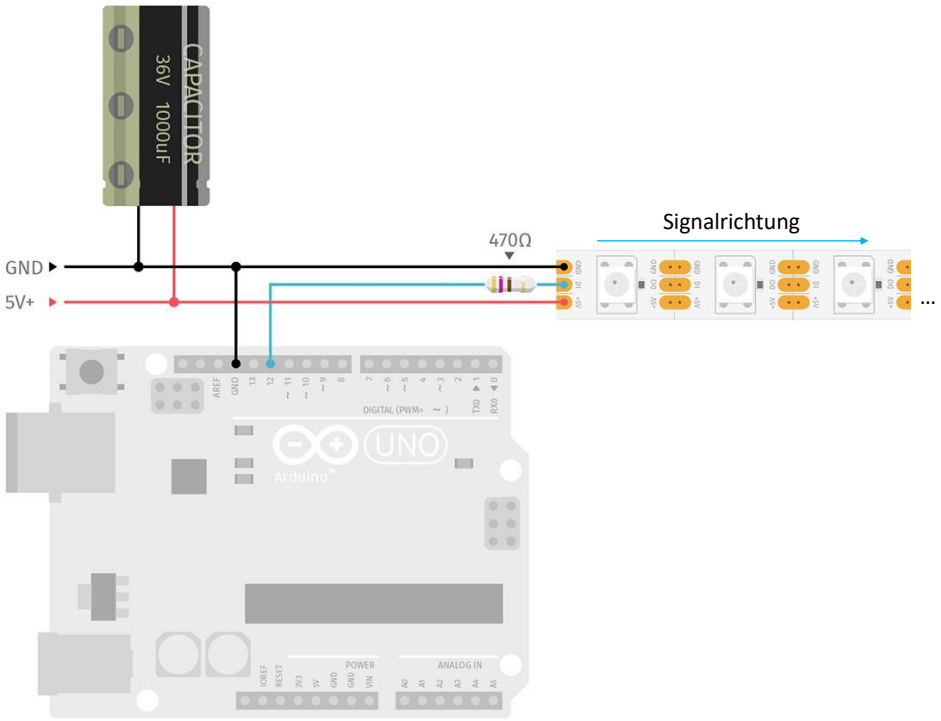


Abbildung 12: Verdrahtung der LED-Streifen

2.3.3 Kamera

Für die Aufnahme der Bilddaten wurde eine UI-1240SE-C-HQ Kamera von IDS Imaging verwendet. Diese Kamera ist speziell für industrielle Bildverarbeitungsanwendungen gedacht und kann Farbbilder mit einer Auflösung von 1280×1024 in 25,8 Bildern pro Sekunde aufnehmen. Aufgrund der geringen Helligkeit innerhalb der Lichtkuppel wurde die Belichtungszeit mit 200-300 Millisekunden auf einen recht hohen Wert eingestellt, wodurch jedoch auch die Bildrate abnimmt. Für dunkle Materialien muss die Belichtungszeit ggf. angepasst werden. Außerdem wurden helligkeits- und farbverändernde Funktionen wie Weißabgleich oder Auto-Exposure ausgeschaltet, um die Ergebnisse so wenig wie möglich zu verfälschen. Die Kamera wird über eine USB-2.0-Schnittstelle mit dem PC verbunden und die Bilddaten können dann

mit OpenCV ausgelesen werden.

3 Aufnahme von BRDFs

Die Aufnahme von BRDFs basiert im Wesentlichen aus dem iterativen Ansteuern der Lichtquellen und der Motoren, und einer anschließenden Bildaufnahme. Im Folgenden wird die Realisierung der Steuerungssoftware, sowie der nötigen Aufnahmeprogramme genauer beleuchtet.

3.1 Programmierung des Arduino-Controllers

Der Code für den Arduino wurde in der Arduino-eigenen Programmiersprache geschrieben. Da der Arduino nur über begrenzte Speicher- und Rechenleistung verfügt, dient der Mikrocontroller lediglich als „Übersetzer" zwischen PC und elektronischen Aktoren (Motoren und LEDs). Der Arduino erhält einen Befehl zum Bewegen eines Motors oder zum Schalten von LEDs über die serielle Schnittstelle. Dieser wird verarbeitet, um den Befehl in seinen Typ und seine Parameter zu zerlegen. Anschließend sendet der Arduino das entsprechende Signal an die Motoren bzw. die LEDs. In Code-Ausschnitt 4 ist dieser Algorithmus veranschaulicht.

Code-Ausschnitt 4: Zerlegung und Verarbeitung eines Steuerungsbefehls

```
void processCommands(String allCommands)
{
  // iteriere solange es neue Befehle gibt
  while(allCommands.length() > 0)
  {
    // bestimme Art und Parameter des aktuellen Befehls
    int cmdLength = getCommandLength(allCommands);
    String cmd = allCommands.substring(0,2);
    String cmdData = allCommands.substring(2, cmdLength);

    // Beispiel: Verwerten des Arm-Bewegen-Befehls
    if(cmd.equalsIgnoreCase("am"))
    {
      float angle = cmdData.toFloat();
      stepArm(angle);
    }

    // [...] hier folgen alle anderen moeglichen Befehle

    // entferne abgearbeiteten Befehl aus Befehlsliste
    allCommands = allCommands.substring(cmdLength + 1);
  }
}
```

3.1.1 Programmierung der Motoransteuerung

Für die Ansteuerung der Motoren wird zwischen dem Bewegen des Kameraarms und der Drehscheibe unterschieden. Das Signal wird in beiden Fällen prinzipiell nach dem gleichen Muster erzeugt, weshalb der Fokus im Folgenden auf den Kameraarm gelegt wird.

Die Ansteuerung lässt sich in zwei Teile aufspalten: Die Bestimmung der Drehrichtung und das Drehen um einen gewissen Winkel.

Drehrichtung Die Drehrichtung der Motoren hängt vom Status des Dir-Pins des Motor-Treibers ab. Dieser Pin wird also entweder ein oder aus geschaltet, um die Drehrichtung zu ändern. Der Code-Ausschnitt 5 zeigt die dafür nötige Programmierung.

Code-Ausschnitt 5: Steuerung der Motordrehrichtung

```
void setArmDirection(bool forward)
{
  // schalte den Pin Z_DIR an bzw. aus
  digitalWrite(Z_DIR, forward);
  delay(10);
}
```

Bewegung Um den Arm um einen gewissen Winkel zu drehen, muss zuerst die dafür benötigte Anzahl an Schritten errechnet werden. Dafür wird der valide Winkelbereich ([0, 180]) auf einen validen Schrittbereich (z.B. [0, 1600]) projiziert. Anschließend oszilliert man den Bewegungs-Pin des entsprechenden Motor-Treibers diese Anzahl an Schritten zwischen an und aus. Die Frequenz dieser Oszillierung bestimmt dabei die Geschwindigkeit, mit der sich der Motor bewegt. Um ein Überdrehen der Motoren zu vermeiden, wird in jedem Oszillationsdurchlauf geprüft, ob ein Motoranschlag ausgelöst wurde. Da es dabei passieren kann, dass nicht die gewollte Anzahl an Schritten ausgeführt worden ist, wird die tatsächlich ausgeführte Schrittanzahl zurückgegeben. In Code-Ausschnitt 6 ist dieses Vorgehen zu sehen.

Code-Ausschnitt 6: Steuerung der Motorbewegung

```
int stepArm(float angle)
{
  // berechne Schrittanzahl aus Drehwinkel
  int steps = int(abs(mapf(angle, 0.0f, 180.0f, 0, 1600)));

  int stepCount = 0;

  // oszilliere steps-mal
  for(int i = 0; i < steps; i++)
  {
    // pruefe, ob ein Motoranschlag ausgeloest wurde
    if(digitalRead(Y_END) == LOW || digitalRead(X_END) == LOW)
    {
      Serial.println("Endstop reached");
      break;
    }

    // oszilliere einen Zyklus
    digitalWrite(Z_STP, HIGH);
    delay(_armSpeed);
    digitalWrite(Z_STP, LOW);
    delay(_armSpeed);

    // zaehle tatsaechlich ausgefuehrte Schritte
    stepCount++;
  }
  return stepCount;
}
```

3.1.2 Programmierung der LED-Ansteuerung

Die Programmierung der LED-Ansteuerung gestaltet sich aufgrund der großen Anzahl an verbauten LEDs (809) als schwierig. Der Arduino-Mikrocontroller verfügt nur über 2KB an internem Flash-Speicher, der für Variablen genutzt werden kann. Vorhandene Bibliotheken zur Ansteuerung von WS2812B-LEDs allokieren jedoch immer ein Array mit 24 Bit pro LED, was einen Speicherbedarf von $24Bit \cdot 809 = 19416Bit = 2427Byte \approx 2,4KB$ ergibt. Da dieser Speicherbedarf zu groß ist und daher nicht allokiert werden kann, ist es nicht möglich, den Status der LEDs zu speichern und das Signal muss zur Laufzeit erzeugt werden. Da es außerdem keine Bibliothek gibt, welche dieses Verfahren unterstützt, wurde die Signalerzeugung selbst implementiert.

LED-Signal Um ein Signal wie in Abbildung 4 zu erzeugen, müssen sehr präzise Zeitintervalle eingehalten werden. Damit diese erreicht werden, muss eine Interrupt Service Routine (ISR) benutzt werden, welche durch die Befehle *cli();* und *sei();* gesteuert wird (siehe Code-Ausschnitt 8). Anschließend muss der Pin, an welchen die LEDs angeschlossen sind, im richtigen Zeitintervall

an- und ausgeschaltet werden. Da es sein kann, dass der Arduino-Compiler den Quellcode zur Optimierung etwas umordnet, dies aber fatal für das erzeugte Signal wäre, wird dieser Teil des Programms in Assembler-Sprache geschrieben. Der Code-Ausschnitt 7 zeigt dies beispielhaft für das Senden eines 1-Bits.

Die benötigten Zeitintervalle wurden dabei wie folgt aus der WS2812B-Spezifikation entnommen (vgl. Abb. 4 rechts):

- 1-Bit: $0.8\mu s$ HIGH $\rightarrow$ $0.45\mu s$ LOW
- 0-Bit: $0.4\mu s$ HIGH $\rightarrow$ $0.85\mu s$ LOW
- Reset: $> 50\mu s$ LOW
- Toleranz: $\pm 150ns$

Code-Ausschnitt 7: Arduino-Assembler-Coder zum Senden eines 1-Bits

```
asm volatile (
    // setze output bit auf 1
    "sbi %[port], %[bit] \n\t"
    // fuehre NOPs aus um Zeitintervall zu erzeugen
    ".rept %[onCycles] \n\t"
    "nop \n\t"
    ".endr \n\t"
    // setze output bit auf 0
    "cbi %[port], %[bit] \n\t"
    //fuehre NOPs aus um Zeitintervall zu erzeugen
    ".rept %[offCycles] \n\t"
    "nop \n\t"
    ".endr \n\t"
    ::
    // Variablendefinitionen
    [port]     "I" (_SFR_IO_ADDR(PIXEL_PORT)),
    [bit]    "I" (PIXEL_BIT),
    [onCycles]   "I" (NS_TO_CYCLES(T1H) - 2),
    [offCycles]    "I" (NS_TO_CYCLES(T1L) - 2)
);
```

Da über diesen Assembler-Code nur einzelne Bits gesendet werden, wurden zusätzliche Hilfsfunktionen hinzugefügt, um das Signal einfacher zusammenbauen zu können. Dies beinhaltet zum einen Low-Level-Funktionen zum Senden von einzelnen Bytes sowie zum Erzeugen eines 24-Bit-Pakets pro LED. Zum anderen wurden High-Level-Funktionen zum Steuern einzelner, mehrerer oder aller LEDs auf einmal implementiert. Code-Ausschnitt 8 zeigt die Hilfsfunktion, um die Farbe einer einzelnen LED zu setzen. Die *show()*-Funktion sendet abschließend den in Abschnitt 1.2.4 beschriebenen Reset-Code.

