UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Raytracing von Distanzfeldern

Masterarbeit

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

vorgelegt von
Bastian Krayer

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergrafik)
Zweitgutachter:	Anna Katharina Hebborn, M.Sc. (Institut für Computervisualistik, AG Computergrafik)

Koblenz, im April 2016

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

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

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Aufgabenstellung für die Masterarbeit
Bastian Krayer
(Matr.-Nr. 210 100 751)

Thema: **Raytracing von Distanzfeldern**

Eine der grundlegenden Techniken der Computergrafik ist das Raytracing. Generell werden hierbei Daten als Polygone dargestellt. Eine andere Variante ist ein Distanzfeld, bei dem für jeden Punkt die Distanz zur nächstgelegenen Oberfläche bereitgestellt wird. Dies erlaubt eine Vielzahl von Effekten, die für Polygonmodelle schwer zu realisieren sind, wie etwa die Schnittmenge mehrere Modelle. Weiterhin können auch sehr komplexe Objekte, wie Fraktale oder implizite Oberflächen, ohne gesonderte Vorverarbeitung dargestellt werden.

Ziel dieser Arbeit ist es, die verschiedenen Verfahren zum Rendern und Erzeugen von Distanzfeldern zu analysieren und mindestens ein geeignetes zu implementieren. Dabei sollen sowohl Vor- und Nachteile als auch die Möglichkeiten und Grenzen aufgezeigt werden. Ein besonderes Augenmerk soll dabei auf implizite Oberflächen gelegt werden.

Schwerpunkte dieser Arbeit sind:

1. Einarbeitung
2. Recherche und Analyse bestehender Verfahren
3. Auswahl geeigneter Verfahren
4. Prototypische Implementierung
5. Vergleich und Bewertung der Verfahren
6. Dokumentation der Ergebnisse

Koblenz, den 30.10. 2015

- Bastian Krayer - - Prof. Dr. Stefan Müller -

Zusammenfassung

Eine der grundlegenden Entscheidungen bei der Entwicklung eines Systems ist die Darstellung der Daten. Üblicherweise werden in der Computergrafik Objekte durch Dreiecke dargestellt. Allerdings existieren viele weitere Varianten, welche andere Stärken und Schwächen besitzen. In dieser Arbeit soll die Repräsentation von Objekten durch Distanzfelder untersucht werden. Distanzfelder sind Funktionen, welche für jeden Raumpunkt die Distanz zum nächsten Oberflächenpunkt angeben. Aus dieser einfachen Beschreibung lassen sich viele interessante Eigenschaften ableiten, welche zur Darstellung einer Vielzahl von Formen, Operationen und Effekten genutzt werden können. Es wird ein Überblick über die Hintergründe und Methoden des Distanzfeld-Renderings gegeben. Weiterhin werden verschiedene neue oder erweiterte Ansätze vorgestellt, etwa zur Darstellung impliziter Oberflächen, approximativer indirekter Beleuchtung oder einer GPU Implementation.

Abstract

One of the fundamental decisions during the development of any system is the representation of data. In computer graphics, objects are usually represented as sets of triangles. There are however many different variants with their own strengths and weaknesses. This thesis will explore distancefields as a representation for objects. Distancefields are functions, which map every point in space to the distance to the closest surface point. While this description is very simple, a number of interesting properties can be derived, allowing for a multitude of shapes, operations and effects. An overview of the necessary background and methods is given. Furthermore, some extended or new approaches are presented, such as displaying implicit surfaces, approximating indirect illumination or implementing a GPU tracer.

Inhaltsverzeichnis

1 Einleitung **3**

2 Grundlagen **3**
2.1 Notation 3
2.2 Raytracing 4

3 Distanzfelder **5**
3.1 Was sind Distanzfelder? 6
3.2 Operationen 10
3.2.1 Operationen auf Distanzfeldern 10
3.2.2 Operationen auf dem Definitionsbereich 11
3.3 Erzeugung von Distanzfeldern 17
3.3.1 Einfache Körper und Objekte 17
3.3.2 Punkte und Punktmengen 23
3.3.3 Eindimensionale Funktionen 23
3.3.4 Quadratische Bézier Kurven 27
3.3.5 Implizite Oberflächen 28
3.3.6 Fraktale 34
3.3.7 Diskrete Felder 38
3.4 Bounding Volumes 41

4 Texturierung **42**

5 Tracing Verfahren **43**
5.1 Überblick 44
5.2 Sphere Tracing 44
5.3 Erweiterung 46

6 Besondere Effekte **47**
6.1 Weiche Schatten 48
6.2 Ambient Occlusion 50
6.3 Indirekte Beleuchtung 52

7 Implementation **55**
7.1 CPU Implementation 55
7.2 GPU Implementation 56
7.2.1 Andere Vorgehensweisen 56
7.2.2 Codeerzeugung 56

8 Ergebnisse **60**
8.1 Cornell Box mit verschiedenen Effekten 60
8.2 Voxeltracing 61
8.3 Verschiedene Szenen 62

8.4 Implizite Oberfläche 63
8.5 Diskussion und Ausblick 65

9 Zusammenfassung **67**

1 Einleitung

In der Computergrafik ist es üblich, Objekte als Menge von Dreiecken darzustellen. Diese einfache Repräsentation erlaubt eine einheitliche Beschreibung und effiziente Verarbeitung durch Grafikkarten. Zur genauen Darstellung von abgerundeten Oberflächen sind allerdings große Mengen an Dreiecken nötig. Obwohl Grafikkarten optimiert sind, ist es schwierig, für einen gegebenen Punkt Informationen über die Umgebung zu erhalten. Auch ist es nur mit komplexen Algorithmen möglich, mehrere Objekte zu kombinieren, beispielsweise mittels Vereinigung oder Schnittmenge. Solche Operationen erlauben allerdings eine einfache Art der Modellierung. Eine weitere Möglichkeit der Objektrepräsentation ist ein Distanzfeld. Dies ist eine Funktion, welche für jeden Punkt im Raum angibt, wie weit entfernt dieser von der nächsten Oberfläche ist. Während solch eine Funktion auf den ersten Blick sehr unscheinbar wirkt, lassen sich erstaunlich viele Dinge damit erreichen. So reichen die Effekte von der Modellierung, wo Schnittmenge oder Vereinigung einer einzigen Rechenoperation entsprechen, über Text Darstellung [1] bis hin zu modernen Techniken wie Ambient Occlusion oder weichen Schatten in Echtzeit [2]. Diese Arbeit soll einen Überblick über die Grundlagen von Distanzfeldern geben und verschiedene Techniken vorstellen, die mit anderen Darstellungsarten wesentlich aufwendiger wären. Es werden Beispiele und Ansätze gezeigt, welche Distanzfelder auch für Echtzeit-Rendering attraktiv machen. Da selten eine Methode der Datenrepräsentation für alle Anwendungsfälle optimal ist, wird an geeigneten Stellen auch auf Nachteile eingegangen.

2 Grundlagen

Dieser Abschnitt gibt einen kurzen Überblick über die verwendete Notation, sowie das grundsätzliche Prinzip eines Raytracing Verfahrens.

2.1 Notation

Vektoren werden generell klein und fett geschrieben. Punkte haben die gleiche Schreibweise, wobei nicht explizit zwischen Punkt und dem dazugehörigen Ortsvektor unterschieden wird. Einzelne Komponenten eines Vektors werden durch einen Index gekennzeichnet. So ist beispielsweise die zweite Komponente eines Vektors $\mathbf{x}$ geschrieben als $\mathbf{x}_2$. Da zwei- und dreidimensionale Vektoren im weiteren Verlauf von besonderer Bedeutung sind, wird auch auf die Indizierung der passenden Koordinatenachse zurückgegriffen. Die x-Koordinate eines Punktes $\mathbf{p}$ könnte etwa geschrieben werden als $\mathbf{p}_x$.
Ein hochgestellter Index wird zur Unterscheidung von Vektoren verwen-

det. Da keine Potenzen definiert sind, führt dies zu keiner Verwechslung. Für Skalare gilt die übliche Notation mit tiefgestelltem Index.
Oft kommt es vor, dass nur eine Teilmenge der Koordinaten eines Vektors als eigenständiger Vektor verwendet wird. So lässt sich ein zweidimensionaler Vektor aus einem dreidimensionalen bilden, indem die letzte Koordinate gestrichen wird. Dazu wird hier die $\boldsymbol{\pi}$ Funktion eingeführt.

$$\boldsymbol{\pi}_{(i_1,\ldots,i_k)} : \mathbb{R}^n \to \mathbb{R}^k \tag{1}$$

$$\boldsymbol{\pi}_{(i_1,\ldots,i_k)}(\mathbf{x}) = \begin{pmatrix} \mathbf{x}_{i_1} \\ \vdots \\ \mathbf{x}_{i_k} \end{pmatrix} \tag{2}$$

Das vorhergehende Beispiel ließe sich dann schreiben als $\boldsymbol{\pi}_{(1,2)}(\mathbf{x})$. Wichtig ist hier, dass die Reihenfolge der gewählten Indizes von Bedeutung ist, also generell etwa $\boldsymbol{\pi}_{(1,2)} \neq \boldsymbol{\pi}_{(2,1)}$ ist.
Falls es vom Kontext ersichtlich oder irrelevant ist, welche Dimensionalität die beteiligten Elemente haben, wird eine explizite Benennung der Indexbereiche ausgelassen. So ließe sich etwa ein Vektor definieren, dessen Komponenten gleich dem Betrag der entsprechenden Komponenten eines anderen Vektors sind, als

$$\mathbf{y}_i = |\mathbf{x}_i| \tag{3}$$

Je nach Kontext wäre $i = 1, \ldots, 3$ oder $i = 1, \ldots, n$. Die explizite Schreibweise ist:

$$\mathbf{y}_i = |\mathbf{x}_i| \qquad\qquad i = 1, \ldots, n \tag{4}$$

2.2 Raytracing

Raytracing ist ein grundlegendes Verfahren der Computergrafik. Es basiert auf der geometrischen Optik. Licht verbreitet sich über Strahlen, welche mit dem Computer nachverfolgt werden. Ein einfacher Raytracer besteht aus einer Kamera, einer Menge von Objekten und einer Menge von Punktlichtquellen. Beginnend von der Kamera werden einzelne Strahlen für jeden Pixel ausgesendet. Für einen Strahl wird der Schnittpunkt mit dem nächstgelegenen Objekt der gegebenen Szene berechnet. Von diesem Schnittpunkt aus wird das ankommende Licht aller gegebenen Punktlichtquellen gesammelt. Dazu wird mit jeweils einem weiteren Strahl getestet, ob die Lichtquellen sichtbar sind. Die sichtbaren Lichter werden dann mit einem gegebenen Beleuchtungsmodell verrechnet. Diese Idee kann auf viele Weisen erweitert werden. Statt Punktlichtquellen können beliebige Objekte oder Teile von Objekten als Lichtemitter genutzt werden. Weiterhin kann indirekte Beleuchtung, von nicht leuchtenden Oberflächen reflektiertes Licht, zugelassen werden. Diese Erweiterung erfordert dann, dass nicht

nur ein Strahl von einem Punkt ausgesendet wird, sondern beliebig viele, sodass die ganze obere Hemisphäre (oder auch die untere bei Refraktion) abgedeckt ist. Eine Formulierung dieses Vorgangs ist durch die Rendering Equation [3] gegeben:

$$I(\mathbf{x}, \mathbf{x}') = g(\mathbf{x}, \mathbf{x}')(\epsilon(\mathbf{x}, \mathbf{x}') + \int_S \rho(\mathbf{x}, \mathbf{x}', \mathbf{x}'') I(\mathbf{x}', \mathbf{x}'') d\mathbf{x}'') \tag{5}$$

$I(\mathbf{x}, \mathbf{x}')$ gibt die Intensität des Lichtes an, die von einem Punkt $\mathbf{x}'$ zu einem Punkt $\mathbf{x}$ transportiert wird. g ist ein geometrischer Term, der Sichtbarkeit oder Abstand enthält. $\rho(\mathbf{x}, \mathbf{x}', \mathbf{x}'')$ gibt die indirekte Beleuchtung an, die $\mathbf{x}$ über $\mathbf{x}'$ von $\mathbf{x}''$ erhält. S entspricht der Menge aller Oberflächenelemente. Die gesamte bei $\mathbf{x}$ eintreffende Intensität ergibt sich durch Integration über S von $I(\mathbf{x}, \mathbf{x}')$. Eine weitere Formulierung nutzt Richtungen statt Oberflächenelementen und geht zurück auf [4]. Eine Integralversion lautet [5]:

$$L(\mathbf{x}, \omega_o) = L(\mathbf{x}, \omega_o) + \int_\Omega L(\mathbf{x}', \omega_i)\, \mathrm{f}_r(\mathbf{x}, \omega_i, \omega_o) |\omega_i \cdot \mathbf{n}| d\omega_i \tag{6}$$

ω ist dabei eine Richtung, die Indizes i und o bedeuten "eingehend" und "ausgehend". f_r ist die bidirektionale Reflektanzverteilung und $\mathbf{n}$ die Normale am Punkt $\mathbf{x}$. Auch hier wird wieder über die Kugel oder Halbkugel integriert. Dieses Integral wird numerisch etwa durch Abtasten der Hemisphäre und Aussenden von Strahlen in diese Richtungen gelöst. Die Rekursion im Inneren von Gleichung 6 sorgt für die Aufnahme von indirektem Licht. Bei einer Berechnung wird dann die maximale Indirektion angegeben. Weitere Verfahren, die sogenannten Pathtracer, basieren auf Monte-Carlo Methoden der Stochastik. Hier wird nicht für jeden Punkt eine große Menge von Strahlen ausgesandt, sondern stets nur ein zufälliger. Dieser Vorgang wird viele Male wiederholt und die Ergebnisse gemittelt. Für einen umfassenderen Einstieg, siehe etwa [5]. Der in Abschnitt 5 vorgestellte Algorithmus ist eine spezielle Version einer Schnittpunktsuche. Dementsprechend lassen sich Distanzfelder auch für beliebige Ray- und Pathtracer nutzen. Dementsprechend können alle Ergebnisse zur Modellierung der Beleuchtung ohne Änderung auch für Distanzfelder genutzt werden. Da genauere Lösungen der gerade knapp vorgestellten Modelle sehr aufwendig sind, waren sie für lange Zeit nur für vorberechnete Bilder nutzbar. Mit der Entwicklung moderner GPUs ist es jedoch auch immer mehr möglich, die Berechnungen in Echtzeit durchzuführen, siehe beispielsweise [6]. In Abschnitt 7.2 und den Tests in Abschnitt 8 wird auch darauf eingegangen, wie Distanzfelder mithilfe der GPU dargestellt werden können.

3 Distanzfelder

Im Folgenden wird ein Überblick über Distanzfelder gegeben. Dabei wird darauf eingegangen, was ein Distanzfeld ist. Es wird besprochen, wie Di-

stanzfelder für verschiedene Effekte kombiniert werden können. Außerdem werden Formeln und Erklärungen zum Berechnen von Distanzfeldern für einige Beispiele gezeigt.

3.1 Was sind Distanzfelder?

Eine Möglichkeit, Objekte zu repräsentieren, ist ein Distanzfeld. Dieses Feld ist definiert für Punkte in Raum oder Ebene und gibt die Entfernung dieser Punkte zum gegebenen Objekt an. Im Folgenden wird im Allgemeinen vom $\mathbb{R}^n$ ausgegangen, jedoch für viele Erklärungen speziell der $\mathbb{R}^2$ oder $\mathbb{R}^3$ gewählt, etwa für besondere Transformationen oder Objekte. Ein Objekt O ist eine Teilmenge des Raums.

$$O \subset \mathbb{R}^n \tag{7}$$

Allgemein lässt sich mit einem geeigneten Abstandsmaß $\|.\|$ ein Distanzfeld d_O zum Objekt O definieren als

$$\mathrm{d}_O : \mathbb{R}^n \to \mathbb{R} \tag{8}$$

$$\mathrm{d}_O(\mathbf{x}) = \min_{\mathbf{p} \in O} \|\mathbf{x} - \mathbf{p}\| \tag{9}$$

Es sind allerdings auch weitere Charakterisierungen möglich. d_O ordnet also jedem Raumpunkt den kleinsten Abstand zu allen Punkten im gegebenen Objekt zu. Aus dieser Definition ist sofort ersichtlich, dass jeder Raumpunkt, welcher dem Objekt angehört, den Wert 0 von d zugewiesen bekommt, während alle anderen Werte größer als 0 sein müssen. Das Objekt entspricht damit der Nullstellenmenge seines Distanzfeldes, beziehungsweise der Niveaumenge von d zum Wert 0.

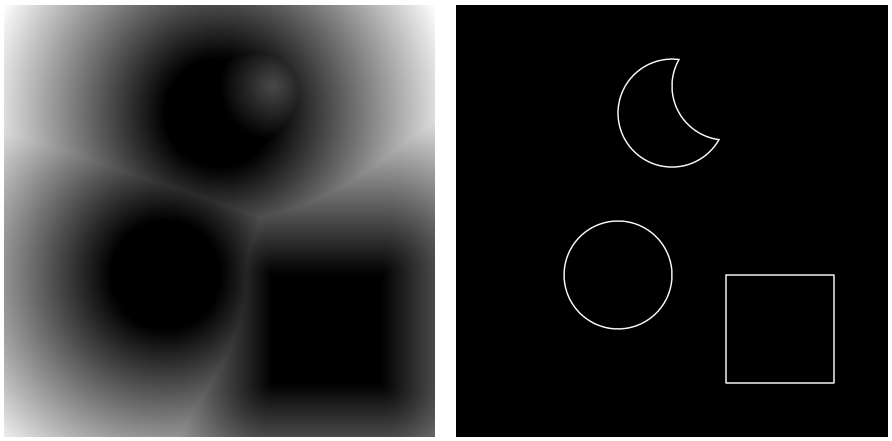
$$\begin{aligned} O &= \{\mathbf{x} \in \mathbb{R}^n | \, \mathrm{d}_O(\mathbf{x}) = 0\} \\ &= \mathrm{d}_O^{-1}(0) \end{aligned} \tag{10}$$

Durch diese nicht-explizite Beschreibung sind Distanzfunktionen eine Klasse von impliziten Funktionen, auf welche in 3.3.5 näher eingegangen wird. Diese haben im Allgemeinen nicht die Einschränkung, eine exakte Distanz anzugeben. Die grundsätzliche Vorgehensweise zum Raytracing eines Distanzfeldes besteht mit dieser Grundlage daraus, entlang eines Strahls die erste Nullstelle von d_o zu bestimmen. Dadurch, dass für jeden Punkt bekannt ist, wie nah der nächste mögliche Objektpunkt ist, kann der Strahl wesentlich schneller abgesucht werden. Hierauf wird genauer in 5 eingegangen. Im weiteren Verlauf wird auf den Index der Distanzfunktion verzichtet, solange nicht explizit zwischen verschiedenen Objekten unterschieden wird.
Aus Definition 8 geht hervor, dass die Art der darstellbaren Objekte sehr beliebig ist, solange ein Abstand berechnet werden kann. Dies erlaubt eine

Vielzahl von Operationen und Objekten, die mit traditionelleren Verfahren weitaus komplexer oder kaum möglich wären. Genaueres dazu folgt in späteren Abschnitten.
Bei der Modellierung werden Körper meist durch ihre Oberflächen definiert. Dies erlaubt etwa die Definition einer Normalen für Punkte auf dem Körper, um die Beleuchtung zu berechnen. Dreidimensionale Körper können dabei eine Ausdehnung im Raum haben. Dementsprechend teilen sie den Raum auf in Äußeres und Inneres. Das bisherige Distanzfeld ignoriert diese Unterscheidung, kann aber leicht erweitert werden. Ein *signiertes Distanzfeld* s ist negativ im inneren und positiv im äußeren Bereich (je nach Literatur kann dies auch andersherum definiert sein). Der Betrag entspricht dabei dem unsignierten Distanzfeld.

$$\mathrm{s}(\mathbf{x}) = \begin{cases} -\mathrm{d}(\mathbf{x}) & \text{, falls } \mathbf{x} \in O^\circ \\ \mathrm{d}(\mathbf{x}) & \text{sonst} \end{cases} \tag{11}$$

Hierbei bezeichnet $^\circ$ das innere einer Menge. Der Gradient der signier-



(a) Beispielhaftes 2D Distanzfeld von drei Objekten

(b) Eingefärbte Version des Distanzfeldes. Die Färbung erfolgt durch einen einfachen Distanztest

Abbildung 1: Beispiel eines Distanzfeldes in 2D. Alle Pixel mit einer gewissen absoluten Distanz wurden im zweiten Bild eingefärbt

ten Distanzfunktion liefert am Rand des Objekts die normalisierte Normale. Für die später angesprochenen nicht exakten Distanzfunktionen ist die Normale nicht unbedingt normalisiert. Anschaulich kann dies dadurch verdeutlicht werden, dass der Gradient in die Richtung der größten Änderung zeigt. Da die Umgebung auf der Oberfläche um einen Punkt den gleichen Wert wie der Punkt hat, nämlich 0, ist entlang der Oberfläche keine Änderung. Außerdem muss sie nach außen zeigen, da die Werte außerhalb des Objekts positiv und innerhalb negativ sind. Genauer kann dies gezeigt

werden, indem eine beliebige Kurve $\boldsymbol{\gamma}$ auf der Oberfläche um einen Punkt betrachtet wird, siehe etwa [7, Kapitel 14.2]. Da die Kurve auf der Oberfläche liegt, gilt $\mathrm{s}(\boldsymbol{\gamma}(t)) = 0$. Differentiation beider Seiten ergibt

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{s}(\boldsymbol{\gamma}(t))) &= \frac{\mathrm{d}}{\mathrm{d}t}0 \\ \nabla(\mathrm{s}(\boldsymbol{\gamma}(t))) \cdot \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\gamma}(t) &= 0 \end{aligned} \tag{12}$$

Da die Ableitung einer Kurve der Tangente entspricht, also auf der Oberfläche am Punkt liegt, und das obige Skalarprodukt 0 ergibt, steht der Gradient der Distanzfunktion senkrecht auf der Tangente. Da dies für jede beliebige Kurve gilt, steht der Gradient also allgemein senkrecht auf der Oberfläche.
Falls die Oberfläche glatt genug ist, kann davon ausgegangen werden, dass die Normale auch in einem kleinen Bereich noch ungefähr gleich ist. Dies folgt direkt aus den Ergebnissen in [8]. Dadurch kann gewährleistet werden, dass die Beleuchtung auch dann noch korrekt funktioniert, falls etwa durch Genauigkeitsprobleme der Oberflächenpunkt nur ungefähr bestimmt wird.
Im Allgemeinen sind exakte Distanzfunktionen zu bevorzugen, da diese eine schnellere Konvergenz beim Sphere Tracing Verfahren erlauben, was in 5 genauer erläutert wird. In manchen Fällen ist es nicht oder nur schwer möglich, eine exakte Distanz zu errechnen, beispielsweise bei den allgemeinen impliziten Oberflächen in Abschnitt 3.3.5. In solchen Fällen reicht ein approximatives Distanzfeld aus, welches ungefähr einem exakten entsprechen sollte. Je nach Unsicherheit in der Approximation können die Parameter des Verfahrens angepasst werden. Eine wichtige Klasse von Distanzfeldern sind jene, die stets kleiner oder gleich der exakten sind. Nach Hart in [9] werden diese Funktionen "signed distance bound", also signierte Distanzschranken genannt.

$$|\,\mathrm{f}(\mathbf{x})| \leq |\,\mathrm{s}(\mathbf{x})| \tag{13}$$

Falls also eine solche Funktion f gefunden werden kann, ist dies bereits ausreichend zur korrekten Darstellung. Hierunter fallen etwa die von Hart genauer beschriebenen Lipschitz stetigen Funktionen. Das sind solche Funktionen, deren Änderungsrate beschränkt ist.

$$|\,\mathrm{f}(\mathbf{x}) - \mathrm{f}(\mathbf{y})| \leq L\,\|\mathbf{x} - \mathbf{y}\| \tag{14}$$

$L \geq 0$ wird Lipschitz Konstante genannt [10, Kapitel 19.2]. Sie gibt an, wie stark der Funktionswert sich maximal ändern kann, abhängig vom Abstand der Funktionswerte. Hart hat gezeigt, dass durch diese Konstante automatisch eine Distanzschranke definiert werden kann. Liege der Punkt

$\mathbf{y}$ auf der Oberfläche, so gilt $\mathrm{s}(\mathbf{y}) = 0$. Eine gegebene Funktion f sei Lipschitz mit Konstante L und erfülle ebenfalls $\mathrm{f}(\mathbf{y}) = 0$. Dann gilt

$$\begin{aligned} |\,\mathrm{f}(\mathbf{x})| &= |\,\mathrm{f}(\mathbf{x}) - \mathrm{f}(\mathbf{y})| \\ &\leq L\,\|\mathbf{x} - \mathbf{y}\| \\ &= L|\,\mathrm{s}(\mathbf{x})| \end{aligned} \tag{15}$$

Daraus erfolgt durch Division durch L

$$\frac{1}{L}|\,\mathrm{f}(\mathbf{x})| \leq |\,\mathrm{s}(\mathbf{x})| \tag{16}$$

Dieses Vorgehen erlaubt auch die Transformation von Objekten im Definitionsbereich. Durch den Schrankensatz für vektorwertige Funktionen kann eine geeignete Lipschitz Konstante gefunden werden als [11, Kapitel 21.5]

$$L = \max_{\mathbf{x}\in\mathbb{R}^n} \left\| \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\,\mathbf{T}(\mathbf{x}) \right\| \tag{17}$$

$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}$ bezeichnet hier die Jacobi Matrix. Für die Wahl einer Norm für Matrizen gibt es mehrere Möglichkeiten, wobei die gewählte mit der Vektornorm verträglich sein sollte. Verträglichkeit bedeutet, dass $\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\,\|\mathbf{x}\|$ gilt. Zwei dieser Normen sind die Spektralnorm und die Frobeniusnorm. Die Spektralnorm $\|.\|_2$ ist von der euklidischen Vektornorm induziert und erlaubt die kleinste Abschätzung des obigen Produktes. Sie ist definiert als Wurzel des Spektralradius' der Matrix, also dem betragsmäßig größtem Eigenwert von $\mathbf{A}^H\mathbf{A}$, wobei $\mathbf{A}^H$ die adjungierte Matrix von $\mathbf{A}$ ist. Für eine symmetrische Matrix vereinfacht sich die Spektralnorm zum Betrag des betragsmäßig größten Eigenwertes. Die Frobeniusnorm $\|.\|_F$ ist nicht induziert, aber ähnelt vom Aufbau der euklidischen Vektornorm. Sie ist definiert als Wurzel aller quadrierten Elemente und ist dementsprechend leichter zu errechnen als die Spektralnorm. Weiteres zu den Eigenschaften findet sich unter anderem in [10, Kapitel 11.3].
Die Bedingung an den Bereich, über den maximiert wird, lässt sich auf verschiedene Weisen abschwächen. Falls der Bereich der Szene bekannt oder manuell festgelegt wird, muss nicht global optimiert werden, sondern es reicht, die Optimierung auf den darzustellenden Bereich zu beschränken. Dies hat den weiteren Vorteil, dass auch Funktionen betrachtet werden können, welche global unbeschränkt sind. Problematisch ist allerdings für beide Varianten die praktische Berechnung, da die Schranken eventuell sehr groß werden können. Nach der Ungleichung 16 wird durch diese Schranke geteilt. Da nur mit begrenzter Genauigkeit gerechnet wird, kann dies dafür sorgen, dass viel zu große Werte unter eine Schranke fallen, wo sie als 0 angesehen werden, was zu vorzeitigen Schnittpunkten führen würde. Außerdem sorgen kleine Schranken für sehr kleine Schrittweiten

des Verfahrens, wie in Abschnitt 5 genauer beschrieben wird. Aus diesem Grund könnte es sinnvoll sein, eine hierarchische Struktur zu wählen, bei der verschiedene Konstanten für größer werdende Bereiche berechnet werden.

