



# Volume ray casting — basics & principles

## Projektarbeit 1

**MTE7101**

Studiengang: Informatik  
Autor: Sven Osterwalder<sup>1</sup>  
Betreuer: Prof. Claude Fuhrer<sup>2</sup>  
Datum: 06.12.2015  
Version: 0.16



Licensed under the Creative Commons Attribution-ShareAlike 3.0 License

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

<sup>2</sup>claudio.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, Dokumentaufbau, 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 Kapitels über Ray Tracing 5.2 sowie über Darstellung von impliziten Oberflächen 6.2
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-Beleuchtungsmodelles 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 Kapitel über die Umsetzung eines Prototypen 7
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.

# Todo list

Check and rewrite when finished with the rest of document . . . . .	1
Describe scope . . . . .	3
Describe motivation . . . . .	3
Loose some words about demoscene! . . . . .	3
Describe initial situation . . . . .	3
Describe objectives . . . . .	3
Describe preliminaries . . . . .	3
Describe new learning contents . . . . .	3
Describe theoretical background . . . . .	5
Scale illustration down, normalize vectors to same length . . . . .	8
Explain $L_i$ ! . . . . .	9
Expand this section. Add formulas as well as examples. . . . .	10
Punkte mehr ausführen; Integration zeigen mit Diskretition; auch Berechnungen . . . . .	10
Explain illustration 5.3, provide a better resolution, use bigger font, use same length for vectors . .	13
Explain transmission and refraction . . . . .	13
Add an (sphere) example, explain basic terminology (illum. and shadow rays, aliasing and so on) .	13
Translate/cite . . . . .	13
Image explaining average calculation . . . . .	14
Translate/cite . . . . .	14
Provide example by means of a sphere . . . . .	16
Introduce distance fields? . . . . .	18
Add an introduction . . . . .	27
Explain lighting model . . . . .	27
Comment source code of prototype . . . . .	32

# Management Summary

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus scelerisque, leo sed iaculis ornare, mi leo semper urna, ac elementum libero est at risus. Donec eget aliquam urna. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc fermentum nunc sollicitudin leo porttitor volutpat. Duis ac enim lectus, quis malesuada lectus. Aenean vestibulum suscipit justo, in suscipit augue venenatis a. Donec interdum nibh ligula. Aliquam vitae dui a odio cursus interdum quis vitae mi. Phasellus ornare tortor fringilla velit accumsan quis tincidunt magna eleifend. Praesent nisl nibh, cursus in mattis ac, ultrices ac nulla. Nulla ante urna, aliquet eu tempus ut, feugiat id nisl. Nunc sit amet mauris vitae turpis scelerisque mattis et sed metus. Aliquam interdum congue odio, sed semper elit ullamcorper vitae. Morbi orci elit, feugiat vel hendrerit nec, sollicitudin non massa. Quisque lacus metus, vulputate id ullamcorper id, consequat eget orci.

# Inhaltsverzeichnis

<b>Management Summary</b>	<b>iv</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. Ausgangslage . . . . .	3
3.3. Ziele und Abgrenzung . . . . .	3
<b>4. Vorgehen</b>	<b>4</b>
4.1. Arbeitsorganisation . . . . .	4
4.2. Projektphasen . . . . .	4
4.3. Technologien . . . . .	6
<b>5. Theoretischer Hintergrund</b>	<b>7</b>
5.1. Beleuchtungsmodelle . . . . .	7
5.2. Ray Casting und Ray Tracing . . . . .	10
5.3. Modelle zur Schattierung (shading models) . . . . .	13
<b>6. Oberflächen</b>	<b>16</b>
6.1. Oberflächen . . . . .	16
6.2. Darstellung von impliziten Oberflächen . . . . .	18
6.3. Rendering von impliziten Oberflächen . . . . .	27
<b>7. Prototyp</b>	<b>32</b>
7.1. Architektur . . . . .	32
7.2. Umsetzung . . . . .	34
<b>8. Diskussion und Fazit</b>	<b>40</b>
8.1. Diskussion . . . . .	40
8.2. Erweiterungsmöglichkeiten . . . . .	40
8.3. Fazit . . . . .	40
<b>Glossar</b>	<b>41</b>
<b>Literaturverzeichnis</b>	<b>41</b>
<b>Abbildungsverzeichnis</b>	<b>41</b>
<b>Tabellenverzeichnis</b>	<b>42</b>
<b>Auflistungsverzeichnis</b>	<b>43</b>
<b>Anhang</b>	<b>45</b>
<b>A. Meeting minutes</b>	<b>46</b>

# 1. Einleitung

Seit dem Bestehen moderner Computer existiert auch die Computergrafik. Ziel der Computergrafik ist es unter Anderem den dreidimensionalen Raum auf eine zweidimensionale Fläche abzubilden, da die Ausgabe meist auf den zweidimensionalen Raum limitiert ist.

Dabei wird zwischen statischen Bildern und dynamischen Bildern unterschieden. Statische Bilder werden bei Bedarf dargestellt und ändern sich in der Regel nicht. Dynamische Bilder können sich hingegen ständig ändern und müssen — bedingt durch das menschliche Auge — mit 25 Bildern pro Sekunde ausgegeben werden. Es bestand bereits früh das Bestreben eine möglichst realistische Darstellung zu erhalten. Eine Darstellung also, die möglichst nahe an der menschlichen Wahrnehmung liegt.

Im Laufe der Zeit entstanden verschiedene Ansätze um eine solche Darstellung zu bieten. Ein Teilgebiet davon 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 Strahlenverfolgung), welches 1980 von Turner Whitted vorgestellt wurde wurde. Das Verfahren besticht durch seine Einfachheit und bietet dabei eine hohe Bildqualität mit perfekten Spiegelungen und Transparenzen. Mit entsprechenden Optimierungen ist das Verfahren auch relativ schnell.

Mit schnell ist dabei die Zeit gemeint, die benötigt wird um ein einzelnes Bild darzustellen. Möchte man jedoch eine Darstellung in Echtzeit erreichen, so war das Verfahren lange zu langsam.

Im Rahmen der Weiterentwicklung der Computer und vor allem durch die Weiterentwicklung der Grafikkarten (GPUs), ist Ray Tracing jedoch 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.

Check and rewrite when finished with the rest of document

## 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

Der Aufbau der vorliegenden Arbeit ist 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

---

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

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



## 3. Aufgabenstellung

Describe scope

### 3.1. Motivation

Describe motivation

#### 3.1.1. Demoszene

Loose some words about demoscene!

### 3.2. Ausgangslage

Describe initial situation

### 3.3. Ziele und Abgrenzung

Describe objectives

#### 3.3.1. Vorgängige Arbeiten

Describe preliminaries

#### 3.3.2. Neue Lerninhalte

Describe new learning contents

## 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 mindestens alle zwei Wochen statt, sie wurden in Form eines Protokolles festgehalten.

### 4.2. Projekphasen

#### 4.2.1. Meilensteine

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

- Projektstart
- Erarbeitung und Festhalten der Anforderungen
- Erarbeitung der theoretischen Grundlagen
- Erstellung der abschliessenden Dokumentation

Die Phasen der Erarbeitung der theoretischen Grundlagen sowie die Erstellung der abschliessenden Dokumentation liefen parallel ab.

## 4.2.2. Zeitplan / Projektphasen

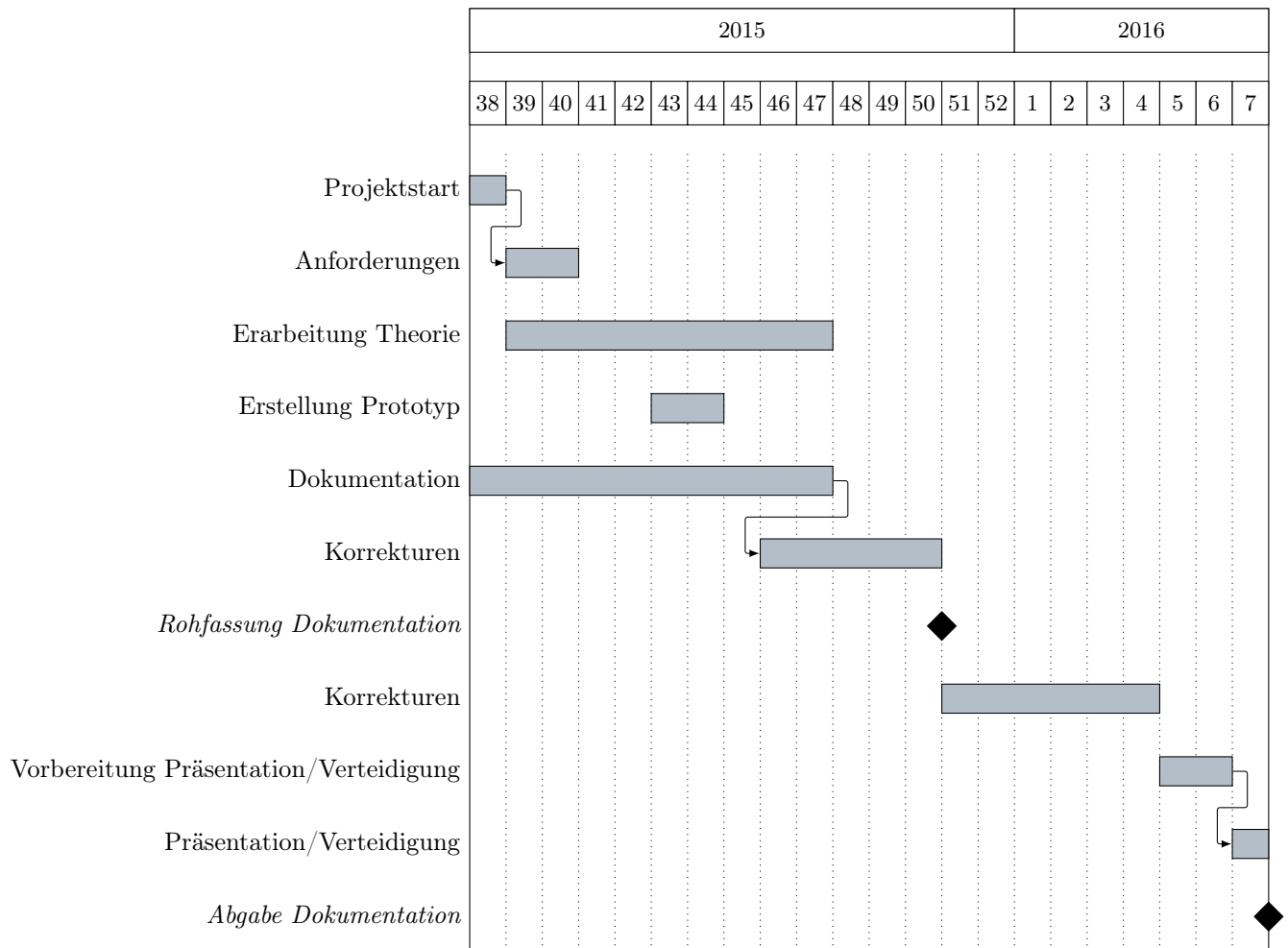


Abbildung 4.1.: Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Kalenderwochen dar

### Projektstart

In der ersten Phase wurden die Meilensteine der Arbeit identifiziert und skizziert. Um Details der Aufgabe zu verstehen, wurde das notwendige Vorwissen über globale Beleuchtungsalgorithmen erarbeitet. Weiter wurde das Grundgerüst dieser Dokumentation erstellt.

### Anforderungen

In dieser Phase wurde das Ziel dieser Projektarbeit festgelegt. Vom Ziel ausgehend wurden die dazu erforderlichen Projektphasen festgelegt.

### Erarbeitung theoretische Grundlagen

Describe theoretical background

## 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*

**LaTeX** Eine Makro-Sammlung für das TeX-System. Wurde zur Erstellung dieser Dokumentation eingesetzt. Diese Dokumentation wurde mittels LaTeX geschrieben.

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

**zotero** Ein freies, quelloffenes Literaturverwaltungsprogramm zum Sammeln, Verwalten und Zitieren unterschiedlicher Online- und Offline-Quellen [Wik15a].