Code-Ausschnitt 8: Hilfsfunktion zum Setzen der Farbe (r, g, b) einer bestimmten LED (id)

```
void showColorLED(uchar r, uchar g, uchar b, int id)
{
  cli();
  for(int p = 0; p < PIXELS; ++p)
  {
    if(p == id)
      sendPixel( r , g , b );
    else
      sendPixel( 0 , 0 , 0 );
  }
  sei();
  show();
}
```

3.2 Befehlssatz zur Ansteuerung über USB

Zur Kommunikation zwischen PC und Arduino wurde ein Befehlssatz entwickelt, mit dem alle Funktionen des Arduinos ausgeführt werden können. Die Befehle folgen dabei einem bestimmten Schema:

- Jeder Befehl ist zwei Zeichen (plus Parameter) lang
- Befehle, die dem Arm bewegen, beginnen mit „a”
- Befehle, die die Scheibe bewegen, beginnen mit „d”
- Befehle, die LEDs steuern, beginnen mit „l”
- Die Befehle „cl” und „sc” sind Sonderfälle
- Jeder Befehl endet mit einem Semikolon (;)
- Mehrere Befehle können verkettet werden

In der folgenden Tabelle 1 sind alle verfügbaren Befehle aufgeführt:

Befehl	Effekt
cl	Kalibrierung des Kameraarms durchführen
ar	Kameraarm auf Ausgangsposition (-90°) zurücksetzen
dr	Drehscheibe auf Ausgangsposition (0°) zurücksetzen
asX	Geschwindigkeit des Kameraarms auf X setzen (kleinere Werte bedeuten höhere Geschwindigkeit, Standardwert: 3)
dsX	Geschwindigkeit der Drehscheibe auf X setzen (Standardwert: 2)
amX	Kameraarm um X Grad bewegen (relativ)
dmX	Drehscheibe um X Grad bewegen (relativ)
aaX	Kameraarm auf Position X (-90° bis 90°) bewegen (absolut)
daX	Drehscheibe auf Position X (0° bis 360°) bewegen (absolut)
ap	Aktuelle Position des Kameraarms (in Grad) über USB senden
dp	Aktuelle Position der Drehscheibe (in Grad) über USB senden
sc	Verfügbare Anzahl an Schritten der Kameraarmmotoren über USB senden
lo	Alle Lichtquellen ausschalten
liXXXRRRGGGBBB	Licht XXX auf Farbe $[RRR, GGG, BBB]$ setzen ($XXX \in [0..808]$; $RRR, GGG, BBB \in [0..255]$)
laRRRGGGBBB	Alle Lichter auf die Farbe $[RRR, GGG, BBB]$ setzen
lmXXXRRRGGGBBB[...]	Farbe für mehrere Lichter setzen (*li*-Befehl für mehrere Lichter, Lichtnummern müssen aufsteigend sortiert sein)
lsXXXRGB[...]	Farbe für mehrere Lichter setzen (*lm*-Befehl mit geringerer Auflösung [0..9], um mehr Lichter ansprechen zu können. Wichtig für Light-Staging)
lrMMMNNN	Alle Lichter im Intervall MMM bis NNN einschalten

Tabelle 1: Befehle zur Steuerung des Gonioreflektometers über USB

3.3 Konzeption der Ansteuerungs-Software

Da es nicht praktikabel wäre, alle Befehle über eine Konsole an den Arduino zu senden, wurde eine Ansteuerungs-Software implementiert, die Teile der seriellen Kommunikation automatisiert. Ziel dabei war es, genügend Funktionalität zu bieten, um Kalibrierungen und Aufnahmen größtenteils

automatisch ausführen zu lassen, aber trotzdem direkte Kontrolle über die Motoren und Lichter zu haben. Das alles sollte außerdem in einer gut bedienbaren Oberfläche zu steuern sein, sodass der Benutzer nicht alle Befehle auswendig kennen muss, um das Programm bzw. das Gonioreflektometer zu bedienen. Diese Anwendung zeigt außerdem den aktuellen Status der Motoren und ein Live-Kamerabild aus dem Inneren des Gonioreflektometers. Abbildung 13 zeigt einen Screenshot der Benutzeroberfläche, welcher die einzelnen Komponenten veranschaulicht.

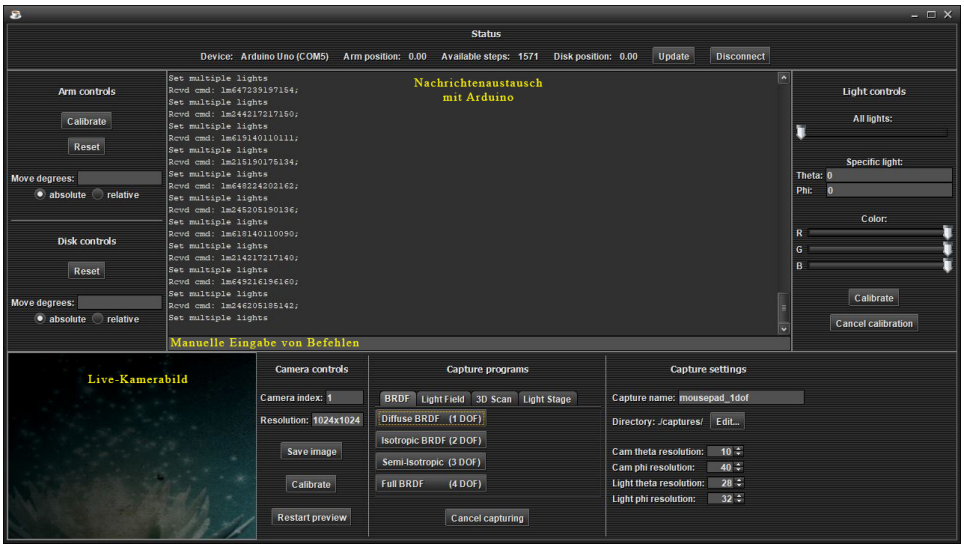


Abbildung 13: Benutzeroberfläche der Ansteuerungs-Software. Die Funktionen sind größtenteils selbsterklärend, der Rest wird durch den gelben Text beschrieben.

Als Programmiersprache für die Ansteuerungs-Software wurde Java gewählt, da es hierfür viele Bibliotheken gibt, welche die Handhabung mit einer grafischen Benutzeroberfläche, sowie der seriellen Schnittstelle erleichtern. Des Weiteren gibt es eine offiziell unterstützte Java-Version von OpenCV, was zur Bildmanipulation und -kalibrierung verwendet wird. Der größte Nachteil von Java - die etwas schlechtere Performance als z.B. C++ - fällt hier nicht ins Gewicht, da die Aufnahme von BRDFs ohnehin ein langwieriger Prozess ist und daher die Laufzeit nicht von großer Bedeutung ist.

Die Komponenten der Benutzeroberfläche lassen sich wie folgt einteilen:

- **Linkes Panel:** Motorsteuerung. Hier kann der Kameraarm und die Drehscheibe bewegt werden. Außerdem wird hier die Kalibrierung des Kameraarms gestartet.
- **Oberes Panel:** Statusanzeige. Hier wird der aktuelle Status der Motoren und der Verbindung zum Arduino angezeigt.

- **Rechtes Panel:** LED-Steuerung. Hier können entweder alle oder einzelne Lichtquellen angesteuert werden. Es kann eine Helligkeit oder eine Farbe eingestellt werden und die Kalibrierung der LEDs gestartet bzw. gestoppt werden.
- **Mittleres Panel:** Nachrichtenaustausch mit Arduino. Hier werden alle Nachrichten aufgelistet, die der Arduino an die Steuerungssoftware schickt. Über das untere Textfeld können Befehle auch direkt an den Arduino gesendet werden.
- **Unteres Panel:**
  - **Links:** Live-Kamerabild und Kamerasteuerung. Hier kann die zu verwendende Kamera ausgewählt, die Bildauflösung eingestellt, sowie die Kamerakalibrierung gestartet werden.
  - **Mitte:** Aufnahmeprogramme. Hier können verschiedene Aufnahmeverfahren gewählt und gestartet bzw. gestoppt werden.
  - **Rechts:** Aufnahmeeinstellungen. In diesem Panel wird die Schrittauflösung der Kamera und der Lichtquellen eingestellt. Außerdem kann ein Zielordner und ein Name für die Aufnahme gesetzt werden.

3.4 Kalibrierung

Eine essenzielle Voraussetzung für die spätere Aufnahme von BRDFs ist die korrekte Kalibrierung aller Komponenten, die dabei verwendet werden. Dabei muss nicht nur die Kamera, sondern auch die Motoren und die Lichtquellen kalibriert werden. Im Folgenden werden die einzelnen Kalibrierungsschritte genauer erklärt.

3.4.1 Kameraarm

Das am einfachsten zu kalibrierende Bauteil ist der Kameraarm. Hier muss lediglich sichergestellt werden, dass der Arm in einem Winkel von 90° zur Drehscheibe steht, wenn eine absolute Armposition von 0° an das Gonioreflektometer gesendet wird. Dafür fährt der Arm einmal von einem Motoranschlag zum Anderen und zählt die dabei ausgeführten Schritte. Anschließend wird diese Schrittanzahl auf den Winkelbereich von -90 bis 90 Grad projiziert.

3.4.2 Kamera

Die Kamerakalibrierung besteht im Allgemeinen aus der extrinsischen und der intrinsischen Kalibrierung. Die extrinsische Kalibrierung beinhaltet die Position und Rotation der Kamera und ist für diese Anwendung nicht relevant, da diese Parameter eindeutig aus der Position der Motoren bestimmt werden

können. Die intrinsische Kalibrierung beinhaltet die Kamera-Matrix und die Verzerrungskoeffizienten und ist in diesem Fall wichtig für das korrekte Berechnen des Bildes.

Als Eingabe für die Kalibrierung dienen mehrere Fotos eines bekannten Musters (hier ein Schachbrettmuster) aus verschiedenen Blickwinkeln. Für die Aufnahme dieser Fotos werden alle Lichtquellen eingeschaltet und der Kameraarm über eine auf der Drehscheibe liegende Kalibriervorlage bewegt.

Die dabei aufgenommenen Bilder werden anschließend an OpenCV übergeben und mittels der Funktion *Calib3d.calibrateCamera(...)* verarbeitet. Die Funktion bekommt als Eingabe die mittels *Calib3d.findChessboardCorners(...)* extrahierten Eckpunkte aus den Schachbrettbildern und die idealen Positionen der Eckpunkte. Diese Punkte werden jeweils verglichen und aus der Abweichung der Positionen werden dann die Kameramatrix und die Verzerrungskoeffizienten berechnet.

Alle Kamerakalibrierungsparameter werden anschließend in eine JSON-Datei serialisiert. Sobald die Software eine Verbindung zur Kamera herstellt, werden die gespeicherten Daten geladen und können verwendet werden.

Zur Laufzeit werden die intrinsischen Kameraparameter dann auf das aufgenommene Bild mit dem OpenCV-Befehl *Calib3d.undistort(...)* angewendet, wodurch es entzerrt wird und für weitere Berechnungen verwendet werden kann.

3.4.3 Lichtquellen

Der schwierigste Teil der Kalibrierung sind die Lichtquellen, da aufgrund von Produktionstoleranzen und Ungenauigkeiten beim Einkleben nicht alle LEDs die gleiche Farbe und Intensität aufweisen. Die Korrektheit dieser beiden Eigenschaften ist jedoch essenziell für die Aufnahme von BRDFs und muss daher ebenfalls kalibriert werden.

Da die Werte dabei von LED zu LED verschieden sind, muss jede LED einzeln kalibriert werden. Außerdem müssen verschattete Bereiche aus der Berechnung ausgeschlossen werden, da diese die Messergebnisse (Mittelwerte) verfälschen würden (mehr zu Maskierung in Kapitel 3.4.4).

Intensität Aufgrund der nicht immer gleichen Ausrichtung der LEDs und unterschiedlichen maximalen Helligkeitswerten, muss die Helligkeit jeder LED einzeln kalibriert werden. Zur Kalibrierung wird als erstes ein möglichst weißes, planares und diffuses Material auf die Drehscheibe des Gonioreflektometers gelegt. Hier wurde ein weißes Stück Pappe genommen.

Anschließend wird die Kamera auf die Mittelposition bewegt und der Mittelwert des Kamerabildes bestimmt. Da durch den kuppelbedingten unterschiedlichen Lichteinfallswinkel nicht einfach versucht werden kann, eine feste Intensität zu erreichen, wird ein Modell angewendet. Da das Material maximal diffus ist, kann von einer lambertschen Helligkeitsverteilung ausgegangen

werden.

