universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Über die Verwendung von Dreiecks-Bézier-Patches in Raytracing-Algorithmen

Verfasser

Markus Pumberger

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat)

Studienkennzahl lt. Studienblatt:	A 405
Studienrichtung lt. Studienblatt:	Mathematik
Betreuer:	Ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl

Wien, am 12. Oktober 2010

Inhaltsverzeichnis

Danksagung

Zuallererst möchte ich Herrn Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl für seine hervorragende Betreuung dieser Arbeit danken. Er war immer bemüht, Zeit für Besprechungstreffen zu finden, und zudem waren diese Treffen der wichtigste Impuls zur Entwicklung dieser Arbeit.

Ebenso sei Herrn Univ.-Prof. Dr. Arnold Neumaier gedankt, der im Seminar zu Angewandter Mathematik immer ein offenes Ohr für Fragen zu dieser Arbeit hatte. Besonders danke ich auch den Professoren Harald Rindler und Michael Grosser, die mir am Beginn des Studiums durch ihre Begeisterung die Tür zur höheren Mathematik geöffnet haben.

Meine lieben Freunde möchte ich auch genannt wissen. Mit ihnen verbinde ich wunderbare Erinnerungen an die großartige Studentenzeit in Wien. Und in manch schwieriger Zeit haben sie mir in Gesprächen einen möglichen Weg vorgezeichnet.

Im Besonderen möchte ich meiner Familie danken, und dies im weiter gefassten Sinn. So denke ich an meine wiener Verwandten Jakob, Katharina, Klara, Sebastian, Wolfgang und David - dafür, dass mir Wien von Anfang an Heimat sein konnte und dafür dass sie mir die Tür zu den kulturellen Möglichkeiten der Stadt geöffnet haben.
Und auch meinen Großeltern Marianne und Hans Stumpf sowie Aloisia und Franz Pumberger gegenüber empfinde ich Dankbarkeit. Jede/r von ihnen hat mich geprägt - auf seine/ihre Weise.

Doch vor allem möchte ich meinen Eltern Renate und Franz Pumberger danken. Natürlich für ihre großzügige finanzielle und auch moralische Unterstützung, ohne die der Studienabschluss nicht möglich gewesen wäre. Doch vielmehr noch für die Gelegenheit im Kreise meiner Geschwister in einem warmherzigen Elternhaus aufzuwachsen und eine Erziehnung in christlich-sozialer Tradition genossen zu haben.

Ihnen allen ist daher diese Arbeit gewidmet.

Abstract

Unter Raytracing fasst man eine große Klasse von Algorithmen aus dem Bereich der Computergrafik zusammen. Im ursprünglichen Sinn verstand man darunter Methoden zur Verdeckungsberechnung und somit zur Bestimmung von Schattenflächen. Diese Algorithmen wurden jedoch mit verbesserter Rechnerleistung um die Modellierung vieler Eigenschaften, vor allem aus dem Bereich der Optik, erweitert. So liegt zumindest in den letzten drei Jahrzehnten der Schwerpunkt der Entwicklungen auf dem Versuch, Eigenschaften des Lichts im weitesten Sinn möglichst realistisch nachzubilden.
Damit wurden Raytracing-Algorithmen zu einem mächtigen Instrument der Simulationstechnik und sind daher auch von wirtschaftlicher Bedeutung. Dementsprechend groß ist die Bandbreite an aufwendigen Weiterentwicklungen der ursprünglichen Idee.
Von zunehmendem Interesse ist ebenso die Modellierung von komplexeren Objekten, welche betrachtet werden, um die Wirklichkeit realitätsgetreu nachzubilden. Hierbei sind jedoch Schnittpunktberechnungen mit manchen Primitiven, die zur Approximation von Objekten verwendet werden, zum Teil nur iterativ bestimmbar.

Das Ziel des anwendungsbezogenen Teil dieser Arbeit besteht nun in der Modellierung folgender Situation:
In einem abgeschlossenen Raum befindet sich ein aus Dreiecks-Bézier-Patches aufgebauter Spiegel, auf den von einer Lampe ausgesendetes Licht trifft. Somit ist der nach der Reflexion entstehende Lichtkegel zu berechnen. Die zu diesem Zwecke in MATLAB implementierten Algorithmen sind in Kapitel 5 zu finden.

Zu Beginn werden jedoch grundlegende Elemente von Raytracing-Algorithmen vorgestellt (Kapitel 1). Danach wird die für die Implementation wichtige Frage der Reflexion behandelt und kurz werden Ergebnisse der Optik (perfekte Transmission, totale innere Reflexion) präsentiert. Skizziert werden zudem Methoden der Beschleunigung von Raytracing Algorithmen, da diese für die Anwendbarkeit von großer Wichtigkeit sind.

Von zentraler Bedeutung sind die in Kapitel 2 definierten Dreiecks-Bézier-Patches, da sie

die Objekte der Betrachtung im Algorithmus sein werden. Sie werden aus den mit ihnen eng verwandten Tensorprodukt-Patches hergeleitet, von denen sie sich durch entscheidende Vorteile abheben. Des Weiteren werden zahlreiche Eigenschaften von Bézier-Patches besprochen und einige Algorithmen im Zusammenhang mit diesen Flächen vorgestellt. Aus diesen Eigenschaften wird sich die Verwendung von Bézier-Patches im Algorithmus erklären.
Leider ist die Schnittpunktberechnung mit Dreiecks-Bézier-Patches recht aufwendig, zumal dabei keine expliziten Lösungen existieren. Daher sind hier Iterationsmethoden das Vorgehen der Wahl, allen voran das Newton-Verfahren.

Dieses wird mit einigen Varianten wie dem modifizierten Newton-Verfahren und Verfahren mit Liniensuche in Kapitel 3 behandelt. Zuerst werden jedoch das allgemeine Newton-Verfahren sowie Iterationsverfahren im Generellen eingeführt. Neben dem modifizierten Newton-Verfahren werden in Kapitel 3 zudem noch das vereinfachte Newton-Verfahren und Konvergenzeigenschaften des modifizierten Newton-Verfahren besprochen.

Das Hauptresultat dieser Diplomarbeit findet sich im Kapitel 4. Nachdem Grundlagen der Strahlenoptik besprochen werden und das Brechungs- sowie Reflexionsgesetz aus dem Fermatschen Prinzip hergeleitet werden, folgt eine mathematische Beschreibung der implementierten Algorithmen. Ein durchgerechnetes Beispiel ist in diesem Teil ebenso zu finden wie die Skizze eines alternativen Lösungsansatzes, der als Vorteil das Umgehen der zeitlich aufwendigen Schnittpunktberechnung hat.

In Kapitel 5 sind schließlich die Implementationen, realisiert auf MATLAB R2009a, enthalten. Zudem sind sie dort mit kurzen Erklärungen versehen und beschließen somit die Arbeit.

1 Raytracing-Algorithmen

Unter Raytracing *(dt. Strahlen(ver)folgung, gelegentlich auch: Ray Tracing)* versteht man eine Klasse von Algorithmen, die in einem ersten Schritt, zur Ermittlung der Sichtbarkeit von zwei- bzw. dreidimensionalen Objekten in einer Ebene bzw. einem Raum von einem bestimmten Punkt aus dienen. Den einfachsten Fall kann man sich als eine Art Lochkamera vorstellen: Man blickt durch eine Kamera (hier also ein Loch) in einen Raum auf beleuchtete Elemente und bildet das Bild auf einer Ebene ab.
Natürlich kann man dieses Grundprinzip stark erweitern, zum einen indem man immer komplexere Figuren betrachtet, zum anderen durch Miteinbeziehung von physikalischen Erkenntnissen: Reflexionen, Spiegelungen, Brechungen, weiche Schatten, dem sogannten Antialising, mehrere Beleuchtungsquellen, Fresnelsche Formeln und vor allem von Eigenschaften des Lichts. Es ist naheliegend, dass die Rechendauer mit dieser Komplexität stark ansteigt.
Hier sollen jedoch nur grundlegende Modelle präsentiert werden: Zu Beginn die Schnittpunktberechnung einiger Quadriken, danach das auch unsere Implementation betreffende rekursive Raytracing. Abschließend werden noch Methoden zur Beschleunigung von Raytracing-Algorithmen skizziert.

1.1 Grundprinzip

Wir starten mit der Einführung einiger grundlegender Begriffe:

Zu Beginn ist es notwendig, die Art sowie die Position der einzelnen PRIMITIVEN festzulegen. Unter Primitiven versteht man im Raytracing geometrische Objekte wie Polygone und Kugeln, welche sich im beobachteten Raum befinden. In weiterer Folge werden auch Dreiecks-Bézier-Patches 2.2.3.1 als Primitiven auftreten.
Zusätzlich wird beim Raytracing noch die Position eines AUGPUNKTES angegeben, von dem aus die Primitiven betrachtet werden. Zwischen Augpunkt und den Primitiven stelle man sich ein Raster vor, die BILDEBENE, durch welche die Strahlen geleitet werden. Trifft

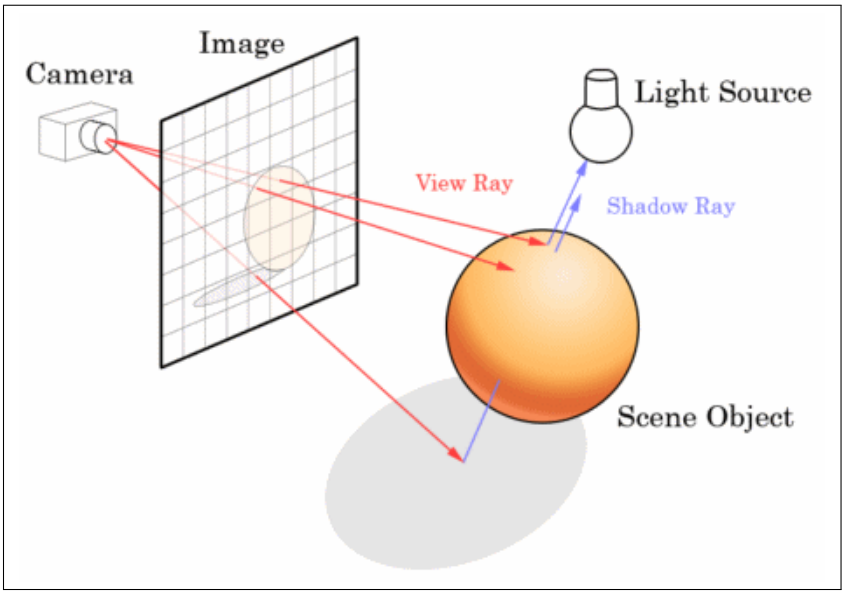

Abbildung 1.1: Grundprinzip des Backward Raytracing (FRIEDRICH A.LOHMÜLLER)

nun ein Strahl auf ein Objekt, so wird der entsprechende Pixel auf der Bildebene eingefärbt.

Erstmals beschrieb A. Appel [1] im Jahre 1968 das aus der Idee der Lochkamera entwickelte FORWARD RAYTRACING. Dabei bildet man von einer Lichtquelle im Raum ausgehend Strahlen und testet sie auf Schnitte, Reflexionen und Brechungen an Objekten im Raum. Selbstverständlich treffen nicht alle Strahlen die Bildebene vor dem Augpunkt. Bezieht man zudem noch physikalische Eigenschaften der Photonen sowie der Oberflächen der Primitiven mit ein, so ist klar, dass der Aufwand eines solchen Algorithmus enorm groß sein wird.
Eine Verringerung dieses Aufwandes erreicht man durch Anwendung des BACKWARD RAYTRACING. Dem Namen entsprechend wird dabei die Richtung umgedreht - man folgt also den Strahlen vom Augpunkt zur Lichtquelle und reduziert so die Anzahl der Strahlen, welche betrachtet werden müssen, ganz entscheidend. Dem Backward Raytracing entspricht Abbildung 1.1.

Den ebenso wichtigen Reflexionsberechnungen von Strahlen werden wir uns etwas später zuwenden. Zuerst beschäftigen wir uns mit der Berechnung des Schnittes von Strahl und Primitiv, dem zentralen Teil eines Raytracing-Algorithmus.

1.2 Schnittberechnung

Im Folgenden werden die Schnitte mit einigen algebraischen Oberflächen vom Grad $n < 5$ beschrieben, da hier explizite analytische Lösungen existieren. Verwendet man jedoch Bézier-Patches als Primitiven, so muss der Schnittpunkt iterativ näherungsweise berechnet werden (siehe Kapitel 3).

Definition 1.2.0.1. *Eine* ALGEBRAISCHE FLÄCHE *ist eine implizite Oberfläche, die als polynomiale Gleichung geschrieben werden kann. Die allgemeine algebraische Oberfläche ist gegeben durch*

$$F(x, y, z) = \sum_{i+j+k \leq d} a_{ijk} x^i y^j z^k.$$

Läuft dabei der Index i bis l, der Index j bis m und k bis n, so ist der Grad dieser Oberflächen $d = l + m + n$. Zudem ist die Anzahl von Termen in einer Oberfläche, in der alle Termen von Totalgrad d auftauchen, gleich $(n + 1)(n + 2)(n + 3)/6$.
Wir betrachten nun den Schnitt einiger Quadriken (oder im Dreidimensionalen auch Flächen zweiter Ordnung) mit einer Geraden der Form

$$g : g(\vec{x}) = \vec{x}_1 t + \vec{x}_0 \tag{1.1}$$

oder, falls man jede Koordinate schreibt, mit

$$x = x_1 t + x_0, \quad y = y_1 t + y_0 \quad z = z_1 t + z_0.$$

1. SPHÄRE
   Die Gleichung einer Einheitssphäre im kanonischen Koordinatensystem ist

   $$x^2 + y^2 + z^2 - 1 = 0.$$

   Substitution von 1.1 ergibt

   $$t^2(x_1^2 + y_1^2 + z_1^2) + 2t(x_0 x_1 + y_0 y_1 + z_0 z_1) + (x_0^2 + y_0^2 + z_0^2) - 1 = 0.$$

   Wichtig ist hierbei, dass der Richtungsvektor der Geraden normiert ist, d.h. $x_1^2 + y_1^2 + z_1^2 = 1$.

2. ZYLINDER

Die kanonische Gleichung eines unendlichen Zylinders lautet

$$x^2 + y^2 - 1 = 0.$$

Wieder substituiert man 1.1 und erhält

$$t^2(x_1^2 + y_1^2) + 2t(x_0x_1 + y_0y_1) + (x_0^2 + y_0^2) - 1 = 0.$$

3. KEGEL
   Aus der kanonischen Gleichung des unendlichen Kegels,

$$x^2 + y^2 - z^2 = 0,$$

   folgt durch Substitution von 1.1

$$t^2(x_1^2 + y_1^2 - z_1^2) + 2t(x_0x_1 + y_0y_1 - z_0z_1) + (x_0^2 + y_0^2 - z_0^2) = 0.$$

4. PARABOLOID
   Die kanonische Gleichung des Paraboloids ist

$$x^2 + y^2 + z = 0.$$

   Wieder erhalten wir

$$t^2(x_1^2 + y_1^2) + 2t(x_0x_1 + y_0y_1 + z_1) + (x_0^2 + y_0^2 + z_0^2) = 0.$$

5. HYPERBOLOID
   Hier unterscheiden wir zwischen dem zweischaligen Hyperboloid

$$x^2 + y^2 - z^2 + 1 = 0$$

   und dem einschaligen

$$x^2 + y^2 - z^2 - 1 = 0.$$

   Daraus ergeben sich die folgenden Gleichungen in t:

$$t^2(x_1^2 + y_1^2 - z_1^2) + 2t(x_0x_1 + y_0y_1 - z_0z_1) + (x_0^2 + y_0^2 - z_0^2) \pm 1 = 0.$$

Auch für Gleichungen dritten und vierten Grades gibt es geschlossene Formen [2]. Häufig benötigt man jedoch, wie bereits angedeutet, Iterationsverfahren, auf die in Kapitel 3 näher eingegangen wird.
Angenommen, wir haben einen Schnittpunkt berechnet, wie gehen wir dann bei der Durchführung des Raytracings weiter vor?
Damit beschäftigt sich das Rekursive Raytracing.

1.3 Rekursives Raytracing

Rekursives Raytracing ermöglicht das Miteinbeziehen von lichtdurchlässigen sowie spiegelnden Objekten. Grundlegend unterscheidet man vier Mechanismen (ggf. auch Arten genannt [8]) des Lichttransportes:

1. PERFEKTE (SPIEGELNDE) REFLEXION

   Sei $\vec{n}$ der Normalvektor der Oberfläche am Schnittpunkt, $\vec{v}$ der Richtungsvektor des Strahls und θ_i der Winkel zwischen diesen beiden Vektoren. Dann gilt mit $\vec{u}$ als Richtungsvektor des reflektierten Strahls und θ_j der Winkel zwischen $\vec{n}$ und $\vec{u}$

   $$\vec{u} = \alpha\vec{v} + \beta\vec{n}, \qquad \theta_i = \theta_j. \tag{1.2}$$

   Die erste Eigenschaft inkludiert, dass alle drei Vektoren in der selben Ebene liegen (man nennt diese Ebene auch gelegentlich REFLEXIONSEBENE). Rechnerisch wollen wir die Reflexion als Rotation um π realisieren mit dem normierten Normalvektor beim Schnittpunkt als Drehachse.
   Der Normalvektor errechnet sich über das Kreuzprodukt, zudem wird er noch normiert. Habe man als Fläche

   $$F : U \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad (x, y) \mapsto F(x, y),$$

   so spannen die partiellen Ableitungen

   $$F_x(x, y) := \frac{\partial F}{\partial x}(x, y) \text{ und } F_y(x, y) := \frac{\partial F}{\partial y}(x, y)$$

   die Tangentialebene bei $F(x, y)$ auf. Der normierte Normalvektor ist dann definiert durch

   $$\vec{n} := \frac{F_x(x, y) \times F_y(x, y)}{|F_x(x, y) \times F_y(x, y)|}. \tag{1.3}$$

Der Vektor $\vec{n}$ der Form $(n_1, n_2, n_3)^T$ fungiert dann als Drehachse. Die Rotation erfolgt dabei durch Multiplikation von $\vec{v}$ mit folgender Drehmatrix ROT:

$$\text{ROT} := \begin{pmatrix} \cos\alpha + n_1^2(1-\cos\alpha) & n_1n_2(1-\cos\alpha) - n_3\sin\alpha & n_1n_3(1-\cos\alpha) + n_2\sin\alpha \\ n_1n_2(1-\cos\alpha) + n_3\sin\alpha & \cos\alpha + n_2^2(1-\cos\alpha) & n_2n_3(1-\cos\alpha) - n_1\sin\alpha \\ n_1n_3(1-\cos\alpha) - n_2\sin\alpha & n_2n_3(1-\cos\alpha) + n_1\sin\alpha & \cos\alpha + n_3^2(1-\cos\alpha) \end{pmatrix}.$$

Bei der Reflexion setzt man in ROT den Winkel $\alpha = \pi$.

Eine zweite Möglichkeit zur Errechnung des Vektors $\vec{n}$ ist die Folgende:
Es gilt

$$\cos(\theta_i) = -\vec{v} \cdot \vec{n}$$

sowie

$$\cos(\theta_j) = \vec{n} \cdot \vec{u}.$$

Damit ersetzen wir die zweite Eigenschaft in 1.2 durch

$$\begin{aligned} \cos(\theta_i) &= \cos(\theta_j) \\ -\vec{v} \cdot \vec{n} &= \vec{n} \cdot \vec{u} \\ &= \vec{n} \cdot (\alpha\vec{v} + \beta\vec{n}) \\ &= \alpha(\vec{n} \cdot \vec{v}) + \beta(\vec{n} \cdot \vec{n}) \\ &= \alpha(\vec{n} \cdot \vec{v}) + \beta. \end{aligned}$$

Der letzte Schritt gilt, da für normiertes $\vec{n}$ gilt, dass $\vec{n} \cdot \vec{n} = 1$ ist. Wir setzen $\alpha = 1$ und damit folgt dann

$$\begin{aligned} \beta &= -2(\vec{n} \cdot \vec{v}) \\ \vec{u} &= \vec{v} - 2(\vec{n} \cdot \vec{v})\vec{n}. \end{aligned}$$

Der Fall der perfekten spiegelnden Reflexion wird jener sein, auf den ich mich bei der Implementation beschränken werde. Die restlichen drei Arten des Lichttransportes seien dennoch kurz angeführt.

2. PERFEKT DIFFUSE REFLEXION

Das Reflexionsgesetz gilt nur für glatte, spieglende Oberflächen. Ist die Oberfläche jedoch rau, erfolgt Diffusion. Eine erste (einfache) Approximation dieser Situation

wird durch das LAMBERTSCHE GESETZ [1] erreicht. Sei *L* die Leuchtdichte, *A* eine Fläche und α der Einfallwinkel, dann gilt

$$I = A \cdot \cos(\alpha) \cdot L, \tag{1.4}$$

wobei *I* die Lichtstärke bezeichnet. Anschaulich bedeutet ein flacher Einfallswinkel eine geringere Lichtstärke. Lamberts Gesetz gibt eine gute Approximation für matte, diffus reflektierende Flächen (ebensolche Flächen werden auch Lambertsche Flächen genannt).
Gerade in dem Bereich der Beleuchtungsmodelle gab es in den letzten Jahren viele neue Entwicklungen, welche sich nun aufgrund leistungsstärkerer Computer berechnen lassen [2].

3. PERFEKTE TRANSMISSION

Transmission ist ein Maß für die Durchlässigkeit eines Mediums für Wellen. Bei lichtdurchlässigen Primitven kommt das BRECHNUNGSGESETZ VON SNELLIUS [2] zur Anwendung.
Bezeichne dazu wieder θ_1 den Aufprallwinkel sowie θ_2 den Winkel nach der Brechung, dann gilt

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \text{const.} \tag{1.5}$$

Die Konstante ist genau gleich dem Verhältnis der Lichtgeschwindigkeiten in den beiden unterschiedlichen Medien: Sei η_{21} der Brechungsindex (auch Brechzahl genannt) des zweiten Mediums bezüglich des ersten, η_1 der Brechungsindex des ersten Mediums bezüglich des Vakuums sowie η_2 der Brechungsindex des zweiten Mediums bezüglich des Vakuums dann gilt also

$$\eta_{21} = \frac{\eta_2}{\eta_1}. \tag{1.6}$$

Dieser Index hängt einerseits vom Medium, andererseits von der Wellenlänge des eintreffenden Lichts ab. Er ist also der Quotient aus Lichtgeschwindigkeit und Pha-

[1] Johann Heinrich Lambert (* 26.8.1728 in Mulhouse; † 25.9.1777 in Berlin), französischer Mathematiker, Physiker und Philosoph. Beiträge zur Akustik (da selber von Geburt an schwerhörig) und Optik. Mitbegründer der philosophischen Disziplin des Rationalismus und damit Wegbereiter Kants. Bewies 1761 die Irrationalität von π mittels Kettenbrüchen. Heute ist im angloamerikanischen Raum die Einheit der Leuchtdichte nach ihm benannt.

[2] Willebrord van Roijen Snell, genannt Snellius (* 13.6.1580 in Leiden; † 30.10.1626 ebenda). Niederländischer Mathematiker, Astronom und Jurist. Ab 1613 Professor für Mathematik in Leiden. Beschäftigte sich neben der Optik auch u.a. mit der Berechnung der Kreiszahl π.

sengeschwindigkeit im Medium. Aus diesem Grund spalten Prismen Licht in das Lichtspektrum auf, was schon Newton in seinem Werk über Optik feststellte. Typische Werte für den Brechungsindex η_{21} sind z.B. 1,0003 für Luft bei 20°C und auf Meereshöhe oder 1,33 für Wasser.
Wenn dann $\vec{t}$ der Richtungsvektor nach der Brechung ist, $\vec{u}$ und $\vec{v}$ wie zuvor, dann zeigt Glassner [[8], Kapitel 2.6], dass gilt

$$\vec{t} = \eta_{12}\vec{v} + (\eta_{12}\cos(\theta_1) - \sqrt{\Big(1 + \eta_{12}^2(\cos(\theta_1)^2 - 1)\Big)}\vec{n}. \tag{1.7}$$

4. TOTALE INNERE REFLEXION

   Damit wird ein Phänomen beschrieben, welches im Inneren eines lichtdurchlässigen Körpers stattfindet. Es tritt auf beim Übertritt von Licht vom dichteren in ein weniger dichtes Medium. Dabei erfolgt ab einem kritischen Winkel totale Reflexion ins Innere des Körpers.
   Beim Übertritt von Glas auf Luft liegt dieser Winkel zum Beispiel bei 41,8°.

Damit haben wir die für uns nötigen Teile eines Ray Tracing Algorithmus kennen gelernt. Natürlich lassen sich diese Algorithmen stark verfeinern, vor allem wenn man Eigenschaften des Lichtes miteinbezieht.
So haben wir nur perfekte Spiegelung bzw. Transmission betrachtet. Bezieht man den weiten Bereich dazwischen mitein, so erreicht man das Feld des SHADING (*eng.: shade - Farbton, Abstufung*). Dabei wird die Oberfläche in Abhängigkeit des Einfallswinkel eingefärbt [1].
Kurz beprochen werden soll noch die Frage der Beschleunigung von Ray Tracing Algorithmen.

1.4 Beschleunigungsmethoden

Um das Ray Tracing sinnvoll gestalten zu können, ist die größte Herausforderung die Reduzierung der Rechenzeit. Um dies zu erreichen, gibt es zahlreiche Möglichkeiten. In [[8], Kapitel 6] unterteilen James Arvo und David Kirk Beschleunigungsalgorithmen in folgende drei Gruppen: Algorithmen mit

- schnellerer Berechnung des Schnittes mit dem Primitiv,

- einer geringeren Anzahl an Strahlen,
- einer Verallgemeinerung des Strahlenbegriffes.

Grundsätzlich ist es nicht erforderlich jedes Primitiv gegen jeden Strahl zu testen, da ebenjene Schnittpunkttests die grösste Laufzeit beim Raytracing beanspruchen, zwischen 75% und gar 95% bei komplexen Modellen [20].
In unserem Fall der Dreiecks-Bézier-Patches werden wir uns Eigenschaft 3 in Proposition 2.2.3.2 zugute machen und nur jene Geraden testen, welche die konvexe Hülle des Patches schneiden. Dies entspricht dem Konzept der BOUNDING BOXES, bei dem Umhüllende für Primitive konstruiert werden und in der Folge auf Schnitte getestet.
In den letzten Jahren haben sich jedoch SUBDIVISIONSMETHODEN unter der Verwendung von kd-Bäumen als die beliebtesten etabliert.

Definition 1.4.0.2. *Sei R ein Teilraum des $\mathbb{R}^n$. Dann ist ein* KD-BAUM *über R ein binärer Baum, der R rekursiv untereilt. Die Konstruktion eines kd-Baums erfordert $O(n(k+\log n))$ Rechenoperationen.*

Nun beschreiben Wald und Havran in [19] ihr Konzept der SURFACE AREA HEURISTIC (SAH). Dabei wird die Geometrie der beim Aufteilen entstandenen sogenannten Voxel berücksichtigt, um möglichst große leere Voxel zu erhalten. Nimmt man nun noch folgendes an:

- Die Strahlen sind gleichmäßig verteilt.
- Der Aufwand für den Schritt der Teilung der Voxel und den Dreiecksschnitt sind bekannt. Man bezeichnet sie mit $\mathcal{K}_T$ bzw. $\mathcal{K}_I$.
- Der Aufwand des Schnittes von n Dreiecken sei $n \cdot \mathcal{K}_I$, d.h. linear in der Anzahl der Dreiecke.

Nun betrachten wir einen Strahl, von dem wir wissen, dass er den Voxel V trifft. Die Wahrscheinlichkeit dass er einen Teilvoxel $V_{sub} \subset V$ trifft ist dann

$$\mathcal{P}_{[V_{sub}|V]} = \frac{\mathcal{A}(V_{sub})}{\mathcal{A}(V)}, \tag{1.8}$$

wobei $\mathcal{A}(V)$ die Oberfläche von V bezeichnet. Wir sind am Rechenaufwand interessiert, welcher sich für einen vollständigen kd-Baum wiefolgt darstellt:

$$\mathcal{C}(\mathcal{T}) = \sum_{n \in Knoten} \frac{\mathcal{A}(V_n)}{\mathcal{A}(V_S)} \mathcal{K}_T + \sum_{l \in Flaechen} \frac{\mathcal{A}(V_l)}{\mathcal{A}(V_S)} \mathcal{K}_I, \tag{1.9}$$

wobei V_S die Box beschreibt, in der die ganze Szene enthalten ist. Der beste kd-Baum ist nun jener welcher diese Gleichung minimiert.
Das Erzeugen eines beliebigen kd-Baums basierend auf SAH benötigt $O(N^2)$ Rechenoperationen. Wald/Havran zeigen einen ebenso SAH-basierenden Algorithmus mit Aufwand von $O(N \log N)$ Rechenoperationen[19].

2 Bezier-Patches

Unter Bézier-Patches versteht man eine spezielle Art von parametrisierten Flächen des $\mathbb{R}^n$, benannt nach PIERRE BÉZIER [1], der diese für den französischen Autohersteller Renault entwickelte Methode 1961 vorstellte. Schon 1959 fand PAUL DE FAGET DE CASTELJAU [2] denselben Zugang für seinen Arbeitgeber Citroen, durfte seine Resultate aber aus Wettbewerbsgründen nicht veröffentlichen. Als Anerkennung ist jedoch der Evaluationsalgorithmus 2.2.4.1 nach ihm benannt.
Im Folgenden werden Bézier-Patches, ihre wichtigen Eigenschaften sowie einige Algorithmen vorgestellt. Daraus wird die Bedeutung dieser Patches für die Computergrafik im Allgemeinen und Ray Tracing Algorithmen im Speziellen hervorgehen.
Im ersten Teil des Kapitels wenden wir uns einigen im weiteren wichtigen Definitionen aus dem Bereich der Analysis zu. Ähnliche Resultate wie jene, welche für Flächen im $\mathbb{R}^n$ präsentiert werden, existieren auch für Kurven. Details dazu finden sich in jedem umfangreichen Analysisbuch [z.B. in [10]].
Daraus sollen Tensorprodukt-Patches entwickelt sowie die dafür verwendeten B-Splines kurz eingeführt werden. Um manche Nachteile der Tensorprodukt-Patches zu beheben, werden wir anschließend Dreiecks-Bézier-Patches einführen, welche im MATLAB-Algorithmus Verwendung finden werden. Abschließend werden einige zentrale Algorithmen und Eigenschaften dieser Dreiecks-Bézier-Patches definiert.

[1] Pierre Étienne Bézier (* 1.September 1910 in Paris; † 25.November 1999 ebenda), französischer Ingenieur. Abschluss in Maschinenbau an der École Nationale Supérieure d'Arts et Métiers 1930, in Elektrotechnik an der École Supérieure d'Electricité 1931 sowie Doktor der Mathematik an der Université de Paris 1977; arbeitete 1933-1975 für Renault, wo er sich mit der Entwicklung von CAD-Programmen beschäftigte. Bézier unterrichtete u.a. von 1968 - 1979 am Conservatoire National des Arts et Métiers und erhielt 1985 den Steven A. Coons-Award der Association for Computing Machinery.

[2] Paul de Faget de Casteljau (* 1930 in Bésancon), französischer Physiker und Mathematiker bei Citroen. Entwickelte 1959 den Algorithmus zur Berechnung von Bézier-Kurven. Autor des Buches *Mathématiques et CAO, Vol 2: Formes et pôles*.

2.1 Tensorprodukt-Patches

Vor der Definition der Tensorprodukt-Patches seien einige dafür zentrale Ergebnisse der Analysis angeführt.