**VIM** Vi IMproved. Ein freier, quelloffener Texteditor zur Textbearbeitung.

#### *Arbeitsorganisation*

**Git** Freie Software zur verteilten Versionsverwaltung, wurde für die Entwicklung dieser Dokumentation 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

#### **Pseudocode**

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

---

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

## 5. Theoretischer Hintergrund

Sofern nicht anders vermerkt, basiert das folgende Kapitel auf [Fol96].

Um überhaupt Bilder generieren und darstellen zu können, braucht es zwei Faktoren: Was dargestellt werden soll und wie es dargestellt werden soll. Prinzipiell geht es darum zu bestimmen, welche Farbe eine Oberfläche an einem bestimmten Punkt hat. Dabei haben sich die Begriffe des *Beleuchtungsmodelles* (*illumination model*) und des *Modelles zur Schattierung* (*shading model*) etabliert.

(author?) nutzt den Begriff *shading model* als Überbegriff, welcher auch Beleuchtungsmodelle umfasst. Daher definiert ein *shading model* wann ein Beleuchtungsmodell mit welchen Parametern 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 nicht anders vermerkt, basiert der folgende Abschnitt auf [Whi80][S. 343] sowie auf [HVDFF13].

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

Es wird dabei zwischen lokalen und globalen Beleuchtungsmodellen unterschieden.

#### 5.1.1. Lokale Beleuchtungsmodelle

Lokale Beleuchtungsmodelle aggregieren Daten von benachbarten, eben lokalen, Oberflächen. Diese Modelle sind in deren 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. Dies ist dadurch bedingt, dass die traditionell verwendeten Algorithmen zur Berechnung der Sichtbarkeit von Oberflächen, über keine globalen Daten verfügen. Das bekannteste lokale Beleuchtungsmodell ist das **Phong**-Beleuchtungsmodell, welches 1975 von Phong Bui-Tong entwickelt wurde. Es beschreibt nicht-perfekte Reflektoren wie zum Beispiel einen Apfel [Fol96][Kapitel 16, Seite 729].

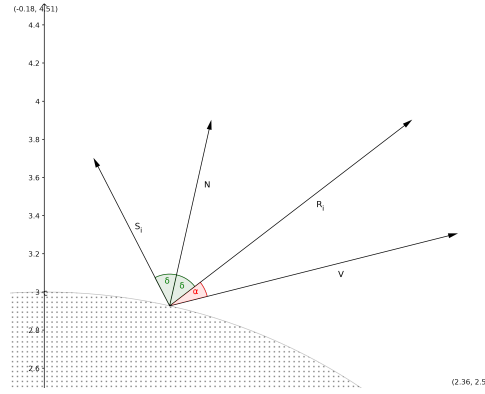


Abbildung 5.1.: Illustration des Phong-Beleuchtungsmodells<sup>1</sup>

Scale illustration down, normalize vectors to same length

Es beschreibt die reflektierte (Licht-) Intensität 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:

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

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.
- $k_d$ : Konstante für die diffuse Komponente des reflektierten Lichtes, basierend auf der Wellenlänge bzw. 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
- $\vec{R}_i$ : Richtung, in welcher das Licht der  $i$ -ten Lichtquelle reflektiert wird, normalisierter Einheitsvektor
- $\vec{V}$ : Blickrichtung des Betrachters bzw. der Kamera
- $k_e$ : Exponent, welcher von der Rauheit bzw. Reflexion der Oberfläche abhängt

<sup>1</sup>Eigene Darstellung mittels Geogebra, angelehnt an [Fol96][Kapitel 16, Seite 731, Abbildung 16.12]

### Explain $L_i$ !

Meist wird der emissive Term bewusst weggelassen, da dieser meist eher für Spezialeffekte statt für die Beleuchtung “normaler” Objekte benutzt wird.

Der reflektive Vektor  $R_i$  ist gegeben durch

$$R_i = S_i + 2(S_i \cdot N)N \quad (5.3)$$

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 nicht anders vermerkt, basiert der folgende Abschnitt auf [Fol96][S. 775ff]

Globale Beleuchtungsmodelle beschreiben die reflektierte (Licht-) Intensität eines Punktes aufgrund direkter Lichteinstrahlung durch Lichtquellen sowie durch alles Licht, welches diesen Punkt nach Reflektion von bzw. Durchdringen der eigenen oder anderer Oberflächen erreicht.

Bei globalen Beleuchtungsmodellen unterscheidet man zwischen blickwinkelabhängigen Algorithmen, wie etwa Ray Tracing, und zwischen blickwinkelunabhängigen Algorithmen, wie etwa Photon Mapping.

Blickwinkelabhängige Algorithmen verwenden eine Diskretisierung der sichtbaren Fläche bzw. Bildfläche um zu entscheiden, an welchen Punkten, in Blickrichtung des Betrachters, die Beleuchtungsberechnung durchgeführt werden soll. Blickwinkelunabhängige Algorithmen hingegen diskretisieren und verarbeiten die Umgebung um genügend Informationen für die Beleuchtungsberechnung zu haben. Dies erlaubt ihnen die Beleuchtungsberechnung an einem beliebigen Punkt aus einer beliebigen Blickrichtung.

Beide Arten von Algorithmen haben jedoch Vor- und Nachteile. So sind blickwinkelabhängige Algorithmen gut geeignet um Spiegelungen, basierend auf der Blickrichtung des Betrachters, zu berechnen, eignen sich aber weniger um gleichbleibende diffuse Anteile über weiter Flächen eines Bildes zu berechnen. Bei blickwinkelunabhängigen Algorithmen verhält es sich genau umgekehrt.

## Renderinggleichung

Die unter 5.1.2 genannten Verfahren versuchen auszudrücken, wie sich Licht von einem Punkt im Raum zu einem anderen bewegt. Dabei beschreiben sie die Intensität des Lichtes, ausgehend vom ersten Punkt zum zweiten Punkt. Zusätzlich wird die Intensität des Lichtes, ausgehend von allen anderen Punkten, welche den ersten Punkt erreichen, und zum zweiten Punkt emittiert werden, beschrieben.

James (Jim) Kajiya stellte 1986 die so genannte Renderinggleichung auf, welche genau dieses Verhalten beschreibt [Kaj86] und [Fol96]:

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

wobei gilt:

- $x, x'$  und  $x''$ : Punkte in der Umgebung
- $I(x, x')$ : Lichtintensität von Punkt  $x'$  nach Punkt  $x$
- $g(x, x')$ : 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
- $\epsilon(x, x')$ : 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

Expand this section. Add formulas as well as examples.

Punkte mehr ausführen; Integration zeigen mit Diskretition; auch Berechnungen

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [HVDFF13][Kapitel 15, S. 387ff].

Um ein Bild möglichst realistisch darzustellen muss berechnet werden, wieviel Licht zu jedem Pixel der sichtbaren Bildfläche (also dem Betrachter) transportiert wird. Da Photonen die Energie des Lichtes transportieren, muss man also das physikalische Verhalten dieser simulieren. Es ist allerdings nicht möglich *alle* Photonen zu simulieren, da der Aufwand schlicht zu gross wäre. Daher macht es Sinn nur einige Photonen (exemplarisch) zu betrachten und dann eine Abschätzung des gesamten Lichtes vorzunehmen.

### 5.2.1. Ray Casting

Bei **Ray Casting** handelt es sich grundsätzlich um eine Strategie zur Simulation, wieviel Licht anhand eines (Licht-) Strahles zu der sichtbaren Bildfläche (also dem Betrachter) transportiert wird.

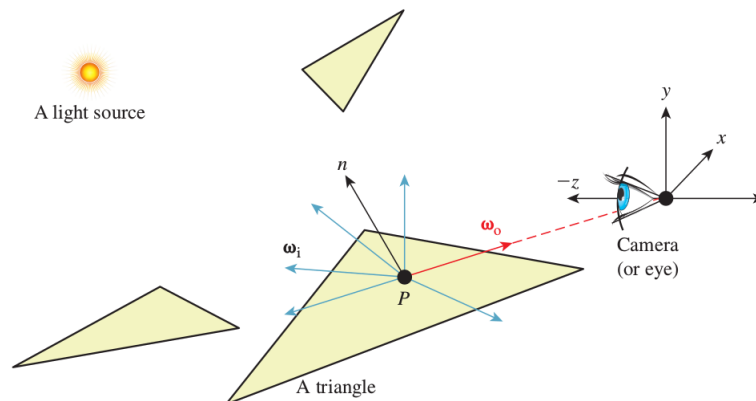


Abbildung 5.2.: Punkt  $P$  auf einer Oberfläche eines Dreieckes, welcher für die Kamera bzw. den Betrachter sichtbar ist. Der Betrachter nimmt dabei das Licht, welches aus verschiedenen Richtungen  $\omega_i$  kommt, über den Punkt  $P$  in Richtung  $\omega_o$  wahr.<sup>2</sup>

Wie in Abbildung 5.2 ersichtlich, gelangt Licht aus vielen Richtungen durch den Punkt  $P$  zu dem Betrachter. Dies beinhaltet auch die Möglichkeit, dass Licht nicht nur von einer Lichtquelle aus, sondern von vielen Lichtquellen aus via  $P$  zum Betrachter gelangt. Weiter ist es möglich, dass Licht zuvor an anderen Punkten gestreut und/oder gespiegelt und erst dann via  $P$  zum Betrachter gelangte.

Dies führt zu den folgenden Schlussfolgerungen:

- Es müssen alle möglichen Richtungen, aus denen Licht kommen könnte, an Punkt  $P$  untersucht werden.

<sup>2</sup>Darstellung von [HVDFF13][Kapitel 15, Seite 389, Abbildung 15.1]



- Da, bedingt durch technische Limitierungen, nur diskretes Abtasten möglich ist, müssen die Richtungen auf eine endliche Anzahl beschränkt werden, was zu Abtastfehlern führen kann.

Um die Abtastfehler zu minimieren, können die Richtungen des Abtastens anhand der Lichtquellen priorisiert werden.

Ein möglicher Algorithmus, wie solch ein Verfahren umgesetzt werden kann, findet sich in 5.1.

```
def ray_cast():
    # "pixels" is a list of all pixels of the image plane
    for pixel in pixels:
        # Save all intersections for given pixel
        intersections = []

        # Returns the ray passing through the given
        # pixel from the eye
        ray = ray_at_pixel(pixel)

        # "scene_triangles" is a list of all triangles
        # coming from meshes contained in the scene to render
        for triangle in scene_triangles:
            p = intersect(ray, triangle)
            sum = 0

            for light in incoming_lights_at_p:
                sum = sum + l.value
            end

            if is_smallest_intersection(p, intersections):
                pixel = sum
            intersections.append(p)
```

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

Das Verfahren wurde erstmals 1968 in [App68] vorgeschlagen und auch 1968 von der Mathematical Applications Group Inc. in [Arl68] erfolgreich umgesetzt.

### 5.2.2. Ray Tracing

Bei dem heute als Ray Tracing bekannten Verfahren, handelt es sich um eine verbesserte Version des unter 5.2.1 genannten Ray Casting Verfahrens. Dieses wurde im Juni 1980 durch [Whi80][S. 345] verbessert.

So schlägt [Whi80] vor, dass die Berechnung der Sichtbarkeit (von Objekten) nicht bei dem nächsten gefundenen Schnittpunkt abgebrochen wird, sondern dass jedes Auftreffen eines (Licht-) Strahles mehr (Licht-) Strahlen durch Transmission bzw. Reflektion sowie in Richtung jeder Lichtquelle gesendet werden. Dieser Prozess wird so lange wiederholt, bis keiner der neu generierten (Licht-) Strahlen mehr auf ein Objekt trifft [Whi80][S. 345].

Es handelt sich dabei also um ein rekursives Verfahren und wird daher teilweise auch rekursives Ray Tracing genannt.