Um dieses Modell anwenden zu können, muss zuerst die minimale und maximale Helligkeit bestimmt werden. Dafür wird einmal senkrecht beleuchtet und einmal in einem Beleuchtungswinkel von 90°. Die modellierten Helligkeitswerte werden dann anhand des bekannten Lichteinfallswinkels mit dem Lambertschen Kosinusgesetz berechnet.

Im nächsten Schritt wird jede LED einzeln auf eine feste Helligkeit gestellt und die Helligkeit des aufgenommenen Bildes mit dem modellierten Helligkeitswert verglichen. Wird hier ein Unterschied festgestellt, so wird die Helligkeit der LED iterativ so lange angepasst, bis die absolute Differenz unter einem Toleranzwert liegt. Im Code-Ausschnitt 9 ist der Algorithmus veranschaulicht.

Code-Ausschnitt 9: Kalibrierung der Helligkeit. Dieser Code wird für jede Lichtquelle solange ausgeführt, bis sie kalibriert ist.

```
void calibrateIntensity(Light currentLight)
{
  // berechne erwartete Helligkeit
  double theta = Light.getCurrentAngle();
  double expected =
    map(theta, 0.0, 1.0, minIntensity, maxIntensity);

  // berechne Mittelwert mit Schattenmasken
  Color mean = Camera.getMeanNoShadow();
  double measured = (mean.r + mean.g + mean.b) / 3.0;

  // berechne Unterschied
  int delta = (expected - measured);

  // Helligkeit ok --> naechste LED
  if (abs(delta) <= 5)
  {
    nextLight();
  }
  // Helligkeit anpassen
  else
  {
    double deltaFactor = 1.0 + 0.002 * delta;
    currentLight.rgb = currentLight.rgb * deltaFactor;
  }
}
```

Farbe Auch die Farbe der einzelnen LEDs ist nicht uniform, da durch Fertigungstoleranzen oft ein Rot- oder Blaustich wahrzunehmen ist. Da es sich um RGB-LEDs handelt, kann die Farbe jedoch einfach angepasst werden. Der Algorithmus zum Kalibrieren der Farbe folgt ebenfalls einem iterativen Schema. Hier wird jede LED einzeln auf weiß (255, 255, 255) gesetzt.

Auch hier wird der mittlere Farbwert des Kamerabildes bestimmt. An-

schließend wird geprüft, ob die gemessene Farbe einen Grauwert darstellt, die einzelnen Farbkanäle also bis auf einen Toleranzwert gleich sind. Ist dies nicht der Fall, so wird iterativ die dominante Farbe bestimmt und der entsprechende Farbkanal der LED angepasst. Dieser Algorithmus ist in Code-Ausschnitt 10 zu sehen.

Code-Ausschnitt 10: Kalibrierung der Farbe. Dieser Code wird für jede Lichtquelle solange ausgeführt, bis sie kalibriert ist.

```
void calibrateColor(Light currentLight)
{
  // berechne Mittelwert mit Schattenmasken
  Color mean = Camera.getMeanNoShadow();

  // pruefe ob Mittelwert grau ist
  if (isGray(mean))
  {
    nextLight();
  }
  // verringere dominante Farbe
  else
  {
    currentLight.maxComponent -= 5;
  }
}
```

Algorithmus Da Farbe und Intensität nicht nacheinander kalibriert werden können, wurden die beiden Kalibrierungsalgorithmen zu einem kombiniert. Insgesamt handelt es sich nach wie vor um einen iterativen Algorithmus, bei dem jede LED einzeln kalibriert wird. In jedem Iterationsschritt wird nun jedoch jeweils einmal die Helligkeit und einmal die Farbe kalibriert, wie in Code-Ausschnitt 11 zu sehen ist. Erst wenn beide Eigenschaften innerhalb ihrer jeweiligen Toleranz liegen, gilt die LED als kalibriert.

Code-Ausschnitt 11: Kalibrierung der Lichtquellen

```
void calibrateLight()
{
  // setze Kamera auf Mittelposition
  setCameraPosition(0);

  // iteriere ueber alle Lichtquellen
  for(Light currentLight in allLights)
  {
    // wiederhole bis kalibriert...
    while(!currentLight.isCalibrated())
    {
      turnOnLight(currentLight);
      calibrateIntensity(currentLight);
      calibrateColor(currentLight);
    }
    saveCalibration();
  }
}
```

Der ermittelte RGB-Wert zur Ansteuerung der entsprechenden LED kann nun abgespeichert werden. Da diese Kalibrierung mehrere Stunden dauern kann, werden alle Werte zwischengespeichert. Die Kalibrierung kann so zu einem späteren Zeitpunkt wieder aufgenommen werden.

3.4.4 Schatten-Maskierung

Ein großes Problem sowohl für die Kalibrierung der Lichtquellen, als auch für die Aufnahme von BRDFs ist die partielle Verschattung der Materialprobe durch den Kameraarm. Dieser befindet sich für einige Licht-Kamera-Winkel-Kombinationen zwischen der Lichtquelle und der Materialprobe auf der Drehscheibe, wodurch ein Schatten auf den aufzunehmenden Bereich geworfen wird.

Dieser Schatten ist problematisch, da zum einen die Mittelwertberechnungen für die Kalibrierung der Lichtquellen fehlerhaft werden. Zum anderen sind bei der Aufnahme von BRDFs in den schattierten Bereichen keine verwendbaren Informationen zur Reflexionscharakteristik des Materials vorhanden.

Um diesem Problem entgegenzuwirken, werden vor allen anderen Algorithmen Schattenmasken erstellt. Dabei müssen möglichst alle Kombinationen aus Lichtquelle und Kameraposition betrachtet werden. Um nur den Schatten zu erkennen, wird auch hier ein möglichst weißes, planares und diffuses Material auf der Drehscheibe platziert.

Für jede Kombination wird zuerst das Kamerabild binarisiert. Dieses Binärbild wird anschließend einmal dilatiert und 70-mal erodiert, um kleine Fehler zu beseitigen und die Schatten etwas zu vergrößern. Als strukturierendes Element für die Morphologie wird hier ein 5×5 Pixel großes Quadrat genutzt.

Wurde ein Schatten im Bild gefunden, existieren also schwarze Pixel, so wird es abgespeichert. Der Dateiname enthält dabei Informationen zur Kamera- und Lichtposition der Aufnahme.

Die gespeicherten Masken können später wieder geladen werden, um bei der Kalibrierung oder der Aufnahmen von BRDFs schattierte Bereiche auszugleichen.

3.5 Aufnahmeprogramme

Für die Aufnahme von BRDFs können mehrere Parameter eingestellt werden. Zum einen kann ausgewählt werden, wie viele Dimensionen bzw. Freiheitsgrade (engl. degrees of freedom, DoF) aufgenommen werden sollen. Von komplett diffusen Materialien bis zu einer kompletten Aufnahme aller Licht- und Kamerawinkelkombinationen ist dabei alles möglich. Zum anderen kann die Auflösung bzw. Schrittanzahl für Kamera- und Lichtwinkel, jeweils in Longitudinal- und Latitudinalrichtung angegeben werden. Die maximale Kameraschrittauflösung ist dabei von der maximalen Schrittanzahl der Schrittmotoren abhängig und die Lichtauflösung von der Anzahl der LEDs.

Da auch hier das Schattenproblem wie in 3.4.4 beschrieben auftritt, werden auch für die Aufnahme der BRDFs Schattenmasken benutzt. Diese werden allerdings nicht wie bei der Kalibrierung für die Bestimmung eines Mittelwerts genutzt, sondern um schattierte Bereiche mittels *Holefilling* bzw. *Inpainting* durch eine Approximation zu füllen.

Für das Inpainting werden zuerst die Bereiche definiert, in denen Bildinformationen fehlen, also Inpainting angewendet werden soll. Dazu dienen die Schattenmasken, da diese bereits die entsprechenden Bereiche enthalten. Anschließend können das Bild und die Maske an die OpenCV-Funktion *Photo.inpaint(...)* übergeben werden, welche das Inpainting durchführt. Die Funktion arbeitet dabei nach dem von Alexandru Telea in [Tel04] vorgestellten Verfahren. Abbildung 14 zeigt eine schattierte Aufnahme vor und nach dem Inpainting.

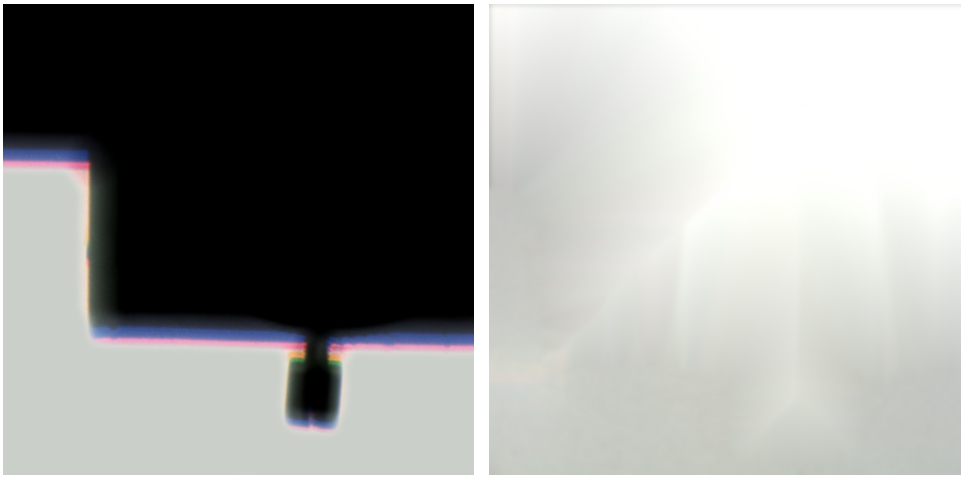


Abbildung 14: Links: Schattierte Aufnahme ohne Inpainting, rechts: gleiche Aufnahme mit Inpainting

Die Informationen zu den Licht- und Kamerawinkeln eines einzelnen Bildes werden in den Dateinamen kodiert. Das Präfix „ct" steht beispielsweise für den Theta-Winkel-Index (t) der Kamera (c), wohingegen „lp" für den Phi-Winkel-Index (p) der Lichtquelle (l) steht. Ein Dateiname sieht dann beispielsweise folgendermaßen aus: *mousepad_4dof_ct4cp15_lt2lp3.jpg*. Zusätzlich wird für die Gesamtaufnahme eine JSON-Datei mit Informationen zum Namen und zur Schrittauflösung gespeichert.

3.5.1 Diffuse BRDF (1 DoF)

Die einfachste Aufnahme ist die diffuse BRDF. Sie hängt lediglich vom vertikalen Lichteinfallswinkel ab und hat somit nur einen Freiheitsgrad. Der entsprechende Algorithmus ist in Code-Ausschnitt 12 als Pseudocode dargestellt.

Code-Ausschnitt 12: Aufnahmealgorithmus für eine diffuse BRDF

```
void capture1Dof()
{
  // setze Kamera auf Mittelposition
  setCameraPosition(0);

  // iteriere ueber alle Theta-Winkel
  for(float lightTheta = 0; lightTheta < 90; lightTheta += step)
  {
    Img weightedSum;
    Img maskSum;

    // iteriere ueber vier Phi-Winkel
    for (int i = 0; i < 4; i++) {
      // schalte LED fuer Winkelkombination ein
      setLight(lightTheta, lightPhis[i]);
      // summiere maskierte Bilder und Masken auf
      weightedSum.accumulate(Camera.getFrame(), mask);
      maskSum += mask;
    }

    // berechne gewichteten Mittelwert aus vier Bildern
    Img avgImage = weightedSum / maskSum;
    // fuelle schattierte Bereiche mit Inpainting
    avgImage.doInpainting();
    // speichere Bild ab
    avgImage.save();
  }
}
```

Um die Verschattung durch den Kameraarm weiter zu reduzieren, wird die Materialprobe für jeden Lichteinfallswinkel aus vier verschiedenen horizontalen Richtungen beleuchtet. Dies hat außerdem den Vorteil, dass der bei unidirektionaler Beleuchtung entstehende Helligkeitsverlauf dadurch kompensiert wird. Die einzelnen Bilder werden anschließend unter Verwendung der Schattenmasken gemittelt. Dabei werden schattierte Bereiche nicht in die Berechnung des Mittelwertes miteinbezogen. Mit diesem Verfahren können durch Inpainting entstehende Artefakte drastisch reduziert werden, da nur noch in sehr wenigen Bereichen Bildinformationen fehlen bzw. Inpainting betrieben werden muss. Es sind also mehr echte Bildinformationen vorhanden. Abbildung 15 zeigt alle Bilder einer fertig aufgenommenen 1-DoF-BRDF eines weißen Papierblatts.

