Erzeugung und Verfeinerung von Finite-Elemente-Triangulierungen

Diplomarbeit

vorgelegt von
Joachim Kohlhammer
aus
Ulm

angefertigt am
Institut für
Numerische und Angewandte Mathematik
der
Georg-August-Universität zu Göttingen
2000

Inhaltsverzeichnis

Einleitung **1**

I Erzeugung einer Ausgangstriangulierung 5

1 Grundlegende Definitionen **7**
1.1 Simplizes 7
1.2 Triangulierung 11
1.3 Affine Transformationen 12

2 Die entwickelte Datenstruktur **15**
2.1 Anforderungen an die Datenstruktur 15
2.2 Der Aufbau einer Triangulierung im $\mathbb{R}^n$ 15
2.3 Zusammenfassung von Simplizes zu Triangulierungen 16
2.4 Die Hierarchie 17
2.5 Die implementierten Klassen im Detail 17
2.6 Beispielinhalt einer Hierarchie 19
2.7 Aufbau der Generatoren 21
2.8 Erweiterungsmöglichkeiten 22

3 Die Advancing-Front-Methode **25**
3.1 Das Verfahren 26
3.2 Die Konstruktion im $\mathbb{R}^2$ 26
3.3 Anmerkungen 30

4 Die Delaunay-Methode **33**
4.1 Definitionen 33
4.2 Voronoi-Diagramme 34
4.3 Das Delaunay-Lemma 36
4.4 Inkrementelle Methode und Delaunay-Kern 37
4.5 Überführung in eine Delaunay-Triangulierung 40
4.6 Nebenbedingungen 41
4.7 Die Erzeugung der leeren Triangulierung 44
4.8 Generierung der inneren Punkte 45

4.9 Der Algorithmus im Überblick 47
4.10 Alternative Methoden 47

5 Verbesserung der Qualität einer Triangulierung im $I\!R^2$ 49
5.1 Anforderungen an einen Glättungs-Operator 49
5.2 Topologische Operatoren 49
5.3 Geometrische Operatoren 50
5.4 Globale Anwendung eines lokalen Glätters 52
5.5 Verbesserung im $I\!R^3$ 52

II Verfeinerung einer Triangulierung 53

6 Grundlagen der Verfeinerung 55
6.1 Grundlagen 55

7 Verfeinerung bestehender Triangulierungen im $I\!R^2$ 59
7.1 Rot-Grün–Verfeinerung in zwei Dimensionen 59
7.2 Weitere Verfahren im $I\!R^2$ 61

8 Der Algorithmus von Freudenthal 63
8.1 Die Kuhn-Triangulierung 63
8.2 Verfeinerung der Kuhn-Triangulierung 67

9 Realisierung der Verfeinerung 75
9.1 Verfeinerung für mehr als zwei Raumdimensionen 75
9.2 Markieren der Simplizes 75
9.3 Verfeinerung der Komponenten einer Triangulierung 76
9.4 Der globale Algorithmus 78

10 Ergebnisse 81
10.1 Einfluß der Parameter auf die Generatoren 81
10.2 Unterschiede zwischen den Generatoren 88
10.3 Auswirkungen der Glätter 97
10.4 Die Verfeinerung 106
10.5 Fazit 108

A Programmierhandbuch 109
A.1 Die Klasse `indexSetC` 109
A.2 Die Geometrie-Bibliothek 111
A.3 Die **meshMan**-Bibliothek 115
A.4 Die Generator-Bibliothek 124
A.5 Die IO-Bibliothek 127
A.6 Ausnahmebehandlung 129

A.7 Abhängigkeiten der Bibliotheken 130

Einleitung

Partielle Differentialgleichungen haben sich als wichtige Komponenten zur Modellierung technischer Fragestellungen wie Temperatur- oder Druckverteilung im Raum erwiesen.
Für solche Gleichungen lassen sich in den seltensten Fällen geschlossene Lösungen angeben. Somit werden andere, numerische Verfahren benötigt. Eines der wichtigsten und verbreitetsten Verfahren ist das der finiten Elemente. Alle Strategien dieser Familie lassen sich grob in drei Schritten beschreiben:

1. Eingabe und Beschreibung des Ausgangsproblems, Generierung einer Triangulierung.

2. Erzeugung und Lösung des endlichdimensionalen algebraischen Problems, Abschätzung des Fehlers, evtl. Verfeinerung der Triangulierung und Wiederholung dieses Schritts.

3. Aufbereitung der erhaltenen Ergebnisse, Ableitung unmittelbarer Resultate und graphische Darstellung.

Die vorliegende Arbeit setzt sich mit der Generierung und Verfeinerung von Triangulierungen nach Punkt 1 und 2 auseinander. Zum einen muß das gegebene Gebiet, auf dem eine Lösung gefunden werden soll, in eine Menge kleinerer Gebiete zerlegt werden, zum anderen sollen diese Teilgebiete weiter zerlegt werden können. In den meisten Fällen, so auch in dieser Arbeit, handelt es sich dabei um Simplizes. Für den 2-dimensionalen Fall sind dies Dreiecke. Soweit das Gebiet geometrisch einfach ist, d.h. die Begrenzungsgeraden sind paarweise entweder orthogonal oder parallel zueinander, lassen sich mit relativ geringem Aufwand sogenannte strukturierte Triangulierungen angeben. Für den allgemeinen Fall polygonal berandeter Gebiete stellt die Generierung von Triangulierungen eine komplexe Aufgabe dar, zumal die resultierenden Dreiecke qualitativ nicht entarten sollen. Des weiteren muß darauf geachtet werden, daß keine hängenden Knoten während der Generierung entstehen. Mit jedem Simplex ist ein Maß verbunden, das als Qualität dieses Simplexes bezeichnet wird. Diese Qualität kann nach der Generierung mittels sogenannter Glättungsverfahren teilweise erheblich verbessert werden.
Wenn die Triangulierung vorliegt, ist noch nicht sichergestellt, daß der Fehler der daraus resultierenden Lösung innerhalb einer vorgegebenen Toleranz liegt. Durch

lokale Verfeinerung einzelner Elemente kann die Lösung verbessert werden. Bedarfsgerechte lokale Verfeinerung kann den Rechenaufwand im Vergleich zu Triangulierungen, die von vornherein global sehr fein angelegt sind, relativ gering halten. Die nachträgliche Verfeinerung soll folgende Bedingungen erfüllen:

- Die Triangulierung muß konform bleiben, d.h. es dürfen keine hängenden Knoten generiert werden.
- Die einzelnen Simplizes sollen beliebig tief verfeinert werden können, ohne dabei geometrisch zu entarten.
- Die Verfeinerungsstrategie sollte nicht auf den 2-dimensionalen Fall beschränkt sein.

Der Schwerpunkt der vorliegenden Arbeit liegt in der Generierung und lokalen Verfeinerung von Triangulierungen, sowie deren Implementierung. Die dafür entwickelte Datenstruktur speichert die Triangulierung — im Gegensatz zu den meisten anderen Implementierungen — unabhängig von der Raumdimension und ermöglicht darüberhinaus effizienten Zugriff auf die einzelnen Objekte der Triangulierung, ohne unnötig viel Speicherkapazität in Anspruch zu nehmen.
Für eine ausführliche Einführung in die Methode der finiten Elemente sei auf [19] verwiesen.
Im Rahmen dieser Diplomarbeit sollte ein Code zur Erstellung und Verfeinerung von Familien von Triangulierungen in zwei Dimensionen entwickelt werden. Die Erweiterbarkeit auf den 3-dimensionalen Fall sollte jedoch gewährleistet sein. Der Code sollte modular ausgelegt sein und ein genau definiertes API (Application Programming Interface) aufweisen, so daß einzelne Teile jederzeit ausgetauscht werden können. Eine weitere Voraussetzung war die Implementierung in der Sprache C++. Die Hauptgründe für C++ waren die Ermöglichung einer objektorientierten Programmierweise und die mächtige Standardbibliothek STL[13]. Ein weiterer Pluspunkt für C++ ist dessen weite Verbreitung.
Es existieren bereits sehr viele Codes zur Generierung oder Verfeinerung von Triangulierungen, warum also noch ein zusätzlicher? Zu Anfang dieser Arbeit stand eine ausführliche Untersuchung mehrerer Programme. Die meisten, wie zum Beispiel *NETGEN*[17], konnten aus Lizenzgründen nicht eingesetzt werden, da sie eigene Weiterentwicklungen oder das Linken gegen eigenen Code verboten. Übrig blieben stark spezialisierte Programme wie *Triangle*[7], *Delaundo*[2] oder *EasyMesh*[3]. Durch ihre Festlegung auf eine Anwendung — in diesem Fall Generierung von Triangulierungen im 2-dimensionalen Raum — beruhen sie auf einer starren, unflexiblen Datenstruktur, die nur unter sehr großem Aufwand zur Herstellung adaptiv verfeinerter Folgen von Triangulierungen oder für den 3-dimensionalen Fall genutzt werden kann.
Das meistversprechende Paket war AGM^{3D}[1]. Es enthält unter anderem eine Verfeinerungsstrategie für Tetraedertriangulierungen mit krummlinigem Rand. Diese Routinen wurden isoliert, in eine Bibliothek gepackt und von mir mitsamt der Datenstruktur an den 2D-Fall angepaßt. Dabei stellte sich heraus, daß die verwendete

Datenstruktur weder besonders übersichtlich noch effizient war. Der Aufwand, die 2D- und 3D-Versionen zu vereinigen, war aufgrund des teilweise undurchsichtigen Programmierstils (Bitoperationen auf Zeiger, die nicht immer und überall funktionierten) mit vielen Makros schlecht absehbar. Ein weiterer Minuspunkt für AGM^{3D} war die Implementierung in C.
Folglich wurde als Bestandteil dieser Arbeit eine komplett neue Anwendung entwickelt. Diese ist aus vier Bibliotheken aufgebaut:

- geom
  Elementare geometrische Operationen.
- **meshMan**
  Die Datenstruktur mit Zugriffsfunktionen und Verfeinerungsroutinen.
- meshIO
  Ein- und Ausgabefilter für die Triangulierungen. Diese Bibliothek enthält des weiteren einen Parser zum Einlesen einer Gebietsbeschreibung, der von E. Stafilarakis beigesteuert wurde.
- meshCreator
  Routinen zur Generierung einer Anfangstriangulierung.

Das Gesamtpaket wird mit **meshMan** bezeichnet.
Anhang A enthält eine Einführung in die Programmierung der oben genannten Bibliotheken.
Im folgenden wird des öfteren Bezug auf Typen genommen werden. Namen von Klassen und einfachen Typen werden in der Form `::`*`Namespace`*`::`*`Typname`* angegeben, ist kein Namespace vorhanden nur mit *`Typname`*. Klassen enden mit dem Suffix `C`, einfache Typen mit `T`.
Beispiele:

- `::base::indexT` — vorzeichenbehafteter, ganzzahliger Typ
- `::base::floatT` — vorzeichenbehafteter Fließkommatyp
- `indexSetC` — spezialisierter Vektor
- `::geom::pointC` — geometrischer Punkt
- `::net::netC` — eine Triangulierung

Teil I

Erzeugung einer Ausgangstriangulierung

Kapitel 1

Grundlegende Definitionen

Dieses Kapitel soll dazu dienen, einige für diese Arbeit grundlegende Begriffe einzuführen. Die meisten Beweise zu den Lemmata können [10] und [14] entnommen werden.

1.1 Simplizes

Definition 1.1 (konvexe Hülle)
Sei $M = \{x^{(0)}, \ldots, x^{(k-1)}\}$ eine Menge von k Punkten im $\mathbb{R}^n$. Dann ist die konvexe Hülle von M definiert durch

$$\mathcal{C}onv(M) := \left\{ x = \sum_{i=0}^{k-1} \lambda_i x^{(i)} \;\middle|\; \lambda_i \geq 0,\ 0 \leq i < k;\ \sum_{i=0}^{k-1} \lambda_i = 1 \right\}.$$

Die konvexe Hülle ist die kleinste konvexe Teilmenge des $\mathbb{R}^n$, die alle Punkte aus M enthält.

Definition 1.2 (Simplex)
Sei $n \in \mathbb{N}_0$ und $0 \leq k \leq n$. Eine abgeschlossene Menge $T \subset \mathbb{R}^n$ heißt (k)-Simplex im $\mathbb{R}^n$, falls gilt

$$T = \left[x^{(0)}, \ldots, x^{(k)}\right] := \mathcal{C}onv\{x^{(0)}, \ldots, x^{(k)}\}.$$

Im Fall $k = n$ heißt T auch Simplex. Die Punkte $x^{(j)}$, $0 \leq j \leq k$ heißen Eckpunkte von T.

