

---

Volume ray casting — basics & principles

**MTE7101**

**Projektarbeit 1**

Studiengang: Informatik

Autor: Sven Osterwalder<sup>1</sup>

Betreuer: Prof. Claude Fuhrer<sup>2</sup>

Datum: 14.02.2016

Version: 1.1



Licensed under the Creative Commons Attribution-ShareAlike 3.0 License

<sup>1</sup>sven.osterwalder@students.bfh.ch

<sup>2</sup>claude.fuhrer@bfh.ch



# Versionen

Version	Datum	Autor(en)	Änderungen
0.1	25.09.2015	SO	Initiale Erstellung des Dokumentes
0.2	27.09.2015	SO	Entwickeln einer initialen Struktur, Aufbau der Dokumentation, Entwicklung Kapitel 4, Beschreibung von Ray Tracing in Kapitel 5, Hinzufügen des Kapitels 2, Einführen von TODO-Notizen
0.3	29.09.2015	SO	Einführen von Meeting Minutes A, Erweitern des Kapitels 5 um Beleuchtungsmodelle, Beschreiben von lokalen Beleuchtungsmodellen
0.4	04.10.2015	SO	Hinzufügen von Standards und Richtlinien 4.3.2, Erweitern des Kapitels 5 um globale Beleuchtungsmodelle 5.1.2 sowie Ray Casting 5.2.1, Entfernen der Schriftart ‘cm-bright’, Hinzufügen des Kapitels über (implizite) Oberflächen 6.1
0.5	11.10.2015	SO	Neuordnung des Kapitels 5: Hinzufügen des Abschnitt 5.2 über Ray Tracing sowie Abschnitt 6.2 über Darstellung von impliziten Oberflächen.
0.6	14.10.2015	SO	Hinzufügen von TODO-Notizen, Anpassung der Textdarstellung in Formeln
0.7	16.10.2015	SO	Hinzufügen einer Illustration des Phong-Beleuchtungmodells in Kapitel 5.1.1, Abarbeiten von TODO-Notizen in diversen Kapiteln, Erweitern des Kapitels über (implizite) Oberflächen 6.1
0.8	17.10.2015	SO	Hinzufügen einer Illustration des Ray Tracing Algorithmus in Kapitel 5.1.2
0.9	19.10.2015	SO	Beginn Kapitel über Rendering von impliziten Oberflächen 6.3
0.10	21.10.2015	SO	Erweiterung Kapitel über Rendering von impliziten Oberflächen, Einführung des Kapitel 7 über die Umsetzung eines Prototypen
0.11	24.10.2015	SO	Umstrukturierung des Dokumentes, Anpassung der Dokumentvorlage, Nachführen der Versionshistorie, Ausführen der Meeting Minutes vom 18.10.2015, Erstellen der Vorlage für Meeting Minutes vom 25.10.2015
0.12	31.10.2015	SO	Nachführen der Versionshistorie, Ausführen der Meeting Minutes vom 25.10.2015, Erstellen der Vorlage für Meeting Minutes vom 02.11.2015, Erweiterung des Kapitels 7, Abarbeiten von TODO-Notizen
0.13	07.11.2015	SO	Ausführen der Meeting Minutes vom 02.11.2015, Erweiterung des Kapitels 7
0.14	15.11.2015	SO	Erarbeiten des Kapitels 6.3.3, Erweiterung des Kapitels 7 um weiche Schatten, Erstellen und Hinzufügen von Bildmaterial zu Kapiteln 6.2.1 und 6.2.2.
0.15	29.11.2015	SO	Komplette Überarbeitung des Bildmaterials zu Kapiteln 6.2.1 und 6.2.2.
0.16	06.12.2015	SO	Nachführen von Meeting Minutes und der Versionierung. Überarbeiten des Zeitplanes, weiter Überarbeitung des Bildmaterials. Hinzufügen des Kapitels 5.3, Hinzufügen einer Einleitung zu Kapitel 5.

0.17	17.12.2015	SO	Einführen des Kapitels über Modelle zur Schattierung 5.3, Fertigstellung der Beschreibung von Gouraud-Shading 5.3.2.
0.18	22.12.2015	SO	Erweiterung des Kapitels über Modelle zur Schattierung 5.3 um diverse Illustrationen.
0.19	28.12.2015	SO	Erweiterung des Kapitels über Modelle zur Schattierung 5.3 um diverse Illustrationen, Hinzufügen eines (bildlichen) Vergleiches der verschiedenen Modelle zur Schattierung, Abarbeiten von TODO-Notizen, Hinzufügen eines Beispieles zur parametrischen Darstellung einer Kugel in Kapitel 6.1, Korrektur diverser Rechtschreibfehler in Kapitel 6.2.
0.20	03.01.2016	SO	Erweiterung des Kapitels 5.1.2 um Ray Casting und Ray Tracing Verfahren, Erstellen und Hinzufügen diverser Illustrationen zu Kapitel 5.1, Erweiterung des Kapitels 6.1 um Abschnitt 6.1.3 über Distanzfelder, Überarbeitung der L <sup>A</sup> T <sub>E</sub> X-Vorlage.
0.21	04.01.2016	SO	Aufteilen des Kapitels Beleuchtungsmodelle in Unterkapitel, Erweiterung der Kapitel 5.2.1 Ray Casting und 5.2.2 Ray Tracing.
0.22	05.01.2016	SO	Fertigstellung des Kapitels 5.2.2 Ray Tracing.
0.23	07.01.2016	SO	Erstellen und Hinzufügen von diversen Illustrationen zu Kapitel 5.2.2 Ray Tracing, Hinzufügen eines Abschnittes über Shader in Kapitel 5.3 Modelle zur Schattierung (shading models), Korrektur des Beispiels zur parametrischen Darstellung einer Kugel in Kapitel 6.1, Abarbeiten von TODO-Notizen, Hinzufügen einer Einleitung zu Kapitel 6.3 Rendering von impliziten Oberflächen, Hinzufügen von Kommentaren zum Quellcode des Prototyps.
0.24	09.01.2016	SO	Einleitung der Projektarbeit 1 überarbeitet, Korrektur der Beschreibung und Referenz von impliziten Oberflächen in Kapitel 6.1.3, Erarbeiten des Fazits 8, Beschreibung der Aufgabenstellung 3.
0.25	10.01.2016	SO	Fertigstellung der Kapitel 3 Aufgabenstellung und Kapitel 4 Vorgehen, Abarbeiten und Nachführen von verbleibenden TODO-Notizen, Nachführen der Versionshistorie.
0.90	11.01.2016	SO	Überarbeitung der Dokumentation bis und mit Kapitel 5, Nachführen der Versionshistorie.
0.91	13.01.2016	SO	Überarbeitung der Dokumentation ab Kapitel 5, Verfassen von Abschnitt 8.1 Erweiterungsmöglichkeiten, Nachführen der Versionshistorie.
1.0	13.01.2016	SO	Überarbeitung von Kapitel 8 Schlusswort, Verfassen von Kapitel Abstract, Nachführen der Versionshistorie.
1.1	14.02.2016	SO	Überarbeitung des gesamten Dokumentes hinsichtlich Rechtschreibung sowie Formulierung. Nachführen der Meeting Minutes vom 25.01.2016. Einarbeiten der besprochenen Punkte. Nachführen der Versionshistorie.

# Abstract

Das Fachgebiet “Computergrafik” war schon immer bestrebt eine möglichst realitätsnahe Darstellung von Szenen und Modellen zu erzeugen. Eine Darstellung, welche möglichst nahe an der Realität respektive der menschlichen Wahrnehmung liegt.

Im Laufe der Zeit entstanden verschiedene Ansätze um eine solche Darstellung zu erreichen. Ein Teilgebiet davon sind Beleuchtungsmodelle, welche die Beleuchtung einer Darstellung bzw. einer Szene berechnen.

Ein relativ realistisch wirkendes Beleuchtungsmodell ist Ray Tracing (zu Deutsch Strahlen-Verfolgung), welches auf den physikalischen Grundlagen von Licht und Materialien von Oberflächen basiert.

Diese Verfahren waren lange Zeit zu langsam um damit eine Darstellung zu erreichen. Durch die Weiterentwicklung der Computer, vor allem der Grafikkarten (GPUs), ist Ray Tracing jedoch für die Darstellung von Szenen in Echtzeit wieder interessant geworden.

Diese Projektarbeit stellt die (theoretischen) Grundlagen zur Erzeugung und Darstellung von Szenen und Modellen sowie ein spezielles Ray Tracing Verfahren zur Darstellung von Bildern in Echtzeit vor. Dabei handelt es sich um *Volume Ray Casting* bzw. *Sphere Tracing*.

Um die Hypothese zu stützen, dass die komplexen, realistischer wirkenden Verfahren wieder in den Fokus der Darstellung von Szenen in Echtzeit gerückt sind, wurde ein Prototyp entwickelt, der Sphere Tracing in Echtzeit auf der Grafikkarte (GPU) umsetzt.

Mit dem vorgestellten Verfahren gelingt mittels moderner Grafikkarten eine Darstellung von Szenen und Modellen in Echtzeit.

# Inhaltsverzeichnis

<b>Abstract</b>	<b>v</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Administratives</b>	<b>2</b>
2.1. Beteiligte Personen . . . . .	2
2.2. Aufbau des Dokumentes . . . . .	2
2.3. Ergebnisse (Deliverables) . . . . .	2
<b>3. Aufgabenstellung</b>	<b>3</b>
3.1. Motivation . . . . .	3
3.2. Ziele und Abgrenzung . . . . .	3
<b>4. Vorgehen</b>	<b>5</b>
4.1. Arbeitsorganisation . . . . .	5
4.2. Projektphasen . . . . .	5
4.3. Technologien . . . . .	7
<b>5. Theoretischer Hintergrund</b>	<b>9</b>
5.1. Beleuchtungsmodelle . . . . .	9
5.2. Ray Casting und Ray Tracing . . . . .	12
5.3. Modelle zur Schattierung (shading models) . . . . .	24
<b>6. Oberflächen</b>	<b>28</b>
6.1. Oberflächen . . . . .	28
6.2. Darstellung von impliziten Oberflächen . . . . .	32
6.3. Rendering von impliziten Oberflächen . . . . .	42
<b>7. Prototyp</b>	<b>48</b>
7.1. Architektur . . . . .	48
7.2. Umsetzung . . . . .	50
<b>8. Schlusswort</b>	<b>59</b>
8.1. Erweiterungsmöglichkeiten . . . . .	59
<b>Glossar</b>	<b>61</b>
<b>Literaturverzeichnis</b>	<b>61</b>
<b>Abbildungsverzeichnis</b>	<b>61</b>
<b>Tabellenverzeichnis</b>	<b>61</b>
<b>Auflistungsverzeichnis</b>	<b>62</b>
<b>Anhang</b>	<b>64</b>
<b>A. Meeting minutes</b>	<b>65</b>

# 1. Einleitung

Seit Bestehen moderner Computer existiert auch die Computergrafik. Eines ihrer Ziele ist die Projektion des dreidimensionalen Raumes auf eine zweidimensionale Fläche. In der Regel ist die Ausgabe auf einen zweidimensionalen Raum beschränkt.

Bei der Ausgabe wird zwischen statischen und dynamischen Bildern unterschieden. Während statische Bilder konstant sind, können sich dynamische Bilder ständig ändern. Aufgrund der Reaktionszeiten des menschlichen Auges müssen für eine ruckfreie Darstellung mindestens 25 Bilder pro Sekunde berechnet werden. Das Fachgebiet „Computergrafik“ war schon immer bestrebt eine möglichst realitätsnahe Darstellung zu erhalten. Eine Darstellung, die möglichst nahe an der menschlichen Wahrnehmung liegt.

Im Laufe der Zeit entstanden verschiedene Ansätze um eine naturgetreue Darstellung zu erreichen. Einer der Ansätze sind Beleuchtungsmodelle, welche die Beleuchtung einer Darstellung bzw. einer Szene berechnen. Dabei wird zwischen lokalen und globalen Beleuchtungsmodellen unterschieden.

Ein globales Beleuchtungsmodell ist Ray Tracing (zu Deutsch etwa „Strahlenverfolgung“), welches seinen Ursprung in der von Appel 1968 vorgestellten Arbeit „Some Techniques for Shading Machine Renderings of Solids“ hat [1]. 1980 wurde das Verfahren von Whitted in „An Improved Illumination Model for Shaded Display“ verbessert [2].

Das Verfahren besticht durch Einfachheit und bietet eine hohe Bildqualität mit perfekten Spiegelungen und Transparenzen. Mit entsprechenden Optimierungen ist das Verfahren schneller geworden, aber nicht ausreichend.

Mit „schnell“ ist die Zeitspanne für die Darstellung eines einzelnen Bildes gemeint. Für eine Darstellung in Echtzeit ist auch das verbesserte Verfahren noch zu langsam.

Dank Weiterentwicklung der Computer, vor allem der Grafikkarten (GPUs), ist Ray Tracing wieder in den Fokus der Darstellung von Szenen in Echtzeit gerückt.

Diese Projektarbeit stellt ein spezielles Ray Tracing Verfahren zur Darstellung von Bildern in Echtzeit vor: *Volume Ray Casting* bzw. *Sphere Tracing* genannt.

## 2. Administratives

Einige administrative Aspekte der Projektarbeit werden angesprochen, obwohl sie für das Verständnis der Resultate nicht notwendig sind.

Im gesamten Dokument wird nur die männliche Form verwendet, womit aber beide Geschlechter gemeint sind.

### 2.1. Beteiligte Personen

Autor      Sven Osterwalder<sup>1</sup>  
Betreuer    Prof. Claude Fuhrer<sup>2</sup>

*Begleitet den Studenten bei der Projektarbeit*

### 2.2. Aufbau des Dokumentes

Die vorliegende Arbeit ist aufgebaut wie folgt:

- Einleitung zur Projektarbeit
- Beschreibung der Aufgabenstellung
- Vorgehen des Autors im Hinblick auf die gestellten Aufgaben
- Lösung der gestellten Aufgaben
- Verwendete Technologien

Der Schwerpunkt dieser Arbeit liegt in der Beschreibung der theoretischen Grundlagen (unter praktischen Aspekten) des Volume Ray Casting Verfahrens.

### 2.3. Ergebnisse (Deliverables)

Nachfolgend sind die abzugebenden Objekte aufgeführt:

- **Abschlussdokument**

Das Abschlussdokument beinhaltet die theoretischen Grundlagen (unter praktischen Aspekten) des Volume Ray Casting Verfahrens.

- **Prototyp**

Der Prototyp entstand im Rahmen der Bearbeitung der Thematik. Es handelt sich um eine praktische Umsetzung des vorgestellten Verfahrens.

---

<sup>1</sup>sven.osterwalder@students.bfh.ch

<sup>2</sup>claude.fuhrer@bfh.ch

# 3. Aufgabenstellung

## 3.1. Motivation

Seit der Entstehung der Computergrafik Ende der 1960er Jahre möchte man möglichst realitätsnahe, echte Bilder generieren können. Angefangen mit *Photon Tracing* (1968), über *Ray Tracing* (1980), bis hin zu *Photon Mapping* (1997/2001), wurden immer realistischer wirkende Verfahren zur Darstellung von Bildern entwickelt.

Da der Aufwand möglichst realistisch wirkende Bilder zu erzeugen jedoch relativ hoch ist, wurden diese Verfahren in der Regel nur für statische Bilder verwendet. Und auch dort war und ist der Grad der Details begrenzt bzw. an Rechenzeit gebunden. Möchte man mehr Details, so wird mehr Rechenzeit benötigt. Daher wurde oder wird für eine Darstellung in Echtzeit — also eine Anzeige von mindestens 25 Bildern pro Sekunde — lange nur eine Näherung, basierend auf vereinfachten *Beleuchtungsmodellen* genutzt.

Durch die Weiterentwicklung der Computer und vor allem durch die Weiterentwicklung der Grafikkarten (GPUs), sind die komplexeren, realistischer wirkenden Verfahren, wie z.B. Ray Tracing, jedoch wieder in den Fokus der Darstellung von Szenen in Echtzeit gerückt.

Diese Projektarbeit beschreibt theoretische Grundlagen zur Erzeugung von möglichst realistisch wirkenden Bildern anhand *Beleuchtungsmodellen*, *praktischen Verfahren* (wie z.B. Ray Marching oder Ray Tracing), *Modellen zur Schattierung* und stellt schliesslich eine effiziente Methode des Ray Tracings, das so genannte *Sphere Tracing* vor.

Um die Hypothese zu stützen, dass die komplexen, realistischer wirkenden Verfahren wieder in den Fokus der Darstellung von Szenen in Echtzeit gerückt sind, wurde ein Prototyp entwickelt, der Sphere Tracing in Echtzeit auf der Grafikkarte (GPU) umsetzt.

### 3.1.1. Demoszene

In den 1980er-Jahren entwickelte sich “unter Anhängern der Computerszene … während der Blütezeit der 8-Bit-Systeme” [3] eine Bewegung namens Demoszene. “Ihre Mitglieder, die häufig Demoszener oder einfach Szener genannt werden, erzeugen mit Computerprogrammen auf Rechnern so genannte Demos – Digitale Kunst, meist in Form von musikalisch unterlegten Echtzeit-Animationen.” [3]

Es handelt sich bei der Demoszene um ein sehr aktives und kreatives Umfeld, in welchem die Technologie ständig an die Grenzen ihrer Möglichkeiten gebracht wird. In diesem Umfeld entstehen regelmässig neue Ideen zur Erzeugung von noch realistischer wirkenden Bildern. Dabei findet eine wechselseitige Beeinflussung zwischen dem akademischen Umfeld und der Demoszene statt.

So wurde auch das hier vorgestellte *Sphere Tracing* Verfahren relativ früh von in der Demoszene aktiven Personen aufgegriffen und behandelt.

## 3.2. Ziele und Abgrenzung

Diese Projektarbeit besteht aus drei Teilen. Der Beschreibung der theoretischen Grundlagen, der Beschreibung des als Sphere Tracing bekannten Verfahrens sowie der Umsetzung eines Prototypen.

In dieser Projektarbeit liegt der Fokus vor allem auf der (mathematischen) Beschreibung und Anwendung von Oberflächen, sowie der Darstellung von Bildern mittels Sphere Tracing.

Bei dem entwickelten Prototypen handelt es sich um eine Art Machbarkeitsstudie. Diese zeigt, dass Bilder mittels Sphere Tracing in Echtzeit erzeugt und dargestellt werden können.

Diese Projektarbeit dient als Vorarbeit und Grundlage für das MSE-Modul *MTE7102* — “Vertiefungsprojekt 2” (Folgemodul).

### **3.2.1. Vorgängige Aktivitäten**

Die Grundlagen dieser Projektarbeit waren dem Autor durch sein vorgängiges Studium an der Berner Fachhochschule mit Schwerpunkt *Computer Perception and Virtual Reality* bereits bekannt, jedoch nur partiell.

Durch sein Interesse für die Demoszene wurde der Autor auf die Thematik dieser Arbeit aufmerksam. Zuvor hatte er sich nie eingehend damit beschäftigt.

### **3.2.2. Neue Lerninhalte**

Zusätzlich zu den formalen Lerninhalten hatte die Arbeit für den Autor die folgenden neuen Lerninhalte:

- Implizite Oberflächen
- Distanzfunktionen
- Distanzfelder
- Ray Marching
- Sphere Tracing
- Darstellung von (weichen) Schatten
- Praktische Umsetzung der vorgestellten Verfahren

# 4. Vorgehen

## 4.1. Arbeitsorganisation

### 4.1.1. Regelmässige Treffen

Regelmässige Besprechungen mit dem Betreuer der Arbeit halfen, die gesteckten Ziele zu erreichen und Fehlentwicklungen zu vermeiden. Der Betreuer unterstützte den Autor dabei mit Vorschlägen. Die Treffen fanden zumeist alle zwei Wochen statt. Sie wurden in Form eines Protokolles festgehalten. Das Protokoll findet sich in Anhang A.

## 4.2. Projektphasen

### 4.2.1. Meilensteine

Um bei der Arbeit ein möglichst strukturiertes Vorgehen einzuhalten, wurden folgende Projektphasen gewählt:

- Start der Projektarbeit
- Erarbeitung und Festhalten der Anforderungen
- Erarbeitung der theoretischen Grundlagen
- Erstellung der abschliessenden Dokumentation
- Erstellung eines Prototypen

Die Phasen *Erarbeitung der theoretischen Grundlagen*, *Erstellung der abschliessenden Dokumentation* sowie *Erstellung eines Prototypen* liefen parallel ab. Erkenntnisse einer Phase flossen jeweils in die anderen Phasen ein.

#### 4.2.2. Zeitplan / Projektphasen

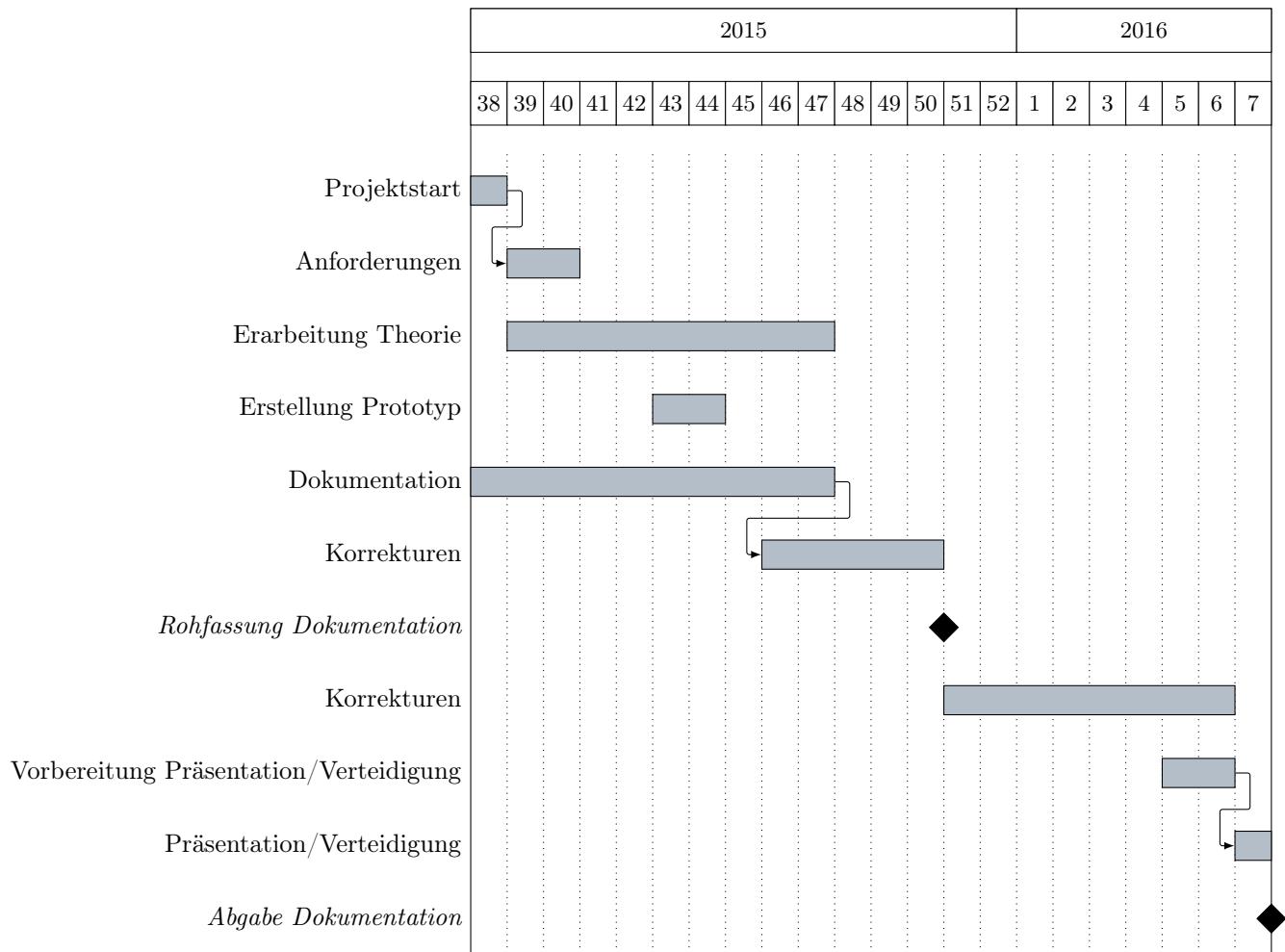


Abbildung 4.1.: Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Kalenderwochen dar

##### **Projektstart**

In der ersten Phase wurden die Zwischenziele der Arbeit identifiziert und skizziert. Um Einzelheiten der Aufgabe zu verstehen, wurde das notwendige Vorwissen über Algorithmen für globalen Beleuchtung erarbeitet. Weiter wurde die Grundlage dieser Dokumentation erstellt.

##### **Anforderungen**

In dieser Phase wurde das Ziel dieser Projektarbeit festgelegt. Ausgehend vom Ziel wurden die dazu erforderlichen Projektpasen festgelegt.

##### **Erarbeitung theoretische Grundlagen**

Diese Phase bedeutete Publikationen zu lesen und durch zuarbeiten, was immer zu vielen neuen Denkanstößen weiteren Recherchen führte. Die Erkenntnisse dieser Phase flossen stetig in die Dokumentation ein, ergänzten und beeinflussten diese.

## Dokumentation

Die vorliegende Arbeit entspricht der Dokumentation. Sie wurde während der gesamten Projektarbeit stetig erweitert und diente zur Reflexion von fertiggestellten Teilen.

## 4.3. Technologien

### 4.3.1. Tools und Software

#### Dokumentation und Prototyp

**L<sup>A</sup>T<sub>E</sub>X** Eine Makro-Sammlung für das T<sub>E</sub>X-System. Wurde zur Erstellung dieser Dokumentation eingesetzt. Diese Dokumentation wurde mittels L<sup>A</sup>T<sub>E</sub>X geschrieben.

**Make** Build-Automations-Werkzeug, wurde zur Erstellung dieses Dokumentes eingesetzt.

**CMake** Build-Automations-Werkzeug, wurde zur Erstellung des Prototypen eingesetzt.

**zotero** Ein freies, quelloffenes Literaturverwaltungsprogramm zum Sammeln, Verwalten und Zitieren unterschiedlicher Online- und Offline-Quellen [4].

**VIM** Vi IMproved. Ein freier, quelloffener Texteditor zur Textbearbeitung. Er wurde zum Verfassen der Dokumentation sowie zur Entwicklung des Prototypen eingesetzt.

**LLVM** Low Level Virtual Machine. Eine Compiler-Architektur zum Kompilieren von Applikationen. Wurde zur Kompilation des Prototypen eingesetzt.

**clang** Ein C-Sprachen-Frontend für LLVM. Wurde zur Kompilation des Prototypen eingesetzt.

#### Arbeitsorganisation

**Git** Freie Software zur verteilten Versionsverwaltung, wurde für die Entwicklung dieser Dokumentation sowie des Prototypen verwendet. Die Projektarbeit findet sich unter GitHub<sup>1</sup>.