3.2 Operationen

Ein großer Vorteil der Darstellung von Objekten als Distanzfeld ist die Möglichkeit, gegebene Distanzfelder zu transformieren oder miteinander zu kombinieren. Dies reicht von einfachen Operationen wie Drehung oder Skalierung bis zu komplexen Effekten wie Schnittmengen verschiedener Objekte oder Deformationen. Dabei ist keine zusätzliche Vorverarbeitung von Nöten, es müssen nur die Werte oder Argumente von gegebenen Distanzfunktionen angepasst werden. Im Folgenden werden zwei Arten von Operationen betrachtet. Zum einen solche, die direkt auf den Distanzfeldern arbeiten, zum anderen diejenigen, welche auf den Definitionsbereich wirken.

3.2.1 Operationen auf Distanzfeldern

Ein Einsatzgebiet der Distanzfelder ist CSG (Constructive Solid Geometry). Hierbei werden komplexe Objekte durch geschicktes Kombinieren einfacherer erzeugt. Ein Beispiel ist eine Halbkugel, welche als Schnittmenge einer Kugel und eines Quaders entsteht. In [12] sind die dafür relevanten Operationen bewiesen, welche im Folgenden erläutert werden. Analog zu Mengenverknüpfungen lassen sich Vereinigung, Schnittmenge und Komplement definieren. Es seien zwei Objekte A und B mit zugehörigen Distanzfunktionen s_A und s_B gegeben. Bei der Vereinigung von zwei Objekten liegt jeder Punkt im neuen Objekt, der im ersten oder im zweiten liegt. Außerhalb beider Objekte entspricht die Distanz dem nächsten, also dem kleineren der beiden Distanzwerte. Befindet sich ein Punkt in einem, aber nicht im anderen Objekt, so muss offensichtlich der negative Wert genutzt werden. Falls der Punkt in beiden Objekten liegt, so muss der größte absolute Abstand ausgewählt werden, was dem kleineren der beiden negativen Werte entspricht. Zusammengefasst gilt also

$$\mathrm{s}_{A\cup B}(\mathbf{x}) = \min(\mathrm{s}_A(\mathbf{x}), \mathrm{s}_B(\mathbf{x})) \tag{18}$$

Eine analoge Argumentation lässt sich für die Schnittmenge der Objekte machen und führt darauf, dass stets der maximale Wert ausgewählt werden muss.

$$\mathrm{s}_{A\cap B}(\mathbf{x}) = \max(\mathrm{s}_A(\mathbf{x}), \mathrm{s}_B(\mathbf{x})) \tag{19}$$

Bei den durch die Distanzfunktionen definierten Objekten ist der Rand bestimmt durch die Gleichung $\mathrm{s}(\mathbf{x}) = 0$, was zum Objekt zugehörig gesehen

wird. Als Komplement des Objekts A sei im Folgenden das Komplement des Inneren A° definiert, sodass der Rand auch zum komplementären Objekt gehört, geschrieben als A^C. Dies erlaubt eine sehr einfache Definition, die daraus besteht, Inneres und Äußeres zu vertauschen.

$$\begin{aligned} \mathrm{s}_{A^C}(\mathbf{x}) &= \mathrm{s}_{\mathbb{R}^n \setminus A^\circ}(\mathbf{x}) \\ &= -\mathrm{s}_A(\mathbf{x}) \end{aligned} \tag{20}$$

Daraus folgt auch direkt eine Formel für die Subtraktion eines Objektes von einem anderen. Dies kann ausgedrückt werden als Schnittmenge des ersten Objektes und dem Komplement des zweiten.

$$\begin{aligned} \mathrm{s}_{A \setminus B}(\mathbf{x}) &= \mathrm{s}_{A \cap B^C}(\mathbf{x}) \\ &= \max(\mathrm{s}_A(\mathbf{x}), -\mathrm{s}_B(\mathbf{x})) \end{aligned} \tag{21}$$

Eine interessante Operation ist eine Variante der Vereinigung. Unter anderem ist es bei vielen natürlichen Objekten nicht wünschenswert, dass harte Kanten zu sehen sind. Es entspricht eher der Erwartung, dass Formen weich ineinander übergehen. Mit Distanzfeldern kann dies auf verschiedene Weisen gelöst werden. Generell besteht das Vorgehen daraus, die Minimumsoperation, durch eine differenzierbare Approximation zu ersetzen. In [12] ist etwa zu finden:

$$\mathrm{smin}_p(\mathbf{x}) = (\mathrm{s}_A(\mathbf{x})^{-p}, \mathrm{s}_B(\mathbf{x})^{-p})^{-\frac{1}{p}} \tag{22}$$

Dabei ist p eine positive reelle Zahl. Weitere Möglichkeiten und die Bezeichnung smin finden sich beispielsweise in [13], wovon eine im Folgenden gezeigt wird.

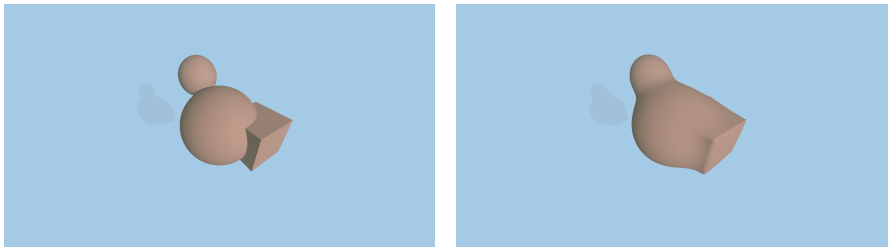
$$\mathrm{smin}_k(\mathbf{x}) = -\frac{\log(\exp(-k\,\mathrm{s}_A(\mathbf{x})) + \exp(-k\,\mathrm{s}_B(\mathbf{x}))}{k} \tag{23}$$

smin steht dabei für "Smooth minimum". In Abbildung 2 wird der Unterschied zwischen Vereinigung und weicher Vereinigung (Blending) gezeigt.

3.2.2 Operationen auf dem Definitionsbereich

Während die vorhergehenden Operationen direkt auf den Werten der Distanzen operieren, ist es auch möglich auf dem Wertebereich zu arbeiten. Sei eine Transformation $\mathbf{T} : \mathbb{R}^n \to \mathbb{R}^n$ gegeben. Der Wunsch besteht darin, das Objekt auf eine bestimmte Art zu verformen. Dafür sollen alle Punkte des Objekts transformiert werden.

$$\mathbf{T}(\mathbf{x}) = \mathbf{y} \tag{24}$$



(a) Vereinigung von zwei Objekten **(b)** Blending von zwei Objekten

Abbildung 2: Unterschied zwischen Vereinigung und Blending

Weiterhin sei $\mathbf{T}$ invertierbar, es ist also möglich, $\mathbf{x}$ wiederherzustellen durch

$$\mathbf{x} = \mathbf{T}^{-1}(\mathbf{y}) \tag{25}$$

Einsetzen in eine vorgegebene Distanzfunktion ergibt

$$\begin{aligned} s_{\mathbf{T}}(\mathbf{x}) &= s(\mathbf{T}^{-1}(\mathbf{x})) \\ &= (s \circ \mathbf{T}^{-1})(\mathbf{x}) \end{aligned} \tag{26}$$

Wie auch in [9] angemerkt, ist die Komposition zweier Lipschitz Abbildungen mit Konstanten L_1 und L_2 wieder Lipschitz, mit $L_1 L_2$. Unter der Annahme, dass die Distanzfunktion bereits die Konstante in ihrer Definition enthält, fehlt also nur noch diejenige von $\mathbf{T}^{-1}$. Eine Möglichkeit ist bereits in 3.1 beschrieben worden als Maximum der Norm der Ableitung der Transformation. Hierbei stehen beispielsweise die Frobenius- oder Spektralnorm zur Verfügung. Im Falle von linearen Abbildungen ist die Ableitung exakt die Matrix der Abbildung. Bei Hart werden zuerst die Isometrien betrachtet, also Rotationen, Translationen und Spiegelungen. Rotationen und Spiegelungen sind orthogonale Matrizen mit einer Determinante von 1 und -1. Sei die Dreh-, beziehungsweise Spiegelungsmatrix gegeben als $\mathbf{R}$, dann gilt nach Definition $\mathbf{R}^T\mathbf{R} = \mathbf{I}$. Die Wurzel des größten Eigenwertes von $\mathbf{R}^T\mathbf{R}$ entspricht der Spektralnorm. Da $\mathbf{R}^T\mathbf{R}$ allerdings gleich der Einheitsmatrix ist, deren einziger Eigenwert 1 ist, folgt, dass die Lipschitzkonstante ebenso 1 ist und demnach die Distanzfunktion nicht angepasst werden muss. Es folgt also

$$s_{\mathbf{T}}(\mathbf{x}) = s(\mathbf{R}^T\mathbf{x}) \tag{27}$$

Für Translationen entspricht die Ableitung direkt der Einheitsmatrix. Bei einer Verschiebung um $\mathbf{t}$ ist die Distanzfunktion gegeben als

$$s_{\mathbf{T}}(\mathbf{x}) = s(\mathbf{x} - \mathbf{t}) \tag{28}$$

Beide Operationen können auch zusammengefasst werden zu

$$s_{\mathbf{T}}(\mathbf{x}) = s(\mathbf{R}^T\mathbf{x} - \mathbf{t}) \tag{29}$$

Eine weitere wichtige Operation ist die Skalierung. Eine allgemeine Skalierung für die verschiedenen Achsen kann als eine Matrix $\mathbf{S}$ dargestellt werden, mit

$$\mathbf{S} = \operatorname{diag}(a_1, \ldots, a_n) \tag{30}$$

$$\mathbf{S}^{-1} = \operatorname{diag}(1/a_1, \ldots, 1/a_n) \tag{31}$$

Hier beschreibt $\operatorname{diag}(a_1, \ldots, a_n)$ eine Diagonalmatrix mit den in Klammern angegebenen Werten als Diagonaleinträgen. Da dies symmetrische Matrizen sind, entspricht die Spektralnorm dem betragsmäßig größten Eigenwert. Weiterhin stehen bei Diagonalmatrizen die Eigenwerte auf der Hauptdiagonalen, sind hier also gleich den Skalierungsparametern. Der größte Eigenwert der Umkehrabbildung ist damit offensichtlich derjenige mit dem kleinsten Skalierungsfaktor $a_{min} = \min_{i=1,\ldots,n} a_i$. Durch die Formel 16 ergibt sich damit die Distanzschranke für eine Skalierung $\mathbf{S}$ durch

$$\begin{aligned} \mathrm{s}_{\mathbf{T}}(\mathbf{x}) &= \frac{1}{\frac{1}{a_{min}}}\, \mathrm{s}(\mathbf{S}^{-1}\mathbf{x}) \\ &= a_{min}\, \mathrm{s}(\mathbf{S}^{-1}\mathbf{x}) \end{aligned} \tag{32}$$

Bei gleichförmiger Skalierung in allen Achsen um einen Faktor a folgt damit auch die bei Hart zu findende Formel, bei der $a_{min} = a$ ist:

$$\mathrm{s}_{\mathbf{T}}(\mathbf{x}) = a\, \mathrm{s}(\frac{1}{a}\mathbf{x}) \tag{33}$$

Eine weitere interessante Transformation ist ein "Twist". Hierbei wird ein Objekt abhängig von einer Achse lokal in den beiden weiteren Achsen gedreht. Für den Fall, dass die Drehachse die z-Achse ist, lässt sich dies mit folgender Formel berechnen [9].

$$\mathbf{Twist}(\mathbf{x}) = \begin{pmatrix} x\cos \mathrm{f}(z) - y \sin \mathrm{f}(z) \\ x \sin \mathrm{f}(z) + y \cos \mathrm{f}(z) \\ z \end{pmatrix} \tag{34}$$

Die beiden ersten Koordinaten entsprechen offensichtlich einer Rotation um den Winkel $\mathrm{f}(z)$, könnten also auch mit einer Rotationsmatrix geschrieben werden. Der Winkel $\mathrm{f}(z)$ kann dabei eine beliebige Funktion sein. Für T Drehungen pro z-Einheit entspricht dies $\mathrm{f}(z) = 2\pi T z$. Die inverse Transformation ergibt sich dadurch, die Rotation rückgängig zu machen, also um den entgegengesetzten Winkel zu drehen. Somit bleibt die Formel 34 erhalten, es wird lediglich f durch $-\mathrm{f}$ ersetzt. Zur besseren Lesbarkeit sei $\mathrm{g}(z) = -\mathrm{f}(z)$. Damit ergibt sich die Jacobi Matrix der inversen Twist-Transformation durch

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\, \mathbf{Twist}^{-1}(\mathbf{x}) = \begin{pmatrix} \cos \mathrm{g}(z) & -\sin \mathrm{g}(z) & -\mathrm{g}'(z)(x \sin \mathrm{g}(z) + y \cos \mathrm{g}(z)) \\ \sin \mathrm{g}(z) & \cos \mathrm{g}(z) & \mathrm{g}'(z)(x \sin \mathrm{g}(z) - y \cos \mathrm{g}(z)) \\ 0 & 0 & 1 \end{pmatrix} \tag{35}$$

Zur Abschätzung der Lipschitz Konstanten kann die Frobeniusnorm genutzt werden.

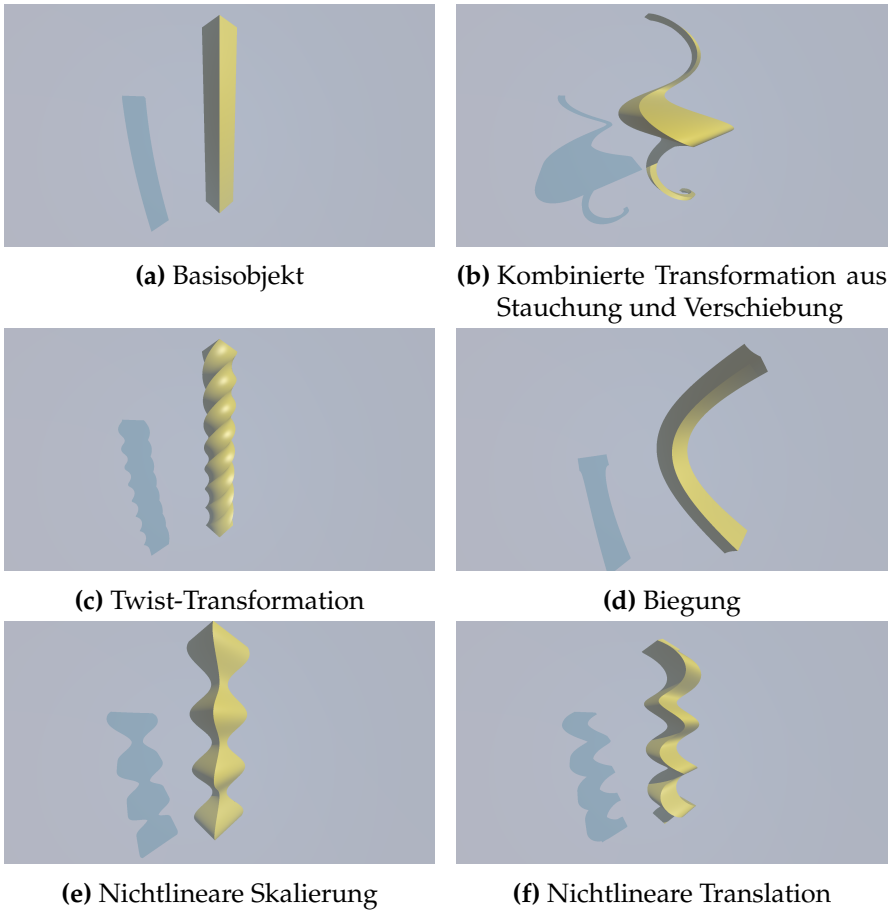
$$\begin{aligned}
\left\| \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}} \mathbf{Twist}^{-1}(\mathbf{x}) \right\|_F^2 &= \cos^2 \mathrm{g}(z) + \sin^2 \mathrm{g}(z) + \sin^2 \mathrm{g}(z) \\
&\quad + \cos^2 \mathrm{g}(z) + 1^2 + (\mathrm{g}'(z))^2 (x \sin \mathrm{g}(z) + y \cos \mathrm{g}(z))^2 \\
&\quad + (\mathrm{g}'(z))^2 (x \sin \mathrm{g}(z) - y \cos \mathrm{g}(z))^2 \\
&= 3 + (\mathrm{g}'(z))^2 \\
&\quad * ((x \sin \mathrm{g}(z) + y \cos \mathrm{g}(z))^2 + (x \sin \mathrm{g}(z) - y \cos \mathrm{g}(z))^2) \\
&= 3 + (\mathrm{g}'(z))^2 \\
&\quad * (x^2 (\sin^2 \mathrm{g}(z) + \cos^2 \mathrm{g}(z)) + y^2 (\cos^2 \mathrm{g}(z) + \sin^2 \mathrm{g}(z))) \\
&= 3 + (\mathrm{g}'(z))^2 (x^2 + y^2) \\
&= 3 + (\mathrm{g}'(z))^2 r^2
\end{aligned} \tag{36}$$

In der letzten Zeile wurde $x^2 + y^2$ durch den quadrierten Radius r^2 ersetzt. Weiterhin lässt sich g durch die ursprüngliche Definition ersetzen, wobei durch den Betrag das Minuszeichen wegfällt. Damit ergibt sich

$$\begin{aligned}
\left\| \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}} \mathbf{Twist}^{-1}(\mathbf{x}) \right\|_F^2 &= 3 + (r\, \mathrm{f}'(z))^2 \\
\left\| \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}} \mathbf{Twist}^{-1}(\mathbf{x}) \right\|_F &= \sqrt{3 + (r\, \mathrm{f}'(z))^2}
\end{aligned} \tag{37}$$

Diese Schranke entspricht nicht derjenigen in [9], wo auch fälschlicherweise die inverse Transformation durch $\frac{1}{\mathrm{f}(z)}$ angegeben ist. Offensichtlich ist die hier gefundene Schranke nicht global beschränkt. Für größer werdende Radien wächst die Funktion bei gleichbleibendem z immer weiter. Im linearen Fall $az + b$ ergibt sich als Ableitung a und somit eine direkte Formel $L = \sqrt{3 + (ra)^2}$. Für gegebene maximale und minimale Werte von r kann damit die Lipschitz Konstante bestimmt werden. Für beliebige Funktionen kann beispielsweise ein beliebiges Optimierungsverfahren genutzt werden.
Während das manuelle Bestimmen für bestimmte Transformationen sinnvoll sein kann, um eine geeignete Konstante zu bestimmen, ist es doch sehr mühsam, vor allem bei steigender Komplexität der Abbildungen. Eine allgemeinere Variante, welche auch zum Erstellen der Bilder in Abbildung 3 genutzt wurde, ist es, direkt für mehrere Bereiche passende Maxima der Norm der Jacobi-Matrix einer gegeben Transformation zu finden, mithilfe eines Optimierungsverfahrens. Durch automatische Differenzierung (wobei auch symbolische Varianten möglich sind) lassen sich exakte Jacobi Matrizen von beliebigen Funktionen erzeugen. Das Maximum kann

dann leicht über Gradientenabstieg oder Ähnliches gefunden werden. Dazu wird lediglich die negierte Norm verwendet, wodurch das Maximierungsproblem zu einem Minimierungsproblem umgewandelt wurde.
Eine letzte hier vorgestellte Transformation ist die Wiederholung. Distanz-



(a) Basisobjekt

(b) Kombinierte Transformation aus Stauchung und Verschiebung

(c) Twist-Transformation

(d) Biegung

(e) Nichtlineare Skalierung

(f) Nichtlineare Translation

Abbildung 3: Anwendung verschiedener Transformationen auf ein Basisobjekt, welches in 3a dargestellt wird

felder erlauben es, auf sehr einfache Weise ganze Objekte bis ins Unendliche fortgesetzt zu wiederholen. Das Problem sei im Folgenden im Eindimensionalen betrachtet, da alle Dimensionen getrennt verarbeitet werden. Es sei eine Schrittweite $c \in \mathbb{R}$ gegeben. Ein Punkt $p \in O$ soll nun im Abstand c wiederholt werden. Zur Vereinfachung sei angenommen, dass $0 \leq p \leq c$ ist. Er definiert damit eine Menge von Punkten mit $q_k = p + kc, k \in \mathbb{Z}$. Hieraus wird bereits ersichtlich, dass diese Transformation nicht wie vorhergehende funktioniert. Ein Punkt wird nicht auf einen anderen abgebildet, sondern auf eine ganze Menge, es liegt also keine Abbildung mehr vor. Trotzdem lässt sich eine Umkehrtransformation bestimmen, wenn die-

se auch nicht injektiv oder surjektiv ist außerhalb von $[0, c)$. Zum Auflösen wird die Modulo Operation $\mathrm{mod}(x, y) = x - y\lfloor\frac{x}{y}\rfloor$ verwendet, welche auch für nicht ganzzahlige c verwendet werden kann. Es gibt weitere Definitionen, allerdings ist die hier verwendete etwa in GLSL [14] implementiert. Andere Varianten benötigen eventuell leichte Anpassungen. Daraus und den Rechenregeln des Modulo folgt

$$\begin{aligned}\mathrm{mod}(q, c) &= \mathrm{mod}(p + kc, c)\\ &= \mathrm{mod}(\mathrm{mod}(p, c) + \mathrm{mod}(kc, c), c)\\ &= \mathrm{mod}(p + 0, c)\\ &= \mathrm{mod}(p, c)\\ &= p\end{aligned} \tag{38}$$

Problematisch an dieser Umstellung ist der Wertebereich von mod, welcher sowohl für positiven, als auch für negativen Input in $[0, c)$ abbildet. Außerdem ist die Funktion an den Rändern, wo der Funktionswert vom einen zum anderen Rand wechselt, unstetig, was Distanzfelder unbrauchbar machen kann. Besonders bei sehr asymmetrischen Körpern. Um das erste Problem zu beheben, bietet sich eine Verschiebung des ursprünglichen Körpers an, etwa so, dass der Ursprung auf der Hälfte des Intervalls liegt, wie es in [15] gemacht wurde. Diese Verschiebung geht natürlich davon aus, dass Objekte zentriert sind. Auch hier muss wieder die inverse Transformation angewendet werden. Daraus folgt die Umkehrfunktion

$$p = \mathrm{mod}(q, c) - \frac{c}{2} \tag{39}$$

Komponentensweise angewendet lassen sich so verschiedene Schrittweiten in den einzelnen Dimensionen festlegen und es ergibt sich die folgende Funktion zur Transformation der für die Distanzfunktionen gegebenen Punkte.

$$\mathbf{p}_i = \mathrm{mod}(\mathbf{q}_i, \mathbf{c}_i) - \frac{\mathbf{c}_i}{2} \tag{40}$$

$$\mathrm{s}_{\mathrm{twist}}(\mathbf{q}) = \mathrm{s}(\mathbf{p}) \tag{41}$$

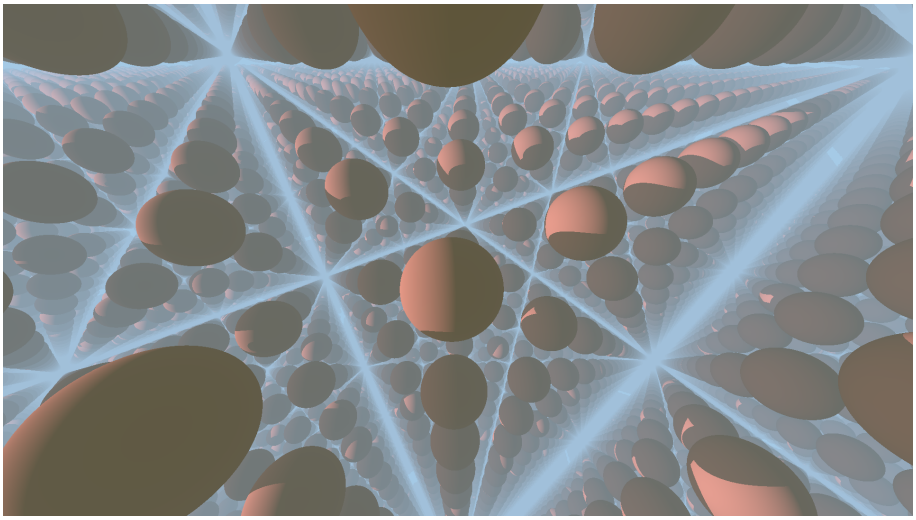


Abbildung 4: Wiederholung eines Objekts

3.3 Erzeugung von Distanzfeldern

Bisher wurde lediglich behandelt, was ein Distanzfeld ist und auf welche Weise sich bereits gegebene Distanzfelder transformieren, beziehungsweise kombinieren lassen. Im Folgenden werden die Distanzfelder, oder zumindest Approximationen davon, verschiedener Objekte hergeleitet. Diese reichen von einfachen geometrischen Körpern und Objekten wie Kugeln und Dreiecken bis hin zu hochkomplexen Objekten wie impliziten Oberflächen und Fraktalen.

3.3.1 Einfache Körper und Objekte

Von großer Bedeutung sind Körper wie Kugeln oder Würfel. Durch ihre sehr einfache Gestalt können sie gut veranschaulicht werden und erlauben intuitives Kombinieren, sodass sehr komplexe Objekte erzeugt werden können.

Kugel Der Abstand eines Punktes zu einer Kugel ist bekanntlich die Differenz aus Abstand zum Kugelmittelpunkt $\mathbf{m}$ und Radius r. Um ein signiertes Distanzfeld zu erhalten, muss lediglich der Betrag ausgelassen werden.

$$s_{\text{kugel}}(\mathbf{x}) = ||\mathbf{x} - \mathbf{m}|| - r \tag{42}$$

Diese Formel funktioniert für beliebige Dimensionen.

Ebene Die Distanz für einen Punkt zu einer im Raum verlaufenden Ebene entspricht der Länge des Vektors zwischen diesem Punkt und seiner Projektion auf die Ebene. Die wohl einfachste Variante ist die Darstellung der Ebene in Hessescher Normalform

$$\mathbf{n} \cdot \mathbf{p} - d = 0 \tag{43}$$

Hierbei ist der Normalenvektor $\mathbf{n}$ normalisiert und d der Abstand der Ebene zum Ursprung. Die signierte Distanz eines Punktes ist dann gegeben durch

$$\mathrm{s}(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} - d \tag{44}$$

Alternativ zu d kann auch ein Punkt $\mathbf{p}$ in der Ebene genutzt werden. Die Distanz ist dann der Anteil des Verbindungsvektors $\mathbf{x} - \mathbf{p}$, der in die Richtung der Normalen zeigt. Damit ergibt sich

$$\mathrm{s}(\mathbf{x}) = \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) \tag{45}$$

In beiden Fällen ist s positiv für Punkte im Halbraum, in den die Normale zeigt, und negativ sonst. Als Distanzfeld teilt diese Formel also den Raum in zwei Hälften. Falls nur die Ebene selbst das Objekt ist, kann der Betrag genommen werden.