Abbildung 15: Einzelbilder einer aufgenommenen 1-DoF-BRDF. Material: Weißes Papier

3.5.2 Rotationssymmetrische BRDF (2 DoF)

Rotationssymmetrische BRDFs sind zusätzlich zum vertikalen Lichteinfallswinkel vom vertikalen Kamerawinkel abhängig. Hier können also auch blickwinkelabhängige Materialien vermessen werden. Dabei wird angenommen, dass die Abstrahlcharakteristik entlang der Normalen symmetrisch ist, von einer Drehung um diese also nicht beeinflusst wird. Solche Materialien werden auch isotrope Materialien genannt.

Um dies zu erreichen, wird prinzipiell der gleiche Algorithmus wie für diffuse BRDFs verwendet. Allerdings wird dieser Algorithmus für alle gewünschten Kamerawinkel wiederholt. Es wird also zuerst der Kamerawinkel eingestellt und anschließend für diesen Kamerawinkel alle Lichtwinkel gemessen. Dann wird iterativ die Kamera auf den nächsten Winkel eingestellt.

Da hierfür deutlich mehr Winkelkombinationen aufgenommen werden müssen, dauert eine Aufnahme folglich deutlich länger. Auch hier können Schatten, wie beim diffusen Aufnahmeprogramm auch, durch Variieren des horizontalen Lichtwinkels sowie Inpainting kompensiert werden.

3.5.3 Halb-Isotrope BRDF (3 DoF)

Bei BRDFs mit drei Freiheitsgraden kommt zusätzlich zu der vertikalen Vermessung von Licht- und Kamerawinkel der horizontale Lichtwinkel hinzu. Die Abstrahlcharakteristik aus Sicht der Kamera wird also als isotrop angenommen, während der Lichteinfallswinkel nun nicht mehr rotationssymmetrisch angenommen wird. Daher werden diese BRDFs als „Halb-Isotrop" bezeichnet.

Algorithmisch betrachtet bedeutet dies, dass nicht wie vorher über vier horizontale Lichtwinkel gemittelt wird, sondern alle Phi-Lichtwinkel abgegangen werden müssen. Das heißt allerdings auch, dass ein Schattenausgleich durch Beleuchten aus mehreren Richtungen nicht mehr möglich ist. Schattenartefakte können hier also nur noch durch Inpainting ausgeglichen werden.

Alternativ zum Lichtwinkel könnte theoretisch auch der horizontale Kamerawinkel hinzugenommen werden. Dies ist aber bei gleichbleibendem horizontalen Lichtwinkel nicht besonders interessant, da so keine besonderen Materialeigenschaften aufgenommen werden können. Daher wurde auf diese Aufnahmemethode verzichtet.

3.5.4 Vollständige BRDF (4 DoF)

Um eine vollständige BRDF aufzunehmen, muss über alle möglichen Licht- und Kamerawinkelkombinationen sowohl in der Vertikalen, als auch in der Horizontalen iteriert werden. Dies ergibt insgesamt vier Freiheitsgrade oder eine vierdimensionale Beleuchtungsfunktion. Nimmt man die Pixelkoordinaten und die drei Farbkanäle hinzu, erhält man eine Gesamtanzahl von neun Dimensionen. Eine solche BRDF hat den Vorteil, dass sie alle verfügbaren Informationen enthält und somit die akkurateste Aufnahmemethode darstellt.

Dies nimmt allerdings zum einen viel Zeit für die Aufnahme in Anspruch, weshalb die Schrittauflösung sowohl für Kamera, als auch für Lichtquellen verringert werden sollte. Zum anderen ist auch der Speicherverbrauch nicht zu unterschätzen, da sehr viele Daten gespeichert werden müssen. Um dieses Problem zu bewältigen, kann beispielsweise die Bildauflösung reduziert werden. Auch hierbei hilft die Reduzierung der Schrittauflösung, da insgesamt weniger Bilder gespeichert werden müssen.

4 Rendering der aufgenommenen BRDFs

Nachdem eine BRDF aufgenommen wurde, kann sie in das Rendering-Programm geladen und auf virtuellen Objekten gerendert werden. Dabei muss für jede Art von BRDF anders vorgegangen werden, um jeweils die richtigen Bilder auszuwählen und anzuzeigen. Abbildung 16 zeigt die grafische Benutzeroberfläche des Renderers.

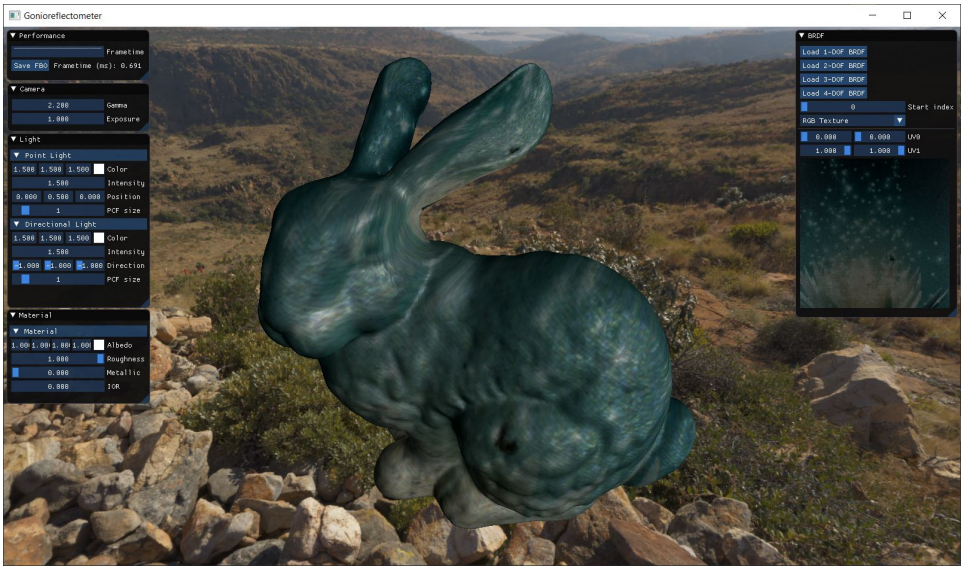


Abbildung 16: Grafische Benutzeroberfläche des BRDF-Renderers. Linke Panels (von oben nach unten): Performance-Anzeige, Kamera-Einstellungen, Lichtquellen, Materialien. Rechtes Panel: Laden von BRDFs, Rendermodus und Anpassung der UV-Koordinaten

4.1 Framework

Das Rendering-Programm wurde in C++ 17 und modernem OpenGL 4.6 mit *Direct-State-Access*, *Bindless-Textures* und den Bibliotheken *GLBinding* und *GLM* geschrieben. Für die graphische Benutzeroberfläche wurde *GLFW* und *ImGui* verwendet. Zum Laden der Bilddateien wurden die Bibliotheken *stb_image* und *Tinyfd* eingebunden. Für das Laden von 3D-Modellen und JSON-Dateien wurden *Assimp* und *RapidJSON* benutzt.

Um die Arbeit mit OpenGL zu erleichtern und häufig genutzte Funktionen wie das Laden von Modellen, Lichtquellen und Materialien nicht mehrfach programmieren zu müssen, wurde ein Framework implementiert. Dieses bietet zum einen Klassen zur Kapselung von OpenGL-Objekten wie Texturen, Shadern und Buffern. Zum anderen sind Klassen vorhanden, die den Datenaustausch zwischen CPU und GPU für Materialien, Lichtquellen sowie Mesh- bzw. Modelldaten verwalten.

Mit diesem Framework können schnell und einfach Testanwendungen implementiert werden. Es wurde darauf geachtet, OpenGL dabei nicht zu stark zu abstrahieren, damit die feinere Kontrolle mit reinem OpenGL erhalten bleibt. Der generelle Programmablauf ist in Abbildung 17 dargestellt.

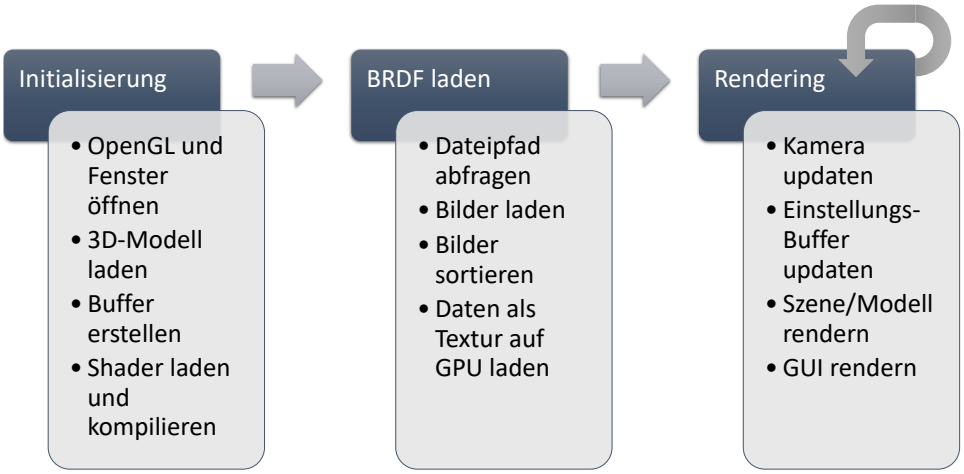


Abbildung 17: Programmablauf der Rendersoftware

Außerdem vereinfacht das Framework viele Arbeitsabläufe. Die Fenstererstellung mittels GLFW und die Initialisierung des OpenGL-Kontextes werden mit nur einem Befehl von der Window-Klasse abgearbeitet. Teile der graphischen Benutzeroberfläche können automatisch generiert werden und Performance-Messungen können mittels der Timer-Klasse und OpenGL-Queries direkt und präzise auf der Grafikkarte ausgeführt werden. Auch ein Screen-Filling-Quad kann mit nur einem Befehl gerendert werden.

Des Weiteren können Shader zur Laufzeit neu geladen und kompiliert werden. Auch einfache Präprozessor-Funktionen wie Defines sind beim Laden von Shadern möglich. Dabei werden Shader nur dann in das laufende Programm injiziert, wenn sie fehlerfrei kompilieren, um einen Absturz von OpenGL zu vermeiden.

Die Buffer-, Textur-, und VertexArray-Klassen kapseln die entsprechenden OpenGL-Objekte. Dabei wird der Zugriff durch Verwalten von aktuellen Speichergrößen und Anbindung an Datenstrukturen aus der C++-Standardbibliothek wie *std::vector* oder *std::array* erleichtert und Fehlerquellen minimiert. Außerdem wird eine Typisierung der OpenGL-Objekte durch C++-Templates erreicht, wodurch besonders Buffer deutlich einfacher verwendet werden können.

Alle OpenGL-Objekte werden üblicherweise nur durch eine Integer-ID verwaltet. Daher kann weder Typsicherheit, noch Kontrolle über deren Lebenszeit garantiert werden, was häufig zu Fehlern wie *Memory-Leaks* führen kann. Um dem entgegenzuwirken, wurde für alle OpenGL-Objekte das Design-Pattern „Ressourcenbelegung ist Initialisierung" (auch *resource acquisition is initialization*, kurz *RAII*) implementiert. Hierbei wird für jeden OpenGL-Objekt-Typ ein neuer C++-Typ definiert. Sobald nun eine Instanz eines solchen Objekts

erstellt wird, wird automatisch die zugehörige *glCreate*-Funktion aufgerufen und die resultierende OpenGL-ID intern gespeichert. Wird diese Instanz gelöscht (z.B. weil sie out-of-scope geht), wird automatisch der Destruktor aufgerufen, welcher wiederum den objektspezifischen *glDelete*-Befehl aufruft. Durch diese Technik ist sichergestellt, dass die Lebensdauer von OpenGL-Objekten auf der CPU sowie der GPU synchronisiert sind und es nicht zu unnötigem Speicherverbrauch kommt.