(2)- und (3)-Simplizes werden üblicherweise als Dreiecke und Tetraeder bezeichnet. Der Rand eines (k)-Simplex T besteht aus niederdimensionalen Randsimplizes. Deren Eckpunkte sind eine Teilmenge der Eckpunkte von T.

Definition 1.3 (Randsimplex)
Sei $T = \left[x^{(0)}, \ldots, x^{(k)}\right]$ ein (k)-Simplex im $\mathbb{R}^n$, $0 < k \leq n$. Ein (l)-Simplex S, $0 \leq l < k$, heißt (l)-Randsimplex von T, falls Indizes $i_0, \ldots, i_l$ mit $0 \leq i_0 < \cdots < i_l \leq k$ existieren, so daß $S = \left[x^{(i_0)}, \ldots, x^{(i_l)}\right]$ gilt.

Die (0)-Randsimplizes von T sind die Eckpunkte; die (1)-Randsimplizes werden als Segmente bezeichnet. Die Anzahl der (l)-Randsimplizes von T ist gegeben durch $\binom{k+1}{l+1}$. Ein (k)-Simplex besitzt also $\frac{k(k+1)}{2}$ (1)-Simplizes.
Das k-dimensionale Volumen eines (k)-Simplexes T wird mit vol_T bezeichnet.

Lemma 1.1
Sei $T = \left[x^{(0)}, \ldots, x^{(n)}\right]$ ein Simplex im $\mathbb{R}^n$. Dann gilt für das Volumen vol_T von T

$$vol_T = \frac{|\det B_T|}{n!}$$

mit der Matrix

$$B_T := \begin{pmatrix} x_1^{(0)} & x_1^{(1)} & \cdots & x_1^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(0)} & x_n^{(1)} & \cdots & x_n^{(n)} \\ 1 & 1 & \cdots & 1 \end{pmatrix}. \tag{1.1}$$

Aus Lemma 1.1 folgt, daß $\mathrm{vol}_T = 0$ genau dann gilt, wenn die Matrix B_T singulär ist. Geometrisch bedeutet dies, daß die Eckpunkte von T auf einer gemeinsamen $(n-1)$-dimensionalen Hyperebene liegen. Solche Simplizes sind im allgemeinen unerwünscht und werden als entartet bezeichnet.

Definition 1.4 (Baryzentrische Koordinaten)
Sei $T = \left[x^{(0)}, \ldots, x^{(n)}\right]$ ein nichtentartetes Simplex und x ein beliebiger Punkt im $\mathbb{R}^n$. Dann existieren eindeutige reelle Zahlen $\lambda_0, \ldots, \lambda_n$ mit

$$x = \sum_{j=0}^{n} \lambda_j x^{(j)}, \qquad \sum_{j=0}^{n} \lambda_j = 1. \tag{1.2}$$

Die Zahlen $\lambda_0, \ldots, \lambda_n$ heißen die baryzentrischen Koordinaten von x bzgl. T.

Da jedes Simplex die lineare Hülle seiner Eckpunkte ist, folgt:

Lemma 1.2
Sei $T = \left[x^{(0)}, \ldots, x^{(n)}\right] \subset \mathbb{R}^n$ ein nichtentartetes Simplex, x ein beliebiger Punkt im $\mathbb{R}^n$. So gilt $x \in T$ genau dann, wenn die baryzentrischen Koordinaten $\lambda_0, \ldots, \lambda_n$ von x bzgl. T sämtlich nichtnegativ sind.

Ein Simplex T ist unabhängig von der Reihenfolge seiner Eckpunkte.

Definition 1.5 (Gleichheit von Simplizes)
Seien $T = \left[x^{(0)}, \ldots, x^{(k)}\right]$ und $T' = \left[y^{(0)}, \ldots, y^{(k)}\right]$ zwei (k)-Simplizes im $\mathbb{R}^n$. T und T' heißen gleich, falls eine Permutation π der Zahlen $\{0, \ldots, k\}$ existiert mit $y^{(j)} = x^{(\pi(j))}$, $0 \leq j \leq k$. In diesem Fall schreibt man $T = T'$.

Definition 1.6 (Außenkugel eines Simplex)
Sei T ein Simplex im $I\!R^n$. Dann heißt die kleinste Kugel K, die T komplett enthält, Außenkugel von T. Der Radius von K wird mit ρ_T bezeichnet.

Definition 1.7 (Innenkugel eines Simplex)
Sei T ein Simplex im $I\!R^n$. Dann heißt die größte Kugel K, die komplett in T enthalten ist, Innenkugel von T. Der Radius von K wird mit h_T bezeichnet.

Bemerkung 1.1
Sei T ein Simplex im $I\!R^2$. Dann wird die Außenkugel von T mit *Umkreis*, seine Innenkugel mit *Inkreis* bezeichnet.
Der Umkreismittelpunkt von T ist gegeben durch den Schnittpunkt der Mittelsenkrechten von T, der Umkreisradius ist dann der Abstand dieses Mittelpunktes zu einem der drei Eckpunkte von T.
Der Schnittpunkt der drei Winkelhalbierenden von T ist identisch mit dem Inkreismittelpunkt. □

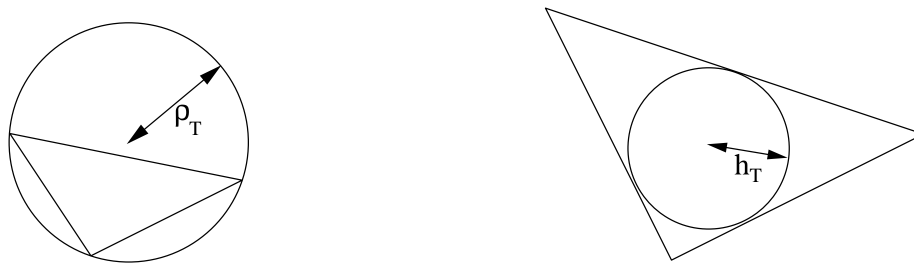

Abbildung 1.1: Umkreis und Inkreis

Lemma 1.3
Sei T ein Simplex im $I\!R^2$. Bezeichne l_i, $i = 1, \dots, 3$ die Längen der drei Segmente von T und $vol_T T$ die Fläche von T. Dann ergibt sich der Umkreisradius von T aus

$$\rho_T = \frac{l_1 l_2 l_3}{4 vol_T}.$$

Lemma 1.4
Sei T ein Simplex im $I\!R^2$. Seien l_i und vol_T wie in Lemma 1.3. Sei $p_T := \sum_{i=1}^{3} l_i$ der Umfang von T. Dann ist der Inkreisradius von T gegeben durch

$$h_T = \frac{vol_T}{p_T}.$$

Definition 1.8
Sei $T = \left[x^{(0)}, \ldots, x^{(n)}\right] \subset I\!R^n$ *ein nichtentartetes Simplex. Dann heißt*

$$d_T := \max_{0 \le i,j \le n} dist(x^{(i)}, x^{(j)})$$

der Elementdurchmesser von T*, wobei* $dist(x, y)$ *den euklidischen Abstand zweier Punkte im* $I\!R^n$ *bezeichnet.*

Definition 1.9 (Qualitätsmaß eines Simplex)
Sei T *ein Simplex im* $I\!R^2$*.* $\mathcal{Q}_T$ *ist ein Qualitätsmaß, falls* $\mathcal{Q}_T \ge 1$ *und*

- $\mathcal{Q}_T = 1$ *genau dann, wenn* T *gleichseitig ist,*
- $\mathcal{Q}_T = \infty$ *genau dann, wenn* $vol_T = 0$*. In diesem Fall ist* T *entartet.*

Beispiel 1.1
Sei T ein Simplex im $I\!R^2$. ρ_T bezeichne den Umkreis-, h_T den Inkreisradius, vol_T die Fläche von T. l_i, $i = 1, \ldots, 3$, sei die Länge von Segment i. $p_T := \sum_{i=1}^{3} l_i$ bezeichne den Umfang des Dreiecks, d_T dessen Durchmesser.
Es gibt unterschiedliche Qualitätsmaße für die Beurteilung von T. Einige davon sind:

- $\mathcal{Q}_T := \alpha \frac{\rho_T}{h_T}$

  Das Verhältnis von Umkreis- zu Inkreisradius mit Normalisierungsfaktor $\alpha = \frac{1}{2}$. Dieses Kriterium wurde in der vorliegenden Implementierung verwendet.

- $\mathcal{Q}_T := \alpha \frac{d_T}{h_T}$

  Das Verhältnis zwischen Element-Durchmesser und Inkreisradius mit Normalisierungsfaktor $\alpha = \frac{\sqrt{3}}{6}$.

- $\mathcal{Q}_T := \alpha \frac{\sum_{i=1}^{3} l_i^2}{\mathrm{vol}_T}$

  Mit Normalisierungsfaktor $\alpha = \frac{\sqrt{3}}{12}$.

Bemerkung 1.2
Die obige Definition der Qualität kann auf andere Kriterien als die Gleichseitigkeit genormt werden. Eine Erweiterungsmöglichkeit besteht in der Kombination des Volumens eines Simplexes mit einem Maß für seine Form.
Eine Erweiterung für das Qualitätsmaß im $I\!R^n$ erfolgt analog. □

1.2 Triangulierung

Definition 1.10 (Polyeder)
Sei $P \subset \mathbb{R}^n$ eine abgeschlossene, zusammenhängende Menge. P heißt Polyeder, wenn P als endliche Vereinigug von Simplizes dargestellt werden kann. Ein Gebiet $\Omega \subset \mathbb{R}^n$ heißt polyedrisch, falls $\overline{\Omega}$ ein Polyeder ist und der Rand von Ω als endliche Vereinigung von $(n-1)$-Simplizes dargestellt werden kann.

Definition 1.11 (simpliziale Überdeckung)
Sei $\Omega \subset \mathbb{R}^n$ ein polyedrisches Gebiet. Eine endliche Menge $\mathcal{T}$ von Simplizes $T \subset \overline{\Omega}$ heißt simpliziale Überdeckung von Ω, falls gilt:

- $vol_T > 0$, *für alle* $T \in \mathcal{T}$,
- $\bigcup_{T \in \mathcal{T}} \overline{T} = \overline{\Omega}$
- $\mathring{T} \cap \mathring{T}' = \emptyset$, *für alle* $T, T' \in \mathcal{T}$ *mit* $T \neq T'$.

Definition 1.12 (Triangulierung)
Sei $\mathcal{T}$ eine simpliziale Überdeckung eines polyedrischen Gebietes $\Omega \subset \mathbb{R}^n$. Dann ist $\mathcal{T}$ eine (konforme) Triangulierung von Ω, falls der Schnitt zweier Simplizes $T, T' \in \mathcal{T}$, $T \neq T'$ entweder die leere Menge oder ein (k)-Randsimplex $(k < n)$ von T und T' ist.

Bemerkung 1.3
Ist eine simpliziale Überdeckung nicht konform, so spricht man auch von einer *nicht-konformen Triangulierung.* □

Bemerkung 1.4 (hängende Knoten)
Eine nicht-konforme Triangulierung zeichnet sich durch sogenannte *hängende Knoten* aus, siehe Abbildung 1.2. □

Abbildung 1.2: konforme und nicht-konforme Triangulierung

Definition 1.13 (Nachbarn)
Sei $\mathcal{T}$ eine Triangulierung eines polyedrischen Gebietes $\Omega \subset \mathbb{R}^n$. Dann heißen $T, T' \in \mathcal{T}$ Nachbarn, falls gilt $T \cap T' = S$, wobei S ein $(n-1)$-Randsimplex von T und T' ist.

Definition 1.14 (Qualität einer Triangulierung)
Sei $\mathcal{T}$ Triangulierung eines polyedrischen Gebietes $\Omega \subset \mathbb{R}^n$. Die Größe

$$\mathcal{Q}_{\mathcal{T}} := \max_{T \in \mathcal{T}} \mathcal{Q}_T$$

heißt Qualität von $\mathcal{T}$.

1.3 Affine Transformationen

Definition 1.15
Unter einer affinen Transformation im $\mathbb{R}^n$ versteht man eine Abbildung $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ der Form

$$F : x \mapsto v + Bx, \qquad x, v \in \mathbb{R}^n, \qquad B \in \mathbb{R}^{n \times n}$$

wobei B eine nichtsinguläre Matrix ist. Man schreibt statt $F(x)$ auch Fx.