Polyeder Ein Polyeder ist ein Körper, der durch eine Menge von Ebenen begrenzt wird. Aus dieser Definition folgt sofort eine Konstruktionsvorschrift mit Distanzfeldern. Nach vorherigem Abschritt lassen sich Ebenen konstruieren. Deren Normalen werden so ausgerichtet, dass sie aus dem Objekt heraus zeigen. Dadurch ist ein Polyeder einfach die Schnittmenge all dieser Grenzebenen.

Quader Eine einfache Möglichkeit der Berechnung wäre es, die Distanz des Punktes zu jeder der Flächen des Quaders zu berechnen. Die kleinste dieser Distanzen wäre dann auch die Distanz des Punktes zum Quader. Alternativ entspricht ein Quader der Schnittmenge von sechs Halbräumen, gegeben durch Ebenen, als Spezialfall eines Polyeders. Eine weitere Möglichkeit, welche die besonderen Eigenschaften eines Quaders ausnutzt, wird im Folgenden hergeleitet. Die Formel basiert auf der in [15].
Die erste wichtige Feststellung ist, dass die Achsen des Quaders parallel zu den Koordinatenachsen verlaufen (hier wird kein rotierter oder verzerrter Quader betrachtet). Der Mittelpunkt befinde sich im Ursprung. Dadurch existiert auch eine Symmetrie. Die Distanzberechnung kann somit auf den komplett positiven Bereich der Koordinaten verlagert werden. Dies wird erreicht, indem jede Komponente des Punktes durch ihren Betrag ersetzt wird.

$$\mathbf{x}'_i = |\mathbf{x}_i| \tag{46}$$

Der nächste Schritt besteht darin, die Kante des Quaders als Ursprung eines neuen Koordinatensystems zu sehen. Der vorher errechnete Betragsvektor kann in dieses System transformiert werden, indem die Kantenpunkt Koordinaten $\mathbf{q}$ abgezogen werden.

$$\mathbf{x}'' = \mathbf{x}' - \mathbf{q} \tag{47}$$

Der Vorteil dieses Systems ist, dass sich das innere des Quaders nur in dem Teilraum befinden kann, in dem alle Koordinaten negativ sind. Dies wird in Abbildung 5 anhand eines Rechtecks gezeigt. Liegt der Punkt nun inner-

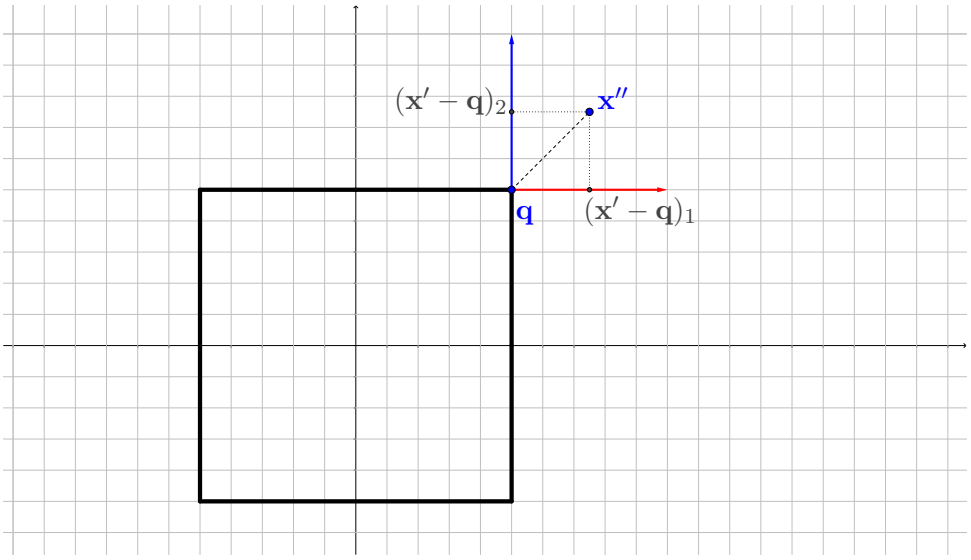


Abbildung 5: Veranschaulichung der genutzten Geometrie eines Hyperquaders in zwei Dimensionen.

halb des Hyperquaders, also sind alle seine Koordinaten negativ, so findet sich der Abstand als der kürzeste Abstand zu den n Koordinatenflächen. Das ist allerdings nichts anderes als die Beträge der einzelnen Koordinaten. Da diese alle negativ sind, entspricht der kleinste Betrag der größten Koordinate. Um diesen Fall für alle Bereiche außerhalb des Hyperquaders verschwinden zu lassen, kann eine Minimums-Operation auf das Ergebnis mit 0 angewandt werden, in diesem Fall ist das Maximum im vorhergehenden Schritt positiv und damit ungültig. Dieser innere Abstand wird weiterhin $\mathrm{s_{in}}$ genannt.

$$\mathrm{s_{in}}(\mathbf{x}) = \min(0, \max_{j=1\ldots n} \mathbf{x}''_j) \tag{48}$$

Als nächstes wird der Term für den äußeren Abstand hergeleitet. Zuerst wird der dem vorhergehenden entgegengesetzte Fall betrachtet, bei dem

alle Koordinaten positiv sind. Die Distanz ist einfach der Abstand zum am weitest außen liegenden Punkt, dem Eckpunkt $\mathbf{q}$. Dies entspricht der Länge von $\mathbf{x}''$. Ist eine Komponente des Vektors negativ, so liegt sie bezüglich derjenigen Hyperfläche, deren Normale in der gleichen Komponente ungleich 0 ist, innerhalb. Da der Fall im Inneren bereits mit s_{in} abgefangen ist, kann die Komponente ignoriert werden. Die entsprechende Operation ist eine Projektion des Vektors auf die genannte Ebene, welche alle Abstände zu anderen Flächen intakt lässt, da alle Hyperflächen im Hyperquader senkrect aufeinanderstehen. Die tatsächliche Distanz $\mathrm{s}_{\mathrm{out}}$ ist damit die Länge desjenigen Vektors $\mathbf{x}''$, der nach der Projektion aller negativen Komponenten übrig bleibt.

$$\mathbf{x}_i''' = \max(\mathbf{x}_i'', 0) \tag{49}$$

$$\mathrm{s}_{\mathrm{out}}(\mathbf{x}) = ||\mathbf{x}'''|| \tag{50}$$

Die Maximums-Operation sorgt weiterhin dafür, dass $\mathrm{s}_{\mathrm{out}}$ gleich 0 für alle Punkte innerhalb des Hyperquaders wird. Die endgültige signierte Distanzfunktion $\mathrm{s}_{\mathrm{quad}}$ ist die Kombination aus innerer und äußerer Distanz.

$$\mathrm{s}_{\mathrm{quad}}(\mathbf{x}) = \mathrm{s}_{\mathrm{in}}(\mathbf{x}) + \mathrm{s}_{\mathrm{out}}(\mathbf{x}) \tag{51}$$

Torus Ein Torus besteht aus einem Ring, um den an jedem Punkt ein Kreis angebracht ist. Diese Kreise bilden die Oberfläche. Der Torus ist gekennzeichnet durch zwei Parameter c und a [16]. Dabei beschreibt c den Abstand des Ringes zum Mittelpunkt und a den Radius der Kreise. Durch die radiale Symmetrie bezüglich der z-Achse kann der Abstand auf ein zweidimensionales Problem reduziert werden, bei dem nur der Abstand zu einem der Ringkreise bestimmt werden muss. Der Radius in der xy-Ebene ist dabei die Länge des in die Ebene projizierten Ortsvektors. Die zweite Koordinate des 2D Vektors ist die unveränderte z-Koordinate. Der Kreis des Rings liegt im neuen Koordinatensystem an der Position $\binom{c}{0}$. Damit kann die Formel für Kugeln, welche auch für Kreise gilt, genutzt werden. Somit ergibt sich folgende Distanzfunktion für einen Torus:

$$\begin{aligned} \mathrm{s}_{\mathrm{torus}}(\mathbf{x}) &= \left\| \begin{pmatrix} \left\| \boldsymbol{\pi}_{(1,2)}(\mathbf{x}) \right\| \\ \mathbf{x}_z \end{pmatrix} - \begin{pmatrix} c \\ 0 \end{pmatrix} \right\| - a \\ &= \left\| \begin{pmatrix} \left\| \boldsymbol{\pi}_{(1,2)}(\mathbf{x}) \right\| - c \\ \mathbf{x}_z \end{pmatrix} \right\| - a \end{aligned} \tag{52}$$

Zylinder Der darzustellende Zylinder sei so ausgerichtet, dass die runde Grundfläche mit Radius r parallel zur xy-Ebene liegt, mit Mittelpunkt im Ursprung und der Körper senkrecht dazu in Richtung z fortgesetzt wird. Der einfachere Fall ist nun, dass der Zylinder sich ins Unendliche erstreckt. Da der Zylinder konstant in z-Richtung ist, kann lediglich die Projektion

des Punktes in die xy-Ebene betrachtet werden. Die signierte Distanz zum Schnittkreis des Zylinders mit der Ebene kann nun analog zur Kugel berechnet werden.

$$\mathrm{s}(\mathbf{x}) = \left\|\boldsymbol{\pi}_{(1,2)}(\mathbf{x})\right\| - r \tag{53}$$

Für einen abgeschnittenen Zylinder sind zwei verschiedene Fälle zu beachten. Im ersten Fall liegt der Punkt innerhalb des vorher beschriebenen in z-Richtung unbeschränkten Zylinders. Die Höhe sei gegeben durch $2h$, wobei h die Ausdehnung in z und $-z$ beschreibt. Der Abstand zu den beiden dadurch gegebenen Grenzflächen ist durch die Symmetrie

$$|\mathbf{x}_z| - h \tag{54}$$

Falls diese Differenz negativ ist, liegt der Punkt innerhalb und die gesuchte Distanz ist das Maximum aus diesem Wert und dem Abstand zu den Seitenflächen, gegeben durch Gleichung 53. Andernfalls ist die Distanz direkt durch die Differenz gegeben.
Der andere zu betrachtende Fall ist, dass der Punkt auch außerhalb des unbeschränkten Zylinders liegt. Liegt der Punkt in z-Richtung zwischen h und $-h$, ist der Abstand genau derjenige zum Kreis auf der korrespondierenden Ebene, also Formel 53. Liegt der Punkt oberhalb des Körpers, kann ein Vektor vom nächstgelegenen Punkt auf dem Zylinder zum Punkt konstruiert werden. Durch die Symmetrie der Grundfläche lässt sich die Richtung innerhalb der xy-Ebene ignorieren. Die z-Komponente ist lediglich die Höhendifferenz aus Formel 54. Der Abstand ist dementsprechend die Länge des zweidimensionalen Vektors mit den beiden Komponenten 53 und 54. Zur besseren Lesbarkeit seien die vorhergehenden Funktionen wie folgt benannt:

$$\mathrm{c}(\mathbf{x}) = \left\|\boldsymbol{\pi}_{(1,2)}(\mathbf{x})\right\| - r \tag{55}$$

$$\mathrm{dh}(\mathbf{x}) = |\mathbf{x}_z| - h \tag{56}$$

Diese Überlegungen zusammengefasst ergeben die folgende signierte Distanzfunktion:

$$\mathrm{s}(\mathbf{x}) = \begin{cases} \max(\mathrm{c}(\mathbf{x}), \mathrm{dh}(\mathbf{x})) & , \mathrm{c}(\mathbf{x}) \leq 0 \wedge \mathrm{dh}(\mathbf{x}) \leq 0 \\ \mathrm{dh}(\mathbf{x}) & , \mathrm{c}(\mathbf{x}) \leq 0 \wedge \mathrm{dh}(\mathbf{x}) > 0 \\ \mathrm{c}(\mathbf{x}) & , \mathrm{c}(\mathbf{x}) > 0 \wedge \mathrm{dh}(\mathbf{x}) \leq 0 \\ \left\| \begin{pmatrix} \mathrm{c}(\mathbf{x}) & \mathrm{dh}(\mathbf{x}) \end{pmatrix}^T \right\| & , \mathrm{c}(\mathbf{x}) > 0 \wedge \mathrm{dh}(\mathbf{x}) > 0 \end{cases} \tag{57}$$

Für eine elegante Implementierung in GLSL siehe [15].

Dreieck Da generell modellierte Objekte durch Dreiecke gegeben sind, sollte zur Kompatibilität mit solchen Daten auch eine Distanzfunktion für Dreiecke existieren. Verschiedene Methoden existieren für dieses Problem. Eine geometrische Konstruktion verläuft ähnlich zu der Methode beim Zylinder. Falls der Punkt projiziert auf die durch das Dreieck gegebene Ebene innerhalb des Dreiecks liegt, ist die Distanz lediglich diejenige zur Ebene. Sonst kann der am nächsten an der Projektion liegende Punkt auf den Dreieckskanten gesucht werden. Der Abstand ergibt sich dann als Länge dieses nächsten Punktes zum gegebenen Punkt. Eine andere Variante ist in [17] gegeben. Dabei wird in einem Vorberechnungsschritt eine Transformation für jedes Dreieck berechnet, welches dies in eine Standardform bringt. Diese Transformation kann dann auf den gegebenen Punkt angewandt werden, womit die Suche nach dem nächsten Punkt auf den 2D Raum verlegt werden kann. Den Autoren zufolge ist diese Technik bis zu vier mal schneller als die normalen 3D Methoden. Allerdings ist zu beachten, dass eine große Menge an Zusatzdaten vorberechnet und gespeichert werden muss. Eine letzte Variante demonstriert die Nutzbarkeit bisher betrachteter Vorgehensweisen und kann auch als Rezept für komplexere Objekte genutzt werden.
Zuerst sei ein zweidimensionales Dreieck gegeben. Für jede der drei Seiten kann ein Normalenvektor $\mathbf{n}_i, i = 1, \ldots, 3$ berechnet werden, welcher nach außen zeigt. Damit kann die Distanz zu den drei aufgespannten Ebenen E_i errechnet werden mit den zugehörigen Distanzfunktionen s_{E_i}. Das Innere des Dreiecks ist damit die Schnittmenge aus den drei negativen Halbräumen. Also gilt

$$\begin{aligned} \mathrm{s}(\mathbf{x}) &= \mathrm{s}_{E_1 \cap E_2 \cap E_3}(\mathbf{x}) \\ &= \max(\mathrm{s}_{E_1}(\mathbf{x}), \mathrm{s}_{E_2}(\mathbf{x}), \mathrm{s}_{E_3}(\mathbf{x})) \end{aligned} \tag{58}$$

Im Dreidimensionalen müssen zwei Änderungen gemacht werden. Zum einen reicht die Dreieckskante allein nicht aus, um die Ebene zu bestimmen. Dazu kann allerdings die Dreiecksnormale hinzugezogen werden. Bei einheitlicher Drehrichtung der Dreieckspunkte kann auch die Normale direkt so bestimmt werden, dass das Kreuzprodukt aus Seite und Normale stets nach Außen zeigt. Die Anwendung von Formel 58 ergibt in 3D ein unendlich hohes Prisma mit dem Dreieck als Grundfläche. Um das Dreieck zu erhalten, kann dieses Prisma allerdings mit der Ebene E_d, in der das Dreieck liegt, geschnitten werden. Allerdings muss darauf geachtet werden, eine unsignierte Ebenengleichung zu nutzen. Eine unsignierte Distanzfunktion kann aus einer signierten durch Nutzung des Betrags gewonnen werden. Zusammengefasst ergibt sich damit die Formel für das Dreieck in 3D.

$$\begin{aligned} \mathrm{s}(\mathbf{x}) &= \mathrm{s}_{E_1 \cap E_2 \cap E_3 \cap E_d}(\mathbf{x}) \\ &= \max(\mathrm{s}_{E_1}(\mathbf{x}), \mathrm{s}_{E_2}(\mathbf{x}), \mathrm{s}_{E_3}(\mathbf{x}), \mathrm{d}_{E_d}(\mathbf{x})) \end{aligned} \tag{59}$$

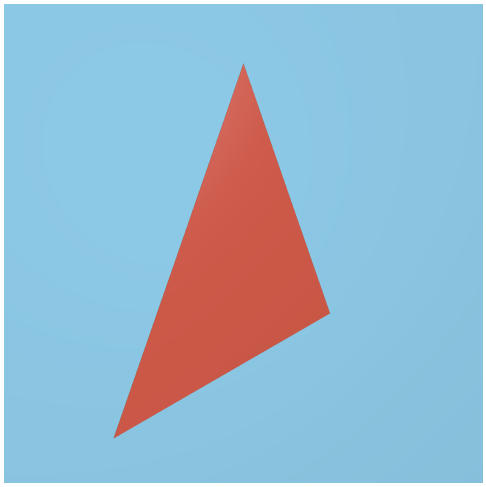


Abbildung 6: Dreieck konstruiert mit dem Distanzfeld aus Formel 59

3.3.2 Punkte und Punktmengen

Das Distanzfeld für einen Punkt $\mathbf{p}$ kann sehr einfach angegeben werden, als der Abstand zu diesem Punkt.

$$\mathrm{d}(\mathbf{x}) = ||\mathbf{p} - \mathbf{x}|| \tag{60}$$

Da es theoretisch unwahrscheinlich ist, einen Punkt im Raum mit einem Strahl exakt zu treffen, ist es wünschenswert, dem Punkt eine Ausdehnung zu geben. Dies kann analog zu Kugeln durch die Einführung eines Radius-Parameters erreicht werden.

$$\mathrm{s}(\mathbf{x}) = ||\mathbf{p} - \mathbf{x}|| - r, r \in \mathbb{R} \tag{61}$$

Die Verallgemeinerung auf Punktmengen P ist dabei trivial. Der Abstand zu einem einzelnen Punkt wird lediglich ersetzt durch die Distanz des Parameters zu seinem nächsten Nachbarn in der Punktmenge. Daraus folgt das signierte Distanzfeld

$$\mathrm{s}(\mathbf{x}) = \min_{\mathbf{p} \in P} ||\mathbf{p} - \mathbf{x}|| - r, r \in \mathbb{R} \tag{62}$$

Mithilfe von Beschleunigungsstrukturen, wie etwa KD-Bäumen, kann diese Funktion auch für große Punktmengen effzient durchgeführt werden.

3.3.3 Eindimensionale Funktionen

Ein weiteres Element zur Modellierung sind eindimensionale Funktionen. Diese können auf verschiedene Weise im Dreidimensionalen dargestellt

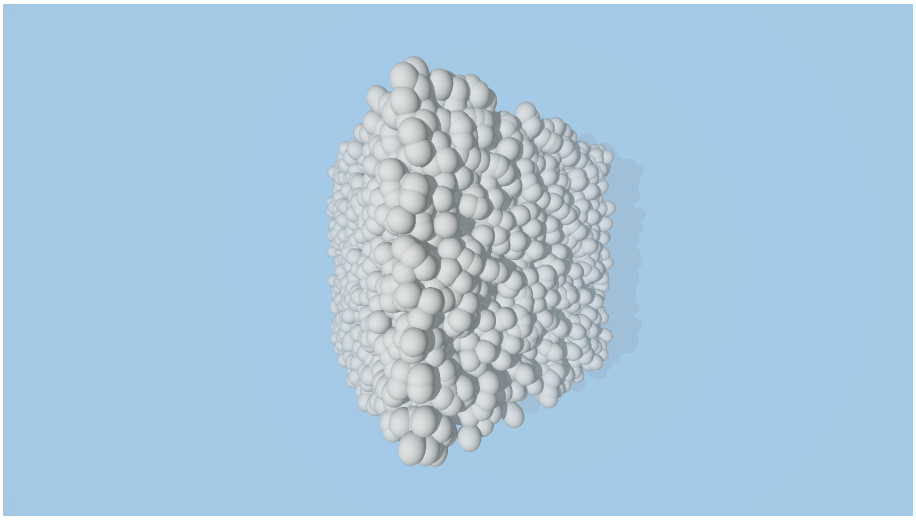


Abbildung 7: Eine Punktmenge, die mit dem in Gleichung 62 definierten Distanzfeld und einem KD-Baum gerendert wurde

werden. Hier werden drei Varianten vorgestellt: Rotation um die x-Achse oder y-Achse und Konstanz in Richtung einer Achse. Alle dieser Versionen bauen auf der Wahl der korrekten Koordinatentransformation auf, um einen dreidimensionalen Punkt in der zweidimensionalen Ebene zu platzieren. Nach diesem Schritt muss eine Distanzfunktion gegeben sein, welche den Abstand des Punktes zum Graphen der Funktion angibt.
Rotation um x-Achse: Dieser Fall entspricht einem klassischen Rotationskörper. Die gegebene Funktion wird dabei um die x-Achse im Raum gedreht und bildet so eine Oberfläche. Der Parameterpunkt spannt zusammen mit der x-Achse eine Ebene auf. Die x-Koordinate des transformierten Punktes entspricht somit der ursprünglichen x-Koordinate. Die y-Koordinate entspricht der Projektion des Punktes auf die in die Ebene gedrehte z-Achse. Hierbei ist zu beachten, dass nur positive Werte der Funktion beachtet werden.
Rotation um z-Achse: Bei dieser Rotation entspricht die z-Koordinate des Punktes der transformierten y-Koordinate. Die neue x-Koordinate entspricht der Länge des in die xy-Ebene projizierten Punktes. Dies geschieht durch ersetzen der letzten Komponente durch 0. Hier muss darauf geachtet werden, dass für nicht symmetrische Funktionen sowohl der positive, als auch der negative Wert der Länge beachtet werden muss. Die Distanz ist dann das Minimum der beiden Distanzwerte.
Konstanz in Richtung einer Achse: Dieser letzte Fall ist der einfachste. Es wird einfach eine der Komponenten ausgelassen.
Jede dieser Methoden benötigt eine Distanzfunktion zu einer Funktion im Zweidimensionalen. Im Folgenden wird ein möglicher Ansatz vorgestellt.

Eine eindimensionale Funktion kann als Kurve im 2D aufgefasst werden.

$$\mathbf{c}(x) = \begin{pmatrix} x \\ \mathrm{f}(x) \end{pmatrix} \tag{63}$$

Für einen gegebenen Punkt $\mathbf{p}$ ist der Differenzvektor $\boldsymbol{\Delta}_\mathbf{p}$

$$\begin{aligned} \boldsymbol{\Delta}_\mathbf{p}(x) &= \mathbf{c}(x) - \mathbf{p} \\ &= \begin{pmatrix} x - \mathbf{p}_x \\ \mathrm{f}(x) - \mathbf{p}_y \end{pmatrix} \end{aligned} \tag{64}$$

Die kleinste Distanz ist für das Argument $x_{\min}$ gegeben, für welches die Länge von $\boldsymbol{\Delta}_\mathbf{p}$ minimal ist. Alternativ kann auch der quadratische Abstand betrachtet werden, was keine Änderung der Lage von $x_{\min}$ zur Folge hat.

$$x_{\min} = \underset{x \in \mathbb{R}}{\arg\min} \,||\, \boldsymbol{\Delta}_\mathbf{p} \,||^2 \tag{65}$$

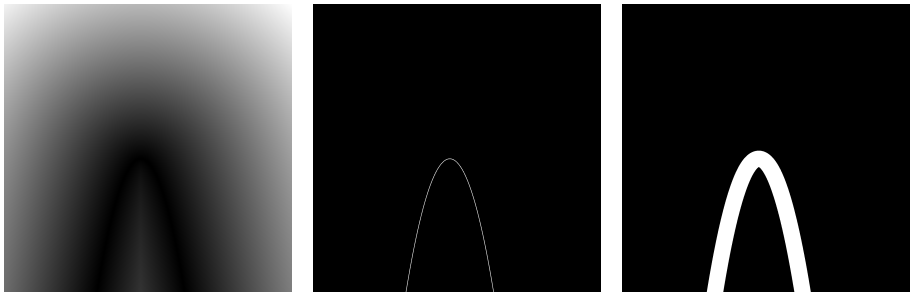
Der tatsächliche Abstand ist dann gegeben durch $||\, \boldsymbol{\Delta}(x_{\min})||$. Die Suche nach diesen Minima erfolgt durch Extremstellensuche. Dazu wird zuerst $\boldsymbol{\Delta}_\mathbf{P}$ nach x abgeleitet.

$$\begin{aligned} \frac{\mathrm{d}||\, \boldsymbol{\Delta}_\mathbf{p}(x)||^2}{\mathrm{d}x} &= \frac{\mathrm{d}}{\mathrm{d}x}((x - \mathbf{p}_x)^2 + (\mathrm{f}(x) - \mathbf{p}_y)^2) \\ &= 2(x - \mathbf{p}_x) + 2(\mathrm{f}(x) - \mathbf{p}_y)\frac{\mathrm{d}\,\mathrm{f}(x)}{\mathrm{d}x} \end{aligned} \tag{66}$$

Anschließend können die Nullstellen gesucht werden. Bei mehreren Nullstellen kann diejenige mit kleinstem Wert von $\|\boldsymbol{\Delta}_\mathbf{p}\|$ gewählt werden.

$$\begin{aligned} 2(x - \mathbf{p}_x) + 2(\mathrm{f}(x) - \mathbf{p}_y)\frac{\mathrm{d}\,\mathrm{f}(x)}{\mathrm{d}x} &= 0 \\ (x - \mathbf{p}_x) + (\mathrm{f}(x) - \mathbf{p}_y)\frac{\mathrm{d}\,\mathrm{f}(x)}{\mathrm{d}x} &= 0 \end{aligned} \tag{67}$$

Die Suche nach den Nullstellen kann dabei beliebig verlaufen, beispielsweise mit dem Newton Verfahren. Je nach Funktion könnte auch eine grobe Annäherung schon ausreichende gute Ergebnisse bringen.
Einen Spezialfall bilden dabei Polynome. Zum einen existieren spezialisierte Lösungsverfahren, zum anderen lassen sich schnelle Approximationen generieren. Außerdem existieren bis zum Grad 2 analytische Lösungen. Für Polynome höheren Grades ist dies nicht mehr möglich. Angenommen, ein gegebenes Polynom habe Grad n. Dann ist $||\, \boldsymbol{\Delta}_\mathbf{p} \,||^2$ vom Grad $2n$. Die Ableitung führt dadurch zum Grad $2n - 1$. Es ist bekannt, dass sich Nullstellen von Polynomen mit Grad höher als 4 nicht mehr algebraisch schreiben lassen. Für ein kubisches Polynom ergibt sich als Nullstellenfunktion ein Polynom vom Grad $2 * 3 - 1 = 5$, was nicht mehr geschlossen lösbar ist.