2.1.1 Grundlegende Definitionen

Wir werden zuerst die Begriffe „Flächenstück“ und „Flächenparametrisierung“ definieren. Zwischen ihnen besteht ein ähnlicher Zusammenhang wie zwischen „Weg“ und „Bogen“.

Definition 2.1.1.1. *Die nichtleere beschränkte Menge $B \subseteq \mathbb{R}^p$ heißt* JORDAN-MESSBAR, *wenn ihre charakteristsiche Funktion χ_B (in anderen Worten: die konstante Funktion 1 für ein $x \in B$) auf B Riemann-integrierbar ist.*

Definition 2.1.1.2. *Sei B eine nichtleere, kompakte und Jordan-messbare Teilmenge des $\mathbb{R}^p$. Unter einer p-dimensionalen* FLÄCHENPARAMETRISIERUNG Φ *mit dem Parameterbereich B versteht man die injektive Einschränkung $\Phi|_B$ einer C^1-Abbildung $\Phi : M \longrightarrow \mathbb{R}^n$ auf B. Dabei ist M eine offene Teilmenge des $\mathbb{R}^p$, die B enthält.*
Die Bildmenge $F := \Phi(B)$ wird nun ein p-dimensionales FLÄCHENSTÜCK *genannt. Dann bezeichnet man*

$$r = \Phi(u), \qquad u \in B$$

als PARAMETERDARSTELLUNG *von F mit dem Parameterbereich B.*

F muss dabei keineswegs flächenhaft aussehen, es kann durchaus auf einen Bogen oder sogar Punkt zusammenschrumpfen. Zweites tritt ein, falls Φ konstant ist. Durch entsprechende Voraussetzungen an Φ sind solche Degenerationen ausschließbar.
Wir untersuchen nun den Flächeninhalt der parametrisierten Fläche. Seien zuerst $p = 2$ sowie $n = 3$, so errechnet sich der Inhalt von Φ durch

$$I(\Phi) = \int_B |\nu(u_1, u_2)| \mathrm{d}(u_1, u_2), \tag{2.1}$$

wobei $\nu(u_1, u_2)$ der durch das Kreuzprodukt errechnete Normalenvektor von Φ am Flächenpunkt $\Phi(u_1, u_2)$ ist:

$$\nu(u_1, u_2) = \frac{\partial \Phi}{\partial u_1}(u_1, u_2) \times \frac{\partial \Phi}{\partial u_2}(u_1, u_2).$$

Im Allgemeinen Fall gilt folgende

Definition 2.1.1.3. *Der* INHALT EINER PARAMETRISIERTEN FLÄCHE Φ *ist gegeben durch das Integral*

$$I(\Phi) = \int_B \mathrm{vol}(P_\Phi(u))\mathrm{d}u. \tag{2.2}$$

Mit $\mathrm{vol}(P_\Phi(u))$ *sei das p-dimensionale Volumen des Parallelepipeds* $P_\Phi(u)$ *bezeichnet, welches von den p Vektoren*

$$\frac{\partial\Phi(u_1,...,u_p)}{\partial u_i}, \qquad i = 1,...,p$$

aufgespannt wird.

Von Bedeutung ist hier die Frage, ob der Inhalt $I(F) := I(\Phi)$ eines Flächenstückes $F = \Phi(B)$ unabhängig von einer speziellen Parameterdarstellung von F mit dem Parameterbereich B ist. Dazu betrachten wir folgende Definition:

Definition 2.1.1.4. *Seien* M *und* B *wie in 2.1.1.2. Sei dann* B' *eine weitere Jordan-meßbare Teilmenge und* M' *ein Gebiet des* $\mathbb{R}^p$*, das* B' *enthält. Dann heißt eine bijektive* C^1*-Abbildung* $g : M' \to M$ PARAMETERTRANSFORMATION *falls* g *ebenso bijektiv von* B' *auf* B *ist und die Funktionalmatrix*

$$J_g = \left(\frac{\partial g(u)}{\partial u}\right)$$

in jedem Punkt $u \in M'$ *regulär ist, d.h.* $\det J_g \neq 0$*.*
Ist $\det J_g$ *sogar überall positiv, so bezeichnet man die Parametertransformation* g *zusätzlich als* ORIENTIERUNGSERHALTEND*.*

Betrachte nun die eben definierte Parametertransformation von B' auf B, $g : M' \to M$ und eine Flächenparametrisierung Φ, deren Parameterbereich B sei. Dann ist die Verknüpfung $\Psi := \Phi \circ g$ eine Flächenparametrisierung mit Parameterbereich B', welche dasselbe Flächenstück F wie Φ definiert. Aus der Kettenregel folgt dann

$$\mathrm{vol}(P_\Psi(u)) = \mathrm{vol}(P_\phi(g(u))) \cdot |\det(J_g)|,$$

und aus der Transformationsregel für Mehrfachintegrale

$$I(\Psi) = \int_{B'} \mathrm{vol}(P_\Psi(u))\mathrm{d}u = \int_B \mathrm{vol}(P_\Phi(v))\mathrm{d}v = I(\Phi). \tag{2.3}$$

Wir sehen also, dass der Inhalt einer parametrisierten Fläche invariant unter Parameterwechseln ist. Aus diesem Grund ist es zulässig vom FLÄCHENINHALT $I(F) := I(\Phi)$

eines Flächenstücks $F = \Phi(B)$ zu sprechen. Daher besteht auch die Möglichkeit eine p-dimensionale Fläche als p-dimensionales Flächenstück zusammen mit einer Äquivalenzklasse $[\Phi]$ von Flächenparametrisierungen zu bezeichnen.
Als Äquivalenzrelation wählt man $\Psi \sim \Phi$ genau dann, wenn es eine orientierungserhaltende Umparametrisierung g gibt mit $\Phi = \Psi \circ g$.
Entscheidend ist nun noch die Wahl der Parameterbereiche. Zuerst werden wir hier (für $p = 2$) achsenparallele Rechtecke wählen, was uns zu Tensorprodukt-Patches führt. Danach gehen wir zu Simplizes über, da man mit Triangulierungen viel komplexere Flächen betrachten kann. Darauf werden dann schließlich die für den Raytracing-Algorithmus verwendeten Dreiecks-Bézier-Patches entwickelt.

2.1.2 Tensorprodukt-Patches

Bevor wir zur Definition der Tensorprodukt-Patches kommen, soll noch der Begriff der Splines eingführt werden, da wir die Tensorprodukt-Patches daraus aufbauen werden.
Ursprünglich stammt das Wort „Splines“ aus dem Schiffsbau, wo man darunter die Bretter der Schiffsbeplankung in Längsrichtung meinte, welche sich, ähnlich dem kubischen Spline, um einzelne Querverstrebungen biegen.
Mathematisch ist die Grundidee dabei, Funktionen aus kürzeren Stücken zusammenzusetzen und durch bestimmte Punkte zu leiten, sodass die so entstehenden Funktionen k-mal stetig differenzierbar sind. Diese Funktionsstücke, aus denen man Splines aufbaut, bezeichnet man als B-Splines.

Dazu haben wir eine Folge von Punkten $t_0, ..., t_m$ in $\mathbb{R}$ mit $t_i \leq t_{i+1} \forall i$, genannt KNOTEN. Ebenso setzt man für $1 \leq j \leq m + 1 - i$

$$\omega_{i,j}(x) = \begin{cases} \frac{x-t_i}{t_{i+j}-t_i}, & \text{falls } t_i < t_{i+j} \\ 0 & \text{sonst.} \end{cases}$$

und bezeichnet $\omega_{i,j}(x)$ als GEWICHTSFUNKTIONEN. Mit diesen Notationen folgt nun

Definition 2.1.2.1. *Sei* $t = (t_0, ..., t_m)$ *und* $x \in \mathbb{R}$. *Für* $0 \leq i \leq m-k-1$ *ist der* B-SPLINE

$B_{i,k}(x)$ *(der Index* t *fällt bei fixer Knotenfolge weg) induktiv definiert durch*

$$\begin{cases} B_{i,0}(x) = \begin{cases} 1 & \textit{falls } t_i \leq x < t_{i+1} \\ 0 & \textit{sonst} \end{cases} \\ \\ B_{i,k}(x) = \omega_{i,k}(x)B_{i,k-1}(x) + (1 - \omega_{i+1,k}(x))B_{i+1,k-1}(x) \quad \textit{für } k \geq 1 \end{cases}$$

Die obige Rekursionsformel ist auch als REKURSIONSFORMEL VON DE BOOR [3]-COX-MANSFIELD bekannt.
Bevor wir damit zur Definition der oben erwähnten Tensorprodukt-Patches kommen, seien einige fundamentale Eigenschaften von B-Splines angführt.

Proposition 2.1.2.2.

1. $B_{i,k}(x)$ *ist stückweise polynomial mit Grad* k*.*
2. $B_{i,k}(x) = 0$ *für* $x \notin [t_i, t_{i+k+1}[$*.*
3. $B_{i,k}(x) > 0$ *für* $x \in]t_i, t_{i+k+1}[$*. Es gilt also* $B_{i,k}(t_i) = 0$ *außer* t_i *ist ein Knoten der Ordnung* $k+1$*, d.h.* $t_i = t_{i+1} = ... = t_{i+k} < t_{i+k+1}$*. In diesem Fall ist* $B_{i,k}(t_i) = 1$*.*
4. *Sei* $[a, b]$ *ein Intervall mit* $t_k \leq a$ *und* $t_{m-k} \geq b$*. Dann ist*

$$\sum_{i=0}^{m-k-1} B_{i,k}(x) = 1 \quad \forall x \in [a, b[. \tag{2.4}$$

   Das heißt, die $B_{i,k}$ *bilden eine Partition der Eins von [a,b].*

5. *Sei* $x \in]t_i, t_{i+k+1}[$*. Dann ist* $B_{i,k,t}(x) = 1$ *dann und nur dann wenn* $t_{i+1} = ... = t_{i+k} = x$*.*
6. $B_{i,k}$ *ist rechtsstetig* $\forall x \in \mathbb{R}$ *(d.h.* $\exists \lim_{\epsilon \to 0} f(x + \epsilon)$ *und* $\lim_{\epsilon \to 0} f(x + \epsilon) = f(x)$*). Zudem ist* $B_{i,k}$ *rechtsdifferenzierbar mit*

$$B'_{i,k} = k\Big[\frac{B_{i,k-1}}{t_{i+k} - t_i} - \frac{B_{i+1,k-1}}{t_{i+k+1} - t_{i+1}}\Big], \tag{2.5}$$

   wobei man jene Ausdrücke deren Nenner gleich 0 *ist, durch* 0 *ersetzt.*

[3] Carl R. de Boor (* 1937 in Slupsk/Stulp in Pommern, heutiges Polen), deutsch-amerikanischer Mathematiker. Abschluss an der Universität Hamburg, Doktor an der University of Michigan 1966. Ab 1972 Professor an der University of Wisconsin-Madison, wo er mit Isaac Schönberg, dem Entdecker der Splines, zusammenarbeitete. Ebenso 1972 Veröffentlichung des Algorithmus in *On calculating with B-Splines*, J.Approx Theory 6. 2003 Verleihung der National Medal of Science in Mathematics.

Beweis: Die Eigenschaften *1.,2.,3.* sowie *6.* sind klar für $k = 0$ und folgen daher für $B_{i,k}$ durch Induktion nach k. Zudem ist *5.* eine Konsequenz von *3.* und *4.*
Es bleibt also *4.* zu zeigen. Auch das gelingt mittels Induktion nach k. Der Induktionsanfang für $k = 0$ ist offensichtlich. Für den Induktionsschritt sei $x \in [a, b]$. Dann gibt es j mit $k \leq j \leq m - k - 1$, sodass $x \in [t_j, t_{j+1}[$.
Falls $x = t_j$ und $B_{j,k} = 1$, so folgt aus *3.*, dass x ein Knoten der Ordnung k ist und damit die Behauptung.

Ansonsten ist wegen *2.*

$$\sum_{i=0}^{m-k-1} B_{i,k}(x) = \sum_{i=j-k}^{j} B_{i,k}(x)$$

und zudem definitionsgemäß

$$\sum_{j-k}^{j} B_{i,k}(x) = \sum_{j-k}^{j} \omega_{i,k} B_{i,k-1}(x) + \sum_{j-k}^{j} (1 - \omega_{i+1,k}) B_{j+1,k-1}(x).$$

Fassen wir diese Terme zusammen, so erhalten wir

$$\sum_{i=j-k}^{j} B_{i,k}(x) = \omega_{j-k} B_{j-k,k-1}(x) + \sum_{i=j+1-k}^{j} B_{i,k-1}(x) + (1 - \omega_{j+1,k}) B_{j+1,k-1}(x).$$

Weil $x \in [t_j, t_{j+1}[$ folgt aus Eigenschaft *2.* $B_{j-1,k-1}(x) = 0$ und $B_{j+1,k-1}(x) = 0$ und aus der Induktionsannahme

$$\sum_{i=j+1-k}^{j} B_{i,k-1}(x) = \sum_{i=0}^{m-k} B_{i,k-1}(x) = 1,$$

woraus *4.* folgt.
Als letztes soll noch die Formel für $B'_{i,k}$ in Eigenschaft *6* gezeigt werden, was wiederum mittels Induktion nach k funktioniert. Der Fall $k = 0$ ist trivial, da dann $B'_{i,k} = 0$. Nun gilt per Definition 2.1.2.1

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - ti} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x),$$

falls $t_i < t_{i+k}$ und $t_{i+1} < t_{i+k+1}$. Aus Differentiation und Induktion folgt

$$\begin{aligned}
B'_{i,k} &= \frac{B_{i,k-1}}{t_{i+k}-t_i} - \frac{B_{i+1,k-1}}{t_{i+k+1}-t_{i+1}} + (k-1)\left(\frac{x-t_i}{t_{i+k}-t_i}\left[\frac{B_{i,k-2}}{t_{i+k-1}-t_i} - \frac{B_{i+1,k-2}}{t_{i+k}-t_{i+1}}\right] + \right. \\
&\quad \left. + \frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+1}}\left[\frac{B_{i+1,k-2}}{t_{i+k}-t_{i+1}} - \frac{B_{i+2,k-2}}{t_{i+k+1}-t_{i+2}}\right]\right) = \\
&= \frac{B_{i,k-1}}{t_{i+k}-t_i} - \frac{B_{i+1,k-1}}{t_{i+k+1}-t_{i+1}} + \frac{k-1}{t_{i+k}-t_i}\left[\frac{x-t_i}{t_{i+k-1}-t_i}B_{i,k-2} + \frac{t_{i+k}-x}{t_{i+k}-t_{i+1}}B_{i+1,k-2}\right] - \\
&\quad - \frac{k-1}{t_{i+k+1}-t_{i+1}}\left[\frac{x-t_{i+1}}{t_{i+k}-t_{i+1}}B_{i+1,k-2} + \frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+2}}B_{i+2,k-2}\right].
\end{aligned}$$

Aus 2.1.2.1 für $B_{i,k-1}$ sowie für $B_{i+1,k-1}$ folgt dann die Behauptung.

□

Für B-Splines existieren zahlreiche Algorithmen zur Evaluierung, Differentiation sowie zum Einfügen neuer Knoten. Des weiteren lassen sich damit die Bézier-Kurven definieren, das zweidimensionale Äquivalent der Bézier-Patches. Da sie jedoch nicht direkten Einfluss auf die Herleitung der Dreiecks-Bézier-Patches haben, soll hier der Hinweis genügen [[14], Kapital 1.5 sowie Kapitel 2].

Wir jedoch definieren nun die oben erwähnten Tensorprodukt-Patches. Dazu wählen wir als Parameterbereich das Rechteck $[a,b] \times [c,d]$. Nun sei $u = (u_i)_{0 \le i \le m+k}$ eine Knotenfolge auf $[a,b]$ ebenso wie $v = (v_j)_{0 \le j \le n+l}$ auf $[c,d]$, welche die B-Splines $B_{i,k}(u)$ bzw. $B_{j,l}(v)$ definieren.

Definition 2.1.2.3. *Seien $P_{i,j} (0 \le i \le m-1, 0 \le j \le n-1)$ Punkte im $\mathbb{R}^s$. Dann definiert man mit obiger Notation einen* TENSORPRODUKT-PATCH *als Fläche der Form*

$$S(u,v) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} P_{i,j} B_{i,k}(u) B_{j,l}(v), \tag{2.6}$$

wobei $(u,v) \in [a,b] \times [c,d]$.

Äquivalent zu dieser Schreibweise ist auch

$$S(u,v) = \sum_{i=0}^{m-1} P_i(v) B_{i,k}(u) = \sum_{j=0}^{n-1} P_j(u) B_{j,l}(v),$$

wobei

$$P_i(v) = \sum_{j=0}^{n-1} P_{i,j} B_{j,l}(v),$$

$$P_j(u) = \sum_{i=0}^{m-1} P_{i,j} B_{i,k}(u).$$

Wie für den Fall der B-Splines gibt es auch für Tensorprodukt-Patches einen Evaluationsalgorithmus, zudem einen Algorithmus, welcher das Hinzufügen von neuen Punkten zum Kontrollpolygon ermöglicht (Osloalgorithmus), einen Subunterteilungsalgorithmus und einen Algorithmus zur Berechnung der Ableitungen, auf welche hier nicht näher eingegangen werden soll [[14], S.96, ff].
Möchte man nun mittels Tensorprodukt-Patches interpolieren, so ist dies möglich falls die zu interpolierenden Punkte auf einem rechteckigen Raster gegeben sind. Wir wollen also eine Spline-Oberfläche

$$S(u,v) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} P_{i,j} B_{i,k}(u) B_{j,l}(v)$$

finden, welche der Interpolationsbedingung

$$S(\alpha_\lambda, \beta\mu) = Q_{\lambda,\mu}, \qquad (0 \leq \lambda \leq m-1, 0 \leq \mu \leq n-1)$$

genügt.
Definiere man nun noch Matrizen A und B mit

$$A = (a_{\lambda,i}) = \Big(B_{i,k}(\alpha_\lambda)\Big) \qquad (\lambda, i) \in [0, m-1] \times [0, m-1]$$

$$B = (b_{\mu,j}) = \Big(B_{j,l}(\beta_\mu)\Big) \qquad (\mu, j) \in [0, n-1] \times [0, n-1].$$

Sei $Q = (Q_{\lambda,\mu})$ die Matrix der gegebenen Punkte und $P = (P_{i,j})$ jene der gesuchten Punkte. Damit werden die Interpolationsbedingungen zu

$$Q = AP^tB.$$

Diese Gleichung ist nun genau dann lösbar, wenn die Matrizen A und B invertierbar sind. Das ist der Fall, wenn $B_{i,k}(\alpha_\lambda) \neq 0$ und $B_{j,l}(\beta_\mu) \neq 0$, wie uns folgender Satz lehrt.
Gegeben seien n Punkte $P_0, ..., P_{n-1}$ im $\mathbb{R}^s$. Wir betrachten eine Spline-Kurve $B(t)$ vom

Grad k, welche durch die Punkte P_i geht. Weiters sei $(t_i)_{0\leq i\leq n+k}$ eine Knotenfolge (wir nehmen an dass $t_i < t_{i+k+1}, 0 \leq i \leq n-1$). Wir wollen Kontrollpunkte Q_j und Punkte $u_i \in \mathbb{R}$ finden, sodass die Kurve

$$B(t) = \sum_{j=0}^{n-1} B_{j,k}(t) Q_j$$

der Bedingung $B(u_i) = P_i$ genügt. Wir lösen also das lineare System

$$\sum_{j=0}^{n-1} B_{j,k}(u_i) Q_j = P_i$$

Satz 2.1.2.4. *Die Matrix $N = \Big(B_{j,k}(u_i)\Big)$ ist dann und nur dann invertierbar, wenn alle Diagonalelemente ungleich null sind, d.h. falls $B_{j,k}(u_j) \neq 0$ $(0 \leq j \leq n-1)$ oder falls $t_i < u_i < t_{i+k+1}$ $(0 \leq i \leq n-1)$.*

Vor dem Beweis noch zwei Anmerkungen zur Wahl der u_i:

1. Im Allgemeinen wählt man

$$u_i = t_i^* = \frac{t_{i+1} + ... + t_{i+k}}{k}.$$

   Dies impliziert, dass die Bedingung von Satz 2.1.2.4 erfüllt ist, außer wenn t_{i+1} ein Knoten der Ordnung $k+1$ ist, d.h. $t_i < t_{i+1} = ... = t_{i+k+1}$. Aus $B_{j,k} \neq 0$ folgt somit die eindeutige stabile Lösbarkeit des Interpolationsproblems. Zudem impliziert die Wahl $u_i = t_i^*$ eine geringere Oszillation als sie das Polygon (P_i) hat.

2. In manchen Fällen stehen die Parameter u_i mit $B(u_i) = P_i$ schon a priori fest. Dann hat man die Knotenpunkte t_j richtig zu wählen. Hierzu setzt man

$$\begin{cases} t_0 = ... = t_k = u_0 \\ t_{i+k} = \frac{u_i + ... + u_{i+k-1}}{k} \qquad (1 \leq i \leq n-k-1) \\ t_n = ... = t_{n+k} = u_{n-1} \end{cases}$$

Beweis: von Satz 2.1.2.4.
Wir beweisen zuerst die NOTWENDIGKEIT.

Angenommen es existiert ein i sodass $B_i(u_i)$ = 0, wobei $0 \leq i \leq n-1$. Zudem sei $u_i \leq t_i$ und bei $u_i = t_i$ sei $t_i < t_{i+k}$.

Dann enthält jede der ersten $(i+1)$ Reihen der Matrix N höchstens i Einträge ungleich 0, und zwar in der r-ten Reihe (mit $r \leq i$) $B_0(u_r), ..., B_{i-1}(u_r)$, da ja $B_j(u_r) = 0$ für $i \leq j$. Nun sind diese $(i+1)$ Reihen linear abhängig, woraus folgt dass die Matrix N singulär und damit nicht invertierbar ist.

Beweis der HINREICHENDEN BEDINGUNG.

Diese ist viel aufwendiger zu zeigen als die Notwendigkeit. Sei also zuerst

$$\sum_{j=0}^{n-1} \lambda_j B_j(u_i) = 0 \qquad (0 \leq i \leq n-1)$$

und $B(t) = \sum_{j=0}^{n-1} \lambda_j B_j(t)$. Aus der Annahme folgt dann $B(u_i) = 0$ $(0 \leq i \leq n-1)$. Wir wollen zeigen, dass dies zusammen mit der Annahme $B_i(u_i) \neq 0$ $(0 \leq i \leq n-1)$ impliziert, dass $\lambda_j = 0$ $(0 \leq j \leq n-1)$.
Wir benötigen zudem noch folgende Eigenschaften bezüglich der linearen Unabhängigkeit von B-Splines in Abhängigkeit von der Knotenfolge $(t_i)_{0 \leq i \leq n+k}$:

Lemma 2.1.2.5.

1. *Angenommen $i \geq k$ und $t_i < t_{i+1}$. Dann sind die B-Splines $B_{i-k}(t), ..., B_i(t)$, welche auf das Intervall $[t_i, t_{i+1}[$ beschränkt sind, linear unabhängig.*

2. *Falls $0 \leq i < k$ und $t_i < t_{i+1}$, dann sind die B-Splines $B_0(t), ..., B_i(t)$ beschränkt auf das Intervall $[t_i, t_{i+1}]$ linear unabhängig.*

Wir schließen aus diesem Lemma, dass falls $B(t) \equiv 0$ für $t \in [t_i, t_{i+1}[$ mit $(t_i < t_{i+1})$, dann ist $\lambda_{i-k} = ... = \lambda_i = 0$. Weiters gilt für alle Intervalle $[t_j, t_{j+k+1}[$, dass falls ein i aus $[j, j+k+1]$ mit $t_i < t_{i+1}$ existiert, sodass $B(t)|_{[t_i, t_{i+1}[} \equiv 0$, dass dann $B(t) \equiv 0$. Dies würde schon als Beweis genügen, daher können wir annehmen, dass ein Intervall $I = [t_r, t_s[$ mit folgenden Eigenschaften existiert:

1. Auf keinem der in $[t_r, t_s[$ enthaltenen Teilintervalle $[t_i, t_{i+1}[$ (mit $t_i < t_{i+1}$) ist $B(t) \equiv 0$;

2. $s \geq r + k + 1$;

3. I ist maximal mit diesen Eigenschaften.

Wir setzen nun

$$\tilde{B}(t) = \lambda_{r-k} B_{r-k}(t) + ... + \lambda_{s-1} B_{s-1}(t) = \sum_{i=r-k}^{s-1} \lambda_i B_i(t).$$

Aus Eigenschaft 3 folgt nun, dass $B(t) \equiv 0$ auf $[t_{r-1}, t_r[$ sowie $[t_s, t_{s+1}[$. Aus 2.1.2.5 folgt $\lambda_{r-k} = ... = \lambda_{r-1} = 0$ und $\lambda_{s-k} = ... = \lambda_s = 0$, was wiederum

$$\tilde{B}(t) = \lambda_r B_r(t) + ... + \lambda_{s-k-1} B_{s-k-1}(t) = \sum_{i=r}^{s-k-1} \lambda_i B_i(t)$$

impliziert. Aus der Voraussetzung des Satzes 2.1.2.4 $B_i(u_i) \neq 0$ gilt zudem

$$t_r < u_r < ... < u_{s-k-1} < t_s.$$

Weil zudem $\tilde{B}(u_i) = B(u_i)$ für $r \leq i \leq s-k-1$ gilt und $\tilde{B}(t)$ nicht identisch Null auf irgendeinem Teilintervall von $[t_i, t_{i+1}[$ ist, hat die Funktion $\tilde{B}(t)$ mindestens $s-k-r$ verschiedene Nullstellen im Intervall $[t_r, t_s[$.
Sei nun $V(\tilde{B})$ die Variation der Funktion $\tilde{B}$, d.h. die Anzahl der Vorzeichenwechsel. Es gilt aber folgende Eigenschaft für Spline-Funktionen: Sei $a = (a_0, a_1, ..., a_{n-1})$ eine Folge und $S(x) = \sum_{i=0}^{n-1} a_i B_{i,k}(x)$ eine Spline-Funktion auf $[a_0, a_{n-1}]$ mit $t_k \leq a_0$ und $t_n \geq a_{n-1}$. Dann gilt $V(S) \leq V(a_i)$.
Angenommen, dass alle Nullstellen einfach sind. Dann muss, da wir mindestens $s-k-r$ verschiedene Nullstellen haben, $V(\tilde{B}) \geq s-r-k$ sein. Dies widerspricht aber $V(\tilde{B}) \leq V(\lambda_r, ...\lambda_{s-k-1})$, was aus der eben beschriebenen Eigenschaft der Variation folgt, da klarerweise $V(\lambda_r, ...\lambda_{s-k-1}) \leq s-r-k-1$ gilt.
Im allgemeinen Fall (d.h. nicht alle Nullstellen sind einfach) folgt aus der Tatsache, dass $\tilde{B}$ mindestens $s-k-r$ verschiedene Nullstellen in $[t_r, t_s[$ hat, dass für beliebig kleines $\epsilon > 0$ eine der beiden Funktionen

$$\tilde{B}(t) + \epsilon \sum_{j=r}^{s-k-1} B_j(t) \text{ bzw. } \tilde{B}(t) - \epsilon \sum_{j=r}^{s-k-1} B_j(t)$$

mindestens $s-r-k$ einfache Nullstellen in $[t_r, t_s[$ hat. Mit oben verwendeter Argumentation folgt, dass solche $\tilde{B}$ nicht existieren und daher alle $\lambda_i = 0$ sind. Das heißt aber, dass die Spalten von N linear unabhängig sind und somit N invertierbar ist.

□

Mit diesen Eigenschaften und der Übertragbarkeit von Algorithmen werden Tensorprodukt-Patches häufig verwendet. Sie sind jedoch nicht frei von Nachteilen:

- Es ist schwierig Flächen zu approximieren, die nicht auf einem rechteckigen Parameterbereich definiert sind (z.B. auf Triangulierungen). Triangulierungen erlauben aber viel freiere Parameterbereiche.

- Es existieren zwei bevorzugte Familien von Kurven auf einem Tensorprodukt-Patch, und zwar jene, welche durch u = const sowie durch v = const definiert sind. Da dies Splines in einer Variable sind, lassen sich hier Eigenschaften wie die Krümmung leicht bestimmen [14]. Im Gegensatz dazu sind Kurven in andere Richtungen, etwa auch $u + v$ = const, viel schwieriger zu untersuchen.

- Selbst im am häufigsten verwendeten Fall, nämlich dem bikubischen mit k = l = 3, ist jede Koordinate vom Totalgrad 6 in u und v. Daraus folgt aber, dass die implizite Gleichung $F(x, y, z) = 0$ eines bikubischen Tensorprodukt-Patches eine algebraische Gleichung vom Grad 18 ist. Der Schnitt zweier solcher Oberflächen ist dann eine algebraische Gleichung vom Grad $18^2 = 324$, was Berechnungen schnell äußerst aufwendig macht.

Aus diesen Überlegungen heraus wählt man nun einen anderen Zugang zu B-Spline-Flächen, und zwar jenen, in dem die Flächenteile der Parametrisierung Dreiecke sind. Durch Triangulierungen ermöglicht dies nun die Verwendung von sehr allgemeinen Parameterbereichen.

2.2 Bézier-Patches

Bevor wir den Begriff des Dreiecks-Bézier-Patches einführen, definieren wir noch die dafür wichtigen baryzentrischen Koordinaten und danach die Bernstein-Polynome höheren Grades.

2.2.1 Baryzentrische Koordinaten

Definition 2.2.1.1 (Baryzentrische Koordinaten)**.** *Sei $\Delta \subset \mathbb{R}^2$ ein Dreieck mit den Eckpunkten A, B und C und Fläche 1. Dann definiert man die* BARYZENTRISCHEN KOORDINATEN (α, β, γ) *eines Punktes $P \in \mathbb{R}^2$ bezüglich (A, B, C) durch folgende Eigenschaften:*

- $\alpha + \beta + \gamma = 1$.
- *P liegt im Inneren des Dreicks Δ dann und nur dann, wenn $\alpha > 0, \beta > 0, \gamma > 0$.*
- *Liegt P im Inneren des Dreiecks Δ, so sind r, s und t jeweils der Flächeninhalt der Dreiecke PBC, PAC und PAB.*
- *Schließlich lässt sich das Dreieck wie folgt schreiben:*
  $\Delta := \{\alpha A + \beta B + \gamma C | \alpha, \beta, \gamma \geq 0, \alpha + \beta + \gamma = 1\}$.

Bei Dreiecks-Bézier-Patches spielen baryzentrische Koordinaten eine wichtige Rolle, da sie die drei Variablen der parametrisierten Fläche bilden werden. Seien nun die Punkte $P_{i,j,k} \in \mathbb{R}^3$ die Punkte des Kontrollpolygons des Bézier-Patches, so bildet die Funktion $\pi : \mathbb{R}^3 \longrightarrow \mathbb{R}^2$ die Punkte auf das dem Bezierpatch zugrunde liegende Dreieck mit den Eckpunkten P_{300}, P_{030} sowie P_{003} ab. Dann errechnen sich die baryzentrischen Koordinaten α, β und γ von P wie folgt:

Seien

$$A = P_{003} = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}, B = P_{030} = \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix}, C = P_{300} = \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix}, P = \begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix}.$$

Die baryzentrischen Koordinaten α, β, γ von P werden durch folgende Ausdrücke berechnet:

$$\gamma(x_P, y_P) = \frac{(y_A - y_B) * x_P + (x_B - x_A) * y_P + x_A * y_B - x_B * y_A}{(y_A - y_B) * x_C + (x_B - x_A) * y_C + x_A * y_B - x_B * y_A},$$

$$\beta(x_P, y_P) = \frac{(y_A - y_C) * x_P + (x_C - x_A) * y_P + x_A * y_C - x_C * y_A}{(y_A - y_C) * x_B + (x_C - x_A) * y_B + x_A * y_C - x_C * y_A}, \tag{2.7}$$

$$\alpha(x_P, y_P) = 1 - \beta - \gamma.$$

2.2.2 Bernsteinpolynome vom Grad n

In Äquivalenz zum zweidimensionalen Fall der Bézier-Kurven [[14], Kapitel 2] stellen die Bernsteinpolynome die Grundlage zur Definition der Bézier-Patches dar und sollen daher als Erstes eingeführt werden.