Lemma 1.5 (Eigenschaften affiner Transformationen)
Sei F eine affine Transformation mit $F : x \mapsto v + Bx$, $x \in \mathbb{R}^n$. Dann gilt

1. *F ist bijektiv und die Umkehrabbildung $F^{-1} : x \mapsto B^{-1}(x - v)$ ist ebenfalls eine affine Transformation.*
2. *Geraden im $\mathbb{R}^n$ werden durch F wieder auf Geraden abgebildet.*
3. *Der Mittelpunkt der Verbindungsstrecke zwischen zwei Punkten $x^{(0)}, x^{(1)} \in \mathbb{R}^n$ wird durch F auf den Mittelpunkt der Strecke von $Fx^{(0)}$ nach $Fx^{(1)}$ abgebildet.*

Ist F eine affine Transformation und M eine beliebige Teilmenge des $\mathbb{R}^n$, so ist die transformierte Menge $M' = F(M)$ definiert durch

$$F(M) := \{Fx \mid x \in M\}.$$

Definition 1.16 (affine Transformation von Simplizes)
Sei $F : x \mapsto v + Bx$ eine affine Transformation im $\mathbb{R}^n$. Für ein (k)-Simplex $T = \left[x^{(0)}, \ldots, x^{(k)}\right]$ im $\mathbb{R}^n$, $0 \leq k \leq n$, wird das transformierte (k)-Simplex $T' = F(T)$ durch

$$F(T) := \left[Fx^{(0)}, \ldots, Fx^{(k)}\right]$$

definiert. Man verwendet auch die Schreibweise $F(T) = v + BT$. Ist B ein Vielfaches der Einheitsmatrix, d.h. $B = cI$, $c \neq 0$, so schreibt man statt BT auch cT.

Aus Lemma 1.5 (2) folgt, daß jedes (k)-Simplex T im $\mathbb{R}^n$ durch eine affine Transformation F wieder auf ein (k)-Simplex $F(T)$ abgebildet wird. Da F per Definition nichtsingulär ist, ist $F(T)$ genau dann entartet, wenn T entartet ist.
Die affinen Transformationen werden im folgenden hauptsächlich dazu verwendet, für ein Referenzsimplex T — ein beliebiges aber festes nichtentartetes Simplex im $\mathbb{R}^n$ — hergeleitete Eigenschaften auf beliebige Simplizes zu übertragen.

Lemma 1.6
Sei T ein nichtentartetes Simplex im $\mathbb{R}^n$ und $F : x \mapsto v + Bx$ eine affine Transformation. Dann gelten für das transformierte Simplex $T' = F(T)$ die Beziehungen

$$\|B\| \leq \frac{d_{T'}}{\rho_T}, \qquad \|B^{-1}\| \leq \frac{d_T}{\rho_{T'}}, \qquad |\det B| = \frac{vol_{T'}}{vol_T},$$

wobei $\|B\|$ die euklidische Norm der Matrix B ist.

Definition 1.17 (Ähnlichkeit von Simplizes)
Zwei Simplizes T, T' im $\mathbb{R}^n$ heißen einander ähnlich, falls eine orthogonale Matrix $Q \in \mathbb{R}^{n\times n}$, ein Translationsvektor $v \in \mathbb{R}^n$ und ein Skalierungsfaktor $c > 0$ existieren, so daß

$$T' = v + cQT$$

gilt. In diesem Fall gehören T und T' zur gleichen Ähnlichkeitsklasse.

Definition 1.18 (affine Transformation von Triangulierungen)
Sei $\mathcal{T}$ Triangulierung eines polyedrischen Gebietes $\Omega \subset \mathbb{R}^n$ und F eine affine Transformation. Dann ist die transformierte Triangulierung $F(\mathcal{T})$ definiert durch

$$F(\mathcal{T}) := \{F(T) \mid T \in \mathcal{T}\}$$

Lemma 1.7
Sei $\mathcal{T}$ simpliziale Überdeckung eines polyedrischen Gebietes $\Omega \subset \mathbb{R}^n$ und F eine affine Transformation. Dann ist $F(\mathcal{T})$ eine konforme Triangulierung genau dann, wenn $\mathcal{T}$ eine konforme Triangulierung ist.

Kapitel 2

Die entwickelte Datenstruktur

2.1 Anforderungen an die Datenstruktur

Die Datenstruktur ist das zentrale Element der Implementierung. Sie soll den einfachen und effizienten Zugriff auf die gewünschten Objekte und Daten ermöglichen. Daher muß sie einige Anforderungen erfüllen.
Der erste Punkt, der bei der Planung beachtet werden mußte, war die beabsichtigte Dimensionsunabhängigkeit. Es sollen also mit einer einzigen Struktur Triangulierungen in beliebigen Dimensionen möglich sein.
In der Struktur sollen auch Folgen adaptiv verfeinerter Triangulierungen gespeichert werden können. Ein Level soll sich dabei selbst verfeinern können.
Die Generierung der Triangulierung soll auf Wunsch automatisch erfolgen können, ohne daß ein spezieller Generator vorgeschrieben ist. Der Benutzer soll sich leicht seine eigenen Generatoren schreiben können.
Die gespeicherte Triangulierung soll nicht auf simpliziale Elemente beschränkt sein. Die Struktur soll auf nichtlinear berandete Gebiete erweiterbar sein.
Natürlich müssen Erweiterungen, die heute noch nicht absehbar sind, ohne große Probleme in die Struktur eingebunden werden können.
Nicht zu vergessen ist der effiziente Zugriff auf die Objekte der Triangulierung und die Daten der Struktur.

2.2 Der Aufbau einer Triangulierung im $\mathbb{R}^n$

Sei $\mathcal{T}$ eine Triangulierung des polygonal berandeten Gebietes $\Omega \subset \mathbb{R}^n$. Die Elemente $T \in \mathcal{T}$ sind laut Definition (n)-Simplizes. Diese (n)-Simplizes sind aus $(n-1)$-Randsimplizes aufgebaut, die wiederum von $(n-2)$-Randsimplizes aufgespannt werden. Diese Kette läßt sich bis zu den (0)-Simplizes — den Knotenpunkten — fortsetzen. Damit können Triangulierungen unabhängig von ihrer Dimension in einer einzigen Datenstruktur gespeichert werden.
Der Trick besteht darin, jedes (k)-Simplex lediglich mit Informationen über die an

es angrenzenden $(k-1)$- und die aus ihm aufgebauten $(k+1)$-Simplizes zu versehen. Auf diese Weise sind für die Simplizes lediglich drei Klassen nötig:

- Ein n-Simplex (`::net::elementC`) hält Informationen über die $(n-1)$-Randsimplizes.
- Ein 0-Simplex (`::net::nodeC`) hält Informationen über die (1)-Simplizes, die hieraus aufgebaut sind.
- Die (k)-Simplizes $(k = 1 \ldots n-1)$ (`::net::segmentC`) halten Informationen über die $(k-1)$-Randsimplizes und die $(k+1)$-Simplizes, die hieraus aufgebaut sind.

Ein (2)-Simplex kennt seine Knotenpunkte nicht direkt. Um diese zu erfahren, müssen die (1)-Randsimplizes erfragt werden, die von den Punkten wissen. Auf diese Weise kann eine Triangulierung unabhängig von ihrer Dimension gespeichert werden.
Die obigen Klassen besitzen einige gemeinsame Eigenschaften. Um hier den Aufwand gering zu halten, wurde eine Basisklasse `::net::simBaseC` geschrieben. Deren Fähigkeiten gehen durch Vererbung auf die Simplexklassen über.
Die Implementierung ist derzeit noch auf den 2-dimensionalen Fall festgelegt.

2.3 Zusammenfassung von Simplizes zu Triangulierungen

Eine Triangulierung $\mathcal{T}$ von $\Omega \subset I\!R^n$ besteht aus einer Menge von (n)-Simplizes. Diese werden wie oben angegeben gespeichert. Damit werden die Nachbarschaftsinformationen der (n)-Simplizes automatisch mitgespeichert. Da jedes $(n-1)$-Simplex weiß, von welchem (n)-Simplex es ein Randsimplex ist, können so die Nachbarn eines (n)-Simplex in Erfahrung gebracht werden.
Eine Triangulierung besteht aus einer Liste aller $(0 \ldots n)$-Simplizes, die in ihm enthalten sind. Wenn jetzt noch vermieden wird, daß zwei (k)-Simplizes sich gegenseitig überlappen, so ist garantiert, daß die Triangulierung konform ist.
Die (k)-Simplizes $(k = 1 \ldots n)$ werden im folgenden als die Objekte der Triangulierung bezeichnet. Aufgrund der Beschränkung der Implementierung auf zwei Dimensionen werden (0)-Simplizes als Knotenpunkte oder Punkte, (1)-Simplizes als Segmente und (2)-Simplizes als Elemente bezeichnet.

2.3.1 Eine andere Sicht auf eine Triangulierung

In zwei Dimensionen wäre es denkbar, daß die Elemente direkt ihre Knoten, nicht jedoch ihre Segmente kennen. Auf diese Weise ließen sich die Segmente komplett einsparen. Dies würde zu einem nicht unerheblich verringerten Speicherverbrauch

führen. Ziel bei der Realisierung dieser Datenstruktur war jedoch, unabhängig von der verwendeten Raumdimension zu bleiben. Soll eine höherdimensionale Triangulierung gespeichert werden, muß die Hierarchie um weitere Komponenten erweitert werden. Für jede zusätzliche Dimension muß eine neue Komponentenmenge angebaut werden. Diese Komponentenklassen sind durch Vererbung zu realisieren. Die Klasse `::net::segmentC` kann aufgrund ihrer Lage "zwischen den Dimensionen", d.h. sie kennt sowohl niedriger als auch höherdimensionale Objekte, zu einer Basisklasse umgebaut werden. Von dieser Basisklasse muß für jede Raumdimension eine Komponentenklasse abgeleitet werden. Für jede Klasse müssen Verfeinerungsregeln angegeben werden.

2.4 Die Hierarchie

Mit den bisher vorgestellten Mechanismen kann eine Triangulierung unabhängig von ihrer Dimension gespeichert werden. Mit der Hierarchie wird eine Erweiterung für adaptiv verfeinerte Folgen von Triangulierungen eingeführt. Um den Speicherverbrauch zu minimieren, sollen Mehrfachspeicherungen vermieden werden. Mehrfachspeicherungen können auftreten, wenn ein Element beim Übergang zum nächsten Level unverfeinert bleibt.

Diese Mehrfachspeicherung kann dadurch verhindert werden, daß die Triangulierungen selbst lediglich die Indizes ihrer Objekte kennen. Also werden alle (k)-Simplizes der Triangulierung in der Hierarchie gespeichert, die ihnen einen für ihre Dimension eindeutigen Index zuweist. Dieses Konzept paßt sehr gut zu dem Aufbau der (k)-Simplizes aus ihren $(k-1)$-Randsimplizes, da auch diese "Aufbau-Informationen" gerade aus diesen Indizes bestehen.

Da die Simplizes nicht mehr nur in einer Triangulierung existieren, müssen sie wissen, zu welcher Hierarchie sie gehören. Erlaubt ist nur die Zugehörigkeit zu genau einer Hierarchie. Sie müssen auch ihre Vorgänger und Kinder kennen (auch das ist durch Verweise auf die Hierarchie-ID's realisiert). Für die Verfeinerungsregeln muß bekannt sein, welche Simplizes regulär und welche irregulär verfeinert sind.

Die Hierarchie ist die zentrale Stelle, an der alle Komponenten aller Triangulierungen und die Triangulierungen selbst gespeichert werden, wobei Mehrfachspeicherungen identischer Objekte, wie sie etwa bei Verfeinerungen einer Triangulierung auftreten können, vermieden werden.

2.5 Die implementierten Klassen im Detail

Die aktuelle Implementierung von **meshMan** ist auf den 2-dimensionalen Fall beschränkt. Dieser Abschnitt beschreibt die verwendeten Datenfelder der Klassen.

2.5.1 Die Hierarchie

Die Hierarchie enthält alle Objekte und versieht sie mit ihren eindeutigen Indizes. Es wurde darauf verzichtet, mit Zeigern auf die Objekte zu arbeiten, da dadurch eine zukünftige Parallelisierung des Codes unnötig erschwert würde; die vergebenen Indizes sind auf allen verarbeitenden Maschinen eindeutig.