(a) Distanzfeld

(b) Plot generiert mit Distanzfeld

(c) Plot mit höherer Breite generiert aus gleichem Distanzfeld

Abbildung 8: Distanzfeld der in diesem Abschnitt hergeleiteten Distanzformel für ein quadratisches Polynom. Generierte Plots verschiedener Breite anhand des Distanzfeldes

Ein allgemeines quadratisches Polynom q ist gegeben als

$$\mathrm{q}(x) = ax^2 + bx + c \tag{68}$$

Damit ergibt sich $\boldsymbol{\Delta}_{\mathbf{p}}$

$$\boldsymbol{\Delta}_{\mathbf{p}}(x) = \begin{pmatrix} x - \mathbf{p}_x \\ ax^2 + bx + c - \mathbf{p}_y \end{pmatrix} \tag{69}$$

Nach Formel 67 ist folgendes Nullstellenproblem zu lösen:

$$\begin{aligned} (x - \mathbf{p}_x) + (ax^2 + bx + c - \mathbf{p}_y)(2ax + b) &= 0 \\ (2a^2)x^3 + (3ab)x^2 + (1 + 2ac - 2a\mathbf{p}_y + b^2)x + (bc - b\mathbf{p}_y - \mathbf{p}_x) &= 0 \\ x^3 + \frac{3b}{2a}x^2 + (\frac{1 - b^2}{2a^2} + \frac{c - \mathbf{p}_y}{a})x + \frac{(bc - b\mathbf{p}_y - \mathbf{p}_x)}{2a^2} &= 0 \end{aligned} \tag{70}$$

Die letzte Umformung ist dabei gültig, da das Polynom vom Grad 2 und somit $a \neq 0$ ist. Dadurch ist ein neues Polynom $x^3 + rx^2 + sx + t$ bestimmt, mit

$$r = \frac{3b}{2a} \tag{71}$$

$$s = \frac{1 - b^2}{2a^2} + \frac{c - \mathbf{p}_y}{a} \tag{72}$$

$$t = \frac{(bc - b\mathbf{p}_y - \mathbf{p}_x)}{2a^2} \tag{73}$$

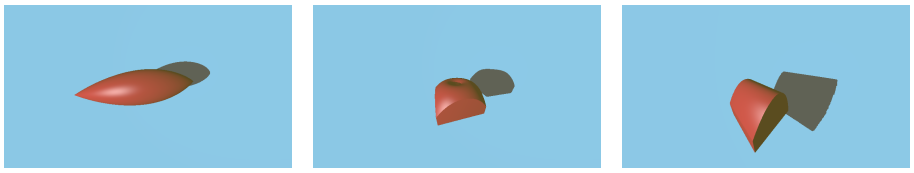
Dieses lässt sich mit Standardmethoden lösen. Eine einfach zu nutzende Methode ist gegeben in Numerical Recipes [18, Kapitel 5.6].

Für ein lineare Polynom $ax + b$ folgt mit gleichen Überlegungen direkt der eindeutige Minimalwert

$$x_{\min} = \frac{2a\mathbf{p}_y + \mathbf{p}_x - ab}{1 + a^2} \tag{74}$$

Im Falle einer konstanten Funktion c ist das Ergebnis noch einfacher, denn es entspricht einfach dem x-Wert des Punktes.

$$x_{\min} = \mathbf{p}_x \tag{75}$$



(a) Eine um die x-Achse rotierte quadratische Funktion

(b) Eine um die z-Achse rotierte quadratische Funktion

(c) Eine quadratische Funktion in einer Achse fortgesetzt

Abbildung 9: Generierte Bilder eines quadratischen Polynoms mit den drei in diesem Abschnitt aufgeführten Methoden

3.3.4 Quadratische Bézier Kurven

Ein wichtiges Modellierungswerkzeug sind Bézier Kurven. Hier werden quadratische Kurven betrachtet, da diese mit den in 3.3.3 vorgestellten Methoden eine direkte Lösung erlauben. Eine Betrachtung von Verarbeitung und Darstellung von kubischen Kurven mittels Distanzfeldern kann beispielsweise in [19] nachgeschlagen werden. Alternativ lassen sich analog Polynomformeln herleiten, die allerdings nur noch numerisch lösbar sind. Eine quadratische Bézier Kurve $\mathbf{b}$ wird definiert durch drei Punkte $\mathbf{p}^1$, $\mathbf{p}^2$, $\mathbf{p}^3$. Für einen Parameter $t \in [0, 1]$ wird dabei ein Punkt auf der Kurve bestimmt. Dieser ist gegeben durch

$$\mathbf{b}(t) = (1-t)^2\mathbf{p}^1 + 2t(1-t)\mathbf{p}^2 + t^2\mathbf{p}^3 \tag{76}$$

Wie auch bei den eindimensionalen Funktionen zuvor besteht die Suche nach der kleinsten Distanz zur Kurve aus der Minimierung der Länge, beziehungsweise der quadrierten Länge, des Differenzvektors von gegebenem Punkt und Kurvenpunkt.

$$\mathrm{d}(\mathbf{x}) = \min_{t\in[0,1]} ||\,\mathbf{b}(t) - \mathbf{x}|| \tag{77}$$

Die Ableitung der quadrierten Funktion führt wiederum auf ein kubisches Polynom, welches durch geeignete Methoden gelöst werden kann. Die genauen Schritte sollen hier aufgrund ihrer Länge nicht wiedergegeben werden, lediglich das Resultat. Folgende Größen hängen nur von den Kontrollpunkten ab und können somit bei der Definition einer Kurve vorberechnet werden:

$$a = \mathbf{p}^1 \cdot \mathbf{p}^1 - 4\mathbf{p}^1 \cdot \mathbf{p}^2 + 2\mathbf{p}^1 \cdot \mathbf{p}^3 + 4\mathbf{p}^2 \cdot \mathbf{p}^2 - 4\mathbf{p}^2 \cdot \mathbf{p}^3 + \mathbf{p}^3 \cdot \mathbf{p}^3 \tag{78}$$

$$b = 9\mathbf{p}^1 \cdot \mathbf{p}^2 - 3\mathbf{p}^1 \cdot \mathbf{p}^1 - 3\mathbf{p}^1\mathbf{p}^3 - 6\mathbf{p}^2 \cdot \mathbf{p}^2 + 3\mathbf{p}^2 \cdot \mathbf{p}^3 \tag{79}$$

$$c_{\text{const}} = 3\mathbf{p}^1 \cdot \mathbf{p}^1 - 6\mathbf{p}^1 \cdot \mathbf{p}^2 + \mathbf{p}^1\mathbf{p}^3 + 2\mathbf{p}^2 \cdot \mathbf{p}^2 \tag{80}$$

$$d_{\text{const}} = \mathbf{p}^1 \cdot \mathbf{p}^2 - \mathbf{p}^1 \cdot \mathbf{p}^1 \tag{81}$$

Für einen gegebenen Punkt $\mathbf{x}$ lassen sich die beiden Größen c_{var}, d_{var} berechnen als

$$c_{\text{var}} = 2\mathbf{p}^2 \cdot \mathbf{x} - \mathbf{p}^1 \cdot \mathbf{x} - \mathbf{p}^3 \cdot \mathbf{x} \tag{82}$$

$$d_{\text{var}} = \mathbf{p}^1 \cdot \mathbf{x} - \mathbf{p}^2 \cdot \mathbf{x} \tag{83}$$

Damit ergibt sich ein kubisches Polynom in Standardform

$$\begin{aligned} \mathrm{q}(t) &= at^3 + bt^2 + (c_{\text{const}} + c_{\text{var}})t + (d_{\text{const}} + d_{\text{var}}) \\ &= at^3 + bt^2 + ct + d \end{aligned} \tag{84}$$

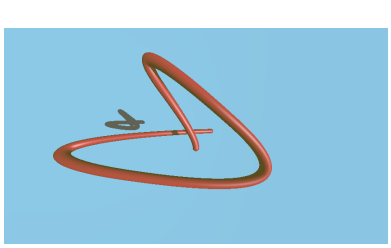
Nach dem Lösen von $\mathrm{q}(t) = 0$ kann die Nullstelle mit geringstem Abstand gefunden werden. Hier ist zu beachten, dass Lösungen außerhalb von $[0, 1]$ auf der ins Unendliche fortgesetzten Kurve liegen. Falls die Grenzen der Kurve respektiert werden sollen, müssen die Lösungen vor dem Auswerten noch auf den korrekten Bereich zugeschnitten werden, beispielsweise durch $x'_{\min} = \min(\max(x_{\min}, 0), 1)$.
Die somit gefundene Distanzfunktion kann noch abgeändert werden, sodass die Kurve eine Ausdehnung erhält. Diese sei eine positive Zahl r. Sei $t_{\min}$ die Lösung mit kleinster Distanz, gewonnen aus dem Polynom in Gleichung 84, dann lässt sich das signierte Distanzfeld berechnen als

$$\begin{aligned} \mathrm{s}(\mathbf{x}) &= \min_{t \in [0,1]} || \mathbf{b}(t) - \mathbf{x}|| \\ &= || \mathbf{b}(t_{\min}) - \mathbf{x}|| - r \end{aligned} \tag{85}$$

Es ist anzumerken, dass die hier genutzten Formeln für beliebige Dimensionen funktionieren, in der Praxis also sowohl in 2D, als auch 3D.

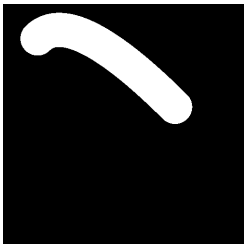
3.3.5 Implizite Oberflächen

Wie bereits in 3.1 angedeutet, werden in diesem Abschnitt implizite Oberflächen betrachtet. Beschrieben werden sie durch implizite Funktionen, also solche, die in der Form $\mathrm{f}(\mathbf{x}) = 0$ geschrieben werden können. Wie bei



(a) Ein Kurvenzug bestehend aus mehreren quadratischen Bézier Kurven im Raum

(b) Ein zweidimensionales Distanzfeld einer Bézier Kurve



(c) Ein durch das Distanzfeld aus 10b erstelltes Bild

Abbildung 10: Beispiele von Bézier Kurven

Distanzfeldern ergibt sich die durch die Funktion gegebene Menge als all jene Punkte, welche die Gleichung erfüllen. Anders gesagt ist die Menge die Nullstellenmenge von f. Wie auch bei signierten Distanzfeldern soll hier die Konvention gelten, dass negative Werte von f dem Inneren und positive dem Äußeren entsprechen. Anders als bei Distanzfeldern entspricht ein Wert von f allerdings nicht unbedingt dem geometrischen Abstand. Beispielsweise entspricht die Gleichung $x^2+y^2+z^2-1=0$ einer Kugeloberfläche, ebenso wie die in 3.3.1 definierte Distanzfunktion $\sqrt{x^2+y^2+z^2}-1=0$, jedoch unterscheiden sich Werte außerhalb der Kugeloberfläche. Ob eine Funktion eine Oberfläche beschreibt, lässt sich anhand des Gradienten bestimmen. Allgemein ist dies ein Fall einer Untermannigfaltigkeit des R^3. Nach [20, Kapitel 1.1] gilt für eine glatte C^∞ k-dimensionale Untermannigfaltigkeit M des $\mathbb{R}^n$:

$\forall p \in M \exists$ eine Umgebung $U \overset{offen}{\subset} \mathbb{R}^{n+k}$ sowie eine C^∞-Abbildung $\mathrm{f} : U \to \mathbb{R}^k$ mit $\mathrm{f}^{-1}(0) = U \cap M$ und surjektiver Ableitung f' an jedem Punkt von $\mathrm{f}^{-1}(0)$.

Die hier betrachteten impliziten Funktionen bilden ab in $\mathbb{R}$, also ist $k = 1$. Der umgebende Raum ist der R^3, also gilt $n = 3 - k = 2$ als Dimension der Oberflächen, wie erwartet. Die Ableitung, hier der Gradient, muss nun für alle Punkte auf der Oberfläche surjektiv auf $\mathbb{R}$ sein. Dies entspricht dem Kriterium, dass der Rang des Gradienten gleich 1 sein muss. Der Gradient darf also nicht $\mathbf{0}$ sein. Alternativ können alle Punkte identifiziert werden, für die $\nabla \mathrm{f}(\mathbf{x}) = \mathbf{0}$ gilt und danach die Zugehörigkeit zur Oberfläche überprüft werden, mit $\mathrm{f}(\mathbf{x}) = 0$, was für keinen Punkt gelten sollte.

Dies soll kurz am Beispiel einer Kugeloberfläche erläutert werden. f wurde bereits oben definiert.

$$\mathrm{f}(\mathbf{x}) = x^2 + y^2 + z^2 - 1 \tag{86}$$

$$\nabla \mathrm{f}(\mathbf{x})^T = \begin{pmatrix} 2x & 2y & 2z \end{pmatrix} \tag{87}$$

$\nabla \mathrm{f}(\mathbf{x})$ ist offensichtlich nur im Ursprung gleich $\mathbf{0}$. Für diesen Punkt gilt allerdings $\mathrm{f}(\mathbf{0}) = 0^2 + 0^2 + 0^2 - 1 = -1 \neq 0$, er liegt also nicht in der Oberfläche. Somit ist die Kugeloberfläche eine glatte 2-Untermannigfaltigkeit des R^3. Im weiteren Verlauf wird davon ausgegangen, dass gegebene Funktionen diese Bedingung erfüllen und keine pathologischen Fälle betrachtet werden müssen. Allerdings ist zu beachten, dass der Gradient durchaus außerhalb der Oberfläche verschwinden kann, wie das obige Beispiel zeigt. Die Erzeugung eines Distanzfeldes beliebiger durch implizite Funktionen beschriebener Oberflächen ist allerdings sehr schwierig und im Allgemeinen auch nicht unbedingt exakt lösbar. Eine Variante ist die Lösung des Eikonals einer Funktion [21]. In allgemeiner Form beschreibt diese partielle Differentialgleichung (PDE) die Ausbreitung einer Oberfläche S in Richtung ihrer Normalen mit einer Geschwindigkeit F. Die relevanten Bedingungen sind dann

$$|\nabla \mathrm{u}| = \frac{1}{F} \tag{88}$$

$$\mathrm{u}|_S = 0 \tag{89}$$

Die zweite Zeile bedeutet, dass die Funktion u, eingeschränkt auf die Oberfläche S, gleich 0 ist. Für den Fall $F = 1$ ist u eine signierte Distanzfunktion. Anschaulich bewegt sich die Oberfläche mit einer Geschwindigkeit von 1, sodass für jeden Zeitschritt δt auch eine Strecke von δt zurückgelegt wurde. Die Oberfläche erreicht also den nächstgelegenen Punkt in der Zeit, die genau der Distanz entspricht. Zur numerischen Lösung existieren verschiedene Methoden, etwa die Fast-Marching-Methode [22]. Diese Lösungsverfahren generieren dabei diskrete Distanzfelder, welche vorberechnet und dann zum Rendern genutzt werden können. Im Folgenden wird eine Methode hergeleitet, welche lediglich die Kenntnis der impliziten Funktion und deren Ableitungen benötigt.

Approximation erster Ordnung: Eine einfache und je nach gewünschter Genauigkeit ausreichende Methode ist die Approximation der Distanzfunktion mittels des Gradienten. Die Herleitung lehnt an [23] an. Die Grundidee besteht daraus, einen beliebigen Punkt $\mathbf{q}$ und denjenigen Punkt $\mathbf{p}$ auf der Oberfläche M zu betrachten, welcher $\mathbf{q}$ am nächsten liegt. Der Abstand dieser Punkte sei $h = ||\mathbf{p} - \mathbf{q}||$. Die Oberfläche sei gegeben durch die glatte implizite Funktion $\mathrm{f}(\mathbf{x}) = 0$. An $\mathbf{p}$ lässt sich eine Normale $\mathbf{n}$ definieren, da $\mathbf{p}$ in der Oberfläche liegt. Da dies auch der nächste Punkt ist, muss der kürzeste Weg in Richtung der Normalen verlaufen. Die Normale sei normalisiert, also $||\mathbf{n}|| = 1$. Dementsprechend lässt sich $\mathbf{q}$ darstellen als $\mathbf{q} = \mathbf{p} + h\mathbf{n}$. Für f lässt sich eine Taylor Expansion um $\mathbf{p}$ schreiben durch

$$\begin{aligned} \mathrm{f}(\mathbf{q}) &= \mathrm{f}(\mathbf{p} + h\mathbf{n}) \\ &= \mathrm{f}(\mathbf{p}) + h\nabla \mathrm{f}(\mathbf{p}) \cdot \mathbf{n} + \mathcal{O}(h^2) \end{aligned} \tag{90}$$

Hier bezeichnet $\mathcal{O}(f)$ die Landau-Notation für Terme maximaler Ordnung f. Analog zu 3.1 entspricht der Gradient auf der Oberfläche von M der Normalen.

$$\mathbf{n}(\mathbf{p}) = \frac{\nabla \mathrm{f}(\mathbf{p})}{||\nabla \mathrm{f}(\mathbf{p})||} \qquad \mathbf{p} \in O \tag{91}$$

Hierbei ist die Annahme, dass der Gradient nicht verschwindet, was prinzipiell für die geforderten Eigenschaften der Oberfläche als Mannigfaltigkeit erfüllt ist. Weiterhin gilt für $\mathbf{p} \in O$: $\mathrm{f}(\mathbf{p}) = 0$. Daraus folgt

$$\begin{aligned} \mathrm{f}(\mathbf{q}) &= \mathrm{f}(\mathbf{p}) + h\nabla \mathrm{f}(\mathbf{p}) \cdot \mathbf{n} + \mathcal{O}(h^2) \\ &= h\nabla \mathrm{f}(\mathbf{p}) \cdot \frac{\nabla \mathrm{f}(\mathbf{p})}{||\nabla \mathrm{f}(\mathbf{p})||} + \mathcal{O}(h^2) \\ &= h\frac{||\nabla \mathrm{f}(\mathbf{p})||^2}{||\nabla \mathrm{f}(\mathbf{p})||} + \mathcal{O}(h^2) \\ &= h||\nabla \mathrm{f}(\mathbf{p})|| + \mathcal{O}(h^2) \end{aligned} \tag{92}$$

Da der Punkt $\mathbf{p}$ nicht bekannt ist, soll der Gradient abgeschätzt werden. Durch die Glattheit von f variiert der Gradient in kleinen Umgebungen nur wenig und lässt sich ausdrücken als

$$\nabla \mathrm{f}(\mathbf{p}) = \nabla \mathrm{f}(\mathbf{q}) + \mathcal{O}(h) \tag{93}$$

Durch diese Annäherung kann es allerdings auch passieren, dass der Gradient verschwindet. In diesem Fall kann beispielsweise direkt der Funktionswert als Distanzschätzung genutzt werden, was natürlich Fehler mit sich bringt. Einsetzen in Gleichung 92 ergibt

$$\begin{aligned} \mathrm{f}(\mathbf{q}) &= h||\nabla \mathrm{f}(\mathbf{p})|| + \mathcal{O}(h^2) \\ &= h||\nabla \mathrm{f}(\mathbf{q}) + \mathcal{O}(h)|| + \mathcal{O}(h^2) \\ &\leq h||\nabla \mathrm{f}(\mathbf{q})|| + \mathcal{O}(h^2) \end{aligned} \tag{94}$$

Da $\nabla \mathrm{f}(\mathbf{q})$ nach Voraussetzung stetig ist, lässt sich der Betrag für eine Umgebung durch eine konstante c abschätzen: $||\nabla \mathrm{f}(\mathbf{q})|| \leq c$. Dementsprechend führt Division durch $||\nabla \mathrm{f}(\mathbf{q})||$ auf

$$\begin{aligned} \mathrm{f}(\mathbf{q}) &\leq h||\nabla \mathrm{f}(\mathbf{q})|| + \mathcal{O}(h^2) \\ \frac{\mathrm{f}(\mathbf{q})}{||\nabla \mathrm{f}(\mathbf{q})||} &\leq h + \mathcal{O}(h^2) \end{aligned} \tag{95}$$

Vernachlässigung der quadratischen Terme ergibt die finale Approximation

$$h \simeq \frac{\mathrm{f}(\mathbf{q})}{||\nabla \mathrm{f}(\mathbf{q})||} \tag{96}$$

Durch den quadratischen Fehlerterm wird auch direkt klar, dass diese Approximation für $h \to 0$ der tatsächlichen Distanz entspricht. Außerdem ist eine exakte Distanzfunktion einer C^k Mannigfaltigkeit selbst wieder C^k [8]. Dadurch ist sichergestellt, dass die tatsächliche Distanz sich in einem kleinen Bereich nicht sehr viel ändert und die Approximation dadurch auch in kleinen Bereich akkurat bleibt.

Approximation zweiter Ordnung: Für eine bessere Approximation, welche auch dann noch funktioniert, falls der Gradient verschwindet, lässt sich eine Taylor Entwicklung zweiter Ordnung verwenden. Wie am Anfang dieses Abschnitts kann dieser Fall schon bei einfachen Objekten wie der Kugel eintreten. Sei $\mathbf{H}_\mathrm{f}$ die Hesse-Matrix von f mit

$$\mathbf{H}_\mathrm{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathrm{f}(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial \mathrm{f}(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{f}(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial \mathrm{f}(\mathbf{x})}{\partial x_n^2} \end{pmatrix} \tag{97}$$

Dadurch lässt sich eine Taylor Entwicklung schreiben als

$$\mathrm{f}(\mathbf{q}) = \mathrm{f}(\mathbf{p}) + h\nabla \mathrm{f}(\mathbf{p})^T \mathbf{n} + \frac{h^2}{2} \mathbf{n}^T \mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n} + \mathcal{O}(h^3) \tag{98}$$

Wiederum ist $\mathrm{f}(\mathbf{p}) = 0$. Des Weiteren wird der $\mathcal{O}(h^3)$ Term direkt ignoriert.

$$\mathrm{f}(\mathbf{q}) \simeq h\nabla \mathrm{f}(\mathbf{p})^T \mathbf{n} + \frac{h^2}{2} \mathbf{n}^T \mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n} \tag{99}$$

Als nächstes wird auf beiden Seiten der Betrag genommen und die Dreiecksungleichung genutzt.

$$\begin{aligned} |\mathrm{f}(\mathbf{q})| &\simeq |h\nabla \mathrm{f}(\mathbf{p})^T \mathbf{n} + \frac{h^2}{2} \mathbf{n}^T \mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n}| \\ &\leq |h\nabla \mathrm{f}(\mathbf{p})^T \mathbf{n}| + |\frac{h^2}{2} \mathbf{n}^T \mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n}| \end{aligned} \tag{100}$$

Da das Skalarprodukt $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$ auch als $\|\mathbf{a}\| \, \|\mathbf{b}\| \cos \alpha$ geschrieben werden kann, mit $|\cos \alpha| < 1$ folgt direkt:

$$\begin{aligned} |\mathbf{a} \cdot \mathbf{b}| &= |\, \|\mathbf{a}\| \, \|\mathbf{b}\| \cos \alpha| \\ &\leq \|\mathbf{a}\| \, \|\mathbf{b}\| \end{aligned} \tag{101}$$

Verbunden mit der Tatsache, dass $\mathbf{n}$ normalisiert ist folgt aus 100

$$\begin{aligned} |h\nabla \mathrm{f}(\mathbf{p})^T \mathbf{n}| + |\frac{h^2}{2} \mathbf{n}^T \mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n}| &\leq |h| \, \|\nabla \mathrm{f}(\mathbf{p})\| \, \|\mathbf{n}\| + \frac{|h|^2}{2} \|\mathbf{n}\| \, \|\mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n}\| \\ &= |h| \, \|\nabla \mathrm{f}(\mathbf{p})\| + \frac{|h|^2}{2} \|\mathbf{H}_\mathrm{f}(\mathbf{p}) \mathbf{n}\| \end{aligned} \tag{102}$$

Um die Größe $||\mathbf{H}_\mathrm{f}(\mathbf{p})\mathbf{n}||$ abzuschätzen, kann eine mit der euklidischen Vektornorm verträgliche Matrixnorm genutzt werden, wie schon in Abschnitt 3.1 beschrieben wurde. Da $\mathrm{f} \in C^\infty$ ist, sind nach dem Satz von Schwarz [24, Kapitel 1.5] die zweiten partiellen Ableitungen in ihrer Reihenfolge vertauschbar. Damit ist $\mathbf{H}_\mathrm{f}$ symmetrisch und die Spektralnorm reduziert sich zu $\|\mathbf{H}_\mathrm{f}\| = |\lambda_\mathrm{max}|$. Hierbei bezeichnet λ_max den betragsmäßig größten Eigenwert von $\mathbf{H}_\mathrm{f}$.
Die Frobeniusnorm ist eine weniger knappe Abschätzung, allerdings leichter zu berechnen. Jedoch lässt sich ein Schätzwert des größten Eigenwertes beispielsweise mit der Potenzmethode annähern. Falls keine zu große Genauigkeit verlangt wird, reichen schon wenige Matrixmultiplikationen aus, was beispielsweise auf der GPU mit optimierter Hardware sehr schnell berechnet werden kann.
Es sei m entweder der Wert $\|\mathbf{H}_\mathrm{f}(\mathbf{p})\|_2$ oder $\|\mathbf{H}_\mathrm{f}(\mathbf{p})\|_F$. Dann gilt

$$\begin{aligned} \|\mathbf{H}_\mathrm{f}(\mathbf{p})\mathbf{n}\| &\leq m \|\mathbf{n}\| \\ &= m \end{aligned} \tag{103}$$

Eingesetzt in Formel 102 ergibt sich

$$|h| \|\nabla \mathrm{f}(\mathbf{p})\| + \frac{|h|^2}{2} \|\mathbf{H}_\mathrm{f}(\mathbf{p})\mathbf{n}\| \leq |h| \|\nabla \mathrm{f}(\mathbf{p})\| + \frac{m|h|^2}{2} \tag{104}$$

Zusammen mit der Ausgangsgleichung 98 führt dieses Ergebnis auf

$$\begin{aligned} \mathrm{f}(\mathbf{q}) &\leq |h| \|\nabla \mathrm{f}(\mathbf{p})\| + \frac{m|h|^2}{2} \\ 0 &\leq |h| \|\nabla \mathrm{f}(\mathbf{p})\| + \frac{m|h|^2}{2} - \mathrm{f}(\mathbf{q}) \end{aligned} \tag{105}$$

Die Approximation für den Betrag von h lässt sich also als Nullstelle einer quadratischen Gleichung schreiben. Hierbei seien, wie auch vorher, Gradient und Hesse-Matrix an der Stelle $\mathbf{p}$ angenähert durch die Werte an der Stelle $\mathbf{q}$.

$$|h| = \sqrt{\frac{\|\nabla \mathrm{f}(\mathbf{q})\|^2}{m^2} + 2\frac{\|\mathrm{f}(\mathbf{q})\|}{m}} - \frac{\|\nabla \mathrm{f}(\mathbf{q})\|}{m} \tag{106}$$

Dies ist auch die einzig mögliche Lösung, da $|h|$ positiv ist. Nach der Vorzeichenregel von Descartes existiert bei dem in Formel 105 stehenden Polynom exakt eine positive Nullstelle, da es einen Vorzeichenwechsel gibt. Dies muss die größere der beiden Lösungen sein, weshalb das Vorzeichen der Wurzel positiv gewählt werden kann. Dieses Ergebnis entspricht demjenigen in [23], wobei dort mit einzelnen partiellen Ableitungen gearbeitet wurde und somit ein Faktor $\frac{1}{2}$ weniger in den quadratischen Termen

steht. Ein Vorteil der Matrix-Variante ist, dass beispielsweise GPU Hardware ausgenutzt werden kann. Außerdem lässt sich die Hesse-Matrix mittels automatischer Differenzierung zusammen mit Gradienten direkt beim Berechnen mitbestimmen. Auch kann mit der Spektralnorm eine bessere Abschätzung gewonnen werden, als es die polynombasierte Abschätzung in [23] erlaubt. Allerdings ermöglicht diese Polynom-Variante eine einfache Möglichkeit der Verallgemeinerung für höhere Ableitungen, wobei vermutlich für die meisten Anwendungen die quadratische Approximation ausreichend ist. Sie löst beispielsweise das Problem, dass der Gradient für einen Punkt verschwindet, solange die Hesse-Matrix dies nicht auch tut. Weitere Untersuchungen könnten in dieser Richtung gemacht werden.