Definition 2.2.2.1. *Sei Δ ein Dreieck im $\mathbb{R}^2$ und $n \in \mathbb{N}$ fix. Weiters sollen (r,s,t) die baryzentrischen Koordinaten von Punkten $P \in \mathbb{R}^2$ bezüglich Δ sein. Dann sind die* BERNSTEIN-POLYNOME VOM GRAD N *definiert durch*

$$B^n_{i,j,k} = \frac{n!}{i!j!k!} r^i s^j t^k, \qquad i + j + k = n. \tag{2.8}$$

Da im Weiteren zur Konstruktion der Bézier-Patches die Bernsteinpolynome vom Grad 3 verwenden werden, sollen ebenjene Basiselemente explizit angeführt werden (es existieren 10, wie aus der nachfolgenden Proposition hervorgehen wird).

Beispiel 2.2.2.2 (Bernsteinpolynome für n=3)**.**

$$s^3$$

$$3s^2t \qquad 3rs^2$$

$$3st^2 \qquad 6rst \qquad 3r^2s$$

$$t^3 \qquad 3rt^2 \qquad 3r^2t \qquad r^3$$

Proposition 2.2.2.3 (Eigenschaften der Bernsteinpolynome)**.**

1. *Es existieren $\frac{(n+1)(n+2)}{2}$ Bernstein-Polynome vom Grad $\leq$ n. Für n = 3 existieren somit genau 10 Bernsteinpolynome.*
2. *Die Funktionen $B^n_{i,j,k}(r, s, t)$ bilden eine Basis für den Raum aller Polynome in r und s vom Grad $\leq$ n. Diese Basis wird* BERNSTEIN-BASIS *genannt.*

3. *Zudem gilt folgende Induktionsformel:*

$$B^n_{i,j,k} = rB^{n-1}_{i-1,j,k} + sB^{n-1}_{i,j-1,k} + tB^{n-1}_{i,j,k-1}.$$

4. *Weiters ist mit* $t = 1 - r - s$

$$\sum_{i+j+k=n} B^n_{i,j,k}(r, s, 1 - r - s) = 1.$$

5. *Falls* $P \in \mathbb{R}^2$ *die baryzentrischen Koordinaten bezüglich* Δ *hat, so ist* $B^n_{i,j,k}(r, s, t) > 0$ *im Inneren der Menge* Δ.

6. *Es ist möglich, den Grad von* $B^n_{i,j,k}(r, s, t)$ *zu erhöhen, d.h. man betrachtet* $B^n_{i,j,k}(r, s, t)$ *als Polynom vom Grad* $\leq$ *n. Es gilt*

$$B^n_{i,j,k} = (r + s + t)B^n_{i,j,k}$$

$$B^n_{i,j,k} = \frac{1}{n+1}\left[(i+1)B^{n+1}_{i+1,j,k} + (j+1)B^{n+1}_{i,j+1,k} + (k+1)B^{n+1}_{i,j,k+1}\right].$$

Beweis:

1. Um dies zu zeigen, genügt ein einfaches induktives Abzählargument, da bei jeder Erhöhung des Grades von n auf $n + 1$ um genau $n + 2$ Polynome mehr erzeugt werden.

2. Unter Verwendung von Eigenschaft 1 ist die lineare Unabhängigkeit der Polynome ausreichend. Nun ist diese aber klar, weil der Term mit dem niedrigsten Grad in dem Ausdruck $r^i s^j (1 - r - s)^k$ genau $r^i s^j$ ist.

3. Betrachte den Koeffizienten des Terms $r^i s^j t^k$ auf der rechten Seite. Für diesen gilt

$$\frac{(n-1)!}{(i-1)!(j-1)!(k-1)!}\left[\frac{1}{jk} + \frac{1}{ik} + \frac{1}{ij}\right] = \frac{n!}{i!j!k!}.$$

   Die Gleichheit folgt aus der Multiplikation und anschließender Substitution durch $i + j + k = n$.

4. Aus dem multinomischen Lehrsatz folgt dass $B^n_{i,j,k} = \frac{n!}{i!j!k!} r^i s^j t^k$ gleich der Entwicklung von $(r + s + t)^n$ entspricht. Das Resultat ergibt sich aus $t = 1 - r - s$.

5. Folgt direkt aus der Definition der Bernstein-Polynome sowie jener der baryzentrischen Koordinaten, da $r, s, t > 0$.

6. Da man den Bruch in der Definition der Bernsteinpolynome ebenso als Multinominalkoeffizienten auffassen kann, folgt die Eigenschaft 6 wiederum aus der Induktionsformel für Multinomialkoeffizienten.

□

2.2.3 Dreiecks-Bézier-Patch

In Äquivalenz zum 2-dimensionalen Fall der B-Splines lässt sich nun mithilfe der Bernsteinpolynome der Begriff des Bézier-Patches einführen.

Definition 2.2.3.1. *Gegeben seien Punkte $P_{i,j,k}$ mit $i+j+k=n$ im $\mathbb{R}^s$ sowie ein Dreieck $\Delta \in \mathbb{R}^2$. Als* DREIECKS-BÉZIER-PATCH *bezeichnet man dann jene parametrische Fläche erzeugt durch die Abbildung $B : \Delta \longrightarrow \mathbb{R}^s$, welche definiert ist durch*

$$B(r,s,t) = \sum_{i+j+k=n} P_{i,j,k} B^n_{i,j,k}(r,s,t). \tag{2.9}$$

Hierbei ist die Wahl der Punkte $P_{i,j,k}$, welche das sogenannte Kontrollpolyeder bilden, keineswegs eindeutig. Verschiedene Arten der Triangulierungen, so z.B. Delaunay - Triangulierungen über Voronoi - Diagrammen [siehe dazu [14], Kapitel 5], kommen hier zur Anwendung.
Wenden wir uns wieder dem Bézier-Patch zu und seinen Eigenschaften, welche eine Verallgemeinerung jener der Bézier-Kurven sind.

Proposition 2.2.3.2.

1. *Der Patch $B(r,s,t)$ berührt die Eckpunkte $P_{n,0,0}, P_{0,n,0}$ und $P_{0,0,n}$.*
2. *Der Rand des Bézier-Patches wird gebildet durch drei Bézier-Kurven, d.h. den Bildern der Seiten des Dreiecks Δ, welche durch $r=0, s=0$ und $t=0$ definiert sind. Das Kontrollpolygon der ersten Randkurve ist dann $(P_{0,0,n}, P_{0,1,n-1}, ..., P_{0,n,0})$.*
3. *Der Patch ist in der konvexen Hülle der Punkte $P_{i,j,k}$ enthalten.*
4. *Die Tangentialebenen an $B(r,s,t)$ in den Endpunkten $P_{n,0,0}, P_{0,n,0}$ und $P_{0,0,n}$ werden jeweils von den zwei benachbarten Punkten aufgespannt, so wird z.B. die Ebene in $P_{0,0,n}$ aufgespannt durch die Punkte $(P_{0,0,n}, P_{0,1,n-1}, P_{1,0,n-1})$.*
5. *Das Bild unter B einer beliebigen geraden Linie der (r,s,t) -Ebene ist eine Kurve vom Grad n.*

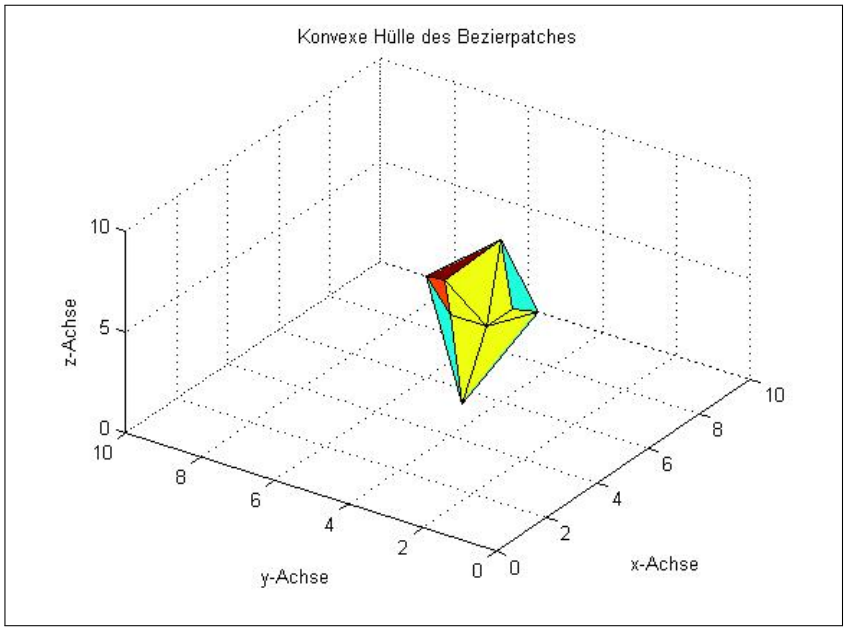

Abbildung 2.1: 2.2.3.2 Eigenschaft 3 - Bézier-Patch in seiner konvexen Hülle, gebildet durch die Punkte des Kontrollpolygons.

Beweis:

1. Dass der Patch die Eckpunkte berührt, folgt direkt aus

$$B^n_{n,0,0}(1,0,0) = B^n_{0,n,0}(0,1,0) = B^n_{0,0,n}(0,0,1) = 1.$$

   In diesem Fall reduziert sich die Funktionsgleichung auf den entsprechenden Punkt des Kontrollpolygons.

2. Setze zum Beispiel $i = 0$ und erhalte mit $B^n_{0,j,k}(0, s, 1-s)$ die eindimensionalen Bernstein-Polynome in s vom Grad n. Daraus folgt die Behauptung.

3. Es gilt $\sum_{i+j+k=n} B^n_{i,j,k}(r, s, 1-r-s) = 1$. Daher ist $\sum_{i+j+k=n} P_{i,j,k} B^n_{i,j,k}(r, s, t)$ als Linearkombination eine konvexe Kombination.

4. Dies folgt aus der Tatsache, dass die Vektoren $\overrightarrow{P_{0,0,n}P_{0,1,n-1}}$ sowie $\overrightarrow{P_{0,0,n}P_{1,0,n-1}}$ die Tangenten an die zwei Bézier-Kurven sind, welche den Rand des Patches bei $P_{0,0,n}$ bilden. [[14], Punkt 2.2.2.a]

5. Dies lässt sich durch Nachrechnen überprüfen. So z.B. für den Fall $n = 3$:

$$B(r,s,t) = P_{3,0,0}r^3 + P_{0,3,0}s^3 + P_{0,0,3}t^3 + P_{0,2,1}s^2t + P_{0,1,2}st^2 + P_{1,0,2}rt^2 + P_{2,0,1}r^2t + P_{2,1,0}r^2s + P_{1,2,0}rs^2 + P_{1,1,1}rst$$

Dies ist eine parametrisierte Kurve vom Grad n. Diese Aussage ist im Gegensatz zu Bézier-Patches für Tensorprodukt-Patches nicht gültig.

□

Eine Verallgemeinerung des Bézier-Patches erreicht man durch folgende

Definition 2.2.3.3. *Ein* DREIECKIGER RATIONALER BÉZIER-PATCH *vom Grad n zum Kontrollpolygon $P_{i,j,k}$ und zu den Gewichten $w_{i,j,k}$ ist definiert als das Bild des Simplex Δ unter der Abbildung der Form*

$$R(r,s,t) = \frac{P(r,s,t)}{Q(r,s,t)}$$

mit

$$P(r,s,t) = \sum_{i+j+k=n} P_{i,j,k}w_{i,j,k}B^n_{i,j,k}(r,s,t),$$

$$Q(r,s,t) = \sum_{i+j+k=n} w_{i,j,k}B^n_{i,j,k}(r,s,t).$$

Bevor wir uns nun einigen Algorithmen, die für Berechnungen mit Bézier-Patches nützlich sind, zuwenden, betrachten wir noch folgende Eigenschaft, und zwar jene der GRADERHÖHUNG.
Wir haben

$$B(r,s,t) = \sum_{i+j+k=n} P^*_{i,j,k}B^n_{i,j,k}.$$

Nun wenden wir 2.2.2.3, Eigenschaft 6, an und erhalten

$$B(r,s,t) = \sum_{i+j+k=n+1} P^*_{i,j,k}B^{n+1}_{i,j,k}$$

mit einem neuen Kontrollpolygon $(P^*_{i,j,k})_{i+j+k=n+1}$ definiert durch

$$P^*_{i,j,k} = \frac{1}{n+1}\Big(iP_{i-1,j,k} + jP_{i,j-1,k} + kP_{i,j,k-1}\Big).$$

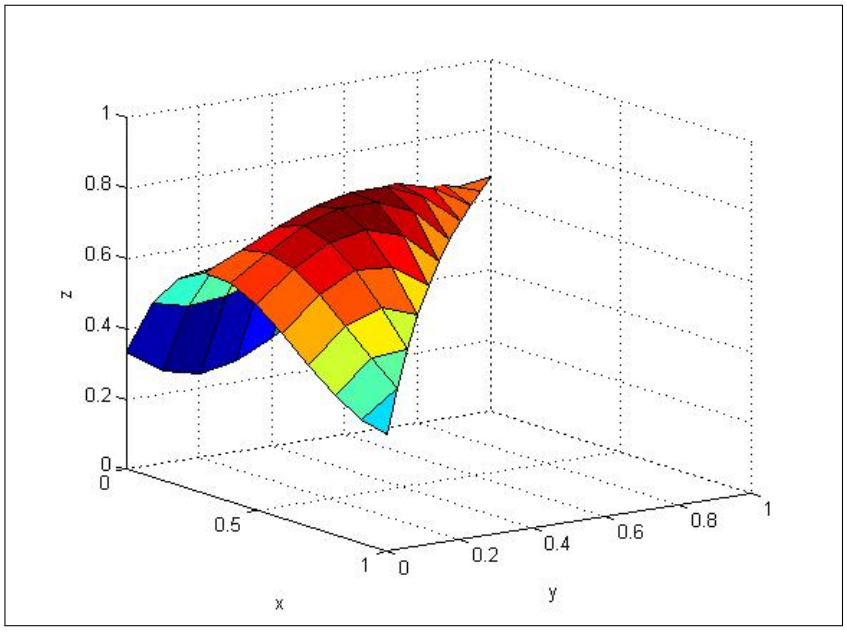

Abbildung 2.2: Beispiel eines Dreiecks-Bézier-Patch, dessen Kontrollpunkte einer Delaunay-Triangulierung entsprechen

Nun werden wir zwei wichtigen Algorithmen für Dreiecks-Bézier-Patches vorstellen, die für Anwendungen von großer Bedeutung sind.

2.2.4 Algorithmen für Bézier-Patches

Algorithmus 2.2.4.1 (DE CASTELJAU ALGORITHMUS)**.**

Der Algorithmus verwendet die Induktionsformel aus 2.2.2.3.

Sei also $B(r,s,t) = \sum_{i+j+k=n} P_{i,j,k} B^n_{i,j,k}(r,s,t)$ *ein Dreiecks-Bézier-Patch, so wollen wir* $B(r,s,t)$ *für fixe Werte von* r,s,t *auswerten.*

1. *Setze* $P^0_{i,j,k} = P_{i,j,k} (i+j+k=n)$.

2. *Löse für* $1 \leq l \leq n$ *mit* $i+j+k = n-l$

$$P^l_{i,j,k} = rP^{l-1}_{i+1,j,k} + sP^{l-1}_{i,j+1,k} + tP^{l-1}_{i,j,k+1}.$$

3. *Es gilt* $B(r,s,t) = P^n_{0,0,0}(r,s,t)$.

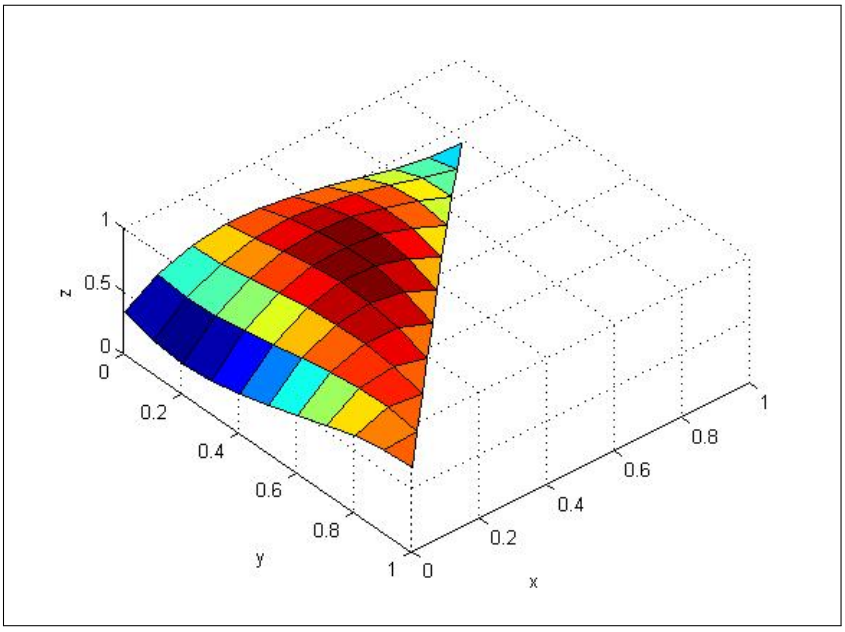

Abbildung 2.3: Beim Blick von unten wird die dreieckige Grundform des Bézier-Patches deutlich

Der De Casteljau Algorithmus benötigt eine hohe Anzahl von Rechenoperationen, liefert jedoch auch mehr Informationen als nur die Evaluation von $B(r,s,t)$.
Wir werden im folgenden sehen, dass jene Punkte, welche während des De Casteljau Algorithmus berechnet wurden, verwendet werden können zur Berechnung der Ableitungen des Bézier-Patches. So ist die Tangentialebene an den Patch im Punkt $B(r,s,t)$ jene Ebene die $P^{n-1}_{1,0,0}, P^{n-1}_{0,1,0}$ und $P^{n-1}_{0,0,1}$ enthält.
Außerdem lässt sich aus dem De Casteljau Algorithmus folgender Subuntereilungsalgorithmus ableiten.

Algorithmus 2.2.4.2 (SUBUNTERTEILUNGSALGORITHMUS)**.**

Ähnlich dem eindimensionalen Fall ermöglicht der De Casteljau Algorithmus den Bézier-Patch zu unterteilen und in Folge durch einen Polyeder zu approximieren.

1. *Verwende den De Casteljau Algorithmus 2.2.4.1 für* $(r,s,t) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$

2. *Nun ist der Patch geteilt in drei Teilpatches mit den Kontrollpolyedern* K_1, K_2 *bzw.* K_3

$$K_1 = \left\{P^i_{0,n-i,0}, P^i_{0,0,n-i}, P^0_{0,i,n-i} | i = 0,...,3\right\},$$

$$K_2 = \left\{P^i_{n-i,0,0}, P^i_{0,0,n-i}, P^0_{i,0,n-i} | i = 0,...,3\right\},$$

$$K_3 = \left\{P^i_{0,n-i,0}, P^i_{n-i,0,0}, P^0_{i,n-i,0} | i = 0,...,3\right\}.$$

Dieses Verfahren lässt sich nun natürlich iterieren, um im u-ten Schritt den ursprünglichen Bézier-Patch in 3^u Teile zu unterteilen.
Mit Π_u^i wollen wir nun die neuen Kontrollpolyeder der im u-ten Schritt entstandenen Flächenteile bezeichnen, wobei $i = 0, ..., 3^u - 1$. Im Gesamten haben diese Polyeder $\frac{3^u k(k+1)}{2} + 1$ Ecken, unter denen $3^u + 1$ Punkte des ursprünglichen Flächenstücks enthalten sind. Diese Punkte sind genau $B(r/3^r, s/3^r, t/3^r)$ mit $r = 0, ..., 3^u, s = 0, ..., 3^u$ sowie $t = 0, ..., 3^u$. Zudem gilt $r + s + t = 1$. Des weiteren ergibt sich noch folgende

Proposition 2.2.4.3. *Die Folge der Polyeder Π_u^i konvergiert für $n \to \infty$ mit Konvergenzgeschwindigkeit $\lambda/3^u$ gegen das Flächenstück $\bar{B}$ des Bézier-Patches B(r,s,t), wobei λ eine von u unabhängige Konstante bezeichnet.*

Diese Eigenschaft hat nun in der Bildverarbeitung große Relevanz: Man rufe sich die nur endliche Auflösung von Bildschrimen in Erinnerung, so lässt sich mit dem Subunterteilungsalgorithmus eine Folge von Polyedern konstruieren, die nach wenigen Schritten nicht mehr von $\bar{B}$ unterscheidbar ist.
Ebenso aus dem De Casteljau Algorithmus ableitbar sind folgende Eigenschaften der Differentiation von Bézier-Patches.

2.2.5 Differentiation von Bézier-Patches

Man betrachte eine Gerade der Form $X(u) = X_0(1-u) + X_1 u$, mit $X_0 = (r_0, s_0, t_0)$ und $X_1 = (r_1, s_1, t_1)$. Wir berechnen nun den Differenialoperator D_X von B in Richtung der Kurve $B(X(u))$.

Proposition 2.2.5.1. *Es gilt*

$$D_X B(X_0) = n \sum_{i+j+k=n-1} B_{i,j,k}^{n-1}(r_0, s_0, t_0)\Big(\Delta r_0 P_{i+1,j,k} + \Delta s_0 P_{i,j+1,k} + \Delta t_0 P_{i,j,k+1}\Big)$$

mit

$$\Delta r_0 = r_1 - r_0$$

$$\Delta s_0 = s_1 - s_0$$

$$\Delta t_0 = t_1 - t_0.$$

Beweis: Bilde die Ableitung nach u von

$$B_{i,j,k}^n(u) = \frac{n!}{i!j!k!} r^i(u) s^j(u) t^k(u)$$

und setze danach $u = 0$, daraus folgt die Eigenschaft.

□

Diese Formel lässt sich auf Ableitungen höherer Ordnung verallgemeinern und ermöglicht so die Bestimmung einer Taylorreihe von $B(r, s, t)$ an einem gegebenen Punkt. Zudem gilt folgende Eigenschaft:

Korollar 2.2.5.2. *Die* TANGENTIALEBENE AN DEN BÉZIER-PATCH *im Punkt* $B(r, s, t)$ *ist jene Ebene, welche durch die im De Casteljau Algorithmus 2.2.4.1 errechneten Punkte* $P^{n-1}_{1,0,0}$, $P^{n-1}_{0,1,0}$ *und* $P^{n-1}_{0,0,1}$ *definiert ist.*

Beweis: Um das Korollar zu beweisen, verwenden wir den De Casteljau Algorithmus 2.2.4.1. Wegen der Proposition 2.2.5.1 gilt

$$\begin{aligned} D_X B(X_0) =& n\Big[\Delta r_0 \sum B^{n-1}_{i,j,k}(r_0, s_0, t_0) P_{i+1,j,k} + \\ &+ \Delta s_0 \sum B^{n-1}_{i,j,k}(r_0, s_0, t_0) P_{i,j+1,k} + \Delta t_0 \sum B^{n-1}_{i,j,k}(r_0, s_0, t_0) P_{i,j,k+1}\Big], \end{aligned}$$

wobei die Summen wie vorher über $i + j + k = n - 1$ laufen.
Nun wenden wir den De Casteljau Algorithmus 2.2.4.1 auf die drei Summen an, und zwar für fixe Werte $(r, s, t) = (r_0, s_0, t_0)$. Dann ist z.B. die erste Summe

$$\sum_{i+j+k=n-1} B^{n-1}_{i,j,k} P_{i+1,j,k}$$

ein Dreiecks-Bézier-Patch vom Grad $n - 1$, welcher erzeugt wird durch das Kontrollpolygon $(P_{i,j,k})_{(i\geq 1)}$. Der De Casteljau Algorithmus läuft bis zum Punkt $P^{n-1}_{1,0,0}$. Schließlich erhalten wir

$$D_X B(X_0) = n\Big[\Delta r_0 P^{n-1}_{1,0,0} + \Delta s_0 P^{n-1}_{0,1,0} + \Delta t_0 P^{n-1}_{0,0,1}\Big].$$

Dies impliziert, dass die Tangentialebene bei $B(r_0, s_0, t_0)$ an die Kurve $B(X(u))$ in der Ebene enthalten ist, welche durch $P^{n-1}_{1,0,0}$, $P^{n-1}_{0,1,0}$ sowie $P^{n-1}_{0,0,1}$ definiert ist.

□

Mit diesen Eigenschaften und Algorithmen kommen wir abschließend zu einer für den Anwendungsbereich der Computergrafik entscheidenen Eigenschaft der Bézier-Patches, nämlich der Junktion.
Diese ist notwendig bei einer Erweiterung des Ganzen auf Flächen, welche als Zusammensetzung von Dreiecks-Bézier-Patches über einer entsprechenden Triangulierung des Parameterbereichs realisiert werden.

2.2.6 Junktion zwischen Bézier-Patches

Seien zwei Dreiecke Δ und Δ' gegeben mit den baryzentrischen Koordinaten (r, s, t) bzw. (r', s', t') und seien $B(r, s, t)$ sowie $B'(r', s', t')$ die dazugehörigen Bézier-Patches. Wir können aufgrund der Eigenschaft der Graderhöhung annehmen, dass beide Bézier-Patches vom selben Grad n sind. O.B.d.A. nehmen wir zudem an, dass Δ und Δ' eine Kante gemeinsam haben, und zwar jene mit den baryzentrischen Koordinaten $r = 0$ sowie $r' = 0$.
Nun gelten für die Parameter folgende Relationen:

$$\begin{cases} r' = -\alpha r \\ s' = s + \beta r \qquad\qquad (\alpha > 0, 1 + \alpha = \beta + \gamma), \\ t' = t + \gamma r \end{cases}$$

Um C^0-STETIGKEIT zu erhalten, müssen die Bilder bei $r = 0$ und $r' = 0$, d.h. $B(0, s, 1-s)$ und $B'(0, s', 1 - s')$, übereinstimmen. Daraus folgt, dass $P_{0,j,k} = P'_{0,j,k}$ gilt mit $j + k = n$.

Für C^1-STETIGKEIT müssen zusätzlich zu dieser Bedingung noch die ersten Ableitungen von B bzw. B' entlang der gemeinsamen Kurve, d.h. dem Bild unter B oder B' des Segments $r = r' = 0$, übereinstimmen.
Dazu seien für fixes j affine Abbildungen g_j und g'_j gegeben, welche Δ bzw. Δ' auf das Dreieck $D_j = (P_{0,j,k}, P_{0,j+1,k-1}, P_{1,j,k-1})$ bzw. auf das Dreieck $D'_j = (P_{0,j,k}, P_{0,j+1,k-1}, P'_{1,j,k-1})$ abbilden.
Bezeichne nun den Vektor $\overrightarrow{X_1X_2}$ durch X, dann gilt ja wegen 2.2.5.1 für den Differentiationsoperator D_X in Richtung X

$$D_X B(X) = n \sum_{i+j+k=n-1} B^{n-1}_{i,j,k} \Big(\Delta_{r_0} P_{i+1,j,k} + \Delta_{s_0} P_{i,j+1,k} + \Delta_{t_0} P_{i,j,k+1} \Big).$$

Nun sind die baryzentrischen Koordinaten von X gerade jene $\Delta_{r_0}, \Delta_{s_0}$ und Δ_{t_0}, welche in 2.2.5.1 definiert wurden. Daraus folgt

$$g_j(X) = P_{0,j,k}\Delta_{r_0} + P_{0,j+1,k-1}\Delta_{s_0} + P_{1,j,k-1}\Delta_{t_0}.$$

Liegt nun der Punkt x auf der Gerade $r = 0$ in Δ dann folgt

$$D_X B(x) = n \sum_{j+k=n-1} B^{n-1}_{0,j,k}(x) g_j(X).$$

Ebenso ist

$$D_X B'(x) = n \sum_{j+k=n-1} B^{n-1}_{0,j,k}(x) g'_j(X).$$

Die C^1-Stetigkeit entlang $r = r' = 0$ liegt dann vor, wenn die Abbildungen g_j und g'_j für $0 \leq j \leq n$ übereinstimmen. Äquivalent dazu ist, dass $D_j \cup D'_j$ das Bild von $\Delta \cup \Delta'$ unter einer eindeutig bestimmten Abbildung ist. Daraus folgt schließlich dass die Dreiecke D_j und D'_j KOMPLANAR sind.

Begnügt man sich mit G^1-STETIGKEIT, d.h. fordert man nur, dass die Tangentialebenen in den gemeinsamen Punkten übereinstimmen, so erreicht man eine größere Freiheit bei der Wahl der Punkte, auch wenn die entsprechenden Formel komplizierter werden, wie wir nun sehen werden.
Wir beginnen unsere Betrachtungen dazu mit zwei Bézier-Patches $B(u, v)$ bzw. $B'(u, v)$ mit $(u, v) \in [0, 1]^2$, die entlang der Kurve $B(1, v) = B'(0, v)$ verbunden werden. Die Eigenschaft, dass die Tangentialebenen entlang der gemeinsamen Kurve übereinstimmen, entspricht folgender Relation:

$$\alpha(v) k \overbrace{\sum_{j=0}^{l} (P_{k,j} - P_{k-1,j}) B^l_j(v)}^{\frac{\partial B}{\partial u}} + \beta(v) k \overbrace{\sum_{j=0}^{l} (P'_{1,j} - P'_{0,j}) B^l_j(v)}^{\frac{\partial B'}{\partial s}} +$$

$$+ \gamma(v) l \underbrace{\sum_{j=0}^{l-1} (P_{k,j+1} - P_{k,j}) B^{l-1}_j(v)}_{\frac{\partial B}{\partial v}} = 0.$$

Hierbei sollen $\alpha(v), \beta(v)$ und $\gamma(v)$ Polynome in v sein. Am besten erscheint es, α und β als Konstanten und γ als linear in v zu wählen, und wie folgt zu definieren:

$$\begin{cases} \alpha(v) = \alpha \\ \beta(v) = \beta \\ \gamma(v) = \gamma_0(1 - v) + \gamma_1(v). \end{cases}$$

Aus der Defintion der Bernsteinpolynome 2.2.2.1 folgt

$$\begin{cases} (1 - v) B^{l-1}_j(v) = \frac{l-j}{l} B^l_j(v) & (0 \leq j \leq l - 1) \\ v B^{l-1}_j(v) = \frac{j+1}{l} B^l_{j+1}(v) & (0 \leq j \leq l - 1), \end{cases}$$

daraus ergeben sich folgende Relationen, welche Bedingungen für die G^1-Stetigkeit definieren

$$\begin{cases} P_{k,j} = P'_{0,j}, \\ \alpha k(P_{k,j} - P_{k-1,j}) + \beta k(P'_{i,j} - P'_{0,j}) + \\ +\gamma_0(l-j)(P_{k,j+1} - P_{k,j}) + \gamma_1(j)(P_{k,j} - P_{k,j-1}) = 0 \qquad (0 \leq j \leq l). \end{cases}$$

Somit haben wir die wichtigsten Eigenschaften und Algortihmen von Bézier-Patches kennengelernt und wenden uns dem nächsten Teil eines Raytracing-Algorithmus zu, der Schnittpunktberechnung und in weiterer Folge der Reflexion des Lichtes.