**GitHub** Eine freie Hosting-Plattform für Git mit Weboberfläche.

### 4.3.2. Standards und Richtlinien

#### Programmcode

Der Programmcode des Prototypen, welcher in C++ geschrieben wurde, folgt den offiziellen Richtlinien für C++ von Google<sup>2</sup>.

#### Pseudocode

Da der Autor, bedingt durch seine tägliche Arbeit, in der Programmiersprache *Python* relativ bewandert ist, wird diese als Sprache zur Beschreibung von Pseudocode verwendet. Dabei wird kein Augenmerk auf die formale Korrektheit, geschweige denn auf die Lauffähigkeit des Pseudocodes gelegt.

---

<sup>1</sup> <https://www.github.com/sosterwalder/mte7101-project1>

<sup>2</sup> <https://google.github.io/styleguide/cppguide.html>

## Projekt-Struktur

Um die Übersicht zu wahren und den Verwaltungsaufwand minimal zu halten, wurde eine entsprechende Projekt-Struktur gewählt. Diese ist in Auflistung 4.1 ersichtlich.

Projekt-Struktur

```
23      bin/    -- Compiled, binary file of the prototype
24      build/   -- Temporary directory for build output
25      data/    -- GLSL shader files
26      doc/    -- Documentation
27          img/    -- Images used for the documentation
28          inc/    -- LaTeX files used for inclusion to maintain
29              readability and managabilty
30          static/   -- Static files used for inclusion, e.g. the
31              bibliography, versioning of the document
32              and so on
33          attachment/ -- Attachments for the documentation,
34              e.g. the minutes of the held meetings
35      inc/    -- External source files for inclusion, needed by the
36          prototype
37      lib/    -- External libraries for linking against when building
38          the prototype
39      src/    -- Source code of the prototype
```

Auflistung 4.1: Projekt-Struktur.

# 5. Theoretischer Hintergrund

Sofern im Text nicht anders vermerkt, basiert das folgende Kapitel auf [5, S. 721ff].

Für die Erzeugung und Darstellung von Bildern werden zwei Angaben benötigt: Was dargestellt werden soll und wie dieses dargestellt werden soll. Prinzipiell geht es darum zu bestimmen, welche Farbe eine Oberfläche an einem bestimmten Punkt hat. Dabei haben sich die Begriffe des *Beleuchtungsmodells* (*illumination model*) und des *Modelles zur Schattierung* (*shading model*) etabliert.

Foley nutzt den Begriff *shading model* als Überbegriff [5, S. 721]. Dieser schliesst auch Beleuchtungsmodelle ein. Ein *shading model* definiert, wann und mit welchen Parametern ein Beleuchtungsmodell angewendet wird. So nutzen manche *shading models* ein Beleuchtungsmodell für jeden Pixel, andere wiederum nur für einzelne Pixel und interpolieren dabei die Werte der anderen Pixel.

## 5.1. Beleuchtungsmodelle

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [2, S. 343], sowie auf [6].

Beleuchtungsmodelle beschreiben, wieviel Licht von einem sichtbaren Punkt einer Oberfläche zum Betrachter emittiert wird. In der Regel wird das Licht als Funktion in Abhängigkeit folgender Faktoren beschrieben:

- Richtung der Lichtquelle
- Lichtstärke
- Position des Betrachters
- Orientierung der Oberfläche
- Oberflächenbeschaffenheit
- Globale Umgebung

Dabei wird zwischen lokalen und globalen Beleuchtungsmodellen unterschieden.

### 5.1.1. Lokale Beleuchtungsmodelle

Lokale Beleuchtungsmodelle aggregieren Daten lokal, also von benachbarten Oberflächen. Diese Modelle sind in ihrem Umfang allerdings limitiert, da sie normalerweise nur Lichtquellen sowie die Orientierung einer Oberfläche einbeziehen. Sie ignorieren dabei aber die globale Umgebung, in welcher sich eine Oberfläche befindet. Die traditionell verwendeten Algorithmen zur Berechnung der Sichtbarkeit von Oberflächen verfügen über keine globalen Daten.

Das bekannteste lokale Beleuchtungsmodell ist das *Phong-Beleuchtungsmodell*, welches 1975 von Phong Bui-Tong entwickelt wurde[7]. Es beschreibt nicht-perfekte Reflektoren wie sie zum Beispiel ein Apfel darstellt [5, Kapitel 16, Seite 729].

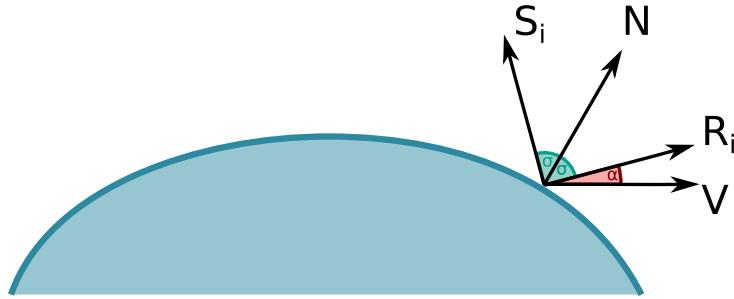


Abbildung 5.1.: Illustration des Phong-Beleuchtungsmodells<sup>1</sup>

Das *Phong-Beleuchtungsmodell* beschreibt die reflektierte (Licht-) Intensität  $I$  als Zusammensetzung aus der ambienten, der diffusen und der ideal spiegelnden Reflexion einer Oberfläche:

$$I = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} + I_{\text{emissive}} \quad (5.1)$$

Oder mathematisch ausgedrückt:

$$I(\vec{V}) = k_a \cdot L_a + k_d \sum_{i=0}^{n-1} L_i \cdot (\vec{S}_i \cdot \vec{N}) + k_s \sum_{i=0}^{n-1} L_i \cdot (\vec{R}_i \cdot \vec{V})^{k_e} \quad (5.2)$$

In der obigen Gleichung 5.2 wurde der emissive Term  $I_{\text{emissive}}$  bewusst weggelassen, da er meistens für Spezialeffekte statt für die Beleuchtung "normaler" Objekte benutzt wird [6].

Wobei gilt:

- $I(\vec{V})$   
Die reflektierte (Licht-) Intensität in Richtung des Vektors  $\vec{V}$ .
- $n$   
Anzahl Lichtquellen.
- $k_a \cdot L_a$   
Ambiente Komponente des Beleuchtungsmodells. Mittels diesem Faktor wird versucht allem indirekten Licht der Szene gerecht zu werden. Bei  $k_a$  handelt es sich um eine Konstante, welche den ambienten Anteils des Lichtes  $L_a$  skaliert.
- $k_d$   
Konstante für die diffuse Komponente des reflektierten Lichtes, basierend auf der Wellenlänge bzw. der Frequenz.
- $\vec{S}_i$   
Richtung, in welcher das Licht der  $i$ -ten Lichtquelle ankommt, normalisierter Einheitsvektor.
- $\vec{N}$   
Einheitsnormale der Oberfläche.
- $k_s$   
Koeffizient der spiegelnden Komponente, basierend auf der Wellenlänge bzw. Frequenz.
- $\vec{R}_i = \vec{S}_i + 2(\vec{S}_i \cdot \vec{N})\vec{N}$   
Richtung, in welche das Licht der  $i$ -ten Lichtquelle reflektiert wird, normalisierter Einheitsvektor.
- $\vec{V}$   
Blickrichtung des Betrachters bzw. der Kamera.

---

<sup>1</sup>Eigene Darstellung mittels Geogebra, angelehnt an [5][Kapitel 16, Seite 731, Abbildung 16.12]

- $k_e$   
Exponent, welcher von der Rauheit bzw. Reflexion der Oberfläche abhängt.
- $L_i$   
Licht- bzw. Farbenintensität der  $i$ -ten Lichtquelle.

Der reflektive Vektor  $\vec{R}_i$  ist gegeben durch

$$\vec{R}_i = \vec{S}_i + 2(\vec{S}_i \cdot \vec{N})\vec{N} \quad (5.3)$$

Damit die Energieerhaltung gewährleistet ist, muss weiter  $k_d + k_s < 1$  gelten. Der Winkel zwischen  $\vec{R}$  und  $\vec{V}$  wird mittels  $\cos(\alpha)$  ermittelt.

### 5.1.2. Globale Beleuchtungsmodelle

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [5, S. 775ff].

Globale Beleuchtungsmodelle beschreiben die (Licht-) Intensität eines Punktes. Die Intensität setzt sich aus direkter und indirekter Einstrahlung des Lichtes zusammen.

Direktes Licht kommt unbeeinflusst aus einer Lichtquelle direkt zum Betrachter bzw. zu einem Bildpunkt. Indirektes Licht entsteht durch Diffusion, Reflexion und Transmission, je nach Beschaffenheit der Körper sowie deren Oberfläche.

Bei globalen Beleuchtungsmodellen unterscheidet man zwischen Blickwinkel abhängigen, z.B. Ray Tracing, und zwischen Blickwinkel unabhängigen Algorithmen, z.B. Photon Mapping.

Blickwinkel abhängige Algorithmen verwenden eine Diskretisierung der sichtbaren Fläche bzw. Bildfläche (Zerlegung in kleine Abschnitte, also Bildpunkte). Dadurch kann entscheiden werden, von welchen Punkten, in Blickrichtung des Betrachters gesehen, die Berechnung der Beleuchtung durchgeführt werden soll.

Blickwinkel unabhängige Algorithmen hingegen diskretisieren die Umgebung. Sie stellen so genügend Informationen zur Verfügung, um die Berechnung der Beleuchtung an einem beliebigen Punkt unabhängig von der Blickrichtung des Betrachters zu berechnen.

Beide Arten von Algorithmen haben Vor- und Nachteile. Blickwinkel abhängige Algorithmen eignen sich gut für die Berechnung von Spiegelungen, welche auf der Blickrichtung des Betrachters basieren. Sie eignen sich aber weniger um gleichbleibende diffuse Anteile über weite Flächen eines Bildes zu berechnen. Bei Blickwinkel abhängigen Algorithmen ist genau das Gegenteil der Fall.

### Renderinggleichung

Die in Unterabschnitt 5.1.2 genannten Verfahren berechnen, wie sich Licht von einem Punkt  $A$  zu einem anderen Punkt  $B$  im Raum bewegt. Dabei beschreiben sie die Intensität des Lichtes von  $A$  nach  $B$ . Zusätzlich werden alle Intensitäten von beliebigen Punkten im Raum, die den Punkt  $A$  erreichen und zu Punkt  $B$  reflektiert werden, mit einberechnet.

James Kajiya stellte 1986 die so genannte Renderinggleichung auf, welche dieses Verhalten beschreibt [8, 5]:

$$I(x, x') = g(x, x')[\varepsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx''] \quad (5.4)$$

Wobei gilt:

- $x, x'$  und  $x''$   
Punkte im Raum.

- $I(x, x')$   
Intensität des Lichtes, die von Punkt  $x'$  zu Punkt  $x$  gelangt.
- $g(x, x')$   
Ein auf die Geometrie bezogener Term  
0:  $x$  und  $x'$  verdecken sich.  
 $1/r^2$ :  $x$  und  $x'$  sehen sich, wobei  $r$  die Distanz zwischen  $x$  und  $x'$  ist.
- $\varepsilon(x, x')$   
Intensität des Lichtes, welches von  $x'$  nach  $x$  emittiert wird.
- $\rho(x, x', x'')$   
Intensität des Lichtes, welches von  $x''$  durch die Oberfläche bei  $x'$  nach  $x$  gestreut wird.
- $\int_S$   
Integral über die Vereinigung aller Flächen, daher  $S = \bigcup S_i$ .  
Dies bedeutet, dass die Punkte  $x$ ,  $x'$  und  $x''$  über alle Flächen aller Objekte der Szene "streifen". Wobei es sich bei  $S_0$  um eine zusätzliche Fläche handelt, welche als Hintergrund verwendet wird.  $S_0$  ist dabei eine Hemisphäre, welche die gesamte Szene umspannt.

## 5.2. Ray Casting und Ray Tracing

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [6, Kapitel 15, S. 387ff].

Für die realistische Darstellung eines Bildes, muss berechnet werden, wieviel Licht ausgehend von einer Lichtquelle zu jedem Pixel der sichtbaren Bildfläche (also dem Betrachter) transportiert wird. Da Photonen die Energie des Lichtes ( $E_{\text{photon}} = h\nu$  wobei  $h = \frac{E}{f}$  und  $\nu$  die [Licht-] Frequenz respektive die Wellenlänge ist) transportieren, möchte man das physikalische Verhalten dieser simulieren.

Es ist allerdings nicht möglich *alle* Photonen einer Szene oder eines Bildes zu simulieren, da der Aufwand dafür schlicht zu gross wäre. Daher ist es sinnvoll, exemplarisch nur einige Photonen zu betrachten und danach eine Abschätzung des gesamten Lichtes vorzunehmen.

Jede Lichtquelle emittiert Photonen (als Welle und als Teilchen) in alle Richtungen. Man modelliert diese als Partikel, welche anhand Lichtstrahlen (*light rays*) auf Objekte einer Szene treffen.

Jedes Photon hat dabei eine spezifische Wellenlänge  $\lambda$ , welche die Farbe definiert, sowie eine Energie respektive Frequenz  $f$ , welche die Intensität der Farbe des Photons definiert.

Trifft ein Photon auf ein Objekt, wird ein Teil der Energie absorbiert, ein Teil reflektiert und ein Teil durchdringt das Objekt (Transmission). Photonen treffen solange auf Objekte, bis ihre gesamte Energie absorbiert ist, bis sie die Szene verlassen oder sie auf die sichtbare Bildfläche treffen und somit zum eigentlichen Bild beitragen.

Bei *Ray Casting* bzw. *Ray Tracing* handelt es sich um zwei relativ einfache Verfahrens-Strategie um globale Beleuchtungsmodelle zu implementieren.

Die einfachste Art das oben genannte Verhalten von Licht zu modellieren ist das so genannte *Ray Casting*.

### 5.2.1. Ray Casting

*Ray Casting* ist ein Verfahren zur Simulation, wieviel Licht anhand eines (Licht-) Strahles zu der sichtbaren Bildfläche (also dem Betrachter) transportiert wird.

Das Verfahren wurde erstmals in „Some Techniques for Shading Machine Renderings of Solids“ 1968 von Appel vorgeschlagen und auch 1968 von der Arlington Mathematical Applications Group, Inc in *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I* erfolgreich umgesetzt [1, 9].

Ein möglicher Algorithmus, wie Ray Casting umgesetzt werden kann, findet sich in Auflistung 5.1.

Bei dem Ray Casting Verfahren wird ein *Projektionszentrum* (das Auge eines Betrachters) sowie eine Region einer beliebigen Bildfläche gewählt. Dabei kann die Region als gerasterte Fläche angenommen werden. Jedes Raster entspricht den Bildpunkten (Pixeln) der gewünschten Auflösung. Je feiner die Rasterung, desto höher die Auflösung.

Für jeden Bildpunkt der gewählten Region wird ein Strahl generiert, welcher dann vom Projektionszentrum durch das Zentrum des Bildpunktes auf die Szene “geworfen” wird.

Es wird dann das Objekt gesucht, welches den nächsten Schnittpunkt mit dem Strahl bildet. Für jede Lichtquelle der Szene wird geprüft, ob die Lichtquelle vom Schnittpunkt aus sichtbar ist. Ist dies der Fall, wird schliesslich die Farbe und die Intensität der Farbe an diesem Schnittpunkt anhand eines Beleuchtungsmodells (z.B. dem Phong-Beleuchtungsmodell) berechnet. Andernfalls befindet sich der Punkt im Schatten, er wird also nicht beleuchtet.

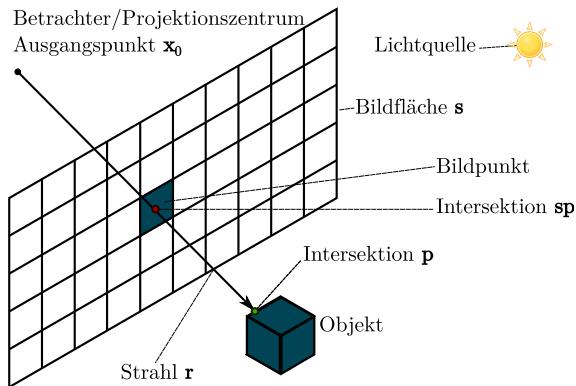


Abbildung 5.2.: Illustration des Ray Casting Verfahrens<sup>2</sup>

Die obenstehende Abbildung 5.2 zeigt das Prinzip des Ray Castings. Ausgangspunkt bildet der Betrachter mit  $x_0 = (x_{x_0}, y_{x_0}, z_{x_0})$ . Es wird ein Strahl  $r = (x_0, d)$  in Richtung  $d$  der Bildfläche  $s$  “geworfen”. Der Strahl  $r$  schneidet die Bildfläche  $s$  an einem Punkt  $sp = (x_{sp}, y_{sp}, z_{sp})$ . Somit ergibt sich die Richtung des Strahles  $d = sp - x_0 = (x_{sp} - x_{x_0}, y_{sp} - y_{x_0}, z_{sp} - z_{x_0}) = (x_d, y_d, z_d)$ . Der Strahl  $r$  trifft und schneidet schliesslich ein Objekt am Punkt  $p = r(t) = x_0 + t \cdot d$  wobei  $t \in [1, \infty]$  ist. Dies führt dazu, dass dem Raster respektive dem Bildpunkt, durch welchen der Strahl  $r$  hindurch geht, die Farbe des getroffenen Objektes zugewiesen wird.

### Berechnung von Schnittpunkten

Um den Schnittpunkt eines Lichtstrahles mit einem Objekt zu berechnen, wird grundsätzlich die mathematische Gleichung des Lichtstrahles in die des Objektes eingesetzt. Existiert eine reelle Lösung, so schneidet der Lichtstrahl das Objekt am nächsten Punkt zwischen dem Schnittpunkt und der Oberflächennormale des Objektes.

Glassner beschreibt mehrere Methoden zur Prüfung von Schnittpunkten: Beispielsweise seien hier die Intersektionen eines Strahles mit einer Kugel oder mit einem Dreieck gezeigt [10, S. 35 bis 64].

#### Schnittpunkt mit einer Kugel

Um den Schnittpunkt bzw. die Schnittpunkte eines Lichtstrahles mit einer Kugel zu erhalten, wird die Gleichung des Lichtstrahles

$$r(t) = r_0 + t \cdot r_d \quad (5.5)$$

---

<sup>2</sup>Eigene Darstellung mittels Inkscape

in die implizite Gleichung einer Kugel

$$\|\mathbf{x} - c\|^2 - r^2 = 0 \quad (5.6)$$

eingesetzt. Dies führt zu folgender Gleichung zweiten Grades:

$$\|r(t) - c\|^2 - r^2 = 0 \quad (5.7)$$

$$\|r_0 + t \cdot r_d - c\|^2 - r^2 = 0 \quad (5.8)$$

Das Auflösen dieser Gleichung ergibt folgende Fälle:

- Zwei Lösungen: Der Strahl geht durch die Kugel hindurch.
- Eine Lösung: Der Strahl streift die Kugel an einem Punkt als Tangente.
- Imaginäre Lösung: Der Strahl verfehlt die Kugel.

#### *Schnittpunkt mit einem Dreieck*

Um den Schnittpunkt eines Lichtstrahles mit einem Dreieck zu erhalten, wird die Gleichung 5.5 des Lichtstrahles als Lösung der Gleichung eines Dreiecks mit baryzentrischen Koordinaten verwendet.

Gegeben sei ein Dreieck  $P$  mit den Ecken  $A$ ,  $B$  und  $C$ :

$$P = \alpha A + \beta B + \gamma C \quad (5.9)$$

Dabei sind  $\alpha$ ,  $\beta$  und  $\gamma$  baryzentrische Koordinaten in Form von reellen Zahlen. Es gilt  $\alpha + \beta + \gamma = 1$ , da baryzentrische Koordinaten normalisiert sind. Möchte man einen Punkt  $(\beta, \gamma)$  innerhalb des Dreieckes so führt dies zu folgender Gleichung:

$$x(\beta, \gamma) = A + \beta \cdot AB + \gamma \cdot AC \quad (5.10)$$

Dabei gilt  $\beta > 0$ ,  $\gamma > 0$  und  $\beta + \gamma \leq 1$ . Ansonsten befindet sich der Punkt nicht innerhalb des Dreieckes. Daraus ergibt sich folgende Gleichung:

$$x(\beta, \gamma) = (1 - \beta - \gamma) \cdot A + \beta \cdot B + \gamma \cdot C \quad (5.11)$$

Setzt man nun die Gleichung des Lichtstrahles (5.5) als Lösung obenstehender Gleichung ein, führt dies zu folgender Gleichung:

$$r_0 + t \cdot r_d = (1 - \beta - \gamma) \cdot A + \beta \cdot B + \gamma \cdot C \quad (5.12)$$

Damit der Lichtstrahl das Dreieck schneidet, gelten dieselben Bedingungen wie bei Gleichung 5.10.

Es leuchtet ein, dass es sehr aufwändig ist jegliches Objekt einer Szene auf Schnittpunkte zu testen (wie dies in Auflistung 5.1 getan wird). Um das Auffinden von Schnittpunkten zu beschleunigen, möchte man den durchschnittlichen Aufwand reduzieren. Dies kann geschehen durch effizientere Algorithmen, z.B. durch Verwendung von Hüllkörpern, oder durch Reduktion der Schnittstellen, z.B. durch Verwendung von Hierarchien von Hüllkörpern oder der Unterteilung des Raumes einer Szene.

Einen guten Überblick bietet das Kapitel “A Survey of Ray Tracing Acceleration Techniques” in *An Introduction to Ray Tracing*, S. 202ff, von [10].

```

1 def ray_cast():
2     # "pixels" is a list of all pixels of the image plane
3     for pixel in pixels:
4         # Save all intersections for given pixel
5         intersections = []
6
7         # Returns the ray passing through the given
8         # pixel from the eye
9         ray = ray_at_pixel(pixel)
10
11        # "scene_triangles" is a list of all triangles
12        # coming from meshes contained in the scene to render
13        for triangle in scene_triangles:
14            p = intersect(ray, triangle)
15            sum = 0
16
17            for light in incoming_lights_at_p:
18                sum = sum + l.value
19            # end for lights
20
21            if is_smallest_intersection(p, intersections):
22                pixel = sum
23            # end if
24            intersections.append(p)
25        # end for triangles
26    # end for pixels
27 # end def ray_cast

```

Auflistung 5.1: Eine abstrakte Umsetzung des Ray Casting Verfahrens<sup>3</sup>.

## 5.2.2. Ray Tracing

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [10, S. 1 bis 77].

Bei dem heute als Ray Tracing bekannten Verfahren handelt es sich um eine verbesserte Version des unter 5.2.1 genannten Ray Casting Verfahrens. Whitted publizierte das Verfahren 1980 im Paper „An Improved Illumination Model for Shaded Display“.

Ein möglicher Algorithmus, wie Ray Tracing umgesetzt werden kann, findet sich in Auflistung 5.2.

Analog dem Ray Casting Verfahren wird wieder ein Projektionszentrum (das Auge eines Betrachters) sowie eine Region einer beliebigen Bildfläche gewählt.

Jeder Bildpunkt der gewählten Region erhält Licht aus nur einer Richtung — der Richtung der Lichtstrahlen (*light rays*), welche durch die gewählte Region und die sichtbare Bildfläche gehen. Somit trägt jedes Photon, welches aus dieser Richtung kommt, zum Farbwert bzw. der Intensität der Farbe eines Bildpunktes bei. Strahlen, welche das Licht direkt zur sichtbaren Bildfläche transportieren, werden *Pixel- bzw. Augen-Strahlen* genannt [10, S. 10].

---

<sup>3</sup>Algorithmus in Pseudocode gemäss [6, Kapitel 15, Seite 391, Auflistung 15.2]

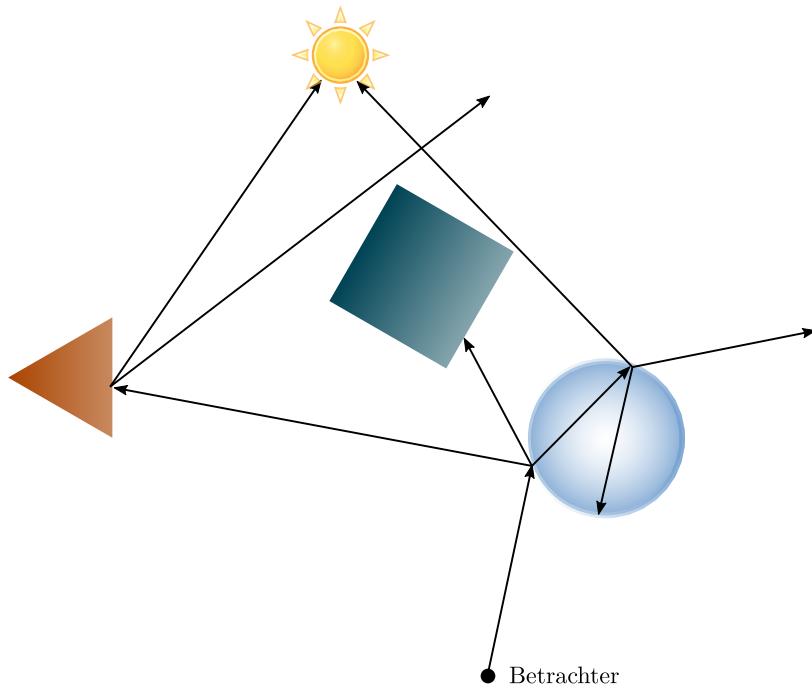


Abbildung 5.3.: Illustration des Prinzips des Ray Tracing Verfahrens<sup>4</sup>

Die Strahlen werden vom Betrachter aus verfolgt, um festzustellen wie das Licht “erzeugt” wird. Trifft der Strahl ins Leere, wird seine Verfolgung beendet und der entsprechende Bildpunkt wird mit der Farbe des Hintergrundes eingefärbt. Trifft der Strahl direkt auf eine Lichtquelle wird seine Verfolgung beendet und der Bildpunkt wird mit der Farbe und Intensität der Lichtquelle eingefärbt. Trifft der Lichtstrahl auf eine Oberfläche, wird der Prozess der Strahlenverfolgung von diesem Punkt aus neu gestartet um festzustellen, wie die Beleuchtung dort zu Stande kam.

Wie der letzte Punkt zeigt, handelt es sich um ein rekursives Verfahren und wird daher zum Teil auch rekursives Ray Tracing genannt. Im Unterschied zu Ray Tracing kennt Ray Casting keine Rekursion.

Zur Berechnung des emittierten Lichtes an einem bestimmten Punkt auf der Oberfläche eines Objektes, wird in einem ersten Schritt die Intensität des Lichtes dieses Punktes bestimmt. Man spricht dabei von *Licht- bzw. Schatten-Strahlen* [10, S. 10].

In einem weiteren Schritt wird berechnet, wie die Oberfläche an diesem Punkt Licht in eine spezifische Richtung weiterleitet, basierend auf der physikalischen Beschaffenheit der Oberfläche.