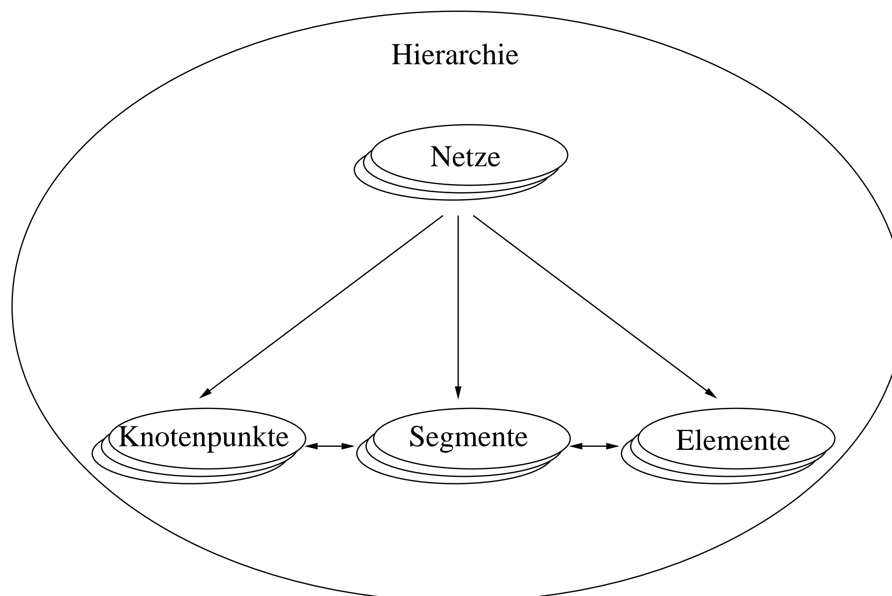

Abbildung 2.1: Aufbau der Hierarchie

Die Hierarchie speichert alle Objekte in Containern. Für die Triangulierungen wurde ein STL-Vektor verwendet, da die Levels geordnet vorliegen. Die Simplizes unterliegen keiner Ordnung. Auf sie wird nur mittels ihrer ID Bezug genommen. Ihr Container ist die Klasse `indexSsetC` (siehe Anhang A.1). Diese Klasse ermöglicht ein schnelles Durchlaufen aller Elemente, sowie Löschen und Einfügen.

2.5.2 Die Triangulierung

Eine Triangulierung ist ein Level einer Familie adaptiv verfeinerter Triangulierungen. Sie enthält Verweise auf alle Objekte dieses Levels und einen Index, der das Level angibt. Level 0 hat insofern eine Sonderbedeutung, als es lediglich die Randbeschreibung des Gebiets enthält.

2.5.3 Die Simplizes

Die Basisklasse

Alle Objekte der Triangulierungen (i.e. Simplizes) haben einige gemeinsame Eigenschaften. Zu deren Realisierung wurde auf den C++-Mechanismus der Vererbung

zurückgegriffen. Die Gemeinsamkeiten wurden in einer abstrakten Basisklasse zusammengefaßt.
Jedes Simplex wird von der Hierarchie mit einer eindeutigen ID versehen. Diese ID kann von dem Simplex jederzeit abgefragt werden. Da mehrere Hierarchien parallel existieren können, muß jedes Simplex zusätzlich noch seine Hierarchie angeben können. Die Angabe seiner Hierarchie ist auch für die Anforderung der auf dieses Objekt bezugnehmenden Objekte von Bedeutung. Ebenso gehört ein Simplex zu einer oder mehreren Triangulierungen. Diese Netze des Simplexes können mit einem in dieser Klasse vordefinierten Iterator durchlaufen und angefragt werden. Um mehreren Netzen anzugehören, muß jedes Simplex verfeinert oder auf ein neues Level kopiert werden. Daher gehört auch eine Verfeinerungs-Markierung und der Aufruf zum Verfeinern zu jedem Simplex.
Für den Benutzer stehen an Verfeinerungen folgende Markierungen zur Verfügung:

- red — Verfeinere das Objekt regulär.
- green — Verfeinere das Objekt irregulär.

Punkte

Die (0)-Simplizes sind die einzigen Objekte der Triangulierung, die Informationen über ihre absolute Position enthalten. Alle anderen Simplizes müssen, um ihre absolute Position zu erfahren, auf ihre Randsimplizes zurückgreifen.
Ein Punkt kennt die Segmente, die ihn als Begrenzung haben. Ihre Anzahl ist beliebig und nur durch die Ressourcen des Rechners begrenzt.

Segmente

Segmente wissen von den zwei Punkten, aus denen sie aufgebaut sind. Ebenso sind ihnen die Indizes der Elemente, deren Randsimplizes sie sind, bekannt. Die Anzahl dieser Elemente ist variabel. Dies kommt daher, daß Elemente irregulär verfeinert werden können und ihre Kinder dieses Segment auch als Begrenzung haben.

Elemente

Elemente sind die Objekte, die am ehesten für den n-dimensionalen Fall unverändert übernommen werden können. Derzeit ist dies die einzige Klasse, die weiß, ob das Objekt regulär oder irregulär verfeinert ist. Ansonsten kennt die Klasse die Indizes der Segmente, die Randsimplizes des Objekts sind.

2.6 Beispielinhalt einer Hierarchie

Nun soll ein Beispiel für eine Hierarchie in zwei Dimensionen angegeben werden; sie ist mit zwei Triangulierungen eines Würfels gefüllt. Die erste besteht aus zwei

Elementen, für die zweite wurde ein Element der ersten regulär verfeinert. Die beiden Levels sind in Abbildung 2.2 dargestellt.

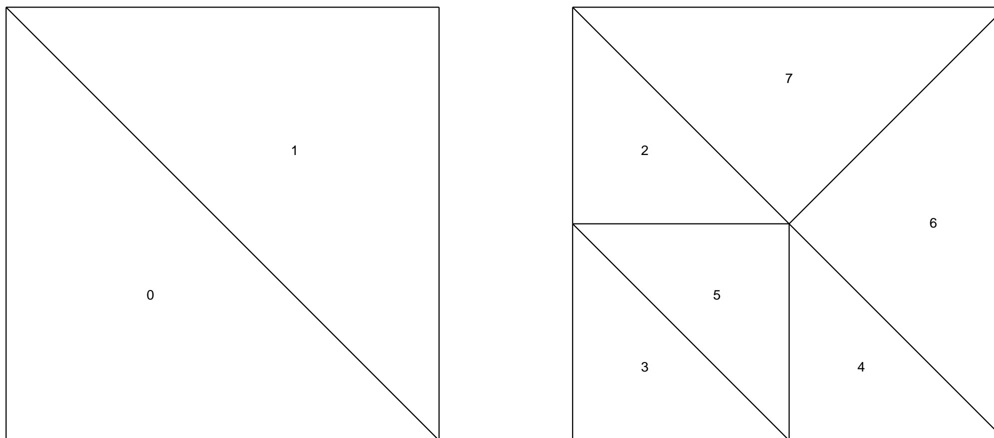

Abbildung 2.2: Zwei Levels einer Hierarchie

Index	Punkte				Segmente						Elemente		
0	0	1	2	3	0	1	2	3					
1	0	1	2	3	4	5	6	7	8		0	1	
2	0	1	2	3	9	10	5	6	11	12	2	3	4
	4	5	6		13	14	15	16	17	18	5	6	7

Tabelle 2.1: Netze

Index	Koordinaten		Segmente						
	x	y							
0	0.0	0.0	0	3	4	7	9	12	
1	1.0	0.0	0	1	4	5	8	10	13
2	1.0	1.0	1	2	5	6	18		
3	0.0	1.0	2	3	6	7	8	11	14
4	0.5	0.0	9	10	16	17			
5	0.0	0.5	11	12	15	16			
6	0.5	0.5	13	14	15	17	18		

Tabelle 2.2: Punkte

Index	Punkte		Elemente		Söhne		Vater
	from	to					
0	0	1			4		
1	1	2			5		
2	2	3			6		
3	3	0			7		
4	0	1	0		9	10	0
5	1	2	1	6			1
6	2	3	1	7			2
7	3	0	0		11	12	3
8	1	3	0	1	13	14	
9	0	4	3				4
10	4	1	4				4
11	3	5	2				7
12	5	0	3				7
13	1	6	4	6			8
14	6	3	2	7			8
15	5	6	2	5			
16	5	4	3	5			
17	6	4	4	5			
18	6	2	6	7			

Tabelle 2.3: Segmente

Index	Segmente			Söhne				Vater	Regulär
0	7	8	4	2	3	4	5		Ja
1	6	8	5	6	7				Ja
2	11	14	15					0	Ja
3	12	9	16					0	Ja
4	13	10	17					0	Ja
5	15	16	17					0	Ja
6	18	13	5					1	Nein
7	18	14	6					1	Nein

Tabelle 2.4: Elemente

2.7 Aufbau der Generatoren

Die verwendete Struktur ermöglicht es, die Triangulierungsgeneratoren nach Dimensionen zu trennen, und bei Bedarf beliebig zu kombinieren. Die Implementierung enthält lediglich einen 1D-Generator, der die vorgegebenen Randsegmente auf

eine vorgegebene Länge bringt. Er kann zu Laufzeit vor einen beliebigen der 2D-Generatoren geschaltet werden.

2.8 Erweiterungsmöglichkeiten

Die derzeitige Implementierung der Datenstruktur kann mit relativ geringem Aufwand erweitert werden.

2.8.1 Elemente mit mehr als drei Eckpunkten

Elemente mit einer beliebigen Anzahl von Eckpunkten sind in der Struktur prinzipiell bereits enthalten. Diese Elemente können jedoch noch nicht verfeinert werden. Es fehlen hierfür brauchbare Verfeinerungsstrategien. Für jede Element-Form muß eine eigene Regel samt Abschluß implementiert werden.
Die Unabhängigkeit von der Eckenzahl wird durch Einsatz der Klasse `indexSetC` (siehe Anhang A.1) zum Speichern der Segment-Indizes erreicht. Durch diese flexible Speichermethode wird jedoch Speicherplatz vergeben. Soll ein Element auf Dreiecke beschränkt bleiben, könnte hier ein Vektor von fester Größe (drei Einträge) verwendet werden.

2.8.2 Gebiete mit krummlinigen Rändern

Aufgrund des konsequenten Einsatzes von Indizes statt Zeigern auf die anderen Objekte der Triangulierung muß einzig die Klasse, die mit einer besonderen Eigenschaft versehen werden soll, verändert werden. Natürlich ist von dieser Änderung die Hierarchie als der zentrale Container betroffen; in ihr muß der Typ des entsprechenden Objekt-Vektors angepaßt werden.
Krummlinige Ränder müssen irgendwie eingelesen und beschrieben werden. Hier bietet sich der Funktionenparser von E. Stafilarakis an. Damit können analytisch gegebene Funktionen eingelesen, bearbeitet — beispielsweise abgeleitet oder addiert — und ausgewertet werden.
Durch krumme Ränder können neue Probleme auftreten. Bei der Generierung mit einer Advancing-Front-Methode (AFM) muß oft geprüft werden, ob sich zwei Segmente schneiden. Eine Lösung für dieses Problem besteht darin, die Differenz der beiden die Segmente beschreibenden Funktionen zu bilden und die Nullstellen der Differenz zu bilden. Zuvor muß jedoch noch der Definitionsbereich der beiden Funktionen angeglichen werden. Ein zweites absehbares Problem entsteht bei der Ausgabe der Segmente. Die einfachste Lösung, die auch am besten in das Gesamt-Konzept paßt, besteht darin, die Segmente als stückweise linear zu betrachten. Soll ein Element verfeinert werden, wird der Mittelpunkt des Segments als der Funktionswert des Mittelpunkts des Definitionsbereichs der Funktion berechnet.

2.8.3 Erweiterung auf n Dimensionen

Die Datenstruktur wurde von Anfang an darauf ausgelegt, Triangulierungen in beliebigen Dimensionen aufnehmen zu können. Im allgemeinen spielen Triangulierungen für $n > 3$ jedoch keine bemerkenswerte Rolle. Die allgemeine Formulierung wurde vor allem aus dem Grund gewählt, 2- und 3-dimensionale Triangulierungen in einer einzigen Struktur speichern zu können.
Indem man die Klasse `::net::elementC` als (n)-Simplex betrachtet und `::net::segmentC` für den k-dimensionalen Fall verwendet ($k = 1 \ldots n - 1$), kann die Struktur für Triangulierungen in n Dimensionen verwendet werden. Die Hierarchie muß hierfür noch mit Containern für die zusätzlichen Dimensionen erweitert werden. Für diese Container müssen Zugriffsfunktionen implementiert werden.
Bei der 2-dimensionalen Implementierung konnten bei der Implementierung der Segmente einige vereinfachende Voraussetzungen gemacht werden. Segmente können in zwei Dimensionen nur regulär verfeinert werden, brauchen also keine Markierung hierfür. Des weiteren sind für eine Strecke genau zwei Endpunkte sinnvoll, im allgemeinen kann solch eine Annahme nicht gemacht werden. Daher braucht man für den allgemeinen Fall einige zusätzliche Attribute für die Segment-Klasse.