3 Mehrdimensionales Newton-Verfahren

Die Frage der Schnittpunktberechnung ist ein zentraler Punkt in einem Raytracing-Algorithmus. Entschließt man sich zur Verwendung von Bézier-Patches als Primitiven aufgrund von in Kapitel 2 dargelegten Vorteilen, so steht man vor einer umfangreichen Aufgabe. Tatsächlich gibt es keine geschlossene Form der Lösung dieses Problems, sondern man hat iterative Verfahren zu verwenden, um mögliche Schnitte approximativ zu finden [2]. Grundsätzlich sind hier zwei Kategorien zu nennen:

- mehrdimensionale Newton-Verfahren bzw. Quasi-Newton-Verfahren,
- rekursive Subdivisionsmethoden.

Das rekursive Subdivisionsverfahren wurde bereits in einer grundlegenden Arbeit über Raytracing [20] verwendet. Hierbei wird für den Bézier-Patch ein begrenzendes Volumen wie zum Beispiel eine Sphäre oder ein Würfel definiert, welche den Patch umschließen. Danach wird jeder Strahl auf Schnitte mit diesen begrenzenden Volumina getestet. Tritt so ein Schnitt auf, so wird der Bézier-Patch mittels dem De Casteljau Algorithmus sowie dem Subunterteilungsalgorithmus 2.2.4.2 in zwei Teilpatches geteilt, wobei dieser Schritt in weiterer Folge iteriert wird.
Eine Weiterentwicklung dieser Subdivisionsmethode wurde im Jahre 1990 von Nishita vorgestellt. Hierbei wird der Bézier-Patch in Teile geteilt, die kleiner sind als die Hälfte des gesamten Patches (falls dies aufgrund von Eigenschaften bezüglich der Konvexität möglich ist). Damit erreicht man eine viel raschere Konvergenz, vor allem in Fällen, in denen der Schnittwinkel zwischen Patch und Strahl nahe bei π liegt.
Im Allgemeinen ist jedoch die Konvergenz dieses Algorithmus schlechter als bei Quasi-Newton-Verfahren mit geeigneten Startwerten. Daher soll zur Schnittpunktberechnung ein mehrdimensionales modifiziertes Newton-Verfahren mit Liniensuche verwendet werden.

Im Folgenden werden zu diesem die theoretische Grundlagen dargelegt. Beginnen werde

ich mit einigen Eigenschaften von Iterationsverfahren im Allgemeinen. Daran anschließend werden im zweiten Abschnitt das allgemeine Newton-Verfahren und als dessen Spezialfall das vereinfachte Newton-Verfahren eingeführt. Zudem wird das Konvergenzverhalten dieser Verfahren einer genaueren Betrachtung unterzogen.
Im dritten Abschnitt wenden wir uns dem modifizierten Newton-Verfahren zu und des weiteren sollen dabei auch die exakte Liniensuche sowie die Armijo-Liniensuche besprochen werden.
Mit diesem Wissen wird in Kapitel 4 schließlich ein durchgerechnetes Beispiel des Schnittes einer Geraden mit dem Dreiecks-Bézier-Patch präsentiert werden.

3.1 Iterationsverfahren

Grundlegend gestaltet sich die Aufgabe wie folgt: Wir haben mit C und D zwei Teilmengen des $\mathbb{R}^n$ sowie eine Funktion $f : C \to D$. Zu dieser Funktion suchen wir ein $\xi \in C$ sodass $f(\xi) = 0$. Ist zum Beispiel $C = D = \mathbb{R}^n$, so wird die Abbildung $f : \mathbb{R}^n \to \mathbb{R}^n$ durch n reellen Funktionen $f_i(x_1, ..., x_n)$ mit je n reellen Variablen $x_1, ..., x_n$ beschrieben:

$$f(x) = \begin{bmatrix} f_1(x_1, ..., x_n) \\ \vdots \\ f_n(x_1, ..., x_n) \end{bmatrix}, \qquad x^T = (x_1, ..., x_n).$$

Noch sind die Voraussetzungen jedoch viel zu weit gefasst, um im Allgemeinen numerische Resultate liefern zu können. Wir fordern also zudem:

- Die Funktion f ist stetig, zumindest jedoch stückweise stetig.
- Die Teilmengen C und D des $\mathbb{R}^n$ sollen abgeschlossen und zusammenhängend sein.

Im Fall des Schnitts zwischen Bézier-Patch und Gerade sei nun $B(r, s, t)$ der Dreiecks-Bézier-Patch und die Gerade definiert durch einen Punkt P und einen Richtungsvektor $\vec{v}$. Wir suchen also Werte für r, s und den Parameter τ, sodass für die Funktion

$$f : \mathbb{R}^3 \to \mathbb{R}^3 \text{ mit } \; f(r, s, \tau) = B(r, s, t) - P + \tau \cdot \vec{v}$$

gilt

$$B(r, s, t) - P + \tau \cdot \vec{v} = 0. \tag{3.1}$$

Erwähnt werden soll auch noch, dass Nullstellenprobleme und Minimierungsaufgaben für Funktionen von $\mathbb{R}^n$ eng zusammenhängen. Denn für stetig differenzierbare Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ ist $\xi \in \mathbb{R}^n$ nur dann ein lokales Minimum von f, wenn ξ eine Nullstelle des Gradienten ∇f ist.

Nur in den seltesten Fällen lässt sich jedoch die Nullstelle ξ einer Funktion $f : C \to D$ in endlich vielen Schritten explizit berechnen. Daher ist man auf Näherungsmethoden angewiesen, welche in der Regel Iterationsmethoden folgender Gestalt sind:
Ausgehend vom Startwert x_0 berechnet man eine Folge von Näherungswerten $x_i (i = 1, 2, ...)$ für ξ mittels der Funktion $\Phi : C \to D$, gegeben durch

$$x_{i+1} := \Phi(x_i), \qquad i = 0, 1, 2, ...$$

Φ wird dabei als ITERATIONSFUNKTION bezeichnet.

Wir wollen uns eingehender mit dem Konvergenzverhalten einer solchen durch eine Iterationsvorschrift Φ erzeugten Folge x_i nahe eines Fixpunktes ξ von Φ beschäftigen.
Dazu sei $C = D = \mathbb{R}^n$ und die Konvergenz auf $\mathbb{R}^n$ mittels der Norm $\|\circ\|$ definiert. Dann konvergiert eine Folge von Vektoren $x_i \in \mathbb{R}^n$ gegen einen Vektor ξ falls es zu jedem $\epsilon > 0$ ein N_ϵ gibt mit

$$\|x_l - x\| < \epsilon \qquad \forall\, l \geq N_\epsilon.$$

Die so definierte Konvergenz von Vektoren ist unabhängig von der Wahl der Norm [[18], Satz (4.4.6)].

Definition 3.1.0.1. *Sei ξ ein* FIXPUNKT *der Iterationsfunktion Φ, d.h. es gilt*

$$\Phi(\xi) = \xi,$$

und es gelte für alle Startvektoren x_0 aus einer Umgebung U von ξ und für jede zugehörige Folge $(x_n)_n$ mit $x_n = \Phi(x_{n-1})$ folgende Ungleichung:

$$\|x_{i+1} - \xi\| \leq C \|x_i - \xi\|^p, \tag{3.2}$$

wobei $C < 1$ für $p = 1$. Man bezeichnet dann das durch Φ erzeugte Iterationsverfahren als ein VERFAHREN VON MINDESTENS p-TER ORDNUNG.

Zudem gilt:

Satz 3.1.0.2. *Jedes Verfahren p-ter Ordnung zur Bestimmung des Fixpunktes ξ ist* LOKAL KONVERGENT *in dem Sinne, dass es eine Umgebung $U(\xi)$ von ξ gibt, sodass für alle Startpunkte $x_0 \in U(\xi)$ die durch die Iterationsfunktion Φ erzeugte Folge x_i gegen ξ konvergiert. Kann man $U(\xi)$ auf ganz $\mathbb{R}^n$ ausdehnen, so heißt das Verfahren* GLOBAL KONVERGENT.

Im eindimensionalem Fall kann man die Ordnung eines durch Φ erzeugten Iterationsverfahrens häufig leicht bestimmen, und zwar dann, wenn $\Phi(x)$ in einer Umgebung $U(\xi) \subseteq \mathbb{R}$ genügend oft differenzierbar ist.

Ist etwa $x_i \in U(\xi)$ und gilt $\Phi^{(k)}(\xi) = 0$ für $k = 1, 2, \ldots, p-1$, jedoch $\Phi^{(p)}(\xi) \neq 0$, so folgt

$$x_{i+1} = \Phi(x_i) = \Phi(\xi) + \frac{(x_i - \xi)^p}{p!}\Phi^{(p)}(\xi) + O(\|x_i - \xi\|^{p+1})$$

sowie

$$\lim_{i\to\infty} \frac{x_{i+1} - \xi}{(x_i - \xi)^p} = \frac{\Phi^{(p)}}{p!}.$$

Für $p = 2, 3, \ldots$ liegt dann also ein Verfahren p-ter Ordnung vor. Bei einem Verfahren erster Ordnung gilt zusätzlich zu $p = 1$ auch noch $|\Phi'(\xi)| < 1$.

Beispiel 3.1.0.3. *Betrachten wir nun die Iterationsfolge des in der Folge vorgestellten Newton-Verfahrens, $\Phi(x) = x - \frac{f(x)}{f'(x)}$ auf $\mathbb{R}$. f sei dazu genügend oft stetig differenzierbar auf einer Umgebung der einfachen Nullstelle ξ, d.h. $f'(\xi) \neq 0$.*
Dann gilt

$$\Phi(\xi) = \xi, \quad \text{da } f(\xi) = 0$$

und daher bei $x = \xi$:

$$\Phi'(\xi) = \frac{f(x)f''(x)}{\big(f'(x)\big)^2} = 0,$$
$$\Phi''(\xi) = \frac{f''(\xi)}{f'(x)}.$$

Somit ist das Newton-Verfahren zumindest lokal quadratisch konvergent.

Der folgende allgemeinere Konvergenzsatz zeigt, dass eine durch die Iterationsfunktion $\Phi : C \to D$ gegebene Folge $(x_n)_n$ in jedem Fall gegen ξ konvergiert, falls Φ eine kontrahierende Abbildung ist.

Definition 3.1.0.4. *Eine Funktion $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ heißt* KONTRAHIEREND *in $M \subseteq \mathbb{R}^n$ falls*

für alle $x, y \in M$ gilt

$$\|\Phi(x) - \Phi(y)\| \leq K \|x - y\|, \qquad K < 1. \tag{3.3}$$

Satz 3.1.0.5. *Die Funktion $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ besitze einen Fixpunkt ξ. Sei ferner $B_r(\xi) := \{z| \|z - \xi\| < r\}$ eine Umgebung von ξ, sodass Φ in $B_r(\xi)$ eine kontrahierende Abbildung ist. Dann besitzt die Folge $(x_n)_n$ für alle $x_0 \in B_r(\xi)$ und $x_{i+1} = \Phi(x_i)$ mit $i = 0, 1, \ldots$ folgende Eigenschaften:*

- $x_i \in B_r(\xi)$ *für alle* $i = 0, 1, 2, \ldots,$
- $\|x_i - \xi\| \leq K^i \|x_0 - \xi\|$. *Damit konvergiert die Folge (x_i) mindestens linear gegen ξ.*

Beweis: Aus der Kontraktionseigenschaft folgt, dass beide Aussagen für $i = 0$ richtig sind. Angenommen sie gelten für $j = 0, 1, \ldots, i$, dann folgt

$$\|x_{i+1} - \xi\| = \|\Phi(x_i) - \Phi(\xi)\| \leq \|x_i - \xi\| \leq K^{i+1} \|x_0 - \xi\| \leq r.$$

Letztere Ungleichung folgt aus $K < 1$ und der Definition von $B_r(\xi)$.

□

Mit bedeutend mehr Aufwand kann sogar gezeigt werden, dass die Existenz eines Fixpunktes nicht unbedingt vorausgesetzt werden muss.
Doch wie erhält man nun die oben beschriebenen Iterationsfunktionen Φ? Dies führt uns zum Newton-Verfahren und seinen Varianten.

3.2 Allgemeines Newton-Verfahren

Im Folgenden soll das allgemeine Newton-Verfahren sowie die Variante des vereinfachten Newton-Verfahrens hergeleitet werden. Dies erreicht man über den Begriff des Newton-Verfahrens erster Ordnung.

3.2.1 Newton-Verfahren erster Ordnung

Gegeben sei wieder eine Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ mit einer Nullstelle ξ. Weiters sei f in einer Umgebung $U(\xi)$ hinreichend oft differenzierbar, so entwickelt man f um einen Punkt

$x_0 \in U(\xi)$ in das Taylorpolynom

$$f(\xi) = 0 = f(x_0) + (\xi - x_0)f'(x_0) + \frac{(\xi - x_0)^2}{2!}f''(x_0) + \dots$$
$$+ \frac{(\xi - x_0)^k}{k!}f^{(k)}\big(x_0 + \lambda(\xi - x_0)\big), \quad 0 < \lambda < 1.$$

Durch Vernachlässigen der höheren Potenzen erhält man Gleichungen, denen die Nullstelle ξ bei gegebenem x_0 näherungsweise genügt:

$$0 = f(x_0) + (\overline{\xi} - x_0)f'(x_0)$$

Daraus ergibt sich für Näherungswerte $\overline{\xi}$, dass

$$\overline{\xi} = x_0 - \frac{f(x_0)}{f'(x_0)}$$

gilt. Iterieren wir dieses Verfahren, so erhält man das bekannte Newton-Verfahren in der Form

$$x_{i+1} := \Phi(x_i), \qquad \Phi(x) := x - \frac{f(x)}{f'(x)}. \tag{3.4}$$

Man bezeichnet es noch mit dem Zusatz NEWTON-VERFAHREN ERSTER ORDNUNG, da man nach dem ersten Glied der Taylorentwicklung abgebrochen hat. In den meisten Numerikbüchern findet man ausführliches Material darüber (so z.B. in [12]).

3.2.2 Allgemeines Newton-Verfahren

Durch Linearisierungen lassen sich nun auch Iterationsverfahren zur Lösung von Gleichungssystemen der Form

$$f(x) = \begin{bmatrix} f_1(x_1, \dots, x_n) \\ \vdots \\ f_n(x_1, \dots, x_n) \end{bmatrix} = 0$$

gewinnen. So erhält man für eine Nullstelle x von f die Näherung

$$f(x) \approx f(x_0) + Df(x_0)(x - x_0)$$

mit

$$Df(x_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x^1} & \cdots & \frac{\partial f_1}{\partial x^n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x^1} & \cdots & \frac{\partial f_n}{\partial x^n} \end{pmatrix}, \qquad \text{wobei } x = (x^1, x^2, \ldots, x^n)^T.$$

Die Matrix Df wird oft auch als JACOBI-MATRIX bezeichnet. Ist diese Funktionalmatrix nicht singulär (d.h. die Determinante ist ungleich Null und existiert somit Df^{-1}), so löst man

$$f(x_0) + Df(x_0)(x_1 - x_0) = 0$$

nach x_1 auf und erhält den Startwert zur Berechnung von x_2. Iteriert man dies, so ergibt sich

$$x_{i+1} = x_i - (Df(x_i))^{-1} f(x_i), \qquad i = 0, 1, 2, \ldots.$$

Die Iterationsfunktion Φ des ALLGEMEINEN NEWTON-VERFAHREN ist dann also definiert durch:

$$\Phi(x) := x - (Df(x))^{-1} f(x). \tag{3.5}$$

Die Iteration führt man so lange durch, bis die Norm von $f(x)$ kleiner einer im Vorhinein festgelegten Schranke ϵ ist.
Algorithmisch können wir das so formulieren:

Algorithmus 3.2.2.1 (ALLGEMEINES NEWTON-VERFAHREN)**.**

1. *Wähle Startwert $x_0 = x$ und Toleranz $\epsilon > 0$, bestimme $y = f(x)$.*
2. *Bestimme $\|y\|_2$. Falls $\|y\|_2 < \epsilon$, dann ist x Lösung. Ist $\|y\| \geq \epsilon$, dann weiter.*
3. *Berechne die Jacobi-Matrix $Df(x)$.*
4. *Löse $Df(x)z = y$ nach z.*
5. *Setze $x = x - z$ und bestimme $y = f(x)$. Weiter mit 2.*

Betrachten wir noch den Aufwand des Algorithmus, so stellen wir fest, dass der Hauptaufwand in der Berechnung der Jacobi-Matrix sowie im Lösen des darauffolgenden Gleichungssystems liegt. Die Ermittlung von $Df(x)$ erfordert die Bestimmung von n^2 Zahlen. Zudem benötigt das Lösen des linearen Gleichungssystems noch $O(n^3)$ elementare Rechenoperationen [[18], Kapitel 4.1]. Daher sollte der Algorithmus gerade bei hochdimensionalen Problemen höchstens $O(1)$ Iterationsschritte benötigen, um vernünftige Anwendbarkeit zu garantieren.

3.2.3 Vereinfachtes Newton-Verfahren

Möchte man zudem den Aufwand noch reduzieren, so besteht die Möglichkeit die Jacobi-Matrix nicht in jedem Schritt neu zu berechnen, sondern stattdessen mit der Jacobi-Matrix des Startwertes zu rechnen. Man nennt dies das VEREINFACHTE NEWTON-VERFAHREN. Es besitzt folgende Iterationsfunktion:

$$\Phi(x) := x - (Df(x_0))^{-1} f(x). \tag{3.6}$$

Damit jedoch $Df(x_0)$ eine akzeptable Näherung für die $Df(x_k)$ darstellt, muss der Startwert x_0 schon gut gewählt sein. Zudem ist das Verfahren nur von linearer Konvergenzordnung, wie wir im Folgenden sehen werden.

Satz 3.2.3.1. *Die Funktion f sei auf dem konvexen Gebiet $G \subseteq \mathbb{R}^n$ definiert und genüge folgenden Eigenschaften:*
(A) *f sei stetig differenzierbar und die partiellen Ableitungen der Komponenten $f_1, ..., f_n$ von f seien beschränkt.*
(B) *Im Punkt $x_0 \in G$ sei $Df(x_0)$ invertierbar, sodass die durch die Iterationsfunktion (3.6) erzeugte vereinfachte Newton-Folge existiere.*
(C) *Mit $\|\circ\|_\infty$ bezeichne man die Zeilensummennorm. Dann gelte*

$$\left\|Df(x_0)^{-1}\right\|_\infty \cdot \|Df(x_0) - Df(z)\|_\infty \leq q < 1, \quad \forall z \in G.$$

(D) *Wenigstens eine der beiden abgeschlossenen Kugeln*

$$U := \left\{ x \in \mathbb{R}^n : \|x - x_0\|_\infty \leq \frac{1}{1-q} \|x_1 - x_0\|_\infty \right\},$$

$$V := \left\{ x \in \mathbb{R}^n : \|x - x_1\|_\infty \leq \frac{q}{1-q} \|x_1 - x_0\|_\infty \right\}$$

liege in G.
Unter diesen Voraussetzungen existiert die Iterationsfunktion $\Phi(x)$ (3.6) mit dem Startpunkt x_0 und konvergiert gegen die Lösung ξ von $f(x) = 0$. Dabei liegt ξ in derjenigen der Kugeln U und V, in der G enthalten ist.

Beweis: Der Beweis leitet sich aus dem Mittelwertsatz für vektorwertige Funktionen her. Aus Voraussetzung (C) folgt zudem, dass f eine kontrahierende Abbildung ist. Mit dem Banachschen Fixpunktsatz folgt der Satz [[10], Kapitel 189].

□

In der Praxis weiß man jedoch häufig, dass eine Nullstelle existiert, und hat zudem eine ungefähre Vorstellung von ihrer Lage. Ist nun auch das vereinfachte Newton-Verfahren geeignet, eben diese Nullstellen zu berechnen?
Nur unter gewissen Anforderungen an die Lage des Startwertes x_0 in der Nähe von ξ, wie uns folgender Satz lehrt.

Satz 3.2.3.2. *Die Komponenten $f_1, \ldots, f_n$ der Funktion*

$$f : G \subseteq \mathbb{R}^n \to \mathbb{R}^n, \qquad f(x) = \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{bmatrix} = 0$$

mögen auf G stetige und beschränkte partielle Ableitungen zweiter Ordnung haben. Ist dann ξ eine Nullstelle von f, wobei $Df(\xi)$ invertierbar ist, so konvergiert für jedes x_0 aus einer δ-Umgebung von ξ die durch das Iterationsverfahren (3.6) erzeugte zugehörige vereinfachte Newton-Folge gegen ξ.

Beweis: [10] S.414 ff

□

3.2.4 Konvergenzverhalten des Allgemeinen Newton-Verfahrens

Wir wenden uns der Konvergenz des allgemeinen Newton-Verfahrens zur Lösung des durch $f : \mathbb{R}^n \to \mathbb{R}^n$ gegebenen Gleichungssystems $f(x) = 0$ zu.
Aus der Analysis [[10], Kapitel 164] ist bekannt, dass so eine Funktion im Punkt $x_0 \in \mathbb{R}^n$ differenzierbar ist, wenn eine $n \times n$-Matrix A existiert, sodass

$$\lim_{x \to x_0} \frac{\|f(x) - f(x_0) - A(x - x_0)\|}{\|x - x_0\|} = 0.$$

A entspricht der zuvor definierten Funktionalmatrix $Df(x_0)$. Bevor wir zum zentralen Satz über die Konvergenz kommen, zeigen wir noch ein Lemma, welches wir zum Beweis des Satzes benötigen werden.

Definition 3.2.4.1. *Eine Teilmenge $M \subseteq \mathbb{R}^n$ heißt* KONVEX*, falls zu jedem Paar $(x, y) \in M \times M$, gebildet von Punkten aus M, auch deren Verbindungsstrecke in M liegt. Die Verbindungsstrecke ist definiert durch*

$$\overline{xy} := \{tx + (1-t)y \mid t \in [0,1]\}.$$

Lemma 3.2.4.2. *Existiert $Df(x)$ für alle $x \in M$ aus einem konvexen Gebiet $M \subseteq \mathbb{R}^n$ und gibt es zudem eine Konstante λ mit*

$$\|Df(x) - Df(y)\| \leq \lambda \|x - y\|$$

für alle $x, y \in M$, dann gilt für alle $x, y \in M$ die Abschätzung

$$\|f(x) - f(y) - Df(y)(x-y)\| \leq \frac{\lambda}{2} \|x - y\|^2.$$

Beweis: Für beliebige $x, y \in M$ sei die Funktion $\varphi : [0,1] \to \mathbb{R}^n$ gegeben durch

$$\varphi(t) := f(tx + (1-t)y) = f\big(y + t(x-y)\big).$$

Diese Funktion ist differenzierbar für $t \in [0,1]$, da sie eine Zusammensetzung differenzierbarer Funktionen ist. Nach der Kettenregel gilt

$$\varphi'(t) = Df\big(y + t(x-y)\big)(x-y).$$

Weiters lässt sich wiefolgt abschätzen:

$$\begin{aligned}\|\varphi'(t) - \varphi'(0)\| &= \big\|\big(Df\big(y + t(x-y)\big) - Df(y)\big)(x-y)\big\| \\ &\leq \big\|Df\big(y + t(x-y)\big) - Df(y)\big\| \, \|x - y\| \\ &\leq \lambda t \|x - y\|^2\end{aligned}$$

$$\Delta := f(x) - f(y) - Df(y)(x-y) = \varphi(1) - \varphi(0) - \varphi'(0) = \int_0^1 \big(\varphi'(t) - \varphi'(0)\big) dt$$

und damit mit obiger Abschätzung

$$\|\Delta\| \leq \int_0^1 \Big\|\varphi'(t) - \varphi'(0)\Big\| \, dt = \frac{\lambda}{2} \|x - y\|^2.$$

□

Wir kommen damit zum Satz über die Konvergenz des allgemeinen mehrdimensionalen Newton-Verfahrens, welchen wir nun auch beweisen können.

Satz 3.2.4.3 (Leonid Kantorowitsch)**.** *Es sei eine offene Menge $D \subseteq \mathbb{R}^n$ gegeben, ferner eine konvexe Menge C, für deren Abschluss $\overline{C}$ gilt $\overline{C} \subseteq D$. Zudem betrachten wir eine Funktion $f : D \to \mathbb{R}^n$, die für alle $x \in C$ differenzierbar und an allen $x \in D$ stetig ist.*
Für ein $x_0 \in C$ gebe es positive Konstanten r, α, β, γ und h mit folgenden Eigenschaften:

$$B_r(x_0) := \{x | \, \|x - x_0\| < r\} \subseteq C,$$

$$h := \alpha\beta\gamma/2 < 1,$$

$$r := \alpha/(1-h).$$

Zudem habe $f(x)$ folgende Eigenschaften:

- $\|Df(x) - Df(y)\| \leq \gamma \|x - y\|$ *für alle* $x, y \in C$,
- $Df(x)$ *ist regulär und es gilt* $\|(Df(x))^{-1}\| \leq \beta$ *für alle* $x \in C$,
- $\|(Df(x_0))^{-1} f(x_0)\| \leq \alpha$.

Dann gelten folgende Aussagen:

1. *Wählt man x_0 als Startwert, so ist jedes Element der Iterationsfolge*

$$x_{i+1} := x_i - Df(x_i)^{-1} f(x_i), \qquad i = 0, 1, ...$$

   wohldefiniert und es gilt: $x_i \in B_r(x_0)$ für alle i.

2. $\lim_{k\to\infty} x_k = \xi$ *existiert und es gilt* $\xi \in \overline{B_r(x_0)}$ *und* $f(\xi) = 0$

3. *Für alle $k \geq 0$ gilt*

$$\|x_k - \xi\| \leq \alpha \frac{h^{2^k - 1}}{1 - h^{2^k}}.$$

   Wegen der Voraussetzung $0 < h < 1$ ist das Newton-Verfahren zumindest lokal QUADRATISCH KONVERGENT.

Beweis: Wir zeigen zuerst *1.*
Da $Df(x)$ regulär ist und daher $Df(x)^{-1}$ für $x \in D$ existiert, ist x_{k+1} dann wohldefiniert,

wenn $x_k \in B_r(x_0)$ gilt. Für $k = 0$ und $k = 1$ folgt dies aus der dritten Eigenschaft von $f(x)$. Für $k \leq 2$ zeigt man die Aussage mittels vollständiger Induktion:

$$\begin{aligned}\|x_{k+1} - x_k\| = \left\|-Df(x_k)^{-1} f(x_k)\right\| \leq \beta \|f(x_k)\| = \\ = \beta \|f(x_k) - f(x_{k-1}) - Df(x_{k-1})(x_k - x_{k-1})\|,\end{aligned}$$

letzteres da nach der Definition von x_k gilt

$$f(x_{k-1}) + Df(x_{k-1})(x_k - x_{k-1}) = 0.$$

Wegen 3.2.4.2 ist dann

$$\|x_{k+1} - x_k\| \leq \frac{\beta\gamma}{2} \|x_k - x_{k-1}\|^2$$

und daraus folgt aus der Definition von h sowie Eigenschaft *3* von $f(x)$

$$\|x_{k+1} - x_k\| \leq \alpha h^{s^k - 1}. \tag{3.7}$$

Daraus und der Dreieckunsgleichung folgt nun

$$\begin{aligned}\|x_{k+1} - x_0\| &\leq \|x_{k+1} - x_k\| + \|x_k - x_{k-1}\| + ... + \|x_1 - x_0\| \\ &\leq \alpha(1 + h + h^3 + h^7 + ... + h^{2^k - 1}) < \alpha/(1 - h) = r.\end{aligned} \tag{3.8}$$

Dies impliziert $x_{k+1} \in B_r(x_0)$ und somit Behauptung *1*.
Wir kommen zu Eigenschaft *2*.
Aus Gleichung 3.7 folgt, dass Folge (x_k) eine Cauchyfolge ist, denn für $m \geq n$ hat man

$$\begin{aligned}\|x_{m+1} - x_n\| &\leq \|x_{m+1} - x_m\| + \|x_m - x_{m-1}\| + ... + \|x_{n+1} - x_n\| \\ &\leq \alpha h^{2^n - 1}(1 + h^{2^n} + (h^{2^n})^2 + ...) < \\ &< \frac{\alpha h^{2^n - 1}}{1 - h^{2^n}} < \epsilon\end{aligned} \tag{3.9}$$

für genügend großes n, weil $0 < h < 1$. Also existiert wegen der Vollständigkeit des $\mathbb{R}^n$

$$\lim_{k \to \infty} x_k = \xi \in \overline{B_r(x_0)},$$

weil nach Eigenschaft *1* alle Glieder der Folge in $B_r(x_0)$ liegen. Damit ist Eigenschaft *2* gezeigt.

□

3.3 Modifiziertes Newton-Verfahren

Aus Satz 3.2.4.3 wissen wir, dass das allgemeine Newton-Verfahren nur dann konvergiert, wenn der Starwert x_0 der Iterationsfolge hinreichend nahe an der gesuchten Lösung liegt. Dass diese Tatsache schon im eindimensionalen Fall Schwierigkeiten bereiten kann, zeigt folgendes Beispiel aus [18]:

Beispiel 3.3.0.4. *Sei* $f : \mathbb{R} \to \mathbb{R}$ *gegeben durch* $f(x) := \arctan(x)$. *Die Lösung des Nullstellenproblems* $f(x) = 0$ *ist* $\xi = 0$.
Die Iterationsfolge des allgemeinen Newton-Verfahrens ist jedoch

$$x_{k+1} = x_k - (1 - x_k^2) \arctan(x_k),$$

welche für Startwerte x_0 *mit*

$$\arctan(|x_0|) \geq \frac{2|x_0|}{1 + x_0^2}$$

divergent ist. Denn es ist dann $\lim_{k\to\infty} |x_k| \to \infty$.

Diese nur lokale Konvergenz ist, neben dem großen Rechenaufwand, der Hauptnachteil des Verfahrens, der nun behoben werden soll.
Das heißt, wir wollen ein Iterationsverfahren entwickeln, welches global konvergiert. Wir modifizieren das allgemeine Newton-Verfahren, indem wir SUCHRICHTUNGEN s_k sowie SCHRITTLÄNGEN λ_k definieren und die Folge

$$x_{k+1} := x_k - \lambda_k s_k \tag{3.10}$$

einführen. Setzt man die Suchrichtungen s_k wiefolgt,

$$s_k := Df(x_k)^{-1} f(x_k)$$

so spricht man auch von NEWTON-RICHTUNG. Die Schrittweiten sollen so gewählt werden, dass die Folge $(h(x_k))$ mit $h(x) := f(x)^T f(x)$ streng monoton fallend ist und die x_k gegen ein Minimum von $h(x)$ konvergieren. Die Konstruktion der Iterationsfolge wie in 3.10 nennt man auch LINIENSUCHE.
Im Folgenden sollen aus Resultaten bezüglich der Konvergenz von Minimierungsverfahren eines beliebigen Funktionals $h(x)$ zwei modifizierte Newton-Verfahren hergeleitet werden.

3.3.1 Konvergenz von Minimierungsverfahren

Definition 3.3.1.1. *Sei*

$$D(\gamma, x) := \{s \in \mathbb{R}^n | \, \|s\|_2 = 1 \quad \text{mit } \nabla h(x)s \geq \gamma \, \|\nabla h(x)\|_2\} \tag{3.11}$$

mit $\gamma \in [0, 1]$. *Dabei ist*

$$\nabla h(x) = \Big(\frac{\partial h(x)}{\partial x_1}, ..., \frac{\partial h(x)}{\partial x_n}\Big)$$

der Gradient von $h(x)$. *Die Menge* D *ist die* MENGE DER SINNVOLLEN SUCHRICHTUNGEN, *in dem Sinn, dass die Suchrichtungen einen nicht zu spitzen Winkel mit dem Gradienten* $\nabla h(x)$ *an* x *bilden.*

Nun lehrt uns folgendes Lemma, unter welchen Bedingungen man an der Stelle x die λ sowie s gewinnen kann, sodass für $y = x - \lambda s$ dann gilt, dass $h(y) < h(x)$ ist.