Trifft ein Lichtstrahl auf eine Oberfläche, wird dieser zu gewissen Teilen *absorbiert, reflektiert* und *gebrochen*. Die jeweiligen Anteile hängen dabei vom Medium der Oberfläche bzw. des Objektes, der Frequenz des Lichtes sowie zwischen dem eingehenden Winkel des Lichtstrahles und der Oberflächennormale ab. Licht kann je nach Medium auch gestreut werden.

Für den Transport des Lichtes kennt man vier Mechanismen: Perfekt diffuse, perfekt spiegelnde und totale interne Reflexion sowie perfekt brechende Refraktion [10, S. 130 bis 137]. In der Realität bzw. der Natur treten mehrere Mechanismen gleichzeitig auf. So wird z.B. ein Teil des Lichtes reflektiert, wohingegen ein anderer Teil das Objekt durchdringt.

Die von diesen Mechanismen emittierten Strahlen können gemäß Glassner in *Reflexion-Strahlen*, welche die perfekte diffuse und die perfekt spiegelnde Reflexion beschreiben, und *Transparenz-Strahlen*, welche die perfekt brechende Refraktion und die totale interne Reflexion beschreiben, unterteilt werden [10, S. 10].

---

<sup>4</sup>Eigene Darstellung mittels Inkscape angelehnt an [10][S. 16]

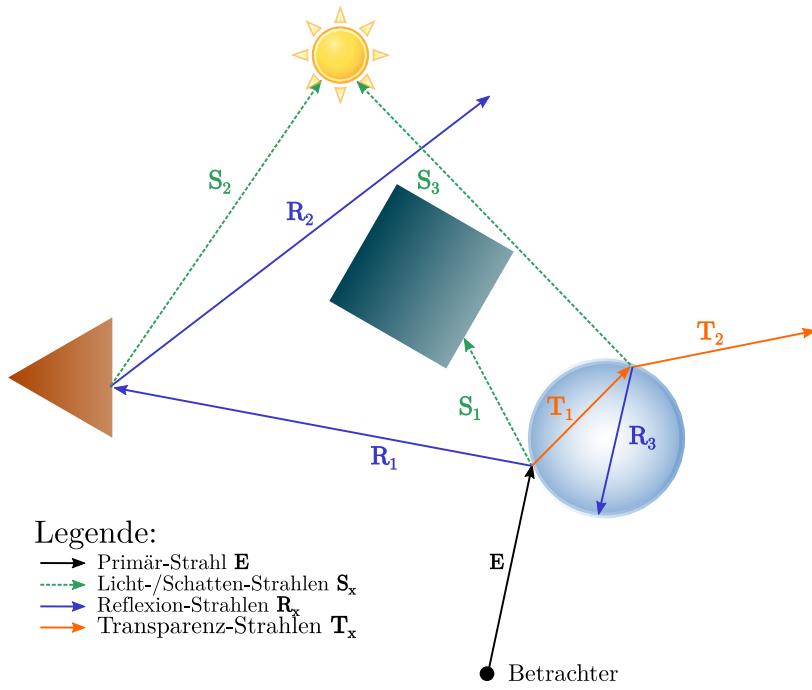


Abbildung 5.4.: Illustration der einzelnen Strahlen und deren Verhalten ausgehend von der Szene in Abbildung 5.3.<sup>5</sup> Das im Bild ersichtliche, orange-farbige Dreieck hat eine undurchlässige, reflexive Oberfläche. Der türkis-farbige Würfel hat eine diffuse Oberfläche. Bei der Kugel, rechts im Bild, handelt es sich um eine Kugel aus Glas, welche das Licht teilweise bricht und reflektiert.

Nachfolgend werden die oben genannten Strahlen im Einzelnen beschrieben.

### **Augen- oder Pixel-Strahlen**

Augen- oder Pixel-Strahlen sind Strahlen, welche das Licht von einer Lichtquelle durch einen Bildpunkt direkt zu der sichtbaren Bildfläche (also dem Betrachter) transportieren.

Die Gleichung solcher Strahlen lautet:

$$r(t) = x_0 + t \cdot (S - x_0) \quad (5.13)$$

Dabei ist  $x_0$  der Ausgangspunkt (also der Betrachter),  $t$  ein Skalierungsfaktor zwischen  $[1, \infty]$  und  $S$  der Schnittpunkt mit der Bildfläche.

### **Licht- oder Schatten-Strahlen**

Bei Licht- bzw. Schatten-Strahlen handelt es sich um Strahlen, welche das Licht von einer Lichtquelle direkt zu der Oberfläche eines Objektes transportieren. Bei jedem Schnittpunkt eines Primär-Strahles (ein Strahl welcher vom Betrachter aus in die Szene „geworfen“ wird) mit einem Objekt wird ein Schatten- bzw. Licht-Strahl in Richtung jeder Lichtquelle der Szene „geworfen“. Trifft der Schatten- bzw. Licht-Strahl die Lichtquelle, wird das Licht zur Berechnung der Farbe und Intensität des Lichtes genutzt. Trifft der Schatten-Strahl keine Lichtquelle, so wird das Licht nicht berücksichtigt. Schatten-Strahlen generieren keine weiteren Strahlen.

---

<sup>5</sup>Eigene Darstellung mittels Inkscape angelehnt an [10, S. 16]

Die Gleichung von Licht- bzw. Schatten-Strahlen lautet:

$$r(t) = p_0 + t \cdot (L_i - p_0) \quad (5.14)$$

Dabei ist  $p_0$  der Ausgangspunkt (also ein Punkt auf einer Oberfläche eines Objektes),  $t$  ein Skalierungsfaktor zwischen  $[\varepsilon, 1]$  und  $L_i$  der Ort der  $i$ -ten Lichtquelle. Bei  $\varepsilon$  handelt es sich um einen Faktor zur Steuerung der Präzision, welcher verhindert, dass ein Objekt lokal auf sich selbst Schatten wirft.

### Reflexion-Strahlen

Das Verhalten von (Licht-) Strahlen, welche auf ein Objekt treffen und an diesem gespiegelt werden, wird durch *Reflexion-Strahlen* beschrieben. Wie oben beschrieben, unterscheidet man zwei Arten der Reflexion: Perfekt diffuse und perfekt spiegelnde Reflexion.

#### Perfekt spiegelnde Reflexion

Bei der perfekt spiegelnden Reflexion verlässt der ausgehende Strahl die Oberfläche im selben Winkel wie der einfallende Strahl. Der Ausfallswinkel entspricht also dem Einfallswinkel.

Die Gleichung eines Strahles, welcher von einer perfekt spiegelnden Reflexion ausgeht, lautet [10, S. 132]:

$$r(t) = p_0 + t \cdot R \quad (5.15)$$

$$R = I - 2(I \cdot N)N \quad (5.16)$$

Dabei sind  $p_0$  der Ausgangspunkt (also ein Punkt auf einer Oberfläche eines Objektes),  $t$  ein Skalierungsfaktor zwischen  $[\varepsilon, 1]$ ,  $I$  der eingehende Vektor, also  $S - x_0$ , und  $N$  die Oberflächennormale. Bei Epsilon handelt es sich wiederum um einen Faktor zur Steuerung der Präzision. Er verhindert, dass ein Objekt lokal in sich selbst gespiegelt wird.

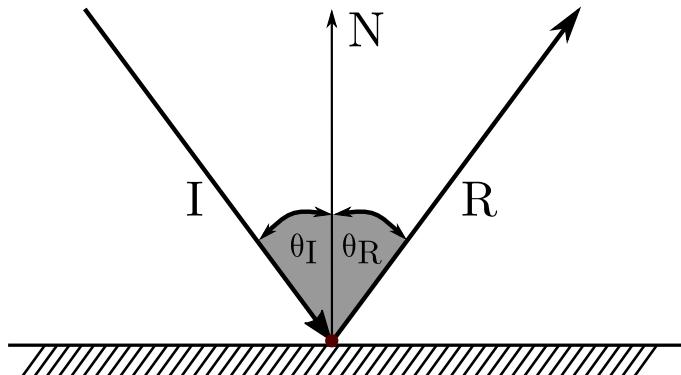


Abbildung 5.5.: Illustration einer perfekt spiegelnden Reflexion.<sup>6</sup>  $I$  ist der eingehende Strahl, welcher am Normalenvektor  $N$  der schraffierte Oberfläche in Richtung  $R$  reflektiert wird. Der Winkel des eingehenden Strahles  $\theta_I$  ist gleichgross wie der Winkel des ausgehenden Strahles  $\theta_R$ .

#### Perfekt diffuse Reflexion

Bei der perfekt diffusen Reflexion wird das eingehende Licht mit gleicher Amplitude (also Stärke) gleichmässig in alle Richtungen gestreut. Die Amplitude ist dabei proportional zum Kosinus des Einfallswinkels und der Oberflächennormale.

---

<sup>6</sup>Eigene Darstellung mittels Inkscape angelehnt an [10, S. 131]

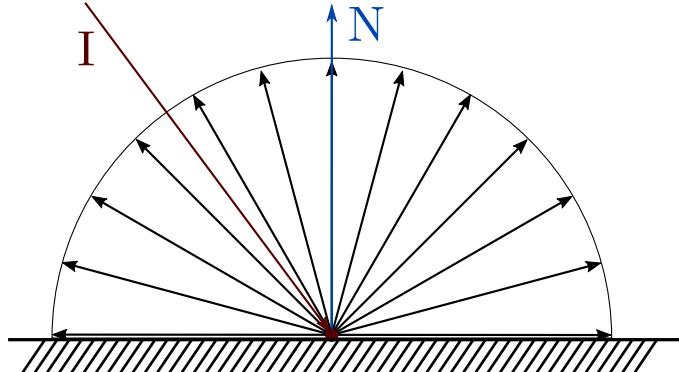


Abbildung 5.6.: Illustration einer perfekt diffusen Reflexion.<sup>7</sup>  $I$  ist der eingehende Strahl, welcher am Ort des Normalenvektors  $N$  der schraffierten Oberfläche auftrifft. Das Licht wird gleichmässig in alle Richtungen gestreut.

### Transparenz-Strahlen

Schliesslich werden die Eigenschaften von Licht, welches durch ein Objekt hindurch geht und vielleicht von diesem gebrochen wird, durch die *Transparenz-Strahlen* beschrieben. Auch hier werden zwei Arten der Reflexion unterschieden: Perfekt brechende Refraktion und total interne Reflexion.

#### Perfekt brechende Refraktion

Tritt Licht von einem Medium (z.B. Luft) in ein anderes Medium (z.B. Glas) ein, so wird das Licht abgelenkt. Die Ablenkung des Lichtes bei Eintritt in ein neues Medium wird auch Transmission oder Refraktion genannt. Dabei hat jedes Medium einen eignen Refraktions-Index (Brechungsindex). Der Refraktions-Index  $n$  gibt das Verhältnis der Vakuumlichtgeschwindigkeit  $c_0$  zur Ausbreitungs geschwindigkeit  $c_M$  des Lichtes in einem Medium  $M$  an:  $n = \frac{c_0}{c_M}$ .

Um zu berechnen, in welche Richtung das Licht abgelenkt wird, werden die Refraktions-Indizes der Medien sowie der Einfalls- bzw. Ausfallswinkel verglichen. Die Beziehung zwischen dem Einfalls- und Ausfallswinkel sowie dem übertragenen Licht wird durch das Gesetz von "Snell" beschrieben:

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \eta_{21} = \frac{\eta_2}{\eta_1} \quad (5.17)$$

Dabei ist  $\theta_1$  der Einfallswinkel,  $\theta_2$  der Ausfallswinkel,  $\eta_1$  der Refraktions-Index des ersten Mediums in Abhängigkeit zu Vakuum,  $\eta_2$  der Refraktions-Index des zweiten Mediums in Abhängigkeit zu Vakuum und  $\eta_{21}$  der Refraktions-Index des zweiten Mediums in Abhängigkeit des ersten Mediums [10, S. 134 bis 135].

Umgekehrt kann das Verhältnis des ausgehenden Strahles zum eingehenden Strahl berechnet werden [10, S. 137 bis 140]:

$$\eta_{12} = \frac{\eta_1}{\eta_2} = \frac{\sin(\theta_2)}{\sin(\theta_1)} \quad (5.18)$$

Mithilfe dieses Verhältnisses kann der gebrochene Strahl berechnet werden wie folgt [10, S. 137 bis 140]:

---

<sup>7</sup>Eigene Darstellung mittels Inkscape angelehnt an [10][S. 134]

$$r(t) = p_0 + t \cdot T \quad (5.19)$$

$$T = \eta_{12}I + (\eta_{12} \cdot C_1 - \sqrt{C_2})N \quad (5.20)$$

$$C_1 = -I \cdot N \quad (5.21)$$

$$C_2 = 1 + \eta_{12}^2(C_1^2 - 1) \quad (5.22)$$

Dabei sind  $p_0$  der Ausgangspunkt (also ein Punkt auf einer Oberfläche eines Objektes),  $t$  ein Skalierungsfaktor zwischen  $[\varepsilon, 1]$ ,  $I$  der eingehende Vektor, also  $S - x_0$ ,  $N$  die Oberflächennormale und  $\eta_{12}$  das oben beschriebene Verhältnis. Bei Epsilon handelt es sich wie beschrieben um einen Faktor zur Steuerung der Präzision. Er verhindert, dass ein Objekt lokal in sich selbst gespiegelt wird.

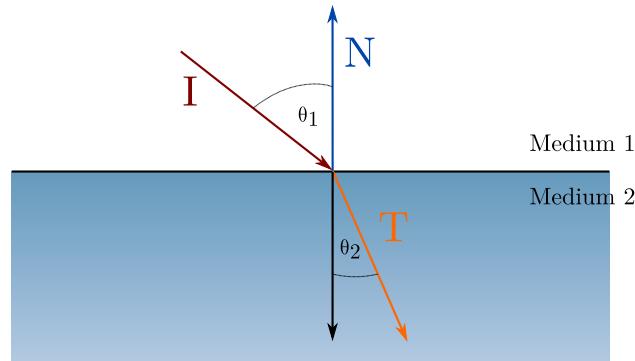


Abbildung 5.7.: Illustration einer perfekt brechenden Refraktion.<sup>8</sup>  $I$  ist der eingehende Strahl, welcher am Ort des Normalenvektors  $N$  im Winkel von  $\theta_1$  von Medium 1 in Medium 2 übergeht und entsprechend mit Winkel  $\theta_2$  anhand  $T$  gebrochen wird.

### Total interne Reflexion

Die totale interne Reflexion tritt dann auf, wenn Licht unter einem zu flachen Winkel versucht von einem dichten Medium in ein weniger dichtes Medium zu gelangen. Das Licht „prallt“ am Übergang der beiden Medien ab und wird gespiegelt anstatt in das andere Medium einzutreten [10, S. 136 bis 137].

Dies geschieht nur dann, wenn der Term  $C_2$  (siehe Gleichung 5.22) negativ ist und somit das Ergebnis der Wurzel aus  $C_2$  in Gleichung 5.20 eine imaginäre Zahl wird [10, S. 137 bis 138].

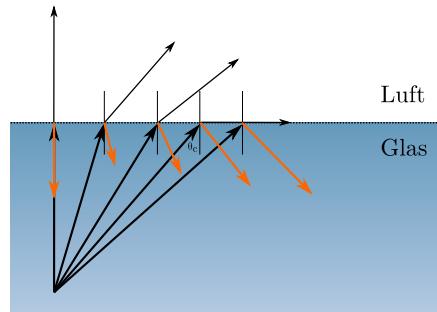


Abbildung 5.8.: Illustration der totalen internen Reflexion.<sup>9</sup> Ist der kritische Winkel  $\theta_c$  beim Übertritt eines Strahles von einem Medium (hier Glas) zu einem anderen Medium (hier Luft) nicht überschritten, so wird der Strahl bzw. Licht sowohl gebrochen als auch reflektiert. Wird der kritische Winkel überschritten, so findet nur noch eine Reflexion statt.<sup>10</sup>

<sup>8</sup>Eigene Darstellung mittels Inkscape angelehnt an [10, S.135]

<sup>9</sup>Eigene Darstellung mittels Inkscape angelehnt an [10, S. 137]

<sup>10</sup>[10, S. 137]

## Modelle zur Schattierung (shading models)

Glassner beschreibt auf der Physik basierende Modelle zur Schattierung (shading models), welche heute unter dem Begriff PBRT — Physically Based Rendering bekannt sind. Diese hier aufzuführen würde den Rahmen dieser Projektarbeit sprengen. Daher wird darauf verzichtet. Der interessierte Leser sei auf *An Introduction to Ray Tracing*, S. 143ff, sowie *Physically Based Rendering, Second Edition: From Theory To Implementation* verwiesen.

## Rekursion und Strahlen-Baum

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [10, S. 16 bis 17].

Wie zu Beginn des Kapitels bereits angesprochen wurde, handelt es sich bei Ray Tracing um ein rekursives Verfahren. Es stellt sich somit die Frage, wie tief die Rekursion gehen soll und auch kann.

Zusätzlich zu dem genannten Fall, dass ein Strahl auf kein Objekt innerhalb der Szene trifft, schlagen Whitted wie auch Glassner den Abbruch in folgenden Fällen vor:

- Nach Erreichen einer festgelegten, maximalen Tiefe, sofern die Rekursion rein auf reflexive Oberflächen trifft.
- Nach dem Auftreffen auf eine rein diffuse Oberfläche.
- Nach Unterschreiten eines minimalen Energiewertes des Lichtes.

Aus den genannten Abbruch-Kriterien ergibt sich nach Heckbert folgende, auf einem regulären Ausdruck basierende Lichtweg-Notation: **LD?S\*E** [12].

Diese beschreibt den Weg, welchen ein Photon ausgehend von einer Lichtquelle **L** zum Auge eines Betrachters **E** nehmen kann. Es kann auf null oder genau eine diffuse Oberfläche (**D?**) so wie auf 0 oder theoretisch unendlich viele reflektierende Oberflächen (**S\***) treffen [12, S. 148].

Ausgehend von der sichtbaren Bildfläche, also dem Auge des Betrachters der Szene, kann so ein Strahlenbaum (*ray tree*) aufgebaut werden.

Jeder Schnittpunkt eines Strahles mit einem Objekt kann Sekundär-Strahlen generieren. Dadurch bilden reflektierte und gebrochene Strahlen den sogenannten Strahlen-Baum.

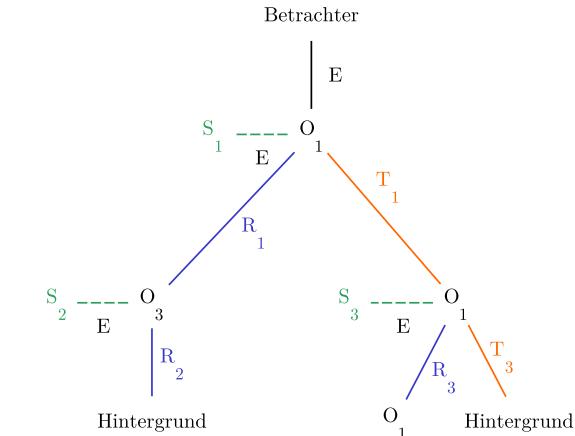
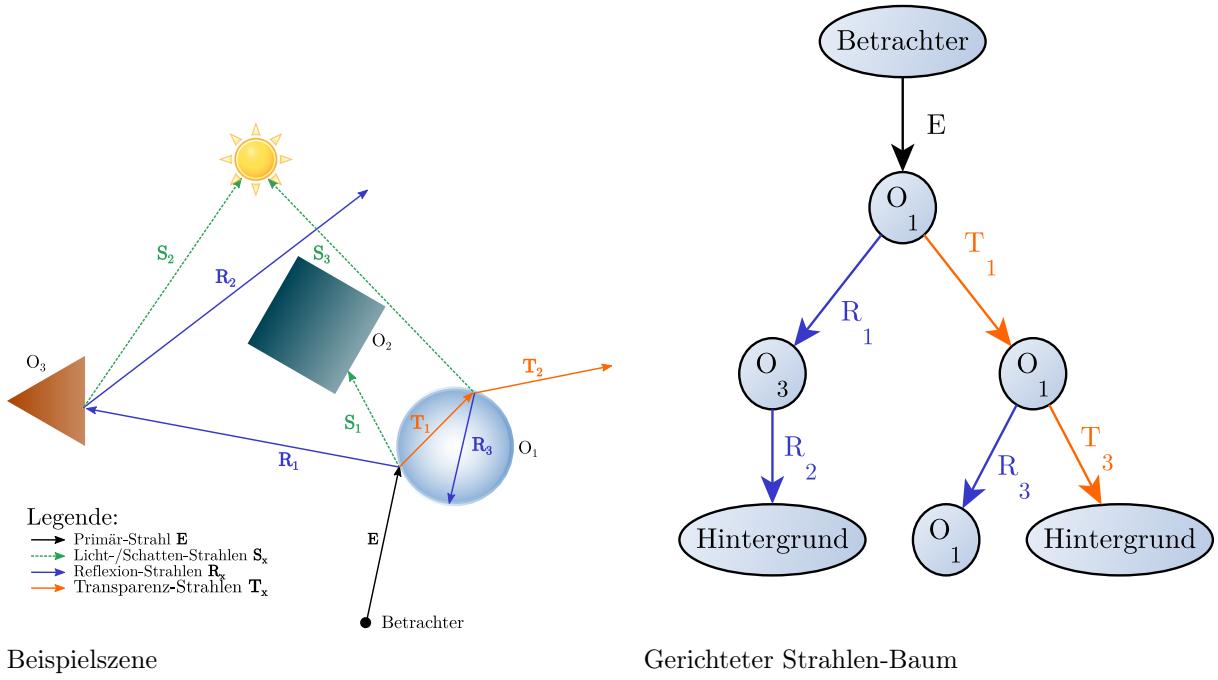
Dabei sind die *Knoten* des Baumes die Schnittpunkte und die *Kanten* sind die reflektierten oder gebrochenen Strahlen.

Wie bereits erwähnt wird bei jedem Schnittpunkt ein Schatten-Strahl ausgesendet, welcher aber keine zusätzlichen Strahlen und somit auch keine zusätzlichen Kanten generiert.

Der Strahlen-Baum kann schliesslich von unten nach oben (*bottom-up*) traversiert werden, was einer Tiefensuche (*depth-first traversal*) entspricht. Die Farbe eines Knoten wird aufgrund der Farbe der Kindes-Knoten (*child node*) berechnet.

Tabelle 5.1.: Darstellung des Strahlen-Baumes anhand einer Beispieldszene. Die Lichtweg-Notation des Ray Tracings ( $LD?S^*E$ ) wird hier ersichtlich: Das Objekt  $O_3$  muss einen spiegelnden Anteil in seiner Oberflächenbeschaffenheit haben, ansonsten würde die Strahlenverfolgung nach dem Auftreffen des Strahles  $R_1$  bereits beendet.

---



Schematischer Strahlen-Baum nach Glassner<sup>11</sup>

---

<sup>11</sup>[10, S. 17]

```

1
2 def ray_trace(current_point, direction):
3     """Traces light rays from current point in space in given direction.
4
5     :param current_point: current point in space
6     :type current_point: three dimensional point object
7     :param direction: the direction to trace
8     :type direction: three dimensional vector
9
10    :return: the color for the given point
11    :rtype: float
12    """
13
14    # Set color to currently set background color
15    color = self.background_color
16
17    # "pixels" is a list of all pixels of the image plane
18    for pixel in pixels:
19
20        # Returns the ray passing through the given
21        # pixel from the eye
22        ray = ray_at_pixel(pixel)
23
24        # object_list is a list containing all the meshes contained in
25        # the scene to render
26        for object in object_list:
27            p = intersect(ray, object)
28
29            if object.is_reflective:
30                reflection_vector = reflect(direction, object)
31                reflected_color = ray_trace(p, reflection_vector)
32                color = color + object.coeff * reflected_color
33            # end if
34
35            if object.is_refractive:
36                refracted_vector = refract(direction, object)
37                refracted_color = ray_trace(p, refracted_vector)
38                color = color + object.coeff * refracted_color
39            # end if
40
41            for light in incoming_lights_at(p):
42                if not is_shadow_ray(p, light.position):
43                    color = color + calc_lighting(p, direction, light, object)
44                # end if
45            # end for lights
46        # end for objects
47    # end for pixels
48
49    return color
50 # end def ray_trace

```

Auflistung 5.2: Eine abstrakte Umsetzung des Ray Tracings <sup>12</sup>.

---

Algorithmus in Pseudocode gemäss [10, Seite 283]

## 5.3. Modelle zur Schattierung (shading models)

Sofern im Text nicht anders vermerkt, basieren die nachfolgenden Abschnitte auf [5, S. 734–739], sowie [6].

Bei der Anwendung von Modellen zur Schattierung (*shading models*) geht es grundsätzlich darum die emittierte Intensität des Lichtes bzw. die Farbe einer Oberfläche an einem bestimmten Punkt zu berechnen. Naheliegend wäre, dies für jeden sichtbaren Punkt der Oberfläche zu berechnen. Jedoch ist dies häufig meist zu aufwändig. Daher berechnen viele Modelle zur Schattierung die Intensität des Lichtes bzw. der Farbe nur an gewissen Schlüssel-Punkten. Und wenden zusätzlich vereinfachte Modelle zur Berechnung der Schattierung an. Dadurch wird Rechenzeit gespart, aber die Plastizität erleidet Einbussen.

Als Beispiel zur Anwendung der Modelle zur Schattierung wird nachfolgend angenommen, dass mittels einem Beleuchtungsmodell an jedem Eckpunkt einer Oberfläche (Polygon) die Farbe berechnet wird. Als Beispiel dienen hier die Eckpunkte  $V_1$ ,  $V_2$ ,  $V_3$  und  $V_4$ .

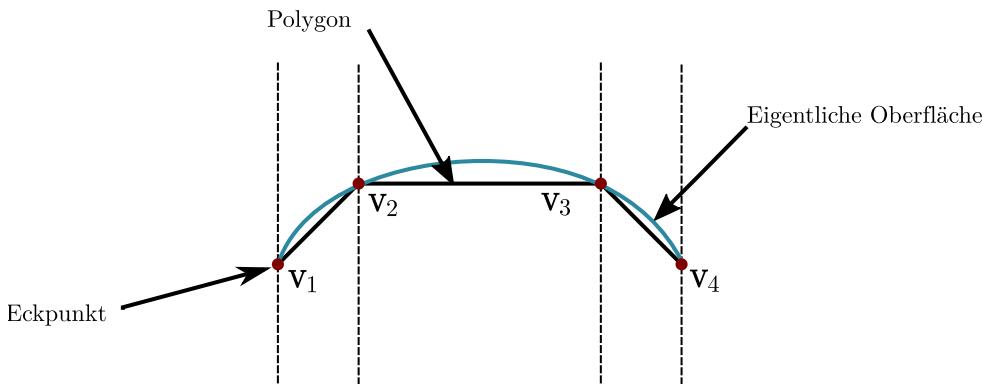


Abbildung 5.9.: Illustration der Ausgangslage<sup>13</sup>

Um die Intensität der Farbe für einen Eckpunkt  $V_1$  zu berechnen, wird der Normalenvektor des Eckpunktes (*vertex normal*) benötigt. Es handelt sich dabei um den Normalenvektor der Oberfläche  $N$  an der Position des Eckpunktes  $V_1$ .