Kapitel 3

Die Advancing-Front-Methode

Die Advancing-Front-Methode (AFM) ist ein einfacher aber effizienter Ansatz, ein polyedrisches Gebiet $\Omega \subset I\!R^n$, das durch seine $(n-1)$-Randsimplizes gegeben ist, mit einer konformen Triangulierung zu füllen.
Diesem Kapitel liegt größtenteils [12] zugrunde. Weitere Informationen zu dem Verfahren können [17] entnommen werden.

Definition 3.1 (teilweise Triangulierung)
Sei $\Omega \subset I\!R^n$ ein polyedrisches Gebiet und $\mathcal{T}$ eine Triangulierung im $I\!R^n$. Dann heißt $\mathcal{T}$ teilweise Triangulierung von Ω, falls

$$\bigcup_{T \in \mathcal{T}} T \subset \overline{\Omega}$$

gilt.

Definition 3.2 (Front)
Sei $\Omega \subset I\!R^n$ ein polyedrisches Gebiet und $\mathcal{T}$ eine teilweise Triangulierung von Ω. Eine Menge F von $(n-1)$-Simplizes heißt Front von $\mathcal{T}$ in Ω, falls folgende Bedingungen erfüllt sind:

- *Falls $\mathcal{T} = \emptyset$ gilt, ist F mit dem Rand von Ω identisch.*
- *Sei $T \in F$ beliebig. Dann ist jedes $(n-2)$-Randsimplex von T Randsimplex genau eines weiteren Elements aus F, das von T verschieden ist.*
- *Mit jedem Element aus F ist eine Information verbunden, auf welcher Seite der Front $\Omega \setminus \mathcal{T}$ liegt.*
- *Sei $T \in \mathcal{T}$ beliebig. Dann gilt für jedes $(n-1)$-Randsimplex von T genau eine der Bedingungen: T ist*
  - *zu dem Rand von Ω gehörig,*
  - *ein Randsimplex eines benachbarten Simplexes von T aus $\mathcal{T}$,*
  - *ein Element der Front.*

3.1 Das Verfahren

Im folgenden werden (n)-Simplizes als Elemente, $(n-1)$-Simplizes als Segmente bezeichnet. Folglich besteht die Front aus Segmenten.
Ziel ist es, im Inneren eines polyedrischen Gebietes $\Omega \subset I\!R^n$ eine Triangulierung zu erzeugen. Dafür wird die Front mit den $(n-1)$-Randsimplizes von Ω initialisiert. Danach wird iterativ ein Element — das sogenannte *Basiselement* — aus F ausgewählt, auf dem versucht wird, ein neues Simplex im Inneren der Front (d.h. in $\Omega \setminus \mathcal{T}$, wobei $\mathcal{T}$ die bisherige, teilweise Triangulierung von Ω ist) aufzusetzen. Schlägt dieser Versuch fehl (z.B. aus Platzgründen), wird ein neues Basissegment ausgewählt. Das Aufsetzen erfolgt nach bestimmten *Konstruktionsregeln.* Nach jedem Iterationsschritt wird die Front aktualisiert, d.h. die Front wird über die neu erzeugten Simplizes geschoben, so daß diese Simplizes außerhalb der Front liegen und die Front wieder gültig ist. Das Verfahren ist beendet, wenn die Front leer ist. Dabei werden doppelt vorhandene Elemente, d.h. $T, T' \in F,\ T = T'$, aus F sofort bei ihrem Auftreten entfernt. Abbildung 3.1 zeigt solch einen Vorgang in zwei Dimensionen.
Ablauf der AFM:

1. Initialisierung der Front mit dem Rand des gewünschten Gebietes
2. Analyse der Front
    - Auswahl eines geeigneten Front-Segments
    - Analyse der Umgebung um das Front-Segment
        - Erzeugung von inneren Punkten, Segmenten und Simplizes
        - Aktualisierung der Front
3. Solange die Front nicht leer ist, gehe zu Punkt 2

Diese Methode ist stark an eine anschauliche Vorgehensweise angelehnt. Vom Rand ausgehend werden in Richtung des Inneren nach bestimmten Regeln möglichst regelmäßige Dreiecke erzeugt.

3.2 Die Konstruktion im $I\!R^2$

In diesem Abschnitt werden konkrete Konstruktionsregeln für eine AFM in zwei Dimensionen vorgestellt.
Laut Definition muß jedes Front-Element wissen, in welcher Richtung von ihm aus gesehen sich das Innere der Front befindet. In mehr als zwei Dimensionen kann dies durch Speicherung des Normalenvektors auf jedes Element realisiert werden. In zwei Dimensionen ist der Normalenvektor jedoch durch die Ordnung der Eckpunkte der Front-Elemente gegeben. In diesem Fall kann also auf explizite Speicherung verzichtet werden.

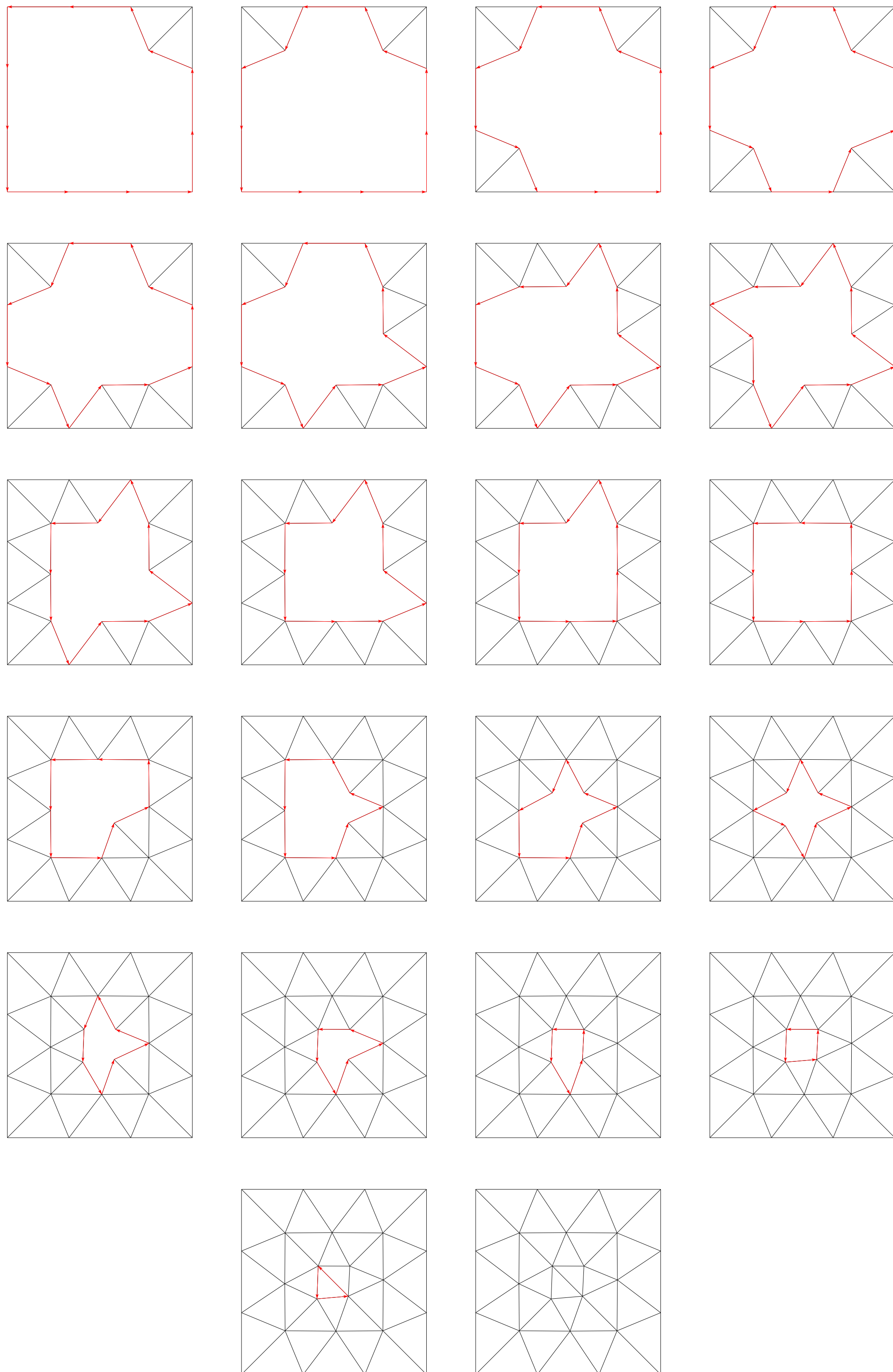

Abbildung 3.1: Beispiel für die AFM

3.2.1 Auswahl eines Basis-Segments aus der Front

Die Auswahl eines geeigneten Front-Segments als Basis für die Erzeugung neuer Elemente ist mit ausschlaggebend für die Qualität der resultierenden Triangulierung. Hierfür gibt es unterschiedliche Strategien, wobei sich eine Kombination mit dem "minimalen Abstand zum Rand" bewährt hat.

Definition 3.3 (minimaler Abstand zum Rand)
Mit dem minimalen Abstand eines Segments zum Rand bezeichnet man die minimale Anzahl an inneren Segmenten, die benötigt wird, um einen geschlossenen Weg von diesem Segment zum Rand des Gebietes zu zeichnen.

Zu jedem Front-Segment wird als Zusatzinformation der Abstand zum Rand gespeichert. Hierüber kann eine gleichmäßige Wanderung der Front nach innen erreicht werden (siehe Abbildung 3.2/3.3).

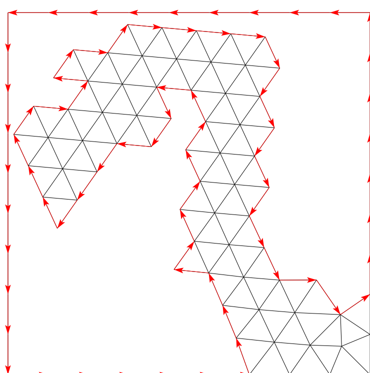

Abbildung 3.2: 65 Schritte, Kriterium: Länge

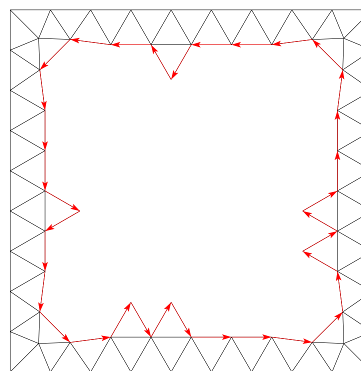

Abbildung 3.3: 65 Schritte, Kriterium: Abstand, dann Länge

In der Implementierung dieses Algorithmus haben sich Auswahlverfahren bewährt, die die Segmente mit dem geringsten Abstand zum Rand bevorzugen und aus diesen das endgültige Segment mittels eines zusätzlichen Kriteriums selektieren.
Solche Eigenschaften können sein:

- minimale Länge des Segments
- minimaler Winkel zu einem der Nachbar-Segmente
- minimale Länge des Segments multipliziert mit dem minimalen Winkel zu einem der Nachbar-Segmente.

Der Abstand zum Rand bindet stärker als eine der drei obigen Eigenschaften. Daher kann es durchaus vorkommen, daß andere Segmente die gewünschte Eigenschaft noch stärker minimieren würden, durch das Abstandskriterium aber ausscheiden.

3.2.2 Die Konstruktionsregeln

Kernstück der AFM sind die Konstruktionsregeln für Elemente. Diese geben vor, wie ein neues Element erzeugt wird. Bei der vorliegenden Implementierung wurde eine AFM mit drei Regeln realisiert. α bezeichnet den kleineren der beiden Winkel zu den Nachbarsegmenten aus der Front.

- $\alpha < \frac{\pi}{2}$: Regel 1 (Abbildung 3.4)

  Aus den existierenden Segmenten s_2 und s_3, sowie aus dem neu zu erzeugenden Segment s_5 wird ein neues Element geformt. Danach werden s_2 und s_3 aus der Front entfernt und s_5 hinzugefügt.