---

<sup>3</sup>Algorithmus in Pseudocode gemäss [HVDFF13][Kapitel 15, Seite 391, Auflistung 15.2]

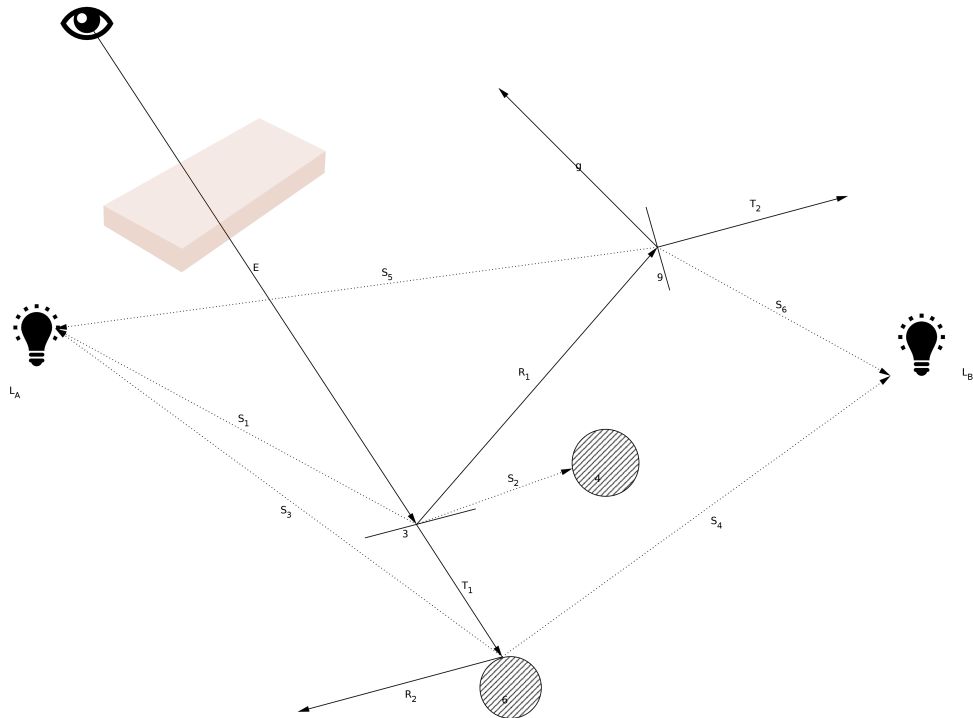


Abbildung 5.3.: Illustration des Ray Tracing Verfahrens<sup>4</sup>

Explain illustration 5.3, provide a better resolution, use bigger font, use same length for vectors

Explain transmission and refraction

Add an (sphere) example, explain basic terminology (illum. and shadow rays, aliasing and so on)

## 5.3. Modelle zur Schattierung (shading models)

Sofern nicht anders vermerkt, basieren die nachfolgenden Abschnitte auf [Fol96][S. 734–739].

Bei der Anwendung von Modellen zur Schattierung (shading models) geht es grundsätzlich darum die emittierte Lichtintensität bzw. die Farbe einer Oberfläche an einem bestimmten Punkt zu berechnen. Es wäre naheliegend dies für jeden sichtbaren Punkt der Oberfläche zu berechnen, dies ist jedoch häufig viel zu aufwändig. Viele Modelle zur Schattierung berechnen daher die Licht- bzw. Farbintensität nur an gewissen Schlüsselpunkten und wenden dann vereinfachte Modelle zur Berechnung an um so Rechenzeit zu sparen.

Als Beispiel zur Anwendung der Schattierungsmodelle 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  $V_4$ .

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

Translate/cite

<sup>4</sup>Eigene Darstellung mittels Geogebra, angelehnt an [Gla89][Kapitel 2, Seite 16, Abbildung 11; Piktogramm Auge erstellt von "Icomoon" auf [www.flaticon.com](http://www.flaticon.com); Piktogramm Glühbirne erstellt von "Simpleicon" auf [www.flaticon.com](http://www.flaticon.com);]

Modern shaders are really graphics programs rather than being restricted to computing colors of points. There are geometry shaders, which can alter the list of triangles to be processed in subsequent stages, and tessellation shaders, which take high-level descriptions of surfaces and produce triangle lists from them; an example is a subdivision surface shader, which might take as input the vertices and mesh structure of a subdivision surface's control mesh, and produce as output a collection of tiny triangles that form a good approximation of the limit surface. There are also vertex shaders that serve only to transform the vertex locations, and generally have nothing to do with eventual color.

### 5.3.1. Flat shading — per vertex lighting

Bei Flat-Shading wird pro Oberfläche (Polygon) ein Eckpunkt  $V_1$  dieser als farb- bzw. intensitätgebender Schlüsselpunkt bestimmt. Danach wird die Farbe des Punktes als Farbe für die Oberfläche genommen.

Diese Annahme ist unter den folgenden Voraussetzungen gültig:

1. Die zugrundeliegende Lichtquelle befindet sich unendlich weit entfernt, so dass der Winkel zwischen dem Normalenvektor der Oberfläche und der Lichtquelle  $\mathbf{n} \cdot \mathbf{l}$  für die gesamte Oberfläche konstant ist.
2. Der Betrachter sich unendlich weit entfernt von der Oberfläche befindet, 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.

Ist eine der ersten beiden Annahmen falsch, so muss für den Vektor der Lichtquelle  $\mathbf{l}$  bzw. den Vektor des Betrachters  $\mathbf{v}$  ein konstanter Wert berechnet werden. (**author?**) gibt hier als Beispiele das Zentrum des Polygons oder den ersten Eckpunkt des Polygons an.

### 5.3.2. Gouraud shading — face interpolated lighting

Bei Gouraud-Shading handelt es sich um ein Shading-Verfahren, welches die Farbintensitätswerte der Eckpunkte von Oberflächen eines Meshes interpoliert.

Um den Farbintensitätswert eines Eckpunktes von Oberflächen zu berechnen, schlägt Gouraud die Berechnung des Durchschnittswertes der Oberflächennormalen zweier adjazenter Liniensegmente (im 2D-Raum) bzw. aller adjazenter Dreiecke (im 3D-Raum) vor.

Image explaining average calculation

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

### 5.3.3. Phong shading — normal interpolated lighting

Translate/cite

Phong shading improves upon Gouraud shading and provides a better approximation of the shading of a smooth surface. Phong shading assumes a smoothly varying surface normal vector. The Phong interpolation method works better than Gouraud shading when applied to a reflection model that has small specular highlights such as the Phong reflection model.

Unlike Gouraud shading, which interpolates colors across polygons, in Phong shading a normal vector is linearly interpolated across the surface of the polygon from the polygon's vertex normals. The surface normal is interpolated and normalized at each pixel and then used in a reflection model, e.g. the Phong

reflection model, to obtain the final pixel color. Phong shading is more computationally expensive than Gouraud shading since the reflection model must be computed at each pixel instead of at each vertex.

## 6. Oberflächen

### 6.1. Oberflächen

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [DM96][S. 1 ff].

Um in Computergrafiken überhaupt etwas darstellen zu können, muss erst einmal definiert werden, was dargestellt werden soll. Häufig orientiert sich die Computergrafik dabei an der realen Welt. In der realen Welt haben Oberflächen von Objekten häufig keine starken Übergänge (Kanten) sondern sind eher von glatter Natur [Fol96][S. 471].

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

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

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

Provide example by means of a sphere

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

Bei der impliziten Darstellung wird eine Oberfläche üblicherweise als Kontur einer Funktion mit Wert 0 definiert, so zum Beispiel:

$$f(\mathbf{p}) = f(x, y, z) = 0 \quad (6.2)$$

Die parametrische Darstellung bringt Vorteile wie die Unabhängigkeit von einem Koordinatensystem oder eine effiziente Berechnung von Punkten auf einer Oberfläche. Die implizite Darstellung erlaubt hingegen eine grössere Einflussnahme aus mathematischer Sicht und ist daher sehr nützlich für Operationen wie Biegung, Vermischung, Schnitte (Intersektion) oder Bool'sche Operationen.

#### 6.1.1. Implizite Oberflächen

Wie in Gleichung 6.1 beschrieben, ist eine implizite Oberfläche gemäss [Har93][S. 1] als Funktion  $f(\mathbf{x}) = \mathbb{R}^3 \rightarrow \mathbb{R}$  definiert. Es wird also jedem Punkt einer Menge  $\mathbf{p} \in \mathbb{R}^3$  ein skalarer Wert  $s \in \mathbb{R}$  zugewiesen. Dabei besteht die Oberfläche aus der Punktmenge  $\mathbf{x} \equiv (x, y, z) \in \mathbb{R}^3$ .

Angenommen  $\mathbf{A}$  ist ein geschlossener Festkörper, welcher durch die Funktion  $f$  beschrieben wird, dann kann gemäss [Har93] Folgendes angenommen werden:

$$x \in \overset{\circ}{\mathbf{A}} \Leftrightarrow f(\mathbf{x}) < 0 \quad (6.3)$$

$$x \in \partial \mathbf{A} \Leftrightarrow f(\mathbf{x}) = 0 \quad (6.4)$$

$$x \in \mathbb{R}^3 - \mathbf{A} \Leftrightarrow f(\mathbf{x}) > 0 \quad (6.5)$$

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

Dies gilt, da es sich bei  $\mathbf{x}$  um eine Punktmenge mit topologischer Struktur handelt.

Gemäss [DM96] finden hauptsächlich drei Methoden Anwendung zur Beschreibung impliziter Oberflächen: algebraische Oberflächen, Blobby-Objekte sowie die funktionale Repräsentation. [Har94] gibt jedoch an, dass die gebräuchlichste Form von impliziten Oberflächen die algebraischen Oberflächen sind. 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:

$$x^2 + y^2 + z^2 - 1 = 0 \quad (6.6)$$

Wobei es sich bei dem letzten Parameter um den Radius  $r$  handelt, welcher — bedingt durch die Einheitskugel — den Wert 1 hat.

Wie [DM96] schreibt, handelt es sich bei impliziten Oberflächen, welche durch eine polynomiale Funktion zweiten Grades beschrieben werden, um quadratische implizite Oberflächen.

[Har94] gibt weiter an, dass — unter Nutzung einer Metrik — die Einheitskugel durch die implizite Gleichung

$$\|\mathbf{x}\| - 1 = 0 \quad (6.7)$$

beschrieben werden kann, was unter Anwendung der allgemeinen Form 6.2 einer impliziten Gleichung 6.2 zu folgender Gleichung führt:

$$f(\mathbf{x}) = \|\mathbf{x}\| - 1 \quad (6.8)$$

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

Die Gleichung 6.6 gibt die algebraische Distanz zurück, Gleichung 6.7 gibt die geometrische Distanz zurück.

## Distanzfunktionen

Gemäss [Har94] wird die geometrische Darstellung von quadrischen Oberflächen bevorzugt, da deren Parameter unabhängig von Koordinaten sind, sie robuster und intuitiver sind. Es handelt sich dabei um eine **Distanzfunktion**.

Wie anfangs erwähnt, definiert [Har94] die allgemeine Form zur Beschreibung bzw. Darstellung von impliziter Oberflächen als Zuweisung von Punkten zu einem skalaren Wert:  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ .

Unter Anwendung der unter 6.3 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:

$$A = \{x : f(x) \leq 0\} \quad (6.9)$$

[Har94] liefert zwei Definition, 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  $A$ :

$$d(\mathbf{x}, A) = \min_{\mathbf{y} \in A} (\|\mathbf{x} - \mathbf{y}\|) \quad (6.10)$$

### Definition 6.1.2. Signed distance bound

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

$$|f(\mathbf{x})| \leq d(\mathbf{x}, f^{-1}(0)) \quad (6.11)$$

Wenn die Gleichung 6.11 für eine Funktion  $f$  gilt, dann ist  $f$  eine *vorzeichenabhängige Distanzfunktion* (*signed distance function*).

## Distanzfelder (distance fields)

Introduce distance fields?

## 6.2. Darstellung von impliziten Oberflächen

Wie [Har94][S. 1] angibt, existieren verschiedene Möglichkeiten zur Darstellung (zum Rendering) von impliziten Oberflächen. So wandeln indirekte Methoden implizite Oberflächen in Polygonmodelle um, was die Nutzung bestehender Techniken und Hardware zur Darstellung von polygonalen Modellen erlaubt. Obwohl die Umwandlung der impliziten Oberflächen mit gängigen Systemen zur Darstellung problemlos dargestellt werden kann, ist die Umwandlung jedoch nicht in jedem Fall gegeben und kann zu nicht zusammenhängenden Flächen oder einer Verminderung des Detailgrades führen.