Im Laufe der Zeit wurde die Berechnung der Schattierung so weit entwickelt, dass sie praktisch ausschliesslich durch Grafikkarte (GPU) gemacht werden kann. Man spricht hier vom Begriff “Shader”. Es handelt sich dabei mittlerweile um eigene (Grafik-) Applikationen, deren Zweck weit über die reine Berechnung von Schattierungen hinausgeht. Man unterscheidet zwischen Shadern für geometrische Berechnungen (*geometry shaders* oder *vertex shaders*) sowie Shader für Berechnungen bezüglich Pixeln oder (Bild-)Fragmenten (*pixel shaders* oder *fragment shaders*) [6, Kapitel 33].

Der interessierte Leser sei für weitere Details auf *Computer Graphics: Principles and Practice*, Kapitel 33 von [6], auf *Rendering Pipeline Overview - OpenGL.org* von [13] sowie *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics* von [14] verwiesen.

### 5.3.1. Flat-Shading — per vertex lighting

Bei Flat-Shading wird pro Oberfläche (Polygon) ein Eckpunkt  $V_1$  als Schlüssel-Punkt für die Farbe bzw. die Intensität bestimmt. Daraufhin wird die Farbe des Punktes als Farbe für die gesamte Oberfläche angenommen [5, S. 734].

Diese Annahme ist nur unter den folgenden Voraussetzungen gültig [5, S. 734]:

1. Die zugrundeliegende Lichtquelle befindet sich unendlich weit entfernt, so dass der Winkel zwischen dem Normalenvektor der Oberfläche  $n$  und der Lichtquelle  $l$ , also  $n \cdot l$ , für die gesamte Oberfläche konstant ist.

<sup>13</sup>Eigene Darstellung mittels Inkscape, angelehnt an [6]

2. Der Betrachter befindet sich ebenfalls unendlich weit von der Oberfläche entfernt, so dass der Winkel zwischen dem Normalenvektor der Oberfläche und dem Betrachter  $\mathbf{n} \cdot \mathbf{v}$  für die gesamte Oberfläche konstant ist.
3. Das Polygon ist eine effektive Repräsentation der Oberfläche und nicht nur eine Näherung einer runden Oberfläche.

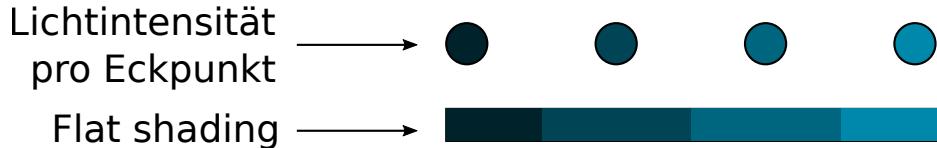


Abbildung 5.10.: Illustration des Flat-Shadings<sup>14</sup>

Ist eine der ersten beiden Voraussetzungen nicht gegeben, so muss für den Vektor der Lichtquelle  $\mathbf{l}$  bzw. den Vektor des Betrachters  $\mathbf{v}$  ein konstanter Wert berechnet werden. Foley gibt hier als Beispiele das Zentrum des Polygones oder den ersten Eckpunkt des Polygones an [5, S. 735].

### 5.3.2. Gouraud-Shading — face interpolated lighting

Bei Gouraud-Shading handelt es sich um ein Shading-Verfahren, welches die Intensität der Farbe der Eckpunkte von Oberflächen eines Polygon-Netzes (*Mesh*) interpoliert.

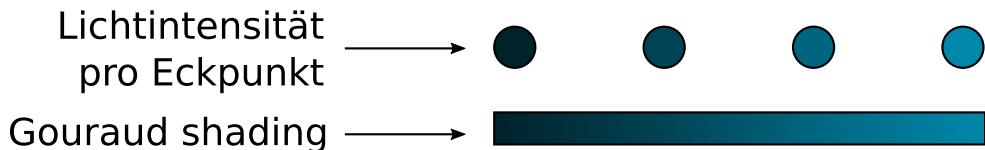


Abbildung 5.11.: Illustration des Gouraud-Shadings<sup>15</sup>.

Um die Farbe eines Eckpunktes von Oberflächen zu berechnen, schlägt Gouraud in seiner Arbeit „Continuous Shading of Curved Surfaces“ von 1971 die Berechnung des Durchschnittswertes der Normalenvektoren der Oberflächen zweier adjazenter Liniensegmente (im 2D-Raum) bzw. aller adjazenter Dreiecke (im 3D-Raum) vor [15, S. 92].

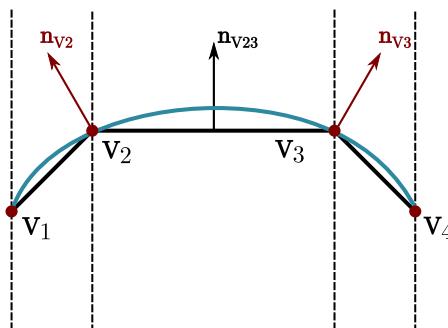


Abbildung 5.12.: Illustration der berechneten Normalenvektoren  $\mathbf{n}_{v_2}$  und  $\mathbf{n}_{v_3}$  an den Eckpunkten  $v_2$  und  $v_3$  sowie des Normalenvektors  $\mathbf{n}_{v_{23}}$  der Kante  $v_2v_3$ <sup>16</sup>.

<sup>14</sup>Eigene Darstellung mittels Inkscape, angelehnt an [6]

<sup>15</sup>Eigene Darstellung mittels Inkscape, angelehnt an [6]

<sup>16</sup>Eigene Darstellung mittels Inkscape, angelehnt an [6]

Die Berechnung des Normalenvektors eines Eckpunktes via Durchschnittswert ist üblicherweise eine genügend gute Näherung an die Oberflächennormale der eigentlichen Oberfläche. Die Präzision hängt dabei aber eindeutig von der Granularität des Modelles (*Mesh*) ab.

### 5.3.3. Phong-Shading — normal interpolated lighting

Phong-Shading ist eine Verbesserung des Gouraud-Verfahrens. Es bietet eine bessere Annäherung an die Schattierung bzw. Darstellung von glatten Oberflächen. Das Verfahren ist geeigneter für ein Beleuchtungsmodell, welches kleine spiegelnde Reflexionen bietet, wie z.B. das Phong-Beleuchtungsmodell.

Bei Phong-Shading wird ein Normalenvektor linear an einer Oberfläche (eines Polygons) interpoliert, ausgehend von den Normalenvektoren der Kanten der Oberfläche (im obigen Beispiel wäre dies  $\mathbf{n}_{v_2 v_3}$ ). Die Oberflächennormale wird an jedem Pixel interpoliert und normalisiert und kommt dann für die Berechnung der Farbwerte anhand eines Beleuchtungsmodells, wie zum Beispiel das Phong-Beleuchtungsmodell, zum Einsatz. Daher ist Phong-Shading viel rechenintensiver als Gouraud-Shading, da das Beleuchtungsmodell für jeden Pixel anstatt für jede Kante berechnet werden muss.

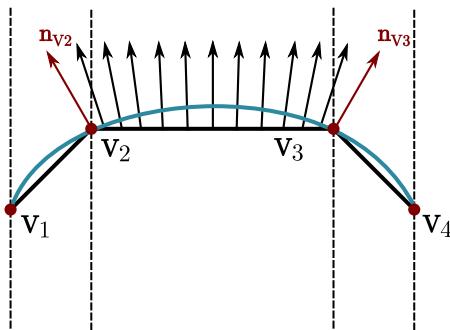


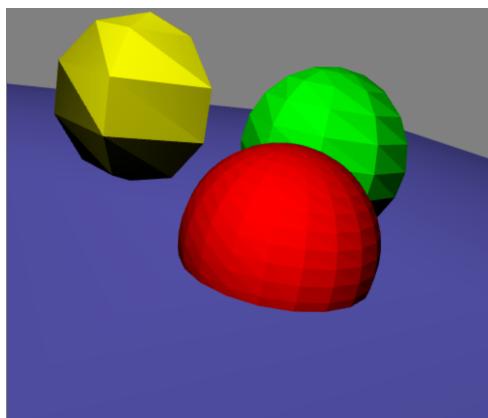
Abbildung 5.13.: Stark vereinfachte Illustration der interpolierten Normalenvektoren anhand der Kante  $v_2 v_3$ <sup>17</sup>.

---

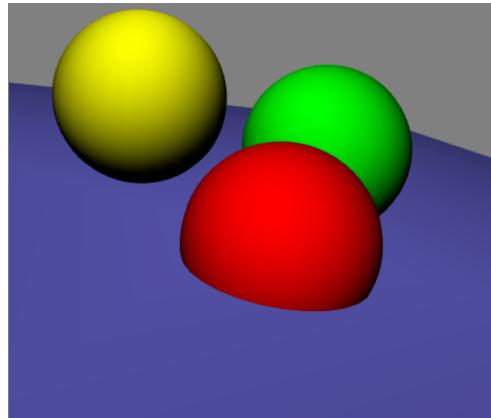
<sup>17</sup>Eigene Darstellung mittels Inkscape, angelehnt an [6]

Tabelle 5.2.: Vergleich der genannten Shading-Verfahren anhand einer Beispielszene<sup>18</sup>.

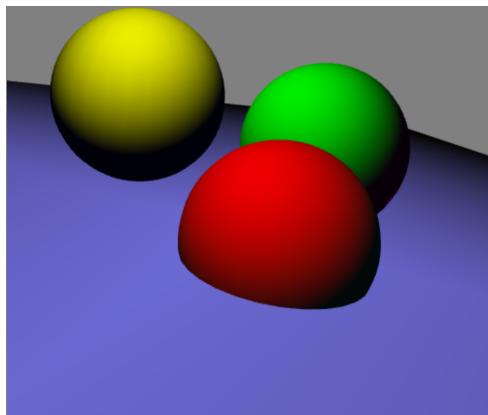
**Flat-Shading**



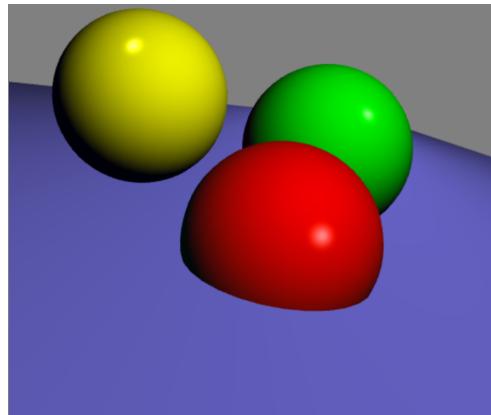
**Gouraud-Shading**



**Gouraud-Shading  
mit Phong-Beleuchtungsmodell**



**Phong-Shading  
mit Phong-Beleuchtungsmodell**



<sup>18</sup>[16]

# 6. Oberflächen

## 6.1. Oberflächen

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [17, S. 1ff], sowie auf [10, S. 79 bis 115].

Für die Darstellung eines Bildes mittels Computergrafik, muss zunächst definiert werden, was dargestellt werden soll. Dabei orientiert sich die Computergrafik dabei an der realen Welt. In dieser haben Oberflächen von Objekten häufig keine starken Übergänge (Kanten) sondern sind eher glatter Natur [5, S. 471].

Die Darstellung von Kurven und Oberflächen führt zu zwei Möglichkeiten: Modellierung von bestehenden Objekten und Modellierung von Grund auf.

Zur Modellierung von Oberflächen werden hauptsächlich zwei Techniken verwendet: Parametrische Modellierung und implizite Modellierung.

### 6.1.1. Parametrische Darstellung von Oberflächen

Bei der parametrischen Darstellung wird eine Oberfläche üblicherweise als eine Menge von Punkten definiert, so zum Beispiel:

$$\mathbf{p}(s, t) = (x(s, t), y(s, t), z(s, t)) \quad (6.1)$$

So kann eine Kugel mittels sphärischen Koordinaten parametrisch dargestellt werden wie folgt:

$$x(s, t) = r \cdot \cos(s) \cdot \sin(t) \quad (6.2)$$

$$y(s, t) = r \cdot \sin(s) \cdot \sin(t) \quad (6.3)$$

$$z(s, t) = r \cdot \cos(t) \quad (6.4)$$

Dabei ist  $r$  der Radius der Kugel,  $s$  der Azimutwinkel  $\in [0, 2\pi]$  und  $t$  der Polarwinkel  $\in [0, \pi]$ .

Die parametrische Darstellung bringt Vorteile wie Unabhängigkeit von einem Koordinatensystem oder effiziente Berechnung von Punkten auf einer Oberfläche.

### 6.1.2. Implizite Darstellung von Oberflächen

Bei der impliziten Darstellung wird eine Oberfläche üblicherweise als Kontur einer Funktion mit Wert 0 definiert [17, S. 1], so zum Beispiel:

$$f(\mathbf{p}) = f(x, y, z) = 0 \quad (6.5)$$

Dabei ist  $\mathbf{p}$  ein Punkt  $(x, y, z)$  auf der Oberfläche, welche durch die Funktion  $f$  beschrieben wird.

Am Beispiel einer Kugel:

$$f(\mathbf{p}, r) = f(x, y, z, r) = 0 \quad (6.6)$$

$$f(\mathbf{p}, r) = \mathbf{p}^2 - r^2 = 0 \quad (6.7)$$

$$f(\mathbf{p}, r) = x^2 + y^2 + z^2 - r^2 = 0 \quad (6.8)$$

Dabei ist  $\mathbf{p}$  ein Punkt  $(x, y, z)$  auf der Oberfläche, welche durch die Funktion  $f$  beschrieben wird, und  $r$  der Radius der Kugel [10, S. 91].

Die implizite Darstellung erlaubt aus mathematischer Sicht eine grössere Einflussnahme als die parametrische Darstellung und ist daher sehr nützlich für Operationen wie Biegung, Vermischung, Schnitte (Intersektion) oder Boolesche Operationen.

### 6.1.3. Implizite Oberflächen

Wie in Gleichung 6.5 beschrieben, ist eine implizite Oberfläche gemäss Menon typischerweise als Kontur einer Funktion mit Wert 0 definiert [17, S. 1]. Hart hingegen definiert eine implizite Oberfläche als Funktion  $f(\mathbf{x}) = \mathbb{R}^3 \rightarrow \mathbb{R}$ . Jedem Punkt einer Menge  $\mathbf{p} \in \mathbb{R}^3$  wird ein skalarer Wert  $s \in \mathbb{R}$  zugewiesen. Dabei besteht die Oberfläche aus der Menge von Punkten  $\mathbf{x} \equiv (x, y, z) \in \mathbb{R}^3$  [19, S. 527].

Angenommen  $\mathbf{A}$  ist ein geschlossener Festkörper, welcher durch die Funktion  $f$  beschrieben wird, dann kann gemäss Hart Folgendes angenommen werden:

$$x \in \overset{\circ}{\mathbf{A}} \Leftrightarrow f(\mathbf{x}) < 0 \quad (6.9)$$

$$x \in \partial \mathbf{A} \Leftrightarrow f(\mathbf{x}) = 0 \quad (6.10)$$

$$x \in \mathbb{R}^3 - \mathbf{A} \Leftrightarrow f(\mathbf{x}) > 0 \quad (6.11)$$

Dies bedeutet, dass sich ein Punkt  $\mathbf{x}$ :

- $\overset{\circ}{\mathbf{A}}$

Genau dann *innerhalb* des Körpers  $\mathbf{A}$  befindet, wenn die implizite Funktion  $f(\mathbf{x})$  *negativ* ist.

- $\partial \mathbf{A}$

Genau dann *auf der Oberfläche* des Körpers  $\mathbf{A}$  befindet, wenn die implizite Funktion  $f(\mathbf{x})$  0 ist.

- $\mathbb{R}^3 - \mathbf{A}$

*Nicht in oder auf* dem Körper  $\mathbf{A}$  befindet, wenn die implizite Funktion  $f(\mathbf{x})$  *positiv* ist.

Obige Annahmen sind gültig, da es sich bei  $\mathbf{x}$  um eine Menge von Punkten mit topologischer Struktur handelt [18, S. 1].

Gemäss Menon finden hauptsächlich drei Methoden zur Beschreibung impliziter Oberflächen Anwendung: Algebraische Oberflächen, Blobby-Objekte sowie die funktionale Repräsentation [17, S. 1].

Hart gibt jedoch an, dass die gebräuchlichste Form zur Darstellung von impliziten Oberflächen die algebraischen Oberflächen sind [19, S. 527]. Diese werden implizit durch polynomiale Funktionen definiert.

### Algebraische und geometrische implizite Oberflächen

Ein Beispiel für eine algebraische Oberfläche ist die Beschreibung der Einheitskugel anhand einer impliziten algebraischen Gleichung zweiten Grades:

$$x^2 + y^2 + z^2 - r = 0 \quad (6.12)$$

$$x^2 + y^2 + z^2 - 1 = 0 \quad (6.13)$$

Dabei hat Radius  $r$  den Wert 1, da es sich um die Einheitskugel handelt.

Menon gibt an, dass algebraische Methoden Oberflächen als implizite Polynome beschreiben. So ist  $F(x, y, z)$  beispielsweise ein Polynom in  $x, y$  und  $z$  [17, S. 2].

Üblicherweise handelt es sich dabei um Polynome tiefen Grades (2, 3 oder 4) [17, S. 2].

Die gebräuchlichsten Oberflächen sind quadratisch implizite Oberflächen (vom Grad 2). Diese werden häufig als "Quadrik" (*quadrics*) bezeichnet werden [17, S. 2].

Hart gibt an, dass — unter Nutzung einer Metrik — die Einheitskugel durch die implizite Gleichung geometrisch beschrieben werden kann:

$$\|\mathbf{x}\| - 1 = 0 \quad (6.14)$$

Unter Anwendung der allgemeinen Form einer impliziten Gleichung (siehe Gleichung 6.5) führt dies zu folgender Gleichung:

$$f(\mathbf{x}) = \|\mathbf{x}\| - 1 \quad (6.15)$$

Dabei ist  $\|\mathbf{x}\|$  als euklidische Metrik definiert und entspricht  $(x^2 + y^2 + z^2)^{\frac{1}{2}}$ .

Die Gleichung 6.12 gibt die algebraische Distanz zurück, Gleichung 6.14 gibt die geometrische Distanz zurück.

Gemäß Hart wird die geometrische Darstellung von quadratischen Oberflächen bevorzugt, da deren Parameter unabhängig von Koordinaten und sie robuster und intuitiver sind.

Implizite Oberflächen werden bei Gleichungen der Form 6.15 durch *Distanzfunktionen* repräsentiert. Diese messen oder binden gemäß Hart die geometrische Distanz zu deren impliziten Oberflächen [19, S. 529].

## Distanzfunktionen

Wie anfangs erwähnt, definiert Hart die allgemeine Form zur Beschreibung bzw. Darstellung von impliziten Oberflächen als Zuweisung von Punkten zu einem skalaren Wert:  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  [19, S. 527].

Unter Anwendung der unter Gleichung 6.9 definierten Bedingungen kann geschlossen werden, dass eine Menge von Punkten  $A$  existiert, welche Teil von  $\mathbb{R}^n$ , also  $A \subset \mathbb{R}^n$  ist. Dies heißt, dass alle Punkte in  $A$  die folgende Bedingung erfüllen:

$$A = \{x : f(x) \leq 0\} \quad (6.16)$$

Hart liefert folgende zwei Definitionen, welche der Beschreibung von Distanzfunktionen dienen.

### Definition 6.1.1. Point-to-set distance

Die Distanz eines Punktes zu einer Menge von Punkten definiert die Distanz eines Punktes  $\mathbf{x} \in \mathbb{R}^3$  zu einer Menge von Punkten  $A \subset \mathbb{R}^3$  als Distanz von  $\mathbf{x}$  zum nächsten Punkt in  $A$ :

$$d(\mathbf{x}, A) = \min_{y \in A} (\|\mathbf{x} - \mathbf{y}\|) \quad (6.17)$$

### Definition 6.1.2. Signed distance bound

Eine Funktion  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  ist eine Vorzeichen abhängige Obergrenze ihrer impliziten Oberfläche  $f^{-1}(0)$ , wenn gilt:

$$|f(\mathbf{x})| \leq d(\mathbf{x}, f^{-1}(0)) \quad (6.18)$$

Wenn die Gleichung 6.18 für eine Funktion  $f$  gilt, dann ist  $f$  eine *Vorzeichen abhängige Distanzfunktion* (*signed distance function*).

## Distanzfelder (distance fields)

Sofern im Text nicht anders vermerkt, basiert der folgende Abschnitt auf [20].

Bei einem Distanzfeld handelt es sich um eine Repräsentation respektive um eine Datenstruktur, bei welcher von jedem Punkt aus die geringste Distanz zu einem beliebigen Objekt der Domäne bekannt ist.

Zusätzlich zu der Distanz können aus einem Distanzfeld andere Parameter, wie beispielsweise die Richtung einer Oberfläche, abgeleitet werden.

Handelt es sich bei einem Distanzfeld um ein mit Vorzeichen behaftetes Distanzfeld, so kann bei jedem Punkt bestimmt werden, ob er sich innerhalb, ausserhalb oder genau auf einem Objekt befindet. Dies verhält sich analog zu Unterabschnitt 6.1.3.

Distanzfelder bilden somit die Grundlage der Algorithmen zur Darstellung von impliziten Oberflächen. Siehe dazu auch Abschnitt 6.3. Die unter Abschnitt 6.1.3 genannte *Point-to-set distance*-Funktion liefert bereits die nächste Distanz zu einer Menge von Punkten und somit auch zu Objekten ausgehend von einem Punkt der Domäne.

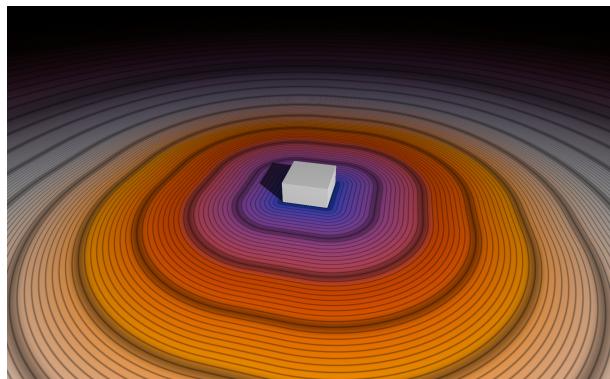


Abbildung 6.1.: Darstellung des Distanzfeldes einer dreidimensionalen Szene anhand Farbwerten<sup>1</sup>

## 6.2. Darstellung von impliziten Oberflächen

Nach Hart, existieren verschiedene Möglichkeiten zur Darstellung (zum Rendering) von impliziten Oberflächen. So wandeln indirekte Methoden implizite Oberflächen in Polygon-Modelle um, was die Nutzung bestehender Techniken und Hardware zur Darstellung von polygonalen Modellen erlaubt. Die Umwandlung in Polygon-Modelle ist jedoch nicht in jedem Falle gegeben. Sie kann zu unzusammenhängenden Flächen oder zu einer Verminderung des Detaillierungsgrades führen. Speicherbedarf und Zeitaufwand können zudem sehr gross sein [19, S. 528].

Eine andere Methode zur Darstellung von impliziten Oberflächen ist das in Unterabschnitt 5.2.2 vorgestellte Ray Tracing Verfahren.

Ein (Licht-) Strahl wird dabei parametrisch als

$$r(t) = r_0 + t \cdot r_d \quad (6.19)$$

beschrieben. Der Strahl startet dabei bei Punkt  $r_0$  in Richtung des Einheitsvektors  $r_d$ , wobei  $t$  die zurückgelegte Distanz des Strahles ist. Dabei ist  $r(t)$  derjenige Punkt im Raum, welchen der Strahl nach dem Zurücklegen der Distanz  $t$  — ausgehend von seinem Ursprung  $r_0$  — erreicht [19, S. 528].

---

<sup>1</sup>Eigene Darstellung

Um nun die Schnittpunkte eines Strahles mit einer impliziten Oberfläche zu finden, wird die Gleichung des Lichtstrahles  $r$  (6.19) in die Funktion einer impliziten Oberfläche  $f$  (6.5) eingesetzt. Wobei  $r : \mathbb{R} \rightarrow \mathbb{R}^3$  und  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ . Dies ergibt die zusammengesetzte Funktion  $F = f \circ r$  wobei  $F : \mathbb{R} \rightarrow \mathbb{R}$  [19, S. 528].

Die Lösungen dieser Gleichung ergibt alle Distanzen  $t$ , welche ein gegebener Strahl zurücklegt und welche die folgende Bedingung erfüllen [19, S. 528]:

$$F(t) = f \circ r = f(r(t)) = 0 \quad (6.20)$$

Für die Lösung der Gleichung 6.20, können numerische Verfahren zur Suche von Nullstellen angewendet werden. Dabei sind die Verfahren vom Typen der Funktion  $F(t)$  abhängig. Bei polynomiauen Funktionen bis zum vierten Grad existieren analytische Lösungen. Idealerweise genügt für das Finden der Nullstellen die Auswertung der Funktion  $F$  am Punkt  $t$ . Dabei können jedoch Nullstellen verloren gehen [19, S. 528].

Für eine beliebige Funktion  $F(t)$  muss daher ein generisches, robustes Verfahren zur Suche von Nullstellen verwendet werden. Eine solches Verfahren benötigt jedoch zusätzliche Informationen und geht somit über eine einfache Auswertung der Funktion hinaus. Die zusätzlichen Informationen können zumeist aus der Ableitung der Funktion gewonnen werden [19, S. 528].

Ein häufiger Nachteil der erwähnten Verfahren Nullstellen-Suche ist, dass sie mehrere Schnittpunkte eines Strahles mit einer impliziten Oberfläche liefern. Zur Umgehung dieser Problematik, wird nur der kleinste Wert von  $t$  — also der geringste Abstand — berücksichtigt [19, S. 531].

Die Ray Marching und Sphere Tracing Algorithmen gehen hier sogar noch einen Schritt weiter. Sie betrachten nur die kleinste positive Nullstelle der Gleichung 6.20 [19, S. 531].

### 6.2.1. Ray Marching

Perlin und Hoffert schlagen eine Abtastung des Strahles mit fixen Abständen  $\Delta x_\mu$  vor [21, S. 259]:

$$x = x_{\mu_0} + k \cdot \Delta x_\mu \quad (6.21)$$

Dabei ist  $k = 0, 1, 2, \dots$  und  $\mu_0 + k\Delta\mu \leq \mu_1$ .

Auf die parametrische Darstellung eines (Licht-) Strahles, Gleichung 6.20, angewendet, ergibt dies folgende Gleichung:

$$r(k) = r_0 + \Delta t \cdot k \cdot r_d \quad (6.22)$$

Dabei stellt  $\Delta t$  die Grösse der Abstände und  $k = 0, 1, 2, \dots$  die Nummer der Schritte dar. Wie Hart, Sandin und Kauffman schreiben, bildet das Abtasten des (Licht-) Strahles mit fixen Abständen die Basis für gewisse Verfahren des volumetrischen Renderings [22, S. 291].