Lemma 3.3.1.2. *Sei* $h : \mathbb{R}^n \to \mathbb{R}$ *eine Funktion, deren Gradient* $\nabla h(x)$ *für alle* $x \in V(\overline{x})$ *aus einer Umgebung* $V(\overline{x})$ *von* $\overline{x}$ *definiert und stetig ist und für den gilt, dass* $\nabla h(\overline{x}) \neq 0$. *Dann gibt es eine Umgebung* $U(\overline{x}) \subseteq V(\overline{x})$ *und* $\gamma, \lambda > 0$, *sodass*

$$h(x - \mu s) \leq h(x) - \frac{\mu\gamma}{4} \|\nabla(\overline{x})\|$$

für alle $x \in U(\overline{x}), s \in D(\gamma, x)$ *sowie* $0 \leq \mu \leq \lambda$.

Beweis: Wegen $\nabla h(x) \neq 0$ und der Stetigkeit von $\nabla h(x)$ auf $V(\overline{x})$ folgt, dass

$$U_1(\overline{x}) := \Big\{x \in V(\overline{x}) | \, \|\nabla h(x) - \nabla h(\overline{x})\| \leq \frac{\gamma}{4} \|\nabla h(\overline{x})\|\Big\}$$

nicht leer und wieder eine Umgebung von $\overline{x}$ ist. Mit der selben Argumentation ist auch

$$U_2(\overline{x}) := \Big\{x \in V(\overline{x}) | D(\gamma, x) \subseteq D\Big(\frac{\gamma}{2}, \overline{x}\Big)\Big\}$$

eine Umgebung von $\overline{x}$. Der Schnitt zweier Umgebungen ist dann wieder eine Umgebung, und da $\mathbb{R}^n$ ein metrischer Raum ist [[9], Kapitel 10], können wir ein $\lambda > 0$ wählen, sodass

$$\overline{S_{2\lambda}(\overline{x})} := \{x | \, \|x - \overline{x}\| \leq 2\lambda\} \subseteq U_1(\overline{x}) \cap U_2(\overline{x}).$$

Mit diesen Definitionen setzen wir nun

$$U(\overline{x}) := \overline{S_\lambda(\overline{x})} = \{x | \, \|x - \overline{x}\| \leq \lambda\}.$$

Damit gilt, dass für $x \in U(\overline{x})$, $0 \leq \mu \leq \lambda$ und $s \in D(\gamma, x)$ ein $\theta \in (0,1)$ existiert mit

$$\begin{aligned} h(x) - h(x - \mu s) &= \mu \nabla h(x - \theta\mu s)s = \\ &= \mu\Big(s\big(\nabla h(x - \theta\mu s) - \nabla h(\overline{x})\big) + \nabla h(\overline{x})s\Big). \end{aligned}$$

Nach Konstruktion gilt für $x \in U(\overline{x})$ und μ, θ, s wie oben, dass auch $x, x - \mu s, x - \theta\mu s \in U_1 \cap U_2$. Damit folgt aber nun

$$\begin{aligned} h(x) - h(x - \mu s) &\geq -\frac{\mu\gamma}{4} \left\|\nabla h(\overline{x})\right\| + \mu \nabla h(\overline{x})s \geq \\ &\geq -\frac{\mu\gamma}{4} \left\|\nabla h(\overline{x})\right\| + \mu\frac{\gamma}{2} \left\|\nabla h(\overline{x})\right\| = \\ &= \frac{\mu\gamma}{4} \left\|\nabla h(\overline{x})\right\|. \end{aligned}$$

□

Daraus wollen wir jetzt unser erstes Verfahren zur Minimierung einer differenzierbaren Funktion $h : \mathbb{R}^n \to \mathbb{R}$ mit Liniensuche entwickeln. Die Idee ist, λ_k so zu wählen, dass h auf dem Strahl $x_k - \mu s_k$ minimiert wird.
Dieses Verfahren bezeichnet man als EXAKTE LINIENSUCHE, als Algorithmus formuliert man dies wie folgt:

Algorithmus 3.3.1.3 (EXAKTE LINIENSUCHE)**.**

1. *Wähle Startwert $x_0 \in \mathbb{R}^n$ und eine Toleranzschranke ϵ.*
   Wähle Zahlenfolgen $(\gamma_k)_k$ und $(\sigma_k)_k$ mit $\inf_k \gamma_k > 0$ sowie $\inf_k \sigma_k > 0$ für $k = 0, 1, \ldots$

2. *Falls $\|\nabla h(x_k)\| > \epsilon$ dann:*
   Wähle einen Vektor $s_k \in D(\gamma_k, x_k)$ (siehe 3.11) und setze

   $$x_{k+1} := x_k - \lambda s_k,$$

   wobei λ_k aus $I_k := [0, \sigma_k \|\nabla h(x_k)\|_2]$ derart bestimmt wird, dass

   $$h(x_{k+1}) = \min_{\mu \in I_k} \{h(x_k - \mu s_k)\}. \tag{3.12}$$

3. *Weiter bis $\|\nabla h(x_{k+1})\| \leq \epsilon$.*

Dass dieses Verfahren konvergiert, ist das Resultat folgenden Satzes.

Satz 3.3.1.4. *Seien $x_0 \in \mathbb{R}^n$ und $h : \mathbb{R}^n \to \mathbb{R}$ eine Funktion derart gewählt, dass Folgendes gilt:*
(A) *Die Menge $K := \{x | h(x) \leq h(x_0)\}$ ist kompakt.*
(B) *h ist auf einer Umgebung von K stetig differenzierbar.*
Dann gilt für jede Folge $(x_k)_k$, die durch den Algorithmus 3.3.1.3 erzeugt wird:

1. *$x \in K$ für alle $k = 0, 1, \ldots,$ daher besitzt $(x_k)_k$ mindestens einen Häufungspunkt $\overline{x}$ in K.*

2. *Jeder Häufungspunkt $\overline{x}$ von $(x_k)_k$ ist stationärer Punkt von h, d.h. $\nabla h(\overline{x}) = 0$.*

Beweis: Zuerst zu Aussage *1.:* Nach Definition der Folge $(x_k)_k$ und der Konstruktion von $h(x_k)$ folgt, dass $h(x_k)$ monoton fallend ist. Daher ist $x_k \in K$ für alle k. Nun ist K aber kompakt, weswegen $(x_k)_k$ mindestens einen Häufungspunkt $\overline{x} \in K$ besitzt [[9], Kap. 38].

Aussage *2.:* Wir beweisen diese Aussage indirekt. Sei als $\overline{x}$ ein Häufungspunkt der Folge $(x_k)_k$, aber kein stationärer Punkt, d.h. $\nabla h(\overline{x}) \neq 0$. Sei zudem o.B.d.A. $\lim_{k\to\infty} x_k = \overline{x}$ und $y := \inf_k \gamma_k > 0, \inf_k \sigma_k > 0$.
Aus Lemma 3.3.1.2 folgt, dass es eine Umgebung $U(\overline{x})$ und eine Zahl $\lambda > 0$ gibt mit

$$h(x - \mu s) \leq h(x) - \frac{\mu\gamma}{4} \left\|\nabla h(\overline{x})\right\|_2 \tag{3.13}$$

für alle $x \in U(\overline{x})$, $s \in D(\gamma, x)$ und $0 \leq \mu \leq \lambda$.
Wegen $\lim_{k\to\infty} x_k = \overline{x}$, der Stetigkeit von $\nabla h(x)$ und der Eigenschaft $\nabla h(\overline{x}) \neq 0$ gibt es ein k_0, sodass für alle $k \geq k_0$ gilt

- $x_k \in U(\overline{x})$,
- $\left\|\nabla h(x_k)\right\|_2 \geq \frac{1}{2} \left\|\nabla h(\overline{x})\right\|_2$.

Definiere nun

$$\Gamma := \min\left\{\lambda, \frac{1}{2}\sigma \left\|\nabla h(x_k)\right\|\right\} \quad \text{sowie} \quad \epsilon := \Gamma\frac{\gamma}{4} \left\|\nabla h(\overline{x})\right\| > 0.$$

Wegen $\sigma_k \geq \sigma$ gilt für alle k die Inklusion $[0, \Gamma] \subseteq [0, \sigma_k \left\|\nabla h(x_k)\right\|]$ und damit nach der Definition der x_{k+1}

$$h(x_{k+1}) \leq \min\left\{h(x_k - \mu s_k) | 0 \leq \mu \leq \Gamma\right\}.$$

Aus 3.13 folgt für alle $k \geq k_0$ wegen $\Gamma \leq \lambda, x_k \in U(\overline{x}), s_k \in D(\gamma_k, x_k) \subseteq D(\gamma, x_k)$ die Abschätzung

$$h(x_{k+1}) \leq h(x_k) - \frac{\Gamma\gamma}{4} \|\nabla h(\overline{x})\| = h(x_k) - \epsilon.$$

Nun ist aber $\epsilon > 0$ und somit $\lim_{k\to\infty} = -\infty$. Dies widerspricht jedoch der Tatsache, dass $h(x_k)$ monoton fallend ist: $h(x_k) \geq h(x_{k+1}) \geq ... \geq h(\overline{x})$. Daher ist h stationär bei $\overline{x}$ mit $\nabla h(\overline{x})$.

□

3.3.2 Armijo-Liniensuche

Die in 3.3.1.3 angegebene Methode ist zwar sehr allgemein, doch in der Praxis schwer anwendbar, da man zur Berechnung von x_k in jedem Rechenschritt das globale Minimum der Funktion

$$\varphi(\mu) := h(x_k - \mu s_k)$$

auf $[0, \|\nabla h(x_k)\|_2]$ bestimmen muss. Dies ist im Allgemeinen jedoch nur näherungsweise und mit großem Rechenaufwand möglich. Wir wollen also die exakte Liniensuche verbessern, indem wir die Minimumsuche durch einen iterativen Suchprozess ersetzen. Der daraus entstehende Algorithmus wird als ARMIJO-LINIENSUCHE bezeichnet.

Algorithmus 3.3.2.1 (ARMIJO-LINIENSUCHE)**.**

1. *Wähle Startwert $x_0 \in \mathbb{R}^n$ und eine Toleranzschranke ϵ.*
   Wähle Zahlenfolgen $(\gamma_k)_k$ und $(\sigma_k)_k$ mit $\inf_k \gamma_k > 0$ sowie $\inf_k \sigma_k > 0$ für $k = 0, 1, \ldots$.

2. *Ist $\|h(x_k)\| > \epsilon$ dann berechnet man x_{k+1} aus x_k wie folgt:*
   Wähle $s_k \in D(\gamma_k, x_k)$, definiere

   $$\rho_k := \sigma_k \|\nabla h(x_k)\|_2, \qquad h_k(\mu) := h(x_k - \mu s_k).$$

   Bestimme die kleinste Zahl $j \geq 0$ so, dass

   $$h_k(\rho_k 2^{-j}) \leq h_k(0) - \rho_k 2^{-j} \frac{\gamma_k}{4} \|\nabla h(x_k)\|_2 . \tag{3.14}$$

   Bestimme zudem $i \in \{0, 1, ..., j\}$ sodass

   $$h_k(\rho_k 2^{-i}) = \min_{1\leq j\leq i} h_k(\rho_k 2^{-j}). \tag{3.15}$$

Setze

$$\lambda_k = \rho_k 2^{-i}, \qquad x_{k+1} = x_k - \lambda_k s_k.$$

3. *Weiter bis* $\|\nabla h(x_{k+1})\| \leq \epsilon$.

Dass es so ein gesuchtes $j \geq 0$ auch tatsächlich gibt, sieht man wie folgt: Ist x_k ein stationärer Punkt, so ist $j = 0$. Ist x_k jedoch nicht stationär, so folgt die Existenz von j aus Lemma 3.3.1.2, und zwar indem man $\overline{x} := x_k$ setzt. Jedenfalls konstruiert man die λ_k und somit auch die Punkte x_{k+1} mit diesem Verfahren in endlich vielen Schritten.
Das Verfahren lässt sich zudem noch beschleunigen, indem man eine obere Schranke M einführt und das Verfahren terminiert, falls $j \geq M$ gilt. Ohne diese obere Schranke gilt für das Verfahren mit Armijo-Liniensuche in Äquivalenz zu 3.3.1.4 folgender Satz zur Konvergenz.

Satz 3.3.2.2. *Unter den Voraussetzungen von Satz 3.3.1.4 gelten auch für jede durch den Algorithmus 3.3.2.1 gelieferte Folge* $(x_k)_k$ *die Aussagen von Satz 3.3.1.4.*

Beweis: Der Beweis ist eine Verfeinerung der Abschätzungen in Satz 3.3.1.4.
Sei also $\overline{x}$ der Häufungspunkt einer durch 3.3.2.1 gelieferten Folge $(x_k)_k$, jedoch kein stationärer Punkt, d.h.

$$\nabla h(\overline{x}) \neq 0.$$

Ebenso sei o.B.d.A. $\lim x_k = \overline{x}$, $\sigma := \inf_k \sigma_k > 0, \gamma := \inf_k y_k > 0$. Wieder folgt aus Lemma 3.3.1.2 wegen $\lim_k x_k = \overline{x}$, der Stetigkeit von $\nabla h(x)$ und $\nabla h(\overline{x}) \neq 0$ die Existenz eines k_0, sodass für alle $k \geq k_0$ gilt

- $x_k \in U(\overline{x})$
- $\|\nabla h(x_k)\| \geq \frac{1}{2} \|\nabla h(\overline{x})\|$.

Nun ist zu zeigen, dass es ein $\epsilon > 0$ gibt, sodass

$$h(x_{k+1}) \leq h(x_k) - \epsilon, \quad \forall k \geq k_0.$$

Dann ist wegen den eben angeführten beiden Eigenschaften und mit $\gamma_k \geq \gamma$

$$\gamma_k \|\nabla h(x_k)\| \geq \frac{\gamma}{2} \|\nabla h(\overline{x}\| , \quad \forall k \geq k_0.$$

Es folgt daher nach Defintion von x_{k+1} und j, dass

$$h(x_{k+1}) \leq h_k(\rho_k 2^{-j}) \leq h(x_k) - \rho_k 2^{-j} \frac{\gamma_k}{4} \|\nabla h(x_k)\| \leq \tag{3.16}$$

$$\leq h(x_k) - \rho_k 2^{-j} \frac{\gamma}{8} \|\nabla h(\overline{x})\| . \tag{3.17}$$

Sei nun $j' \geq 0$ die kleinste Zahl mit

$$h_k(\rho_k 2^{-j'}) \leq h(x_k) - \rho_k 2^{-j'} \frac{\gamma_k}{8} \|\nabla h(\overline{x})\| . \tag{3.18}$$

Wegen 3.17 ist $j' \leq j$ und nach Definition von x_{k+1} ist

$$h(x_{k+1}) \leq h_k(\rho_k 2^{-j'}). \tag{3.19}$$

Nun unterscheiden wir zwei Fälle:

FALL 1: $j' = 0$. Dann folgt aus 3.18 und 3.19 sowie wegen $\rho_k = \sigma_k \|\nabla h(x_k)\| \geq \frac{\sigma}{2} \|\nabla h(\overline{x}\|$, dass für ein von x_k unabhängiges $\epsilon_1 > 0$ gilt

$$\begin{aligned} h(x_{k+1}) &\leq h(x_k) - \rho_k \frac{\gamma}{8} \|\nabla h(\overline{x})\| \leq \\ &\leq h(x_k) - \frac{\sigma\gamma}{16} \|\nabla h(\overline{x})\|^2 = h(x_k) - \epsilon_1. \end{aligned}$$

FALL 2: $j' > 0$. Aus der Minimalitätseigenschaft von j' folgt

$$\begin{aligned} h_k(\rho_k 2^{-(j'-1)}) &> h(x_k) - \rho_k 2^{-(j'-1)} \frac{\gamma}{8} \|\nabla h(\overline{x})\| \\ &\geq h(x_k) - \rho_k 2^{-(j'-1)} \frac{\gamma}{4} \|\nabla h(\overline{x})\| . \end{aligned}$$

Wegen $x_k \in U(\overline{x})$ und $s_k \in D(\gamma_k, x_k) \subseteq D(\gamma, x_k)$ folgt aus Lemma 3.3.1.2

$$\rho_k 2^{-(j'-1)} > \lambda.$$

Rufen wir uns 3.18 sowie 3.19 in Erinnerung, so ergibt dies

$$h(x_{k+1}) \leq h_k(\rho_k 2^{-j'}) \leq h(x_k) - \frac{\lambda\gamma}{16} \|\nabla h(\overline{x})\| = h(x_k) - \epsilon_2$$

mit einem von x_k unabhängigen $\epsilon_2 > 0$. Für $\epsilon = \min(\epsilon_1, \epsilon_2)$ gilt daher $\forall k \geq k_0$

$$h(x_{k+1}) \leq h(x_k) - \epsilon,$$

was ein Widerspruch zu $h(x_k) \geq h(\overline{x})$ für alle k ist. Daraus folgt, dass $\overline{x}$ ein stationärer Punkt von h ist, was den Beweis abschließt.

□

3.3.3 Modifiziertes Newton-Verfahren

Wir wollen nun die Gleichung $f(x) = 0$ mittels dem eben beschriebenen Algorithmus lösen. Dazu setzen wir $h(x) := f(x)^T f(x) = \|f(x)\|_2^2$ und minimieren dieses $h(x)$. Entscheidend ist dabei eine gute Wahl der Suchrichtungen s_k sowie der γ_k und σ_k.
Im modifizierten Newton-Verfahren wählt man als Suchrichtungen s_k gerade die schon unter 3.10 erwähnten normierten Newton-Richtungen:

$$s_k := \frac{d_k}{\|d_k\|_2}, \qquad d_k := Df(x_k)^{-1} f(x_k). \tag{3.20}$$

Natürlich muss hierzu $Df(x)$ regulär sein und $f(x) \neq 0$ gelten. Wir leiten im Folgenden eine Bedingung für $s_k \in D(\gamma_k, x_k)$ her, doch dafür benötigen wir zuvor noch den Begriff der Konditionszahl einer Matrix.

Definition 3.3.3.1. *Sei A eine Matrix und $\|\circ\|$ eine Matrixnorm. Dann definiert man die zu $\|\circ\|$ gehörende* KONDITIONSZAHL *von A als*

$$\kappa(A) := \left\|A^{-1}\right\| \cdot \|A\|.$$

Für singuläre Matrizen setzt man $\kappa(A) := \infty$.

Bezieht man sich auf eine bestimmte Norm, so versieht man auch κ mit dem entsprechenden Index. Im Weiteren soll als Norm immer die euklidische Norm $\|\circ\|_2$ verwendet werden, und daher wird die Konditionszahl mit κ_2 bezeichnet.

Soll nun $s_k \in D(\gamma_k, x_k)$ gelten, so muss

$$\gamma_k \in \left]0, \frac{1}{\kappa_2(Df(x_k))}\right]$$

sein, wie folgende Rechnung zeigt.
Zur Vereinfachung seien $x = x_k$ und $s = s_k$. Wegen $h(x) := f(x)^T f(x)$ erhält man

$$\nabla h(x) = 2f(x)^T Df(x). \tag{3.21}$$

Damit gelten die beiden Abschätzungen

$$\left\|f(x)^T Df(x)\right\| \leq \|Df(x)\| \, \|f(x)\| \,,$$

$$\left\|Df(x)^{-1} f(x)\right\| \leq \left\|Df(x)^{-1}\right\| \, \|f(x)\|$$

und somit

$$\frac{\nabla h(x) s}{\|\nabla h(x)\|} = \frac{f(x)^T Df(x) Df(x)^{-1} f(x)}{\|Df(x)^{-1} f(x)\| \, \|f^T(x) Df(x)\|} \geq \frac{1}{\kappa_2(Df(x))} > 0.$$

Für alle γ_k mit $0 < \gamma_k \leq \frac{1}{\kappa_2(Df(x_k))}$ gilt daher $s_k \in D(\gamma_k, x_k)$. Ist also $Df(x)$ regulär, dann gilt $\nabla h(x) = 0$ genau dann, wenn x Nullstelle von f ist.
Aus diesem Ergebnis können wir nun ein modifiziertes Newton-Verfahren konstruieren und als folgenden Algorithmus formulieren.

Algorithmus 3.3.3.2 (MODIFIZIERTES NEWTON-VERFAHREN)**.**

1. *Wähle Startwert $x_0 \in \mathbb{R}^n$ und Toleranz ϵ.*
   Setze maximale Iterationsanzahlen K und J und $j = 0$, $k = 0$.

2. *Falls $f(x_k) \geq \epsilon$ und Iterationsschritt $k < K$, setze $k = k + 1$ und berechne x_{k+1} aus x_k wiefolgt:*
   Berechne

$$d_k := Df(x_k)^{-1} f(x_k)$$
$$\gamma_k := \frac{1}{\kappa_2(Df(x_k))}$$

   und setze $h_k(\tau) := h(x_k - \tau d_k) = \|f(x_k - \tau d_k)\|^2$ (folgt aus der Definition von h).
   Definiere weiters

$$z_k = \frac{\gamma_k}{4} \|d_k\| \, \|\nabla h(x_k)\| \,.$$

3. *Gilt*

$$h_k(2^{-j}) > h_k(0) - 2^{-j} z_k, \tag{3.22}$$

   dann setze $j = j + 1$ und wiederhole 3.22.

4. *Berechne λ_k, sodass gilt*

$$h(x_k - \lambda_k d_k) = \min_{0 \leq i \leq j} h_k(2^{-i}).$$

Setze $x_{k+1} = x_k - \lambda_k d_k$.

In Analogie zu Satz 3.3.2.2 gilt folgendes Resultat über die Konvergenz des modifizierten Newton-Verfahrens.

Satz 3.3.3.3. *Gegeben sei eine Funktion:* $f : \mathbb{R}^n \to \mathbb{R}^n$ *und ein Punkt* $x_0 \in \mathbb{R}^n$. *Zudem sei wieder* $h(x) = f(x)^T f(x) = \|f(x)\|^2$. *Gelten dann folgende Voraussetzungen:*
(A) *Die Menge* $K := \{x | h(x) \leq h(x_0)\}$ *ist kompakt.*
(B) f *ist auf einer Umgebung von* K *stetig differenzierbar.*
(C) *Für alle* x *existiert* $(Df(x))^{-1}$.
Dann ist die durch Algorithmus 3.3.3.2 definierte Folge $(x_k)_k$ *wohldefiniert und es gilt:*

1. $x_k \in K$ *für alle* $k \in \mathbb{N}$ *und* $(x_k)_k$ *besitzt mindestens einen Häufungspunkt* $\overline{x} \in K$.
2. *Jeder Häufungspunkt* $\overline{x}$ *von* $(x_k)_k$ *ist Nullstelle von* f, *d.h.* $f(\overline{x}) = 0$.

Beweis: Nach Konstruktion ergibt der Algorithmus 3.3.3.2 eine monoton fallende Folge $h(x_k)$ mit

$$h(x_0) \geq h(x_1) \geq \ldots,$$

und somit ist $x_k \in K$. Weiters sei o.B.d.A. $f(x_k) \neq 0$ für alle k. Wegen Voraussetzung (C) sind d_k und γ_k wohldefiniert, wenn nur x_k definiert ist, und es folgt für $s_k := \frac{d_k}{\|d_k\|}$ dass $s_k \in D(\gamma_k, x_k)$.

Aus 3.3.2.2 folgt die Existenz eines $j \geq 0$ mit den in Algorithmus 3.3.3.2 angegebenen Eigenschaften und somit die Wohldefiniertheit von x_{k+1}.
Auch der Rest des Satzes folgt aus 3.3.2.2, denn das modifizierte Newton-Verfahren wird dann mit der Armijo-Liniensuche formal ident, wenn man σ_k durch

$$\sigma_k := \frac{\|d_k\|}{\|\nabla h(x_k)\|}$$

definiert. Wir müssen dann nur mehr zeigen, dass

$$\inf_k \gamma_k > 0, \qquad \inf_k \sigma_k > 0.$$

Nun ist nach Voraussetzungen (B) und (C) $Df(x)^{-1}$ stetig auf der kompakten Menge K und somit auch $\kappa_2(Df(x))$. Damit existiert aber

$$\gamma := \frac{1}{\max_{x \in K} \kappa_2(Df(x))}.$$

Wir nehmen o.B.d.A. an, dass x_k kein stationärer Punkt von h ist. Damit ist wegen der Voraussetzung $f(x_k) \neq 0$ für alle $k \in \mathbb{N}$ wegen $x_k \in K$ gültig, dass

$$\inf_k \gamma_k \geq \gamma > 0.$$

Zur Abschätzung der σ_k haben wir wegen $f(x_k) \neq 0$ und 3.21

$$\begin{aligned} \|d_k\| &= \left\|Df(x_k)^{-1} f(x_k)\right\| \geq \frac{1}{\|Df(x_k)\|} \|f(x_k)\|, \\ \|\nabla h(x_k)\| &\leq 2 \|Df(x_k)\| \|f(x_k)\|. \end{aligned}$$

Nun ist $Df(x)$ für $x \in K$ wegen der Kompaktheit von K nach oben beschränkt und damit gilt

$$\sigma_k \geq \frac{1}{2 \|Df(x_k)\|^2} \geq \sigma > 0.$$

Damit treffen alle Resultate der Sätze 3.3.2.2 bzw. 3.3.1.4 auf die durch den Algorithmus 3.3.3.2 erzeugte Folge $(x_k)_k$ zu. Die Überlegung, dass wegen Voraussetzung (C) und wegen $\nabla h(x) = 0$ genau dann wenn $f(x) = 0$ jeder stationäre Punkt $\overline{x}$ von h auch Nullstelle von f ist, komplettiert den Beweis.

□

Der große Nachteil des modifizierten Newton-Verfahren ist sein Aufwand, denn es erfordert in jedem Schritt die Berechnung von γ_k und somit der Konditionszahl $\kappa_2(Df(x))$. Zudem wird in jedem Iterationsschritt die Jacobi-Matrix $Df(x)$ bestimmt sowie ein lineares Gleichungssystem mit dieser Matrix gelöst. Dies kann jedoch etwas beschleunigt werden.
Wie der eben durchgeführte Beweis gezeigt hat, genügt es, anstelle von $\gamma_k = 1/\kappa_2(Df(x))$ ein beliebige kleinere untere Schranke γ mit $\gamma_k \geq \gamma > 0$ zu wählen. Damit reduziert sich die Abbruchbedingung 3.22 auf

$$h_k(2^{-j}) > h_k(0).$$

Diese Vereinfachung kann sich jedoch nachteilig auf die Konvergenz auswirken - ein Konvergenzbeweis mit obigen Methoden ist mit der Bedingung $\gamma_k > 0$ jedenfalls nicht zu erbringen.
Gelegentlich wird dieses Vereinfachung auch als GEDÄMPFTES NEWTON-VERFAHREN bezeichnet.
Genauer betrachtet werden soll das Verhalten des Verfahrens in einer hinreichend kleinen

Umgebung einer Nullstelle. Dort hat das Verfahren 3.3.3.2 nämlich die angenehme Eigenschaft, dass automatisch $\lambda_k = 1$ gewählt wird, und somit zum gewöhnlichen Newton-Verfahren 3.2.2.1 mit lokal quadratischer Konvergenz transferiert.

Satz 3.3.3.4. *Gelten dieselben Voraussetzungen wie in Satz 3.3.3.3, dann existiert eine Umgebung $U(\overline{x})$ eines Häufungspunktes $\overline{x}$, in der das modifizierte Newton-Verfahren quadratisch konvergiert.*

Beweis: Wegen $\lim_{k\to\infty} x_k = \overline{x}$ und $f(\overline{x}) = 0$ gibt es eine Umgebung $U_1(\overline{x})$ von $\overline{x}$, in der für alle Folgenglieder der vom allgemeinen Newton-Verfahren erzeugten Folge $(z_k)_k$ folgende Abschätzungen gelten:

$$\|z_{k+1} - \overline{x}\| \leq \|z_k - \overline{x}\|^2 \tag{3.23}$$

$$32a^2c^2 \|z_k - \overline{x}\| \leq 1, \qquad c := \kappa_2(Df(x)). \tag{3.24}$$

Zudem gilt

$$\|x - \overline{x}\| \left\|Df(\overline{x})^{-1}\right\|^{-1} \leq \|Df(\overline{x})(x - \overline{x})\| \leq \|Df(\overline{x})\| \|x - \overline{x}\|$$

und für die Taylorentwicklung von f um $\overline{x}$

$$f(x) = Df(\overline{x})(x - \overline{x}) + o(\|x - \overline{x}\|).$$

Daraus folgt, dass es eine weitere Umgebung $U_2(\overline{x})$ gibt mit

$$\frac{1}{4} \left\|Df(\overline{x})^{-1}\right\|^{-2} \|x - \overline{x}\|^2 \leq h(x) \leq 4 \|Df(\overline{x})\|^2 \|x - \overline{x}\|^2$$

für alle $x \in U_2(\overline{x})$. Wählt man nun eine Umgebung $U(\overline{x}) \subset U_1(\overline{x}) \cap U_2(\overline{x})$ und ein k_0, sodass für alle $k \geq k_0$

$$x_k \in U(\overline{x}) := \{x | \|x - \overline{x}\| < r\}$$

gilt, so erhält man mit $x_{k+1} = x_k - Df(x_k)^{-1} f(x_k)$ und unter Verwendung von 3.23 und 3.24 folgende Ungleichungen:

$$\begin{aligned} h(x_{k+1}) &\leq 4 \|Df(\overline{x})\|^2 \|x_{k+1} - \overline{x}\|^2 \leq 16a^2c^2 \|x_k - \overline{x}\|^2 h(x_k) \leq \\ &\leq h(x_k)(1 - \frac{1}{2}) = \frac{1}{2} h(x_k). \end{aligned}$$

Aus der Definition des modifizierten Newton-Verfahrens mit $\lambda_k = 1$ gilt dann

$$\gamma_k \|d_k\| \|\nabla h(x_k)\| \leq 2\gamma_k \left\|Df(x_k)^{-1}\right\| \|Df(x_k)\| h(x_k) = 2h(x_k)$$

und somit die Abschätzung

$$h(x_{k+1}) \leq \frac{1}{2}h(x_k) \leq h(x_k) - \frac{\gamma_k}{4} \|d_k\| \|\nabla h(x_k)\| .$$

Daraus folgt, dass es ein k_0 gibt, sodass der Algorithmus 3.3.3.2 für alle $k \geq k_0$ $j = 0$ und somit $\lambda_k = 1$ wählt. Erreicht also die durch das Verfahren erzeugte Folge die Umgebung $U(\overline{x})$, so ist das Verfahren mit dem gewöhnlichen Newton-Verfahren ident und somit lokal quadratisch konvergent.

□

Der Aufwand des gedämpften Newton-Verfahrens ist noch immer sehr hoch, doch falls die Funktionalmatrix bekannt ist, so lässt es sich nun durchführen. Dies wäre zum Beispiel nicht der Fall, wenn f Lösung einer Differentialgleichung ist und das Lösungverfahren nur Funktionswerte $f(x)$ liefert.