3.3.6 Fraktale

Fraktale sind eine besonders aus ästhetischer Sicht interessante Klasse von Objekten für die Computergrafik. Bilder der Mandelbrotmenge sind oft ein Beispiel für die Schönheit verborgen in mathematischen Formeln. Bekannt geworden durch Benoit B. Mandelbrot eignen sich Fraktale zur Beschreibung vieler komplexer in der Natur vorkommender Phänomene. In "How Long Is the Coast of Britain" [25] wird das Phänomen beschrieben, dass die Länge einer Küstenlinie nicht genau bestimmbar ist und davon abhängt, wie groß der kleinste Maßstab zur Messung ist. Je kleiner dieser gewählt wird, desto länger wird die vermessene Küste. Dies entspricht nicht der Vorstellung, dass eine Küste als natürliches Objekt nur eine begrenzte Länge hat, also mathematisch eine rektifizierbare Kurve ist. Mandelbrot führte dafür den Begriff der *fraktalen Dimension* ein. Objekte können demnach auch eine Dimensionszahl besitzen, welche keine ganze Zahl ist. So ist die Dimension der britischen Küste etwa $1,25$, was den Umstand beschreibt, dass sie wie eine eindimensionale Kurve aussieht, sich aber etwas anders verhält. In diesem Zusammenhang ist auch die Peano Kurve zu nennen, welche jeden Punkt eines Quadrats, einer zweidimensionalen Fläche, erreicht. Eine vollständige Beschreibung von Fraktalen ist im Rahmen dieser Arbeit nicht möglich. Für eine umfangreiche Einführung in das Thema, siehe etwa [26]. Darin werden auch einige Eigenschaften beschrieben, welche oft bei Fraktalen zu finden sind. Eine davon ist die (statistische) Selbstähnlichkeit. Dies bedeutet, dass Teile eines Fraktals aussehen, wie das gesamte Fraktal. Dies wird in Abbildung 11 veranschaulicht. Weiterhin enthalten sie "unendlich viele Details". Damit ist gemeint, dass auf jeder Vergrößerungsstufe Einzelheiten erkennbar sind. Dies steht im Gegensatz etwa zu glatten Kurven oder Oberflächen. Diese sehen nach Definition in genügend kleinen Bereichen genauso aus, wie eine flache Ebene. Bekannt ist hier vor allem die Mandelbrotmenge, die scheinbar endlose verschiedene Formen zeigt, je weiter Bereiche vergrößert werden. Weitere Eigenschaften haben mit der Beschreibung der Fraktale zu tun. Auch wenn es zu erwarten wäre,

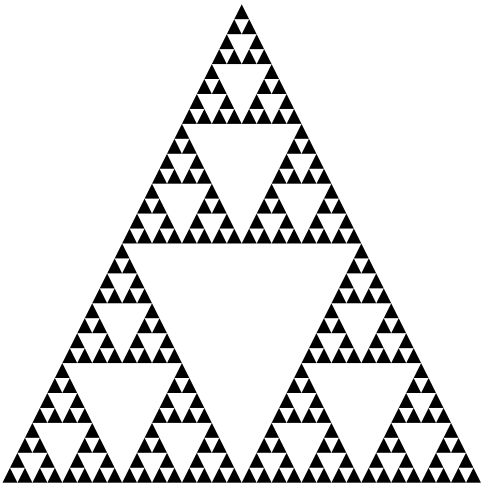
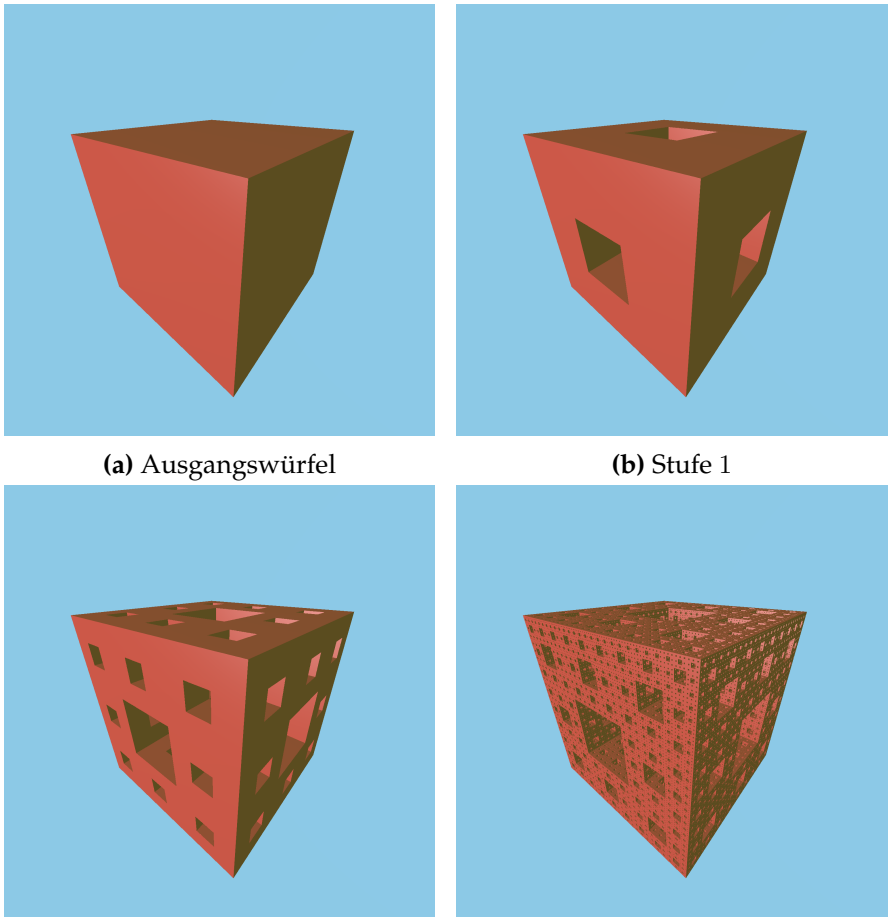


Abbildung 11: Beispiele des Sierpinski Dreiecks

dass sich solch komplexe Objekte nur durch ebenso komplexe Formeln beschreiben lassen, ist das Gegenteil der Fall. Fraktale zeichnen sich oft durch sehr simple Beschreibungen aus. So enthält beispielsweise die Mandelbrotmenge alle komplexen Punkte c, welche bei wiederholter Anwendung der Formel $z^2 + c$ mit initial $z = 0$ nicht gegen unendlich streben. Ein Sierpinski Dreieck entsteht durch wiederholtes Aufteilen eines Dreiecks in 4 Teildreiecke und Entfernen des mittleren. Zusammengefasst entstehen Fraktale oft aus iterativer oder rekursiver Anwendung einer sehr einfachen Vorschrift. Andernfalls lassen sich die entstehenden Formen nicht durch simple bekannte geometrische oder funktionale Beschreibungen ausdrücken. Hieraus wird ersichtlich, was die Probleme für klassische Rendering Verfahren sind. Triangulierung ist schwer möglich und wird unhandlich für hohen Detailgrad. Des Weiteren sind Schnittpunktsuche und Beleuchtung schwierig. Distanzfelder erlauben für einige Fraktale allerdings eine nahtlose Integration. Für Fraktale, welche ähnlich zum Sierpinski Dreieck konstruiert werden, ist eine Distanzfunktion unmittelbar bestimmbar (bis zu einer beliebigen Genauigkeit). Es muss lediglich eine Distanzfunktion der Grundobjekte, etwa Dreiecke, gegeben sein. Das Gesamtobjekt ergibt sich aus einer iterativen Anwendung von Mengenoperationen. Beim Sierpinski Dreieck werden nach und nach Dreiecke von einem Startdreieck subtrahiert, was nur die Berechnung der Dreieckspunkte und Anwendung der Subtraktionsoperation aus 3.2.1 erfordert. Ein bekanntes dreidimensionales Beispiel (bei endlicher Iteration) ist der nach Karl Menger benannte Menger-Schwamm [27]. Als Startobjekt wird ein Würfel genommen. Dessen Seitenflächen werden nun in 27 Würfel zerlegt, also drei Schichten aus jeweils neun Würfeln. Nun wird jeder Würfel, welcher sich in einer Schicht in der Mitte befindet, entfernt. Die Schichten werden dabei sowohl verti-

kal als auch horizontal betrachtet. Damit bleiben noch 20 Würfel übrig, für die dieser Vorgang wiederholt werden kann. Eine andere Variante ist es, eine horizontale und eine vertikale unendliche Säule vom Würfel abzuziehen. Die Grundflächen sind dabei genau $\frac{1}{9}$ der Würfelflächen. Beide Varianten lassen sich einfach mit den bereits betrachteten Distanzfunktionen und Operationen umsetzen. In Abbildung 12 ist diese Art der Konstruktion für mehrere Stufen visualisiert.



(a) Ausgangswürfel **(b)** Stufe 1

(c) Stufe 2 **(d)** Stufe 5

Abbildung 12: Mehrere Stufen der Konstruktion eines Menger-Schwamms über sukzessive Subtraktion von Teilwürfeln

Dieses Konstuktionsverfahren lässt sich allerdings auch nur für solche Objekte nutzen, welche aus Grundprimitiven erzeugt werden. Fraktale, wie etwa Mandelbrot- oder Julia-Mengen erlauben dies nicht. In [28, Kapitel 4.2, Appendix D] sind verschiedene Formeln zur Approximation der Distanz zum Rand der Mandelbrotmenge, beziehungsweise zu einer Julia-

Menge hergeleitet. Eine davon ist gegeben durch

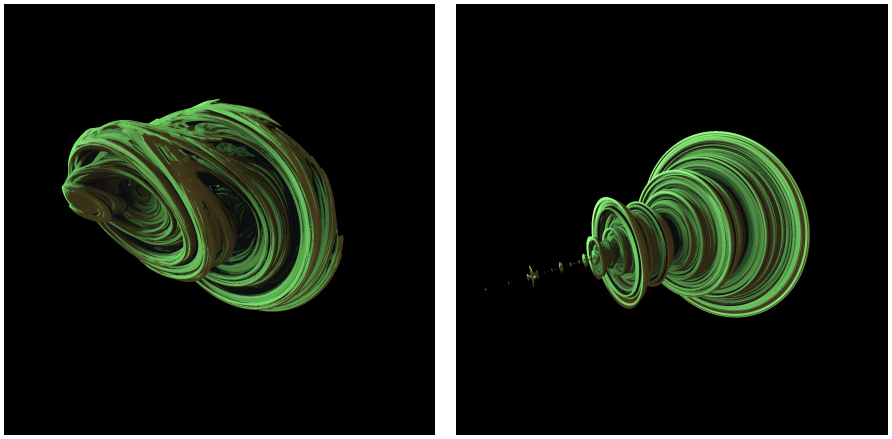
$$\mathrm{s}(z) \approx 2\frac{|z_n|}{|z'_n|}\log|z_n| \tag{107}$$

Hierbei bezeichnet z eine komplexe Zahl. Die z_n sind die Werte der n-ten Anwendung der Iterationsformel für Mandelbrot- oder Julia-Menge. Die Vorgehensweise besteht daraus, die Iteration für eine gegebene Anzahl von Schritten zu wiederholen und aus dem Endergebnis die Distanz zu approximieren. Durch die rekursive Definition der Glieder $z_{n+1} = z_n^2 + c$ lässt sich die Ableitung ebenfalls rekursiv berechnen durch $z'_{n+1} = 2z_n z'_n + 1$, mit $z'_0 = 0$ für eine Mandelbrotmenge und $z'_0 = 1$ für eine Julia-Menge. Da diese Mengen Teilmengen des komplexen Zahlenraumes sind, bieten sie wenig Interessantes für dreidimensionale Darstellungen. Eine einfache Verallgemeinerung auf 3D ist jedoch nicht möglich, da keine 3D Analogie zu den komplexen Zahlen existiert. Abhilfe bieten die Quaternionen. Diese erlauben, analoge Strukturen zu definieren. In [29] werden diese genauer untersucht, auch noch für den Fall der achtdimensionalen Oktonionen. Außerdem wird darin die Gültigkeit der Formel 107 für höhere Dimensionen gezeigt, ebenso wie eine theoretisch fundierte Abschätzung. Dazu sei eine Funktion $\mathbf{F}$ der Form $\mathbf{F}(q) = q^k + c$ gegeben. Die n-fache Anwendung $\mathbf{F}(\mathbf{F}(\dots))$ geschrieben als $\mathbf{F}^n(q)$. q sei dabei ein Quaternion, wobei auch eine komplexe Zahl möglich ist. $D\,\mathbf{F}(q)$ bezeichnet die Ableitung. Dann ist eine approximative Distanz durch folgende Formel gegeben:

$$\mathrm{s}(q) \approx \frac{|\,\mathbf{F}^n(q)|}{|D\,\mathbf{F}^n(q)|} \tag{108}$$

$$\mathrm{s}(q) > \frac{1}{K}\frac{|\,\mathbf{F}^n(q)|}{|D\,\mathbf{F}^n(q)|} \tag{109}$$

K ist eine passend gewählte Konstante. Die Ableitung kann auch in diesem Fall rekursiv berechnet werden als $D\,\mathbf{F}_{n+1}(q) = k\,\mathbf{F}_n\,D\,\mathbf{F}_n$.



(a) Ein 3D Schnitt aus einer Quaternionen-Julia Menge

(b) Ein 3D Schnitt aus einer Quaternionen-Mandelbrot Menge

Abbildung 13: Beispiele von Fraktalen generiert durch Formel 107

Es existieren noch viele weitere Fraktale, auf die hier nicht weiter eingegangen werden kann. Die bisher betrachteten Objekte sind Elemente der Gruppe von deterministischen Fraktalen. Dem entgegen stehen zufällige Fraktale. Als Beispiel sei hier das $1/f^2$-Rauschen genannt, auch als Brownsches- oder braunes Rauschen bekannt. Wie in [28] beschrieben, erfüllt dieses die Eigenschaft der statistischen Selbstähnlichkeit und besitzt eine Dimension von $1,5$. Eine Verallgemeinerung erlaubt auch die Definition von beliebigen Dimensionen zwischen 1 und 2. Weiterhin kann dies auch auf mehrere Dimensionen erweitert werden. Auf diese Weise lassen sich beispielsweise Höhenkarten erstellen, welche natürlichen Bergformationen sehr ähnlich sehen, wie in Abbildung 14 zu sehen ist.

3.3.7 Diskrete Felder

Während die bisher besprochenen Verfahren auf analytischen oder geometrischen Methoden basieren, ist es auch möglich, diskrete Daten darzustellen. Die Quellen für diskrete Daten sind vielfältig. Bereits gegebene Polygonmodelle können konvertiert werden. Medizinische Aufnahmen liefern direkt Volumendaten, wie auch andere Scan-Verfahren. Simulationen werden oft diskret auf einem Gitter ausgeführt, etwa Verformungen eines Körpers. Als letztes Beispiel sei die Abtastung von Funktionen genannt. Darunter fallen beispielsweise die bereits in 3.3.5 besprochenen impliziten Funktionen, aber auch Isoflächen, etwa für Dichtefunktionen von Wolken oder Molekülen. Der Vorteil dieser Variante ist, dass alle Daten in einem einheitlichen Format vorliegen und verarbeitet werden können, ohne dass beispielsweise eine funktionale Beschreibung gegeben sein muss. Ein

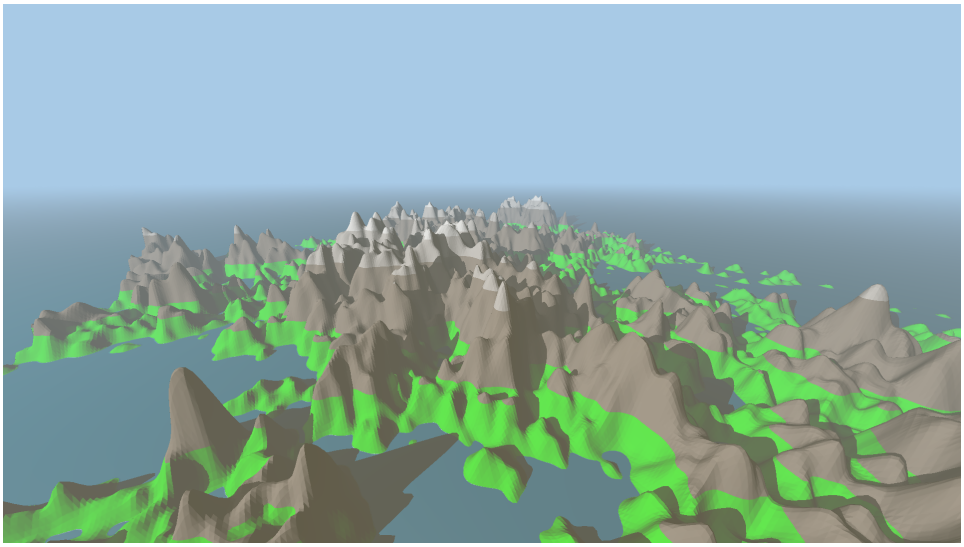
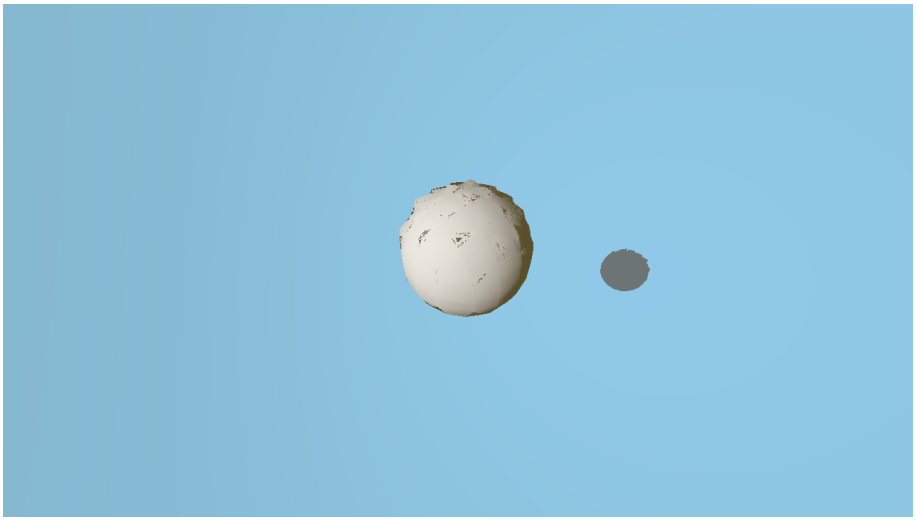


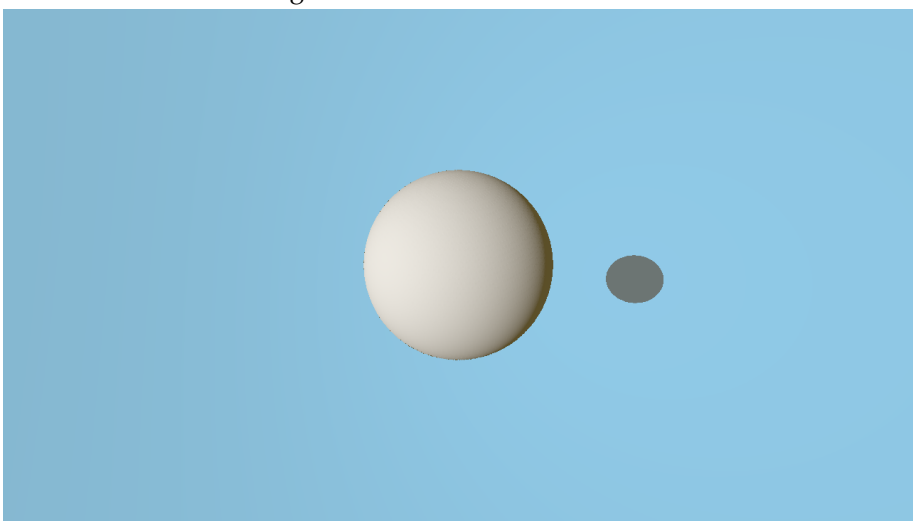
Abbildung 14: Ein zufälliges fraktales Terrain, generiert durch den Algorithmus "SpectralSynthesisFM2D" aus [28, Kapitel 2.6]

großer Nachteil ist generell der Speicherbedarf, da dieser kubisch mit der Auflösung steigt. Ein umfassender Überblick über verschiedene Techniken findet sich beispielsweise in [22]. Die vorhandenen Volumendaten geben allerdings nur die Objektpunkte selbst an, benötigt wird jedoch eine Distanzfunktion. Für implizite Funktionen existieren direkte Lösungsverfahren, welche diskrete Distanzfelder erzeugen. Ein Beispiel dafür ist die Fast-Marching-Methode, welche eine Erweiterung des Dijkstra-Algorithmus' ist. Sie basiert darauf, dass von bekannten Randpunkten des Objektes die Distanz schrittweise ausgebreitet wird. Als Alternative können die Ränder einer impliziten Funktion direkt in einem Binärgrid abgetastet werden. Ausgehend von solch einem Binärgitter, welches allgemein auch für andere vorher genannte Daten erstellt werden kann, lässt sich ein Distanzfeld mit bekannten Algorithmen zur Distanztransformation erzeugen. Hierzu sind etwa die Chamfer Methoden bekannt, welche nicht exakt, dafür aber schnell mit $\mathcal{O}(n)$ sind [22]. Im Vergleich dazu sind Fast-Marching Methoden generell $\mathcal{O}(n \log n)$, wobei auch etwa eine $\mathcal{O}(n)$ Variante existiert. Bei dieser werden Distanzen zur Priorisierung quantifiziert, um eine Tabelle statt einer Priority-Queue zu nutzen, wobei ein vernachlässigbarer Fehler erzeugt wird [30]. Eine weitere Methode ist die Meijster Distanztransformation, welche für 2D definiert wurde, aber auch für höhere Dimensionen verallgemeinert werden kann. Diese hat den Vorteil, exakte euklidische Distanzen in $\mathcal{O}(n)$ zu erzeugen [31, 32]. Das Ergebnis all dieser Verfahren ist ein Gitter, welches die Distanz oder eine Approximation dieser zum Objekt enthält. Dieses Gitter kann dann im Raum positioniert werden. Distanzen für

beliebige Punkte innerhalb einer Zelle können über trilineare Interpolation gewonnen werden. In [33] wird bemerkt, dass sogar niedrig aufgelöste Distanzfelder gute Normalen zur Beleuchtung des Objekts bieten, bei Berechnung des diskreten Gradienten der Gitterzellen. Dieser Umstand und die gute Rekonstruktionsfähigkeit wird in Abbildung 15 verdeutlicht.



(a) Kugeldurchmesser von etwa 6 Voxeln



(b) Kugeldurchmesser von etwa 60 Voxeln

Abbildung 15: Beispiel des Renderings eines diskreten Distanzfeldes. In beiden Fällen wurde das Distanzfeld einer Kugel abgetastet. In 15a entspricht der Kugeldurchmesser gerade einmal ungefähr 6 Voxeln, erlaubt aber bereits eine gute Visualisierung. In 15b ist der Durchmesser etwa 60 Voxel und zeigt bereits eine sehr genaue Rekonstruktion

3.4 Bounding Volumes

Bei größerer Anzahl von Objekten ist es sinnvoll, wie auch bei anderen Darstellungen der Daten, diese zu gruppieren oder anzuordnen, um die Berechnungen zu beschleunigen. Eine Form der Gruppierung sind die sogenannten Bounding Volumes, beziehungsweise Bounding Volume Hierarchies (BVH). Hierbei wird ein Objekt gewählt, welches eine äußere Hülle repräsentiert. Es umgibt eine Menge von weiteren Objekten. Der Abstand zur Hülle ist damit kleiner oder gleich dem Abstand zu jedem enthaltenen Objekt. Sinnvollerweise sollte das Hüllobjekt eine einfache und schnelle Distanzfunktion besitzen. Es bieten sich daher beispielsweise Ebenen, Kugeln oder Quader an. Natürlich können Hüllobjekte weitere Hüllobjekte enthalten, wodurch eine hierarchische Struktur entsteht, die BVH. In Algorithmus 1 wird eine modifizierte Variante zu dem in [9] vorgestellt, welche in ähnlicher Weise auch in [34] erscheint. Dabei wird ein Minimum-Heap verwendet, welcher nach Distanz sortiert. Zur Initialisierung werden alle Grundobjekte hinzugefügt. An oberster Stelle steht nun das Objekt mit der kleinsten Distanz. Falls dies kein Bounding Volume ist, wurde bereits die kleinste Distanz gefunden, denn falls irgendein weiteres Objekt in einem Hüllobjekt existiert, so ist sein Abstand größer als der des Hüllobjekts. Da dieses aber schon weiter als das vorderste Objekt entfernt ist, können die enthaltenen Objekte nicht näher liegen. Falls an vorderster Stelle ein Hüllobjekt liegt, so wird es entfernt. Für alle enthaltenen Objekte wird die Distanz berechnet und sie werden in den Heap eingegliedert. Dieser Vorgang wird solange wiederholt, bis das vorderste Objekt kein Hüllobjekt mehr ist. Falls mindestens ein normales Objekt existiert, terminiert dieser Algorithmus mit der kleinsten Distanz, falls nicht, wird der Heap irgendwann leer werden. Dann kann die Distanz auf ∞ gesetzt werden.
Bounding Volumes können anhand bekannter Ausdehnungen beim Erstellen eines Objektes errechnet werden, im einfachsten Fall zum Beispiel für Kugeln mit bekanntem Radius. Falls keine direkte Formel bekannt ist, etwa nur ein Distanzfeld gegeben ist, kann eine Methode beschrieben in [34] genutzt werden. Dabei wird eine Menge von Ebenen E_i betrachtet, jeweils gekennzeichnet durch einen Normalenvektor $\mathbf{n}^i$. Für einen Punkt $\mathbf{p}$ lässt sich die Distanz, wie auch in 3.3.1 beschrieben, als $d_i = \mathbf{n}^i \cdot \mathbf{p}$ berechnen. Für ein beschränktes Objekt wird es Punkte geben, sodass eine minimale und maximale Distanz angenommen wird. Die Suche nach den Distanzwerten für die umgebenden Ebenen beschränkt sich also auf ein Extremwertproblem mit Einschränkung auf die Punkte des Objektes. Da die Punkte gegeben sind durch eine implizite- oder speziell eine Distanzfunktion der Form $\mathrm{s}(\mathbf{p}) = 0$. Als Lagrange Problem formuliert:

$$\mathrm{h}_i(\mathbf{p}, \lambda) = \mathbf{n}^i \cdot \mathbf{p} + \lambda\, \mathrm{s}(\mathbf{p}) \tag{110}$$

Die gleiche Formulierung für eine Bounding-Sphere:

$$\mathrm{h}(\mathbf{p}, \lambda) = \|\mathbf{p}\| + \lambda\,\mathrm{s}(\mathbf{p}) \tag{111}$$

Das Problem kann mit Standardverfahren für nichtlineare Optimierung mit Gleichheitsnebenbedingungen gelöst werden.