Eine andere Methode zur Darstellung von impliziten Oberflächen ist das unter ?? vorgestellte Ray Tracing Verfahren.

Ein (Licht-) Strahl wird dabei parametrisch als



$$r(t) = r_0 + t \cdot r_d \quad (6.12)$$

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  $r(t)$  der Punkt im Raum, welchen der Strahl nach dem Zurücklegen der Distanz  $t$  — ausgehend von seinem Ursprung  $r_0$  — erreicht.

Um nun die Schnittpunkte eines Strahles mit einer impliziten Oberfläche zu finden, wird die Gleichung des Lichtstrahles  $r$  (6.12) in die Funktion einer impliziten Oberfläche  $f$  (6.2) 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  $F : \mathbb{R} \rightarrow \mathbb{R}$ .

Die Lösungen dieser Gleichung sind alle Distanzen  $t$ , welche ein gegebener Strahl zurücklegt und welche die folgende Bedingung erfüllen:

$$F(t) = f \circ r = f(r(t)) = 0 \quad (6.13)$$

Um die Gleichung 6.13 zu lösen, können numerische Verfahren zur Nullstellensuche angewendet werden, wobei die Verfahren vom Typ der Funktion  $F(t)$  abhängig sind. Bei polynomialen Funktionen bis zum vierten Grad existieren analytische Lösungen, für eine beliebige Funktion muss jedoch ein generisches, robustes Verfahren zur Nullstellensuche verwendet werden. Dies bedingt jedoch meist, dass mehr Informationen über die Funktion zur Verfügung stehen, was beispielsweise durch Ableiten dieser gelöst werden kann.

Die erwähnten Verfahren zur Nullstellensuche haben jedoch häufig den Nachteil, dass sie mehrere Schnittpunkte eines Strahles mit einer impliziten Oberfläche liefern. Um diese Problematik zu umgehen, wird nur der kleinste Wert von  $t$  berücksichtigt. Die Ray Marching und Sphere Tracing Algorithmen gehen hier sogar noch einen Schritt weiter, in dem sie nur die kleinste positive Nullstelle der Gleichung 6.13 betrachten.

### 6.2.1. Ray Marching

[PH89] schlagen eine Abtastung des Strahles mit fixen Abständen  $\Delta\mu$  vor:

$$x = x_{\mu_0} + k \cdot \Delta x_\mu \quad (6.14)$$

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

Auf die parametrische Darstellung eines (Licht-) Strahles, Gleichung 6.13, angewendet:

$$r(k) = r_0 + \Delta t \cdot k \cdot r_d \quad (6.15)$$

wobei  $\Delta t$  die Grösse der Abstände und  $k = 0, 1, 2, \dots$  die Nummer der Schritte darstellt. Wie [HSK89] schreiben, bildet das Abtasten des (Licht-) Strahles mit fixen Abständen die Basis für gewisse Verfahren des volumetrischen Renderings.

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

```

def ray_march():
    step = 0
    intersection = 0
    max_steps = 10

    while step < max_steps:
        intersection = test_intersection(k)

        if intersection <= 0:
            # An intersection has happened
            # intersection < 0: ray is inside surface
            # intersection == 0: ray is exactly on surface
            return ray_travel_distance(step)

        step = step + 1

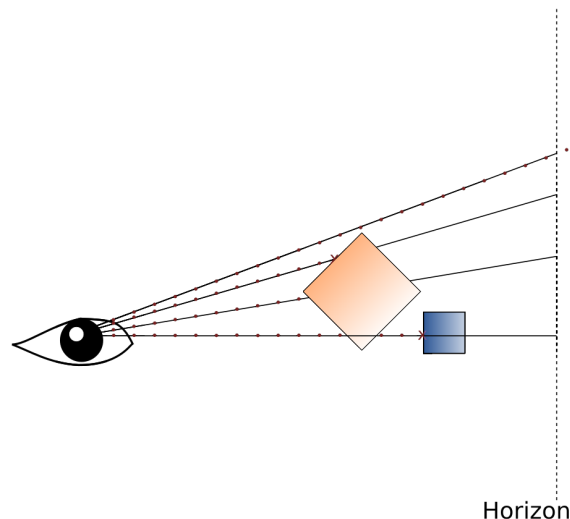
    # When we reach this step, after max_steps, no intersection
    # has happened, so distance is 0
    return 0

```

Auflistung 6.1: Eine abstrakte Umsetzung des Ray Marchings<sup>1</sup>.

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

Abbildung 6.1.: Illustration des Ray Marching Verfahrens und dessen Problemen.<sup>2</sup>



Die Abbildung 6.1 veranschaulicht diese Problematiken. Zu sehen sind vier Primärstrahlen,  $e$ ,  $f$ ,  $g$  und  $h$ , sowie zwei Objekte,  $poly1$  und  $poly2$ .

<sup>1</sup>Algorithmus in Pseudocode gemäss [PH89][S. 259, Abschnitt 3.1]

<sup>2</sup>Eigene Darstellung mittels Geogebra.

Bei den Strahlen  $e$  und  $g$  handelt es sich um “normale” Fälle: Der Strahl  $e$  geht komplett an den Objekten vorbei, das Ray Marching wird also nach Erreichen der maximalen Distanz  $d_{max}$  abgebrochen, der Strahl  $g$  trifft das Objekt *poly1* nach 6 Schritten.

Bei den Strahlen  $f$  und  $h$  handelt es sich um Spezialfälle: Der Strahl  $f$  trifft das Objekt *poly1* nicht, obwohl der Strahl durch das Objekt hindurch geht. Dies geschieht aufgrund der gewählten Distanz zum Abtasten der Strahlen ( $\Delta t$ ). Der Strahl  $h$  trifft das Objekt *poly2*, obwohl er eigentlich das Objekt *poly1* treffen müsste. Das getroffene Objekt *poly2* dürfte so gar nicht zu sehen sein. Dieser Fehler tritt wiederum aufgrund der gewählten Distanz zur Abtastung der Strahlen ( $\Delta t$ ) auf.

[Har94] weist darauf hin, dass Ray Marching durch den möglichst geringen Abstand zwischen den Abtastungen entsprechend langsam und paralleles Abtasten praktisch unumgänglich ist. In der von [Har94] vorgestellten Technik des Sphere Tracings ist der Abstand zwischen den Abtastungen nicht konstant sondern variiert in Abhängigkeit der Geometrie.

### 6.2.2. Sphere Tracing

Das von [Har94] vorgestellte Sphere Tracing Verfahren ist ein Ray Tracing (??) Verfahren für implizite Oberflächen. Es handelt sich nach wie vor um Ray Marching (6.2.1), die Distanz der Schritte zum Abtastens eines (Licht-) Strahles wird jedoch aufgrund einer Distanzfunktion (6.1.1) bestimmt.

[Har94] verweist auf den Term “*unbounding volumes*”, welcher in [HSK89] eingeführt wurde. “Unbounding volumes” (zu Deutsch etwa “negativer Hüllkörper”) wird genutzt um Sphere Tracing zu beschreiben und darzustellen. Der Term steht im Gegensatz zu dem gängigen Konzept des Hüllkörpers (“*bounding volumes*”) — ein Volumen, welches einen Körper umschliesst. Ein “negativer Hüllkörper” (“*unbounding volume*”) umschliesst also eine Fläche im Raum, welche den Körper *nicht* beinhaltet.

Man möchte für einen abzutastenden Punkt im Raum ein Volumen finden, dessen Zentrum im abzutastenden Punkt liegt. Ist der Abstand des Punktes zum nächsten Punkt der Oberfläche eines Objektes bekannt, kann dieser Abstand als Radius einer Kugel angenommen werden. Diese Kugel dient als negativer Hüllkörper (“*unbounding Volume*”) und ist *garantiert nicht* Teil des Objektes und schneidet dieses auch nicht (ist also nicht  $\overset{\circ}{A}$ ) — nur der äusserste Punkt des Abstandes (also des Radius der Kugel) liegt genau auf der Oberfläche des Objektes ( $\partial A$ ). Der Radius solch einer Kugel wird durch Evaluation der Distanzfunktion eines abzutastenden Punktes im Raum bestimmt.

Gemäss [Har94] kommt daher auch der Name Sphere Tracing: Die Schnittpunkte eines (Licht-) Strahles werden durch eine Folge von negativen Hüllkörpern (“*unbounding volumes*”) — bzw. in diesem Fall Kugeln (“*unbounding spheres*”) — beschrieben.

Da [HSK89] die Darstellung von Fraktale im dreidimensionalen Raum beschreibt, wird dort von einer Abschätzung der Distanz gesprochen. Dies ist dadurch bedingt, dass die Distanz für Fraktale nicht effizient berechnet werden kann. Betrachtet man jedoch die Darstellung von “regulären” Objekten, wie zum Beispiel eine Kugel, kann der zur Oberfläche am nächsten gelegene Punkt von einem beliebigen Punkt derselben Domäne exakt berechnet werden. Dies ist durch die implizite Gleichung 6.8 gegeben.

Gemäss [HSK89] verläuft die Strahlenverfolgung bei dem Sphere Tracing Verfahren wie folgt. Ein Strahl wird vom Betrachter (Auge bzw. Lochkamera) durch die Bildebene zu einem Objekt geschossen. Dabei wird beim initialen Ausgangspunkt der Radius eines negativen Hüllkörpers in Form einer Kugel — so wie oben beschrieben — berechnet. Dies ist die Distanz, welcher der Strahl in einem ersten Schritt effektiv zurücklegen wird. Bei jedem Schnittpunkt der Kugel mit dem Strahl wird dasselbe Verfahren wiederholt. Dies geschieht so lange, bis schliesslich der Strahl durch einen Schnittpunkt mit einem Radius auf die Oberfläche des Objektes trifft. Ein weiteres Abbruchkriterium ist eine definierte maximale Distanz eines Strahles. Ist diese erreicht und der Strahl erreicht die Oberfläche des Objektes nicht — weil der Strahl das Objekt nicht schneidet oder das Objekt zu weit weg ist —, wird abgebrochen. Somit ist auch ersichtlich, dass Sphere Tracing nicht die unter 6.2.1 genannten Problematiken aufweist.