4 Raytracing-Algorithmus mit Bézier-Patch

In diesem Kapitel wenden wir uns nun dem eigentlichen Ziel der Arbeit zu, nämlich der Durchführung eines Raytracing-Algorithmus mit Bézier-Patch als Primitiv. Zu diesem Grunde sollen zuallererst einige im Rechengang implizit angenommene Grundlagen der Optik angeführt werden. Wir werden uns hier auf den Fall der Strahlenoptik beschränken. Dabei soll das Fermatsche Prinzip formuliert werden und daraus das Brechungsgesetz sowie das Reflexionsgesetz gefolgert werden.
Danach werden die einzelnen Teilbereiche des in MATLAB R2009a implementierten Algorithmus besprochen. Dabei wird der Wert auf die mathematischen Hintergründe der einzelnen Algorithmen gelegt und nicht so sehr auf die exakten Implemenationen - sie sind in Kapitel 5 zu finden.
Ein durchgerechnetes Beispiel des Schnittes von Gerade (in diesem Fall soll diese als Approximation eines Lichtstrahl verstanden werden) und Dreiecks-Bézier-Patch präsentiert. In diesem Beispiel werden zudem noch als Ergebnis die Schnittpunkte mit der Box nach der Reflexion angeführt. Dies war die zentrale Aufgabenstellung: Die Reflexion an einem aus Bézier-Patches aufgebauten Spiegel zu bestimmen.

4.1 Strahlenoptik

Wir betrachten unsere Lichtquelle, von der aus der Raum beleuchtet wird, unter dem Gesichtspunkt der STRAHLENOPTIK (oft auch als GEOMETRISCHE OPTIK bezeichnet), da sich diese Betrachtungsweise im Gegensatz zur Wellenoptik als für die Anwendungen von großem Nutzen herausstellt. Doch wie unterscheiden sich diese beiden Betrachtungsweisen der Lichtfortplanzung?

Unter dem Begriff der Strahlenoptik versteht man eine Näherung der Optik, in welcher

die Welleneigenschaften des Lichts vernachlässigt werden. Dies ist im Allgemeinen berechtigt, da die mit dem Licht in Wechselwirkung tretenden Primitiven, wie Linsen und Spiegel, sehr groß im Verhältnis zur Wellenlänge des Lichts sind. So liegt der Bereich der Wellenlänge bei sichtbarem Licht zwischen rund 410 nm und 750 nm, was in der Regel viel kleiner ist als die Dimension der meisten optischen Bauteile.
Wichtig für die Modellierung des Algorithmus ist, dass man sich in der Strahlenoptik das Licht aus LICHTSTRAHLEN aufgebaut vorstellt [7]. Im Modell werden diese Lichtstrahlen als von der Lichtquelle oder dem reflektierenden Spiegel ausgehende Geraden verwendet.

Richtungsänderungen dieser Strahlen treten nur in zwei Fällen auf: Zum einen bei der Reflexion an spiegelnden Oberflächen, zum anderen gehen die Strahlen bei Dichteänderungen des Mediums in neue Richtungen über, man spricht dabei von Brechung.
Zudem wird in der Strahlenoptik noch Folgendes angenommen:

- Benachbarte Lichtstrahlen beeinflussen einander nicht, insbesondere ist es für die Geometrie der Lichtfortpflanzung belanglos, ob letztere im Inneren oder am Rand eines Bündels von Lichtstrahlen erfolgt. Dies bedeutet, dass die Strahlenoptik den Begriff der Beugung nicht kennt.

- Kreuzungspunkte von abgelenkten Lichtstrahlen sind bekannt und werden Bildpunkte genannt.

- Ebenso wie die Beugung lassen sich mit der geometrischen Optik die Begriffe der Interferenz (Überlagerung von Wellen) und der Absorption (die Aufnahme von Licht durch einen Stoff) nicht darstellen.

Zudem sei noch erwähnt, dass sich die Strahlenoptik aus der Wellenoptik als Grenzwert ergibt, wenn man nur die Wellenlänge λ gegen 0 gehen lässt.

Die im Abschnitt 1.3 angeführten Eigenschaften zur perfekten (spiegelnden) Reflexion 1, perfekten diffusen Reflexion 2, perfekten Transmission 3 sowie zur totalen inneren Reflexion 4 beziehen sich auf den Fall der Strahlenoptik.
Als wichtigste Grundgesetze, die alles regeln, hat die Strahlenoptik das Snellsche Brechungsgesetz und das Reflexionsgesetz [7]. Diese beiden Gesetze lassen sich aus einem allgemeinen Prinzip von großer Bedeutung herleiten - dem FERMATSCHEN PRINZIP.

Satz 4.1.0.5 (FERMATSCHES PRINZIP)**.** *Der Weg, den das Licht nimmt, um von einem zu einem anderen Punkt zu gelangen, ist stets so, dass die benötigte Zeit minimal ist.*

Bei zwei- oder mehrfach stetig differenzierbaren Funktion ist dazu folgende Formulierung äquivalent:

Der Weg, den das Licht nimmt, um von einem zu einem anderen Punkt zu gelangen, ist stets so, dass die Zeit, die das Licht benötigt invariant gegen kleine Änderungen des Weges ist.

Dass dieses Prinzip gilt, zeigt uns folgende Überlegung:

Wir betrachten zwei Punkte P und Q, die durch eine Grenzfläche G getrennt sind. Über welchen auf dieser Grenzfläche liegenden Punkt A verläuft dann der kürzeste Weg? Dies muss wegen der verschiedenen Lichtgeschwindigkeiten nicht der direkte Weg. So habe die Strahlung die Schwingungszahl ν Hertz und c sei die Lichtgeschwindigkeit im Vakuum (diese entspricht 299792,5 km/sec), dann ist die Wellenlänge im Vakuum $\lambda_0 = \frac{c}{\nu}$. Die Wellenzahl je mm Wegstrecke im Vakuum beträgt dann

$$z_0 = \frac{1}{\lambda_0} = \frac{\nu}{c}.$$

Im Medium Luft ist diese natürlich kleiner, wir bezeichnen sie mit $z_L = \frac{1}{\lambda_L} = \frac{n_L}{\lambda_0}$. Noch kleiner ist sie in Glas, bezeichnet durch $z_G = \frac{n_G}{\lambda_0}$. Somit gilt

$$z_G = z_L * \frac{n_G}{n_L},$$

wobei $n = \frac{n_G}{n_L}$ die Brechungszahl von Glas gegen Luft ist. Die Wellenzahl je mm Wegstrecke ist dann $z_G = n * z_L$. Somit entfallen auf die Wegstrecke s_L in der Luft $z_L * s_L$ Wellen, auf jene im Glas $z_G * s_G$ Wellen. Daher gilt

$$\begin{aligned} z_L * s_L &= z_G * s_G \\ z_L * s_L &= n z_L * s_G \\ s_L &= n * s_G. \end{aligned}$$

Als OPTISCHEN LICHTWEG in Glas bezogen auf Luft bezeichnet man nun $n * s_G$. In einem anderen Stoff mit der Brechungszahl n' gegen Luft ist der einer Wegstrecke s'_G entsprechende optische Lichtweg $n' * s'_G$. Daher ist der optische Lichtweg von P über A nach Q

$$-ns + n's',$$

wobei das Minuszeichen bei s der Konvention der geometrischen Optik entspricht [7]. Nun vergleichen wir hiermit den optischen Lichtweg über einen dicht benachbarten Punkt B (dg von A entfernt). Dann ist hier der Lichtweg $-n(s+ds)+n'(s'+ds')$. Er unterscheidet sich also von dem Weg über A durch $-nds+n'ds'$. In den beiden Dreiecken PAB und QAB gilt dann

$$-s-ds=\sqrt{(-s)^2+dg^2-2sdg\sin\epsilon}\approx -s+dg\sin\epsilon-\frac{dg^2}{2s}(1+\sin^2\epsilon),$$
$$s'+ds'=\sqrt{s'^2+dg^2-2s'dg\sin\epsilon'}\approx s'-dg\sin\epsilon'+\frac{dg^2}{2s'}(1+\sin^2\epsilon').$$

Und weiter

$$-nds\approx dgn\sin\epsilon-\frac{n(1+sin^2\epsilon)}{2s}dg^2,$$
$$+n'ds'\approx -dgn'\sin\epsilon'+\frac{n'(1+\sin^2\epsilon')}{2s'}dg^2,$$

und mit $n\sin\epsilon=n'\sin\epsilon'$ gilt:

$$-nds+n'ds'=\frac{dg^2}{2}\left(\frac{n'(1+sin^2\epsilon')}{s'}-\frac{n(1+sin^2\epsilon)}{s}\right)>0. \tag{4.1}$$

Der optische Lichtweg über einen jeden anderen Punkt als A ist also um Größen zweiter Ordnung länger als der tatsächliche optische Lichtweg über A. Somit folgt das Fermatsche Prinzip.

Für eine alternative Formulierung sei wieder wie zuvor *c* die Lichtgeschwindigkeit im Vakuum und *v* die Lichtgeschwindigkeit im Medium. Dann durchläuft das Licht in einem Medium mit der Brechzahl $n(x)=\frac{c}{v}$ zwischen den Punkten $x(t_1)$ und $x(t_2)$ von allen möglichen Bahnen $X:t\to x(t)$ jene mit der geringsten Laufzeit, nämlich jene, sodass

$$t(X)=\frac{1}{c}\int_{t_1}^{t_2}n(x(t))\mathrm{d}t$$

minimal ist.
Nun soll aus dem Fermatschen Prinzip zuerst das in 1.3 bereits genannte Reflexionsgesetz hergeleitet werden: Als Ausgangspunkt betrachten wir zwei Punkte $P_1=(0|a)$ und $P_2=(e|d)$, wobei die x-Koordinate e von P_2 in die Teilstrecken x und b zerlegt werden kann, wobei x der Reflexionspunkt sei. Von P_1 wird nun ein Lichtstrahl ausgesandt und soll nach Reflexion an der x-Achse P_2 erreichen.

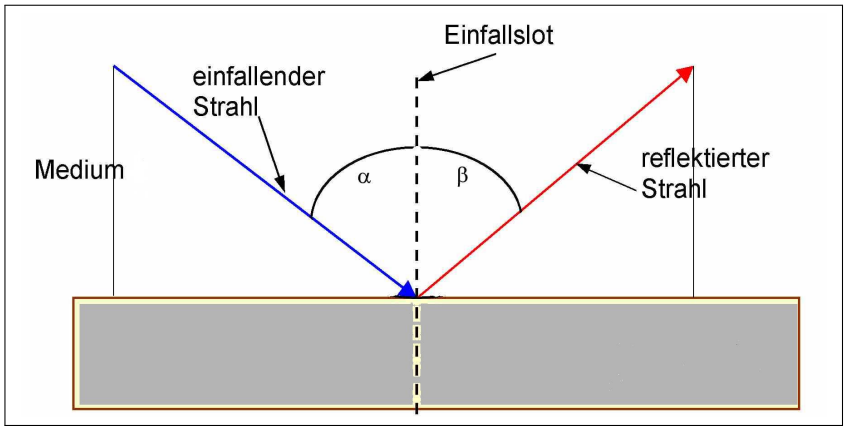

Abbildung 4.1: Das Reflexionsgesetz in der Strahlenoptik

Zudem soll noch vorausgesetzt werden, dass beide Punkte sich in einem homogenen Medium befinden und somit die Lichtgeschwindigkeit unverändert bleibt. Die Länge des Weges $s(x) = \overline{P_1P_2}$ ist dann

$$s(x) = \sqrt{x^2 + a^2} + \sqrt{(x-b)^2 + d^2}. \tag{4.2}$$

Wollen wir nun den Wert x_0 bestimmen, für den 4.2 minimal wird, so bilden wird die erste Ableitung

$$s'(x) = \frac{x}{\sqrt{x^2 + a^2}} + \frac{x-b}{\sqrt{(x-b)^2 + d^2}}. \tag{4.3}$$

und setzen $s'(x_0) = 0$. Somit gilt

$$\frac{x_0}{\sqrt{x_0^2 + a^2}} = \frac{b - x_0}{\sqrt{(x_0 - b)^2 + d^2}}. \tag{4.4}$$

Daraus folgt aber nach kurzer Überlegung, dass $\sin\alpha = \sin\beta$ und somit das bekannte REFLEXIONSGESETZ.

Ebenso soll nun das SNELLSCHE BRECHUNGSGESETZ aus dem Fermatschen Prinzip 4.1.0.5 hergeleitet werden. Dazu betrachten wir die modifzierte Situation, in welcher der Punkt P_2 unterhalb der x-Achse liegt und somit $P_2 = (e| - d)$ ist. Oberhalb und unterhalb befinden sich nun zwei verschiedene homogene Medien mit Lichtgeschwindigkeiten c_1 bzw. c_2, wobei bei Brechung zum Lot $c_1 > c_2$ sein muss und bei Brechung vom Lot $c_2 > c_1$. Wie zuvor hängt der zurückgelegte Weg von der Stelle x_0 ab, an der der Übergang von einem Medium zum anderen erfolgt.
Doch da wir nun unterschiedliche Lichtgeschwindigkeiten haben, betrachten wir die für

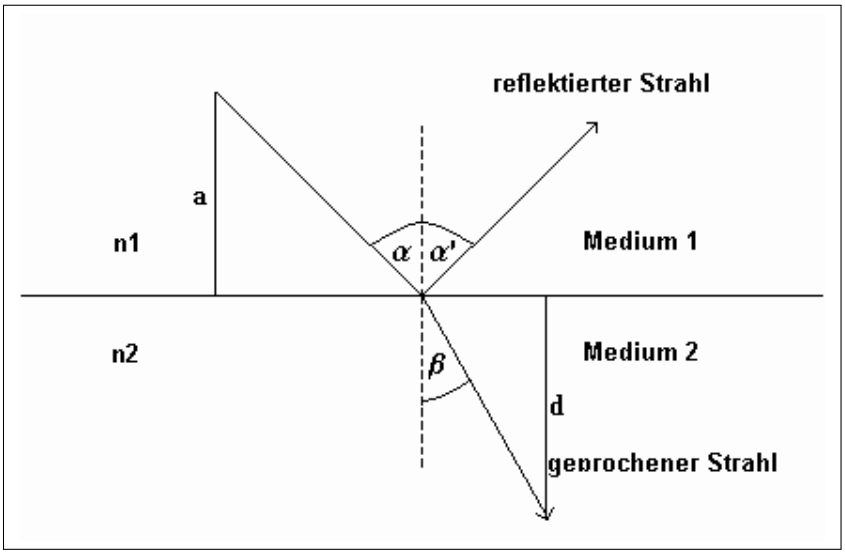

Abbildung 4.2: Illustration des Snellschen Brechungsgesetz

den Weg benötigte Zeit:

$$t(x) = \frac{\sqrt{x^2 + a^2}}{c_1} + \frac{\sqrt{(x-b)^2 + d^2}}{c_2}. \tag{4.5}$$

Wir bilden wieder die Ableitung und erhalten

$$t'(x) = \frac{x}{c_1\sqrt{x^2 + a^2}} + \frac{x-b}{c_2\sqrt{(x-b)^2 + d^2}}. \tag{4.6}$$

Für den optimalen Punkt x_0 gilt dann

$$\frac{x_0}{c_1\sqrt{x^2 + a^2}} = \frac{b - x_0}{c_2\sqrt{(x_0-b)^2 + d^2}} \tag{4.7}$$

und damit

$$\frac{\sin\alpha}{\sin\beta} = \frac{c_1}{c_2} = \frac{\eta_2}{\eta_1}.$$

Hierbei sind η_1 bzw. η_2 die Brechungszahlen der jeweiligen Medien. Dieses Resultat entspricht genau dem schon in Abschnitt 1.3 präsentierten Ergebnis.
Mit diesen Voraussetzungen aus der Strahlenoptik wenden wir uns nun der Implementation des Raytracing-Algorithmus zu.

4.2 Raytracing-Algorithmus

Zu Beginn steht der Aufruf einer Kopffunktion über folgende Eingabe:

```
function [MaSp,SpX,SpZ,SpZ1,SpY,SpY1] = raytracer.
```

Daran anschließend wird die Eingabe der Punkte des Kontrollpolygons erfragt. Aus den Eigenschaften für die Bernsteinpolynome 2.2.2.3, aus denen die Dreiecks-Bézier-Patches aufgebaut sind, folgt, dass für den dreidimensionalen Fall das Kontrollpolygon aus genau zehn Punkten gebildet wird.
Zudem wird noch die Position der Lampe, welche den Raum (o.B.d.A. wird hierfür ein Würfel der Dimension $[10 \times 10 \times 10] \subseteq \mathbb{R}^3$ angenommen) beleuchtet, als Punkt A eingegeben. Über k wird mit $n = 2^k - 1$ die Anzahl n der in den Raum gehende Strahlen abgefragt.
Für jede dieser Eingaben gibt es bei Defaulteingabe 0 vordefinierte Einstellungen. Alle anderen in den Teilalgorithmen verwendeten Argumente werden im Programm gewonnen und an nachfolgende Funktionen übergeben.
Als Ausgabeargumente treten folgende Resultate auf:

- `MaSp` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit dem Bézier-Patch enthält.
- `SpX` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit der y-z-Ebene nach der Reflexion enthält.
- `SpZ` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit der x-y-Ebene nach der Reflexion enthält.
- `SpZ1` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit der Ebene $z = 10$ nach der Reflexion enthält.
- `SpY` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit der x-z-Ebene nach der Reflexion enthält.
- `SpY1` ist eine Matrix, welche die Schnittpunkte der Lichtstrahlen mit der Ebene $y = 10$ nach der Reflexion enthält.

Im ersten Teilalgorithmus

```
function[t,XS,YS,ZS,SP,X]=konvhull
```

werden zu Beginn die die Lichtstrahlen in diskretisierter Form darstellenden Geraden

$$g_i : X = A + \tau \vec{v_i}$$

erzeugt. Um die Richtungsvektoren der Geraden zu bestimmen, legt man um den Augpunkt A eine Kugel in Polarkoordinaten

$$\begin{aligned} x &= r \cdot \cos\phi \cdot \cos\theta \\ y &= r \cdot \cos\phi \cdot \sin\theta \qquad \left(-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2} \wedge -\pi \leq \theta \leq \pi\right) \\ z &= r \cdot \sin\phi \end{aligned}$$

und erhält zu jedem $(x, y, z)^T$ durch die Differenz zu A einen Richtungsvektor $\vec{v_i}$ der Geraden g_i. Als Schnittgerade kommen nun nur jene g_i infrage, welche die konvexe Hülle des Dreiecks-Bézier-Patches schneiden, wie aus Proposition 2.2.3.2 folgt. Deswegen erstellt die Funktion `konvhull` die konvexe Hülle des Dreiecks-Bézier-Patch

$$B(r,s,t) = \sum_{i+j+k=3} P_{i,j,k} B^3_{i,j,k}(r,s,t) = \begin{bmatrix} b_1(r,s,t) \\ b_2(r,s,t) \\ b_3(r,s,t) \end{bmatrix} \tag{4.8}$$

und testet alle g_i auf Schnitte gegen die Ebenen, in denen die Dreiecke liegen, welche die konvexe Hülle aufbauen. Diese Ebenen ϵ_l werden aufgespannt durch die zwei Seitenvektoren $\vec{u_l}$ und $\vec{w_l}$, die durch je zwei Punkte des Kontrollpolygons erzeugt werden, und in Parameterdarstellung als

$$\epsilon_l : X = P_{i,j,k} + t_l \cdot \vec{u_l} + s_l \cdot \vec{w_l}, \qquad i + j + k = 3$$

geschrieben werden können. Schneidet g_i nun genau das in ϵ_l enthaltene Dreieck der konvexen Hülle, so gilt, dass

$$0 \leq t_l + s_l \leq 1, \quad 0 \leq s_l \leq 1, \quad 0 \leq t_l \leq 1.$$

Diese Schnittpunkte liefert die Funktion `konvhull` als Ergebnis und sie sind die Eingabeargumente des nächsten Teilalgorithmus

```
function [ARI,XNE,te,c] = arith(A,XS,YS,ZS,SP).
```

Hier werden aus den Schnittpunkten der Lichtgeraden g_i mit der konvexen Hülle des Bézier-Patches die arithmetischen Mittel gebildet. Sie sollen in weiterer Folge als Startwerte des modifizierten Newton-Verfahrens fungieren.
Um nun Schnittpunkte $SP_i = (r, s, t)^T$ mit dem Bézier-Patch bestimmen zu können, sodass

$$\|B(r, s, t) - A - \tau \cdot \vec{v}\|_2 < \epsilon \tag{4.9}$$

für ein fixes $\epsilon \in \mathbb{R}$ ist, benötigt man eine Umrechnung des Startwertes in baryzentrische Koordinaten. Dies erreicht die Funktion

```
function[alpha,beta,gamma] = baryzent
```

mithilfe der in Abschnitt 2.2.1.1 vorgestellten Formeln.
Als letzte Vorarbeit vor der eigentlichen Schnittpunktberechnung wird mittels der Funktion

```
function[f1dr,f1ds,f1dv,f2dr,f2ds,f2dv,f3dr,f3ds,f3dv,f1,f2,f3]=jacobi
```

die Jacobi-Matrix der zu minimierenden Funktion

$$f(r, s, \tau) = \begin{bmatrix} b_1(r, s, t) \\ b_2(r, s, t) \\ b_3(r, s, t) \end{bmatrix} - g(\tau) \tag{4.10}$$

erstellt. Da $r + s + t = 1$ gilt, setzen wir $t = 1 - r - s$ und haben wegen 4.8 dann nach etwas Rechnung

$$\begin{aligned} f(r, s, \tau) =& r^3(P_{300} - P_{003} + P_{102} - P_{201}) + s^3(P_{030} - P_{003} + P_{012} - P_{021}) + \\ &+ s^2(3P_{003} - 2P_{012} + P_{021}) + r^2(3P_{003} - 2P_{102} + P_{201}) + \\ &+ r^2 s(-3P_{003} + P_{012} + 2P_{102} - P_{201} + P_{210} - P_{111}) + \\ &+ rs^2(-3P_{003} - P_{021} + 2P_{012} + P_{102} + P_{120} - P_{111}) + \\ &+ rs(6P_{003} - 2P_{012} - 2P_{102} + P_{111}) + r(-3P_{003} + P_{102}) + \\ &+ s(-P_{003} + P_{012}) + P_{003} - A - \tau \cdot \vec{v}. \end{aligned}$$

Als f_1, f_2 und f_3 wählen wir jeweils die x, y, z-Koordinaten dieser vektorwertigen Funktion.

Die Jacobi-Matrix dieser Funktion ist dann von der Form

$$J_f(x) = \frac{\partial f_i}{\partial x_j} = \begin{pmatrix} \frac{\partial f_1}{\partial r} & \frac{\partial f_1}{\partial s} & \frac{\partial f_1}{\partial \tau} \\ \frac{\partial f_2}{\partial r} & \frac{\partial f_2}{\partial s} & \frac{\partial f_2}{\partial \tau} \\ \frac{\partial f_3}{\partial r} & \frac{\partial f_3}{\partial s} & \frac{\partial f_3}{\partial \tau} \end{pmatrix}. \tag{4.11}$$

In Abhängigkeit der Punkte $P_{i,j,k}$ des Kontrollpolygons wird diese Matrix von der Funktion `jacobi` bestimmt.
Das Herzstück des Algorithmus, das modifizerte Newton-Verfahren 3.3.3.2, wird nun durch Aufruf der Funktion

```
function[x] = linesearch(...)
```

gestartet. Die Iteration wird solange durchgeführt, bis die Ungleichung (4.9) für $\epsilon = 10^{-10}$ erfüllt ist. Zudem wird die Anzahl der Iterationsschritte begrenzt, um divergente Liniensuchen zu terminieren. Dies ist auch deswegen gerechtfertigt, da sich zeigt, dass im Großteil der Iterationen die Konvergenz rasch erfolgt.
Als weitere Maßnahmen zur Reduktion der Rechenzeit wird die Konditionszahl der Jacobi-Matrix nur im ersten Iterationsschritt berechnet und diese in den darauffolgenden Schleifen als Näherung verwendet. Als Ausgabe liefert diese Funktion nun die Matrix der Schnittpunkte in baryzentrischen Koordinaten.
Nach Bestimmung der Schnittpunkte tritt die Reflexion in den Fokus des Interesses. Deswegen wird als nächstes der Normalvektor an den Bézier-Patch im Schnittpunkt bestimmt. Dies geschieht über

```
function [N,Rx,Ry,Rz,MaSp] = normalv.
```

Als erstes definiere ich Vektoren $\vec{b_1}$ bzw. $\vec{b_2}$ mit

$$\vec{b_r} = \begin{pmatrix} \frac{\partial f_1}{\partial r} \\ \frac{\partial f_2}{\partial r} \\ \frac{\partial f_3}{\partial r} \end{pmatrix} \quad \text{bzw.} \quad \vec{b_s} = \begin{pmatrix} \frac{\partial f_1}{\partial s} \\ \frac{\partial f_2}{\partial s} \\ \frac{\partial f_3}{\partial s} \end{pmatrix},$$

wobei ich die in `jacobi` berechneten partiellen Ableitungen verwenden kann, da der Teil der Funktion f, der die Gerade enthält, bei Differentiation nach r sowie s wegfällt. Damit

lässt sich nun zu jedem Schnittpunkt auf $B(r,s,t)$ der Normalvektor über

$$\vec{n} = \frac{B_r(r,s) \times B_s(r,s)}{|B_r(r,s) \times B_s(r,s)|} = \frac{\vec{b_r} \times \vec{b_s}}{|\vec{b_r} \times \vec{b_s}|} \tag{4.12}$$

berechnen. An diese Berechnung anschließend wird die Funktion

```
function[ref]=rot pi(N,Rx,Ry,Rz,c)
```

aufgerufen, in der der Richtungsvektor der Geraden nach der Reflexion bestimmt wird. Dies wird erreicht, indem eine Rotation durchgeführt wird, in welcher der in der Funktion `normalv` bestimmte Normalvektor als Rotationsachse verwendet wird und der Rotationswinkel $\alpha = \pi$ ist.
Sei dieser Normalvektor nun $\vec{n} = (n_1, n_2, n_3)^T$, dann ist die in Abschnitt 1.3 vorgestellte Rotationsmatrix ROT von der Form

$$\text{ROT} := \begin{pmatrix} -1+2n_1^2 & 2n_1n_2 & 2n_1n_3 \\ 2n_1n_2 & -1+2n_2^2 & 2n_2n_3 \\ 2n_1n_3 & 2n_2n_3 & -1+2n_3^2 \end{pmatrix}. \tag{4.13}$$

Der Richtungsvektor der Lichtstrahlen nach der Reflexion ist dann $\vec{n} * \text{ROT}$ und wird in `ref` ausgegeben.
Im letzten Teilbereich des Raytracing-Algorithmus erfolgt nun die Bestimmung der Schnittpunkte mit den Seitenwänden der Box über die Funktion

```
function [SpX,SpZ,SpZ1,SpY,SpY1] = schnittbox(MaSp,ref,c),
```

d.h. mit folgenden Ebenen:

- Boden des Würfels: $\epsilon_z : z = 0$,
- Decke des Würfels: $\epsilon_{z1} : z = 10$,
- Frontfläche des Würfels: $\epsilon_x : x = 0$,
- vordere Seitenwand der Box: $\epsilon_y : y = 0$,
- hintere Seitenwand des Würfels: $\epsilon_{y1} : y = 10$.

Dies ist der letzte Punkt des Algorithmus und die Ausgabe der zuvor beschriebenen Argumente erfolgt.

4.3 Beispiel Raytracing-Algorithmus

Abschließend soll hier noch ein durchgerechnetes Beispiel eines Raytracing-Algorithmus mit einem Dreiecks-Bézier-Patch als Primitiv angeführt werden. Zuerst wird hierbei der Schnitt eines Lichtstrahls (Gerade) mit einem Dreiecks-Bézier-Patch bestimmt werden. Danach wird die Reflexion wie eben beschrieben durchgeführt und die Schnitte des reflektierten Lichtstrahls mit dem Raum bestimmt. Alle Ergebnisse wurden mit den in MATLAB R2009a implementierten Funktionen bestimmt [siehe Kapitel 5].
Wir beginnen und betrachten jenen Bézier-Patch, der durch folgende zehn Kontrollpunkte erzeugt wird:

$$P_{003} = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}, P_{030} = \begin{pmatrix} 6 \\ 6 \\ 5 \end{pmatrix}, P_{300} = \begin{pmatrix} 6 \\ 3 \\ 5 \end{pmatrix}, P_{120} = \begin{pmatrix} 6 \\ 5 \\ 6 \end{pmatrix}, P_{210} = \begin{pmatrix} 6 \\ 4 \\ 8 \end{pmatrix},$$

$$P_{021} = \begin{pmatrix} 5 \\ 5 \\ 6 \end{pmatrix}, P_{201} = \begin{pmatrix} 5 \\ 3 \\ 6 \end{pmatrix}, P_{111} = \begin{pmatrix} 4,5 \\ 4,5 \\ 7 \end{pmatrix}, P_{102} = \begin{pmatrix} 4 \\ 3 \\ 6 \end{pmatrix}, P_{012} = \begin{pmatrix} 4 \\ 4 \\ 6 \end{pmatrix}.$$

Als Augpunkt setzt man wie zuvor $A = (0, 5, 5)^T$, die Größe des betrachteten Raums sei $[10 \times 10 \times 10] \subseteq \mathbb{R}^3$. Diese Wahl der Punkte entspricht genau der Setzung bei der Defaultwahl 0.
Wir bestimmen nun die Jacobi-Matrix der zu minimierenden Funktion 4.10 und erhalten für die Komponenten der Jacobi-Matrix 4.11 mit unserem Kontrollpolygon folgende Resultate:

$$\begin{aligned}
\frac{\partial f_1}{\partial r} &= 6r^2 - rs + 12r - s^2/2 + (13s)/2 - 5 \\
\frac{\partial f_1}{\partial s} &= (13r)/2 - rs - r^2/2 + 6s^2 + 12s - 5 \\
\frac{\partial f_1}{\partial \tau} &= \vec{v}_x \\
\frac{\partial f_2}{\partial r} &= 12r + (17s)/2 - 5rs - (5s^2)/2 - 6 \\
\frac{\partial f_2}{\partial s} &= (17r)/2 - 5rs - (5r^2)/2 + 6s^2 + 12s - 5 \\
\frac{\partial f_2}{\partial \tau} &= 5 - \vec{v}_y
\end{aligned}$$

$$\frac{\partial f_3}{\partial r} = 6r^2 + 8rs + 6r + 2s^2 + s - 3$$
$$\frac{\partial f_3}{\partial s} = 4r^2 + 4rs + r + 6s^2 + 6s - 3$$
$$\frac{\partial f_3}{\partial \tau} = 5 - \vec{v}_z$$

Eine jener Geraden, welche die konvexe Hülle unseres Bézier-Patches schneidet, ist

$$g_1 : X = (0, 5, 5)^T + \tau \cdot (0.092368038, -0.028980659, -0.025065253)^T.$$

Als Startwert für die Durchführung des modifizierten Newton-Verfahrens 3.3.3.2 verwenden wir, wie eben erwähnt, das arithmetische Mittel der Schnittpunkte mit der konvexen Hülle. Dieses werden noch mit den Formeln 2.7 aus Abschnitt 2.2.1.1 auf baryzentrische Koordinaten umgerechnet.
In unserem Beispiel sind somit $\alpha = 0.654974757$, $\beta = 0.244662274$ und $\gamma = 0.100362969$ die baryzentrischen Koordinaten des arithmetischen Mittels der Schnitte der Gerade g_1 mit der konvexen Hülle.
Das modifizierte Newton-Verfahren führt zu folgendem Ergebnis:

k	x	y	z	τ	$\|f(x_k)\|_2$
0	3.9837909	3.7500775	3.9189472	43.12953949	0.607890332210059
1	4.2222608	3.6752571	3.8542353	45.71127501	0.336791227865944
2	4.2509039	3.6662703	3.8464626	46.02137337	0.009448935349886
3	4.2500641	3.6665337	3.8466905	46.01228067	$2.076371935659099 \cdot 10^{-5}$
4	4.2500625	3.6665343	3.8466909	46.01226366	$1.224881836640226 \cdot 10^{-10}$
5	4.2500625	3.6665342	3.8466909	46.01226366	$1.404333387430681 \cdot 10^{-15}$

Bei einem Startwert nahe an der Lösung, wie eben in diesem Beispiel, sieht man die lokal quadratische Konvergenz des modifizierten Newton-Verfahren. Da wir solche Startwerte mit Hilfe der Schnitte mit der konvexen Hülle kennen, verwendet die Implementation eben genau diese Methode.
Mittels der Funktion `normalv` wird über das Kreuzprodukt nun der Normalvektor an den Bézier-Patch am Schnittpunkt,

$$\vec{n} = (1.15082234, -0.12060381, -1.03010648)^T,$$

errechnet. Anschließend wird über Rotation die Reflexion, wie zuvor beschrieben, bestimmt und die Gerade mit dem Richtungsvektor

$$\vec{v} = (0.21976779, -0.00373005, -0.25432901)^T$$

mit allen Begrenzungsflächen der Box geschnitten. Als einzige Ebene wird innerhalb des betrachteten Würfels die Ebene $\epsilon_z : z = 0$ geschnitten und als Schnittpunkt

$$SpZ = (7.57402011, 3.61011047, 0)^T$$

berechnet. Dies schließt alle Berechnungen des in Abschnitt 4.2 beschriebenen Raytracing-Algorithmus ab.