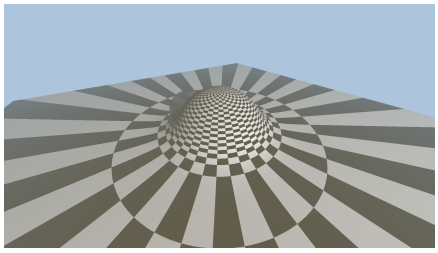
Input : Position $\mathbf{p}$, Menge $\mathrm{ObjectList}$ von Objekten mit zugeordneten Distanzfunktionen s
Output : Minimale Distanz t
$H \leftarrow \mathrm{initializeHeap}()$;
foreach $S \in \mathrm{ObjectList}$ **do**
  $\mathrm{dist} \leftarrow \mathrm{s}_S(\mathbf{p})$;
  $\mathrm{pushWithPriority}(\mathrm{H}, S, \mathrm{dist})$;
end
while $\neg\,\mathrm{empty}(H)$ **do**
  $O \leftarrow \mathrm{popTop}(H)$;
  if $\mathrm{isBoundingVolume}(O)$ **then**
    $\mathrm{SubList} \leftarrow \mathrm{getSubElements}(O)$;
    foreach $S \in \mathrm{SubList}$ **do**
      $\mathrm{dist} \leftarrow \mathrm{s}_S(\mathbf{p})$;
      $\mathrm{pushWithPriority}(\mathrm{H}, S, \mathrm{dist})$;
    end
  else
    return $\mathrm{s}_O(\mathbf{p})$;
  end
end
return ∞ ;

Algorithmus 1 : Algorithmus zum Finden des kleinsten Abstandes in einer BVH basierend auf einer Minimum-Heap Datenstruktur

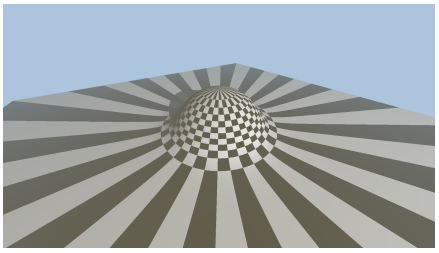
4 Texturierung

Im Gegensatz zu polygonbasierten Verfahren ist es bei der Modellierung durch Distanzfelder generell nicht direkt möglich, Texturen auf bestimmte Oberflächen zu legen. Während gängige 3D Bearbeitungsprogramme das direkte Anpassen der Parametrisierung von Polyygonpunkten zu Texturpunkten erlauben, fällt dies bei Distanzfeldern schwer, da keine direkte Oberflächenrepräsentation gegeben ist. Daher müssen andere Methoden zur Texturierung genutzt werden. Generell stehen als Informationen zur Parametrisierung die Position im Raum, sowie die Normale zur Verfügung, wobei natürlich auch weitere Größen, wie die Krümmung berechnet werden könnten. Ausgehend davon bieten sich verschiedene Varianten an. Ei-

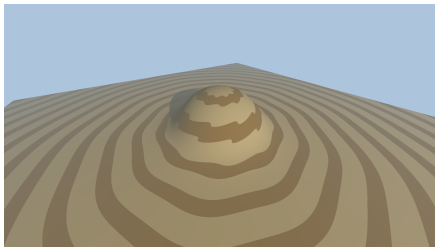
ne davon ähnelt Environment-Maps für Reflektionen. Die Einheitsnormale wird genutzt, um Höhen- und Breitengrad auf der Einheitskugel zu bestimmen. Höhe und Breite können dann als 2D Koordinaten in einer gegebenen Texture genutzt werden. Auf ähnliche Weise lässt sich auch die Weltposition nutzen. Hierbei bieten sich beliebige, auch aus der Kartographie bekannte, Projektionen an. Als Beispiel seien die stereographische Projektion oder die Projektion auf die Randflächen eines Zylinders genannt. Eine weitere Variante sind die sogenannten "Solid Textures" [35]. Hierbei wird eine volumetrische Textur genutzt, entweder als Volumentextur oder direkt als Funktion $\mathbf{t}(\mathbf{x})$. Diese kann beliebige Werte wie Farbe, Rauheit oder Glanz enthalten. Solid Textures bieten sich für Distanzfelder an, da auch automatisch deformierte oder kombinierte Objekte texturiert werden können.



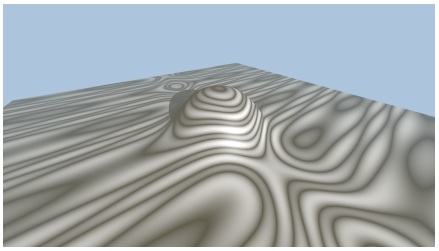
(a) Eine Schachbrett-Textur mithilfe einer Kugelprojektion auf das Objekt gelegt



(b) Eine Schachbrett-Textur mithilfe einer Zylinderprojektion auf das Objekt gelegt



(c) Einfache Solid Texture, welche Ringen in einem Baum nachempfunden ist



(d) Solid Texture mit komplexerer Generierungs-Funktion

Abbildung 16: Beispiele verschiedener Texturierungen eines Objekts

5 Tracing Verfahren

Bisher wurden Distanzfelder an sich betrachtet, also wie sie hergeleitet, kombiniert und verändert werden können. Was noch fehlt, beziehungsweise nur beiläufig erwähnt wurde, ist eine Beschreibung dazu, wie sie explizit auch dargestellt werden können. Das soll der folgende Abschnitt nachholen. Dazu wird ein besonderer Algorithmus, das *Sphere Tracing* vorgestellt. Zuvor sollen kurz weitere Verfahren betrachtet werden.

5.1 Überblick

Eine generelle Variante, ist das Raymarching. Die Idee dahinter ist es, solange auf dem Sichtstrahl zu "wandern", bis ein Objekt erreicht wurde. In der einfachsten Variante ist dabei eine feste Schrittweite vorgegeben. Dies wird in Algorithmus 2 gezeigt. Ein großes direkt offensichtliches Problem

Input : Position $\mathbf{p}$, Richtung $\mathbf{d}$, maximale Distanz $t_{\max}$, Schrittweite s, Funktion zum Prüfen, ob Punkt innerhalb von Objekten ist inside
Output : Distanz t
$t \leftarrow 0$;
while $\neg\,\mathrm{inside}(\mathbf{p} + t\mathbf{d}) \wedge t < t_{\max}$ **do**
  $t \leftarrow t + s$;
end

Algorithmus 2 : Ein simpler Raymarching Algorithmus

dieser Technik ist, dass je nach gewählter Schrittweite zu kleine Strukturen übersprungen werden. Die einzige Möglichkeit, eine bessere Qualität dabei zu erreichen, ist es, die Schrittweite sehr gering zu setzen. Ein Vorteil von dieser Raymarching Variante ist allerdings, dass sie sehr einfach ist und lediglich eine Methode zum Prüfen, ob ein Punkt innerhalb der Objekte liegt, benötigt.
Eine erste Erweiterung besteht daraus, eine geeignete Funktion zu definieren, deren Nullstellen die Oberfläche der Objekte darstellen, also eine implizite Darstellung zu finden, wie in 3.3.5. Damit reduziert sich der Marching-Algorithmus auf das Finden der ersten Nullstelle entlang des Strahls. Dazu existieren verschiedene Methoden, wie das Newtwon-Verfahren oder Intervall-Arithmetik, siehe dazu etwa [36, 37, 38]. Ein Problem dieser Techniken ist, dass meist die Differenzierbarkeit der Funktionen vorrausgesetzt wird und auch eine Ableitung numerisch oder analytisch berechnet werden muss. Dies kann beispielsweise dann ein Problem werden, wenn Operationen wie in 3.2.1 beschrieben implementiert werden sollen. Unter Ausnutzung der besonderen Eigenschaften der Distanzfelder wurde von Hart der Algorithmus Sphere Tracing entwickelt, auf welchen im Folgenden genauer eingegangen wird.

5.2 Sphere Tracing

Das von Hart in [9] vorgestellte Verfahren ist ebenfalls eine spezielle Variante des Raymarchings. Die Grundidee, von der auch der Name "Sphere Tracing" stammt, ist die *Unbounding-Sphere*. In Analogie zu Bounding-Spheres, welche eine Menge von Objekte umschließen, um einen minimalen Abstand für diese zu definieren, geben Unbounding-Spheres einen mi-

nimalen Abstand an, in dem kein Objekt zu finden ist. Sie umschließen also freien Raum. Für einen gegebenen Punkt gibt die Distanzfunktion dabei den Radius einer Unbounding-Sphere an. Während eines Raymarching Durchgangs kann damit die Schrittweite so gewählt werden, dass sie innerhalb dieser Beschränkung liegt. Dadurch ist sichergestellt, dass kein Teil eines Objektes zwischen aktueller Position und neuer liegt. Somit wird kein Schnittpunkt übersehen. Dies wird in Abbildung 17 verdeutlicht.

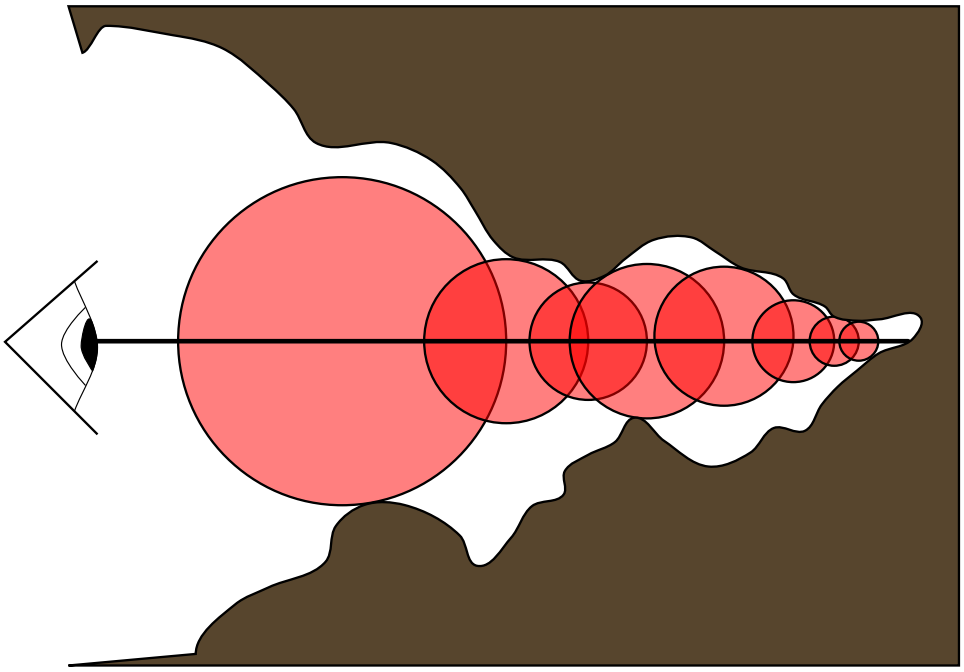


Abbildung 17: Die Idee des Sphere Tracings

Eine wichtige Eigenschaft dieser Methode ist, dass keine Ableitungen benötigt werden, lediglich die Angabe der Distanzfunktion. Dadurch können auch nicht überall differenzierbaren Funktionen wie $\min$ oder $\max$ behandelt werden. Wie bereits in 3.2.1 erwähnt, nutzt Hart Lipschitz-stetige Funktionen. Exakte Distanzfunktionen genügen der Lipschitz Konstante 1. In Gleichung 13 wurde bereits gezeigt, wie die Konstante zur Definition einer Distanzschranke benutzt werden kann. Offensichtlich wird auch dann kein Objekt getroffen, wenn die Schrittweite geringer als der Radius der Unbounding-Sphere ist, weswegen auch eine Distanzschranke ausreichend ist, um den Sphere Tracing Algorithmus auszuführen. Dies kann auch wünschenswert sein, um etwa die Schrittweite im Falle von approximierten Distanzfeldern oder Verzerrungen zu reduzieren, da hier eventuell keine globale obere Schranke existiert. Es wird sonst davon ausgegangen, dass eine gegebene Funktion bereits in ihrer Definition mit der passenden

Lipschitz Konstanten angepasst wurde und stets eine Distanzschranke darstellt. Damit folgt als Erweiterung von Algorithmus 2 das grundlegende Vorgehen nach Hart in Algorithmus 3

Input : Position $\mathbf{p}$, Richtung $\mathbf{d}$, maximale Distanz $t_{\max}$,
Distanzfunktion für minimale Distanz s, Genauigkeit ϵ
Output : Distanz t
$t \leftarrow 0$;
while $t < t_{\max}$ **do**
  $\mathrm{dist} \leftarrow \mathrm{s}(\mathbf{p} + t\mathbf{d})$;
  if $\mathrm{dist} < \epsilon$ **then**
    return t;
  end
  $t \leftarrow t + \mathrm{dist}$;
end
return ∞ ;
Algorithmus 3 : Grundalgorithmus des Sphere Tracing Verfahrens

5.3 Erweiterung

Es existiert eine Erweiterung zum reinen Sphere Tracing, welche in [39] vorgestellt wurde. Eine der Hauptideen ist dabei, ein Overrelaxation Verfahren anzuwenden. Ein Problemfall für Sphere Tracing sind große planare Flächen, die fast parallel zum Sichtstrahl stehen. In diesem Fall wird beim normalen Sphere Tracing bei Sichtstrahlen nah an der Fläche eine sehr kleine Distanz gemeldet. Da aber der Strahl fast parallel ist, wird diese Distanz über große Entfernungen nicht kleiner, obwohl ein Schnittpunkt erst später oder gar nicht existiert. Dieser Umstand wird in Abbildung 18 dargestellt. Wird als Schrittweite nicht die Distanz gewählt, sondern ein Vielfaches, so kann in solchen Fällen offensichtlich viel schneller ein Ergebnis erreicht werden. Da oft eine maximale Anzahl von Schritten vorgegeben ist, kann so verhindert werden, dass der Strahl bei solchen Situationen vorher fälschlicherweise terminiert. Allerdings darf kein beliebiges Vielfaches gewählt werden, da sonst natürlich Objektteile übersprungen werden können. Die Idee der Autoren ist es, aufeinanderfolgende Unbounding Spheres zu betrachten. Schneiden sich diese in einem Bereich, so liegt der gesamte Weg vom ersten zum zweiten Punkt im freien Bereich und der Schritt war ungefährlich. Falls diese sich allerdings nicht schneiden, so wurde eventuell etwas übersprungen. In diesem Fall wird auf den vorhergehenden Punkt zurückgesetzt und auf reguläres Sphere Tracing zurückgegriffen. Weiterhin wurde ein Screenspace basiertes Kriterium vorgestellt, als Ersatz zu einem festem kleinen Wert, ab dem terminiert wird. Das Kriterium besagt dabei, dass der Strahl terminiert werden kann, falls die Distanz kleiner als

die Hälfte eines Pixels ist. Allerdings kann dies Probleme verursachen, falls hohe Deformationen vorliegen. Je nach Verfahren und Rendering-Region wird eventuell eine zu hohe konservative Lipschitz Konstante gewählt, etwa bei einem globalen Optimierungsverfahren, wie beschrieben in 3.2.2. Die nach der Division durch diese Konstante erhaltenen Distanzen können stark unterschätzen, was bei einer Pixelmetrik zu falschen Schnittpunkten führen kann.

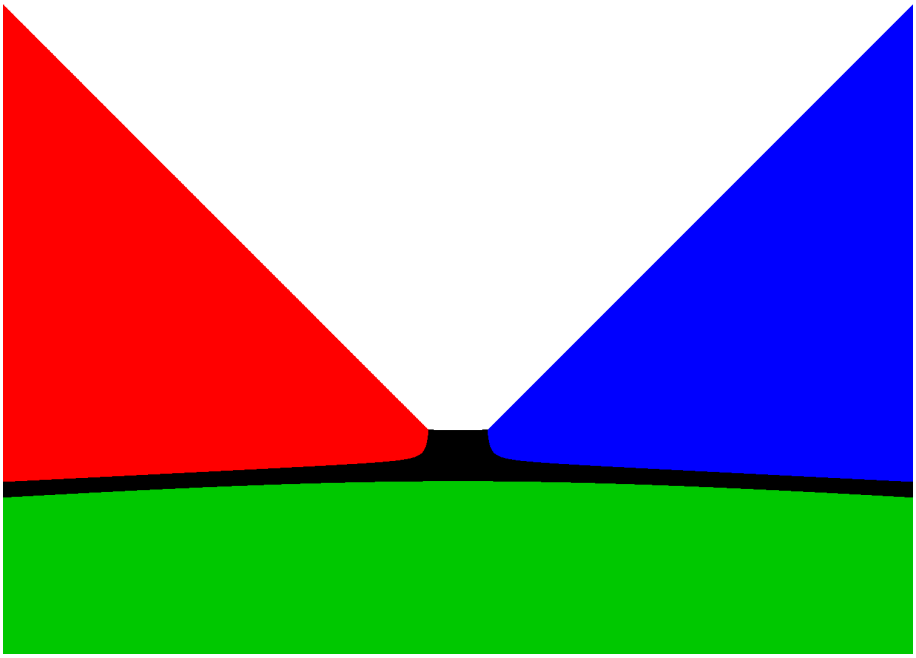


Abbildung 18: Beispiel einer zur Kamera parallelen Ebene. Da kein Ziel erreicht wird, wird der Hintergrund zu früh fälschlicherweise sichtbar

6 Besondere Effekte

Traditionelle Rendering Verfahren haben meist den Nachteil, dass keine oder kaum Informationen über die Umgebung eines Punktes gegeben sind. Distanzfelder allerdings enthalten wichtige Attribute. Wie bereits vorher beschrieben, entspricht der Gradient einer Distanzfunktion an der Oberfläche eines Punktes der Normalen und außerhalb der Normalen der Niveaumenge an dieser Stelle. Die Hesse-Matrix $\mathbf{H}_s$ kann verwendet werden, um Krümmung zu errechnen. Das ergibt sich daraus, dass die Krümmung aus der Änderung der Normalen gewonnen wird. Da die Hesse-Matrix die Ableitung des Gradienten ist, entspricht sie der Ableitung der Normalen. Bei einer exakten Distanzfunktion gilt $\|\nabla s(\mathbf{x})\| = 1$, siehe 3.3.5. Damit entspricht der Gradient der Gauss-Abbildung, die jedem Oberflächenpunkt die Einheitsnormale zuweist und die Hesse-Matrix der Weingarten Ab-

bildung. Die mittlere Krümmung errechnet sich dann als $\frac{1}{2}\,\mathrm{Spur}(\mathbf{H}_\mathrm{s})$ und die Gausskrümmung als $\det(\mathbf{H}_\mathrm{s})$. Hauptkrümmungen können durch die Eigenwerte gewonnen werden. Siehe dazu etwa [22] und Abbildung 19. Im Falle von inexakten Distanzfunktionen sollte der Gradient normalisiert und davon die Ableitung betrachtet werden. Doch auch neben Normalen und Krümmung kann die Distanzfunktion zur Annäherung einiger Effekte genutzt werden, die sonst nur schwer zu berechnen wären, wie im Folgenden genauer beschrieben wird. Allerdings ist zu bemerken, dass diese Techniken eine genaue Distanzfunktion erfordern. Bei Operationen mit starker Verformung oder impliziten Oberflächen kann es daher zu Artefakten kommen.

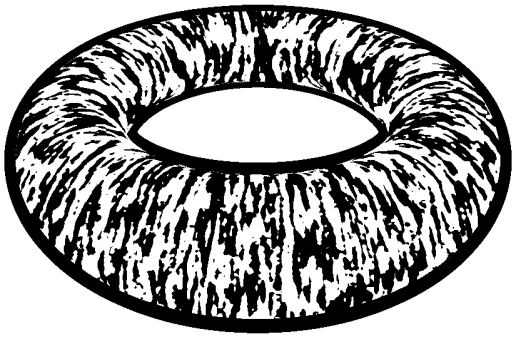


Abbildung 19: Visualisierung der Richtung der größten Krümmung auf einem Torus, gegeben als Eigenvektor zum größten Eigenwert der Hesse-Matrix der Torus-Distanzfunktion

6.1 Weiche Schatten

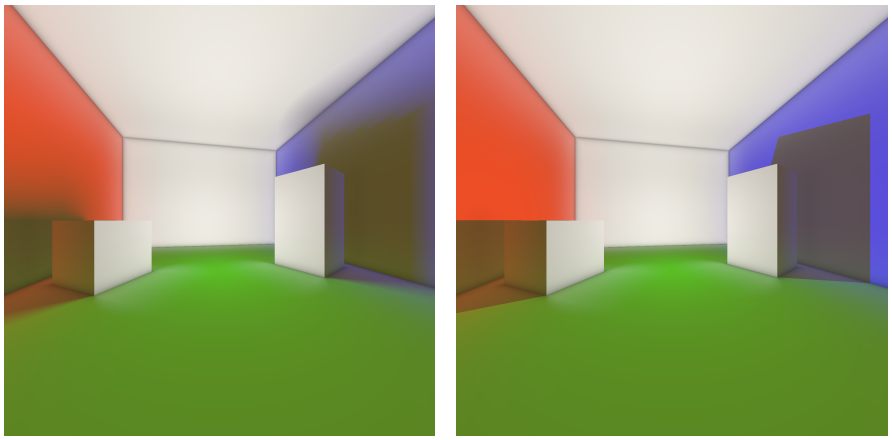
Einer der großen Bereiche der Echzeitgrafik ist die Erzeugung von gut aussehenden weichen Schatten. Hierzu wurden viele Varianten vorgestellt, meist basierend auf Shadow Maps. Generell besteht das Problem, dass viele Samples genutzt werden müssen, um gute Ergebnisse zu erhalten. Hierunter fallen beispielsweise einfaches Percentage-Closer-Filtering oder Variance Shadow Maps. In einem Raytracing System kann eine flächige Lichtquelle an mehreren verschiedenen Punkten abgetastet werden und über einen Strahl auf Sichtbarkeit überprüft werden. Die Ergebnisse können daraufhin zusammengerechnet werden, um eine Annäherung an den Schattierungsanteil zu erhalten. Für einen Überblick über Algorithmen zur Erzeugung weicher Schatten siehe [40] und über harte Schatten mit Shadow Maps [41].

Distanzfelder ermöglichen verschiedene Möglichkeiten, weiche Schatten zu generieren. Die Idee entspricht dem der gerade beschriebenen Schattentaster, allerdings wird nur derjenige Strahl zum Zentrum des Lichts benutzt. Falls eine Verdeckung vorliegt, so wird dieser Strahl stoppen, bevor er das Licht erreicht, der Punkt liegt also im Schatten. Um die Schattenkante herum wird der Strahl allerdings nahe am Objekt vorbei verlaufen. Der minimale Abstand und dessen Position auf dem Strahl erlauben es, eine effektive sichtbare Lichtfläche zu berechnen. Im einfachsten Fall einer genügend großen Lichtkugel entpricht der sichtbare Bereich ungefähr einem Kreis senkrecht zum Strahl an der Minimalposition mit Minimalabstand als Radius. Der Schattierungsfaktor kann dann abhängig von der Position und Größe des Sichtbarkeitskreises bestimmt werden. Wichtig ist jedoch, dass der Minimalabstand nicht direkt bestimmt werden kann, da der zu schattierende Punkt ja auf einem Objekt liegt und dessen Distanzfunktion in dieser Umgebung nahe an 0 ist. Dementsprechend können verschiedene Vereinfachungen vorgenommen werden. Eine davon besteht daraus, eine feste Anzahl von Punkten auf dem Sichtstrahl zu betrachten und deren Distanzwerte mit einer passend gewählten Funktion zusammenzufügen. In [42] werden 6 Punkte vorgeschlagen, wodurch bereits überzeugende Ergebnisse erhalten werden können.
In [2] werden diskrete Distanzfelder zur Schattierung benutzt. Die Idee dahinter ist es, den minimalen Wert der Verdeckung auf der Strecke von Punkt zu Licht zu finden. Die Verdeckung errechnet sich dabei als Verhältnis aus dem aktuellen Radius des größten Kegels zur aktuellen Distanz. Der Radius des Kegels ist dabei der Abstand zum Punkt. Dies kann allerdings zu falschen Ergebnissen führen. Zum einen durch die bereits angesprochenen nicht exakten Distanzfelder, zum anderen die Position des Lichtes. Für ein Licht innerhalb einer Szene wird die Distanzfunktion auch für die Position des Lichts den kleinsten Abstand zur Szene berechnen. Liegt ein Objekt in der Umgebung, beispielsweise nahe hinter dem Licht, betrachtet vom zu testenden Punkt, so wird die Distanz zu dem naheliegenden Objekt am Punkt des Lichtes errechnet. Auch wenn das Licht frei liegt, würde die gerade beschriebene Sichtbarkeitsberechnung stets zu gering ausfallen. Je weiter der Punkt entfernt ist, desto kleiner fiele das Verhältnis von letzter Distanz und Sichtkegelradius (Länge bis zum Licht) aus. Für Richtungslichter existiert dieses Problem offensichtlich nicht. Um dem entgegenzuwirken, müsste der tatsächlich maximale Kegelradius um das Licht herum berechnet werden, eventuell noch verfeinert durch die Größe des Lichts. Dies würde wiederum komplexere Berechnungen erfordern. Zur Vereinfachung könnten die gemessenen Distanzwerte beispielsweise einfach mit einem Faktor vergrößert werden. Für die in Abbildung 20a erzeugten Schatten wurde die Heuristik genutzt, den Distanzwert an der Lichtposition als maximalen Radius zu definieren. Dadurch kann für jeden Messpunkt ein Referenzradius errechnet werden, der dann mit dem

Distanzwert verglichen werden kann. Sei der Messpunkt t Einheiten vom Schnittpunkt entfernt, l die Länge bis zum Licht und d_{light} die Distanz, die beim Licht gemessen wurde. Dann errechnet sich der heuristische Kegelradius r als

$$r = \frac{td_{\text{light}}}{l} \tag{112}$$

Das Resultat ist ein weicher Schatten, welcher im Gegensatz zu sonstigen Verfahren keine einfache Unschärfe ist, sondern geometrisch motiviert ist und nur einen weiteren Strahl benötigt, welcher ohnehin zur Sichtbarkeit der Lichtquelle gebraucht wird.