Abbildung 6.2.: Illustration des Sphere Tracing Verfahrens.<sup>3</sup>

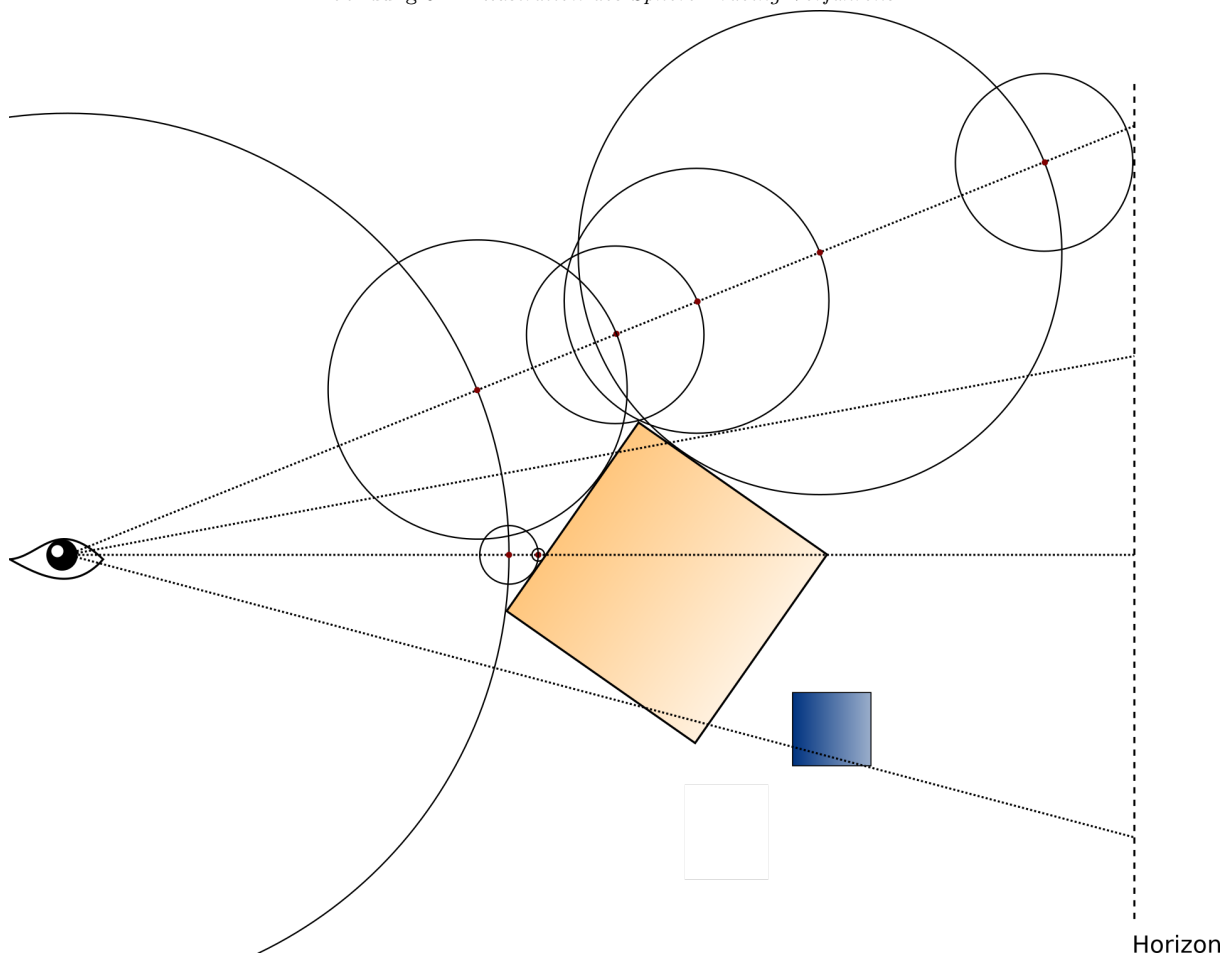
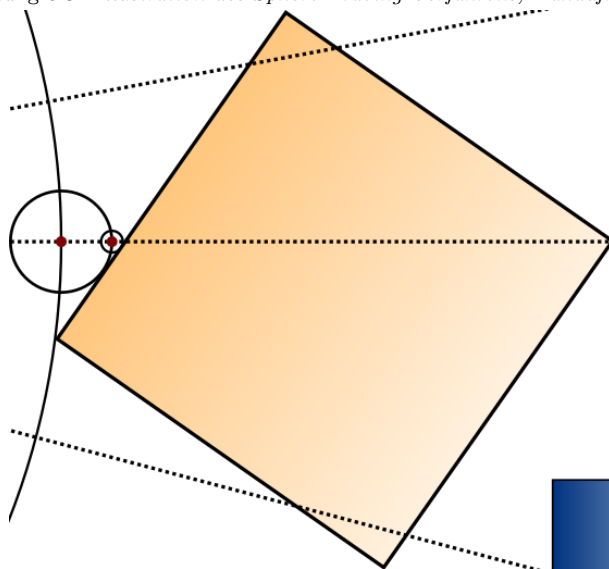
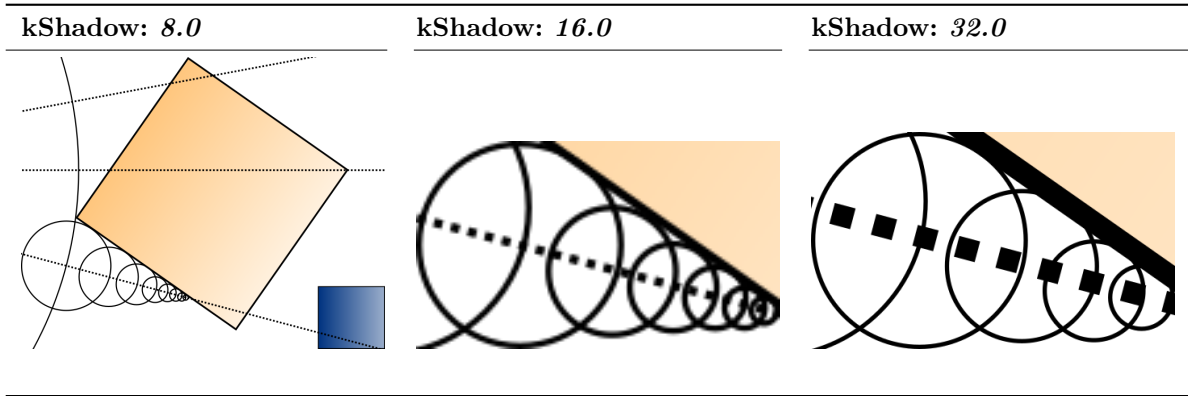


Abbildung 6.3.: Illustration des Sphere Tracing Verfahrens, Nahaufnahme.<sup>4</sup>



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

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

Tabelle 6.1.: *INSERT CAPTION HERE*<sup>5</sup>

Ausgehend von der parametrischen Beschreibung eines (Licht-) Strahles (Gleichung 6.12), beschreibt [Har94] die Richtung  $r_d$  eines Strahles als Einheitsvektor:

$$r_d = \frac{p_{x,y} - r_0}{|p_{x,y} - r_0|} \quad (6.16)$$

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

Um nun den Schnittpunkt eines Strahles  $r_d$  mit der Oberfläche eines Objektes zu finden, muss die Gleichung  $F(t)$  (6.13) gelöst werden. Dabei ist — wie oben definiert — die Funktion  $f(x)$  nun eine Distanzfunktion wie die geometrische Distanzfunktion zur Beschreibung einer Kugel (Gleichung 6.7).

Evaluert man nun die Gleichung  $F(t)$  unter Anwendung der eben beschriebenen Strahlenverfolgung, 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:

$$t_{i+1} = t_i + F(t_i) \quad (6.17)$$

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.

```
def sphere_trace():
    ray_distance = 0
    estimated_distance = 0
    max_distance = 9001
    convergence_precision = 0.000001

    while ray_distance < max_distance:
        # sd_sphere is a signed distance function defining the implicit surface
        # cast_ray defines the ray equation given the current travelled /
        # marched distance of the ray
        estimated_distance = sd_sphere(cast_ray(ray_distance))

        if estimated_distance < convergence_precision:
            # the estimated distance is already smaller than the desired
            # precision of the convergence, so return the distance the ray has
```

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

```

    # travelled as we have an intersection
    return ray_distance

    ray_distance = ray_distance + estimated_distance

    # When we reach this point, there was no intersection between the ray and a
    # implicit surface, so simply return 0
    return 0

```

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

### 6.2.3. Operationen für implizite Oberflächen

Um mit impliziten Oberflächen nicht nur einfache Objekte, wie zum Beispiel eine Kugel darzustellen, möchte man diese auch transformieren können.

Wie [Har94][S. 543] beschreibt, werden implizite Oberflächen durch die Invertierung des Raumes, in welchem sich die Oberfläche befindet, transformiert. 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:

$$f(T^{-1}(\mathbf{x})) = 0 \quad (6.18)$$

Bei Transformationen handelt es sich um vorzeichenabhängige Distanzfunktion (signed distance functions).

Es werden dabei folgende Arten von Transformationen unterschieden:

- Distanzoperationen, wie zum Beispiel Vereinigung, Subtraktion oder Intersektion
- Domänenoperationen, wie zum Beispiel Wiederholung, Rotation, Translation und Skalierung
- Distanzdeformationen, wie zum Beispiel Versatz (displacement) und Vermengung/Vermischung (blend)
- Domänenendeformationen, wie zum Beispiel “Verderung” (twist) und Biegung (bend)

#### Isometrien

Nicht alle Transformationen erhalten dabei die Distanz, welche die Distanzfunktion der transformieren 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 hingegen distanzerhaltend sind, bezeichnet [Har94] als *Isometrien*. Dazu zählen Rotationen, Translationen aber auch Reflektionen.

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

$$d(\mathbf{x}, \mathbf{I} \circ f^{-1}(0)) = d(\mathbf{I}^{-1}(\mathbf{x}), f^{-1}(0)) \quad (6.19)$$

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

<sup>6</sup>Algorithmus in Pseudocode gemäss [Har94][S. 531, Fig. 1]

## Uniforme Skalierung

Eine Skalierung ist *nicht* distanzerhaltend, daher *erhält* sie die Distanz, welche die Distanzfunktion der skalierten Oberfläche zurückgeben würde, *nicht*. Somit muss die zurückgegebene Distanz entsprechend angepasst werden.

[Har94] gibt die uniforme Skalierung als Transformation  $\mathbf{S}(\mathbf{x})$  der Form:

$$\mathbf{S}(\mathbf{x}) = s \cdot \mathbf{x} \quad (6.20)$$

an, wobei  $s$  der Skalierungsfaktor ist. Die Invertierung der Skalierung ist gegeben als:

$$\mathbf{S}^{-1}(\mathbf{x}) = \frac{1}{s} \cdot \mathbf{x} \quad (6.21)$$

Somit ist die Distanz zu der skalierten impliziten Oberfläche:

$$d(\mathbf{x}, \mathbf{S}(f^{-1}(0))) = s \cdot d(\mathbf{S}^{-1}(\mathbf{x}), f^{-1}(0)) \quad (6.22)$$

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

## “Verdrehung” (Twist)

Gemäss [Har94][S. 543] werden bei der “Verdrehung” 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:

$$\text{twist}(\mathbf{x}) = \begin{pmatrix} x \cdot \cos a(z) - y \cdot \sin a(z), \\ x \cdot \sin a(z) + y \cdot \cos a(z), \\ z \end{pmatrix} \quad (6.23)$$

## Vereinigung

Die Vereinigung zweier impliziter Oberflächen  $A$  und  $B$  wird von [Har94] als minimale Distanz der jeweiligen vorzeichenabhängigen Distanzfunktion  $f_A$  respektive  $f_B$  definiert:

$$d(\mathbf{x}, A \cup B) = \min(f_A(\mathbf{x}), f_B(\mathbf{x})) \quad (6.24)$$

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

Wie ?? schreibt, ist die Distanz zu einer Liste von impliziten Oberflächen die kleinste Distanz der jeweiligen Distanzfunktion. Somit erlaubt es die Vereinigung — neben der Kombination von Objekten — mehrere implizite Oberflächen zu kombinieren, ohne dass diese miteinander in Kontakt stehen müssen. So kann beispielsweise eine komplexe Szene modelliert werden.

## Subtraktion

Um die Operation der Subtraktion zu definieren, wird die Distanz zum Komplement eines Objektes  $A$  verwendet. Dabei wird die Eigenschaft der Vorzeichenabhängigkeit von vorzeichenabhängigen Distanzfunktionen genutzt:

$$d(\mathbf{x}, \mathbb{R}^3 \setminus A) = -f_A(\mathbf{x}) \quad (6.25)$$

Somit kann die Subtraktion zweier impliziter Oberflächen  $A$  und  $B$  gemäss [Har94] als Intersektion eines Objektes  $A$  mit der Subtraktion des Raumes bzw. der Domäne mit einem Objekt  $B$  angesehen werden, daher folgt:

$$d(\mathbf{x}, A - B) = A \cap (\mathbb{R}^3 \setminus B) \quad (6.26)$$

$$\geq \max(f_A(\mathbf{x}), -f_B(\mathbf{x})) \quad (6.27)$$

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

## Intersektion

Die Intersektion zweier impliziter Oberflächen  $A$  und  $B$  wird von [Har94] als minimale Distanz der jeweiligen vorzeichenabhängigen Distanzfunktion  $f_A$  respektive  $f_B$  definiert:

$$d(\mathbf{x}, A \cap B) \geq \max(f_A(\mathbf{x}), f_B(\mathbf{x})) \quad (6.28)$$

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

### 6.2.4. Primitive

[Har94] führt in seinem Paper einige (geometrische) Primitive auf, welche nachfolgende erläutert werden.