Die Material-, Light-, und Camera-Klassen dienen primär als Datenkapsel für die entsprechenden Parameter. Sie sind Grafikspeicher-aligned, d.h. sie können als Ganzes ohne zusätzliches Padding in einen GPU-Buffer geladen werden. Zusätzlich bieten die Klassen einige Hilfsfunktionen, wie beispielsweise das Erstellen und Verwalten von Shadow-Maps in der Light-Klasse, das automatische Wechseln zwischen Textur und Farbe in der Material-Klasse oder das Verarbeiten von Kamerabewegungen und die Berechnung der View-Matrix in der Camera-Klasse.

Für die Objektverwaltung kommen die Klassen Mesh und Scene zum Einsatz. Die Mesh-Klasse kann 3D-Modelle mittels der Bibliothek Assimp laden und verwaltet deren Vertex- und Indexlisten. Zusätzlich speichert sie pro Objekt ein Material, eine Modelmatrix und eine Bounding Box.

Die Scene-Klasse verwaltet bis auf die zu verwendenden Shader das gesamte Rendering mittels *GPU-Driven-Rendering*. Hier werden alle Lichtquellen, Meshes und die Kamera verwaltet. Da alle Objekte mit nur einem Draw-Call gerendert werden, müssen alle Vertices und Indices in jeweils einen großen Buffer geladen werden. Die Listen aus den einzelnen Mesh-Objekten werden dazu unter Verwendung von *Multi-Threading* konkateniert und als Ganzes in entsprechende *Multi-Draw-Buffer* geladen. Vor dem eigentlichen Rendern wird anschließend automatisches *View-Frustum-Culling* im *Compute-Shader* auf der Grafikkarte ausgeführt, um nicht sichtbare Objekte gar nicht erst rendern zu müssen. Ob ein Objekt gerendert werden soll oder nicht, wird in den *Indirect-Draw-Buffer* geschrieben. Dieser Buffer enthält außerdem Informationen über den Startindex und die Indexanzahl pro Objekt und wird vom OpenGL-Render-Befehl *glMultiDrawElementsIndirect(...)* für das eigentliche Rendern der Szene verwendet.

4.2 Laden der Bilddaten

Für das spätere Rendering ist es wichtig, dass die Reihenfolge der Bilder korrekt ist. Sobald der Benutzer einen Aufnahmetyp und -ordner im Tinyfd-Dateiauswahl-Dialog (C++ Befehl *tinyfd_selectFolderDialog(...)*) wählt, werden alle darin enthaltenen Bilder mittels *stbi_load(...)* geladen und die zugehörigen Winkelkombinationen aus deren Dateinamen extrahiert. Anschließend wird diese Liste sortiert, sodass die Bilder in einer eindeutigen Reihenfolge vorliegen. Wichtig ist hierbei, dass die Reihenfolge später im Shader rekonstruiert bzw. berechnet werden kann.

Da nun die Bilderliste vollständig und richtig sortiert ist, kann sie in den Texturspeicher der Grafikkarte geladen werden. Dazu wird ein zweidimensionales Texturarray verwendet. Für dieses wird ausreichend Speicherplatz allokiert und anschließend mit den Bilddaten gefüllt.

Zusätzlich wird die beim Aufnehmen gespeicherte JSON-Datei mit dem Befehl *rapidjson::Document::Parse(...)* geladen. Die Informationen über die Schrittauflösung für Kamera und Lichtquellen wird in ein Struct geschrieben, das ebenfalls auf die Grafikkarte geladen wird. Dies geschieht mittels eines *Shader Storage Buffer Objects (SSBO)*, auf welchen später im Shader zugegriffen werden kann, um die richtigen Textur-Offsets bzw. -Layer berechnen zu können.

4.3 Anzeigen der Bilddaten

Das Anzeigen der Bilddaten bzw. der aufgenommenen BRDFs geschieht komplett im Fragment-Shader. 3D-Objekte werden in normaler OpenGL-Weise über Vertex-Array-Objects und Vertex-Shader gerendert. Im Fragment-Shader werden anschließend je nach Einstellung die Bilddaten auf unterschiedliche Art und Weise auf die Objekte angewendet. Das hierbei angewandte Verfahren ist ähnlich zum sog. *Hatching* [PHWF01], allerdings erweitert auf viele mögliche Winkelkombinationen.

Leider ist nicht immer die komplette BRDF-Textur verwendbar, da eine aufgenommene Materialprobe kleiner sein kann als das aufgenommene Kamerabild. Daher wurde in der Benutzeroberfläche eine Möglichkeit zum Begrenzen des Bildbereichs implementiert. Die resultierenden minimalen und maximalen Texturkoordinaten werden zusammen mit dem in 4.2 beschriebenen SSBO auf die Grafikkarte geladen.

Im Fragment-Shader werden dann als Erstes die ursprünglichen auf die begrenzten Texturkoordinaten projiziert. Die nächsten Schritte werden je nach Anzahl an Freiheitsgraden für verschiedene Schrittauflösungen wiederholt. Hier wird beispielhaft das Vorgehen für 1-DoF-BRDFs beschrieben.

Zuerst wird der relevante Winkel bestimmt, beispielsweise der Lichteinfallswinkel, also der Winkel zwischen Lichtvektor und Normale. Dieser wird dann auf die entsprechende verfügbare Schrittanzahl projiziert.

Der Theta-Winkel ist dabei immer der vertikale Winkel und kann durch das Skalarprodukt des entsprechenden Winkels zur Normale bestimmt werden. Er liegt im Bereich $[0, \frac{\pi}{2}]$.

Der horizontale, also Phi-Winkel muss hingegen im Tangentenraum der gerenderten Oberfläche bestimmt werden. Dazu wird der Tangentenraum durch einen Geometry-Shader für jedes gerenderte Dreieck berechnet, falls keine Tangentendaten im 3D-Modell vorhanden sind. Im Fragment-Shader werden nun die relevanten Vektoren in den Tangentenraum transformiert. Der gesuchte Winkel kann dann durch die *atan2*-Funktion bestimmt werden. Diese Funktion ist prinzipiell ein Arkustangens mit dem Unterschied, dass

die Eingabeparameter auf einen von vier Quadranten gemappt werden, um Werte im Bereich [-π, π] (also eine ganze Umdrehung) zu erhalten.

Nun werden die Interpolationsparameter berechnet. Die beiden zu verwendenden Textur-Layer werden durch Auf- und Abrunden des projizierten Winkels bestimmt. Der Nachkommaanteil stellt den Alpha-Wert für die lineare Interpolation dar.

Anschließend erfolgt der Texturzugriff mit den berechneten Texturkoordinaten und -Layern. Die beiden Farbwerte aus der Textur werden dann mit dem GLSL-Befehl *mix(...)* und dem vorher berechneten Alpha-Wert linear interpoliert. Der Code-Ausschnitt 13 zeigt den vereinfachten GLSL-Code dazu. Die daraus resultierende Farbe kann dann auf verschiedene Arten angezeigt werden, was im Folgenden näher beschrieben wird.

Code-Ausschnitt 13: Algorithmus zum Rendern einer diffusen BRDF

```
vec3 getBRDF()
{
    // Zuschneiden der Textur durch Remapping der UV-Koordinaten
    vec2 uv = map(texCoord, 0.0, 1.0, uv0, uv1);

    // Lichteinfallswinkel berechnen
    float lightAngle = acos(max(dot(normal,
        getLightDirection(light, worldPos)), 0.0));

    // Lichtwinkel auf Anzahl an Texturen mappen
    float mappedAngle =
        map(lightAngle, 0.0, PI / 2, 0, lightThetaSteps - 1);

    // Untere und obere Textur bestimmen
    int lowerLayer = int(floor(mappedAngle));
    int upperLayer = int(ceil(mappedAngle));

    // Interpolationsfaktor bestimmen
    float interp = fract(mappedAngle);

    // Texturzugriffe mit UV und Textur-Layern
    vec3 lowerCol = texture(brdfTex, vec3(uv, lowerLayer)).rgb;
    vec3 upperCol = texture(brdfTex, vec3(uv, upperLayer)).rgb;

    // Texturwerte interpolieren
    vec3 brdfColor = mix(lowerCol, upperCol, interp);
    return brdfColor;
}
```

4.3.1 Farbtextur

Soll die Farbtextur angezeigt werden, muss nach dem obenstehenden Vorgehen nichts weiter getan werden. Die erhaltene Farbe kann einfach auf das Objekt bzw. das Fragment angewendet und angezeigt werden.

4.3.2 Grauwerttextur

Möchte man nur den Grauwert, also quasi die Helligkeit der BRDF-Textur verwenden, so muss der Farbwert in einen Grauwert umgewandelt werden. Dafür wird die folgende an die menschliche Wahrnehmung angepasste Formel verwendet:

$$Helligkeit = Farbe * \begin{pmatrix} 0.3 \\ 0.59 \\ 0.11 \end{pmatrix}$$

Der hinten-stehende Vektor spiegelt dabei die Empfindlichkeit der Farbwahrnehmungszäpfchen im menschlichen Auge wieder und wird sehr häufig für die Transformation von RGB zu Graustufen verwendet, da er bessere Ergebnisse liefert, als ein einfaches Mitteln der drei Kanäle.

Vorteilhaft bei dieser Darstellungsvariante ist, dass nun zusätzlich eine Materialfarbe definiert werden kann, mit der das Objekt eingefärbt wird. Dafür wird die Helligkeit mit der Materialfarbe multipliziert.

4.3.3 Mittlere Farbe

Für das Rendering mit der gemittelten Farbe der aufgenommenen BRDF muss der Mittelwert jedes Textur-Layers berechnet werden. Je nach Winkel wird also dieser Mittelwert für ein aufgenommenes Bild berechnet und auf das zu rendernde Objekt angewendet.

Dafür kann praktischerweise die Hardware-beschleunigte Mittelwertberechnung der Grafikkarte ausgenutzt werden. Das hierbei verwendete Verfahren nennt sich *Mipmapping* und kann durch den OpenGL-Befehl *glGenerateMipmap(...)* gestartet werden. Hierbei werden zwar mehrere Mipmap-Stufen generiert, für den Mittelwert ist jedoch nur die höchste Mipmap-Stufe relevant, da sie den Mittelwert des gesamten Bildes darstellt.

4.3.4 Mittlere Helligkeit

Zum Rendern mit mittlerer Helligkeit wird prinzipiell eine Kombination aus Grauwerttextur und mittlerer Farbe verwendet: Zuerst wird mittels Mipmapping der Mittelwert aus der Textur bestimmt. Anschließend wird dieser Farbwert mit dem oben beschriebenen Skalarprodukt in einen Grau- bzw. Helligkeitswert transformiert. Auch hier kann wieder eine Materialfarbe oder eine zusätzliche Textur als Grundfarbe verwendet werden.

5 Weitere Aufnahmeverfahren

Beim Bau des Gonioreflektometers wurde sich bewusst gegen die herkömmliche Zwei-Arm-Bauweise entschieden. Durch die gleichmäßige Verteilung

der Lichtquellen in der Kuppel kann nämlich nicht immer nur aus einer Richtung, sondern auch aus allen Richtungen gleichzeitig beleuchtet werden. Dies ist besonders für solche Anwendungsgebiete, bei denen eine möglichst gleichmäßige Ausleuchtung wichtig ist, von großem Vorteil. Im Folgenden werden die weiteren Aufnahmeverfahren, welche mit dem im Rahmen dieser Arbeit konstruierten Gerät möglich sind, genauer beschrieben.

5.1 3D-Scan

Für einen 3D-Scan muss anstatt einer Materialprobe ein plastisches Objekt auf der Drehscheibe platziert und vom Gonioreflektometer aufgenommen werden. Da es hierbei nur auf die Geometrie des Objektes und nicht dessen Reflexionseigenschaften ankommt, ist der Beleuchtungswinkel nicht von Bedeutung. Im Gegenteil: Eine möglichst uniforme Beleuchtung von allen Seiten ist für die Aufnahme eines 3D-Scans von Vorteil, da alle Features des Objektes dadurch besser sichtbar für die Kamera werden.