Ein möglicher Algorithmus, wie solch ein Verfahren umgesetzt werden kann, findet sich in Auflistung 6.1.

```

1 def ray_march():
2     step = 0
3     intersection = 0
4     max_steps = 10
5
6     while step < max_steps:
7         intersection = test_intersection(k)
8
9         if intersection <= 0:
10             # An intersection has happened
11             # intersection < 0: ray is inside surface
12             # intersection == 0: ray is exactly on surface
13             return ray_travel_distance(step)
14         # end if
15
16         step = step + 1
17     # done while
18
19     # When we reach this step, after max_steps, no intersection
20     # has happened, so distance is 0
21     return 0
22 # end ray_march

```

Auflistung 6.1: Eine abstrakte Umsetzung des Ray Marchings<sup>2</sup>.

---

Algorithmus in Pseudocode gemäss [21][S. 259], Abschnitt 3.1]

Dabei ist zu beachten, dass der Abstand der Abtastung eines Strahles  $\Delta t$  so gering als möglich sein sollte um eine Menge von Punkten bzw. ein Objekt  $A$  — definiert durch implizite Oberflächen — möglichst gut abschätzen zu können. Ist der Abstand zu gross gewählt, so findet ggf. eine Abtastung weit im Inneren des Objektes statt. Damit geht Präzision verloren. Es ist weiter möglich, dass der erste eigentliche Punkt gar nicht abgetastet wird und erst der zweite abgetastete Punkt das Objekt “erkennt”.

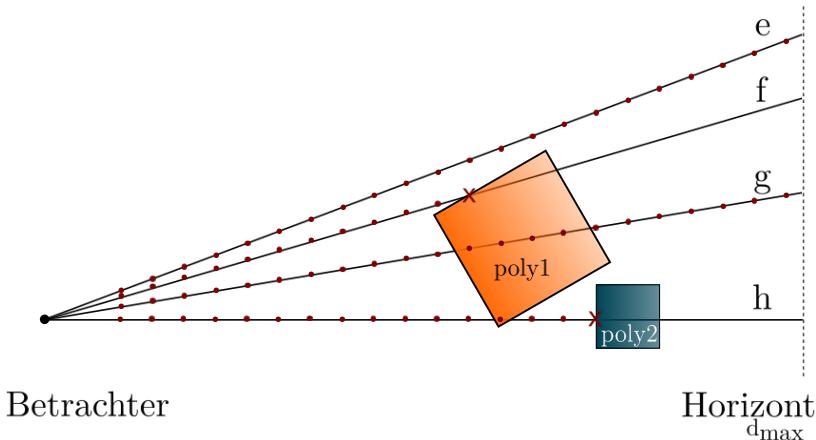


Abbildung 6.2.: Illustration des Ray Marching Verfahrens und dessen Problematiken.<sup>3</sup>

Abbildung 6.2 veranschaulicht diese Problematiken. Zu sehen sind vier Primärstrahlen,  $e, f, g$  und  $h$ , sowie zwei Objekte,  $poly1$  und  $poly2$ .

<sup>3</sup>Eigene Darstellung mittels Inkscape.

Bei den Strahlen  $e$  und  $f$  handelt es sich um “normale” Fälle: Der Strahl  $e$  verfehlt beide Objekte; das Ray Marching wird also nach Erreichen der maximalen Distanz  $d_{max}$  abgebrochen.  
Der Strahl  $f$  trifft das Objekt  $poly1$  nach 12 Schritten.

Die Strahlen  $g$  und  $h$  sind Spezialfälle: Der Strahl  $g$  geht durch das Objekt  $poly1$ , erfasst dieses wegen des gewählten fixen Abstandes der Abtastungspunkte ( $\Delta t$ ) nicht. So liegt ein Abtastungspunkt vor dem Objekt und der nächste Abtastungspunkt bereits in dem Objekt, was zu dem Verfehlten von diesem führt.

Der Strahl  $h$  liefert das Objekt  $poly2$ , obwohl er eigentlich das Objekt  $poly1$  liefern müsste. Das getroffene Objekt  $poly2$  dürfte so gar nicht zu sehen sein. Dieser Fehler tritt wiederum aufgrund des gewählten fixen Abstandes der Abtastpunkte ( $\Delta t$ ) auf.

Hart weist darauf hin, dass Ray Marching durch den möglichst geringen Abstand zwischen den Abtastungen entsprechend langsam und paralleles Abtasten praktisch unumgänglich ist [19, S. 528]. In der von Hart vorgestellten Technik des Sphere Tracings ist der Abstand zwischen den Abtastungen nicht konstant sondern variiert in Abhängigkeit der Geometrie [19, S. 538 bis 540].

Es wurden verschiedene Methoden entwickelt und untersucht um den entscheidenden Nachteil des fixen Abstandes der Abtastungspunkte zu verbessern. So haben zum Beispiel verschiedene Autoren eine Anpassung des Abstandes mittels Intervall-Arithmetik als Lösung der Problematik vorgeschlagen [23, 24].

### 6.2.2. Sphere Tracing

Das von Hart vorgeschlagene Sphere Tracing Verfahren ist ein Ray Tracing Verfahren für implizite Oberflächen. Es ist ebenfalls ein Ray Marching Verfahren. Jedoch wird die Distanz der Abtastungsschritte eines (Licht-) Strahles aufgrund einer Distanzfunktion bestimmt. Das Verfahren wurde erstmalig 1989 in „Ray Tracing Deterministic 3-D Fractals“ von Hart, Sandin und Kauffman vorgestellt [22].

Ein möglicher Algorithmus, wie Sphere Tracing umgesetzt werden kann, findet sich in Auflistung 6.2.

Bei Sphere Tracing werden die Schnittpunkte eines (Licht-) Strahles durch eine Folge von negativen Hüllkörpern (“unbounding volumes”) — bzw. in diesem Fall Kugeln (“unbounding spheres”) — beschrieben. Davon kommt die Bezeichnung Sphere Tracing [19, S. 530].

“Unbounding volumes” (zu Deutsch etwa “negativer Hüllkörper”) wird von den Autoren Hart, Sandin und Kauffman benutzt, um das Sphere Tracing Verfahren zu beschreiben und darzustellen. Der Term steht im Gegensatz zu dem gängigen Konzept des Hüllkörpers (“bounding volume”). Bei einem Hüllkörper wird ein Körper umschlossen. Der “negative Hüllkörper” umschliesst ein Stück des Raumes ohne dabei ein Objekt zu umschließen (dass heisst ohne ein gesuchtes Objekt zu “berühren”) [22, S. 291]. Die “negativen Hüllkörper” sind in Abbildung 6.3 als Kreisflächen dargestellt.

Für die Berechnung eines negativen Hüllkörpers sucht man den Abstand eines Objektes von einem Ausgangspunkt. Ist die kürzeste Distanz zwischen Ausgangspunkt und Objekt bekannt, kann diese als Radius einer Kugel angenommen werden.

Objekte sind bei dem genannten Verfahren durch Distanzfunktionen definiert. So ist z.B. eine Kugel als Punkt im Raum minus deren Radius definiert (siehe Gleichung 6.14). Der Betrachter kann dabei auch als Punkt im Raum angenommen werden. Somit sind alle Distanzen bekannt (siehe auch Distanzfelder, Abschnitt 6.1.3).

Eine Kugel dient als negativer Hüllkörper (“unbounding Volume”). Sie ist aber *nicht* Teil des Objektes und schneidet dieses auch nicht (ist also nicht  $\overset{\circ}{\mathbf{A}}$ ).

Nur der äusserste Punkt des Abstandes liegt genau auf der Oberfläche des Objektes ( $\partial \mathbf{A}$ ). Der Radius einer solchen Kugel wird durch Evaluation der Distanzfunktion eines abzutastenden Punktes im Raum bestimmt.

Hart, Sandin und Kauffman beschreiben in ihrer Arbeit „Ray Tracing Deterministic 3-D Fractals“ die Darstellung von Fraktalen im dreidimensionalen Raum. Es wird von einer Abschätzung der Distanz gesprochen, da diese für Fraktale nicht effizient berechnet werden kann [22, S. 291].

Bei "regulären" Objekten, wie z.B. einer Kugel, kann der zur Oberfläche am nächsten gelegene Punkt von einem beliebigen Punkt derselben Domäne exakt berechnet werden [19, S. 530]. Dies ist durch die implizite Gleichung 6.15 gegeben.

Gemäss [22, S. 291 - 292] geschieht die Verfolgung der Strahlen bei Sphere Tracing wie folgt: Ein Strahl wird vom Betrachter (Auge bzw. Lochkamera) durch die Bildebene zu einem Objekt geschossen. Dabei wird bei dem initialen Ausgangspunkt der Radius eines negativen Hüllkörpers in Form einer ersten Kugel berechnet, so wie oben beschrieben. Dies ist die Distanz, welche der Strahl in einem ersten Schritt effektiv zurücklegen wird. Von diesem Schnittpunkt aus wird erneut der Radius einer Kugel berechnet usw.

Dies geschieht so lange, bis der Strahl schliesslich von einem Schnittpunkt des negativen Hüllkörpers aus auf die Oberfläche eines Objektes trifft.

Ein weiteres Kriterium für den Abbruch ist eine vordefinierte maximale Distanz des Strahles ( $d_{\max}$ ). Ist diese erreicht und der Strahl verfehlt die Oberfläche des Objektes, wird abgebrochen. Somit ist auch ersichtlich, dass Sphere Tracing nicht die genannten Problematiken von Ray Marching aufweist (siehe Unterabschnitt 6.2.1).

Die folgenden Abbildungen 6.3 und 6.4 veranschaulichen das Sphere Tracing Verfahren.

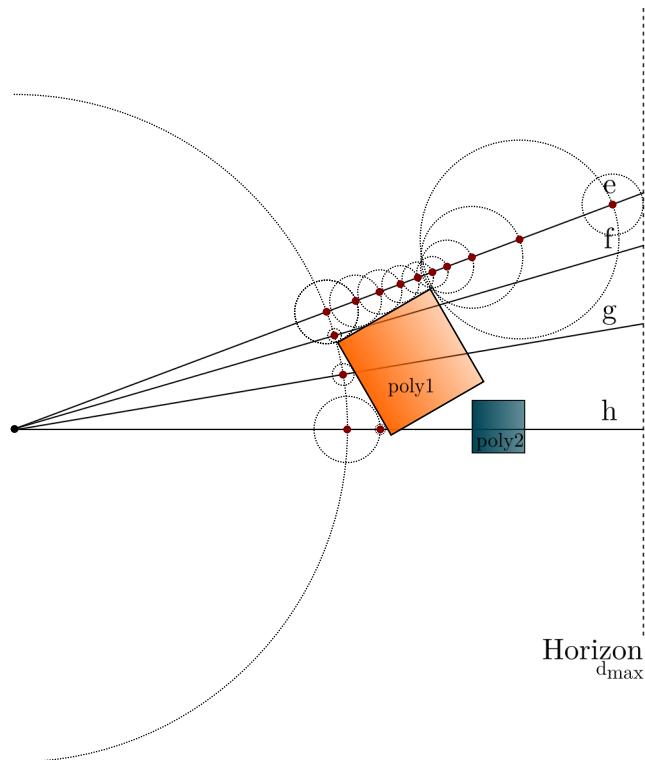


Abbildung 6.3.: Illustration des Sphere Tracing Verfahrens.<sup>4</sup>

---

<sup>4</sup>Eigene Darstellung mittels Inkscape.

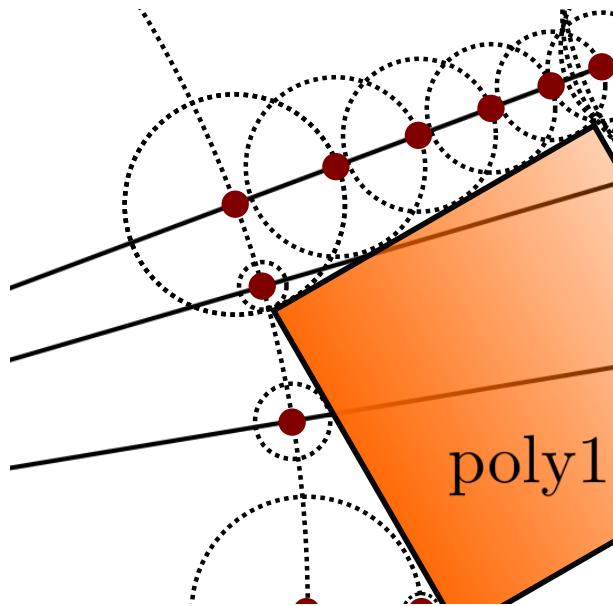


Abbildung 6.4.: Illustration des Sphere Tracing Verfahrens, Nahaufnahme.<sup>5</sup>

Ausgehend von der parametrischen Beschreibung eines (Licht-) Strahles (Gleichung 6.19), beschreiben Hart, Sandin und Kauffman die Richtung  $r_d$  eines Strahles als Einheitsvektor [22, S. 291]:

$$r_d = \frac{p_{x,y} - r_0}{|p_{x,y} - r_0|} \quad (6.23)$$

Dabei ist  $r_0$  der Ursprung eines Strahles und  $p_{x,y}$  ein Punkt der Bildebene.

Um nun den Schnittpunkt eines Strahles  $r_d$  mit der Oberfläche eines Objektes zu finden, muss Gleichung 6.20,  $F(t) = f \circ r = 0$ , gelöst werden. Dabei ist — wie oben definiert — die Funktion  $f(x)$  nun eine Distanzfunktion, wie zum Beispiel die geometrische Distanzfunktion zur Beschreibung einer Kugel (Gleichung 6.14).

Evaluiert man nun die Gleichung  $F(t)$  unter Anwendung der eben beschriebenen Verfolgung der Strahlen, findet man so die erste positive Nullstelle der Gleichung  $F(t)$ . Diese Nullstelle ist die Grenze der Folge von negativen Hüllkörpern (“unbounding spheres”), welche durch die rekursive Gleichung:

$$t_{i+1} = t_i + F(t_i) \quad (6.24)$$

definiert ist. Der Ursprungspunkt ist dabei als  $t_0$  definiert. Diese Folge konvergiert genau dann — und nur dann — wenn der Strahl auf die implizite Oberfläche eines Objektes trifft. Diese Folge bildet den Kern des Algorithmus zur Darstellung von geometrisch definierten, impliziten Oberflächen.

---

<sup>5</sup>Eigene Darstellung mittels Inkscape.

```

1 def sphere_trace():
2     ray_distance = 0
3     estimated_distance = 0
4     max_distance = 9001
5     convergence_precision = 0.000001
6
7     while ray_distance < max_distance:
8         # sd_sphere is a signed distance function defining the implicit surface
9         # cast_ray defines the ray equation given the current travelled /
10        # marched distance of the ray
11        estimated_distance = sd_sphere(cast_ray(ray_distance))
12
13        if estimated_distance < convergence_precision:
14            # the estimated distance is already smaller than the desired
15            # precision of the convergence, so return the distance the ray has
16            # travelled as we have an intersection
17            return ray_distance
18        # end if
19
20        ray_distance = ray_distance + estimated_distance
21    # done while
22
23    # When we reach this point, there was no intersection between the ray and a
24    # implicit surface, so simply return 0
25    return 0
26 # end def sphere_trace

```

Auflistung 6.2: Eine abstrakte Umsetzung des Sphere Tracings<sup>6</sup>.

---

Algorithmus in Pseudocode gemäss [19][S. 531, Fig. 1]

### 6.2.3. Operationen für implizite Oberflächen

Um mit impliziten Oberflächen nicht nur einfache Objekte wie beispielsweise eine Kugel darstellen zu können muss man diese auch transformieren können.

Wie Hart beschreibt, werden implizite Oberflächen durch die Invertierung des Raumes, in welchem sich eine Oberfläche befindet, transformiert [19, S. 543]. Der Raum, in dem sich eine implizite Oberfläche befindet, ist die Domäne der impliziten Funktion der Oberfläche.

Sei  $T(\mathbf{x})$  eine Transformation und  $f(\mathbf{x})$  eine Distanzfunktion, welche eine implizite Oberfläche definiert. Somit ist die transformierte implizite Oberfläche [19, S. 534]:

$$f(T^{-1}(\mathbf{x})) = 0 \quad (6.25)$$

Bei Transformationen handelt es sich um von dem Vorzeichen abhängige Distanzfunktionen (*signed distance functions*).

Es werden folgende Arten von Transformationen unterschieden [18, S. 14]:

- Distanz-Operationen  
Zum Beispiel Vereinigung, Subtraktion oder Intersektion.
- Domänen-Operationen  
Zum Beispiel Wiederholung, Rotation, Translation und Skalierung.

- Distanz-Deformationen  
Zum Beispiel Versatz (displacement) und Vermengung/Vermischung. (*blend*)
- Domänen-Deformationen  
Zum Beispiel “Verwindung” (*twist*) und Biegung (*bend*).

## Isometrien

Nicht alle Transformationen erhalten dabei die Distanz, welche die Distanzfunktion der transformierten Oberfläche zurückgeben würde. In solch einem Falle ist die zurückgegebene Distanz nicht die Distanz eines beliebigen Punktes im Raum zu dem ihm nächsten Punkt einer impliziten Oberfläche.

Transformationen, welche die Distanz hingegen erhalten, bezeichnet Hart als *Isometrien* [19, S. 534]. Dazu zählen Rotationen, Translationen aber auch Reflexionen.

Ist  $\mathbf{I}$  eine Isometrie, dann benötigt die zurückgegebene Distanz der Distanzfunktion  $f(\mathbf{x})$  *keine Anpassung*.

$$d(\mathbf{x}, \mathbf{I} \circ f^{-1}(0)) = d(\mathbf{I}^{-1}(\mathbf{x}), f^{-1}(0)) \quad (6.26)$$

Dabei ist  $\mathbf{I}$  eine Isometrie und  $f^{-1}(0)$  eine implizite Oberfläche.

## Uniforme Skalierung

Eine Skalierung bewahrt keine Distanzen. Somit muss die zurückgegebene Distanz einer Distanzfunktion entsprechend angepasst werden.

Hart gibt die uniforme Skalierung als Transformation  $\mathbf{S}(\mathbf{x})$  der folgenden Form an [19, S. 534]:

$$\mathbf{S}(\mathbf{x}) = s \cdot \mathbf{x} \quad (6.27)$$

Dabei ist  $s$  der Skalierungsfaktor. Die Invertierung der Skalierung ist gegeben als [19, S. 534]:

$$\mathbf{S}^{-1}(\mathbf{x}) = \frac{1}{s} \cdot \mathbf{x} \quad (6.28)$$

Somit ist die Distanz zu der skalierten impliziten Oberfläche [19, S. 534]:

$$d(\mathbf{x}, \mathbf{S}(f^{-1}(0))) = s \cdot d(\mathbf{S}^{-1}(\mathbf{x}), f^{-1}(0)) \quad (6.29)$$

Dabei wird die von der Distanzfunktion der skalierten impliziten Oberfläche zurückgegebene Distanz mit dem Skalierungsfaktor  $s$  multipliziert, was die eigentliche Information der Distanz erhält und die Skalierung somit isometrisch macht.

## “Verwindung” (*Twist*)

Gemäß Hart werden bei der “Verwindung” (*Twist*) einer impliziten Oberfläche zwei Achsen (z.B.  $x$  und  $y$ ) anhand einer linearen Funktion  $a(\cdot)$  in Abhängigkeit der dritten Achse (z.B.  $z$ ) rotiert [19, S. 543]:

$$\text{twist}(\mathbf{x}) = \begin{pmatrix} x \cdot \cos a(z) - y \cdot \sin a(z), \\ x \cdot \sin a(z) + y \cdot \cos a(z), \\ z \end{pmatrix} \quad (6.30)$$

## Vereinigung

Die Vereinigung zweier impliziter Oberflächen  $A$  und  $B$  wird von Hart als minimale Distanz der jeweiligen, vom Vorzeichen abhängigen Distanzfunktion  $f_A$  respektive  $f_B$  definiert [19, S. 531 bis 532]:

$$d(\mathbf{x}, A \cup B) = \min(f_A(\mathbf{x}), f_B(\mathbf{x})) \quad (6.31)$$

Dabei  $\mathbf{x}$  den abzutastenden Punkt im Raum darstellt.

Wie Hart schreibt, ist die Distanz zu einer Menge von Objekten die kürzeste der Distanzen zu jedem der zusammengesetzten Objekte [19, S. 531 bis 532].

Somit erlaubt die Vereinigung die Kombination von mehreren impliziten Oberflächen, ohne dass diese miteinander in Kontakt stehen. So kann beispielsweise eine komplexe Szene modelliert werden.

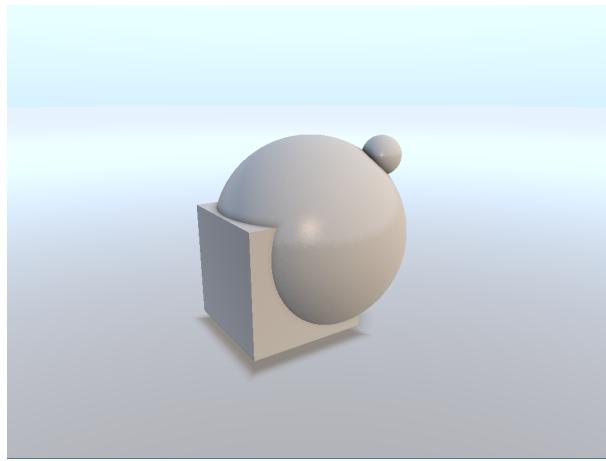


Abbildung 6.5.: Vereinigung von zwei Kugeln und einem Würfel. Bei dem Untergrund der Szene handelt es sich um eine Ebene, welche mit dem Rest der Szene vereinigt wird.<sup>7</sup>

## Subtraktion

Für die Subtraktion wird die Distanz zum Komplement eines Objektes  $A$  verwendet. Dabei wird die Eigenschaft der Abhängigkeit vom Vorzeichen der entsprechenden Distanzfunktionen genutzt [19, S. 532]:

$$d(\mathbf{x}, \mathbb{R}^3 \setminus A) = -f_A(\mathbf{x}) \quad (6.32)$$

Somit kann die Subtraktion zweier impliziter Oberflächen  $A$  und  $B$  gemäss Hart als Intersektion eines Objektes  $A$  mit der Subtraktion des Raumes bzw. der Domäne mit einem Objekt  $B$  angesehen werden, daher folgt [19, S. 532]:

$$d(\mathbf{x}, A - B) = A \cap (\mathbb{R}^3 \setminus B) \quad (6.33)$$

$$\geq \max(f_A(\mathbf{x}), -f_B(\mathbf{x})) \quad (6.34)$$

Dabei stellt  $\mathbf{x}$  den abzutastenden Punkt im Raum dar.

---

<sup>7</sup>Eigene Darstellung

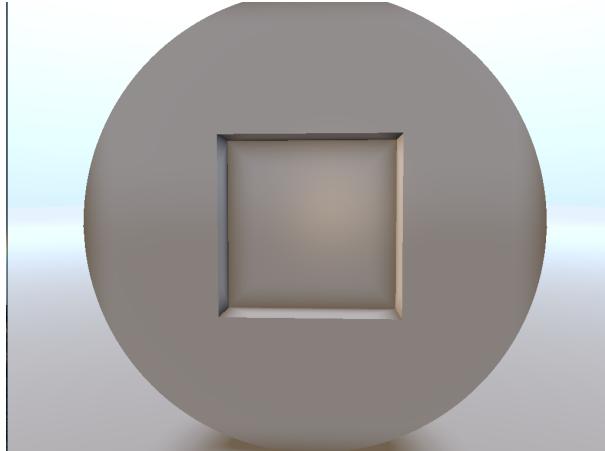


Abbildung 6.6.: Mehrfache Subtraktion: In einem ersten Schritt wird in der Hälfte der Kugel ein Würfel von der Kugel subtrahiert. Der Würfel hat dieselbe Höhe wie die Kugel. In einem zweiten Schritt wird ein wesentlich kleinerer Würfel von der Halbkugel subtrahiert.<sup>8</sup>

### Intersektion

Die Intersektion zweier impliziter Oberflächen  $A$  und  $B$  wird von [19] als minimale Distanz der jeweiligen vorzeichenabhängigen Distanzfunktion  $f_A$  respektive  $f_B$  definiert [19, S. 532]:

$$d(\mathbf{x}, A \cap B) \geq \max(f_A(\mathbf{x}), f_B(\mathbf{x})) \quad (6.35)$$

Dabei stellt  $\mathbf{x}$  den abzutastenden Punkt im Raum dar.

#### 6.2.4. Primitive

Hart führt in seiner Arbeit einige (geometrische) Primitive auf, welche nachfolgend erläutert werden [19, S. 540ff].

### Ebene

Die vorzeichenabhängige Distanz zur einer Ebene  $P$  mit einer Einheitsnormalen  $\mathbf{n}$  und Schnittpunkt  $\mathbf{n} \cdot \mathbf{r}$  wird folgend definiert:

$$d(\mathbf{x}, P) = \mathbf{x} \cdot \mathbf{n} - r \quad (6.36)$$

Dabei ist  $r$  die relative Positionierung der Ebene im Raum.

---

<sup>8</sup>Eigene Darstellung

## Kugel

Eine Kugel ist als eine Menge von Punkten (Locus) in fixem Abstand eines gegebenen Punktes. Die von dem Vorzeichen abhängige Distanz zu einer Kugel  $S$ , ausgehend vom Ursprung, ist wie folgt:

$$d(\mathbf{x}, S) = \|\mathbf{x}\| - r \quad (6.37)$$

Dabei stellt  $r$  den Radius der Kugel dar.

## Zylinder

Die Distanz zu einem um die Z-Achse zentrierten Zylinder mit Einheitsradius wird durch Projektion auf die XY-Ebene und durch Messung der Distanz zum Einheitskreis berechnet:

$$d(\mathbf{x}, Cyl) = \|(x, y)\| - r \quad (6.38)$$

Dabei stellt  $r$  den Radius des Zylinders und  $\mathbf{x}$  den Punkt  $(x, y, z)$  im Raum dar.

## Kegel

Die Distanz zu einem Kegel, welcher am Ursprung zentriert und entlang der Z-Achse orientiert ist, wird wie folgt berechnet:

$$d(\mathbf{x}, Cone) = \|(x, y)\| \cdot \cos(\phi) - |z| \cdot \sin(\phi) \quad (6.39)$$

Dabei stellt  $\phi$  den Winkel zur Z-Achse dar.

## Torus

Beim Torus handelt es sich um das Produkt zweier Kreise, sowie den Abstand der Kreise:

$$d(\mathbf{x}, T) = \|(\|(x, y)\| - R, z)\| - r \quad (6.40)$$

Dabei stellt  $R$  den äusseren Radius und  $r$  den inneren Radius des Torus dar. Der Torus ist am Ursprung zentriert und dreht sich um die Z-Achse.