#### Ebene

Die vorzeichenbehaftete Distanz zu einer Ebene  $P$  mit einer Einheitsnormalen  $\mathbf{n}$ , welche sich mit dem Punkt  $\mathbf{n} \cdot r$  schneidet ist wie folgt:

$$d(\mathbf{x}, P) = \mathbf{x} \cdot \mathbf{n} - r \quad (6.29)$$

wobei  $r$  die relative Positionierung der Ebene — unter Einbezug der Normalen der Ebene im Verhältnis zur Einheitsnormalen  $\mathbf{n}$  — im Raum darstellt.

#### Kugel

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

$$d(\mathbf{x}, S) = \|\mathbf{x}\| - r \quad (6.30)$$

wobei  $r$  den Radius der Kugel darstellt.



## Zylinder

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

$$d(\mathbf{x}, Cyl) = \|(x, y)\| - r \quad (6.31)$$

wobei  $r$  den Radius des Zylinders darstellt,  $\mathbf{x}$  stellt dabei  $(x, y, z)$  dar.

## Kegel

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

$$d(\mathbf{x}, Cone) = \|(x, y)\| \cdot \cos(\phi) - |z| \cdot \sin(\phi) \quad (6.32)$$

wobei  $\phi$  den Winkel zur Z-Achse darstellt.

## Torus

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

$$d(\mathbf{x}, T) = \|(\|(x, y)\| - R, z)\| - r \quad (6.33)$$

wobei  $R$  den Aussenradius und  $r$  den Innenradius des Torus darstellt. Der Torus ist am Ursprung zentriert und dreht sich um die Z-Achse.

## 6.3. Rendering von impliziten Oberflächen

Add an introduction

### 6.3.1. Beleuchtungsmodell

Explain lighting model

Um implizite Oberflächen darstellen zu können, ist es notwendig ein Beleuchtungsmodell zu wählen. Ansonsten wäre das dargestellte Bild nur schwarz. Der Einfachheit halber wird im Rahmen dieser Projektarbeit das in Kapitel 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:

$$I(\mathbf{x}) = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \quad (6.34)$$

Wie bereits zuvor in Kapitel 5.1.1 erwähnt, wird der emissive Term bewusst weggelassen, da keine emittiven Materialien 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 *ambient Anteil*  $I_{\text{ambient}}$  ergibt sich dann wie folgt:

$$I_{\text{ambient}} = k_{\text{ambient}}(\mathbf{x}) \cdot L_{\text{ambient}} \quad (6.35)$$

wobei  $k_{\text{ambient}}(\mathbf{x})$  den ambienten Faktor des Punktes  $\mathbf{x}$  und  $L_{\text{ambient}}$  die Farbe des eingehenden ambienten Lichtes ist.

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

$$I_{\text{diffuse}} = k_{\text{diffuse}}(\mathbf{x}) \cdot L_{\text{diffuse}} \cdot \max(\mathbf{n} \cdot \mathbf{l}, 0) \quad (6.36)$$

wobei  $k_{\text{diffuse}}(\mathbf{x})$  den diffusen Faktor am Punkt  $\mathbf{x}$  und  $L_{\text{diffuse}}$  die Farbe des eingehenden diffusen Lichtes ist. Die Richtung der Lichtquelle, ausgehend von Punkt  $\mathbf{x}$ , ergibt sich durch das Punktprodukt zwischen der Einheitsnormalen  $\mathbf{n}$  des Punktes und dem Einheitsvektor  $\mathbf{l}$ .

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

$$I_{\text{specular}} = n_{\text{facing}} \cdot k_s(\mathbf{x}) \cdot L_{\text{specular}} \cdot \max(\mathbf{n} \cdot \mathbf{h}, 0)^{k_e} \quad (6.37)$$

wobei  $k_{\text{specular}}(\mathbf{x})$  den reflektierenden Faktor des Punktes  $\mathbf{x}$  und  $L_{\text{specular}}$  die Farbe des eingehenden reflektierenden Lichtes ist. 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. Der Exponent  $k_e$  gibt an, wie rau bzw. wie spiegelnd die Oberfläche am Punkt  $\mathbf{x}$  ist. Der Faktor  $n_{\text{facing}}$  definiert, ob die Oberfläche überhaupt einen reflektierenden Anteil hat:

$$n_{\text{facing}} = \begin{cases} 0 & \text{if } \mathbf{n} \cdot \mathbf{l} \leq 0 \\ 1 & \text{if } \mathbf{n} \cdot \mathbf{l} > 0 \end{cases} \quad (6.38)$$

Für die Berechnung der Lichtintensität bzw. der Farbe einer Oberfläche wird die Normale der Oberfläche benötigt. Gemäss [HSK89] kann diese mittels des Gradienten des Distanzfeldes eines Punktes einer impliziten Oberfläche berechnet werden:

$$\mathbf{n}_x = f(x + \varepsilon, y, z) - f(x - \varepsilon, y, z) \quad (6.39)$$

$$\mathbf{n}_y = f(x, y + \varepsilon, z) - f(x, y - \varepsilon, z) \quad (6.40)$$

$$\mathbf{n}_z = f(x, y, z + \varepsilon) - f(x, y, z - \varepsilon) \quad (6.41)$$

$$(6.42)$$

wobei  $\mathbf{n} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}$  die Normale der Oberfläche in Form eines Vektors, und  $f$  eine Distanzfunktion ist.

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

$$\text{grad}f = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad (6.43)$$

$$\text{grad}f = f_x \mathbf{i} + f_y \mathbf{j} + f_z \mathbf{k} \quad (6.44)$$

$$(6.45)$$

[HSK89] gibt dabei  $\varepsilon$  als die minimale Inkrementation eines (Licht-) Strahles an und definiert diesen als Sichtbarkeitsfunktion  $\Gamma_{\alpha,\delta}$ :

$$\Gamma(d) = \alpha d^\delta \quad (6.46)$$

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

$$d = |r_{\mathbf{n}} - r_0| \quad (6.47)$$

wobei  $\delta$  ein so genannter “depth-cueing”-Exponent (“depth-cueing” oder auch “foldback”, “a process for returning a signal to a performer instantly” [lia15]) und  $\alpha$  ein empirischer Anteil, welcher die Tiefenauflösung des Objektes definiert, ist. Details dazu finden sich unter [HSK89][S. 293, Abschnitt 4.2 — “Clarity”].

Es folgt also:

$$\varepsilon = \Gamma_{\alpha,\delta}(|r_{\mathbf{n}} - r_0|) \quad (6.48)$$

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.

Die Normale der Oberfläche sollte schliesslich noch normalisiert werden.

[HSK89] schreibt weiter, dass die oben genannte Gradienten, bestehend aus 6 Punkten, durch Hinzunahme von Punkten, welche eine gemeinsame Kante haben, erweitert werden kann. 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. Dies macht jedoch nur dann Sinn, wenn die Gradienten mit 6 Punkten eine unzureichende Genauigkeit liefert.

### 6.3.2. Rendering

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

Zum eigentlichen Rendern wird der Algorithmus 6.2 mit dem unter Abschnitt 6.3.1 angegebenen Beleuchtungsmodell angewendet.

### 6.3.3. Schatten

Sofern nicht anders vermerkt, basiert folgender Abschnitt auf [RMD11][S. 7].

Mittels Sphere Tracing können Schatten analog den vom Ray Tracing bekannten Verfahren gewonnen werden. Dazu werden Schattenfühler (“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.

$$r_s(t) = \mathbf{x} + t \cdot r_l \quad (6.49)$$

wobei  $r_s(t)$  der Ursprung des Schattenfühlers am Punkt  $\mathbf{x}$  einer impliziten Oberfläche und  $r_l$  die Richtung des Schattenfühlers in Form eines Einheitsvektors ist.

```
def calc_shadows():
    min_distance = 0.01
    max_distance = 9001
    shadow_ray_distance = min_distance
    estimated_distance = 0
    convergence_precision = 0.000001

    while shadow_ray_distance < max_distance:
        # sd_sphere is a signed distance function defining the implicit surface
        # cast_ray defines the ray equation given the current travelled /
        # marched distance of the ray
        estimated_distance = sd_sphere(cast_ray(shadow_ray_distance))

        if estimated_distance < convergence_precision:
            # the estimated distance is already smaller than the desired
            # precision of the convergence, so return zero (0) as we
            # have an intersection and therefore shadows
            return 0

        shadow_ray_distance = shadow_ray_distance + estimated_distance

    # When we reach this point, there was no intersection between the ray and a
    # implicit surface, so no shadows, so simply return 1
    return 1
```

Auflistung 6.3: Algorithmus zur Berechnung von Schatten.

Der Algorithmus 6.3 ist dem des Sphere Tracings 6.2 sehr ähnlich, der Rückgabewert ist jedoch komplett verschieden. Der Algorithmus gibt den Wert 1 zurück, wenn keine Intersektion zwischen dem Schattenfühler und einer Oberfläche stattfand nachdem die maximale Distanz erreicht wurde. Der Wert 0 wird zurückgegeben, wenn der Schattenfühler auf eine Oberfläche getroffen ist und der Punkt  $\mathbf{x}$  einer impliziten Oberfläche sich daher im Schatten befindet.

Um zu verhindern, dass ein Punkt  $\mathbf{x}$  einer impliziten Oberfläche sich selbst verdeckt (sich 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  $\mathbf{x}$  daher immer im Schatten befinden würde.

(author?) geben zudem an, wie weiche Schatten mittels Sphere Tracing relativ einfach dargestellt werden können. Während der Expansion anhand des Schattenfühlers wird die minimale, evaluierte Distanz  $d_{\min}$  zu einem beliebigen Objekt gespeichert. Es wird dabei angenommen, dass sich ein Punkt  $\mathbf{x}$  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 Speichern der minimalen, evaluierten Distanz ist es möglich die Distanz des Schattenfühlers zu der ihn umgebenden Geometrie zu untersuchen und somit die Penumbra-Region sowie weiche Schatten zu bestimmen. Befindet sich der Schattenfühler in der Nähe einer Oberfläche, schneidet diese aber nicht, ist die minimale Distanz  $d_{\min}$  sehr gering. Ist dies der Fall, so kann angenommen werden, dass sich der Ursprungspunkt  $\mathbf{x}$  in einer Penumbra-Region befindet, was sich auf die Schattierung des Punktes auswirkt: Je knapper der Schattenfühler 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.

```
def calc_soft_shadows():
    min_distance = 0.01
    max_distance = 9001
    shadow_ray_distance = min_distance
    estimated_distance = 0
    convergence_precision = 0.000001
    shadow = 1.0
    scale = 32.0

    while shadow_ray_distance < max_distance:
        # sd_sphere is a signed distance function defining the implicit
        # surface, # cast_ray defines the ray equation given the current travelled /
        # marched distance of the ray
        estimated_distance = sd_sphere(cast_ray(shadow_ray_distance))

        if estimated_distance < convergence_precision:
            # the estimated distance is already smaller than the desired
            # precision of the convergence, so return zero (0) as we
            # have an intersection and therefore 'full' shadow
            return 0

        penumbra_factor = estimated_distance / shadow_ray_distance
        shadow = min(shadow, scale * penumbra_factor)
        shadow_ray_distance = shadow_ray_distance + estimated_distance

    # When we reach this point, there was no full intersection between
    # the ray and a implicit surface, so not entirely shadowed, so
    # return current shadow value
    return shadow
```

Auflistung 6.4: Algorithmus zur Berechnung von weichen Schatten.

## 7. Prototyp

Comment source code of prototype

Um das Sphere Tracing Verfahren nicht nur theoretisch zu behandeln, wird im Rahmen dieser Projektarbeit ein Prototyp erstellt. Dieser wird in C++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 können (Dateien mit Dateiendung .vs bzw. .fs) werden die Unterbibliotheken “system”, “filesystem” sowie “regex” der Boost-Bibliothek eingesetzt.

Details der einzelnen Komponenten finden sich in der nachfolgenden Tabelle.