Wichtig ist also, das zu scannende Objekt von möglichst vielen Perspektiven aus aufzunehmen. Dazu werden als erstes alle Lichtquellen mit einer geringen Helligkeit eingeschaltet, um eine gute Ausleuchtung zu erreichen, ohne dabei die Kamera überzubelichten. Anschließend werden Kameraarm und Drehscheibe jeweils schrittweise bewegt und Bilder aufgenommen. Je mehr Winkelkombinationen hierbei aufgenommen werden, desto mehr Informationen sind für das spätere Berechnen des 3D-Modells verfügbar. Dadurch steigt logischerweise auch die Qualität der Modellberechnung.

Die aufgenommenen Bilder können nun in ein beliebiges Fotogrammetrieprogramm geladen werden. Für die vorliegende Arbeit wurde die Software *Agisoft Metashape* verwendet, da sie sehr gute Ergebnisse liefert und außerdem eine 30 tägige kostenlose Testversion bietet. Andere Programme wie beispielsweise *Autodesk ReCap* bieten ähnliche Funktionen, sind aber nicht oder nur sehr eingeschränkt kostenlos nutzbar.

Das Programm generiert nun eine Punktewolke und ein Dreiecks-Mesh aus den geladenen Bilddaten. Zusätzlich wird außerdem eine Textur mit den zugehörigen UV-Koordinaten erstellt. Dieses 3D-Modell kann nun in ein beliebiges CAD-Programm oder einen 3D-Renderer geladen und angezeigt werden. Abbildung 18 zeigt den Workflow in Agisoft Metashape zur Erstellung eines 3D-Modells.

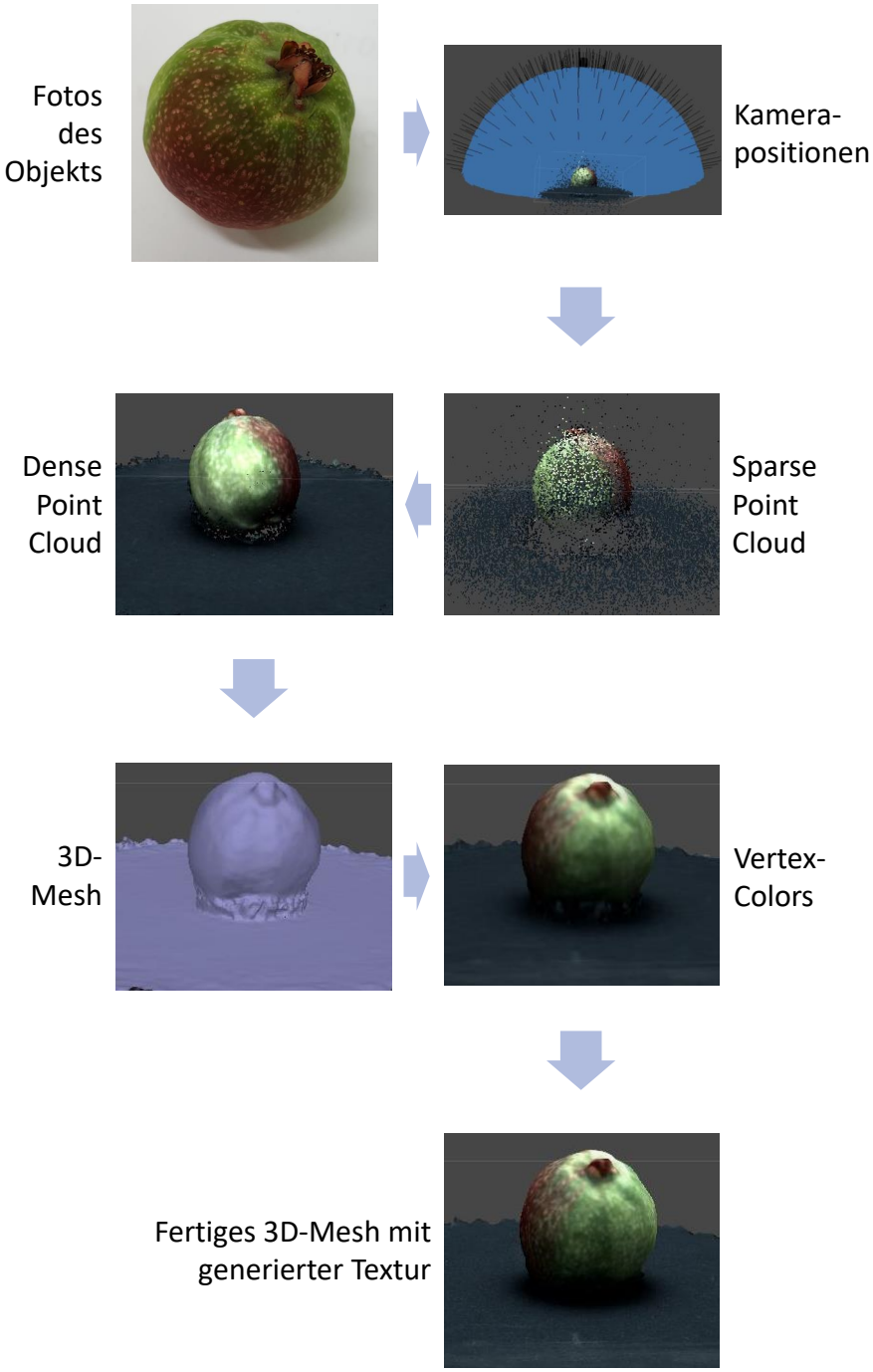


Abbildung 18: Workflow für das Generieren eines 3D-Modells aus Bildern eines 3D-Scans in Agisoft Metashape

5.2 Lichtfeld-Scan

Der Lichtfeld-Scan funktioniert prinzipiell genauso, wie der 3D-Scan. Auch hier wird das aufzunehmende Objekt bei gleichmäßiger Beleuchtung aus möglichst vielen Winkeln fotografiert.

Der Unterschied zum 3D-Scan liegt in der Verarbeitung der Bilddaten, da hier kein 3D-Modell erstellt wird. Zum Anzeigen des gescannten Objektes wird ein Billboard oder ein Screen-Filling-Quad gerendert. Anschließend wird für jedes dabei gerenderte Pixel der genaue Kamerawinkel in Weltkoordinaten bestimmt. Der so berechnete Kamerastrahl wird nun auf die aufgenommenen Bilder und die korrespondierenden Texturkoordinaten projiziert und als Farbe dem jeweiligen Pixel zugewiesen.

Das prinzipielle Vorgehen ist also vergleichbar mit klassischen zweidimensionalen Lichtfeldaufnahmen. Da die Aufnahmen hier jedoch nicht nur in einer Ebene, sondern auf einer Halbkugel liegen, muss das Verfahren auf sphärische Lichtfelder angepasst werden.

5.3 Light-Stage

Das Ziel einer Light-Stage ist es, ein Objekt so zu beleuchten, als würde es sich in einer bestimmten Umgebung befinden. Ausgangspunkt für das Light-Staging ist daher eine Environment-Map. Dies ist prinzipiell ein 360°-Foto einer beliebigen Umgebung. Dieses Bild wird nun an genau den Punkten ausgelesen, an denen sich in der Kuppel die Lichtquellen befinden. Anschließend wird jede Lichtquelle auf die entsprechende Farbe in der Environment-Map gesetzt. Das Objekt ist nun so beleuchtet, als stünde es in der Umgebung, die in der Environment-Map aufgenommen wurde. Das Objekt kann dann aus einem beliebigen Winkel mit der Kamera aufgenommen werden.

Theoretisch können für die Environment-Map sogar Videos verwendet werden, wodurch das Objekt dynamisch beleuchtet werden würde. Auch die Aufnahme könnte dann ein Video sein. Diese Erweiterung wurde jedoch nicht in dieser Arbeit implementiert. Genutzt wird dieses Verfahren hauptsächlich in modernen Filmen, welche reale und computergenerierte Bilder kombinieren. Die realen Protagonisten können somit beispielsweise mit dem Licht einer virtuellen Explosion bestrahlt werden. Natürlich sind die dabei verwendeten Geräte weitaus größer, als das in dieser Arbeit gebaute Gonioreflektometer.

Wichtig anzumerken ist, dass als Environment-Maps Rektangularprojektionen verwendet werden. Hierbei wird ein sphärisches Bild auf eine 2D-Ebene projiziert. Dieses Prinzip ist beispielsweise von Landkarten bekannt. Die x-Koordinate des Bildes repräsentiert dann den Phi-Winkel und die y-Achse den Theta-Winkel. Da das Gonioreflektometer nur in der oberen Hemisphäre leuchten kann, wird auch nur die obere Hälfte der Environment-Map verwendet. Ein weiterer wichtiger Aspekt ist die Filterung der Environment Map. Durch eine starke Glättung des Bildes (z.B. durch eine Gaußfilterung) ist die

spätere Ausleuchtung eines Objektes in der Light-Stage deutlich homogener und nicht von „Ausreißern" verfälscht.

6 Evaluation

Es ist sehr schwierig, für die gescannten Materialien ein Vergleichsbild mit einer mathematischen Funktion als BRDF (z.B. Blinn-Phong) zu rendern. Auch Vergleichsdaten aus anderen Publikationen sind nicht vorhanden oder nicht auf diese Anwendung übertragbar. Der Großteil der Evaluation besteht daher aus dem subjektiven Vergleich des gerenderten Materials zur realen Materialprobe. Nach Möglichkeit wird dazu ein gerendertes Bild einem Foto der Materialprobe gegenübergestellt. Im Folgenden wird die Qualität der aufgenommenen BRDFs sowie der anderen Aufnahmeverfahren evaluiert.

6.1 BRDF-Messung

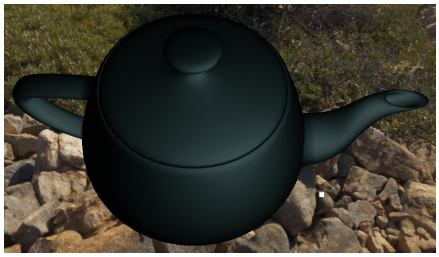
Die Aufnahme von Materialien verlief stets zuverlässig. Das Gonioreflektometer läuft also sehr stabil, ohne, dass während einer Aufnahme eingegriffen werden muss. Des Weiteren ist auch die Bewegung der Motoren als sehr präzise zu bewerten, da selbst nach einer 20-stündigen Aufnahme alle Motoren noch korrekt kalibriert waren. Leider können sich die Materialproben durch die Vibrationen der Motoren jedoch leicht verschieben, was das Ergebnis verfälscht. Diesem Problem kann jedoch durch Festkleben der Materialprobe mit doppelseitigem Klebeband entgegengewirkt werden.

Durch die Kamera-Live-Ansicht, der Ausgabe des Nachrichtenaustausches mit dem Arduino und einer Prozentangabe in der Ansteuerungssoftware konnte außerdem jederzeit der aktuelle Status der Aufnahme eingesehen werden. Auch die vielen Kalibrierungs- und Einstellungsmöglichkeiten in der Benutzeroberfläche haben sich als sehr nützlich erwiesen, da verschiedene Aufnahmen schnell eingestellt und gestartet werden können, ohne das Programm neu starten zu müssen. Die Fehlersuche und das Finden der richtigen Einstellungen für eine Aufnahme wurden dadurch ebenfalls erleichtert.

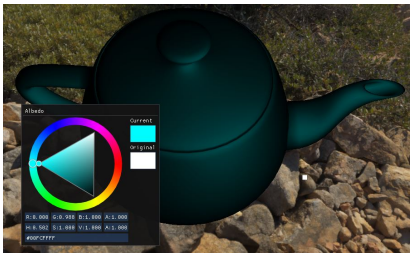
Auch der Renderer erfüllt seinen Zweck. Aufgenommene BRDFs können einfach ausgewählt und die Bilder automatisch in der richtigen Reihenfolge in den Grafikspeicher geladen werden. Die jeweiligen Shader für verschiedene Freiheitsgrade werden dann geladen und rendern die Bilddaten auf entsprechende Art und Weise. Das Rendering der Materialien verläuft flüssig und ohne Ruckeln, sodass sich der Benutzer in Echtzeit in der virtuellen Szene bewegen kann. Zudem können Fehler an den Rändern der Aufnahme sehr einfach durch das Textur-Zuschneide-Tool entfernt werden. Die verschiedenen Rendermodi können ebenfalls zur Laufzeit ausgewählt und ohne Verzögerung auf das Material-Rendering angewendet werden. Abbildung 19 zeigt ein aufgenommenes Material in den verschiedenen Rendermodi.