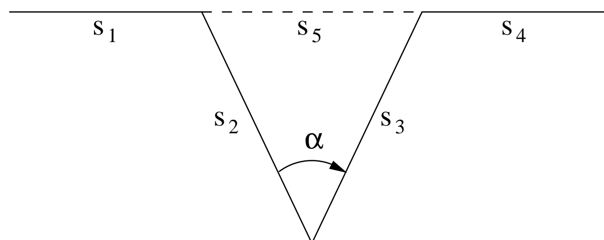

Abbildung 3.4: Regel 1

- $\frac{\pi}{2} \leq \alpha \leq \frac{2\pi}{3}$: Regel 2 (Abbildung 3.5)

  Segment s_6 halbiert α, hat die Länge

$$l(s_6) = \frac{2l(s_2) + 2l(s_3) + l(s_1) + l(s_4)}{6}$$

  und wird zur Erzeugung zweier neuer Dreiecke s_2, s_6, s_5 und s_6, s_3, s_7 benutzt.

  Falls β oder $\gamma < \frac{\pi}{5}$, wird Regel 1 angewandt. Bei $\frac{\pi}{5}$ handelt es sich um einen empirisch gewählten Wert.

  Die Segmente s_2 und s_3 werden in der Front durch s_5 und s_7 ersetzt.

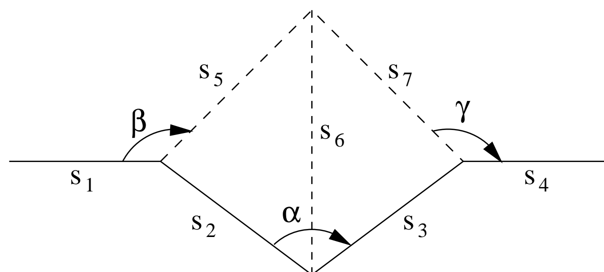

Abbildung 3.5: Regel 2

- $\frac{2\pi}{3} < \alpha$: Regel 3 (Abbildung 3.6)

  Auf Segment s_2 wird mit den neuen Segmenten s_4 und s_5 ein gleichseitiges Dreieck aufgesetzt. s_2 wird aus der Front entfernt, s_4 und s_5 hinzugefügt.

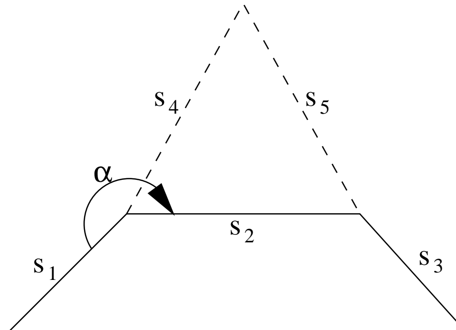

Abbildung 3.6: Regel 3

Bei allen Regeln muß geprüft werden, ob ein neues Segment ein bereits existierendes schneidet bzw. ob ein neu erzeugter Punkt noch innerhalb der aktuellen Front liegt. Des weiteren ist auch hier auf die korrekte Orientierung der Front zu achten.

3.3 Anmerkungen

Ein großer Vorteil der AFM ist die gute Anpaßbarkeit an besondere Anforderungen. Beispielsweise können leicht nichtsimpliziale Triangulierungen mit Viereckselementen oder gemischten Elementen erzeugt werden. In [17] wird ein komplexeres Regelwerk beschrieben, das auf einer abstrakten Speicherung der Regeln basiert.
Die AFM ist ein "naives" Verfahren: Von der Front ausgehend werden lokal Elemente eingefügt, ohne Rücksicht auf globale Gegebenheiten des Gebiets zu legen. So kann die Anzahl der Simplizes der resultierenden Triangulierung im voraus nicht angegeben werden.
Die oben vorgestellten Regeln bereiten Probleme, wenn die Front zusammenwächst, da bei der Erzeugung neuer Punkte vorhandene nicht berücksichtigt werden. Durch dieses Vorgehen ist die Generierung oft auch erfolglos. Indem man in der Umgebung eines zu erzeugenden Punktes nach existierenden sucht und diese nach Möglichkeit nutzt, wird sowohl die Qualität der resultierenden Triangulierung als auch die Erfolgsquote verbessert. Ein weiterer positiver Nebeneffekt dieses "Schnappens" ist, daß erst mit dieser Erweiterung Gebiete mit Löchern behandelt werden können. Jedoch wird die Implementierung durch das mit dem Schnappen verbundene Splitten der Front in Teilfronten erschwert.
Durch gleichmäßig auf dem Rand des Gebiets verteilte Punkte werden im allgemeinen die besten, gleichmäßigsten Triangulierungen erreicht.

Das größte Problem bei dieser Methode ist die fehlende theoretische Absicherung; die Konstruktionsregeln und die Auswahlverfahren für das Basissegment sind rein heuristisch gewählt.
Trotz allem können mit dieser Methode — insbesondere in Verbindung mit Glättern — sehr gute Resultate erzielt werden. Siehe dazu Kapitel 10.

Kapitel 4

Die Delaunay-Methode

Bei der Delaunay-Methode gibt es keine vorgeschriebenen Wege, wie eine Triangulierung konstruiert werden muß. Es wird lediglich ein Kriterium für die Lage der Simplizes der Triangulierung bezüglich ihrer Nachbarn vorgegeben. Eine ausführlichere Behandlung der Delaunay-Methode kann [18] entnommen werden.

4.1 Definitionen

In diesem Abschnitt sollen vorab Element-Mengen vorgestellt werden, die zur Konstruktion einer Delaunay-Triangulierung nötig und hilfreich sind.
Im folgenden bezeichnet $\mathcal{T}_r$ eine Triangulierung, P einen Punkt, der, wenn nicht anders angegeben, kein Eckpunkt eines Elements in $\mathcal{T}_r$ ist. T ist ein Element aus $\mathcal{T}_r$ und κ_T dessen Umkreis.

Definition 4.1 (Delaunay-Kriterium)
Sei $\mathcal{T}_r$ eine Triangulierung von $\Omega \subset I\!R^n$. Dann ist das Delaunay-Kriterium erfüllt, falls im Inneren des Umkreises eines jeden Simplexes von $\mathcal{T}_r$ kein Punkt der Triangulierung enthalten ist.

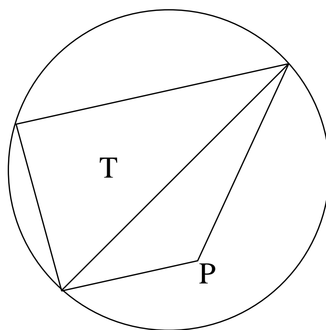

Abbildung 4.1: Verletzung des Delaunay-Kriteriums

Das Delaunay-Kriterium ist bei der Erzeugung von Triangulierungen äußerst nützlich. Es wird von der resultierenden Triangulierung im allgemeinen jedoch nicht erfüllt werden Dieser Effekt tritt durch Rundungsfehler und unpassende Rand-Segmente auf (siehe Abschnitt 4.6).

Definition 4.2
Sei P ein Knotenpunkt aus $\mathcal{T}_r$. Dann ist der Patch $\mathcal{P}_P$ von P die Menge aller Elemente aus $\mathcal{T}_r$, die P als Knoten haben.

Definition 4.3
Unter der Cavity eines Punktes P versteht man die Menge aller Elemente in $\mathcal{T}_r$, deren Umkreis P enthält.

Definition 4.4
Die Pipe eines Segments S mit Endpunkten aus $\mathcal{T}_r$ (d. h. die Endpunkte des Segments sind in $\mathcal{T}_r$ enthalten) ist definiert als die Menge aller Simplizes aus $\mathcal{T}_r$, die mindestens ein Randsimplex haben, das von S geschnitten wird.

Definition 4.5
Sei $circumcenter_T$ der Umkreismittelpunkt eines Simplexes T und ρ_T dessen Umkreisradius.
Das Verhältnis $\alpha(P,T) := \frac{dist(P, circumcenter_T)}{\rho_T}$ wird als Delaunay-Maß des Punktes P bezüglich des Elements T bezeichnet.

Definition 4.6
Eine Menge von Punkten in zwei Dimensionen befinden sich in einer allgemeinen Lage, falls es keine drei kolineare oder vier kozyklische Punkte gibt.

4.2 Voronoi-Diagramme

Definition 4.7 (Voronoi-Diagramm)
Sei $\mathcal{S}$ eine Menge von Punkten P_i, $i = 1, \ldots, k$ in n Dimensionen.
Das Voronoi-Diagramm von $\mathcal{S}$ ist die Menge der Zellen V_i, für die gilt

$$V_i = \{P \mid dist(P, P_i) \leq dist(P, P_j),\ \forall j \neq i\}$$

wobei $dist(.,.)$ den euklidschen Abstand zwischen zwei Punkten bezeichnet.

Eine Voronoi-Zelle ist also die Menge aller Punkte der konvexen Hülle von $\mathcal{S}$, die näher an P_i ist als an jedem anderen Punkt aus $\mathcal{S}$. Die V_i's sind offene, nichtüberlappende, konvexe Polygone (in zwei Dimensionen oder n-Polytope in n Dimensionen). Die Konstruktion einer Delaunay-Triangulierung hängt eng mit dem Voronoi-Diagramm zusammen. Die Konstruktion einer Delaunay-Triangulierung der konvexen Hülle dieser Punkte kann als Gegenstück zu dem Voronoi-Diagramm von $\mathcal{S}$ vervollständigt werden.

4.2.1 Von Voronoi zu Delaunay

Laut Definition ist jede Zelle V_i eine nicht-leere Menge und genau einem Punkt aus $\mathcal{S}$ zugeordnet. Zu diesen Zellen kann das Gegenstück gebildet werden, das dann genau die Delaunay-konforme Triangulierung ergibt. In zwei Dimensionen halbieren die "Zellwände" die Strecke zwischen den Knotenpunkten zweier benachbarter Zellen und damit die Segmente der Triangulierung, denn verbindet man die Knotenpunkte der benachbarten Zellen, so ergibt sich die Delaunay-Triangulierung.

Die Zellwände stehen in jeder Dimension n genau orthogonal auf den $(n-1)$-Randsimplizes der Delaunay-Elemente.

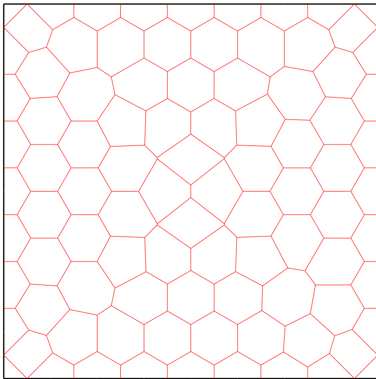

Abbildung 4.2:
Voronoi-Diagramm für ein Quadrat

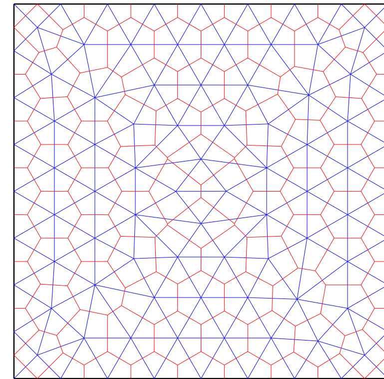

Abbildung 4.3:
Voronoi-Diagramm für ein Quadrat mit zugehöriger Delaunay-Triangulierung

4.2.2 Die Eindeutigkeit der Delaunay-Triangulierung

In den für uns interessanten Fällen von zwei oder drei Dimensionen ist — für eine beliebige Punktmenge, deren Elemente sich in allgemeiner Lage befinden — das zum Voronoi-Diagramm duale Gegenstück die Delaunay-Triangulierung der zugehörigen konvexen Hülle. Darüber hinaus ist diese Triangulierung eindeutig und besteht nur aus Dreiecks- oder Tetraeder-Elementen ($n = 2, 3$). Ist die Voraussetzung der allgemeinen Lage verletzt, besteht die Triangulierung nicht nur aus Simplizes. Im 2-dimensionalen Fall wären dann noch Viereckselemente enthalten. Solche Elemente können aber problemlos mit Dreiecken ausgefüllt werden. Die so resultierende Triangulierung wird auch als Delaunay-Triangulierung $\mathcal{T}_{Del}$ bezeichnet.

4.3 Das Delaunay-Lemma

Lemma 4.1
Sei $\mathcal{T}$ eine gegebene, beliebige Triangulierung der konvexen Hülle einer Menge $\mathcal{S}$ von Punkten. Falls für alle Paare benachbarter Simplizes aus $\mathcal{T}$ das Delaunay-Kriterium gilt, dann gilt das Kriterium global und $\mathcal{T}$ ist eine Delaunay-Triangulierung.