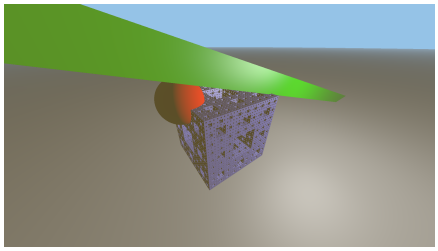
(a) Weiche Schatten **(b)** Genaue Schatten

Abbildung 20: Vergleich von erzeugten weichen und harten Schatten

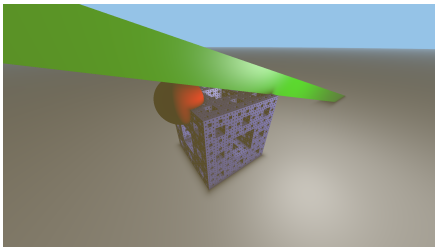
6.2 Ambient Occlusion

In vielen aktuellen Echtzeit-Rendering Systemen ist die Nutzung von sogenannten Ambient Occlusion Verfahren verbreitet, um einen besseren Eindruck von Form und Position von Objekten zu erhalten. Die Grundidee basiert dabei auf der sehr aufwendig zu berechnenden diffusen Beleuchtung. Für einen Oberflächenpunkt wird die Beleuchtung als Integral über die einfallende Beleuchtung aus der in Richtung der Normalen zeigenden Hemisphäre berechnet. Dies ist die bekannte Rendering Gleichung nach Kajiya [3], kurz beschrieben in Abschnitt 2.2. Ambient Occlusion vereinfacht diese dadurch, dass ungefähr berechnet wird, welcher Anteil der Hemisphäre durch Objekte in der näheren Umgebung verdeckt wird. Da Strahlen in diese Richtung nicht direkt in die erleuchtete Umgebung treffen, sondern diese maximal über Sekundärstrahlen erreichen, wird angenommen, dass sie keinen Beitrag zur Beleuchtung liefern.
Es existieren Screenspace Verfahren, welche den Tiefenpuffer einer Szene

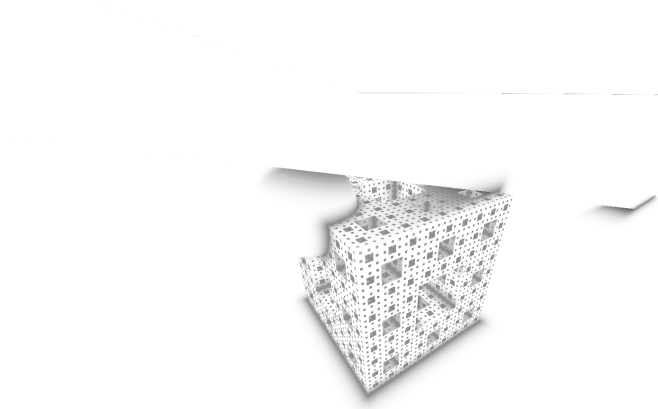
nutzen. Hierbei werden in der einfachsten Variante eine Menge von Punkten in der Hemisphäre ausgewählt. Jeder Punkt kann dabei in den Screenspace transformiert werden, mit seiner korrekten Tiefe. Diese Tiefe kann mit derjenigen aus dem Tiefenpuffer verglichen werden. Ist sie größer, so verdeckt ein anderer Punkt den gewählten. Durch eine gewichtete Kombination der einzelnen Verdeckungen kann so ein Schätzwert für den Anteil der verdeckten Hemisphäre berechnet werden. Dieser Wert wird anschließend mit der weiteren Beleuchtung verrechnet. Nachteile dieser Verfahren sind zum einen die Anzahl an Samples, die für gute Ergebnisse gebraucht wird, sowie die fehlenden Informationen, da lediglich der Tiefenpuffer, aber keine Geometrie genutzt wird. Für einen Überblick über die Entwicklung von Ambient Occlusion, siehe [43] und für eine Beschreibung der ersten Screenspace Variante [44].
In [45] wird eine Variante des Ambient Occlusion mithilfe von Distanzfeldern beschrieben. Im Gegensatz zu Screenspace oder anderen Raytracing Verfahren ist für jeden Punkt genau bekannt, wie weit er von allen Objekten der Szene entfernt ist. Es werden von einem Oberflächenpunkt nun einige wenige nahe Abtastpunkte in Richtung der Normalen gewählt. Ist die Umgebung frei, so entspricht der Abstand vom gewählten Punkt genau dem, welche die Distanzfunktion liefert. Falls aber etwas Verdeckendes in der Nähe ist, so unterscheiden sich beide Werte. Eine gewichtete Summe dieser Unterschiede kann dann zur Verschattung genutzt werden. Offensichtlich erlaubt diese Methode nur die Erkennung von Verschattungen in einem Kegel mit einem Öffnungswinkel von $90°$, da der Durchmesser maximal genauso groß ist, wie die Höhe (der Abstand zum Punkt). In Abbildung 21 werden Ergebnisse dieser Technik dargestellt. Der Mehraufwand dieser Berechnungen waren lediglich 5 Aufrufe der Distanzfunktion. Es ist zu erkennen, dass dieses simple Verfahren bereits stark zum räumlichen Eindruck von Objekten mit vielen Einkerbungen beiträgt.



(a) Szene mit Beleuchtung ohne Schatten



(b) Szene mit Beleuchtung ohne Schatten mit Ambient Occlusion



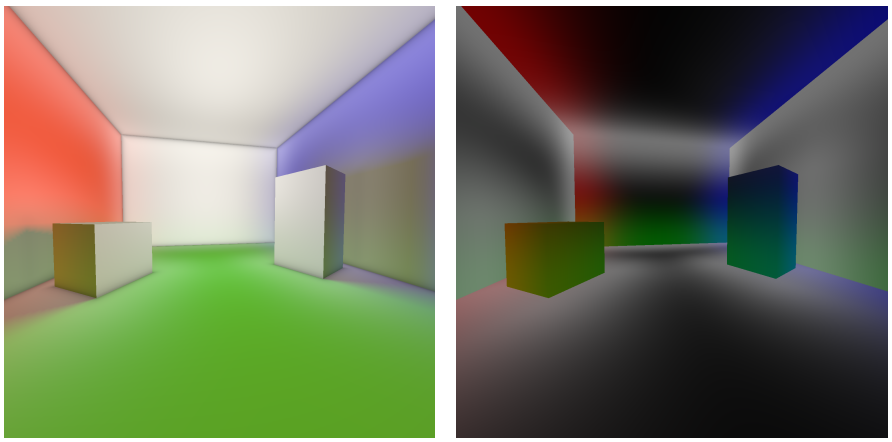
(c) Szene nur mit Ambient Occlusion

Abbildung 21: Darstellung der angehnäherten Ambient Occlusion mit 5 Samples. Auch ohne explizite Schattenberechnungen werden Form und Nähe von Objekten erkennbar

6.3 Indirekte Beleuchtung

In [45] wird neben Ambient Occlusion auch indirekte Beleuchtung berechnet. Beide dort beschriebenen Techniken basieren darauf, die Szene zu voxelisieren und anschließend eine Distanztransformation oder eine Annäherung derer zu berechnen. Die Annäherung besteht aus konsekutivem Blurring und Downsampling. Zur Berechnung der indirekten Beleuchtung werden die einzelnen Farbkanäle getrennt betrachtet. Für einen zu beleuchtenden Punkt können somit verschiedene Distanzwerte pro Farbe gewonnen werden. Diese können dann anhand der Distanzen gewichtet werden und ergeben den indirekten Teil.
Im Folgenden soll eine neue Technik vorgestellt werden. In Abbildung 23 wird das Ergebnis dargestellt. Während eine genaue Berechnung von indirektem Licht sehr aufwendig ist, da theoretisch die ganze Hemisphäre mit sekundären Strahlen abgesucht werden muss, lässt sich der Auf-

bau mit Distanzfunktionen erleichtern. Hierzu seien die einzelnen Distanzfunktionen der in der Szene vorkommenden Objekte gegeben, sowie eine durchschnittliche Farbe. Für einen Oberflächenpunkt ist dabei bekannt, zu welchem Objekt er gehört. Im Folgenden werden Distanzen zu allen anderen Objekten berechnet. Unter der Annahme, dass die Objekte um die nächstgelegenen Punkte eine gewisse Ausdehnung haben, kann davon ausgegangen werden, dass indirekte Beleuchtung von dort empfangen wird. Dementsprechend werden die Objektfarben gewichtet durch eine Funktion ihrer Distanzen, addiert und der Durchschnitt gebildet. Abbildung 22 zeigt das Ergebnis.



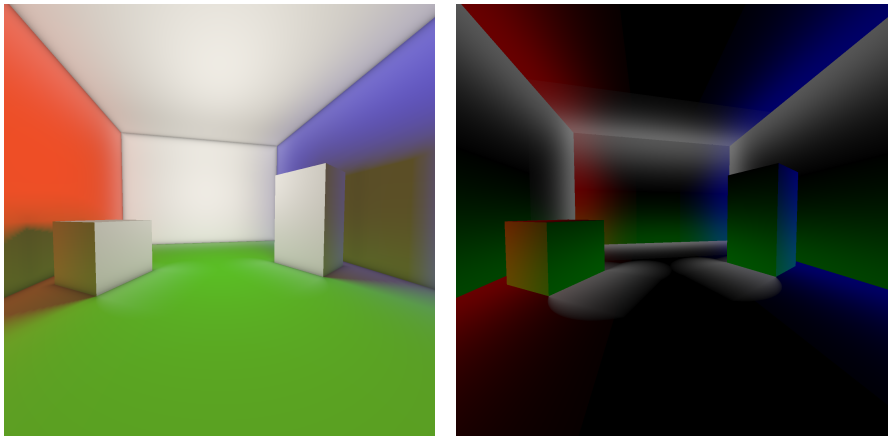
(a) Szene mit direkter und indirekter Beleuchtung

(b) Szene mit indirekter Beleuchtung

Abbildung 22: Darstellung der angehnäherten indirekten Beleuchtung

Allerdings werden zwei Probleme dieser Variante schnell ersichtlich. Zum einen werfen auch schattierte Bereiche Licht auf andere Objekte, wie etwa in den Schlagschatten der beiden Quader zu beobachten ist. Andererseits wird Licht auch durch Objekte hindurch verbreitet, wie auf dem rechten Quader gut zu sehen ist. Die blaue Wand hinter ihm hat Einfluss auf seine vordere Seite, obwohl diese der Wand abgewandt ist. Zur Lösung dieser Probleme kann der Gradient genutzt werden. Falls ein Objekt nah genug ist, um potentiell einen indirekten Beitrag zu leisten, wird der Gradient von dessen Distanzfunktion berechnet. Der Gradient steht senkrecht auf den Niveaumengen der Funktion, welche im nahen Bereich der Form des Objekts entsprechen und sich mit höherer Distanz immer mehr einer Kugelform annähern, zumindest für beschränkte Objekte. Dementsprechend sollte für den Einflussbereich des indirekten Lichts der Gradient der generellen Form des Objekts entsprechen. Ebenso wird der Punkt auf dem Objekt in der Umgebung liegen. Der von da zum Licht zeigende Vektor sollte also ähnlich zu dem am Oberflächenpunkt sein. Dadurch er-

gibt sich die Lösung des ersten Problems daraus, den unbekannten Punkt mithilfe des Lichtvektors und des Gradienten an dieser Stelle grob zu beleuchten. Ein einfaches Skalarprodukt, zwischen 0 und 1 geklemmt, wurde für die hier gezeigten Bilder verwendet. Zur Lösung des zweiten Problems wird genutzt, dass nur solche Flächen direkt Licht austauschen können, deren Normalen in verschiedene Halbräume zeigen. Die Indirektion wird also nur gewertet, falls das Skalarprodukt der Normalen am Punkt und des Gradienten der Distanzfunktion des gewählten Objektes kleiner als 0 sind. Alternativ kann eine Gewichtung gemäß des Skalarproduktes gewählt werden. Das Ergebnis dieser Verbesserungen ist in Abbildung 23 zu sehen.



(a) Szene mit direkter und indirekter Beleuchtung

(b) Szene mit indirekter Beleuchtung

Abbildung 23: Darstellung der angehnäherten indirekten Beleuchtung mit zusätzlicher Gradientenberechnung

Im Gegensatz zur genauen Berechnung der indirekten Beleuchtung, welche viele Abtastungen für gute Ergebnisse braucht, ist dieser Ansatz sehr günstig. Es wird nur ein Aufruf pro Distanzfunktion initial benötigt (ohne Beachtung von Beschleunigungsstrukturen). Für alle Objekte im Einflussbereich, welcher je nach gewählter Gewichtungsfunktion sehr klein sein kann, wird anschließend ein Gradient berechnet, was im einfachsten Fall 8 weitere Aufrufe erfordert. Werden generell nur die c nächsten Objekte aus n Stück mit einbezogen, so ist die Gesamtzahl der Aufrufe $n + 8c$. Im Vergleich dazu muss für eine Strahlenverfolgung an jedem Punkt die minimale Distanz berechnet werden, es wären also n Aufrufe pro Strahlpunkt benötigt, wiederum ohne weitere Beschleunigungsstrukturen. Weiterhin würden mehrere solcher Strahlen berechnet. Der hier vorgestellte Algorithmus benötigt jedoch bei fester Zahl von Objekten für jeden Punkt

maximal $9n$ oder $n + 8c$ Aufrufe, ist also sehr günstig und erzeugt bereits befriedigende Ergebnisse, da indirekte Beleuchtung wenige Details benötigt. Vor allem für Echtzeit-Implementationen kann dies zusammen mit Ambient Occlusion einiges bewirken. Beide Verfahren haben im Gegensatz zu sonst genutzten Screenspace Verfahren den Vorteil, geometriebasiert zu sein und können somit auch in nicht vollständig sichtbaren Bereichen genutzt werden.

7 Implementation

Im Zuge dieser Arbeit wurden zwei Varianten eines Distanzfeld-Raytracers implementiert. Eine Variante ist CPU basiert und wurde in der dynamischen Sprache Julia geschrieben. Diese bietet die Vorzüge von Scriptsprachen, wie dynamische Typisierung, um schnelle prototypische Implementationen zu erlauben. Es ist aber auch möglich, sehr genau typisierten Code zu schreiben, wodurch hohe Geschwindigkeiten im Vergleich zu Sprachen wie Python, Matlab oder Javascript erreicht werden können [46]. Die zweite Variante nutzt WebGL in Kombination mit JavaScript und ermöglicht es, das Tracing Verfahren in Echzeit anzuzeigen. Im Folgenden werden die beiden Implementation genauer erläutert, wobei vor allem auf die GPU Variante eingegangen wird, da sie aufwendiger und interessanter ist. Es werden auch weitere Vorgehensweisen genannt. Die Erklärungen dienen dazu, mögliche Wege aufzuzeigen, wie ein Distanzfeld-Raytracer implementiert werden kann.

7.1 CPU Implementation

Eine CPU-basierte Implementation folgt generell dem gleichen Aufbau wie andere Raytracer. Das grundlegende Objekt ist dabei ein Distanzfeld. Dieses muss lediglich eine Funktion zur Verfügung stellen, welche für einen gegebenen Punkt eine Distanz zurückgibt. Spezialisierungen sind dann die verschiedenen Arten von Distanzfeldern, wie etwa implizite Oberflächen, aber auch die Operationen wie Vereinigung. Jedem Feld kann ein Material zugeordnet werden. Da dies, wie in Abschnitt 4 beschrieben, auf sehr verschiedene Weisen passieren kann, sollte auch ein Material abstrakt gehalten werden. Ein Material muss die verschiedenen Werte, wie etwa Farbe, Glanz oder Emission zur Verfügung stellen. Diese können Abhängig sein von Werten wie Position, Normalen oder Anzahl an Iterationsschritten. Spezialisierungen sind hierbei beispielsweise feste Werte oder Solid Textures. Eine Liste mit Feldern und ihren zugehörigen Materialien ergeben eine Szene. Die wichtigste Funktion ist der eigentliche Strahlenverfolger, wie beschrieben in Abschnitt 5. Diese Funktion nimmt einen Startpunkt mit Richtung, sowie die Szene und liefert den Schnittpunkt, falls vorhan-

den, mit getroffenem Objekt und der berechneten Normalen. Mit dieser Funktion der Strahlenverfolgung, lässt sich ein Distanzfeld-Tracer direkt in bestehende Raytracing-Systeme einbinden.

7.2 GPU Implementation

Während sich eine CPU Implementation eignet, um sehr dynamisch aufgebaute Szenen ohne große Limitierungen darzustellen, etwa mit großer Anzahl von Indirektionen, ist die Berechnung generell sehr langsam und nicht echtzeitfähig, für die Nutzung von Distanzfeldern etwa in der Modellierung oder in Spielen also nicht nutzbar. Es stellt sich die Frage, wie Distanzfelder auf GPUs genutzt werden können. Eine Variante ist es, mit diskreten Feldern zu arbeiten und diese als Voxeltexturen an passende Shader zu übergeben oder als Buffer für Frameworks wie OpenCL oder CUDA. Eine weitere, für diese Ausarbeitung entwickelte Methode, wird in Abschnitt 7.2.2 vorgestellt.

7.2.1 Andere Vorgehensweisen

In [45] wird beschrieben, wie für kleine Objekte ein angenähertes Distanzfeld erzeugt wird, indem zuerst eine 3D Binärtextur gerendert wird und das Ergebnis mehrfach verkleinert und mit Unschärfe gefiltert wird. In [2] wird die Integration von Distanzfeldern in der Unreal Engine 4 beschrieben. Dieses wird vor allem für weiche Schatten, sowie Ambient Occlusion Effekte von statischen Objekten verwendet. Weiterhin bieten sich weitere Möglichkeiten, so wird etwa die Nutzung der Distanzfelder zur Identifikation möglicher Kollisionen bei Partikelsystemen genannt. Für jedes Objekt wird Offline ein niedrig aufgelöstes Distanzfeld generiert, als Beispiel wird ein 50^3 Volumen für einen Baum genannt, welches in 240KB gespeichert werden kann. Mithilfe von Compute Shadern können dynamisch Objekte hinzugefügt oder entfernt werden, sodass das gesamte Tracing Verfahren auf der GPU abläuft.

7.2.2 Codeerzeugung

Das für diese Arbeit implementierte Verfahren nutzt Codeerzeugung, um spezialisierte Shader zu generieren, welche dann beliebige exakte Distanzfelder berechnen können. Dies erlaubt die Flexibilität der CPU Variante, die beispielsweise bei den vorberechneten Diskretisierungen nicht möglich ist, bei schneller Berechnung. Auf CPU Seite wird ein Distanzfeld als Baum repräsentiert. Jeder Knoten im Baum entspricht einem Grundobjekt oder einer Operation und enthält die nötigen Parameter. Grundobjekte wie Kugeln oder Quader sind stets Blätter, während Operationen Kinder besitzen. Ein Beispiel dafür ist in Abbildung 24 zu sehen. In der Implementation ist

es bisher nur möglich ein Gesamtobjekt festzulegen, was sich aber leicht auf mehrere Objekte erweitern lässt. Jeder Knoten muss eine Reihe von

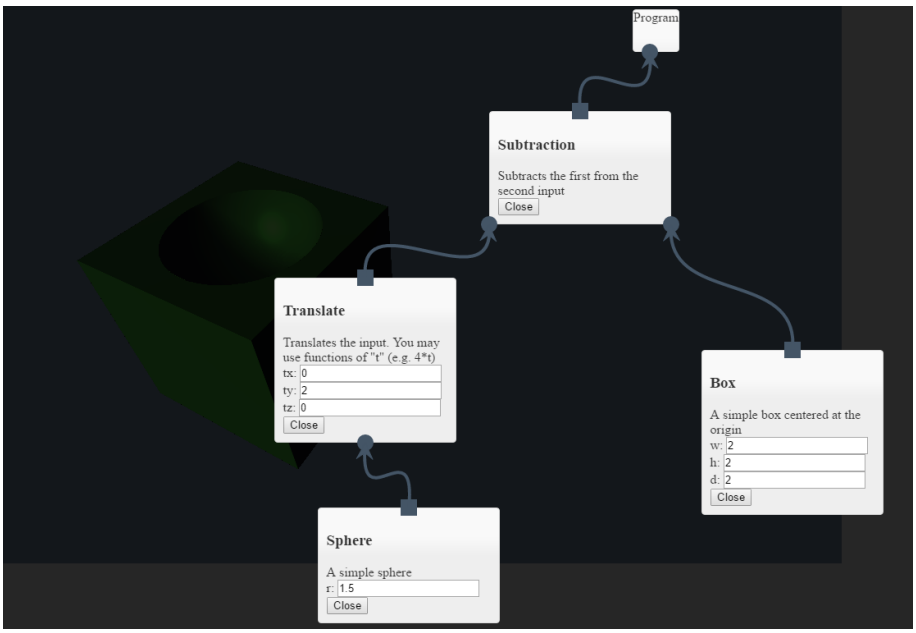


Abbildung 24: Einfaches Beispiel der Baumstruktur eines Distanzfeldes. Es wird eine verschobene Kugel aus einem Quader herausgeschnitten. Es existieren zwei Blattknoten, die Objekte Kugel und Quader. Die unäre Operation "Translation" hat die Kugel als Kind, während die binäre Subtraktion die Translation und den Quader als Kinder besitzt

Operationen definieren. Die wichtigste erzeugt einen String Ausdruck und wird im Folgenden mit *toExpression* bezeichnet. Als Übergabewert dient ein String, der die Variable darstellt. Der Rückgabewert der Methode ist ein valider WebGL/GLSL Ausdruck, welcher eine Distanz berechnet. Operationen rufen dabei die *toExpression* Methode der Kinder auf. Weiterhin definiert ein Knoten mögliche Zusatzfunktionen. Eine Kugel stellt beispielsweise eine Methode zum Berechnen der Distanz zur Kugel zur Verfügung. Zur Erzeugung des gesamten Ausdrucks eines Objektes, wird die *toExpression* Methode des Wurzelknoten aufgerufen, wodurch rekursiv der Baum abgearbeitet wird. Ein Beispiel dafür findet sich in Abbildung 25.

Relevanter Pseudocode:

```
function toExpression(sphere::Sphere,variable::String)
{
  // Sphere ist eine bereitgestellte Funktion und berechnet den
     Abstand zu einer Kugel
  return "Sphere(" + variable + ",vec3("+ sphere.x + "," +
     sphere.y + "," + sphere.z + "),"+sphere.r+")";
}

function toExpression(scale::OpScale,variable::String)
{
  return scale.s + "*" + toExpression(scale.obj,variable + "/" +
      scale.s);
}

// Anlegen der Objekte
sphere = Sphere{x:0, y:0, z:0, r:1};
scale = OpScale{s:2, obj:sphere};

// Ausgabe
print(toExpression(scale,"p"));
```

Ergebniss der Ausgabe:

```
"2*Sphere(p/2,vec3(0,0,0),1)"
```

Abbildung 25: Beispiel einer möglichen *toExpression*-Defintion einer Skalierung und einer Kugel in vereinfachtem Pseudocode. Die Formatierung der Kommazahlen für WebGL wurde hier ignoriert

Dieser Aufbau erlaubt es auch, in WebGL berechenbare mathematische Funktionen direkt in den Code einzuspeisen. In der für diese Arbeit geschriebenen Implementation wurde ein simples Framework für symbolische Mathematik erstellt, um textuelle Eingaben in eine Zwischenform zu bringen, welche direkt in WebGL valide Ausdrücke umgewandelt werden kann. Wird weiterhin beispielsweise eine globale Zeit zur Verfügung gestellt, lassen sich so zeitabhängige Distanzfelder generieren, wodurch Animationen möglich sind. So könnte anstatt einer festen Translation eine Kreisbewegung durch die Translation $(sin(t), cos(t), 0)^T$ erzeugt werden. Solche symbolischen Mathematik Frameworks sind in vielen Sprachen verfügbar, lassen sich also leicht einbauen. Dies ermöglicht auch eine Generierung der nötigen analytischen Ableitungen für implizite Oberflächen, wie in 3.3.5 beschrieben. Alternativ können im Shader relativ leicht Funktionen für automatische Differenzierung bereitgestellt werden, wobei auch simple numerische Verfahren genutzt werden können. Die Variante der automatischen Differenzierung wurde zum Beispiel mit einer eigenen Implementation zur Erzeugung von Abbildung 19 genutzt. Der gesamte Shader wird zusammengebaut, indem ein Grundtemplate, welches Definitionen

wie das Sphere Tracing enthält, mit den Zusatzfunktionen der Knoten und deren Distanzfeldausdrücken zusammengefügt wird. Im einfachsten Fall wird eine ausgerollte Schleife über alle Objekte generiert, welches die minimale Distanz findet, indem jede Distanzfunktion aufgerufen und mit dem aktuellen Wert verglichen wird. Dabei kann auch das Material des Objekts mitbestimmt werden. Die Sphere Trace Methode (und eventuell andere) kann dann generisch diese Schleife nutzen, um die kleinste Distanz unter allen Objekten, sowie das dazu passende Material zu bestimmen. Mehr ist nicht nötig, um die hier vorgestellten Algorithmen zu nutzen. Da Shader dynamisch kompiliert werden, kann auch während der Laufzeit die Szene angepasst werden.

Bewertung des Verfahrens In der hier vorgestellten Version erlaubt das Verfahren bereits das Erstellen von komplexen kleineren Objekten, da Operationen wie Vereinigung oder Subtraktion einfaches Modellieren erlauben. Durch die direkte Erzeugung des Codes kann das Ergebnis auch interaktiv gerendert werden. Mit Effekten wie aus Abschnitt 6 können ansprechende Bilder mit nur leichtem Mehraufwand berechnet werden. Allerdings muss bei jeder Änderung der Szene der zugehörige Shader neu generiert werden. Auch ist die Modellierung vieler oder komplexer Objekte ohne Möglichkeiten der Gruppierung umständlich. Auch ist das Fehlen von Beschleunigungsstrukturen problematisch, da linear alle Distanzfunktionen betrachtet werden, was bei größerer Anzahl sehr ineffizient wird.

Mögliche Verbesserungen Eine mögliche Erweiterung wäre es, Objektparameter in Buffern zu übergeben, und im generierten Code lediglich die passende Stelle im Buffer zu referenzieren. Dadurch könnten bei fester Baumstruktur Parameter geändert werden, ohne, dass neuer Code generiert werden muss. Weiterhin könnten Beschleunigungsstrukturen wie Bounding Volumes direkt im Code erstellt werden, um das lineare Abarbeiten von Objekten zu verbessern. Falls nicht WebGL, sondern modernes OpenGL genutzt wird, könnte eine Volumentextur genutzt werden, um während der Ausführung Distanzwerte zu speichern. In konsekutiven Berechnungsschritten kann dann direkt auf die Textur zugegriffen werden, anstatt möglicherweise komplexe Funktionen aufzurufen. Weiterhin könnte ein einfaches Dateiformat aus den Objektbäumen erstellt werden, beispielsweise ein direkte Speichern im JSON Format. Dadurch könnten erstellte Objekte ausgetauscht werden und etwa in einem Interface als eigene Knoten eingebunden werden. Dies ermöglicht eine modulare Herangehensweise. Da die Objektdefinitionen im Gegensatz zu polygonbasierten Verfahren sehr klein sind, lassen sich auch komplexe Objekte sehr platzsparend speichern. Eventuell lässt sich dieses Verfahren beispielsweise mit solchen aus Abschnitt 7.2.1 kombinieren, sodass sowohl diskrete Felder, als

auch analytische genutzt werden können.

8 Ergebnisse

In diesem Abschnitt werden einige Ergebnisse beschrieben. Alle Beispiele wurden in einem Browser mittels WebGL dargestellt. Die ersten beiden und der letzte Versuch nutzen das Framework *Shadertoy* [47]. Die anderen wurden mit dem für diese Arbeit entwickelten JavaScript basierten Werkzeug erstellt. Das Testgerät ist ein Lenovo Ideapad u430 Touch mit einem Intel Core i5-4200U Prozessor, 8GB RAM und Windows 10 als Betriebssystem. Für die Shadertoy Beispiele wurde die interne Intel HD Graphics 4400 mit dem Chrome Browser, für die anderen die dedizierte NVIDIA GeForce GT 730M mit Firefox genutzt. Es ist zu beachten, dass Messwerte für die Dauer von Zeichenoperationen in Browsern Schwierigkeiten besitzen. So gibt es viele vom Browser kontrollierte Vorgänge, die das Zeichnen betreffen, etwa das *requestAnimationFrame* Framework, welches Aktualisierungen überwacht. Daher sollten die hier vorgestellten Messwerte als grobe Annäherung und genereller Trend verstanden werden.

8.1 Cornell Box mit verschiedenen Effekten

In diesem Beispiel wurde eine ähnliche Szene wie in Abschnitt 6 in Shadertoy erstellt.