RGB-Textur

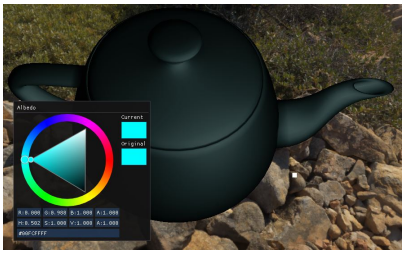
Grauwert-Textur mit Materialfarbe



gemittelte Farbe



gemittelte Intensität mit Materialfarbe



Blinn-Phong-Modell mit Materialfarbe

Abbildung 19: Rendermodi für das BRDF-Rendering. Material: Mousepad

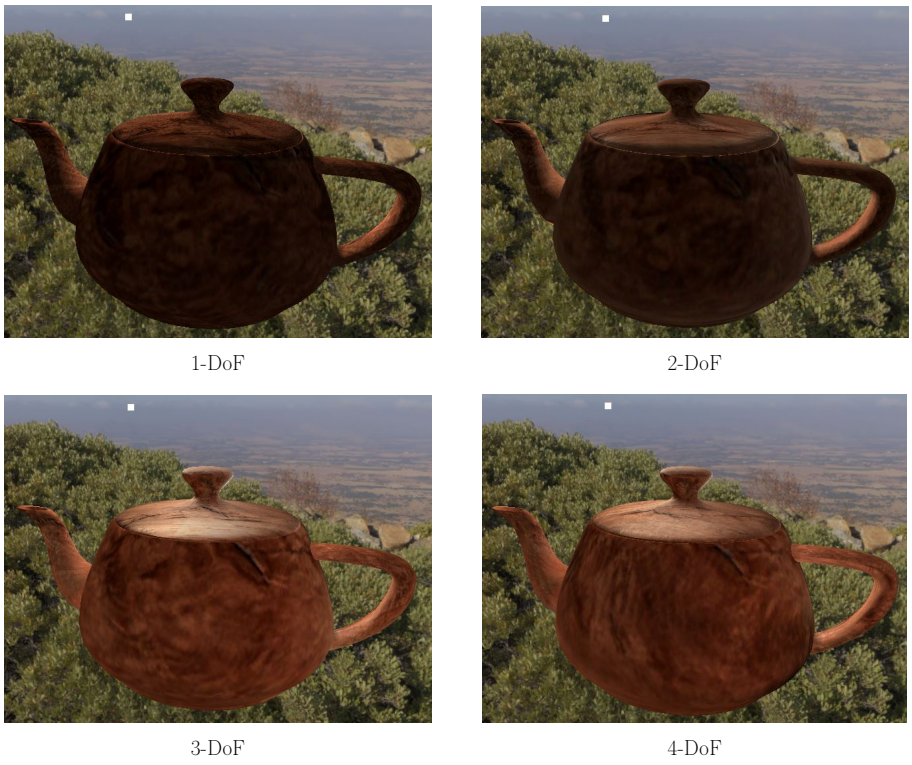
Problematisch beim Rendern der aufgenommenen BRDFs ist besonders der Speicherverbrauch, da immer alle Bilder in den Grafikspeicher (VRAM) geladen werden. Bei 1- und 2-DoF-Aufnahmen passen die Bilder mit einer Auflösung von 1024×1024 noch sehr gut in den VRAM, da es sich in der Regel nur um etwas über 100 Stück handelt. Bei größeren Aufnahmen muss jedoch sowohl die Bild-, als auch die Schrittauflösung der Aufnahme stark verringert werden, damit die Daten noch auf die Grafikkarte passen. Werden die Bilder in komprimierter Form hochgeladen, kann die Auflösung zwar höher sein, jedoch verlängern sich die Ladezeiten dadurch enorm.

Auch die Aufnahmezeiten sind bei größeren Aufnahmen (besonders bei 4-DoF-BRDFs) sehr groß. Eine Aufnahme mit mittlerer Schrittauflösung kann bereits zwischen 15 und 20 Stunden in Anspruch nehmen. Kleinere Aufnahmen sind hingegen schon in einigen Minuten abgeschlossen.

Die meisten Materialien können mit diesem Gonioreflektometer aufgenommen bzw. vermessen werden. Einschränkungen gibt es lediglich bei der physischen Größe der Materialprobe, da diese nicht viel größer als die Drehscheibe sein darf. Außerdem sind die Ergebnisse für dunkle Materialien (z.B.

dunkler Stoff oder schwarzes Plastik) schlecht, da die LEDs nicht genügend Licht abstrahlen, um diese Materialien ausreichend auszuleuchten.

In Abbildung 20 ist das selbe Material mit verschiedenen Anzahlen an Freiheitsgraden aufgenommen und gerendert dargestellt. Abbildung 21 zeigt verschiedene gerenderte Materialien, die als 1-DoF-BRDF aufgenommen und auf ein Teapot-Modell angewendet wurden.



1-DoF 2-DoF

3-DoF 4-DoF

Abbildung 20: Renderings mit verschiedenen DoFs des gleichen Materials (Holz). Die 3- und 4-DoF-Aufnahmen wirken heller, weil aufgrund von nicht ausreichendem Speicherplatz nicht so viele (und somit nicht die dunkleren) Theta-Winkel aufgenommen werden können. Man kann deutlich die spekularen Reflexionen der Lichtquelle (weißer Punkt) auf dem Deckel des Teapots sehen.

Bei den Materialien ist aufgrund der langen Aufnahmezeiten bei vielen Freiheitsgraden darauf zu achten, dass diese während der Aufnahme nicht ihre Reflexionseigenschaften verändern. Besonders organische Materialien wie Blätter können während der Aufnahme vertrocknen und die Aufnahme unbrauchbar machen.

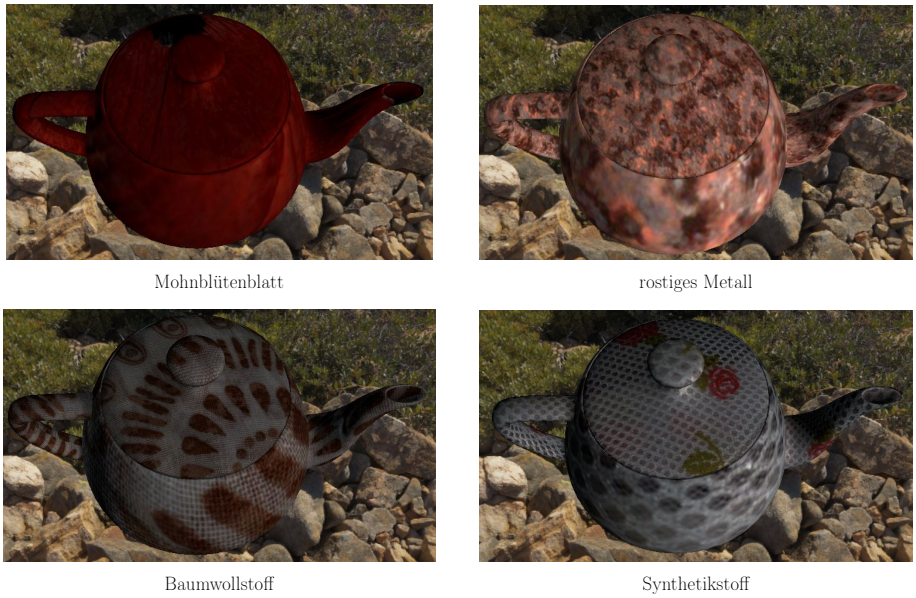


Abbildung 21: Verschiedene Materialien als 1-DoF-BRDF

Besonders hervorzuheben ist, dass mit diesem Gonioreflektometer auch sehr spezielle Lichteffekte aufgenommen werden können. Anisotrope Materialeigenschaften wie Retroreflexion sind in 3- und 4-DoF-Aufnahmen direkt enthalten. Auch Mikroreflexionen an Mikrofacetten, wie beispielsweise das Schimmern des Mousepad-Materials oder Seide können mit Leichtigkeit aufgenommen werden und sind auch im späteren Rendering deutlich zu erkennen. Zum Teil sind in den Aufnahmen sogar dreidimensionale Eigenschaften wie Vertiefungen oder Erhebungen sichtbar. Diese Materialeigenschaften sind mit herkömmlichen BRDF-Modellen wie dem Blinn-Phong-Modell [Bli77] oder dem Cook-Torrance-Modell [CT81] nicht zu bewältigen. Abbildung 22 zeigt eine Nahaufnahme einer solchen Mikroreflexion.

Abbildung 22: Nahaufnahme der Mikroreflexion auf dem Material eines Mousepads

6.2 Weitere Aufnahmenverfahren

Neben den aufgenommenen BRDFs ist auch die Evaluation der anderen Aufnahmeverfahren interessant. Da diese allerdings nicht Schwerpunkt dieser Arbeit sind, werden die Ergebnisse hierbei nur grob besprochen.

6.2.1 3D-Scan

Aus den gescannten Objekten wurde in der Regel ein sehr realitätsnahes und qualitativ hochwertiges 3D-Modell generiert. Eine Aufnahme dauerte dabei etwa zehn Minuten. Abbildung 23 zeigt beispielhaft einige generierte 3D-Modelle und deren reale Vorbilder.

Abbildung 23: Links: Foto von realem Objekt, rechts: Rendering des generierten 3D-Meshs

Probleme gab es lediglich in manchen Fällen bei sehr dunklen bzw. schwarzen Objekten, da hier Teile des Objekts nicht erkannt wurden und im resultierenden Mesh fehlten. Auch Objekte mit wenigen markanten Stellen („Features") wurden nicht gut in ein Mesh umgewandelt, da auch hier zu wenig Informationen aus den Bildern extrahiert werden konnten.

Leider fehlt in den generierten 3D-Modellen stets die Unterseite des gescannten Objekts, da hierfür aufgrund der Bauweise des Gonioreflektometers keine Bilddaten zur Verfügung stehen. Meist handelt es sich hierbei jedoch

ohnehin um eine planare Auflagefläche, weshalb dies in den meisten Fällen kein großes Problem darstellt.

Oft wurden außerdem kleine Artefakt-Meshes in der näheren Umgebung um das gescannte Objekt herum generiert, da die Software hier ebenfalls „Objekte" gefunden hat. Diese konnten jedoch mit entsprechenden Tools in der Software sehr schnell und einfach entfernt werden.

6.2.2 Lichtfelder

Der Lichtfeld-Scan ist im Vergleich zum 3D-Scan von deutlich schlechterer Qualität im Hinblick auf die visuellen Ergebnisse. Abbildung 24 zeigt die Aufnahme eines Lichtfeldes eines Plastikdrachens, einer Zierquitte, sowie einer Drahtspule.

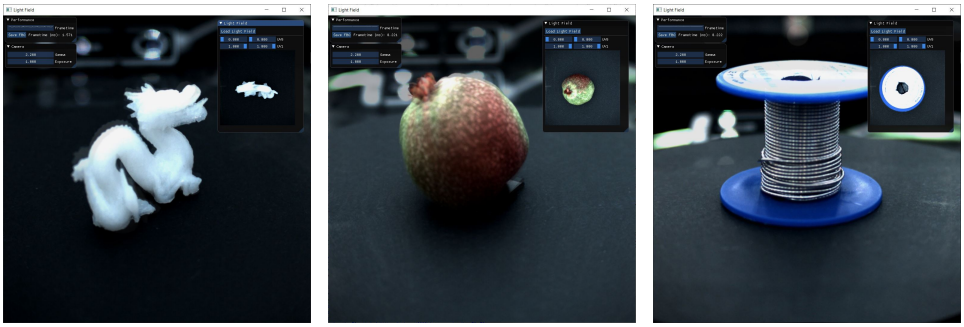


Abbildung 24: Rendering der Lichtfelder eines Plastikdrachen, einer Zierquitte und einer Drahtspule

Prinzipiell wird hier lediglich das Rotieren einer virtuellen Kamera auf die aufgenommenen Bilder übersetzt. Dadurch hat man beim Betrachten der Aufnahme den Eindruck, als würde man die Kamera in Echtzeit innerhalb des Gonioreflektometers um das Objekt auf der Drehscheibe herum bewegen.