## 6.3. Rendering von impliziten Oberflächen

Nachdem in den vorherigen Abschnitten Methoden zur Modellierung von impliziten Oberflächen behandelt wurden, beschreibt das folgende Kapitel wie implizite Oberflächen gerendert werden können. Foley beschreibt Rendering als Erstellung von Bildern ausgehend von Modellen [5, S. 606].

### 6.3.1. Beleuchtungsmodell

Um implizite Oberflächen darstellen zu können, muss ein Beleuchtungsmodell gewählt werden. Andernfalls wäre das dargestellte Bild nur schwarz. Zur Vereinfachung wird im Rahmen dieser Projektarbeit das in Unterabschnitt 5.1.1 vorgestellte Phong-Beleuchtungsmodell verwendet.

Daher wird die resultierende Farbe eines Punktes im Raum  $I(\mathbf{x})$  aus ambienten, diffusen und reflektierenden Anteilen berechnet:

$$I(\mathbf{x}) = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \quad (6.41)$$

Wie bereits zuvor in Unterabschnitt 5.1.1 erwähnt, wird der emissive Term bewusst weggelassen, da keine emissiven Materialen dargestellt werden sollen. Als Lichtquelle wird eine einzelne direktionale Lichtquelle gewählt. Analog zu den vorherigen Abschnitten ist  $\mathbf{x}$  in den folgenden Abschnitten ein Punkt  $(x, y, z)$  auf einer impliziten Oberfläche  $A$ .

Der *ambiente Anteil*  $I_{\text{ambient}}$  ergibt sich dann wie folgt:

$$I_{\text{ambient}} = k_{\text{ambient}}(\mathbf{x}) \cdot L_{\text{ambient}} \quad (6.42)$$

Dabei ist  $k_{\text{ambient}}(\mathbf{x})$  der ambienten Faktor des Punktes  $\mathbf{x}$  und  $L_{\text{ambient}}$  die Farbe des eingehenden ambienten Lichtes [10, S. 723, 14, Kapitel 5, Abschnitt 5.2.1].

Der *diffuse Anteil*  $I_{\text{diffuse}}$  ergibt sich wie folgt:

$$I_{\text{diffuse}} = k_{\text{diffuse}}(\mathbf{x}) \cdot L_{\text{diffuse}} \cdot \max(\mathbf{n} \cdot \mathbf{l}, 0) \quad (6.43)$$

Dabei ist  $k_{\text{diffuse}}(\mathbf{x})$  der diffusen Faktor am Punkt  $\mathbf{x}$  und  $L_{\text{diffuse}}$  die Farbe des eingehenden diffusen Lichtes. Die Richtung der Lichtquelle, ausgehend von Punkt  $\mathbf{x}$ , ergibt sich durch das Produkt zwischen der Einheitsnormalen  $\mathbf{n}$  der Oberfläche an dem Punkt und der Richtung der Lichtquelle  $\mathbf{l}$  [10, S. 724, 14, Kapitel 5, Abschnitt 5.2.1].

Der *reflektierende Anteil*  $I_{\text{specular}}$  ergibt sich wie folgt:

$$I_{\text{specular}} = n_{\text{facing}} \cdot k_{\text{specular}}(\mathbf{x}) \cdot L_{\text{specular}} \cdot \max(\mathbf{n} \cdot \mathbf{h}, 0)^{k_e} \quad (6.44)$$

Dabei ist  $k_{\text{specular}}(\mathbf{x})$  der reflektierenden Faktor des Punktes  $\mathbf{x}$  und  $L_{\text{specular}}$  die Farbe des eingehenden Lichtes. Bei  $\mathbf{h}$  handelt es sich um einen Einheitsvektor, welcher in der Hälfte zwischen der Blickrichtung des Betrachters bzw. der Kamera ( $\vec{V}$ ) und  $\mathbf{l}$  der Richtung der Lichtquelle ausgehend von dem Punkt  $\mathbf{x}$  ist [10, S. 731]. Der Exponent  $k_e$  gibt an, wie rau bzw. wie spiegelnd die Oberfläche am Punkt  $\mathbf{x}$  ist [14, Kapitel 5, Abschnitt 5.2.1]. Der Faktor  $n_{\text{facing}}$  definiert, ob die Oberfläche überhaupt einen reflektierenden Anteil hat [14, Kapitel 5, Abschnitt 5.2.1]:

$$n_{\text{facing}} = \begin{cases} 0 & \text{if } \mathbf{n} \cdot \mathbf{l} \leq 0 \\ 1 & \text{if } \mathbf{n} \cdot \mathbf{l} > 0 \end{cases} \quad (6.45)$$

Für die Berechnung der Intensität des Lichtes bzw. der Farbe einer Oberfläche wird die Normale der Oberfläche benötigt. Gemäß Hart, Sandin und Kauffman kann diese mittels des Gradienten des Distanzfeldes eines Punktes einer impliziten Oberfläche berechnet werden [22, S. 292 bis 293]:

$$\mathbf{n}_x = f(x + \varepsilon, y, z) - f(x - \varepsilon, y, z) \quad (6.46)$$

$$\mathbf{n}_y = f(x, y + \varepsilon, z) - f(x, y - \varepsilon, z) \quad (6.47)$$

$$\mathbf{n}_z = f(x, y, z + \varepsilon) - f(x, y, z - \varepsilon) \quad (6.48)$$

$$(6.49)$$

Dabei ist  $\mathbf{n} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}$  die Normale der Oberfläche in Form eines Vektors und  $f$  eine Distanzfunktion [22, S. 292 bis 293, 18, S. 13].

Der Gradient einer Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  wird wie folgt berechnet:

$$\text{grad}(f) = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad (6.50)$$

$$\text{grad}(f) = f_x \mathbf{i} + f_y \mathbf{j} + f_z \mathbf{k} \quad (6.51)$$

$$(6.52)$$

Dabei sind  $\mathbf{i}$ ,  $\mathbf{j}$  und  $\mathbf{k}$  Vektoren der Form  $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ .

Hart, Sandin und Kauffman geben  $\varepsilon$  als die minimale Inkrementation eines (Licht-) Strahles an und definieren diese als Funktion zur Bestimmung der Sichtbarkeit  $\Gamma_{\alpha, \delta}$  [22, S. 293]:

$$\Gamma(d) = \alpha d^\delta \quad (6.53)$$

in Abhängigkeit der euklidischen Distanz  $d$  des Betrachters respektive der Kamera zur aktuellen Position des (Licht-) Strahles:

$$d = |r_n - r_0| \quad (6.54)$$

Dabei ist  $\delta$  ein so genannter “depth-cueing”-Exponent (“depth-cueing” oder auch “foldback”: “a process for returning a signal to a performer instantly” [25]) und  $\alpha$  ein empirischer Anteil, welcher die Tiefenauflösung des Objektes definiert. Details dazu finden sich unter [22, S. 293, Abschnitt 4.2 — “Clarity”].

Es folgt also:

$$\varepsilon = \Gamma_{\alpha, \delta}(|r_n - r_0|) \quad (6.55)$$

Die Korrektheit der Berechnung der Normalen  $\mathbf{n}$  hängt von der Grösse von  $\varepsilon$  ab. Daher wird für gewöhnlich ein kleiner Wert für  $\varepsilon$  gewählt [22, S. 293].

Die Normale der Oberfläche sollte schliesslich noch normalisiert werden.

Liefert die oben genannte Gradienten, bestehend aus 6 Punkten, eine zu geringe Genauigkeit, so kann diese gemäss Hart, Sandin und Kauffman erweitert werden [22, S. 293].

Die Erweiterung erfolgt durch Hinzunahme von Punkten, welche eine gemeinsame Kante haben. Dies erzeugt eine Gradienten bestehend aus 18 Punkten. Werden noch die Punkte hinzugenommen, welche gemeinsame Eckpunkte haben, so ergibt sich eine Gradienten bestehend aus 26 Punkten [22, S. 293].

### 6.3.2. Rendering

Um implizite Oberflächen zu rendern, werden die in Unterabschnitt 6.2.4 angegebenen Primitiven verwendet.

Zum eigentlichen Rendern wird der Algorithmus in Auflistung 6.2 mit dem unter Unterabschnitt 6.3.1 angegebenen Beleuchtungsmodell angewendet.

### 6.3.3. Schatten

Sofern im Text nicht anders vermerkt, basiert folgender Abschnitt auf [26, S. 7].

Mittels Sphere Tracing können Schatten analog den vom Ray Tracing bekannten Verfahren gewonnen werden. Dazu werden Schatten-Fühler oder auch Schatten-Strahlen (“*shadow rays*”) mit Sphere Tracing abgetastet. Man bildet also eine Folge von negativen Hüllkörpern (“*unbounding volumes*”) bzw. Kugeln (“*unbounding spheres*”) pro Lichtquelle in Richtung dieser ausgehend von einem Punkt einer Oberfläche. Die Folge wird so lange fortgesetzt, bis eine Intersektion stattfindet oder bis eine definierte maximale Distanz erreicht wurde.

Schatten-Fühler werden parametrisch wie folgt beschrieben:

$$r_s(t) = \mathbf{x} + t \cdot \mathbf{r}_l \quad (6.56)$$

Dabei ist  $r_s(t)$  der Ursprung des Schatten-Fühlers am Punkt  $\mathbf{x}$  einer impliziten Oberfläche,  $\mathbf{r}_l$  die Richtung des Schatten-Fühlers in Form eines Einheitsvektors und  $t$  die zurückgelegte Distanz des Strahles.

Der Algorithmus in Auflistung 6.3 ist dem des Sphere Tracings (siehe Auflistung 6.2) sehr ähnlich, der Rückgabewert ist jedoch völlig verschieden. Der Algorithmus gibt den Wert 1 zurück, wenn nach Erreichung der maximalen Distanz keine Intersektion zwischen dem Schatten-Führer und einer Oberfläche statt fand. Der Wert 0 wird zurückgegeben, wenn der Schatten-Führer auf eine Oberfläche getroffen ist und der Punkt  $\mathbf{x}$  einer impliziten Oberfläche sich daher im Schatten befindet.

```

1 def calc_shadows():
2     min_distance = 0.01
3     max_distance = 9001
4     shadow_ray_distance = min_distance
5     estimated_distance = 0
6     convergence_precision = 0.000001
7
8     while shadow_ray_distance < max_distance:
9         # sd_sphere is a signed distance function defining the implicit surface
10        # cast_ray defines the ray equation given the current travelled /
11        # marched distance of the ray
12        estimated_distance = sd_sphere(cast_ray(shadow_ray_distance))
13
14        if estimated_distance < convergence_precision:
15            # the estimated distance is already smaller than the desired
16            # precision of the convergence, so return zero (0) as we
17            # have an intersection and therefore shadows
18            return 0
19        # end if
20
21        shadow_ray_distance = shadow_ray_distance + estimated_distance
22    # done while
23
24    # When we reach this point, there was no intersection between the ray and a
25    # implicit surface, so no shadows, so simply return 1
26    return 1
27 # end def calc_shadows

```

Auflistung 6.3: Algorithmus zur Berechnung von Schatten.

Um zu verhindern, dass ein Punkt  $x$  einer impliziten Oberfläche sich selbst verdeckt (sich also quasi selbst Schatten spendet), wird die initial zurückgelegte Distanz ( $shadow\_ray\_distance$ ) auf einen Minimalwert ( $min\_distance$ ) gesetzt. Dieser Wert sollte jedoch wesentlich grösser als die gewünschte Präzision ( $convergence\_precision$ ) sein, da ansonsten die Bedingung ( $estimated\_distance < convergence\_precision$ ) ggf. initial erfüllt und sich der Punkt  $x$  daher immer im Schatten befinden würde.

Reiner, Mückl und Dachsbacher geben zudem an, wie weiche Schatten mittels Sphere Tracing relativ einfach dargestellt werden können [26, S. 7]. Während der Expansion anhand des Schatten-Fühlers wird die minimale, evaluierte Distanz  $d_{\min}$  zu einem beliebigen Objekt gespeichert. Es wird dabei angenommen, dass sich ein Punkt  $x$  einer impliziten Oberfläche für schmale Distanzen  $0 < d_{\min} < d$  im Schatten befindet. Somit ergibt schliesslich das Verhältnis der minimalen Distanz zu der aktuellen Distanz einen Schatten- bzw. Penumbra-Faktor:  $\text{shadow} = \frac{d_{\min}}{d}$ , wobei  $\text{shadow} \in (0, 1)$ . Mit wachsender Distanz  $d$  wächst auch die so genannte Penumbra-Region.

Durch Speicherung der minimalen, evaluierten Distanz ist es möglich, die Distanz des Schatten-Fühlers zu der ihn umgebenden Geometrie zu untersuchen. Damit werden die Penumbra-Region sowie weiche Schatten bestimmt. Befindet sich der Schatten-Füher in der Nähe einer Oberfläche ohne diese zu scheiden, ist die minimale Distanz  $d_{\min}$  sehr gering. In diesem Fall, kann angenommen werden, dass sich der Ursprungspunkt  $x$  in einer Penumbra-Region befindet. Dies wiederum wirkt sich auf die Schattierung des Punktes aus: Je knapper der Schatten-Füher eine Oberfläche nicht geschnitten hat, desto stärker wird der Punkt schattiert. Zusätzlich wird die Distanz des Ursprungspunktes einbezogen. Je geringer diese ist, desto stärker wird der Punkt schattiert.

Durch Hinzufügen eines Skalierungsfaktors kann ein Schattenwurf härter oder weicher gezeichnet werden.

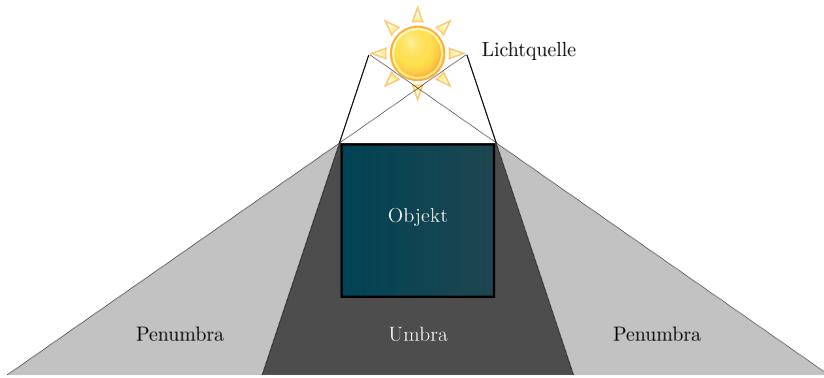


Abbildung 6.7.: Illustration eines Schattenwurfs anhand einer Lichtquelle und einem Objekt. Im Bild sind die Umbra- und Penumbra-Regionen erkennbar.<sup>9</sup>

```

1 def calc_soft_shadows():
2     min_distance = 0.01
3     max_distance = 9001
4     shadow_ray_distance = min_distance
5     estimated_distance = 0
6     convergence_precision = 0.000001
7     shadow = 1.0
8     scale = 32.0
9
10    while shadow_ray_distance < max_distance:
11        # sd_sphere is a signed distance function defining the implicit
12        # surface, # cast_ray defines the ray equation given the current ←
13        # travelled /
14        # marched distance of the ray
15        estimated_distance = sd_sphere(cast_ray(shadow_ray_distance))
16
17        if estimated_distance < convergence_precision:
18            # the estimated distance is already smaller than the desired
19            # precision of the convergence, so return zero (0) as we
20            # have an intersection and therefore 'full' shadow
21            return 0
22        # end if
23
24        penumbra_factor = estimated_distance / shadow_ray_distance
25        shadow = min(shadow, scale * penumbra_factor)
26        shadow_ray_distance = shadow_ray_distance + estimated_distance
27    # done while
28
29    # When we reach this point, there was no full intersection between
30    # the ray and a implicit surface, so not entirely shadowed, so
31    # return current shadow value
32    return shadow
33 # end calc_soft_shadows

```

Auflistung 6.4: Algorithmus zur Berechnung von weichen Schatten.

---

<sup>9</sup>Eigene Darstellung mittels Inkscape, angelehnt an [10, S. 772].

# 7. Prototyp

Um das Sphere Tracing Verfahren nicht nur theoretisch zu betrachten, wurde im Rahmen dieser Projektarbeit ein Prototyp erstellt. Dieser wurde in C++ Version 11 und OpenGL umgesetzt und basiert auf der GLFW-Bibliothek. Um allfällige OpenGL-Erweiterungen (Extensions) nicht selbst verwalten zu müssen, wird GLEW eingesetzt. Als Buildsystem kommt CMake, als Compiler GCC zum Einsatz. Um automatisch Shader erkennen und laden zu können (Dateien mit Dateiendung .vs bzw. .fs), werden die Bibliotheken “system”, “filesystem” sowie “regex” der Boost-Bibliothek benutzt.

Details der einzelnen Komponenten finden sich in der nachfolgenden Tabelle.

Komponente	Version	Beschreibung	Verweise
C++	11	Objektorientierte Programmiersprache	<sup>1</sup>
OpenGL	4.5	Plattformunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafikanwendungen [27].	<sup>2</sup>
GLFW	3.1.2	OpenGL-Bibliothek, welche die Erstellung und Verwaltung von Fenstern sowie OpenGL-Kontexte vereinfacht [28].	<sup>3</sup>
GLEW	1.13	OpenGL Extension Wrangler. Bibliothek zum Abfragen und Laden von OpenGL-Erweiterungen (Extensions) [29].	<sup>4</sup>
CMake	3.3.2	Software zur Verwaltung von Build-Prozessen von Software	<sup>5</sup>
GCC	5.2	GNU Compiler Collection. Compiler-System des GNU-Projektes	<sup>6</sup>
Boost	1.59.0	Freie Bibliothek bestehend aus einer Vielzahl von Bibliotheken, die den unterschiedlichsten Aufgaben von Algorithmen auf Graphen über Metaprogrammierung bis hin zu Speicherverwaltung dienen [30].	<sup>7</sup>

## 7.1. Architektur

Die Architektur des Prototypen wurde bewusst einfach gehalten und ist in Abbildung 7.1 ersichtlich.

<sup>7</sup>[http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=50372](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372)

<sup>7</sup><https://www.opengl.org/registry/doc/glspec45.core.pdf>

<sup>7</sup><http://www.glfw.org>

<sup>7</sup><http://glew.sourceforge.net>

<sup>7</sup><https://www.cmake.org>

<sup>7</sup><http://gcc.gnu.org>

<sup>7</sup><http://www.boost.org>

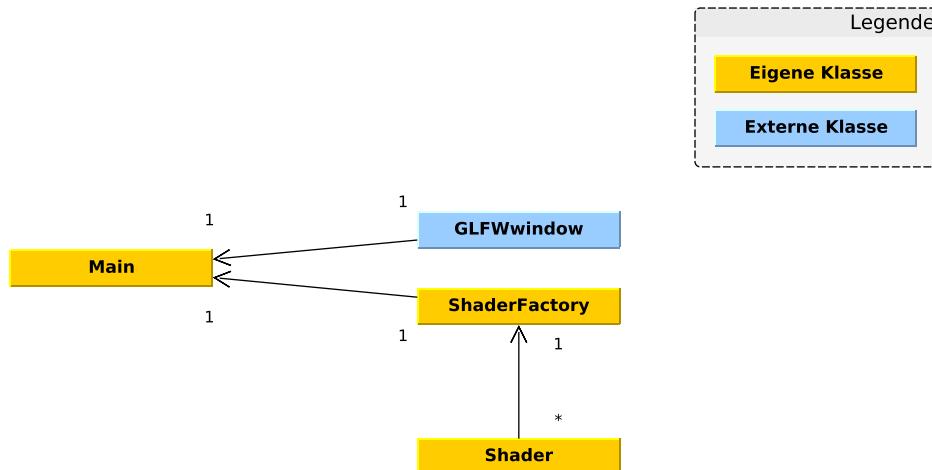


Abbildung 7.1.: Architektur des Prototypen<sup>8</sup>

### 7.1.1. Programmablauf

Wie aus Abbildung 7.1 ersichtlich, besteht die Applikation hauptsächlich aus der Hauptfunktion „Main“. Diese benützt GLEW um OpenGL zu initialisieren. Mittels GLFW erstellt sie ein Fenster sowie einen OpenGL-Kontext. Danach wird eine Instanz der Klasse *ShaderFactory* erstellt, welche ihrerseits alle verfügbaren GLSL-Shader aus einem gegebenen Verzeichnis lädt. Bei diesem Prototypen kommt jedoch nur ein einziger Shader zum Einsatz. Er bestehent aus einem *Vertex-* und einem *Fragment-*Teil.

Die Applikation läuft danach in einer Endlosschleife, hört dabei aber auf Events in Form von Keyboard-Eingaben. Dadurch kann die Applikation jederzeit mit der Abbruch-Taste (ESC, Escape) beendet werden.

Die hauptsächliche Aufgabe der Applikation besteht aus dem Laden und Ausführen eines Vertex- sowie Fragment-Shaders im Rendering-Teil. Dazu wird via OpenGL ein Rechteck über die verfügbare Fläche des Fensters ausgegeben. Der Vertex-Shader adressiert schliesslich das von OpenGL gezeichnete Rechteck. Das eigentliche Rendering von impliziten Oberflächen geschieht schliesslich im Fragment-Shader. Dies ist in der untenstehenden Abbildung 7.2 verdeutlicht.

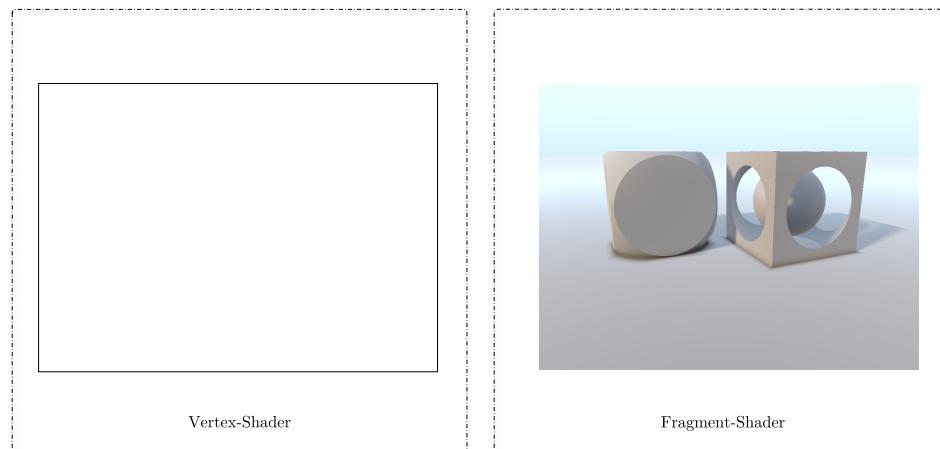


Abbildung 7.2.: Bildliche Darstellung der Funktionsweise von Vertex- und Fragment-Shader der Applikation<sup>9</sup>

<sup>8</sup>Eigene Darstellung mittels yEd.

## 7.2. Umsetzung

Nachfolgend werden die gemäss Abschnitt 6.2 umgesetzten Konzepte genauer beschrieben. Es handelt sich dabei um Ausschnitte des umgesetzten Fragment-Shaders.

Das eigentliche Sphere Tracing geschieht in der Funktion *castRay*. Diese hat als Parameter den Ursprung eines Strahles (*vec3 rayOrigin*), die Richtung eines Strahles (*vec3 rayDirection*), die maximale Distanz (*float maxDistance*), welche berechnet werden soll, die Präzision (*float precision*) sowie die Anzahl Durchgänge (*int steps*).

Ein Vergleich der letzten drei Parameter — die maximale Distanz, die Präzision sowie die Anzahl Durchgänge — findet sich in den Tabellen 7.1, 7.2 und 7.3.

Maschinell bedingt kommen bei den Berechnungen Fliesskommazahlen zum Einsatz, da die Resultate möglichst genau sein sollen. Aufgrund der endlichen Mantisse bei der Darstellung von Fliesskommazahlen lassen sich diese auf einem Computer jedoch nicht beliebig genau darstellen und damit berechnen. Rundung und damit verbundene Rundungsfehler sind unvermeidlich. Um dennoch eine gute Annäherung an Null-Werte zu erhalten, wird ein Parameter ( $\varepsilon$ ) zur Steuerung der minimalen Distanz zu einer Oberfläche verwendet. Dieser kommt bei den Funktionen *render*, *castRay* sowie *calcShadows* in Form der Variablen *minimalDistance* bzw. *precision* zum Einsatz.

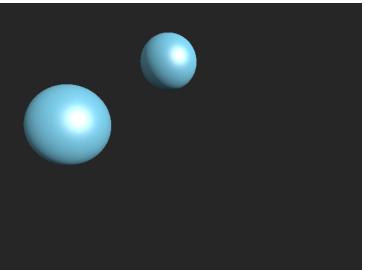
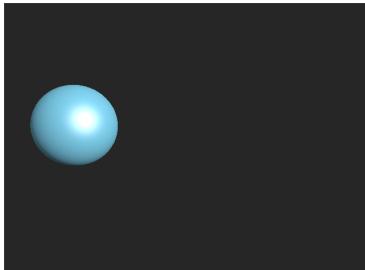
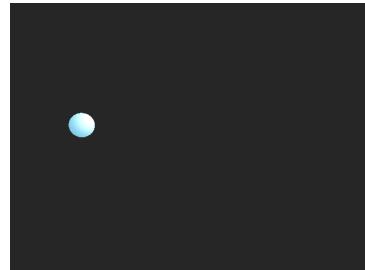
```
1 // Casts a ray from given origin in given direction. Stops at given
2 // maximal distance and after given amount of steps. Maintains given
3 // precision.
4 vec2 castRay(in vec3 rayOrigin, in vec3 rayDirection, in float maxDistance, in float←
    precision, in int steps)
5 {
6     float latest = precision * 2.0;
7     float distance = 0.0;
8     float type = -1.0;
9     vec2 res = vec2(-1.0, -1.0);
10
11    for(int i = 0; i < steps; i++) {
12        if (abs(latest) < precision || distance > maxDistance) {
13            continue;
14        }
15
16        vec2 result = scene(rayOrigin + rayDirection * distance);
17
18        latest = result.x;
19        type = result.y;
20        distance += latest;
21    }
22
23    if (distance < maxDistance) {
24        res = vec2(distance, type);
25    }
26
27    return res;
28 }
```

Auflistung 7.1: Umsetzung des Sphere Tracings in GLSL.

---

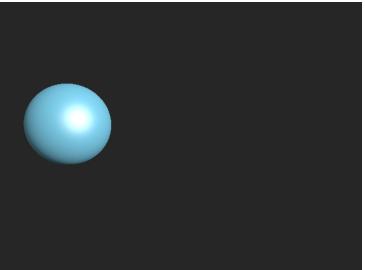
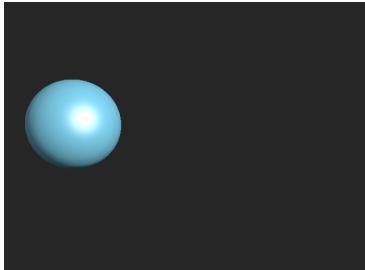
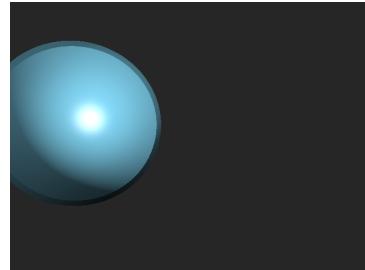
<sup>9</sup>Eigene Darstellung mittels yEd.