Genannt werden sollen noch die Laufzeiten der einzelnen Teilalgorithmen. Diese sind aus nachfolgender Tabelle ersichtlich.
Wie schon zuvor erwähnt, wird der größte Teil der Rechenleistung in der Funktion `linesearch` aufgewandt. Daher ist es von besonderer Bedeutung, diesen Teil zu beschleunigen. So ist als Beispiel für eine solche Beschleunigung die Funktion `linesearch2` angeführt. In ihr wird die Jacobi-Matrix nur im ersten Iterationsschritt bestimmt und in den nachfolgenden jene aus dem ersten Schritt als Approximation verwendet.
Eine weitere Möglichkeit zur Verkürzung der Rechenzeit lässt sich die Anzahl der Iterationschritte heruntersetzen. Dies wurde im Algorithmus `linesearch3` verwendet und dabei K als Anzahl der Iterationsschritte von 10 auf 6 reduziert. Dabei gehen zwar einige Schnittpunkte verloren, da sie nicht die Bedingung an die Norm erfüllen, doch ist eine Drittelung der Laufzeit ein starkes Argument für diesen Schritt.
Der ausgewertete Algorithmus erzeugte 256 Geraden, welche auf Schnitt mit dem Bézier-Patch getestet wurden. Dies entspricht der Starteingabe $k = 4$.

$function$	$Laufzeit(sek)$	$function$	$Laufzeit(sek)$
`konvhull`	0.45588	`linesearch2`	63.56613
`arith`	0.00236	`linesearch3`	36.67718
`baryzent`	0.01002	`normalv`	1.00324
`jacobi`	3.68992	`rotpi`	0.25199
`linesearch`	92.00218	`schnittbox`	0.98133

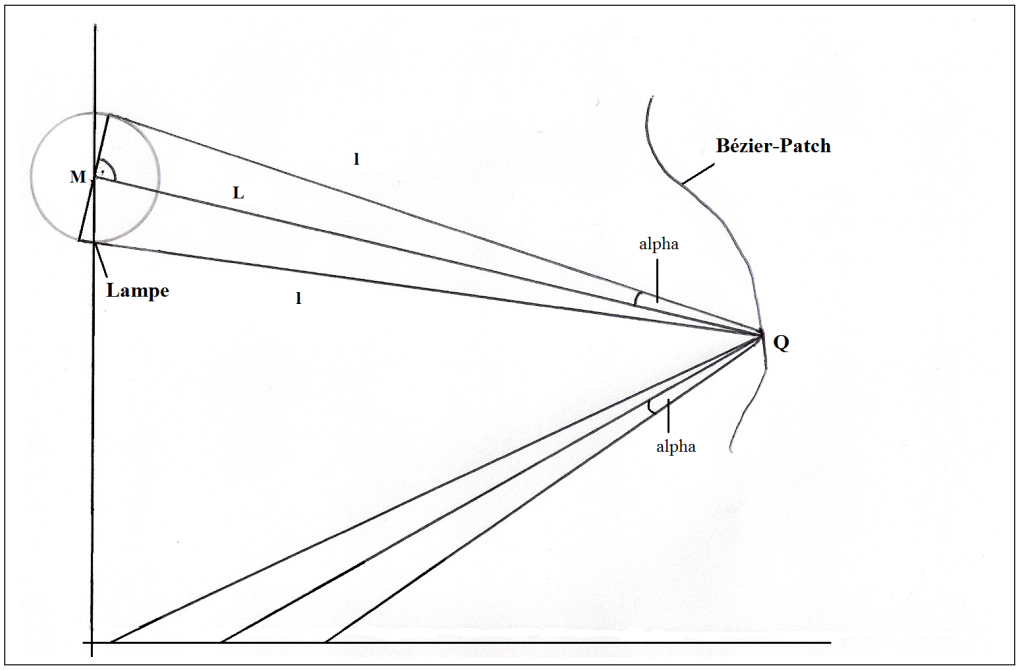

Abbildung 4.3: Illustration des alternativen Lösungsansatz

4.4 Ein alternativer Lösungsweg

Ein zum soeben gezeigten grundsätzlich unterschiedlicher Lösungsweg soll nun noch skizziert werden. Die Grundüberlegung dabei ist das Umgehen der durch das modifizierte Newton-Verfahren durchgeführten Schnittpunktberechnung, zumal der Algorithmus in diesem Teil die mit Abstand größte Rechenleistung aufwenden muss.
Anstelle des Augpunktes, d.h. der Lampe, werden nun Punkte auf dem Bézier-Patch als Startpunkte für das Verfahren verwendet und für jeden dieser Punkte ein kongruentes Verfahren durchgeführt.

Sei nun Q ein solcher Punkt auf dem Dreiecks-Bézier-Patch, von dem aus wir unsere Betrachtungen durchführen wollen. Mit R sei der uns bekannte Radius der Kugel, als welche wir die Lampe modellieren, bezeichnet, zudem setzen wir l wie folgt:

$$l = \overline{QM} = \|M - Q\|_2 = r_Q$$

mit M Mittelpunkt der kugelförmigen Lampe. Mit l' wird weiters die Distanz $\overline{QP}$ bezeichnet, wobei P einen Punkt auf der Oberfläche der Lampe repräsentiert.

Ziel ist nun eine Funktion für die Intensität des Lichts bei Q zu finden. Dazu betrachten

wir das Dreieck MPQ und betreiben darin Trigonometrie. Dabei bezeichne α den Winkel zwischen $\overline{MQ}$ und $\overline{QP}$, ϵ jenen zwischen $\overline{MQ}$ und $\overline{MP}$ und schließlich ν den Winkel, welcher durch $\overline{MP}$ und $\overline{QP}$ eingeschlossen wird.
Dann gelten folgende Eigenschaften:

$$l' = \frac{\|M - Q\| \sin \epsilon}{\sin \nu} \tag{4.14}$$

$$\frac{R}{\sin \alpha} = \frac{\|M - Q\|}{\sin(2\pi - \alpha - \epsilon)} = \frac{\|M - Q\|}{-\sin(\alpha + \epsilon)} \tag{4.15}$$

Die letzte Gleicheit folgt aus dem Additionstheorem $\sin(\gamma - \beta) = \sin\gamma\cos\beta - \sin\beta\cos\gamma$ mit $\gamma = 2\pi$ und $\beta = \alpha + \epsilon$.
Weiters ist

$$\sin(\alpha + \epsilon) = -\frac{\|M - Q\| \sin \alpha}{R} \tag{4.16}$$

sowie

$$\epsilon = \underbrace{\arcsin \frac{\|M - Q\|}{R}}_{\nu} - \alpha, \tag{4.17}$$

da ja

$$\sin \nu = \frac{\|M - Q\| \sin \alpha}{R} \tag{4.18}$$

gilt. Hierbei wird α auch oft als AZIMUTWINKEL bezeichnet und es gilt $\alpha \in [0, 2\pi[$. Anschaulich ist α der Winkel zwischen der positiven $x-$Achse und der senkrechten Projektion des Radius in die $x - y-$Ebene, d.h. der waagrecht liegende Winkel.
In Äquivalenz dazu sei ψ der senkrecht stehende Winkel zwischen positiver $z-$Achse und Radius, genannt POLARWINKEL und es gelte $\psi \in [0, \pi[$.
Dann ist eine Funktion $L(\alpha, \psi)$, die die Lichtstärke angibt, welche ausgehend vom Punkt P auf Q auftrifft, von der Form

$$L(\alpha, \psi) = \begin{cases} 0, & \text{falls} \quad \tan\alpha > \frac{R}{\|M-Q\|} \\ \frac{I(\alpha+\kappa, \psi+\lambda)}{\|M-Q\|^2 \sin^2 \epsilon} \cos^2 \nu \sin^2 \alpha. \end{cases}$$

Die erste Option erklärt sich dadurch, dass bei Eintritt der Bedingung $\tan\alpha > \frac{R}{\|M-Q\|}$ der Strahl an der Lampe vorbeigeht und somit umgekehrt kein Licht von der Lampe ausgehend den Punkt Q erreicht.
Im zweiten Fall kommt eine Funktion der Intensität ins Spiel. Sie hängt nicht nur von α und ψ ab, sondern auch von κ und λ. Diese Winkel ergeben sich aus den Kugelkoordinaten

von Q mit

$$\begin{aligned} Q_x &= r_Q \sin\kappa \cos\lambda \\ Q_y &= r_Q \sin\kappa \sin\lambda \\ Q_z &= r_Q \cos\kappa. \end{aligned}$$

Somit ist

$$Q_x^2 + Q_y^2 = r_Q^2 \sin^2\kappa \cos^2\lambda + r_Q^2 \sin^2\kappa \sin^2\lambda = r_Q^2 \sin^2\kappa.$$

Daher lässt sich Q mit dem Triple $[r_Q, \kappa, \lambda]$ identifizieren mit

$$r_Q = \|M - Q\|, \qquad \kappa = \arcsin\frac{\sqrt{Q_x^2 + Q_y^2}}{r_Q}, \qquad \lambda = \arctan\frac{Q_y}{Q_x}.$$

Mit diesem Wissen wollen wir nun abschließend die Funktion der Intensität I bei Q angeben. Dazu integrieren wir die Funktion $L(\alpha, \psi)$ über die u und v, für die gilt, dass sie aus dem Paramterbereich P_S des B-Splines sind. Zudem multiplizieren wir mit dem Quadrat von PQ^{-1} und erhalten daher

$$I(\alpha + \kappa, \psi + \alpha) = \int_{u,v \in P_S} L(\alpha(u,v), \psi(u,v))\mathrm{d}(u,v)\frac{\sin^2\alpha}{\sin^2\epsilon \cdot R^2}. \tag{4.19}$$

Dass PQ eben genau von der hier verwendeten Darstellung ist, folgt aus den Relationen (4.14) und (4.18). Wir haben also eine Funktion erhalten, welche die Intensität des Lichtes bei Q über ein Integral darstellt. Zur Bestimmung einer Funktion für die Intensität an einem Punkt auf einer der den Raum begrenzenden Ebenen führt man obige Überlegungen nochmals kongruent durch. Dies soll die Skizze eines möglichen alternativen Lösungsweges beschließen.

4.5 Conclusio und Ausblick

Damit haben wir nun die Aufgabenstellung abgearbeitet. Es wurde ein Raytracing-Algorithmus entwickelt, in welchem das reflektierende Primitiv durch Dreiecks-Bézier-Patches dargestellt wird und der, unter Vorgaben der Strahlenoptik, die Lichtreflexion bestimmt.

Es hat sich dabei zweierlei gezeigt. Zum einen rechtfertigen Bézier-Patches durch ihre Eigenschaften ihre Wahl als Objekte im Raytracing. Vor allem die Tatsache, dass sich damit auch komplexere Körper sehr gut approximieren lassen, ist als positiv anzusehen. Hinzu kommt noch eine große Anzahl von Algorithmen, welche dem Anwender bei Bézier-Patches zur Verfügung stehen. Vor allem der Algorithmus zur Differentiation sowie die Junktion von Bézier-Patches sind von Nutzen.

Doch es sind im Laufe der Arbeit auch Nachteile beim Verwenden von Dreiecks-Bézier-Patches zu Tage getreten. In erster Linie ist hier die Berechnung der Schnittpunkte der Gerade mit dem Bézier-Patch zu nennen. Sie verlangsamt den Algorithmus doch beträchtlich, führt jedoch andererseits im Allgemeinen zu zufriedenstellenden Ergebnissen. Mehrere Ideen zur Beschleunigung dieses Teiles des Raytracers wurden eingearbeitet. Das Beschränken auf Geraden, welche die konvexe Hülle des Bézier-Patches schneiden, sei hier ebenso als Beispiel genannt wie die Approximation der Jacobi-Matrix im Newton-Verfahren.

Mit solchen Methoden konnte zwar die Laufzeit markant verbessert werden, dennoch wird eine weitere Beschleunigung der Schlüsselpunkt in der Zukunft sein. Ein interessanter Weg könnte hierbei der in Abschnitt 4.4 beschriebene alternative Lösungsweg sein, da man hier gänzlich ohne die Schnittpunktberechnung arbeiten könnte.
Abschließend bleibt zu sagen, dass aufgrund der vielfältigen Anwendungen von Raytracing-Algorithmen in der Praxis eine weiteres Verfolgen des Themas als durchaus lohnend erscheint.

Im letzten Kapitel sind nun noch die konkreten Implementationen, geschrieben in MATLAB, enthalten. Sie beschließen diese Arbeit.

5 Implementationen

Dieses Kapitel enthält alle Implementation, die zur Durchführung des in Abschnitt 4.2 beschriebenen Algorithmus notwendig sind. Die mathematischen Erklärungen finden sich dort. Ebenso ist die Abfolge der Teilalgorithmen in ebenjenem Abschnitt angeführt, sie ist aber auch der Kopffunktion RAYTRACER zu entnehmen, welche als erstes genannt wird.
Zudem werden bei jeder Funktion Eingabe- und Ausgabeargumente genau angeführt. In den Algorithmen sind immer wieder kurze Erklärungen enthalten, welche mit dem Prozentzeichen am Zeilenanfang gekennzeichnet sind. Außerdem erfolgt bei jeder Funktion die Ausgabe der Rechendauer.

5.1 function raytracer

```
function [MaSp,SpX,SpZ,SpZ1,SpY,SpY1] = raytracer

% Durchführung eines Ray Tracing Algorithmus.
% Zuerst erfolgt die Eingabe des Augpunktes sowie der zehn Punkte
% des Kontrollpolygon. Wird jeweils 0 eingegeben, erfolgen folgende
% Setzungen für die Punkte:
%
% A = [0;5;5]
% P_003 = [3;3;3];
% P_030 = [6;6;5];
% P_300 = [6;3;5];
% P_120 = [6;5;6];
% P_210 = [6;4;8];
% P_021 = [5;5;6];
% P_201 = [5;3;6];
% P_111 = [4.5;4.5;7];
% P_102 = [4;3;6];
% P_012 = [4;4;6];
%
```

```
20 % Zudem wird über k die Anzahl der Geraden, welche den Raum treffen,
21 % festgelegt über n=2^k-1 mit dem Netz auf der Kugel.
22 % Bei k=0 wird k gleich 4 gesetzt. Somit werden 256 Geraden erzeugt.
23 % Danach läuft der raytracing-Algorithmus durch und berechnet die
24 % Schnittpunkte der am Bézier-Patch reflektierten Strahlen mit den
25 % Seitenwänden der Box.
26 %
27 % OUTPUT
28 % MaSp...Schnittpunkte mit dem Bézier-Patch
29 % SpX....Schnittpunkte nach der Reflexion mit der y-z-Ebene
30 % SpZ....Schnittpunkte nach der Reflexion mit der x-y-Ebene
31 % SpZ1...Schnittpunkte nach der Reflexion mit der Ebene z=10.
32 % SpY....Schnittpunkte nach der Reflexion mit der x-z-Ebene
33 % SpY1...Schnittpunkte nach der Reflexion mit der Ebene y=10.
34
35 k=input('Bitte k eingeben (Setzung bei 0 siehe help raytracer): ');
36 if k==0
37     k=4;
38 end
39 A=input('Bitte Augpunkt A eingeben: ');
40 if A==0
41     A=[0;5;5];
42 end
43 P_003=input('Bitte P_003 eingeben: ');
44 if P_003==0
45     P_003=[3;3;3];
46 end
47 P_030=input('Bitte P_030 eingeben: ');
48 if P_030==0
49     P_030=[6;6;5];
50 end
51 P_300=input('Bitte P_300 eingeben: ');
52 if P_300==0
53     P_300=[6;3;5];
54 end
55 P_120=input('Bitte P_120 eingeben: ');
56 if P_120==0
57     P_120=[6;5;6];
58 end
59 P_210=input('Bitte P_210 eingeben: ');
60 if P_210==0
61     P_210=[6;4;8];
62 end
```

```
63 P_021=input('Bitte P_021 eingeben: ');
64 if P_021==0
65     P_021=[5;5;6];
66 end
67 P_201=input('Bitte P_201 eingeben: ');
68 if P_201==0
69     P_201=[5;3;6];
70 end
71
72 P_111=input('Bitte P_111 eingeben: ');
73 if P_111==0
74     P_111=[4.5;4.5;7];
75 end
76 P_102=input('Bitte P_102 eingeben: ');
77 if P_102==0
78     P_102=[4;3;6];
79 end
80 P_012=input('Bitte P_012 eingeben: ');
81 if P_012==0
82     P_012=[4;4;6];
83 end
84
85 fprintf('Berechnung konvexe Hülle: \n')
86 [t,XS,YS,ZS,SP,X] =...
87     konvhull(k,A,P_003,P_030,P_300,P_111,P_120,P_210,P_012,P_021,P_102,P_201);
88
89 fprintf('Berechnung arithmetische Mittel als Startwert: \n')
90 [ARI,XNE,te,c] = arith(A,XS,YS,ZS,SP);
91
92 fprintf('Berechnung baryzentrische Koordinaten: \n')
93 [alpha,beta] = baryzent(X,ARI,P_003,P_030,P_300,k,c);
94
95 fprintf('Berechnung der partiellen Ableitungen der Schnittfunktion mit dem Bézier-Pa
96 [f1dr,f1ds,f1dv,f2dr,f2ds,f2dv,f3dr,f3ds,f3dv,f1,f2,f3] =...
97     jacobi(A,P_003,P_030,P_300,P_111,P_120,P_210,P_012,P_021,P_102,P_201);
98
99 fprintf('Berechnung SP mittels modifiziertem Newtonverfahren: \n')
100 [x] = linesearch(alpha,beta,te,c,XNE,A,P_003,P_030,P_300,P_111,P_120,...
101     P_210,P_012,P_021,P_102,P_201,f1,f2,f3,f1dr,f1ds,f1dv,f2dr,f2ds,...
102     f2dv,f3dr,f3ds,f3dv);
103
104 fprintf('Berechnung NV über Kreuzprodukt: \n')
105 [N,Rx,Ry,Rz,MaSp] = normalv(x,A,XNE,P_003,P_030,P_300,f1dr,f1ds,...
```

```
106     f2dr,f2ds,f3dr,f3ds,c);
107
108 fprintf('Rotation um pi: \n')
109 [ref] = rot_pi(N,Rx,Ry,Rz,c);
110
111 fprintf('Berechnung Schnitte mit der Box: \n')
112 [SpX,SpZ,SpZ1,SpY,SpY1] = schnittbox(MaSp,ref,c);
```

5.2 function konvhull

Die Funktion KONVHULL initialisiert die 'Lichtgeraden' und liest den Augpunkt, den Parameter k sowie die Punkte des Kontrollpolygons ein. Danach wird die konvexe Hülle des Bezierpatches über die MATLAB-Funktion CONVHULL erzeugt und alle 'Lichtgeraden' auf Schnitte mit dieser konvexen Hülle getestet.

```
1  function [t,XS,YS,ZS,SP,X] =...
2  konvhull(k,A,P_003,P_030,P_300,P_111,P_120,P_210,P_012,P_021,P_102,P_201)
3
4  % INPUT
5  % k....Parameter
6  % A....Augpunkt
7  % P_i..10 Punkte des Kontrollpolygons
8  %
9  % OUTPUT
10 % SP...Schnittpunkte mit der konvexen Hülle
11 % XS...x-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
12 % YS...y-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
13 % ZS...z-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
14 % t....zu SP gehörende Parameter der Strahlen
15
16 tic
17
18 n = 2^k-1;
19 ra=.1;
20
21 % Erzeuge Geraden vom Augpunkt aus
22 % Geraden sind bestimmt durch Augpunkt und Punkt auf einer
23 % davorliegenden Kugel in Polarkoordinaten
24
25 theta = pi*(-n:2:n)/n;
```

```
26 phi = (pi/2)*(-n:2:n)/n;
27
28 [TT PP] = meshgrid(theta,phi);
29 XX = A(1)+ra*cos(PP).*cos(TT);
30 YY = A(2)+ra*cos(PP).*sin(TT);
31 ZZ = A(3)+ra*sin(PP);
32
33 % Berechnung der konvexen Hülle des Bezierpatches und Erstellen eines
34 % plots der konvexen Hülle
35
36 XC=[P_003';P_030';P_300';P_120';P_210';P_021';P_201';P_111';...
37     P_102';P_012'];
38 K=convhulln(XC);
39 trisurf(K,XC(:,1),XC(:,2),XC(:,3))
40 axis([0 10 0 10 0 10])
41 xlabel('x-Achse')
42 ylabel('y-Achse')
43 zlabel('z-Achse')
44 title('Konvexe Hülle des Bezierpatches', 'FontSize',10)
45
46 % Überprüfe ob die Gerade die konvexe Hülle schneidet
47 % Betrachte dazu das erste Dreieck das die konvexe Hülle aufbaut
48 % Dieses Dreieck hat die Eckpunkte X(K(1,1),:), X(K(1,2),:),
49 % X(K(1,3),:).
50 % Berechne s,u durch B*(t,s,u)'=XK3
51 % B ist die Matrix [XX YY ZZ; XK1(1-3);XK2(1-3)]
52 % XK3 entspricht A-X(K(1,1),:)
53 % Ist s+u in [0,1] und gilt weiters 0≤u≤1 und 0≤s≤1,
54 % dann schneidet die Gerade das Dreieck
55
56 X=zeros(2^(2*k),12);
57 Y=zeros(2^(2*k),12);
58 Z=zeros(2^(2*k),12);
59
60 s=zeros((2^(2*k))*12,3);
61
62 XC1=zeros(3,12);
63 XC2=zeros(3,12);
64 XC3=zeros(3,12);
65
66 % Erstelle die Vektoren welche die Seitendreiecke der konvexen
67 % Hülle definieren und speichere sie in XC1, XC2, XC3
68 % je 3x10 Matrizen
```

```
69
70  for l=1:1:12
71      XC1(:,l)=XC(K(l,2),:)'-XC(K(l,1),:)';
72      XC2(:,l)=XC(K(l,3),:)'-XC(K(l,1),:)';
73      XC3(:,l)=A-XC(K(l,1),:)';
74  end
75
76  % Berechne nun für jedes Dreieck und ALLE Schnittgeraden die
77  % Matrix B und Parameter s [10*2^2k,3], falls B nicht singulär
78  % ist (d.h. die Norm von B sich ausreichend von 0 unterscheidet)
79
80  for l=1:3:34
81
82   for j=1:1:(2^k)^2
83
84          B=[(A(1)-XX(j)) XC1(l) XC2(l);
85              (A(2)-YY(j)) XC1(l+1) XC2(l+1);
86              (A(3)-ZZ(j)) XC1(l+2) XC2(l+2)];
87
88           if norm(det(B))>(10^-12)
89               ss=B\[XC3(l);XC3(l+1);XC3(l+2)];
90               s(j+((l-1)/3)*(2^k)^2,:)=ss';
91           end
92   end
93  end
94
95  % Falls die Parameter (d.h. Zeilen in s!!) die Bedingungen
96  % erfüllen, dann werden die zugehörigen Vektoren [XX;YY;ZZ]
97  % in [X;Y;Z] geschrieben [2^2k,12]. Jede Spalte entspricht dabei
98  % den Dreiecken, jene X=¬0 schneiden das Dreieck
99
100 for l=1:1:12
101         for m=1:1:(2^k)^2
102             if s(m+(l-1)*(2^k)^2,2)≤ 1
103                 if s(m+(l-1)*(2^k)^2,2)≥0
104                 if s(m+(l-1)*(2^k)^2,3)≤1
105                     if s(m+(l-1)*(2^k)^2,3)≥0
106                     if s(m+(l-1)*(2^k)^2,2) + ...
107                             s(m+(l-1)*(2^k)^2,3) ≤ 1
108                     if s(m+(l-1)*(2^k)^2,2) + ...
109                             s(m+(l-1)*(2^k)^2,3) > 0
110                         X(m,l)=XX(m);
111                         Y(m,l)=YY(m);
```

```
112                         Z(m,l)=ZZ(m);
113                     end
114                     end
115                     end
116                 end
117                 end
118             end
119         end
120 end
121
122 % Fasse nun jene x-,y-,z-Werte zusammen welche die Geraden
123 % bilden, die die konvexe Hülle schneiden und schreibe sie
124 % in die Matrizen XS, YS, ZS.
125
126 u=0;
127 XS=zeros(u,1);
128 YS=zeros(u,1);
129 ZS=zeros(u,1);
130
131 for j=1:1:(2^(2*k))*12
132
133     if X(j)≠0
134         u=u+1;
135         XS(u,1)=X(j);
136         YS(u,1)=Y(j);
137         ZS(u,1)=Z(j);
138     end
139 end
140
141 % Extrahiere nun all jene Parameter, die einen Schnittpunkt mit
142 % der konvexen Hülle erzeugen und speichere sie in t.
143
144 u=0;
145 t=zeros(u,1);
146
147 for j=1:1:((2^k)^2)*12
148     if X(j)≠0
149         u=u+1;
150         t(u,1)=s(j,1);
151     end
152 end
153
154 % Errechne nun die Schnittpunkte der Geraden mit den
```

```
155 % Dreiecken, welche die konvexe Hülle bilden. Speichere diese
156 % Punkte in SP [u,3]. Jede Zeile entspricht einem
157 % Schnittpunkt, u ist die Anzahl der Parameter in t bzw.
158 % numel(XS)/3.
159
160  SP=zeros(u,3);
161  for j=1:1:u
162              SP(j,1)=A(1) + t(j,1).*(XS(j,1)-A(1));
163              SP(j,2)=A(2) + t(j,1).*(YS(j,1)-A(2));
164              SP(j,3)=A(3) + t(j,1).*(ZS(j,1)-A(3));
165  end
166
167 toc
```

5.3 function arith

In der Funktion ARITH werden die arithmetischen Mittel der eben bestimmten Schnittpunkte der Geraden mit der konvexen Hülle des Bézier-Patches gebildet. Dabei schneidet jede Gerade die konvexe Hülle i.A. in zwei Punkten.

```
1  function [ARI,XNE,te,c] = arith(A,XS,YS,ZS,SP)
2
3  % INTPUT
4  % A....Augpunkt
5  % SP...Schnittpunkte mit der konvexen Hülle
6  % XS...x-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
7  % YS...y-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
8  % ZS...z-Koordinaten der Punkte um A, welche Schnittgeraden erzeugen
9  %
10 % OUTPUT
11 % ARI..Arithmetische Mittel der Schnitte mit der konvexen Hülle
12 % XNE..Punkte auf der Kugel um A, die Schnittgeraden ergeben
13 % te...zu ARI gehörende Parameter der Schnittgeraden
14 % c....Anzahl der Schnitte mit der konvexen Hülle
15
16 tic
17
18 % In AM1 sind die ersten drei Spalten die Koordinaten des ersten
19 % Schnittpunktes, die Spalten vier bis sechs die Koordinaten des
```

```
% zweiten.

j=0;
AM1=zeros(j,6);
u=numel(XS);

for j=1:1:u
    for c=1:1:u
    if XS(j)==XS(c)
        if YS(j)==YS(c)
            if ZS(j)==ZS(c)

                AM1(j,1)=SP(j,1);
                AM1(j,2)=SP(j,2);
                AM1(j,3)=SP(j,3);
                AM1(j,4)=SP(c,1);
                AM1(j,5)=SP(c,2);
                AM1(j,6)=SP(c,3);

            end
        end
    end
    end
end

% Setze die doppelt gezählten Werte der Schnittpunkte gleich 1.

for j=1:1:u
    if AM1(j,1)==AM1(j,4)
        if AM1(j,2)==AM1(j,5)
            if AM1(j,3)==AM1(j,6)
                AM1(j,:)=1;

            end
        end
    end
end

% Berechne nun die arithmetischen Mittel jener Elemente aus AM1,
% die ungleich 1 sind.

AM=zeros(j,3);

```

```
63 for j=1:1:u
64     if AM1(j,1)≠1
65         AM(j,1)=(AM1(j,1)+AM1(j,4))/2;
66         AM(j,2)=(AM1(j,2)+AM1(j,5))/2;
67         AM(j,3)=(AM1(j,3)+AM1(j,6))/2;
68     end
69 end
70
71 % Extrahiere nun jene Werte ungleich null und fasse sie in ARI
72 % bzw. in XNE (jene Werte auf der Kugel um den Augpunkt,
73 % welche Schnittpunkte erzeugende Geraden ergeben) zusammen.
74
75 c=0;
76 ARI=zeros(c,3);
77 XNE=zeros(c,3);
78 u=numel(AM)/3;
79
80 for j=1:1:u
81     if AM(j,1)≠0
82         c=c+1;
83         ARI(c,1)=AM(j,1);
84         ARI(c,2)=AM(j,2);
85         ARI(c,3)=AM(j,3);
86         XNE(c,1)=XS(j,1);
87         XNE(c,2)=YS(j,1);
88         XNE(c,3)=ZS(j,1);
89     end
90 end
91
92 % Berechne die zu den arithmetischen Mitteln gehörenden Parameter
93 % der Geraden auf denen die Mittel liegen.
94
95 u=numel(ARI)/3;
96 te=zeros(u,1);
97
98 for j=1:1:u
99     te(j)=(ARI(j,1)-A(1))/(XNE(j,1)-A(1));
100 end
101
102 toc
```

5.4 function baryzent

Die Funktion BARYZENT berechnet die baryzentrischen Koordinaten der arithmetischen Mittel der Schnittpunkte, um sie in der Folge als Startwerte für das Newton-Verfahren zu verwenden. Dazu werden die Formel aus Abschnitt 2.2.1.1 verwendet.

```
function [alpha,beta,gamma] = baryzent(X,ARI,P_003,P_030,P_300,k,c)

% INPUT
% ARI..Arithmetische Mittel der Schnitte mit der konvexen Hülle
% P_003,P_030,P_300...Begrenzungspunkte des Kontrollpolygons
% k....Parameter
% c....Anzahl der Schnitte mit der konvexen Hülle
%
% OUTPUT
% alpha..baryzentrische Koordinate
% beta...baryzentrische Koordinate
% gamma..baryzentrische Koordinate

tic

% Zusatzinformation:
% Zähle wieviele Geraden die jeweiligen Begrenzungsdreiecke
% der konvexen Hülle schneiden und schreibe diese Ergebnisse
% in uz(12,1)

uz=zeros(12,1);

for l=1:1:12
    u=0;
    for j=1:1:(2^k)^2
        if X(j+(l-1)*(2^k)^2)≠0
            u=u+1;
            uz(l,1)=u;
        end
    end
end

% Berechne nun die Baryzentrischen Koordinaten der Schnittpunkte
% bezüglich des der orthogonalen Projektion auf die x-y-Ebene
% zugrundeliegenden Dreiecks
```

```
36
37 gamma=zeros(c,1);
38 beta=zeros(c,1);
39 alpha=zeros(c,1);
40
41 for j=1:1:c
42
43         gamma(j,1)=((P_003(2)-P_030(2))*ARI(j,1)+...
44             (P_030(1)-P_003(1))*ARI(j,2)+P_003(1)*P_030(2)-...
45             P_003(2)*P_030(1))/((P_003(2)-P_030(2))*P_300(1)+...
46             (P_030(1)-P_003(1))*P_300(2)+P_003(1)*P_030(2)-...
47             P_003(2)*P_030(1));
48
49         beta(j,1)=((P_003(2)-P_300(2))*ARI(j,1)+...
50             (P_300(1)-P_003(1))*ARI(j,2)+P_003(1)*P_300(2)-...
51             P_300(1)*P_003(2))/((P_003(2)-P_300(2))*P_030(1)+...
52             (P_300(1)-P_003(1))*P_030(2)+P_003(1)*P_300(2)-...
53             P_300(1)*P_003(2));
54
55         alpha(j,1)=1-gamma(j,1)-beta(j,1);
56
57 end
58
59 toc
```