Da hierbei kein Feature-Point-Matching betrieben wird, verändert das aufgenommene Objekt zum Teil jedoch seine Position im Bild. Platziert man das Objekt perfekt mittig auf der Drehscheibe, verringern sich diese Artefakte deutlich. Allerdings bleibt das Ergebnis aufgrund der Interpolation stets etwas unscharf.

Dieses Aufnahme- bzw. Renderverfahren eignet sich somit eher als Debug-Ansicht zur Fehlersuche und als „See-Through". Für eine realistischere Repräsentation des Objekts sollte deshalb zum 3D-Scan gegriffen werden.

6.2.3 Light-Stage

Das Light-Staging führte zu erstaunlich guten Ergebnissen. Die hiermit beleuchteten Objekte wirken deutlich anders, je nachdem in welcher Umgebung sie beleuchtet werden. Bei Verwendung der Environment-Map eines Waldes

ist beispielsweise ein deutlicher Grünstich in der Beleuchtung zu sehen, als ob das Objekt durch Blätter hindurch beleuchtet werden würde. In einer hölzernen Lagerhalle wird es dagegen eher bräunlich angeleuchtet. Abbildung 25 zeigt verschiedene Objekte in diversen Lichtverhältnissen.

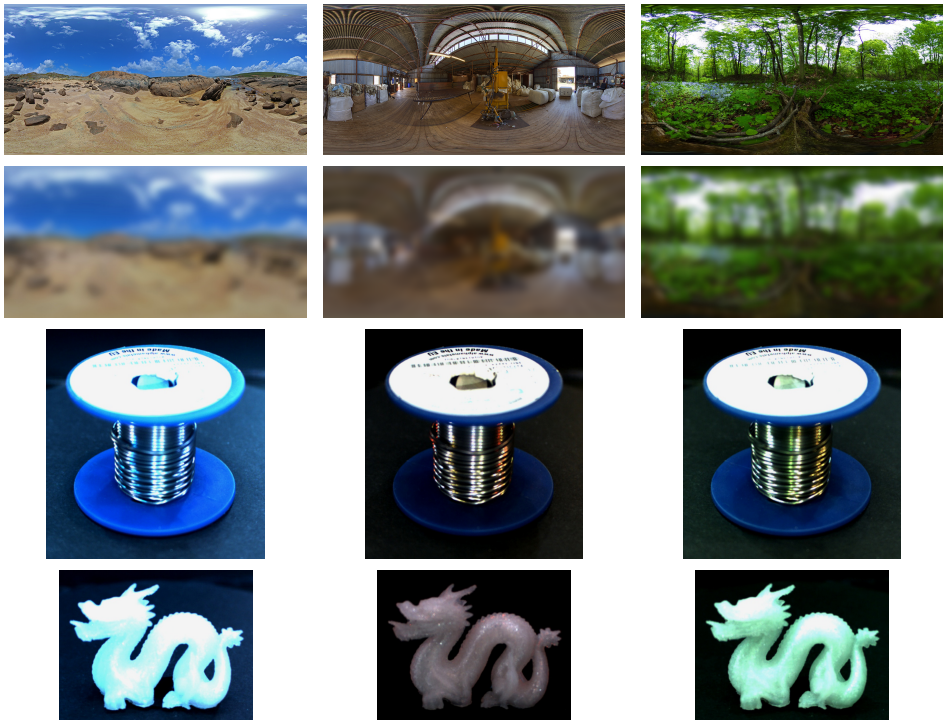


Abbildung 25: Light-Staging in verschiedenen Umgebungen. V.l.n.r.: Wüste, Lagerhalle und Wald. V.o.n.u.: Environment-Map in Rektangularprojektion, gefilterte Env.-Map, Foto von beleuchteter Drahtspule und Plastikdrache

Anfangs gab es Zweifel, ob das Light-Staging überhaupt brauchbare Ergebnisse liefern könnte, da aufgrund des geringen Speicherplatzes des Arduinos nur wenige Lichtquellen auf einmal mit jeweils unterschiedlichen Farben angesteuert werden können. Hierfür wurde noch ein Befehl zum Befehlssatz hinzugefügt, um den Speicherverbrauch zu reduzieren. Damit können nun etwa 25 Lichtquellen angesteuert werden. Diese Zahl ist zwar im Vergleich zu den 809 verbauten LEDs sehr gering, da die Environment-Maps jedoch ohnehin stark gefiltert sind, reicht diese Anzahl bereits vollkommen aus, um gute Beleuchtungsergebnisse zu erreichen.

7 Fazit

In dieser Arbeit wurde erfolgreich ein Gonioreflektometer gebaut, die Ansteuerungssoftware für Arduino und PC geschrieben, sowie ein Renderer für die aufgenommenen Materialien und Objekte implementiert.

Die aufgenommenen Materialien sowie deren gerenderte BRDFs können viele Lichteffekte aufnehmen bzw. darstellen, die mit herkömmlichen mathematischen Beleuchtungsmodellen nicht möglich sind. Der Realitätsgrad der virtuellen Materialien konnte somit stark gesteigert werden.

Die Bauart dieses Gonioreflektometers ist eine Neuheit, da bisherige Publikationen fast ausschließlich auf eine Zwei-Arm-Bauweise ohne Lichtkuppel setzen. Als besonders vorteilhaft an dieser Bauart des Gonioreflektometers hat sich vor allem die vielseitige Anwendbarkeit herausgestellt. So wurde gezeigt, dass ein solcher Aufbau nicht nur für das Messen von BRDFs, sondern auch für 3D-Scans, Lichtfeldaufnahmen und Light-Staging verwendet werden kann.

Ferner konnte gezeigt werden, dass mit handelsüblichen Materialien und einem Budget von unter 500€ ein sehr taugliches Gonioreflektometer gebaut werden kann. Im Vergleich zu den vier- bis fünfstelligen Preisen von kommerziellen Geräten ist dies ein großer Erfolg. Zwar ist die Präzision dieser Geräte in der Regel noch etwas höher und weniger Artefakt-behaftet, das Preis-Leistungs-Verhältnis des in dieser Arbeit gebauten Gerätes ist jedoch deutlich besser.

Als Fazit lässt sich also sagen, dass das in dieser Arbeit gebaute Gonioreflektometer und die dazu entwickelte Software ein gelungenes Gesamtpaket darstellen, mit dem vielseitige Anwendungsbereiche abgedeckt werden, ohne dabei besonders teuer zu sein.

8 Ausblick

Auch wenn schon viele Anwendungen mit dem in dieser Arbeit gebauten Gonioreflektometer möglich sind, so gibt es immer noch einige Aspekte, die verbessert werden könnten.

Eines der größten Probleme bei der Aufnahme von BRDFs ist die Verschattung. Zwar wurden in dieser Arbeit einige Algorithmen zur Kompensierung der Verschattungsartefakte implementiert und vorgestellt, perfekt sind diese Lösungen jedoch nicht. Dies liegt daran, dass die fehlenden Informationen in den verschatteten Bereichen nur „geraten" werden können. Eine Lösung dieses generellen Problems wäre die Verwendung eines halb-durchlässigen Spiegels oder auch „Beam-Splitters". Damit kann das Licht auch dann zur Materialprobe gelangen, wenn die Kamera sonst im Weg wäre. Hierfür müsste allerdings der Aufbau des Kameraarms grundlegend verändert werden.

Eine weitere Möglichkeit, um das Gerät vielseitiger anwendbar zu machen

wäre, die Aufnahmefläche bzw. -volumen zu vergrößern. Dazu müsste der gesamte Aufbau skaliert werden. Vorteilhaft wäre dabei besonders für die Lichtfeldaufnahmen, den 3D-Scan und das Light-Staging, dass auch größere Objekte digitalisiert bzw. aufgenommen werden könnten. Schwierigkeiten könnten hierbei allerdings bezüglich der Helligkeit der LEDs entstehen, da diese bereits nahe an ihrer maximalen Leuchtkraft liegen.

Interessant wäre auch, andere Arten von Kameras zu verwenden. Mit einer Multispektralkamera könnten beispielsweise noch detailliertere Materialaufnahmen gemacht werden. Diese könnten dann auch lichtbrechende bzw. -beugende Effekte wie z.B. sogenannte Strukturfarben aufnehmen. Auch Reflexionen im nicht-sichtbaren Bereich des Lichts könnten mit einer solchen Kamera aufgenommen werden. Mithilfe einer Wärmebildkamera könnte außerdem die Hitzeabstrahlcharakteristik von warmen Objekten vermessen werden.

Das in dieser Arbeit gebaute Gonioreflektometer kann lediglich das Reflexionsverhalten von Materialien messen, da es nur im oberen Halbraum der Materialprobe arbeitet. Man könnte den Aufbau jedoch so erweitern, dass der Kameraarm eine volle Umdrehung um die Materialprobe machen kann. Dafür müsste auch die Drehscheibe umgebaut werden. Letztendlich könnten dann auch die Transmissionseigenschaften von lichtdurchlässigen Materialien aufgenommen werden.

Insgesamt lässt sich sagen, dass bereits viele Funktionen in dieser Arbeit realisiert wurden. Die Möglichkeiten des in dieser Arbeit gebauten Gonioreflektometers sind jedoch noch lange nicht ausgeschöpft.

Literatur

[BEWW+08] Moshe Ben-Ezra, Jiaping Wang, Bennett Wilburn, Xiaoyang Li, and Le Ma. An led-only brdf measurement device. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[Bli77] James F Blinn. Models of light reflection for computer synthesized pictures. In *ACM SIGGRAPH computer graphics*, volume 11, pages 192–198. ACM, 1977.

[BNC09] Réjean Baribeau, William S Neil, and Éric Côté. Development of a robot-based gonioreflectometer for spectral brdf measurement. *Journal of Modern Optics*, 56(13):1497–1503, 2009.

[BPD+14] Laurent Belcour, Romain Pacanowski, Marion Delahaie, Aude Laville-Geay, and Laure Eupherte. Bidirectional reflectance distribution function measurements and analysis of retroreflective materials. *JOSA A*, 31(12):2561–2572, 2014.

[CT81] Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. In *ACM Siggraph Computer Graphics*, volume 15, pages 307–316. ACM, 1981.

[DAD+18] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Transactions on Graphics (TOG)*, 37(4):128, 2018.

[Erb80] W Erb. Computer-controlled gonioreflectometer for the measurement of spectral reflection characteristics. *Applied optics*, 19(22):3789–3794, 1980.

[Foo97] Sing Choong Foo. *A gonioreflectometer for measuring the bidirectional reflectance of material for use in illumination computation*. PhD thesis, 1997.

[HGH06] Dirk Hünerhoff, Ulrich Grusemann, and Andreas Höpe. New robot-based gonioreflectometer for measuring spectral diffuse reflection. *Metrologia*, 43(2):S11, 2006.

[Kaj86] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986.

[LFTW06] Hongsong Li, Sing-Choong Foo, Kenneth E Torrance, and Stephen H Westin. Automated three-axis gonioreflectometer for computer graphics applications. *Optical Engineering*, 45(4):043605, 2006.

[MSY09] Yasuhiro Mukaigawa, Kohei Sumino, and Yasushi Yagi. Rapid brdf measurement using an ellipsoidal mirror and a projector. *IPSJ Transactions on Computer Vision and Applications*, 1:21–32, 2009.

[NMI04] Saulius Nevas, Farshid Manoocheri, and Erkki Ikonen. Gonioreflectometer for measuring spectral diffuse reflectance. *Applied Optics*, 43(35):6391–6399, 2004.

[PHWF01] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581. ACM, 2001.

[Sch] Schrittmotor. Wikipedia. `https://de.wikipedia.org/wiki/Schrittmotor`. Accessed: 2019-05-28.

[Tel04] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[WS2] WS2812B. Datasheet. `https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf`. Accessed: 2019-05-28.

[WSJB+98] David White, Peter Saunders, Stuart J. Bonsey, John van de Ven, and Hamish Edgar. Reflectometer for measuring the bidirectional reflectance of rough surfaces. *Applied optics*, 37:3450–4, 07 1998.

[WZ15] Andreas W Winkler and Bernhard G Zagar. Building a gonioreflectometer-a geometrical evaluation. In *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, pages 1900–1905. IEEE, 2015.