Tabelle 7.1.: Vergleich des Distanz-Parameters anhand einer Beispielszene.

Distanz: 100.0	Distanz: 9.0	Distanz: 6.3
		

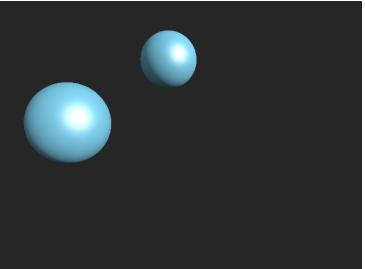
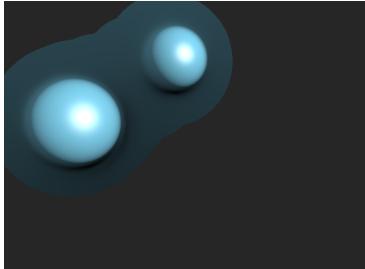
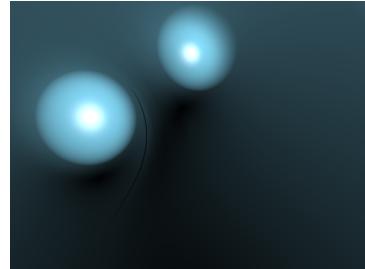
Alle in der Szene definierten Objekte sind sichtbar.  
Es ist nur noch das zum Betrachter näher liegende Objekt sichtbar.  
Es ist nur noch das zum Betrachter näher liegende Objekt sichtbar, jedoch nicht mehr vollständig.

Tabelle 7.2.: Vergleich des Präzisions-Parameters anhand einer Beispielszene.

Präzision: 0.00001	Präzision: 0.1	Präzision: 1.0
		

Die Kugel weist keinerlei sichtbare Abstufungen auf.  
Die Kugel weist am unteren linken Rand sichtbare Farbabstufungen auf.  
Die Kugel wird nicht mehr korrekt dargestellt.

Tabelle 7.3.: Vergleich des Parameters zur Bestimmung der Anzahl der Durchgänge anhand einer Beispielszene.

Anzahl Schritte: 100	Anzahl Schritte: 10	Anzahl Schritte: 5
		

Alle in der Szene definierten Objekte sind korrekt sichtbar.  
Die Grenze zwischen den in der Szene definierten Objekten ist bereits nicht mehr eindeutig.  
Die Szene ist als solche nicht mehr erkennbar, es erfolgt keine klare Trennung zwischen den einzelnen Objekten.

Von den unter Unterabschnitt 6.2.3 beschriebenen Operationen wurden die Operationen *Vereinigung*, *Subtraktion* sowie *Intersektion* umgesetzt.

```

1 // Returns the signed distance for a subtraction of given signed
2 // distance a to signed distance b.
3 float subtract(float a, float b)
4 {
5     return max(-b, a);
6 }
7
8 // Returns the signed distance for a merge of given signed
9 // distance a and signed distance b.
10 vec2 merge(vec2 a, vec2 b)
11 {
12     return min(a, b);
13 }
14
15 // Returns the signed distance for a intersection of given signed
16 // distance a and signed distance b.
17 float intersect(float a, float b)
18 {
19     return (a > b) ?a : b;
20 }
```

Auflistung 7.2: Umsetzung der Operationen *Vereinigung*, *Subtraktion* sowie *Intersektion* für implizite Oberflächen in GLSL.

Von den unter Unterabschnitt 6.2.4 beschriebenen Primitiven wurden die Primitiven *Ebene* sowie *Kugel* umgesetzt.

```

1 // Returns the signed distance to a plane for the given position.
2 float plane(vec3 position)
3 {
4     return position.y;
5 }
6
7 // Returns the signed distance to a sphere with given radius for the
8 // given position.
9 float sphere(vec3 position, float radius)
10 {
11     return length(position) - radius;
12 }
13
14 // Returns the signed distance to a box with given dimension for the
15 // given position.
16 float box(vec3 position, vec3 dimension)
17 {
18     position = abs(position) - dimension;
19     return max(max(position.x, position.y), position.z);
20 }
```

Auflistung 7.3: Umsetzung der Primitiven *Ebene* und *Kugel* in Form von impliziten Oberflächen in GLSL.

Verwendete Parameter sind jeweils die (gewünschte) Position des Objektes (*vec3 position*) sowie der Radius bzw. die Dimension (*float radius* bzw. *vec3 dimension*).

Als Beleuchtungsmodell wurde das unter Abschnitt 6.3 beschriebene Phong-Beleuchtungsmodell umgesetzt.

```

1 // Calculates the lighting for the given position, normal and direction,
2 // the given light (position and color) respecting the 'material'.
3 //
4 // This is mainly applying the phong lighting model including shadows.
5 //
6 // Returns the calculated color as three-dimensional vector.
7 vec3 calcLighting(vec3 normal, vec3 rayDirection) {
8
9     vec3 lightDirection = normalize(vec3(0.0, 4.0, 5.0));
10
11    float kDirectLight = 0.1;
12    float shadows = calcShadows(position, lightDirection);
13    vec3 direct = vec3(kDirectLight * shadows);
14
15    vec3 ambientColor = vec3(0.05, 0.15, 0.2);
16    float kAmbient = clamp(0.5 + 0.5 * normal.y, 0.0, 1.0);
17    vec3 ambient = kAmbient * ambientColor;
18
19    vec3 diffuseColor = vec3(0.2, 0.6, 0.8);
20    float kDiffuse = clamp(dot(lightDirection, normal), 0.0, 1.0);
21    vec3 diffuse = kDiffuse * diffuseColor;
22
23    vec3 specularColor = vec3(1.0);
24    float kSpecularExponent = 24.0;
25    vec3 h = normalize(-rayDirection + lightDirection);
26    float nFacing = clamp(dot(lightDirection, normal), 0.0, 1.0);
27    float kSpecular = pow(clamp(dot(h, normal), 0.0, 1.0), kSpecularExponent);
28    vec3 specular = nFacing * kSpecular * specularColor;
29
30    vec3 light = ambient + diffuse + specular + direct;
31    vec3 color = material * light;
32
33    return color;
34 }

```

Auflistung 7.4: Umsetzung des Phong-Beleuchtungsmodells in GLSL.

Als Parameter werden hier ein Normalvektor einer Oberfläche ( $\mathbf{n}$ ) sowie die Richtung eines eingehenden Strahles (also die Blickrichtung des Betrachters bzw. der Kamera,  $\vec{V}$ ) benötigt. Wie oben ersichtlich, wird zuerst die Richtung der Lichtquelle definiert, danach werden die einzelnen Anteile des Lichtes  $I_{\text{ambient}}$ ,  $I_{\text{diffuse}}$  sowie  $I_{\text{specular}}$  berechnet.

Die Normale einer Oberfläche wird wie unter Unterabschnitt 6.3.1 beschrieben berechnet.

```

1 // Calculates the normal vector for given position with respect to a
2 // certain offset given by epsilon.
3 vec3 calcNormal(in vec3 position, in float epsilon) {
4
5     vec3 eps = vec3(epsilon, 0.0, 0.0);
6     vec3 normal = vec3(
7         scene(position + eps.xyy).x - scene(position - eps.xyy).x,
8         scene(position + eps.yxy).x - scene(position - eps.yxy).x,
9         scene(position + eps.yyx).x - scene(position - eps.yyx).x
10    );
11
12    return normalize(normal);
13 }
14 }
```

Auflistung 7.5: Berechnung der Normalen einer impliziten Oberfläche in GLSL.

Verwendete Parameter sind ein Punkt der Oberfläche eines Objektes (*vec3 position*) sowie die minimale Inkrementation eines (Licht-) Strahles ( $\varepsilon$ ). Wie zuvor erwähnt, sollte für  $\varepsilon$  ein möglichst kleiner Wert gewählt werden. Daher wird standardmäßig der Wert *0.1* eingesetzt.

Bei der Funktion *scene* handelt es sich um eine Distanzfunktion  $f$  gemäß Abschnitt 6.1.3. Diese definiert schliesslich, was an einem gegebenen Punkt dargestellt wird.

```

1 // Defines the scene which will be drawn at given position.
2 float scene(in vec3 position) {
3 {
4     float sphereRadius = 1.0;
5     vec3 sphereOffset = vec3(-2.0, 1.0, -2.0);
6
7     float res = sphere(position - sphereOffset, sphereRadius);
8
9     return res;
10 }
```

Auflistung 7.6: Distanzfunktion  $f$  in GLSL.

In diesem Beispiel wird eine Kugel mit Radius *1.0* dargestellt. Diese wird auf der *X*-Achse um -2 Einheiten, auf der *Y*-Achse um 1 Einheit und auf der *Z*-Achse um -2 Einheiten verschoben (Translation).

Wie in Unterabschnitt 6.3.3 beschrieben, wurden weiche Schatten implementiert.

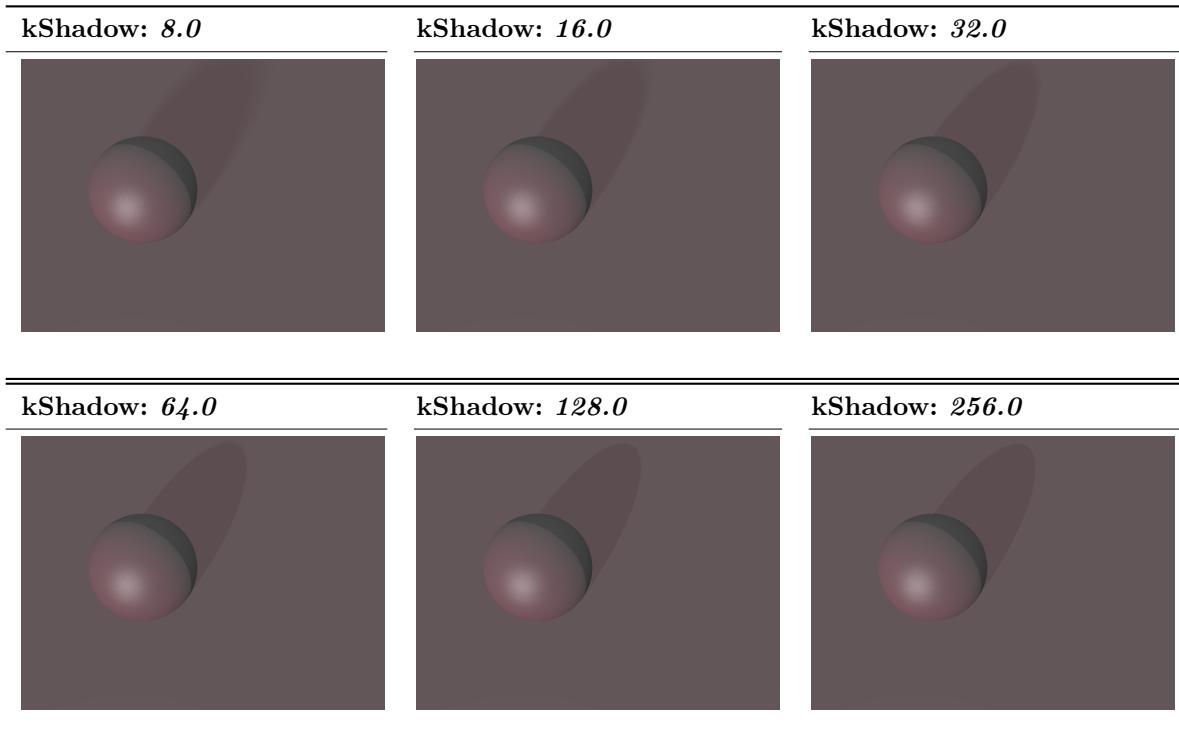
```

1 // Calculates soft shadows for the given ray origin and direction.
2 //
3 // Returns a shadow color for the given origin between 0.0 and 1.0.
4 float calcShadows(in vec3 rayOrigin, in vec3 rayDirection)
5 {
6     float shadow = 1.0;
7     float minimalDistance = 0.01;
8     float maximalDistance = 2.5;
9     float convergePrecision = 0.000001;
10    float kShadow = 8.0;
11    float currentDistance = minimalDistance;
12
13    while (currentDistance < maximalDistance) {
14        vec3 ray = rayOrigin + rayDirection * currentDistance;
15        float estimatedDistance = scene(ray);
16
17        if (estimatedDistance < convergePrecision) {
18            return 0.0;
19        }
20
21        float penumbraFactor = estimatedDistance / currentDistance;
22        shadow = min(shadow, kShadow * penumbraFactor);
23        currentDistance += estimatedDistance;
24    }
25
26    return clamp(shadow, 0.0, 1.0);
27
28 }
```

Auflistung 7.7: Funktion zur Berechnung von weichen Schatten in GLSL.

Verwendete Parameter sind der Ursprung (*vec3 rayOrigin*) sowie die Richtung (*vec3 rayDirection*) eines Strahles.

Tabelle 7.4.: Vergleich des Skalierungsfaktors  $k_{Shadow}$  für weiche Schatten anhand einer Beispielszene.



Das eigentliche Rendering, also die Darstellung der Szene, geschieht in der Funktion *render*.

```

1 // Performs rendering of a scene beginning at given origin in given ray
2 // direction. This invokes calculating the normal vector, the material
3 // as well as the lighting.
4 vec3 render(in vec3 rayOrigin, in vec3 rayDirection)
5 {
6     vec3 color = vec3(0.05, 0.08, 0.10);
7     vec3 res = castRay(rayOrigin, rayDirection, 100.0, 0.00001, 100);
8     float currentDistance = res.x;
9     float renderedScene = res.z;
10    float minimalDistance = -0.5;
11
12    // Perform further calculation only when the distance is not below
13    // the minimal distance (epsilon-factor).
14    if (currentDistance > minimalDistance) {
15        vec3 position = rayOrigin + currentDistance * rayDirection;
16        vec3 normal = calcNormal(position, 0.000001);
17
18        vec3 lightColor = vec3(0.7, 0.2, 0.3);
19        vec3 lightPosition = vec3(-0.6, 0.7, -0.5);
20        vec3 light = calcLighting(position, normal, rayDirection, material, ←
21            lightPosition, lightColor);
22
23        color = light;
24    }
25    color = clamp(color, 0.0, 1.0);
26
27    return color;
}

```

Auflistung 7.8: Funktion zur Darstellung der Szene in GLSL. Die Szene bzw. der Farbwert wird nur dann zurückgegeben bzw. berechnet, wenn eine minimale Distanz nicht unterschritten wird.

Die Hauptfunktion *main* des Fragment-Shaders ruft die Funktion zur Darstellung der Szene auf. Alle Methoden (wie z.B. *getRay* oder *squareFrame*) finden sich im Programmcode des Prototypen, welcher dieser Projektarbeit beiliegt.

```

1 // Main method of the shader.
2 void main()
3 {
4     vec2 resolution = globalResolution;
5     float time = globalTime * 1.1;
6     float cameraAngle = 1.0;
7     float cameraHeight = 2.0;
8     float cameraPane = 4.0;
9     float cameraDistance = 4.5;
10    vec3 rayOrigin = vec3(cameraPane * sin(cameraAngle), cameraHeight, ←
11        cameraDistance * cos(cameraAngle * time));
12    vec3 rayTarget = vec3(0.0, 0.0, 0.0);
13    vec2 screenPosition = squareFrame(resolution);
14    vec3 rayDirection = getRay(rayOrigin, rayTarget, screenPosition, 2.0);
15
16    vec3 color = render(rayOrigin, rayDirection);
17    color = calcPostFx(color, screenPosition);
18
19    gl_FragColor.rgb = color;
20    gl_FragColor.a = 1.0;
}

```

Auflistung 7.9: Einstiegspunkt des Fragment-Shaders.

# 8. Schlusswort

In dieser Projektarbeit wurde Sphere Tracing vorgestellt. Es ist eine optimierte Variante des als Ray Tracing bekannten Verfahrens. Sphere Tracing erlaubt die Darstellung von Szenen in der Qualität von Ray Tracing, aber in Echtzeit.

Zuerst wurden *lokale*, dann *globale Beleuchtungsmodelle* inklusive der *Renderinggleichung* vorgestellt. Weiter wurde das *Ray Tracing* Verfahren ausgehend von der ursprünglichen Idee des Ray Tracings, ein als *Ray Casting* bekanntes Verfahren, vorgestellt. Dabei wurde auch die zugrundeliegende Physik vereinfacht aufgezeigt.

Für die praktische Anwendung der Beleuchtungsmodelle wurden die klassischen Modelle zur Schattierung *Flat-Shading*, *Gouraud-Shading* sowie *Phong-Shading* vorgestellt.

Um die eigentliche Szene darstellen zu können, wurden *Oberflächen* im Allgemeinen eingeführt. Dabei wurde gezeigt, dass zur Modellierung von Oberflächen hauptsächlich zwei Techniken verwendet werden: Die parametrische und die implizite Modellierung bzw. Darstellung. Es wurden dann *implizite Oberflächen* im Speziellen behandelt. Mittels *Distanzfunktionen* wurde eine Grundlage zur Berechnung von Distanzen gezeigt, welche der späteren Modellierung und Darstellung dient. Ausgehend von den Distanzfunktionen wurden schliesslich *Distanzfelder* beschrieben, eine Art Datenstruktur basierend auf den Distanzfunktionen.

Mit *Ray Marching* und *Sphere Tracing* wurden zwei Methoden zur Darstellung von impliziten Oberflächen aufgezeigt. Weiter wurden *Operationen für implizite Oberflächen*, wie beispielsweise die Vereinigung oder Subtraktion, sowie Funktionen zur Modellierung von *geometrischen Primitiven* vorgestellt.

Schliesslich wurde gezeigt, wie die vorgestellten Grundlagen für das Rendering von impliziten Oberflächen genutzt werden können.

Im letzten Abschnitt wurde der umgesetzte *Prototyp* vorgestellt, wobei einzelne Parameter des Prototyps (*Distanz*, *Präzision*, die *Anzahl Schritte* und der *Skalierungsfaktor für Schatten*) anhand einer Beispieldarstellung verglichen wurden.

## 8.1. Erweiterungsmöglichkeiten

Da diese Projektarbeit den begrenzten Zeitrahmen eines Semesters hat, konnten nicht alle von mir gewünschten Themen bearbeitet werden.

Bei Sphere Tracing handelt es sich um ein relativ junges Verfahren, welches ein grosses Potential für Forschung und Entwicklung enthält. Angesichts des grossen Themengebietes kann man viel Zeit investieren.

Während der Recherche sind einige Themen hinzugekommen, sie werden aber nicht bearbeitet. Sie zeigen auf, in welche Richtung die Weiterführung von Projektarbeit und Prototyp gehen kann.

### 8.1.1. Umgebungs-Verdeckung (Ambient Occlusion)

“Umgebungsverdeckung (englisch Ambient Occlusion, AO) ist eine Shading-Methode, die in der 3D-Computergrafik verwendet wird, um mit relativ kurzer Renderzeit eine realistische Verschattung von Szenen zu erreichen. Das Ergebnis ist zwar nicht physikalisch korrekt, reicht jedoch in seinem Realismus oft aus, um auf rechenintensive globale Beleuchtung verzichten zu können.” [31]

Evans liefert in seiner Arbeit „Fast Approximations for Global Illumination on Dynamic Scenes“ interessante Ansätze, wie die Umgebungsverdeckung auf Distanzfelder angewendet werden kann. Dies kann mit relativ wenig Aufwand auch für Sphere Tracing angewendet werden [32].

### 8.1.2. Kantenglättung

Bei der Darstellung von Szenen mittels Sphere Tracing kommt es an Kanten zu harten Übergängen und Artefakten — es tritt das so genannte „Aliasing“ auf.

Hart schlägt in „Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces“ bereits Verfahren zur Kantenglättung vor. Diese verkomplizieren jedoch den Rendering-Prozess und haben Schwächen im Umgang mit angenäherten Obergrenzen von impliziten Oberflächen [19].

Lottes schlägt in seiner Arbeit *FXAA* eine einfache Lösung dieser Problematiken vor. Es handelt sich um einen Bild-Filter, welcher typischerweise in der Nachbearbeitung eines Bildes zum Einsatz kommt. Der Filter wird auf das gerenderte Bild (bzw. auf einem Bild-Puffer) angewendet. Es handelt sich um ein Verfahren, welches auf der Detektion von Kanten basiert [33].

### 8.1.3. Realistische Materialien

In Unterabschnitt 5.2.2 werden Verfahren zur Berechnung der physikalischen Eigenschaften von Oberflächen und damit von Materialien aufgezeigt. Dabei handelt es sich aber nur um Näherungen unter Annahme von perfekten Bedingungen, was von der Realität weit entfernt ist.

Eine allgemeinere und daher realistischerer Form ist die so genannte BRDF: „Eine bidirektionale Reflektanzverteilungsfunktion (engl. Bidirectional Reflectance Distribution Function, BRDF) stellt eine Funktion für das Reflexionsverhalten von Oberflächen eines Materials unter beliebigen Einfallswinkeln dar. Sie liefert für jeden auf dem Material auftreffenden Lichtstrahl mit gegebenem Eintrittswinkel den Quotienten aus Strahlungsdichte und Bestrahlungsstärke für jeden austretenden Lichtstrahl. BRDFs werden unter anderem in der realistischen 3D-Computergrafik verwendet, wo sie einen Teil der fundamentalen Rendergleichung darstellen und dazu dienen, Oberflächen möglichst realistisch und physikalisch korrekt darzustellen. Eine Verallgemeinerung der BRDF auf Texturen stellt die BTF (Bidirectional Texturing Function) dar.“ [34]

Burley liefert mit *Physical-Based Shading at Disney* eine gute Übersicht, wie solche Verfahren umgesetzt werden können [35]. Bagher, Soler und Holzschuch bieten in „Accurate Fitting of Measured Reflectances Using a Shifted Gamma Micro-facet Distribution“ eine Fülle an Materialien und Eigenschaften [36].

### 8.1.4. Optimierungsverfahren

Zur Beschleunigung des Sphere Tracing Verfahrens existieren diverse Möglichkeiten. So stellen z.B. Keinert, Schäfer, Korndörfer et al. in *Enhanced Sphere Tracing* diverse Methoden zur Optimierung und Beschleunigung des Renderings sowie Methoden zum Finden von Schnittpunkten vor [37]. Seven stellt in *Rendering Mandelbox fractals faster with Cone Marching* eine Methode vor, wie Sphere Tracing mittels Aussenden von Kegelförmigen (Primär-) Strahlen optimiert werden kann [38].

# Literatur

- [1] A. Appel, „Some Techniques for Shading Machine Renderings of Solids“, in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, Ser. AFIPS '68 (Spring), New York, NY, USA: ACM, 1968, S. 37–45. DOI: 10.1145/1468075.1468082.
- [2] T. Whitted, „An Improved Illumination Model for Shaded Display“, *Commun. ACM*, Bd. 23, Nr. 6, S. 343–349, Juni 1980, ISSN: 0001-0782. DOI: 10.1145/358876.358882.
- [3] Wikipedia Foundation, *Demoszene*, de, Page Version ID: 149578258, Dez. 2015.
- [4] ——, *Zotero*, Published: Website Abgerufen am 27. September 2015, Aug. 2015.
- [5] J. Foley, *Computer Graphics: PRINCIPLES and Practice*, Ser. Addison-Wesley systems programming series. Addison-Wesley, 1996, ISBN: 978-0-201-84840-3.
- [6] J. Hughes, A. Van Dam, J. Foley und S. Feiner, *Computer Graphics: PRINCIPLES and Practice*, Ser. The systems programming series. Addison-Wesley, 2013, ISBN: 978-0-321-39952-6.
- [7] B. T. Phong, „Illumination for Computer Generated Pictures“, *Commun. ACM*, Bd. 18, Nr. 6, S. 311–317, Juni 1975, ISSN: 0001-0782. DOI: 10.1145/360825.360839.
- [8] J. T. Kajiya, „The Rendering Equation“, *SIGGRAPH Comput. Graph.*, Bd. 20, Nr. 4, S. 143–150, Aug. 1986, ISSN: 0097-8930. DOI: 10.1145/15886.15902.
- [9] Arlington Mathematical Applications Group, Inc, *AFIPS '68 (Fall, part I): PROCEEDINGS of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. New York, NY, USA: ACM, 1968, Library of Congress Catalog Card No.: 55-44701.
- [10] A. S. Glassner, Hrsg., *An Introduction to Ray Tracing*. London, UK, UK: Academic Press Ltd., 1989, ISBN: 0-12-286160-4.
- [11] M. Pharr und G. Humphreys, *Physically Based Rendering, Second Edition: FROM Theory To Implementation*, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010, ISBN: 0-12-375079-2 978-0-12-375079-2.
- [12] P. S. Heckbert, „Adaptive Radiosity Textures for Bidirectional Ray Tracing“, *SIGGRAPH Comput. Graph.*, Bd. 24, Nr. 4, S. 145–154, Sep. 1990, ISSN: 0097-8930. DOI: 10.1145/97880.97895.
- [13] OpenGL foundation, *Rendering Pipeline Overview - OpenGL.org*, Mai 2015.
- [14] R. Fernando und M. J. Kilgard, *The Cg Tutorial: THE Definitive Guide to Programmable Real-Time Graphics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003, ISBN: 0-321-19496-9.
- [15] H. Gouraud, „Continuous Shading of Curved Surfaces“, *IEEE Trans. Comput.*, Bd. 20, Nr. 6, S. 623–629, Juni 1971, ISSN: 0018-9340. DOI: 10.1109/T-C.1971.223313.
- [16] B. Danchilla, *Beginning WebGL - Chapter 4 Examples*, Okt. 2014.
- [17] J. Menon, *An Introduction to Implicit Techniques*, Ser. Research report. IBM T.J. Watson Research Center, 1996.
- [18] J. C. Hart, „Ray tracing implicit surfaces“, *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, S. 1–16, 1993.
- [19] J. C. Hart, „Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces“, *The Visual Computer*, Bd. 12, S. 527–545, 1994.
- [20] M. W. Jones, J. A. Baerentzen und M. Srivastava, „3d Distance Fields: A Survey of Techniques and Applications“, *IEEE Transactions on Visualization and Computer Graphics*, Bd. 12, Nr. 4, S. 581–599, Juli 2006, ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.56.
- [21] K. Perlin und E. M. Hoffert, „Hypertexture“, *SIGGRAPH Comput. Graph.*, Bd. 23, Nr. 3, S. 253–262, Juli 1989, ISSN: 0097-8930. DOI: 10.1145/74334.74359.