Beweis: Betrachte ein Paar von Simplizes T_P und T_Q aus $\mathcal{T}$, die eine $(n-1)$-dimensionale Seite S gemeinsam haben. P bzw. Q sei der Punkt von T_P bzw. T_Q der nicht auf S liegt. Dann gilt

$$Q \notin \kappa_{T_P} \Leftrightarrow P \notin \kappa_{T_Q}$$

wobei κ_{T_P} den Umkreis des Simplexes T_P bezeichnet. Diese Eigenschaft wird als Symmetrie des Delaunay-Kriteriums bezeichnet.
H_{T_P} (bzw. H_{T_Q}) bezeichne die Halb-Ebene, die P (bzw. Q) beinhaltet, beschränkt durch die Hyperebene, die durch das gemeinsame Segment S gegeben wird.
Sei nun $Q \notin \kappa_{T_P}$. Dann gilt bei Betrachtung des Simplex-Paares T_P und T_Q

$$\kappa_{T_P} \cap H_{T_Q} \subset \kappa_{T_Q} \cap H_{T_Q} \tag{4.1}$$

und entsprechend

$$\kappa_{T_Q} \cap H_{T_P} \subset \kappa_{T_P} \cap H_{T_P}.$$

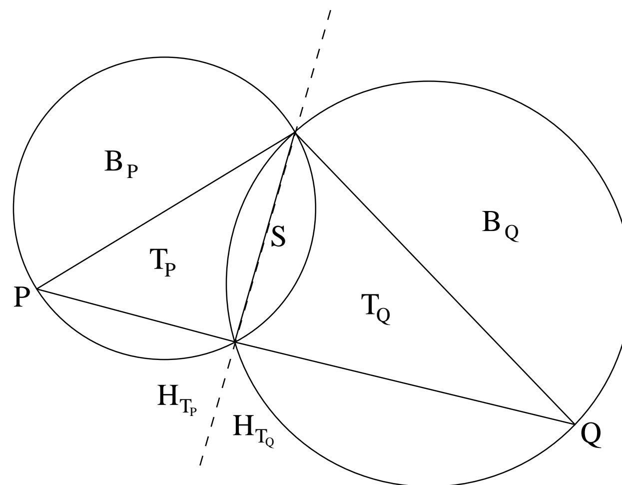

Abbildung 4.4: Umkreise und Hyperebene

Nehmen wir nun an, das Delaunay-Kriterium sei für alle Paare benachbarter Simplizes erfüllt. Weiter existiere ein Punkt P_k in der Triangulierung, der im Umkreis

eines Simplexes liegt, dessen Nachbar-Simplizes P_k nicht als Knoten enthalten. Dieses Simplex sei T_0. G bezeichne den Schwerpunkt von T_0, S_{GP_k} die Strecke GP_k. Diese Strecke verbindet die Simplizes T_0 und T_k (eines der Simplizes des Patches von P_k) und schneidet die $(n-1)$-dimensionalen Seiten mehrerer Simplizes $T_0, T_1, \ldots, T_k$. κ_i bezeichne nun vereinfachend den Umkreis eines Simplexes T_i.

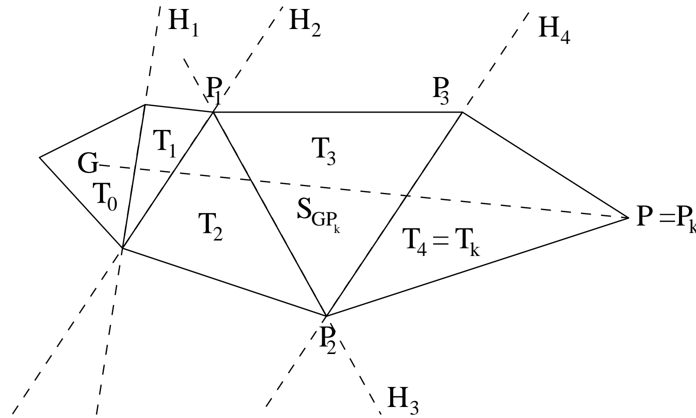

Abbildung 4.5: Konstruktion der Simplizes

Nach Voraussetzung gilt

$$P_k \in \kappa_0.$$

Da T_{k-1} und T_k benachbart sind, gilt für sie nach Voraussetzung das Delaunay-Kriterium und somit

$$P_k \notin \kappa_{k-1}.$$

Die Simplizes T_i seien sortiert von $T_0, \ldots, T_k$. Sei P_i der Knoten in T_i, der nicht zu T_{i-1} gehört und sei H_i die Halbebene, die T_i enthält, nicht aber T_{i-1}. Nach Konstruktion gilt $P_k \in H_i$ für $i \in 1, \ldots, k$.
Sei $i \in \{1, \ldots, k-1\}$ so gewählt, daß $P_k \in \kappa_i$ gilt. Da $P_k \in H_{i+1}$ und mit (4.1)

$$\kappa_i \cap H_{i+1} \subset \kappa_{i+1} \cap H_{i+1} \tag{4.2}$$

erfüllt sind, folgt $P_k \in \kappa_{i+1}$. Da jedoch $P_k \in \kappa_0$ vorausgesetzt wurde, gilt $P_k \in \kappa_{k-1}$, wodurch das Delaunay-Kriterium für die benachbarten Simplizes T_k und T_{k-1} verletzt wird, da P_k Knotenpunkt von T_k ist. □

4.4 Inkrementelle Methode und Delaunay-Kern

Eine Delaunay-Triangulierung kann auf unterschiedliche Arten konstruiert werden. Ein Weg ist die Nutzung der Dualität zwischen dem Voronoi-Diagramm und der

Delaunay-Triangulierung. Ein weiterer ist die inkrementelle Methode, die im folgenden benutzt wird.
Seien $\mathcal{T}_i$ die Delaunay-Triangulierungen der konvexen Hülle der ersten i-Punkte aus $\mathcal{S}$. Wir betrachten den $i+1$-ten Punkt aus dieser Menge, der mit P bezeichnet wird. Das Ziel der inkrementellen Methode ist es $\mathcal{T}_{i+1}$ — die Triangulierung, die P als Knoten enthält — aus $\mathcal{T}_i$ zu erhalten. Zu diesem Zweck wird der Delaunay-Kern eingeführt.

Definition 4.8
Unter dem Delaunay-Kern versteht man die Funktion

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{P}_P \tag{4.3}$$

wobei $\mathcal{C}_P$ die Cavity des einzufügenden Punktes P und $\mathcal{P}_P$ die Menge aller Elemente, die durch Verbinden von P mit allen Randpunkten von $\mathcal{C}_P$ entstehen, bezeichnet.

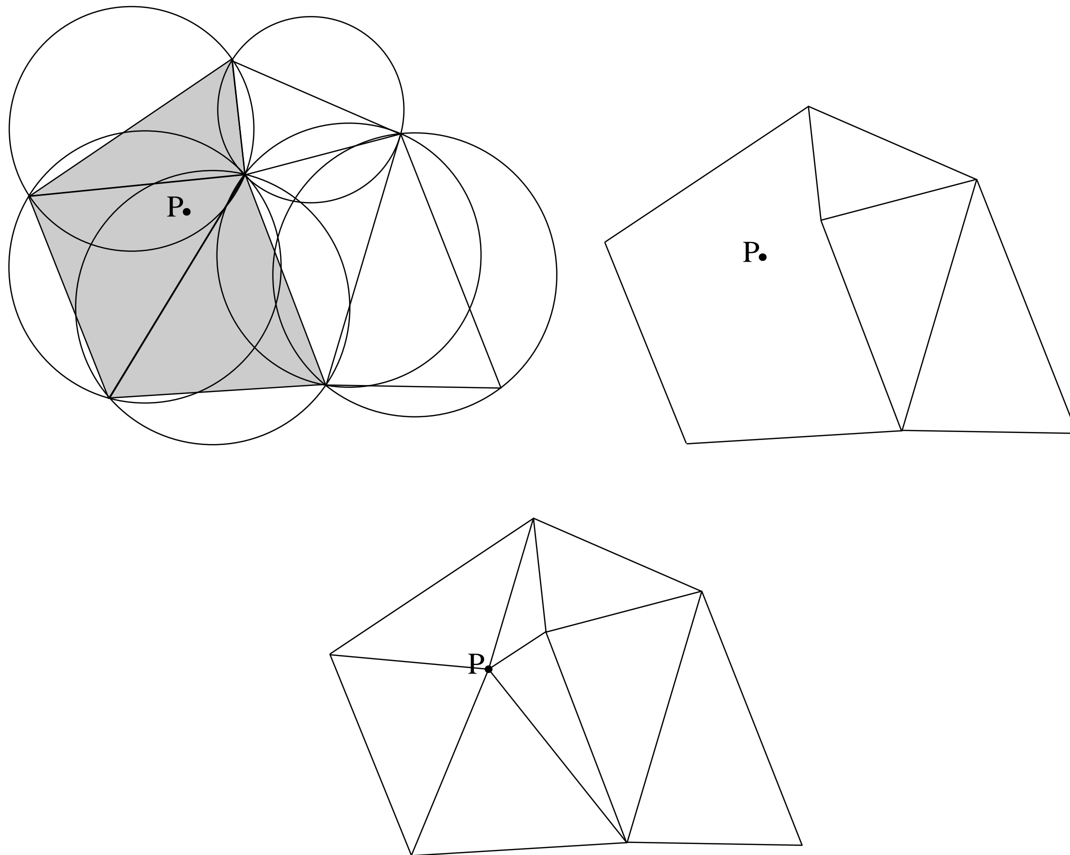

Abbildung 4.6: Die Schritte des Delaunay-Kerns

Bemerkung 4.1
Der Delaunay-Kern ist in Abbildung 4.6 dargestellt. $\mathcal{P}_P$ ist der Patch des Punktes P, nachdem er in die Triangulierung eingefügt wurde. □

Bemerkung 4.2

Es gibt zwei Möglichkeiten für die Lage von P bezüglich der Triangulierung $\mathcal{T}_i$, entweder P liegt inner- oder außerhalb von $\mathcal{T}_i$. Im folgenden soll nur der erste Fall betrachtet werden. Dies bedeutet keine Einschränkung, da die erste Triangulierung $\mathcal{T}_0$ aus den Randpunkten leicht von Hand erzeugt werden kann. Im 2-dimensionalen Fall besteht $\mathcal{T}_0$ aus vier Knoten und zwei Dreiecken. Diese vier Punkte werden so gewählt, daß $\mathcal{S}$ komplett innerhalb von $\mathcal{T}_0$ liegt.

Satz 4.1

Sei $\mathcal{T}_i$ eine Delaunay-Triangulierung der konvexen Hülle der ersten i Punkte einer Menge $\mathcal{S}$. Dann ist $\mathcal{T}_{i+1}$ eine Delaunay-Triangulierung dieser Hülle, die P — den $i+1$-ten Punkt aus $\mathcal{S}$ — als Knotenpunkt enthält.

Beweis: Die Triangulierung des Komplements der Cavity $\mathcal{C}_P$ bleibt unverändert, und da P außerhalb der Umkreise aller Elemente aus $\mathcal{T}_i$ liegt, bleibt das Delaunay-Kriterium für diese Menge von Elementen erhalten.

Nun soll gezeigt werden, daß die einzige Möglichkeit einer Delaunay-Triangulierung von $\mathcal{C}_P$, die P enthält, der Patch von P ist.

Zunächst wird vorausgesetzt, daß sich die Punkte aus $\mathcal{S}$ in allgemeiner Lage befinden. Mit neuer Triangulierung wird die Triangulierung der Cavity $\mathcal{C}_P$ bezeichnet, nachdem P eingefügt wurde.

Sei T ein Simplex aus der neuen Triangulierung, das P nicht als Knoten hat. Laut Konstruktion gilt $T \in \mathcal{C}_P$, da die Punkte aus $\mathcal{T}_{i+1}$ in allgemeiner Lage sind. Dadurch ist die Triangulierung der Cavity eindeutig. Dies ist eine Folge der Dualität zwischen Voronoi-Diagramm und Delaunay-Triangulierung. Daher verletzt T das Delaunay-Kriterium. Da $\mathcal{C}_{i+1}$ eindeutig ist, muß also jedes Element der neuen Triangulierung P als Knoten enthalten. Folglich ist die neue Triangulierung von $\mathcal{C}_P$ der Patch $\mathcal{P}_P$ von P.

Liegen die Punkte aus $\mathcal{S}$ beliebig, wird wiederum angenommen, es gebe ein Simplex T aus der neuen Triangulierung, das P nicht als Knoten enthält. Dann gibt es in $\mathcal{C}_P$ ein oder mehrere Simplizes, deren Umkreis identisch zu dem von T ist. Folglich verletzt T das Delaunay-Kriterium, was wie oben zum gewünschten Ergebnis führt. □

Die Cavity ist bezüglich P sternförmig. Durch diese Eigenschaft ist sichergestellt, daß die resultierende Triangulierung konform ist.