Komponente	Version	Beschreibung	Verweise
C++	C++11	Objektorientierte Programmiersprache	1
OpenGL	4.5	Plattformunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafikanwendungen [Wik15d]	2
GLFW	3.1.2	OpenGL-Bibliothek, welche die Erstellung und Verwaltung von Fenstern sowie OpenGL-Kontexte vereinfacht [Wik15c]	3
GLEW	1.13	OpenGL Extension Wrangler. Bibliothek zum Abfragen und Laden von OpenGL-Erweiterungen (Extensions) [Wik15e]	4
CMake	3.3.2	Software zur Verwaltung von Build-Prozessen von Software	5
GCC	5.2	GNU Compiler Collection. Compiler-System des GNU-Projektes	6
Boost	1.59.0	Freie Bibliothek bestehend aus einer Vielzahl von Unterbibliotheken, die den unterschiedlichsten Aufgaben von Algorithmen auf Graphen über Metaprogrammierung bis hin zu Speicherverwaltung dienen [Wik15b]	7

### 7.1. Architektur

Die Architektur des Protoypen ist in der untenstehenden Abbildung 7.1 ersichtlich.

<sup>1</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>2</sup><https://www.opengl.org/registry/doc/glspec45.core.pdf>

<sup>3</sup><http://www.glfw.org>

<sup>4</sup><http://glew.sourceforge.net>

<sup>5</sup><https://www.cmake.org>

<sup>6</sup><http://gcc.gnu.org>

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

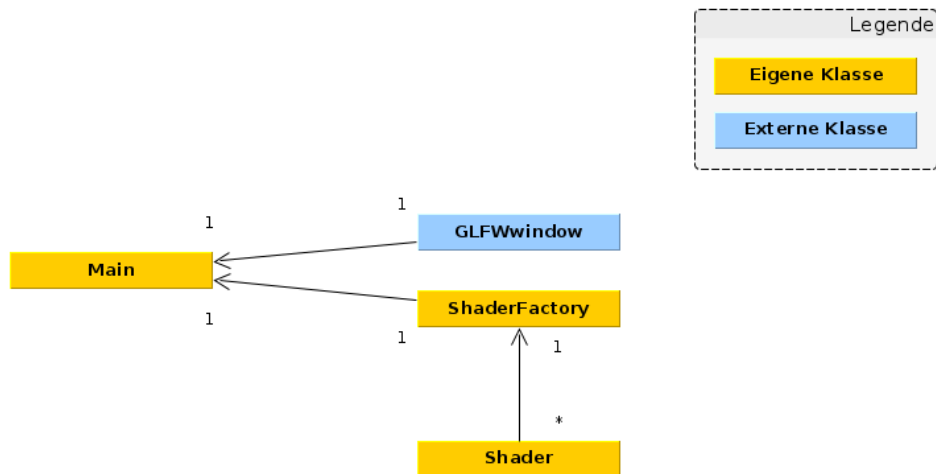


Abbildung 7.1.: Architektur des Prototypen<sup>8</sup>

### 7.1.1. Programmablauf

Wie in Abbildung 7.1 ersichtlich ist, besteht die Applikation hauptsächlich aus der Hauptklasse. Diese initialisiert mittels GLEW OpenGL und erstellt mittels GLFW ein Fenster sowie einen OpenGL-Kontext. Danach wird eine Instanz des ShaderManagers erstellt, welche ihrerseits alle verfügbaren GLSL-Shader in einem gegebenen Verzeichnis lädt. Bei diesem Prototypen kommt jedoch nur ein Shader zum Einsatz — bestehend aus einem Vertex- sowie einem Fragment-Teil.

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

Die Hauptidee der Applikation ist die, dass diese im Rendering-Teil einen Vertex- sowie einen Fragment-Shader lädt und ausführt. Der Vertex-Shader tut nichts anderes als zwei Polygone in Form von Dreiecken über die verfügbare Bildfläche darzustellen. Das eigentliche Rendering von impliziten Oberflächen geschieht dann im Fragment-Shader. Dies ist in der untenstehenden Abbildung 7.2 verdeutlicht.

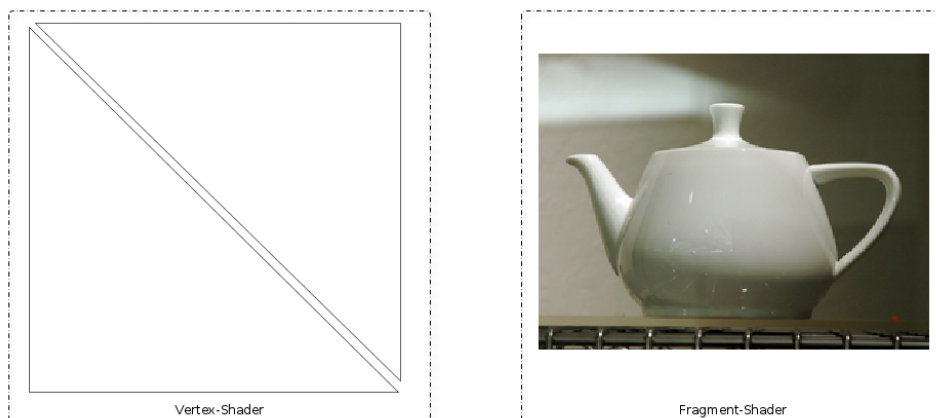


Abbildung 7.2.: Bildliche Darstellung der Funktionsweise von Vertex- und Fragmentshader der Applikation<sup>9</sup>

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

<sup>9</sup>Eigene Darstellung mittels yEd. Bei dem Bild des Fragment-Shaders handelt es sich um den so genannten “Utah Teapot”, bezogen von [http://www.flickr.com/photo\\_zoom.gne?id=352811902&size=o](http://www.flickr.com/photo_zoom.gne?id=352811902&size=o), alle Rechte für das Bild liegen bei Marshall Astor (<http://www.marshallastor.com/>).

## 7.2. Umsetzung

Nachfolgend werden die gemäss Kapitel 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.

```
vec2 castRay(in vec3 rayOrigin, in vec3 rayDirection, in float maxDistance, in float ←
precision, in int steps)
{
    float latest = precision * 2.0;
    float distance = 0.0;
    float type = -1.0;
    vec2 res = vec2(-1.0, -1.0);

    for(int i = 0; i < steps; i++) {
        if (abs(latest) < precision || distance > maxDistance) {
            continue;
        }

        vec2 result = scene(rayOrigin + rayDirection * distance);

        latest = result.x;
        type = result.y;
        distance += latest;
    }

    if (distance < maxDistance) {
        res = vec2(distance, type);
    }

    return res;
}
```

Auflistung 7.1: Umsetzung des Sphere Tracings in GLSL.



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

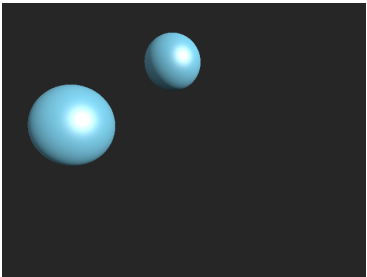
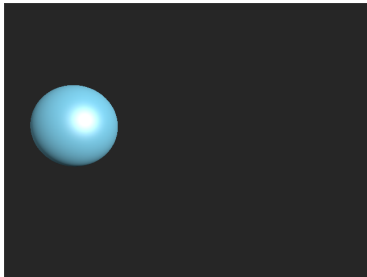
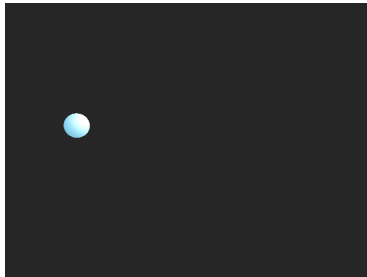
Distanz: 100.0	Distanz: 9.0	Distanz: 6.3
		
Alle in der Szene definierten Objekte sind sichtbar.	Es ist nur noch das zum Betrachter näher liegende Objekt sichtbar.	Es ist nur noch das zum Betrachter näher liegende Objekt sichtbar, jedoch nicht mehr vollständig.

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

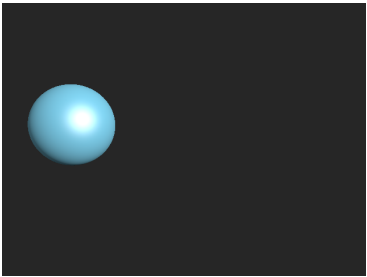
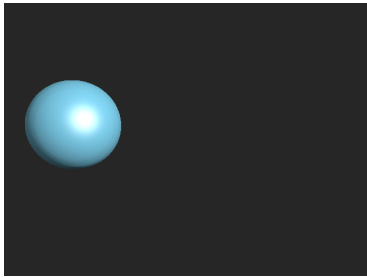
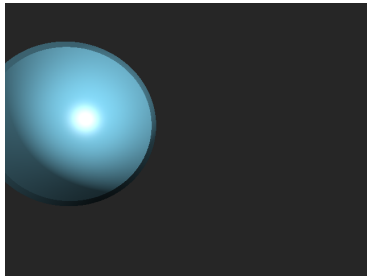
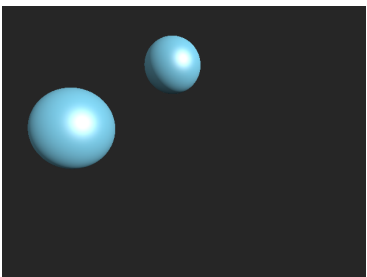
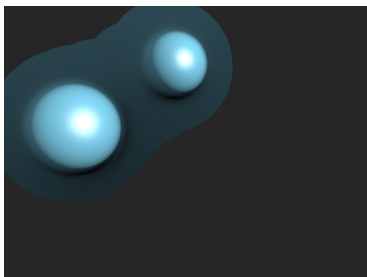
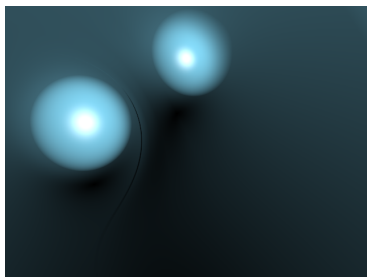
Präzision: 0.00001	Präzision: 0.1	Präzision: 1.0
		
Die Kugel weist keinerlei sichtbare Abstufungen auf.	Die Kugel weist am unteren linken Rand sichtbare Farbabstufungen auf.	Die Kugel wird nicht mehr korrekt dargestellt.

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 mehr.

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

```

float subtract(float a, float b)
{
    return max(-b, a);
}

vec2 union(vec2 a, vec2 b)
{
    return (a.x < b.x) ? a : b;
}

vec2 intersect(vec2 a, vec2 b)
{
    return (a.x > b.x) ? a : b;
}

```

Auflistung 7.2: Umsetzung von Operationen für implizite Oberflächen in GLSL.

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

```

float plane(vec3 position)
{
    return position.y;
}

float sphere(vec3 position, float radius)
{
    return length(position) - radius;
}

float box(vec3 position, vec3 dimension)
{
    position = abs(position) - dimension;
    return max(max(position.x, position.y), position.z);
}

```

Auflistung 7.3: Umsetzung von Primitiven in Form von impliziten Oberflächen in GLSL.

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

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

```

vec3 calcLighting(vec3 normal, vec3 rayDirection) {

    vec3 lightDirection = normalize(vec3(0.0, 4.0, 5.0));

    vec3 ambientColor = vec3(0.05, 0.15, 0.2);
    float kAmbient = clamp(0.5 + 0.5 * normal.y, 0.0, 1.0);
    vec3 ambient = kAmbient * ambientColor;

    vec3 diffuseColor = vec3(0.2, 0.6, 0.8);
    float kDiffuse = clamp(dot(lightDirection, normal), 0.0, 1.0);
    vec3 diffuse = kDiffuse * diffuseColor;
}

```

```

vec3 specularColor = vec3(1.0);
float kSpecularExponent = 24.0;
vec3 h = normalize(-rayDirection + lightDirection);
float nFacing = clamp(dot(lightDirection, normal), 0.0, 1.0);
float kSpecular = pow(clamp(dot(h, normal), 0.0, 1.0), kSpecularExponent);
vec3 specular = nFacing * kSpecular * specularColor;

vec3 color = ambient + diffuse + specular;

return color;
}

```

Auflistung 7.4: Umsetzung des Phong-Beleuchtungsmodelles 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 ist, 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 Kapitel 6.3.1 beschrieben berechnet.

```

vec3 calcNormal(in vec3 position, in float eps) {

    vec3 epsilon = vec3(eps, 0.0, 0.0);
    vec3 normal = vec3(
        scene(position + epsilon.xyy).x - scene(position - epsilon.xyy).x,
        scene(position + epsilon.yxy).x - scene(position - epsilon.yxy).x,
        scene(position + epsilon.yyx).x - scene(position - epsilon.yyx).x
    );

    return normalize(normal);
}

```

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

Verwendete Parameter sind dabei 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 verwendet.