- [22] J. C. Hart, D. J. Sandin und L. H. Kauffman, „Ray Tracing Deterministic 3-D Fractals“, *SIGGRAPH Comput. Graph.*, Bd. 23, Nr. 3, S. 289–296, Juli 1989, ISSN: 0097-8930. DOI: 10.1145/74334.74363.
- [23] O. Caprani, L. Hvidegaard, M. Mortensen und T. Schneider, „Robust and Efficient Ray Intersection of Implicit Surfaces“, *Reliable Computing*, Bd. 6, Nr. 1, S. 9–21, Feb. 2000, ISSN: 1573-1340. DOI: 10.1023/A:1009921806032.
- [24] D. P. Mitchell, „Robust Ray Intersection with Interval Arithmetic“, in *Proceedings on Graphics Interface '90*, Toronto, Ont., Canada: Canadian Information Processing Society, 1990, S. 68–74.
- [25] Liam Collins Sons & Co. Ltd., *Collins English Dictionary - Complete & Unabridged 10th Edition*, Okt. 2015.
- [26] T. Reiner, G. Mückl und C. Dachsbacher, „SMI 2011: Full Paper: Interactive Modeling of Implicit Surfaces Using a Direct Visualization Approach with Signed Distance Functions“, *Comput. Graph.*, Bd. 35, Nr. 3, S. 596–603, Juni 2011, ISSN: 0097-8493. DOI: 10.1016/j.cag.2011.03.010.
- [27] Wikipedia, the free encyclopedia, *Opengl*, de, Page Version ID: 146904140, Okt. 2015.
- [28] ——, *Glfw*, en, Page Version ID: 687716464, Okt. 2015.
- [29] ——, *OpenGL Extension Wrangler Library*, en, Page Version ID: 680716273, Sep. 2015.
- [30] ——, *Boost (C++-Bibliothek)*, de, Page Version ID: 146313866, Sep. 2015.
- [31] ——, *Umgebungsverdeckung*, de, Page Version ID: 147007685, Okt. 2015.
- [32] A. Evans, „Fast Approximations for Global Illumination on Dynamic Scenes“, in *ACM SIGGRAPH 2006 Courses*, Ser. SIGGRAPH '06, New York, NY, USA: ACM, 2006, S. 153–171, ISBN: 1-59593-364-6. DOI: 10.1145/1185657.1185834.
- [33] T. Lottes, *Fxaa*, 2009.
- [34] Wikipedia, the free encyclopedia, *Bidirektionale Reflektanzverteilungsfunktion*, de, Page Version ID: 136247995, Nov. 2014.
- [35] B. Burley, *Physical-Based Shading at Disney*, en, Aug. 2012.
- [36] M. M. Bagher, C. Soler und N. Holzschuch, „Accurate Fitting of Measured Reflectances Using a Shifted Gamma Micro-facet Distribution“, *Comput. Graph. Forum*, Bd. 31, Nr. 4, S. 1509–1518, Juni 2012, ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03147.x.
- [37] B. Keinert, H. Schäfer, J. Korndörfer, U. Ganse und M. Stamminger, *Enhanced Sphere Tracing*, en, 2014.
- [38] Seven, *Rendering Mandelbox fractals faster with Cone Marching*, en, 2012.

# Abbildungsverzeichnis

4.1. Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Kalenderwochen dar . . . . .	6
5.1. Illustration des Phong-Beleuchtungsmodells <sup>1</sup> . . . . .	10
5.2. Illustration des Ray Casting Verfahrens <sup>2</sup> . . . . .	13
5.3. Illustration des Prinzips des Ray Tracing Verfahrens <sup>3</sup> . . . . .	16
5.4. Illustration der einzelnen Strahlen und deren Verhalten ausgehend von der Szene in Abbildung 5.3. <sup>4</sup> Das im Bild ersichtliche, orange-farbige Dreieck hat eine undurchlässige, reflexive Oberfläche. Der türkis-farbige Würfel hat eine diffuse Oberfläche. Bei der Kugel, rechts im Bild, handelt es sich um eine Kugel aus Glas, welche das Licht teilweise bricht und reflektiert. . . . .	17
5.5. Illustration einer perfekt spiegelnden Reflexion. <sup>5</sup> $I$ ist der eingehende Strahl, welcher am Normalenvektor $N$ der schraffierten Oberfläche in Richtung $R$ reflektiert wird. Der Winkel des eingehenden Strahles $\theta_I$ ist gleichgross wie der Winkel des ausgehenden Strahles $\theta_R$ . . . . .	18
5.6. Illustration einer perfekt diffusen Reflexion. <sup>6</sup> $I$ ist der eingehende Strahl, welcher am Ort des Normalenvektors $N$ der schraffierten Oberfläche auftrifft. Das Licht wird gleichmässig in alle Richtungen gestreut. . . . .	19
5.7. Illustration einer perfekt brechenden Refraktion. <sup>7</sup> $I$ ist der eingehende Strahl, welcher am Ort des Normalenvektors $N$ im Winkel von $\theta_1$ von Medium 1 in Medium 2 übergeht und entsprechend mit Winkel $\theta_2$ anhand $T$ gebrochen wird. . . . .	20
5.8. Illustration der totalen internen Reflexion. <sup>8</sup> Ist der kritische Winkel $\theta_c$ beim Übertritt eines Strahles von einem Medium (hier Glas) zu einem anderen Medium (hier Luft) nicht überschritten, so wird der Strahl bzw. Licht sowohl gebrochen als auch reflektiert. Wird der kritische Winkel überschritten, so findet nur noch eine Reflexion statt. <sup>9</sup> . . . . .	20
5.9. Illustration der Ausgangslage <sup>10</sup> . . . . .	24
5.10. Illustration des Flat-Shadings <sup>11</sup> . . . . .	25
5.11. Illustration des Gouraud-Shadings <sup>12</sup> . . . . .	25
5.12. Illustration der berechneten Normalenvektoren $n_{v_2}$ und $n_{v_3}$ an den Eckpunkten $v_2$ und $v_3$ sowie des Normalenvektors $n_{v_{23}}$ der Kante $v_2v_3$ <sup>13</sup> . . . . .	25
5.13. Stark vereinfachte Illustration der interpolierten Normalenvektoren anhand der Kante $v_2v_3$ <sup>14</sup> . . . . .	26
6.1. Darstellung des Distanzfeldes einer dreidimensionalen Szene anhand Farbwerten <sup>15</sup> . . . . .	31
6.2. Illustration des Ray Marching Verfahrens und dessen Problematiken. <sup>16</sup> . . . . .	34
6.3. Illustration des Sphere Tracing Verfahrens. <sup>17</sup> . . . . .	36
6.4. Illustration des Sphere Tracing Verfahrens, Nahaufnahme. <sup>18</sup> . . . . .	36
6.5. Vereinigung von zwei Kugeln und einem Würfel. Bei dem Untergrund der Szene handelt es sich um eine Ebene, welche mit dem Rest der Szene vereinigt wird. <sup>19</sup> . . . . .	40
6.6. Mehrfache Subtraktion: In einem ersten Schritt wird in der Hälfte der Kugel ein Würfel von der Kugel subtrahiert. Der Würfel hat dieselbe Höhe wie die Kugel. In einem zweiten Schritt wird ein wesentlich kleinerer Würfel von der Halbkugel subtrahiert. <sup>20</sup> . . . . .	40
6.7. Illustration eines Schattenwurfes anhand einer Lichtquelle und einem Objekt. Im Bild sind die Umbra- und Penumbra-Regionen erkennbar. <sup>21</sup> . . . . .	46
7.1. Architektur des Prototypen <sup>22</sup> . . . . .	49
7.2. Bildliche Darstellung der Funktionsweise von Vertex- und Fragment-Shader der Applikation <sup>23</sup>	49

# Tabellenverzeichnis

5.1.	Darstellung des Strahlen-Baumes anhand einer Beispielszene. Die Lichtweg-Notation des Ray Tracings ( $LD?S^*E$ ) wird hier ersichtlich: Das Objekt $O_3$ muss einen spiegelnden Anteil in seiner Oberflächenbeschaffenheit haben, ansonsten würde die Strahlenverfolgung nach dem Auftreffen des Strahles $R_1$ bereits beendet. . . . .	22
5.2.	Vergleich der genannten Shading-Verfahren anhand einer Beispielszene <sup>24</sup> . . . . .	27
7.1.	Vergleich des Distanz-Parameters anhand einer Beispielszene. . . . .	51
7.2.	Vergleich des Präzisions-Parameters anhand einer Beispielszene. . . . .	51
7.3.	Vergleich des Parameters zur Bestimmung der Anzahl der Durchgänge anhand einer Beispielszene. . . . .	51
7.4.	Vergleich des Skalierungsfaktors $k_{Shadow}$ für weiche Schatten anhand einer Beispielszene.	56

# Auflistungsverzeichnis

4.1. Projekt-Struktur. . . . .	8
5.1. Eine abstrakte Umsetzung des Ray Casting Verfahrens <sup>25</sup> . . . . .	15
5.2. Eine abstrakte Umsetzung des Ray Tracings <sup>26</sup> . . . . .	23
6.1. Eine abstrakte Umsetzung des Ray Marchings <sup>27</sup> . . . . .	33
6.2. Eine abstrakte Umsetzung des Sphere Tracings <sup>28</sup> . . . . .	37
6.3. Algorithmus zur Berechnung von Schatten. . . . .	45
6.4. Algorithmus zur Berechnung von weichen Schatten. . . . .	47
7.1. Umsetzung des Sphere Tracings in GLSL. . . . .	50
7.2. Umsetzung der Operationen <i>Vereinigung</i> , <i>Subtraktion</i> sowie <i>Intersektion</i> für implizite Oberflächen in GLSL. . . . .	52
7.3. Umsetzung der Primitiven <i>Ebene</i> und <i>Kugel</i> in Form von impliziten Oberflächen in GLSL.	52
7.4. Umsetzung des Phong-Beleuchtungsmodells in GLSL. . . . .	53
7.5. Berechnung der Normalen einer impliziten Oberfläche in GLSL. . . . .	54
7.6. Distanzfunktion $f$ in GLSL. . . . .	54
7.7. Funktion zur Berechnung von weichen Schatten in GLSL. . . . .	55
7.8. Funktion zur Darstellung der Szene in GLSL. Die Szene bzw. der Farbwert wird nur dann zurückgegeben bzw. berechnet, wenn eine minimale Distanz nicht unterschritten wird. . .	57
7.9. Einstiegspunkt des Fragment-Shaders. . . . .	58

# Anhang

# A. Meeting minutes

---

20150921

---

No.: 01  
Date: 21.09.2015 07:30 - 07:55  
Place: Prof. Claude Fuhrer's office  
Involved persons: Claude Fuhrer  
Sven Osterwalder

Meeting minutes 2015-09-21

---

- \* Project issues
  - Requirement document needed?
    - \* No, not directly
  - What are the requirements?
    - \* Project schedule
    - \* External inputs
    - \* Conclusion
    - \* Grading is analogous to bachelor thesis, so the requirements are the same
- \* Goal
  - Read articles about the topic
  - Gain an understanding for the topic
  - Create a summary of read articles including small code segments in pseudo code, e.g. explaining an algorithm
- \* Project 2 (MTE7102)
  - Building a software architecture regarding the master thesis
  - Proof of concept of the algorithms chosen in this project
- \* Meetings
  - Will be held every 14 days
  - Time and location will be defined at the end of each meeting

TODO for next meeting:

- \* Set up project repository
  - GitHub
  - Open source
- \* Choose language for pseudo code

Next meeting:

Date: 04.10.2015 14:00  
Place: Skype

---

No.: 02  
Date: 04.10.2015 14:00 - 14:35  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-10-04

---

\* External documents

- Do external documents, e.g. papers, held in the project repository infringe copy rights? (CF)
- \* Both are not entirely sure about the copy rights, so it is decided to share the documents only between both persons via Dropbox (CF and SO)

\* Theoretical background

- Phong equation
- \* Why is the half way vector used (as described in Whitted's paper) instead of the more common usage of the angle (cosine) in direction of the light? (CF)
  - It is OK to use Whitted's originally proposed formula, as the results are the same for both formulas (CF)
  - Although the specular factor for the specular component is missing and has to be added (CF)

- Structuring / procedure

- \* Is the document structuring and are the plans for further development in good order? (SO)
- The structuring of the document as well as the plans for further development are in a good order and the work may continue in the currently ongoing direction (CF)

\* Literature

- Is it somehow possible to get the second edition of "Computer graphics: principles and practice"? (SO)
- \* Mr. Fuhrer possesses the mentioned book and proposes furthermore the lecture of a book dedicated to ray tracing which he also possesses. He will deposit both of the books on Monday, 5th of October 2015, in a room within the "Rolex" building of the Berne University of applied sciences in Biel (CF)

\* Citations

- Is the current way of citing in good order or do citations need to be more precise? (SO)
- \* The way of citing is precise enough, the schemata is the following: (CF)
  - Source, [Chapter], Page(s)

\* Document template

- Remove currently used font "cmbright" as invoked to the usage of the LaTeX template of the Berne University of applied sciences (CF)
- The lines shall be shortened so that they do not exceed 80 characters per line (CF)
- The margins, especially the left and the right margins, shall be enlarged as they are currently very narrow (CF)

TODO for next meeting:

- \* Present the current state of the work
- \* Discuss the current state of the work
- \* Define further steps/proceeding

Next meeting:

Date: 11.10.2015 14:00  
Place: Skype or in real life, if necessary

---

No.: 03  
Date: 11.10.2015 15:30 - 16:20  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-10-11

---

Presentation and discussion of the current state of the work

---

Theoretical background

---

- \* Local illumination models
  - \* Phong model, equation 5.2 (CF)
    - \* The notation of  $L_{\{j\}}$  and  $L_{\{j\}}^{\{'}}$  is nearly not distinguishable due to the vector arrows. Use another notation to distinguish them.  
(DONE)
    - \*  $k_d$  and  $k_s$  depend on the wave length resp.\ the color, this should be expressed as well  
(DONE)
    - \* The vectors  $L_{\{j\}}$  and  $L_{\{j\}}^{\{'}}$  are normalized unit vectors  
(DONE)
    - \* The exponent 'n', as used in the legend, is missing in the equation  
(DONE)
    - \* An image of the situation could be added for better understanding  
(DONE)
  - \* Global illumination models
    - \* Rendering equation, 5.3 (CF)
      - \* Add the correct reference, either to Kajiya or another source  
(DONE)
      - \* Spell the letter 's', used for the integral as well as for the explanations, always the same case, either upper or lower  
(DONE)
      - \* Use  $\varepsilon$  instead of  $\epsilon$  as this increases the readability  
(DONE)
      - \* Do not use tables for the explanation to an equation as LaTeX may break the layout then  
(DONE)
  - \* Ray casting
    - \* Is the description of ray casting developed enough? (SO)
      - \* No, the description should be further expanded, also formulas should be considered and added, as well as examples (CF)  
(DONE)
    - \* Is the pseudo code of ray casting developed enough? (SO)
      - \* Yes, the pseudo code is developed enough, although the unnecessary spaces of the lstlisting environment could be removed using the columns parameter with the fullflexible value (CF)  
(DONE)
  - \* Ray tracing
    - \* Is an explanation of transmission and refraction needed? (SO)
      - \* Yes, an explanation of those terms would be good (CF)  
(DONE)
    - \* The reference when referring Whitted's publication is missing(CF)
      - \* Add reference (SO)  
(DONE)
    - \* Is the material sufficient? (SO)
      - \* Yes, the material is sufficient, but an illustration would probably be good (CF)  
(DONE)
  - \* Implicit surfaces
    - \* Does one need to explain euclidean distance? (SO)
      - \* No, the term does not need to be explained as it is rather a standard

- term, but the notation may rather be  $(x^2 + y^2 + z^2)^{1/2}$  as this allows the usage of any basis, e.g. ln
  - (DONE)
- \* Equation (5.11) (CF)
  - \* Give an example by means of a sphere (CF)
    - (DONE)
  - \* Functions for implicit surfaces (5.4.1) (CF)
    - \* Use an explanation of the signs as well when explaining the results (CF)
      - (DONE)
  - \* Equation (5.15) (CF)
    - \* In the paragraph below the equation a sentence begins directly with a mathematical function. This is generally not a very good idea, so that sentence should be changed
      - (DONE)

#### Citations

---

- \* Change the style of citations, which one can be decided later on (CF)
  - (DONE), ieee-style is used, SO)

#### Typography

---

- \* Equation 5.1 (CF)
  - \* The notation of the word 'diffuse' is not nicely readable, as there is a gap between the to f letters which should not be there. \text might solve this problem
    - (DONE)

#### Further steps/proceedings (CF, SO)

---

- \* Further steps planned are the introduction of a lighting model for implicit surfaces, the definition of the rendering itself, the introduction of basic operations on implicit surfaces as well as shadows. If time allows it, maybe a prototype is added. Is this alright? (SO)
- \* Yes, absolutely, it is up to you, how far you are developing this project (CF)

#### TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)
- \* Define citation style (CF)

#### Scheduling of the next meeting

---

Date: 2015-10-18, 14:00

Place: Skype

No.: 04  
Date: 18.10.2015 14:00 -14:30  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-10-18

---

Presentation and discussion of the current state of the work (SO)

---

- \* Illustration Phong model
  - \* illustration is too big, should be a bit scaled (CF)  
(DONE)
  - \* Normalize vectors to same length, use polarcoordinates eventually (CF)  
(DONE)
  
- \* Illustration Ray Tracing
  - \* Use a better resolution (CF)  
(DONE)
  - \* Make the fonts a bit bigger (CF)  
(DONE)
  - \* Explain the image (CF)  
(DONE)
  - \* Normalize vectors to same length, use polarcoordinates eventually (CF)  
(DONE)
  
- \* Listings
  - \* Do listing in German also have a bullet in front or rather a dash? (CF)
    - \* A bullet appears to be right, as the ngerman package with the babel option is used (CF)

Further steps/proceedings (CF, SO)

---

- \* Finish writing on chapter about rendering implicit surfaces (SO, CF)
- \* Add a chapter about an eventual prototype (SO, CF)
- \* Process TODOs (SO, CF)

TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)
- \* Define citation style (CF)

Scheduling of the next meeting

---

Date: 25.10.2015 14:00  
Place: Skype, irl if needed

---

No.: 05  
Date: 25.10.2015 14:00 - 14:30  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-10-25

---

Presentation and discussion of the current state of the work (SO)

---

- \* Versioning
  - \* Added new versioning style, based on versionhistory (SO)
    - \* This is much better than the table-based approach used before (CF)
- \* Citations
  - \* May not be at the beginning of a sentence (CF)
  - \* Style is OK (CF)
  - \* When referring to authors, use their name and not only the abbreviation provided by the citation (CF)
    - (DONE)
- \* Introduction
  - \* May resp.\ will most likely need to be rewritten at end of the project work (CF)
    - (DONE)
- \* Project schedule
  - \* Adapt to current situation (CF)
    - \* Prototype is missing (CF)
      - (DONE)
- \* Prototype
  - \* Add version information for all used software (clang, GLEW and CMake are missing e.g.) (CF)
  - \* Use a tabular representation for maintain readability (CF)
    - (DONE)
- \* Local illumination models
  - \* Phong model, equation 5.2 (CF)
    - \* Explan  $L_{\{i\}}$  (CF)
      - (DONE)

Further steps/proceedings (CF, SO)

---

- \* Process TODOs
- \* Begin working on the prototype
- \* Add most relevant shader code to the prototype section
- \* Expand basic chapters about illumination models according to TODOs
- \* Expand chapters "scope" and "administrative"
- \* Re-work introduction
- \* Add management summary

TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)

Scheduling of the next meeting

---

Date: 02.11.2015 07:30

Place: Spetaccolo Biel

No.: 06  
Date: 02.11.2015 07:15 - 07:45  
Place: Spetaccolo Biel  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-11-02

---

Presentation and discussion of the current state of the work (SO)

---

- \* The main progress since the last meeting was the development of the prototype, the paper was not developed much further, only sections concerning the prototype have been added or expanded (SO)
  - \* This is OK, although the prototype has not been reviewed in detail yet (CF)
- \* The source code of the prototype should be commented, right now there are no comments (CF)  
(DONE)
- \* Deadline for handing in a rough draft is end of November resp.\ beginning of December 2015 (CF)  
(DONE)
- \* Deadline for handing in final version end of January 2016 (CF)
- \* Defence of the project work in February 2016, exact date to be defined (CF)

Further steps/proceedings (CF, SO)

---

- \* Process TODOs
- \* Begin working on the prototype
- \* Add most relevant shader code to the prototype section
- \* Expand basic chapters about illumination models according to TODOs
- \* Expand chapters "scope" and "administrative"
- \* Re-work introduction
- \* Add management summary

TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)

Scheduling of the next meeting

---

Date: 15.11.2015, 1400

Place: Skype

No.: 07  
Date: 15.11.2015 14:00 - 14:15  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-11-15

---

Presentation and discussion of the current state of the work (SO)

---

- \* The progress has slowed down a lot unfortunately. This is mostly due to work related issues as well as the focus was laid on the first specialization module of the studies during the last weeks. (SO)
- \* This is OK, there are still a lot of open points / TODOs remaining, the focus should now be set on finishing the document as well as on corrections. Currently there are no further aspects that need special attention. (CF)
- \* Some work was spent on the prototype as well on image generation out of that used for the documentation. (SO)

Further steps/proceedings (CF, SO)

---

- \* Process TODOs
- \* Expand basic chapters about illumination models according to TODOs
- \* Expand chapters "scope" and "administrative"
- \* Re-work introduction
- \* Add management summary

TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)

Scheduling of the next meeting

---

Date: 06.12.2015, 1400

Place: Skype

---

No.: 08  
Date: 06.12.2015 14:00 - 14:15  
Place: Skype  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2015-12-06

---

Presentation and discussion of the current state of the work (SO)

---

- \* The progress is still slowed down a lot unfortunately. This is mostly due to an illness as well as work related issues. (SO)
- \* This is OK, it was originally planned to have some breaks in the first three weeks of December. Those weeks were so to say spent already during November, so the work needs to be kept up during December. (CF)
- \* Currently there are no further aspects that need special attention. (CF)
- \* Make sure to commit also little progress at least to the Git repository, even when not updating the versioning table of the document. This makes progress visible, even if it is not finished yet. Keep that up on a weekly basis. If no progress has been made, e.g. due to illness, inform by e-mail that no progress has been made during a week, so that unnecessary effort may be kept at a minimal level. (CF)
- \* Effort has been put into re-generating the images as the currently used tool (Geogebra) leads to quick results, but makes it difficult to generate accurate images (e.g. all vectors need to have the same length and so on). First, IPE was used as a replacement, but seems not to be suited as it seems to be limited to basic operations. Second, Inkscape was which provide to be very useful/handy. But due to the fact, that most of the work was done during the illness, the sources were unfortunately not saved, only the bitmap formats which prevents (re-) editing. Therefore the decision was made to generate the images using asymptote.
- \* If any help using asymptote is needed, feel free to ask at any time. (CF)
- \* An introduction for the theoretical background was provided, a chapter about shading models was added. (SO)
- \* The timetable was completely re-worked. (SO)

Further steps/proceedings (CF, SO)

---

- \* Meetings  
The next meeting will be arranged by e-mail by Sven Osterwalder as needed.  
At the current stage of the project - which mainly consists of re-arrangement and correction work - it is not useful to keep up a high-frequent meeting-schedule. Mr. Fuhrer has the first week in January 2016 off, so a meeting in real life may be scheduled then
- \* Process TODOs
- \* Expand basic chapters about illumination models according to TODOs
- \* Expand chapters "scope" and "administrative"
- \* Re-work introduction
- \* Add management summary

TODO for the next meeting

---

- \* Present the current state of the work (SO)
- \* Discuss the current state of the work (CF, SO)
- \* Define further steps/proceeding (CF, SO)

Scheduling of the next meeting

---

Date: tbd

Place: Skype or in real life, if needed

---

20151011

---

No.: 09  
Date: 25.01.2016 10:00 - 11:00  
Place: Main building, Berne University of applied sciences  
Involved persons: Claude Fuhrer (CF)  
Sven Osterwalder (SO)

Meeting minutes 2016-01-25

---

Presentation and discussion of the current state of the work (SO)

---

- \* Nearly all TODOs were resolved. (SO)
- \* The document was revised by means of spelling and formulation. (SO)

Further steps/proceedings (CF, SO)

---

- \* References within the text and the bibliography do not seem to have the same abbreviations. (CF)  
(DONE, using ieee-style, SO)
- \* The paragraph right before subsection 5.1.2 should be moved a bit more in direction of equation I (5.1). (CF)  
(DONE)
- \* Instead of the word 'Epsilon' the (mathematical) variable varepsilon should be used throughout the whole document. (CF)  
(DONE)
- \* The equation(s) 5.6 are not a system of linear equations, it is instead only one equation. (CF)  
(DONE)
- \* Try not to use a vector as starting for the equation of a ray hitting a triangle with barycentric coordinates. Try to use the provided coordinates, E.g. use 1 - Beta - Gamma. (CF)  
(DONE)
- \* Try to avoid page breaks within pseudo-code listings. Use comments marking the end of blocks (if, loops etc.) or use parentheses if possible. (CF)  
(DONE)
- \* Citation right in front of subsection 5.2.2 has too much quotes. (CF)  
(DONE)
- \* Nu at equation 5.16 should be phi instead. (CF)  
(DONE)
- \* The principle shown in image 5.7 is rather a refraction instead of a reflexion. (CF)  
(DONE)
- \* Use two rows and two columns instead of three columns and one row only for table 5.1. This will improve readability. Try to use the light transport notation as well. (CF)  
(DONE)
- \* Use two rows and two columns instead of three columns and one row only for table 5.2. This will improve readability. (CF)  
(DONE)
- \* When referencing an authors name reference the paper as well. (CF)  
(DONE)
- \* When referencing something, e.g. an illustration or an equation, add the type as well, not only the referenced number as this will improve readability. (CF)  
(DONE)
- \* In the description of the fixed step size when describing ray marching the type of the delta (x-axis) is missing, only the parameter is being referenced. (CF)  
(DONE)
- \* Increase the size of illustration 6.2 as it has a lot of small details and is hard to recognize like this. (CF)  
(DONE)

- \* Try to explain ray \*g\* a bit better when explaining the problems of fixed step size ray marcing. (CF)  
(DONE)
- \* Try to mention solutions to the problem of using a fixed step size in ray marcing. (CF)  
(DONE)
- \* When referring to the prototype add a reference to the chapter as well. (CF)  
(DONE)
- \* Check the used values for epsilon within the prototype, they seem a bit small from time to time. This may lead to floating point precision problems otherwise. (CF) (DONE. The OpenGL Shading Language GLSL uses the highest precision available on a system. This is normally the `highp` float type and provides therefore - according to IEEE-754 resp. the OpenGL ES standard -  $-2^{62}$  to  $2^{62}$ .)
- \* Try mentioning floating point precision and rounding problems which may occur and the measures used against those problems. (CF)  
(DONE)

TODO for the next meeting

---

- \* Preparation for presenting and defending the whole project (SO)

Scheduling of the next meeting

---

Date: 18.02.2016, 1315  
 Place: Meeting room of RISI building at the Berne University of applied sciences in Biel

---