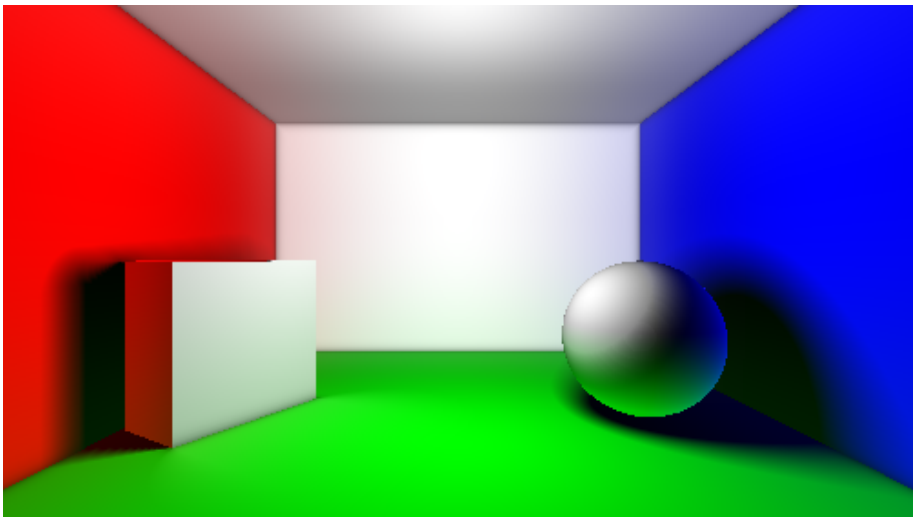


Abbildung 26: Eine Cornell Box artige Szene mit weichen Schatten, Ambient Occlusion und starker indirekter Beleuchtung mit einer Lichtquelle

Der Code wurde so geschrieben, dass er etwa einem generierten Sha-

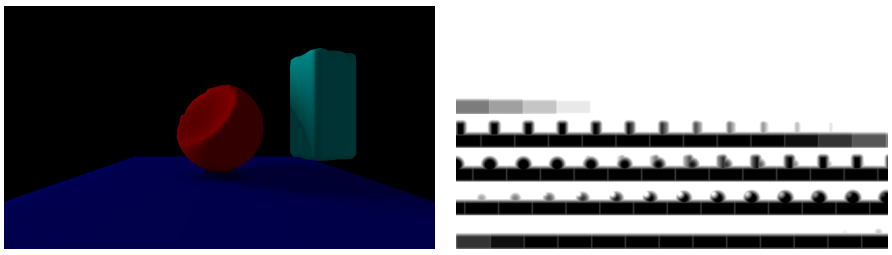
der, wie in Abschnitt 7.2.2 beschrieben, entspricht. Die folgende Tabelle zeigt gemessene FPS Zahlen. Da keine direkte Möglichkeit zur Abfrage einer Zeitreihe besteht, wurden 40 zufällige Screenshots erstellt, von denen die angezeigten FPS als Messwerte genutzt wurden.

Min	Max	Mittelwert	Standardabweichung
42.0	60.2	52.495	6.833

Tabelle 1: Messwerte in FPS

8.2 Voxeltracing

In diesem Beispiel wurde die Nutzung einer Voxelstruktur in einem Shader getestet. Die Voxel enthalten dabei eine gesampelte Distanzfunktion, sowie die Farbe des nächstgelegenen Objekts. Da keine 3D Texturen als Rendertarget zur Verfügung stehen, wird in einem ersten Schritt eine 2D Textur erstellt, mit einzelnen Schichten einer 3D Textur. Beim späteren Darstellen wird aus dieser Textur gelesen. Einzelne Schichten können somit Hardware Interpolation ausnutzen. Die Interpolation zwischen Schichten wird im Shader berechnet. Die generierte Volumentextur, sowie das Ergebnis sind in Abbildung 27 zu sehen. Aufgrund der vorgegebenen Größen in Shadertoy wurde eine Auflösung von 50^3 Voxeln genutzt. Drei Objekte sind in der Szene enthalten. Eine Bodenplatte, eine gedrehte Box und eine Kugel, aus der eine andere Kugel herausgetrennt wurde. Trotz der geringen Auflösung werden die Objekte gut erkennbar dargestellt. Zu beachten ist hier, dass durch die Speicherung in der 2D Textur extra Aufwand für Zugriff und Indizierung entsteht.



(a) Ergebnis des Voxeltracings mit weichen Schatten und Ambient Occlusion

(b) Anordnung der Voxeldaten in der 2D Textur

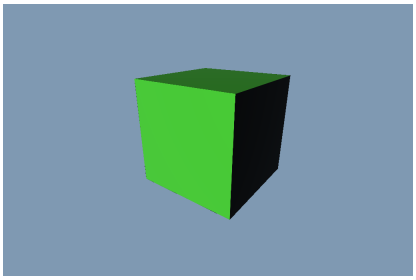
Abbildung 27: Rendering eines abgetasteten Distanzfeldes mit einer Auflösung von 50^3 Voxeln

Min	Max	Mittelwert	Standardabweichung
28.2	53.6	43.968	6.301

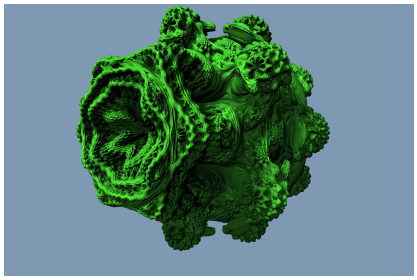
Tabelle 2: Messwerte in FPS

8.3 Verschiedene Szenen

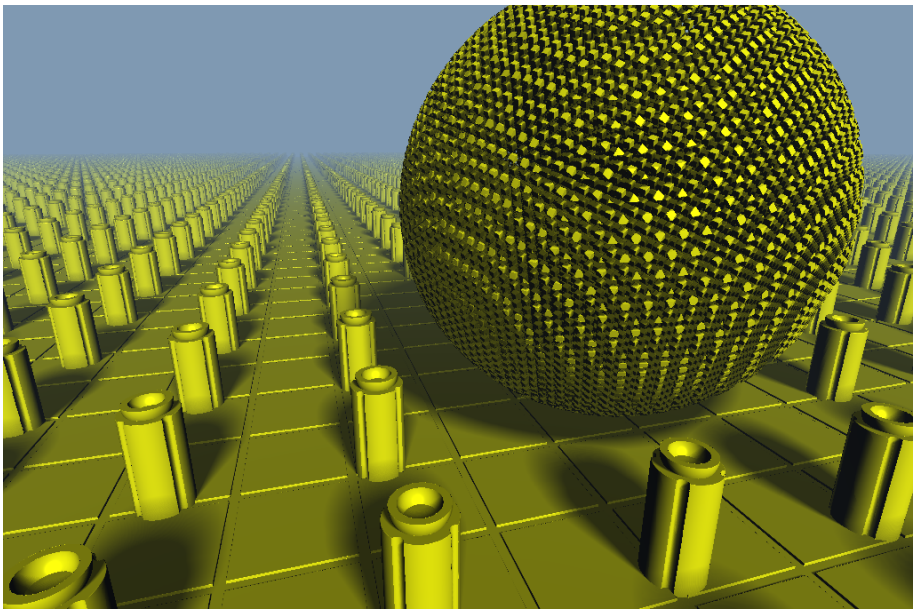
Zum Vergleich wurden drei Szenen mit dem für diese Arbeit entwickelten Editor erstellt. Die erste Szene ist sehr einfach und besteht nur aus einem Quader. Die zweite Szene zeigt ein komplexes Fraktal, Mandelbulb genannt. In der letzten Szene wurden verschiedene Bestandteile durch Kombination mehrerer Objekte erzeugt und zu einem komplexen Objekt zusammengefügt. Die FPS Ergebnisse wurden dabei automatisch berechnet und über eine Zeitreihe hin gemittelt.



(a) Ein Quader



(b) Ein Mandelbulb Fraktal



(c) Komplexe Szene mit vielen Bestandteilen

Abbildung 28: Die verschiedenen Szenen

	Min	Max	Mittelwert	Standardabweichung
Box	51.232	60.217	59.761	1.057
Mandelbulb	10.586	16.537	14.253	0.79
Komplexe Szene	14.706	18.719	16.919	0.650

Tabelle 3: Messwerte der drei Szenen in FPS

8.4 Implizite Oberfläche

Das letzte Beispiel zeigt den Effekt, den das Verfahren mit zweiter Ordnung aus Abschnitt 3.3.5 haben kann. Hierzu wurde als erstes Beispiel ein Objekt

mit folgender Formel genutzt:

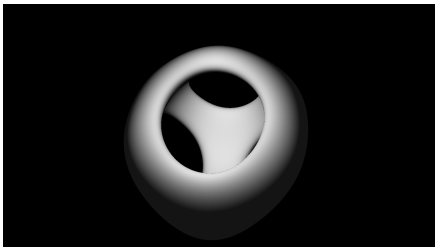
$$\begin{aligned}(x-2)^2(x+2)^2 + (y-2)^2(y+2)^2 + (z-2)^2(z+2)^2 \\ +3(x^2y^2 + x^2z^2 + y^2z^2) + 6xyz \\ -10(x^2 + y^2 + z^2) + 22 = 0\end{aligned} \tag{113}$$

Ein zweites Beispiel ist gegeben durch:

$$\sin x \sin y \sin z + \sin x \cos y \cos z + \cos x \sin y \cos z + \cos x \cos y \sin z = 0 \tag{114}$$

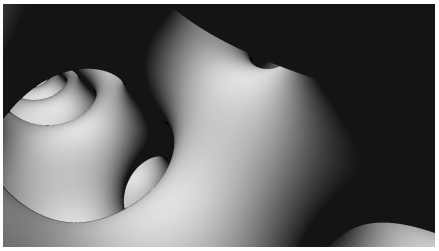
Das letzte Beispiel zeigt einen weiteren Anwendungsfall. Da auch exakte Distanzfelder implizite Oberflächen sind, lässt sich die Approximation auch darauf anwenden, beziehungsweise auf ein transformiertes Distanzfeld. So wurde als Ausgangsobjekt eine Kugel genutzt und auf diese verschiedene Störfunktionen, die abhängig von der Weltposition sind, addiert. Statt wie in Abschnitt 3.2.1 eine Lipschitz Konstante zu bestimmen, wurde die implizite Approximation genutzt.
Bei allen Beispielen zeigten sich bei sonst exakt gleichen Verhältnissen Löcher in der Darstellung mit der Approximation erster Ordnung, eventuell zurückzuführen auf sehr kleine Gradientenwerte. Die Distanzapproximation zweiter Ordnung hat dieses Problem nicht.

(a) Approximation mit Gradient

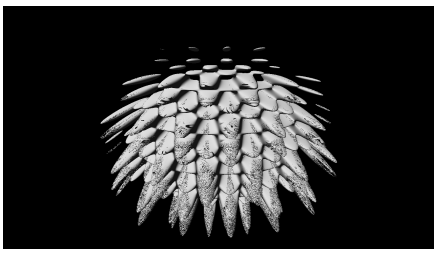


(b) Approximation mit Hesse-Matrix und Gradient

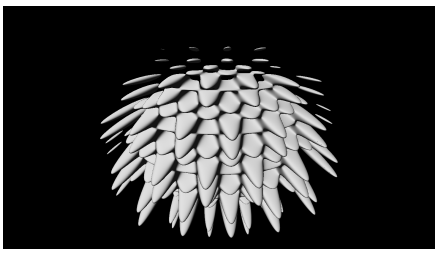
(c) Approximation mit Gradient



(d) Approximation mit Hesse-Matrix und Gradient



(e) Approximation mit Gradient



(f) Approximation mit Hesse-Matrix und Gradient

Abbildung 29: Darstellung verschiedener impliziter Oberflächen mit Annäherung erster und zweiter Ordnung

8.5 Diskussion und Ausblick

Die hier gezeigten Beispiele sollen einen Überblick über einige der Möglichkeiten geben, welche Distanzfeld-Raytracing bieten. Natürlich ist zu beachten, dass die Beispiele nur marginal optimiert wurden. Das erste Beispiel der Cornell Box zeigt, wie verschiedene komplexe Effekte genutzt werden können, besonders die sonst sehr aufwendige indirekte Beleuchtung. Die Ergebnisse zeigen, dass dies mit flüssigen Bildraten möglich ist. Ähnliches gilt für die verschiedenen Szenen in 8.3. Hier zeigt sich zwar, dass auch komplexe Szenen darstellbar sind, aber mit deutlich merkbaren Geschwindigkeitseinbußen. Auch die Berechnung des Fraktals ist bei wei-

tem nicht flüssig in der Darstellung. Beides reagiert allerdings noch interaktiv. Im Vergleich zu polygonbasierten Verfahren ist die Szenenbeschreibung dank der konstruktiven Geometrie jedoch sehr kompakt und leicht zu erstellen. Beispielsweise wurden die Schalen auf den Säulen in Abbildung 28c erstellt, indem eine kleine Kugel von der Schnittmenge einer größeren Kugel und eines Quaders subtrahiert wurde. Eine Richtung für zukünftige Untersuchungen könnte es sein, Verfahren wie die gerade betrachteten für große komplexe Szenen zu betrachten. Eventuell erlauben neuere Shader, wie Compute-Shader, eine bessere Verwaltung der einzelnen Objekte, auch mit Nutzung von Bounding Volumes. Dies ist mit den eingeschränkten Funktionalitäten von WebGL oder OpenGL ES bisher nur schwer möglich. Eventuell sind allerdings auch voxelbasierte Verfahren sinnvoller. Das Beispiel aus 8.2 zeigt, dass diskrete Distanzfelder viel Potential bieten. Da das erzeugte Distanzfeld für statische Daten nur einmal erzeugt werden muss, sind zum Darstellen der Szene lediglich Texturzugriffe nötig, unabhängig von der Anzahl oder Komplexität der Objekte. Das Beispiel zeigt auch, dass selbst mit ineffizienten Shadern bereits sehr flüssige Bilder erzeugt werden können. Trotz der sehr kleinen Auflösung können schon komplexe Objekte, wie die ausgeschnittene Kugel, ansprechend rekonstruiert werden, im Gegensatz zu traditionellem Voxel-Raymarching. Gerade für Effekte wie Schatten, welche nicht die Krankheiten von Shadow Maps haben, könnten diskrete Distanzfelder großes Potential bieten, wie etwa in der Unreal Engine 4 gezeigt [2]. Weitere Untersuchungen könnten sich mit effizienten Strategien zum Kombinieren und Verarbeiten von statischen und dynamischen diskreten Feldern beschäftigen. Das letzte Beispiel zeigt den Nutzen von einer Distanzapproximation zweiter Ordnung für implizite Oberflächen. Mithilfe von entweder analytischen Ableitungen oder Automatischer Differenzierung bedeutet die Berechnung der Hesse-Matrix auch nur einen kleinen zusätzlichen Aufwand, welcher sich aber durch Reduktion von Artefakten auszahlt.

Gegenüber polygonbasierten Verfahren ist natürlich ein großer Nachteil, dass keine optimierte Hardware zur Verfügung steht, was auch nicht leicht möglich ist. Während Dreiecke als diskrete Menge gegeben sind, bildet ein Distanzfeld unendlich viele Punkte ab, lässt sich also nicht nach und nach abarbeiten. Auch sind Schnittpunkttests für einfache Formen oft effizienter, als das stückweise Herantasten mit mehrfachem Aufruf der Distanzfunktion, was auch, wie in Abbildung 18 zu sehen, zu Fehlern führen kann.

Neben der reinen Darstellung sind noch andere Bereiche interessant zur Nutzung von Distanzfeldern. So könnten sie eventuell als unterstützende Struktur für Physiksysteme genutzt werden, da Kollisionen in manchen Fällen auf wenige Distanzabfragen reduziert werden können, beispielsweise bei Kugeln. Über den Gradienten wäre auch direkt eine Kollisionsnormale gegeben.

Ein weiterer Anwendungsfall könnte eine Erweiterung von normaler CSG

sein. In manchen 3D Modellierungswerkzeugen existieren Werkzeuge, die der Bildhauerei angelehnt sind. Formen können durch Hinzufügen oder Entfernen von Masse erstellt werden. Mittels eines Distanzfelds könnten diese Operationen als Vereinigung oder Subtraktion realisiert werden. Werden die Daten diskret repräsentiert, so müssen auch für komplexer werdende Formen keine zusätzlichen Strukturen generiert werden, wie etwa bei polygonbasierten Versionen. Auch wirken die in dieser Arbeit vorgestellten Operationen punktweise auf Distanzfelder. Dadurch entspricht zum Beispiel eine Vereinigung, die auf dem diskreten Gitter berechnet wird, einer solchen, die auf allen Punkten berechnet und dann diskretisiert wurde. Es entstehen also keine zusätzlichen Artefakte durch Unterabtastung. Abbildung 30 zeigt eine sehr simple Variante dieser Idee.

(a) Gestaltete Szene aus Blickwinkel von oben

(b) Gestaltete Szene aus schrägem Blickwinkel

Abbildung 30: Beispiel einer möglichen Verwendung von Distanzfeldern zur bildhauerischen Modellierung

9 Zusammenfassung

In dieser Arbeit wurde ein Überblick über die Grundlagen von Distanzfeldern gegeben. Dabei wurde auf ihre Eigenschaften eingegangen und beispielhaft für einige Arten von Objekten die nötigen Distanzformeln hergeleitet. Insbesondere wurde eine direkte Formel für quadratische Funktionen, sowie quadratische Splines angegeben, welche viele Möglichkeiten zur Modellierung bieten.
Weiterhin wurde eine Approximation zweiter Ordnung für allgemeine implizite Oberflächen gegeben, welche auf der Hesse-Matrix basiert und sich damit für optimierte Hardware, wie etwa GPUs, eignet. Es wurde an einem Beispiel gezeigt, dass diese Approximation in der Lage ist, Artefakte der ersten Ordnung zu verhindern.
Es wurde auf verschiedene Effekte eingegangen, die mithilfe von Distanzfeldern gerade für die Echzeitgrafik günstig erzeugt werden können. Dabei wurde ein möglicher Ansatz für eine stark vereinfachte indirekte Beleuch-

tung vorgestellt, welche lediglich einmal die einzelnen Distanzfunktionen der Objekte, sowie eventuelle Gradientenberechnungen benötigt.
Zuletzt wurde eine Möglichkeit zur Implementierung auf der GPU aufgezeigt, welche passenden Code für beliebige Objekte generiert und damit auch mit älteren OpenGL Versionen oder WebGL funktioniert. Die Testergebnisse legen nahe, dass Distanzfelder sich dank moderner GPUs auch für Echtzeitsysteme nutzen lassen. Besonders diskrete Distanzfelder könnten in Zukunft für das Darstellen selbst oder aber für sekundäre Effekte wie Schatten genutzt werden. Weiterhin sind einfache Varianten eines Distanzfeld Raytracers sehr leicht zu programmieren und erlauben so einen schnellen Einstieg. Auch lässt sich das Verfahren prinzipiell in jede Art von Raytracer einbauen, da das Endergebnis das gleiche ist, wie bei einem Tracer, welcher direkt Schnittpunkte errechnet. Allerdings besteht noch viel Bedarf, die Technik für etwa große und dynamische Szenen nutzbar zu machen. Außerdem basieren Effekte wie weiche Schatten auf exakten Distanzfeldern. Bei approximativen Distanzfunktionen oder Distanzschranken treten Artefakte auf. Trotz einiger Probleme sind Distanzfelder dank ihrer vielseitigen Anwendbarkeit eine vielversprechende Alternative oder Unterstützung für bestehende Rendering-Techniken.

Literatur

[1] C. Green, "Improved alpha-tested magnification for vector textures and special effects," in *ACM SIGGRAPH 2007 Courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281665

[2] N. Tatarchuk, M. Pettineo, D. Wright, S. Aaltonen, and U. Haar, "Advances in real time rendering, part ii," in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2776880.2787702

[3] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[4] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 133–142, Aug. 1986. [Online]. Available: httpdoi.acm.org10.114515886.15901

[5] E. Lafortune, "Mathematical models and monte carlo algorithms for physically based rendering," *Department of Computer Science, Faculty of Engineering, Katholieke Universiteit Leuven*, pp. 20–23, 1996.

[6] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 703–712, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566640

[7] R. Lasser and F. Hofmaier, *Analysis 1 + 2*. Springer Spektrum.

[8] S. G. Krantz and H. R. Parks, "Distance to ck hypersurfaces," *Journal of Differential Equations*, vol. 40, no. 1, pp. 116 – 120, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022039681900139

[9] J. C. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996. [Online]. Available: http://dx.doi.org/10.1007/s003710050084

[10] M. Brokate, N. Henze, F. Hettlich, A. Meister, G. Schranz-Kirlinger, and T. Sonar, *Grundwissen Mathematikstudium*. Springer Spektrum.

[11] T. Arens, R. Busam, F. Hettlich, C. Karpfinger, and H. Stachel, *Grundwissen Mathematikstudium*. Springer Spektrum.

[12] A. Ricci, "A constructive geometry for computer graphics," *The Computer Journal*, vol. 16, no. 2, pp. 157–160, 1973. [Online]. Available: http://comjnl.oxfordjournals.org/content/16/2/157.abstract

[13] I. Quilez. smooth minimum. Besucht am 28/02/16. [Online]. Available: http://iquilezles.org/www/articles/smin/smin.htm

[14] Khronos Group. mod. Besucht am 04/18/16. [Online]. Available: https://www.opengl.org/sdk/docs/man/html/mod.xhtml

[15] I. Quilez. Modeling with distance functions. Besucht am 31/12/15. [Online]. Available: http://iquilezles.org/www/articles/distfunctions/distfunctions.htm

[16] E. W. Weisstein. Torus. Besucht am 04/01/16. [Online]. Available: http://mathworld.wolfram.com/Torus.html

[17] M. W. Jones, "3d distance from a point to a triangle," *Department of Computer Science, University of Wales Swansea Technical Report CSR-5*, 1995.

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992.

[19] C. Loop and J. Blinn, "Resolution independent curve rendering using programmable graphics hardware," in *ACM SIGGRAPH 2005 Papers*, ser. SIGGRAPH '05. New York, NY, USA: ACM, 2005, pp. 1000–1009. [Online]. Available: http://doi.acm.org/10.1145/1186822.1073303

[20] K. Köhler, *Differentialgeometrie und homogene Räume*. Springer Spektrum.

[21] S. Mauch, "A fast algorithm for computing the closest point and distance transform," *Go online to http://www. acm. caltech. edu/seanm/software/cpt/cpt. pdf*, 2000.

[22] M. Jones, J. Baerentzen, and M. Sramek, "3d distance fields: a survey of techniques and applications," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 4, pp. 581–599, July 2006.

[23] G. Taubin, "Distance approximations for rasterizing implicit curves," *ACM Trans. Graph.*, vol. 13, no. 1, pp. 3–42, Jan. 1994. [Online]. Available: http://doi.acm.org/10.1145/174462.174531

[24] O. Forster, *Analysis 2*. Springer Spektrum.

[25] B. Mandelbrot, "How long is the coast of britain? statistical self-similarity and fractional dimension," *Science*, vol. 156, no. 3775, pp. 636–638, 1967. [Online]. Available: http://science.sciencemag.org/content/156/3775/636

[26] K. J. Falconer, *Fraktale Geometrie*. Spektrum Akademischer Verlag.

[27] H.-O. Peitgen, H. Jürgens, and D. Saupe, *Bausteine des Chaos: Fraktale*. Springer-Verlag.

[28] H.-O. Peitgen and D. Saupe, Eds., *The Science of Fractal Images*. New York, NY, USA: Springer-Verlag New York, Inc., 1988.

[29] Y. Dang, L. Kauffman, and D. Sandin, *Hypercomplex Iterations: Distance Estimation and Higher Dimensional Fractals*, ser. Hypercomplex Iterations: Distance Estimation and Higher Dimensional Fractals. World Scientific, 2002, no. v. 1. [Online]. Available: https://books.google.de/books?id=WotDbr_yKe4C

[30] L. Yatziv, A. Bartesaghi, and G. Sapiro, "O(n) implementation of the fast marching algorithm," *Journal of Computational Physics*, vol. 212, no. 2, pp. 393 – 399, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999105003736

[31] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2d euclidean distance transform algorithms: A comparative survey," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 2:1–2:44, Feb. 2008. [Online]. Available: http://doi.acm.org/10.1145/1322432.1322434

[32] A. Meijster, J. B. Roerdink, and W. H. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and its applications to image and signal processing*. Springer, 2000, pp. 331–340.

[33] S. F. F. Gibson, "Using distance maps for accurate surface representation in sampled volumes," in *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, ser. VVS '98. New York, NY, USA: ACM, 1998, pp. 23–30. [Online]. Available: http://doi.acm.org/10.1145/288126.288142

[34] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 269–278. [Online]. Available: http://doi.acm.org/10.1145/15922.15916

[35] D. R. Peachey, "Solid texturing of complex surfaces," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 279–286, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/325165.325246

[36] A. Knoll, Y. Hijazi, C. Hansen, I. Wald, and H. Hagen, "Interactive ray tracing of arbitrary implicits with simd interval arithmetic," in *Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on*, Sept 2007, pp. 11–18.

[37] O. Caprani, L. Hvidegaard, M. Mortensen, and T. Schneider, "Robust and efficient ray intersection of implicit surfaces," *Reliable Computing*, vol. 6, no. 1, pp. 9–21. [Online]. Available: http://dx.doi.org/10.1023/A:1009921806032

[38] D. P. Mitchell, "Robust ray intersection with interval arithmetic," in *Proceedings on Graphics Interface '90*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 1990, pp. 68–74. [Online]. Available: http://dl.acm.org/citation.cfm?id=93267.93276

[39] B. Keinert, H. Schäfer, J. Korndörfer, U. Ganse, and M. Stamminger, "Enhanced Sphere Tracing," in *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, A. Giachetti, Ed. The Eurographics Association, 2014.

[40] J. M. Hasenfratz, M. Lapierre, N. Holzschuch, F. Sillion, and A. GRAVIR/IMAG-INRIA, "A survey of real-time soft shadows algorithms," *Computer Graphics Forum*, vol. 22, no. 4, pp. 753–774, 2003. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2003.00722.x

[41] D. Scherzer, M. Wimmer, and W. Purgathofer, "A survey of real-time hard shadow mapping methods," *Computer Graphics Forum*, vol. 30, no. 1, pp. 169–186, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2010.01841.x

[42] I. Quilez, "Rendering worlds with two triangles with raytracing on the gpu in 4096 bytes," 2008.

[43] À. Méndez-Feliu and M. Sbert, "From obscurances to ambient occlusion: A survey," *The Visual Computer*, vol. 25, no. 2, pp. 181–196, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00371-008-0213-4

[44] M. Mittring, "Finding next gen: Cryengine 2," in *ACM SIGGRAPH 2007 Courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007, pp. 97–121. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281671

[45] A. Evans, "Fast approximations for global illumination on dynamic scenes," in *ACM SIGGRAPH 2006 Courses*, ser. SIGGRAPH '06. New York, NY, USA: ACM, 2006, pp. 153–171. [Online]. Available: http://doi.acm.org/10.1145/1185657.1185834

[46] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," November 2014.

[47] P. Jeremias and I. Quilez, "Shadertoy: Learn to create everything in a fragment shader," in *SIGGRAPH Asia 2014 Courses*, ser. SA '14. New York, NY, USA: ACM, 2014, pp. 18:1–18:15. [Online]. Available: http://doi.acm.org/10.1145/2659467.2659474