Bei der Funktion *scene* handelt es sich um eine Distanzfunktion  $f$  gemäss Kapitel 6.1.1. Diese definiert schliesslich, was an einem gegebenen Punkt dargestellt wird.

```

vec3 scene(in vec3 position) {
{
    float sphereId = 0.0;
    float sphereRadius = 1.0;
    vec3 sphereOffset = vec3(-2.0, 1.0, -2.0);

    vec2 res = vec2(
        sphere(position - sphereOffset, sphereRadius),
        sphereId
    );

    return res;
}

```

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 Kapitel 6.3.3 beschrieben, wurden weiche Schatten implementiert.

```
float calcShadows(in vec3 rayOrigin, in vec3 rayDirection)
{
    float shadow = 1.0;
    float minimalDistance = 0.01;
    float maximalDistance = 2.5;
    float convergePrecision = 0.000001;
    float kShadow = 8.0;
    float currentDistance = minimalDistance;

    while (currentDistance < maximalDistance) {
        vec3 ray = rayOrigin + rayDirection * currentDistance;
        float estimatedDistance = scene(ray);

        if (estimatedDistance < convergePrecision) {
            return 0.0;
        }

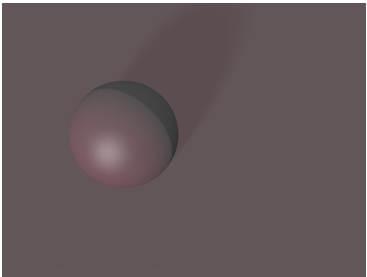
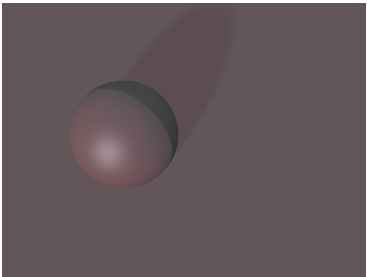
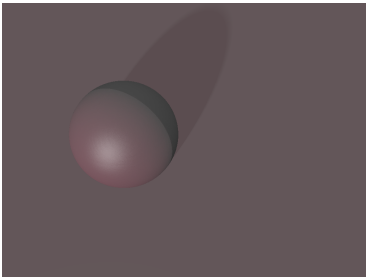
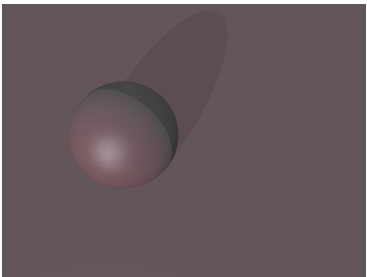
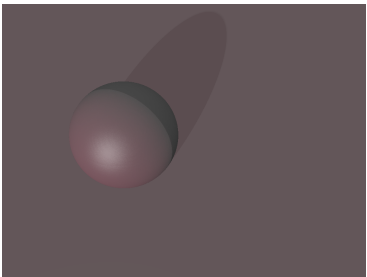
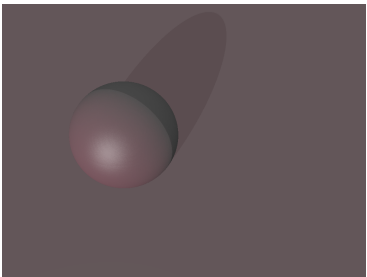
        float penumbraFactor = estimatedDistance / currentDistance;
        shadow = min(shadow, kShadow * penumbraFactor);
        currentDistance += estimatedDistance;
    }

    return shadow;
}
```

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

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

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

<b>kShadow: 8.0</b>	<b>kShadow: 16.0</b>	<b>kShadow: 32.0</b>
		
<b>kShadow: 64.0</b>	<b>kShadow: 128.0</b>	<b>kShadow: 256.0</b>
		

## 8. Diskussion und Fazit

### 8.1. Diskussion

### 8.2. Erweiterungsmöglichkeiten

### 8.3. Fazit

# Literaturverzeichnis

- [App68] Arthur Appel. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.
- [Arl68] Arlington Mathematical Applications Group, Inc. *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. ACM, New York, NY, USA, 1968. Library of Congress Catalog Card No.: 55-44701.
- [DM96] International Business Machines Corporation Research Division and J. Menon. *An Introduction to Implicit Techniques*. Research report. IBM T.J. Watson Research Center, 1996.
- [Fol96] J.D. Foley. *Computer Graphics: Principles and Practice*. Addison-Wesley systems programming series. Addison-Wesley, 1996.
- [Gla89] Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Ltd., London, UK, UK, 1989.
- [Har93] John C Hart. Ray tracing implicit surfaces. *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.
- [Har94] John C. Hart. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer*, 12:527–545, 1994.
- [HSK89] J. C. Hart, D. J. Sandin, and L. H. Kauffman. Ray Tracing Deterministic 3-D Fractals. *SIGGRAPH Comput. Graph.*, 23(3):289–296, July 1989.
- [HVDFF13] J.F. Hughes, A. Van Dam, J.D. Foley, and S.K. Feiner. *Computer Graphics: Principles and Practice*. The systems programming series. Addison-Wesley, 2013.
- [Kaj86] James T. Kajiya. The Rendering Equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.
- [lia15] Iam Collins Sons & Co. Ltd. Collins English Dictionary - Complete & Unabridged 10th Edition, October 2015.
- [PH89] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3):253–262, July 1989.
- [RMD11] Tim Reiner, Gregor Mückl, and Carsten Dachsbacher. SMI 2011: Full Paper: Interactive Modeling of Implicit Surfaces Using a Direct Visualization Approach with Signed Distance Functions. *Comput. Graph.*, 35(3):596–603, June 2011.
- [Whi80] Turner Whitted. An Improved Illumination Model for Shaded Display. *Commun. ACM*, 23(6):343–349, June 1980.
- [Wik15a] Wikipedia Foundation. Zotero, August 2015. Published: Website Abgerufen am 27. September 2015.
- [Wik15b] Wikipedia, the free encyclopedia. Boost (C++-Bibliothek), September 2015. Page Version ID: 146313866.
- [Wik15c] Wikipedia, the free encyclopedia. GLFW, October 2015. Page Version ID: 687716464.
- [Wik15d] Wikipedia, the free encyclopedia. OpenGL, October 2015. Page Version ID: 146904140.
- [Wik15e] Wikipedia, the free encyclopedia. OpenGL Extension Wrangler Library, September 2015. Page Version ID: 680716273.

# Abbildungsverzeichnis

4.1. Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Kalenderwochen dar . . . . .	5
5.1. Illustration des Phong-Beleuchtungsmodelles <sup>1</sup> . . . . .	8
5.2. Punkt $P$ auf einer Oberfläche eines Dreieckes, welcher für die Kamera bzw. den Betrachter sichtbar ist. Der Betrachter nimmt dabei das Licht, welches aus verschiedenen Richtungen $\omega_i$ kommt, über den Punkt $P$ in Richtung $\omega_0$ wahr. <sup>2</sup> . . . . .	10
5.3. Illustration des Ray Tracing Verfahrens <sup>3</sup> . . . . .	13
6.1. Illustration des Ray Marching Verfahrens und dessen Problemen. <sup>4</sup> . . . . .	20
6.2. Illustration des Sphere Tracing Verfahrens. <sup>5</sup> . . . . .	22
6.3. Illustration des Sphere Tracing Verfahrens, Nahaufnahme. <sup>6</sup> . . . . .	22
7.1. Architektur des Prototypen <sup>7</sup> . . . . .	33
7.2. Bildliche Darstellung der Funktionsweise von Vertex- und Fragmentshader der Applikation <sup>8</sup>	33



# Tabellenverzeichnis

6.1. INSERT CAPTION HERE <sup>9</sup> . . . . .	23
7.1. Vergleich des Distanz-Parameters anhand einer Beispielszene. . . . .	35
7.2. Vergleich des Präzisions-Parameters anhand einer Beispielszene. . . . .	35
7.3. Vergleich des Parameters zur Bestimmung der Anzahl der Durchgänge anhand einer Beispielszene. . . . .	35
7.4. Vergleich des Skalierungsfaktors <i>kShadow</i> für Schatten anhand einer Beispielszene. . . . .	39

# Auflistungsverzeichnis

5.1. Eine abstrakte Umsetzung des Ray Castings <sup>10</sup> . . . . .	12
6.1. Eine abstrakte Umsetzung des Ray Marchings <sup>11</sup> . . . . .	20
6.2. Eine abstrakte Umsetzung des Sphere Tracings <sup>12</sup> . . . . .	23
6.3. Algorithmus zur Berechnung von Schatten. . . . .	30
6.4. Algorithmus zur Berechnung von weichen Schatten. . . . .	31
7.1. Umsetzung des Sphere Tracings in GLSL. . . . .	34
7.2. Umsetzung von Operationen für implizite Oberflächen in GLSL. . . . .	35
7.3. Umsetzung von Primitiven in Form von impliziten Oberflächen in GLSL. . . . .	36
7.4. Umsetzung des Phong-Beleuchtungsmodelles in GLSL. . . . .	36
7.5. Berechnung der Normalen einer impliziten Oberfläche in GLSL. . . . .	37
7.6. Distanzfunktion $f$ in GLSL. . . . .	37
7.7. Funktion zur Berechnung von weichen Schatten in GLSL. . . . .	38

# 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

- \* 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)

\* 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 / procdeure
  - \* 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 Univeristy of applied sciences (CF)
- The lines shall be shortened so that they do not exceed 80 caracters 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 (TODO, added)

#### \* Global illumination models

- \* Rendering equation, 5.3 (CF)
  - \* Add the correct reference, either to Kajiya or another source (TODO, added)
  - \* 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) (TODO, added)
- \* 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) (TODO, added)
- \* The reference when referring Whitted's publication is missing (CF)
  - \* Add reference (SO) (TODO, added)
- \* Is the material sufficient? (SO)
  - \* Yes, the material is sufficient, but an illustration would probably be good (CF) (TODO, added)

#### \* 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`  
 (TODO, added)

- \* Equation (5.11) (CF)
  - \* Give an example by means of a sphere (CF)  
 (TODO, added)
- \* Functions for implicit surfaces (5.4.1) (CF)
  - \* Use an explanation of the signs as well when explaining the results (CF)  
 (TODO, added)
- \* 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)  
 (TODO)

#### Typography ~~~~~

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

#### Further steps/proceedings (CF, S0) -----

- \* 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? (S0)
- \* 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 (S0)
- \* Discuss the current state of the work (CF, S0)
- \* Define further steps/proceeding (CF, S0)
- \* 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)  
(TODO, added)
  - \* Normalize vectors to same length, use polarcoordinates eventually (CF)  
(TODO, added)
- \* Illustration Ray Tracing
  - \* Use a better resolution (CF)  
(TODO, added)
  - \* Make the fonts a bit bigger (CF)  
(TODO, added)
  - \* Explain the image (CF)  
(TODO, added)
  - \* Normalize vectors to same length, use polarcoordinates eventually (CF)  
(TODO, added)
- \* 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)
  - (TODO)
- \* Introduction
  - \* May resp. will most likely need to be rewritten at end of the project work (CF)
  - (TODO, added)
- \* Project schedule
  - \* Adapt to current situation (CF)
  - \* Prototype is missing (CF)
  - (TODO, added)
- \* 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)
  - (TODO, added)

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)  
(TODO, added)
- \* 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 (S0)

Meeting minutes 2015-11-15  
=====

Presentation and discussion of the current state of the work (S0)  
-----

- \* 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. (S0)
- \* 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. (S0)

Further steps/proceedings (CF, S0)  
-----

- \* 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 (S0)
- \* Discuss the current state of the work (CF, S0)
- \* Define further steps/proceeding (CF, S0)

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 proved 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

---