5.5 function jacobi

Die Funktion JACOBI verwendet die durch (4.10) definierte Funktionen f_1, f_2 und f_3, welche den Schnitt des Bézier-Patches mit der Geraden darstellen. Anschließend bilde ich die Jacobi-Matrix zur Verwendung im mehrdimensionalen Newtonverfahren. Dazu werden die partiellen Ableitungen nach den baryzentrischen Koordianten r und s sowie nach dem Parameter der Geraden v berechnet.

```
1 function [f1dr,f1ds,f1dv,f2dr,f2ds,f2dv,f3dr,f3ds,f3dv,f1,f2,f3] = ...
2     jacobi(A,P_003,P_030,P_300,P_111,P_120,P_210,P_012,P_021,P_102,P_201)
3
4 % INPUT
5 % A....Augpunkt
6 % P_i..10 Punkte des Kontrollpolygons
```

```
7  %
8  % OUTPUT
9  % f1....x-Koordinate der Schnittfunktion
10 % f2....y-Koordinate der Schnittfunktion
11 % f3....z-Koordinate der Schnittfunktion
12 % f1dr..Ableitung x-Koordinate nach r
13 % f1ds..Ableitung x-Koordinate nach s
14 % f1dv..Ableitung x-Koordinate nach v
15 % f2dr..Ableitung y-Koordinate nach r
16 % f2ds..Ableitung y-Koordinate nach s
17 % f2dv..Ableitung y-Koordinate nach v
18 % f3dr..Ableitung z-Koordinate nach r
19 % f3ds..Ableitung z-Koordinate nach s
20 % f3dv..Ableitung z-Koordinate nach v
21
22 tic
23
24 syms r
25 syms s
26 syms v
27 syms X
28
29 f1=r^3*(P_300(1) - P_003(1) + P_102(1) - P_201(1))+...
30     s^3*(P_030(1) - P_003(1) - P_021(1) + P_012(1))+...
31     s^2*(3*P_003(1) - 2*P_012(1) + P_021(1))+...
32     r^2*(3*P_003(1) - 2*P_102(1) + P_201(1))+...
33     r^2*s*(-3*P_003(1) + P_012(1) + 2*P_102(1)-...
34     P_201(1) + P_210(1) - P_111(1)) + r*s^2*(-3*P_003(1)-...
35     P_021(1) + 2*P_012(1) + P_102(1) + P_120(1) - P_111(1))+...
36     r*s*(6*P_003(1) - 2*P_012(1) - 2*P_102(1) + P_111(1))+...
37     r*(-3*P_003(1) + P_102(1)) + s*(-3*P_003(1)+P_012(1))+...
38     P_003(1) - A(1) - v*(X - A(1));
39
40 f1dr=diff(f1,r);
41 f1ds=diff(f1,s);
42 f1dv=diff(f1,v);
43
44 syms r
45 syms s
46 syms v
47 syms Y
48
49 f2=r^3*(P_300(2) - P_003(2) + P_102(2) - P_201(2))+...
```

```
50      s^3*(P_030(2) - P_003(2) - P_021(2) + P_012(2))+...
51      s^2*(3*P_003(2) - 2*P_012(2) + P_021(2))+...
52      r^2*(3*P_003(2) - 2*P_102(2) + P_201(2))+...
53      r^2*s*(-3*P_003(2) + P_012(2) + 2*P_102(2)-...
54      P_201(2) + P_210(2) - P_111(2)) + r*s^2*(-3*P_003(2)-...
55      P_021(2) + 2*P_012(2) + P_102(2) + P_120(2) - P_111(2))+...
56      r*s*(6*P_003(2) - 2*P_012(2) - 2*P_102(2) + P_111(2))+...
57      r*(-3*P_003(2) + P_102(2)) + s*(-3*P_003(2)+P_012(2))+...
58      P_003(2) - A(2) - v*(Y - A(2));
59
60  f2dr=diff(f2,r);
61  f2ds=diff(f2,s);
62  f2dv=diff(f2,v);
63
64  syms r
65  syms s
66  syms v
67  syms Z
68
69  f3=r^3*(P_300(3) - P_003(3) + P_102(3) - P_201(3))+...
70      s^3*(P_030(3) - P_003(3) - P_021(3) + P_012(3))+...
71      s^2*(3*P_003(3) - 2*P_012(3) + P_021(3))+...
72      r^2*(3*P_003(3) - 2*P_102(3) + P_201(3))+...
73      r^2*s*(-3*P_003(3) + P_012(3) + 2*P_102(3)-...
74      P_201(3) + P_210(3) - P_111(3)) + r*s^2*(-3*P_003(3)-...
75      P_021(3) + 2*P_012(3) + P_102(3) + P_120(3) - P_111(3))+...
76      r*s*(6*P_003(3) - 2*P_012(3) - 2*P_102(3) + P_111(3))+...
77      r*(-3*P_003(3) + P_102(3)) + s*(-3*P_003(3)+P_012(3))+...
78      P_003(3) - A(3) - v*(Z - A(3));
79
80  f3dr=diff(f3,r);
81  f3ds=diff(f3,s);
82  f3dv=diff(f3,v);
83
84  toc
```

5.6 function linesearch

Die Funktion LINESEARCH ist der, auch von der Rechenzeit her, aufwändigste Teil des gesamten Algorithmus. Sie führt ein modifiziertes Newton-Verfahren 3.3.3.2 durch. Die

Schrittweite wird dabei mittels Liniensuche bestimmt, die Suchrichtung entspricht $d_k = Df(x)^{-1}f(x)$. Es werden K Iterationsschritte durchgeführt bzw. bei Erreichen der Schranke ϵ abgebrochen.
Die Rechendauer des hier angeführten Algorithmus lässt sich verkürzen, indem nur im ersten Schritt die Jacobi-Matrix berechnet wird und in den weiteren Schleifen als Approximation verwendet wird.

```
function [x] = ...
    linesearch(alpha,beta,te,c,XNE,A,P_003,P_030,P_300,P_111,P_120,...
    P_210,P_012,P_021,P_102,P_201,f1,f2,f3,f1dr,f1ds,f1dv,f2dr,f2ds,...
    f2dv,f3dr,f3ds,f3dv)

% INPUT
% alpha..baryzentrische Koordinate
% beta...baryzentrische Koordinate
% te....zu ARI gehörende Parameter der Schnittgeraden
% c.....Anzahl der Schnitte mit der konvexen Hülle
% P_i...10 Punkte des Kontrollpolygons
% f1....x-Koordinate der Schnittfunktion
% f2....y-Koordinate der Schnittfunktion
% f3....z-Koordinate der Schnittfunktion
% f1dr..Ableitung x-Koordinate nach r
% f1ds..Ableitung x-Koordinate nach s
% f1dv..Ableitung x-Koordinate nach v
% f2dr..Ableitung y-Koordinate nach r
% f2ds..Ableitung y-Koordinate nach s
% f2dv..Ableitung y-Koordinate nach v
% f3dr..Ableitung z-Koordinate nach r
% f3ds..Ableitung z-Koordinate nach s
% f3dv..Ableitung z-Koordinate nach v
%
% OUTPUT
% x.....3_x_c-Matrix, die in jeder Spalte die baryzentrischen Koordinaten
%       und den Parameter des Schnittpunktes enthält

tic

x=zeros(3,c);
nfx0=zeros(1,c);

for l=1:1:10
```

```
35      r=alpha(l);             % definiere die Startwert
36      s=beta(l);
37      v=te(l);
38      X=XNE(l,1);
39      Y=XNE(l,2);
40      Z=XNE(l,3);
41      xk=[alpha(l);beta(l);te(l)];
42      fx0=[subs(f1);subs(f2);subs(f3)]; % Funktionswert des Startwertes
43      epsilon=10^-10;         % Wähle Toleranz und maximale Iterationszahl K
44      K=10;
45      k=0;
46
47      while (norm(fx0)>epsilon) && k < K && (norm(fx0)<6)
48
49          Jac=zeros(3);               % Berechne in jedem Schritt
50          Jac(1,1)=subs(f1dr);        % die Jacobi-Matrix neu
51          Jac(1,2)=subs(f1ds);
52          Jac(1,3)=subs(f1dv);
53          Jac(2,1)=subs(f2dr);
54          Jac(2,2)=subs(f2ds);
55          Jac(2,3)=subs(f2dv);
56          Jac(3,1)=subs(f3dr);
57          Jac(3,2)=subs(f3ds);
58          Jac(3,3)=subs(f3dv);
59
60          if norm(det(Jac))>(10^-16)  % Führe Newton-Iteration durch
61              invjac=Jac^-1;
62              dk=invjac*fx0;
63
64              if k==0                 % Berechne nur cond der 1.Matrix
65                  kappa=cond(Jac,2);
66              end
67
68              gamma_k=1/kappa;        % Konditionszahl
69              k=k+1;
70              % Berechne nabla_h für modifiziertes Newtonverfahren
71              g = [f1;f2;f3];
72              nabla_h = 2*g.'*Jac;    % Berechnung des Gradienten von h
73              z_k = 0.25*gamma_k*norm(dk)*norm(subs(nabla_h));
74              j = 0;                  % setze j bei jedem Durchlauf gleich 0
75
76              h_k = g(1)^2 + g(2)^2 + g(3)^2; %h_k(x) ist norm(f(x))^2
77
```

```
78              while subs(h_k)>norm(fx0)^2-(2^-j)*z_k
79                      j=j+1;
80                      r=xk(1)-2^-j*dk(1);
81                      s=xk(2)-2^-j*dk(2);
82                      v=xk(3)-2^-j*dk(3);
83              end
84
85              m=zeros(1,j+1);          % m=min h_k(2^-i)
86              for i=0:1:j
87                  r=xk(1)-2^-i*dk(1);
88                  s=xk(2)-2^-i*dk(2);
89                  v=xk(3)-2^-i*dk(3);
90                  m(i+1)=subs(h_k);
91              end
92  % Sortiere m nach der Größe, da ich kleinstes m benötige
93              mi=sort(m);
94
95  % Berechne norm(f(xk-lambda*dk))^2-m und löse nach lambda auf
96 f=@(lambda)(((xk(1)-lambda*dk(1))^3*(P_300(1) - P_003(1) + P_102(1) -...
97     P_201(1))+(xk(2)-lambda*dk(2))^3*(P_030(1) - P_003(1) - P_021(1) +...
98     P_012(1))+(xk(2)-lambda*dk(2))^2*(3*P_003(1) - 2*P_012(1) + ...
99     P_021(1))+(xk(1)-lambda*dk(1))^2*(3*P_003(1) - 2*P_102(1) + ...
100    P_201(1))+(xk(1)-lambda*dk(1))^2*(xk(2)-lambda*dk(2))*(-3*P_003(1) +...
101    P_012(1) + 2*P_102(1)-P_201(1) + P_210(1) - P_111(1)) + ...
102    (xk(1)-lambda*dk(1))*(xk(2)-lambda*dk(2))^2*(-3*P_003(1)-...
103    P_021(1) + 2*P_012(1) + P_102(1) + P_120(1) - P_111(1))+...
104    (xk(1)-lambda*dk(1))*(xk(2)-lambda*dk(2))*(6*P_003(1) - 2*P_012(1) -...
105    2*P_102(1) + P_111(1))+(xk(1)-lambda*dk(1))*(-3*P_003(1) + ...
106    P_102(1)) + (xk(2)-lambda*dk(2))*(-3*P_003(1)+P_012(1))+...
107    P_003(1) - A(1) - (xk(3)-lambda*dk(3))*(X - A(1)))^2+...
108    ((xk(1)-lambda*dk(1))^3*(P_300(2) - P_003(2) + P_102(2) - P_201(2))+...
109    (xk(2)-lambda*dk(2))^3*(P_030(2) - P_003(2) - P_021(2) + P_012(2))+...
110    (xk(2)-lambda*dk(2))^2*(3*P_003(2) - 2*P_012(2) + P_021(2))+...
111    (xk(1)-lambda*dk(1))^2*(3*P_003(2) - 2*P_102(2) + P_201(2))+...
112    (xk(1)-lambda*dk(1))^2*(xk(2)-lambda*dk(2))*(-3*P_003(2) + P_012(2)+...
113    2*P_102(2)-P_201(2) + P_210(2) - P_111(2)) + (xk(1)-lambda*dk(1))*...
114    (xk(2)-lambda*dk(2))^2*(-3*P_003(2)-P_021(2) + 2*P_012(2) + ...
115    P_102(2)+ P_120(2) - P_111(2))+(xk(1)-lambda*dk(1))*(xk(2)-...
116    lambda*dk(2))*(6*P_003(2) - 2*P_012(2) - 2*P_102(2) + P_111(2))+...
117    (xk(1)-lambda*dk(1))*(-3*P_003(2) + P_102(2))+(xk(2)-lambda*dk(2))*...
118    (-3*P_003(2)+P_012(2))+P_003(2) - A(2) - (xk(3)-lambda*dk(3))*...
119    (Y - A(2)))^2+((xk(1)-lambda*dk(1))^3*(P_300(3) - P_003(3) + ...
120    P_102(3) - P_201(3))+(xk(2)-lambda*dk(2))^3*(P_030(3) - P_003(3) -...
```

```
121     P_021(3) + P_012(3))+(xk(2)-lambda*dk(2))^2*(3*P_003(3)-2*P_012(3)+...
122     P_021(3))+(xk(1)-lambda*dk(1))^2*(3*P_003(3) - 2*P_102(3)+P_201(3))+...
123     (xk(1)-lambda*dk(1))^2*(xk(2)-lambda*dk(2))*(-3*P_003(3) + P_012(3)+...
124     2*P_102(3)-P_201(3) + P_210(3) - P_111(3)) + (xk(1)-lambda*dk(1))*...
125     (xk(2)-lambda*dk(2))^2*(-3*P_003(3)-P_021(3) + 2*P_012(3)+P_102(3)+...
126     P_120(3) - P_111(3))+(xk(1)-lambda*dk(1))*(xk(2)-lambda*dk(2))*...
127     (6*P_003(3) - 2*P_012(3) - 2*P_102(3) + P_111(3))+...
128     (xk(1)-lambda*dk(1))*(-3*P_003(3) + P_102(3))+(xk(2)-lambda*dk(2))*...
129     (-3*P_003(3)+P_012(3))+P_003(3)-A(3)-(xk(3)-lambda*dk(3))*...
130     (Z - A(3)))^2)-mi(1);
131
132     lam=fzero(f,[0,1]);
133
134     % Berechne neues xk für nächsten Iterationsschritt
135
136     xk=xk-lam*dk;
137     r=xk(1);
138     s=xk(2);
139     v=xk(3);
140     fx0=[subs(f1);subs(f2);subs(f3)];
141     n=norm(fx0);
142          if norm(fx0)<epsilon
143            x(:,l)=xk;
144            nfx0(1,l)=norm(fx0);
145        else
146            x(1,l)=10;
147            nfx0(1,l)=norm(fx0);
148        end
149        end
150     end
151 end
152 toc
```

5.7 function normalv

Über die Funktion NORMALV wird zuerst die Matrix der Schnittpunkte mit dem Bézier-Patch aus dem Ergebnis des modifizierten Newton-Verfahrens gewonnen. Anschließend wird wie in (4.12) beschrieben der Normalvektor am Schnittpunkt bestimmt und zudem der Richtungsvektor des Lichtstrahls vor der Reflexion berechnet. Beides wird für die Durch-

führung der Reflexion nötig sein.

```
function [N,Rx,Ry,Rz,MaSp] = normalv(x,A,XNE,P_003,P_030,P_300,f1dr,f1ds,...
    f2dr,f2ds,f3dr,f3ds,c)

% INPUT
% A.....Augpunkt
% XNE...Punkte auf der Kugel um A, die Schnittgeraden ergeben
% P_003,P_030,P_300...Begrenzungspunkte des Kontrollpolygons
% f1dr..Ableitung x-Koordinate nach r
% f1ds..Ableitung x-Koordinate nach s
% f2dr..Ableitung y-Koordinate nach r
% f2ds..Ableitung y-Koordinate nach s
% f3dr..Ableitung z-Koordinate nach r
% f3ds..Ableitung z-Koordinate nach s
% c.....Anzahl der Schnitte mit der konvexen Hülle
%
% OUTPUT
% MaSp..Schnittpunkte mit Bézier-Patch
% N.....Normalvektor am Schnittpunkt
% Rx....x-Koordinate des Richtungsvektor vor der Reflexion
% Ry....y-Koordinate des Richtungsvektor vor der Reflexion
% Rz....z-Koordinate des Richtungsvektor vor der Reflexion

tic

MaSp=zeros(c,3);

for j=1:1:c
    if x(3,j)≠0
        MaSp(j,1)=A(1)+x(3,j)*(XNE(j,1)-A(1));
        MaSp(j,2)=A(2)+x(3,j)*(XNE(j,2)-A(2));
        MaSp(j,3)=A(3)+x(3,j)*(XNE(j,3)-A(3));
    end
end

% Berechne Baryzentrische Koordinaten der Schnittpunkte und setze sie
% in die partiellen Ableitungen ein. Damit bilde ich über das
% Kreuzprodukt die Normlavektoren an die Schnittpunkte.

barysp=zeros(c,3);      % Baryzentrische Koord des SP
B_dr=zeros(3,c);
B_ds=zeros(3,c);
```

```
42 N=zeros(3,c);
43
44 for j=1:1:c
45         barysp(j,1)=((P_003(2)-P_030(2))*MaSp(j,1)+...
46             (P_030(1)-P_003(1))*MaSp(j,2)+P_003(1)*P_030(2)-...
47             P_003(2)*P_030(1))/((P_003(2)-P_030(2))*P_300(1)+...
48             (P_030(1)-P_003(1))*P_300(2)+P_003(1)*P_030(2)-...
49             P_003(2)*P_030(1));
50
51         barysp(j,2)=((P_003(2)-P_300(2))*MaSp(j,1)+...
52             (P_300(1)-P_003(1))*MaSp(j,2)+P_003(1)*P_300(2)-...
53             P_300(1)*P_003(2))/((P_003(2)-P_300(2))*P_030(1)+...
54             (P_300(1)-P_003(1))*P_030(2)+P_003(1)*P_300(2)-...
55             P_300(1)*P_003(2));
56
57         barysp(j,3)=1-barysp(j,1)-barysp(j,2);
58
59         r=barysp(j,1);
60         s=barysp(j,2);
61
62         B_dr(1,j)=subs(f1dr);
63         B_dr(2,j)=subs(f2dr);
64         B_dr(3,j)=subs(f3dr);
65         B_ds(1,j)=subs(f1ds);
66         B_ds(2,j)=subs(f2ds);
67         B_ds(3,j)=subs(f3ds);
68
69         B1=B_dr(:,j);
70         B2=B_ds(:,j);
71         N(:,j)=cross(B1,B2);        % Normalvektor über Kreuzprodukt
72
73 end
74
75 % Richtungsvektoren
76 % Berechne die zur Rotation notwendigen Richtungsvektoren der
77 % Schnittgeraden.
78
79 Rx=zeros(c,1);
80 Ry=zeros(c,1);
81 Rz=zeros(c,1);
82
83 for j=1:1:c
84     Rx(j)=XNE(j,1)-A(1);
```

```
85      Ry(j)=XNE(j,2)-A(2);
86      Rz(j)=XNE(j,3)-A(3);
87  end
88
89  toc
```

5.8 function rotpi

Die Funktion ROTPI berechnet die Reflexion als Rotation um π mit dem Normalvektor als Rotationsachse (siehe 4.13). Zudem lässt sich mit dem zweiten Teil noch überprüfen, ob die Richtungsvektoren vor und nach der Reflexion tatsächlich in einer Ebene liegen.

```
1   function [ref] = rotpi(N,Rx,Ry,Rz,c)
2
3   % INPUT
4   % N.....Normalvektor am Schnittpunkt
5   % Rx....x-Koordinate des Richtungsvektor vor der Reflexion
6   % Ry....y-Koordinate des Richtungsvektor vor der Reflexion
7   % Rz....z-Koordinate des Richtungsvektor vor der Reflexion
8   % c.....Anzahl der Schnitte mit der konvexen Hülle
9   %
10  % OUTPUT
11  % ref...Matrix der Richtungsvektoren nach der Rotation
12
13  tic
14
15  Rot=zeros(c,9);
16  ∆=pi;
17
18  for j=1:1:c
19
20  Rot(j,1)=cos(∆)+N(1,j)^2*(1-cos(∆));
21  Rot(j,2)=N(2,j)*N(1,j)*(1-cos(∆))+N(3,j)*sin(∆);
22  Rot(j,3)=N(3,j)*N(1,j)*(1-cos(∆))-N(2,j)*sin(∆);
23  Rot(j,4)=N(1,j)*N(2,j)*(1-cos(∆))-N(3,j)*sin(∆);
24  Rot(j,5)=cos(∆)+N(2,j)^2*(1-cos(∆));
25  Rot(j,6)=N(3,j)*N(2,j)*(1-cos(∆))+N(1,j)*sin(∆);
26  Rot(j,7)=N(1,j)*N(3,j)*(1-cos(∆))+N(2,j)*sin(∆);
27  Rot(j,8)=N(2,j)*N(3,j)*(1-cos(∆))-N(1,j)*sin(∆);
```

```
28 Rot(j,9)=cos(a)+N(3,j)^2*(1-cos(a));
29
30 end
31
32 % Errechne die Richtungsvektoren ref nach der Reflexion
33 % durch Multiplikation mit den Rotationsmatrizen
34
35 ref=zeros(c,3);
36
37 for j=1:1:c
38
39     F=[Rot(j,1), Rot(j,4), Rot(j,7); Rot(j,2), Rot(j,5), Rot(j,8);
40     Rot(j,3), Rot(j,6),Rot(j,9);];
41     C=[Rx(j);Ry(j);Rz(j)];
42     ref(j,:)=F*C;
43
44 end
45
46 toc
47
48 % Überprüfe ob RV ref auch wirklich in der Einfallsebene liegt
49
50 % N=(NormEB(1,:))';
51 % P=[MaSpX(1);MaSpY(1);MaSpZ(1)];
52 % NP=N.*P;
53 % Norm1=norm(NP);
54 % F=(ref(1,:))';
55 % NL=P+F;
56 % NL1=N.*NL;
57 % Norm2=norm(NL1);
```

5.9 function schnittbox

In der Funktion SCHNITTBOX werden die endgültigen Ausgabeparamater SPX,SPZ,SPZ1,SPY und SPY1 berechnet. Sie ergeben sich als Schnittpunkt der reflektierten Lichtstrahlen mit den Seitenwänden der Box.

```
1 function [SpX,SpZ,SpZ1,SpY,SpY1] = schnittbox(MaSp,ref,c)
2
```

```
3  % INPUT
4  % ref...Matrix der Richtungsvektoren nach der Rotation
5  % MaSp..Schnittpunkte mit Bézier-Patch
6  % c.....Anzahl der Schnitte mit der konvexen Hülle
7  %
8  % OUTPUT
9  % SpX....Schnittpunkte nach der Reflexion mit der y-z-Ebene
10 % SpZ....Schnittpunkte nach der Reflexion mit der x-y-Ebene
11 % SpZ1...Schnittpunkte nach der Reflexion mit der Ebene z=10.
12 % SpY....Schnittpunkte nach der Reflexion mit der x-z-Ebene
13 % SpY1...Schnittpunkte nach der Reflexion mit der Ebene y=10.
14
15 tic
16
17 % Beginne mit dem Schnitt mit Front der (d.h. y-z-Ebene) d.h. x=0
18 % Speichere die Schnittpunktkoordinaten in SpX
19
20 SpX=zeros(3,c);
21 u_yz=0;
22
23 for j = 1:1:c
24     ts = (-MaSp(j,1))*1/ref(j,1);
25     if ts > 0
26         u_yz=u_yz+1;
27         SpX(1,j)=MaSp(j,1) + ts*ref(j,1);
28         SpX(2,j)=MaSp(j,2) + ts*ref(j,2);
29         SpX(3,j)=MaSp(j,3) + ts*ref(j,3);
30     end
31 end
32
33 % Schnitt mit dem Boden (d.h. x-y-Ebene) d.h. z=0
34 % Speichere die Schnittpunktkoordinaten in SpZ
35
36 u_xy=0;
37 SpZ=zeros(3,j);
38
39 for j = 1:1:c
40     ts = (-MaSp(j,3))*1/ref(j,3);
41     if ts > 0
42         u_xy=u_xy+1;
43         SpZ(1,j)=MaSp(j,1) + ts*ref(j,1);
44         SpZ(2,j)=MaSp(j,2) + ts*ref(j,2);
45         SpZ(3,j)=MaSp(j,3) + ts*ref(j,3);
```

```
46      end
47  end
48
49  % Schnitt mit der Decke, d.h. z=10
50  % Speichere die Schnittpunktkoordinaten in SpZ
51
52  u_xy=0;
53  SpZ1=zeros(3,j);
54
55  for j = 1:1:c
56      ts = (10-MaSp(j,3))*1/ref(j,3);
57      if ts > 0
58          u_xy=u_xy+1;
59          SpZ1(1,j)=MaSp(j,1) + ts*ref(j,1);
60          SpZ1(2,j)=MaSp(j,2) + ts*ref(j,2);
61          SpZ1(3,j)=MaSp(j,3) + ts*ref(j,3);
62      end
63  end
64
65  % Schnitt mit der Seitenwand 1 (d.h. y=10)
66  % Speichere die Schnittpunktkoordinaten in SpY1
67
68  u_y1=0;
69  SpY1=zeros(3,j);
70
71  for j = 1:1:c
72      ts = (10-MaSp(j,2))*1/ref(j,2);
73      if ts > 0
74          u_y1=u_y1+1;
75          SpY1(1,j)=MaSp(j,1) + ts*ref(j,1);
76          SpY1(2,j)=MaSp(j,2) + ts*ref(j,2);
77          SpY1(3,j)=MaSp(j,3) + ts*ref(j,3);
78      end
79  end
80
81  % Schnitt mit Seitenwand 2, der x-z-Ebene (d.h. y=0)
82  % Speichere die Schnittpunktkoordinaten in SpY
83
84  u_y=0;
85  SpY=zeros(3,j);
86
87  for j = 1:1:c
88      ts = (-MaSp(j,2))*1/ref(j,2);
```

```
89      if ts > 0
90          u_y=u_y+1;
91          SpY(1,j)=MaSp(j,1) + ts*ref(j,1);
92          SpY(2,j)=MaSp(j,2) + ts*ref(j,2);
93          SpY(3,j)=MaSp(j,3) + ts*ref(j,3);
94      end
95  end
96
97  toc
```

Index

Literaturverzeichnis

[1] Appel, A.: Some Techniques for Shading Machine Renderings of Solids. In Proceedings of the Spring Joint Computer Conference 1968, p. 37-45. AFIPS Press, Arlington

[2] Buss, S.R.: 3-D computer graphics: a mathematical introduction with OpenGL. Cambridge University Press (2003)

[3] Carathéodory, C.: Geometrische Optik. Springer Verlag Berlin (1937)

[4] Classen, J.W.: Mathematische Optik. Verlag Göschen Leipzig (1901)

[5] Cook, R. u.a.: Distributed ray tracing. ACM SIGGRAPH Computer Graphics 18, 3 (July 1984): p. 137-145

[6] De Boor, C.: A practical guide to splines. Springer New York (1978)

[7] Flügge, J.: Leitfaden der geometrischen Optik und des Optikrechnens. Vandenhoeck und Ruprecht in Göttingen (1956)

[8] Glassner, A.S. u.a.: An Intruduction to Raytracing. Academic Press Limited (1991)

[9] Heuser, H.: Lehrbuch der Analysis, Teil 1 (13. Auflage). B.G.Teuber Stuttgart (2000)

[10] Heuser, H.: Lehrbuch der Analysis, Teil 2 (11. Auflage). B.G.Teuber Stuttgart (2000)

[11] Meisel, F.: Geometrische Optik. Verlag Schmidt, Halle an der Saale (1886)

[12] Neumaier, A.: Introduction to Numerical Analysis. Cambridge University Press (2001)

[13] Obergugger, A. und Ostermann, A.: Analysis für Informatiker: Grundlagen, Methoden, Algorithmen. Springer-Verlag Berlin Heidelberg (2005)

[14] Risler, J.J.: Mathematical Methods for CAD. Cambridge University Press (1991)

[15] Rubin, S. and Whitted, T.: A three-dimensional represantation for fast rendering of complex scenes. Comput. Graph. 14(3), p.110-116 (July 1980)

[16] Schwarz, H.R. und Köckler N.: Numerische Mathematik (7. Auflage). Vieweg + Teubner (2009)

[17] Stein, U.: Einstieg in das Programmiern mit MATLAB. Carl Hanser Verlag (2007)

[18] Stoer, J.: Einführung in die Numerische Mathematik 1. Springer-Verlag, Heidelberger Taschenbücher Band 105 (1983)

[19] Wald,I. und Havran, V.: On building fast kd-Trees for Ray Tracing, and on doing that in O(NlogN). Proceedings of IEEE Symposium on Interactive Ray Tracing, p.61-69, Salt Lake City 2006

[20] Whitted, T.: An Improved Illumination Model for Shaded Display. Communications of the ACM 23, 6 (June 1980), p.343-349

Lebenslauf

Name:	Markus Wolfgang Pumberger
Geburtsdaten:	12.05.1981, Ried im Innkreis, Österreich
Familienstand:	ledig
Schulbildung:	
1987 - 1991:	Volksschule 1 in Ried im Innkreis
1991 - 1999:	Bundesgymnasium in Ried im Innkreis
Studium:	
2000 - 2010	Diplomstudium Mathematik, Universität Wien,
	Schwerpunkt Angewandte Mathematik und Scientific Computing
2002 - 2006	Betriebswirtschaftslehre, Wirtschaftsuniversität Wien
Berufliche Laufbahn:	
07. - 12. 1999:	Wintersteiger Gmbh, Ried im Innkreis
01. - 09. 2000:	Präsenzdienst, General Zehner Kaserne Ried
2004 - 2010:	Schülerhilfe Wien,
	Nachhilfe Mathematik (1060, 1120, 1160 Wien)
	Nachhilfe Physik (1120 Wien)
03. - 06. 2006:	Mitarbeit Mathspace Wien:
	Ausstellung: "Kurt Gödel - ein Jahrhundert"
Interessen:	Naturwissenschaften und Technik im weiten Sinn
	Sport - vor allem Fussball und Bergsteigen
	Reisen - im Speziellen Mittelmeerraum
	Architektur - v.a. italienische Renaissance, Gotik und Moderne
	Musik - von Klassik bis Progressive Rock