Die Delaunay-Triangulierung der konvexen Hülle von $\mathcal{S}$ wird also dadurch erreicht, daß jeder Punkt aus $\mathcal{S}$ mittels des Delaunay-Kerns in die Triangulierung eingefügt wird. Dieser Prozeß wird — wie in Bemerkung 4.2 beschrieben — mit einer Anfangstriangulierung initiiert, die $\mathcal{S}$ komplett überdeckt.

4.5 Überführung in eine Delaunay-Triangulierung

Das Ergebnis dieses Abschnitts gilt nur in zwei Dimensionen. Edge-Swapping ist ein topologischer Vorgang, der zwei Simplizes mit einem gemeinsamen Segment, deren Vereinigung konvex ist, dahingehend manipuliert, daß das gemeinsame Segment "herumgeklappt" wird (Abbildung 4.7). Durch nochmalige Anwendung auf die erzeugten Dreiecke werden die ursprünglichen Simplizes wiederhergestellt.

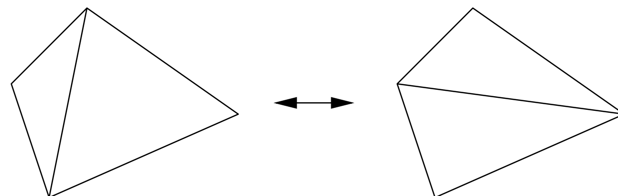

Abbildung 4.7: Edge-Swapping

Satz 4.2
Sei $\mathcal{T}_1$ eine beliebige Triangulierung der konvexen Hülle einer Punktmenge $\mathcal{S}$. Die Delaunay-Triangulierung $\mathcal{T}_{Del}$ von $\mathcal{S}$ kann aus $\mathcal{T}_1$ durch Edge-Swapping erhalten werden.

Beweis: Es ist grundsätzlich möglich, bei zwei benachbarten Dreiecken, die das Delaunay-Kriterium verletzen, das gemeinsame Segment zu swappen, da diese notwendigerweise ein konvexes Polygon formen. Nach dem Swappen ist das Delaunay-Kriterium jedoch erfüllt.
Daher besteht der gesamte Vorgang darin, den obigen Prozeß überall dort anzuwenden, wo das Delaunay-Kriterium verletzt ist. Aus Lemma 4.1 ergibt sich das gewünschte Ergebnis. □

Satz 4.3
Sei $\mathcal{S}$ eine Menge von Punkten in allgemeiner Lage. Sei $\mathcal{T}_1$ eine beliebige Triangulierung der konvexen Hülle von $\mathcal{S}$. Jede beliebige andere Triangulierung $\mathcal{T}_2$ von $\mathcal{S}$ kann aus $\mathcal{T}_1$ durch Edge-Swapping erhalten werden.

Beweis: Wir benutzen Satz 4.2, um aus der gegebenen Triangulierung $\mathcal{T}_1$ eine Delaunay-Triangulierung $\mathcal{T}_{Del}$ zu erhalten. Danach benutzen wir denselben Satz in der anderen Richtung, um $\mathcal{T}_{Del}$ in $\mathcal{T}_2$ zu wandeln. Das ist möglich, da Edge-Swapping ein reversibler Vorgang ist. □

Dieses Ergebnis wird später benutzt werden, um Nebenbedingungen für eine Delaunay Triangulierung in zwei Dimensionen zu konstruieren. Die obigen Sätze wurden bislang noch nicht auf drei Dimensionen ausgeweitet.

4.6 Nebenbedingungen

In den vorigen Abschnitten war immer die Rede von einer Triangulierung der konvexen Hülle einer Menge von Punkten. Ziel ist aber eine Triangulierung eines gegebenen Gebietes $\Omega \subset I\!R^n$. Es kann dabei vorkommen, daß der Rand von Ω nicht als $(n-1)$-Simplex eines Elements $T \in \mathcal{T}$ in $\mathcal{T}$ enthalten ist. Die Einhaltung dieses Randes wird als Erfüllung von Nebenbedingungen formuliert.

4.6.1 Definitionen zu den Nebenbedingungen

Definition 4.9 (Delaunay-Triangulierung mit Nebenbedingung)
Sei $\mathcal{C}onst$ eine Menge von $(n-1)$-Simplizes im $I\!R^n$. Dann erfüllt eine Delaunay-Triangulierung $\mathcal{T}_r$ die Nebenbedingungen $\mathcal{C}onst$, falls sämtliche Segmente aus $\mathcal{C}onst$ in $\mathcal{T}_r$ enthalten sind.

Abbildung 4.8 illustriert das Problem von nichterfüllten Nebenbedingungen für ein konkaves Gebiet. Die nach innen gerichteten Randsegmente werden von der zugehörigen Triangulierung nicht eingehalten.

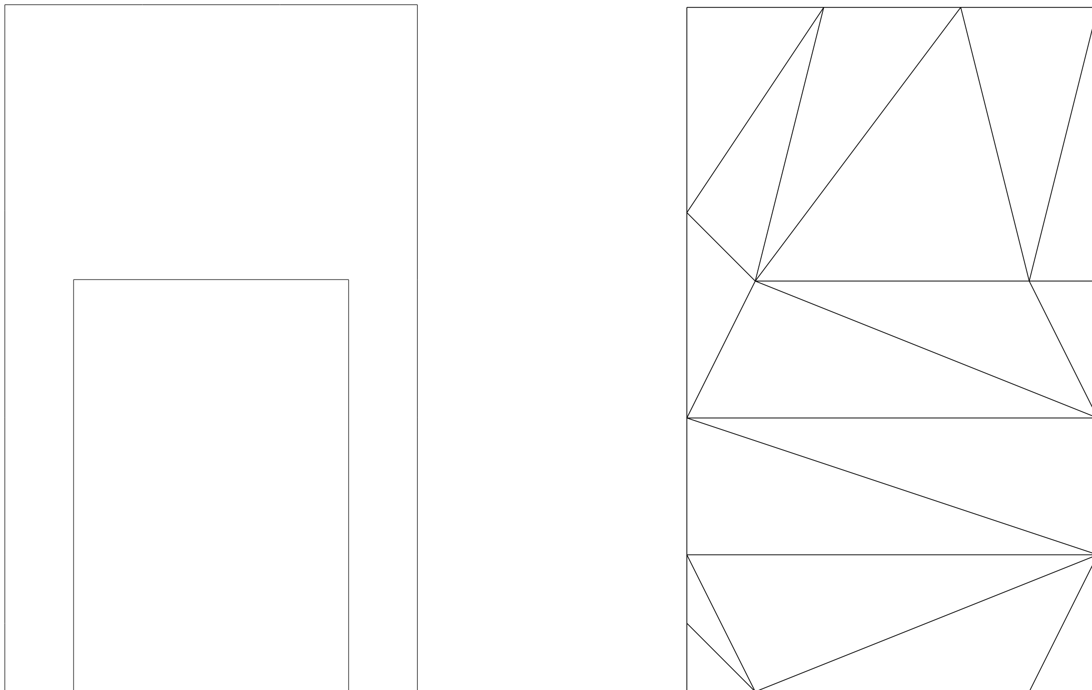

Abbildung 4.8: Nichteingehaltene Nebenbedingungen

Es kann auch vorkommen, daß eine vorgegebene Menge $\mathcal{C}onst$ keine Triangulierung zuläßt, die überall das Delaunay-Kriterium einhält. Für solch einen Fall gibt es mehrere Möglichkeiten, die eine existierende Triangulierung manipulieren, um die Nebenbedingungen zu erfüllen. Die resultierende Triangulierung erfüllt das Delaunay-Kriterium im allgemeinen jedoch nicht mehr.

Definition 4.10 (exaktes Einhalten der Nebenbedingungen)
Eine gegebene Triangulierung $\mathcal{T}_r$ erfüllt die Nebenbedingungen $\mathcal{C}onst$ exakt, falls jedes Element $S \in \mathcal{C}onst$ Randsimplex eines Elements $T \in \mathcal{T}_r$ ist.

Definition 4.11 (schwaches Einhalten der Nebenbedingungen)
Eine Triangulierung $\mathcal{T}_r$ erfüllt die Nebenbedingungen $\mathcal{C}onst$ schwach, falls jedes Element $S \in \mathcal{C}onst$ in $\mathcal{T}_r$ exakt oder als Vereinigung mehrerer Randsimplizes enthalten ist.

Die Abbildungen 4.9–4.11 veranschaulichen diese Definitionen. Abbildung 4.10 erfüllt die Nebenbedingungen aus Abbildung 4.9 exakt, während in Abbildung 4.11 ein Segment durch zwei kürzere ersetzt wurde. Sie zeigen auch, daß die vorgegebenen Nebenbedingungen nicht auf den Rand des Gebietes beschränkt sind, sondern auch $(n-1)$-Simplizes im Inneren vorschreiben können.

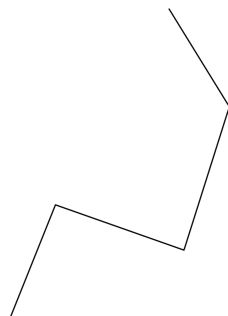

Abbildung 4.9: vorgegebene Nebenbedingungen

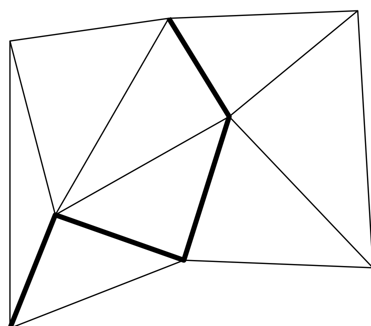

Abbildung 4.10: exaktes Einhalten der Nebenbedingungen

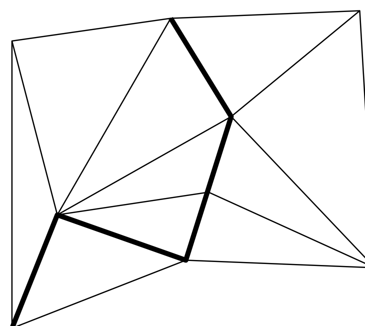

Abbildung 4.11: schwaches Einhalten der Nebenbedingungen

In den folgenden Abschnitten wird ausschließlich der 2-dimensionale Fall untersucht. Dabei wird vorausgesetzt, daß kein Punkt der Triangulierung auf einem offenen Segment aus $\mathcal{C}onst$ liegt.

4.6.2 Manipulation der Nebenbedingungen

Sei $\mathcal{T}$ eine Triangulierung, die die gegebenen Nebenbedingungen $\mathcal{C}onst$ nicht komplett erfüllt. Ziel ist eine Triangulierung, die $\mathcal{C}onst$ schwach — d.h. nach Definition 4.11 — erfüllt.
Die Grundidee besteht darin, jedes Dreieck der Pipe (Definition 4.4) eines nicht eingehaltenen Segments aus $\mathcal{C}onst$ S wie in Abbildung 4.12 derart umzuändern, daß das gewünschte Segment in der Triangulierung enthalten ist.

Abbildung 4.12: Manipulation der Nebenbedingungen

Der folgende Algorithmus erzwingt eine fehlende Nebenbedingung $S \in \mathcal{C}onst$, das zwei Punkte A und B miteinander verbindet:

- Bilde die Pipe B_S des Segments S. Die Punkte A und B sind in $\mathcal{T}$ enthalten.
- Finde die k Schnittpunkte $P_1, \ldots, P_k$ von S und alle inneren Segmente der Pipe B_S.
- Füge die Segmente $AP_1, P_1P_2, \ldots, P_kB$ in die Triangulierung ein.
- Bilde Dreiecke so, daß die obige Liste von Segmenten in der Triangulierung erhalten bleibt.

4.6.3 Erzwingen der Nebenbedingungen durch Edge-Swapping

Es wurde bereits in Satz 4.3 gezeigt, daß eine beliebige Triangulierung aus einer weiteren, aus den selben Punkten bestehenden, erhalten werden kann. Dieses Resultat soll nun dazu verwendet werden, die fehlenden Segmente der Triangulierung herzustellen. Die resultierende Triangulierung wird Definition 4.10 erfüllen.
Sei $S \in \mathcal{C}onst$ eine fehlende Nebenbedingung einer Triangulierung $\mathcal{T}$; seien A und B die Endpunkte von S.

- Bilde die Pipe B_S des Segments S.
- Wende wiederholt Edge-Swapping bei jedem inneren Segment aus B_S an, bis das gewünschte Resultat erreicht ist. Hierbei ist darauf zu achten, daß eine Endlosschleife vermieden wird. Das kann durch zufälliges Auswählen eines Segments geschehen. Die Pipe muß nach jedem Swap-Vorgang erneuert werden, oder zumindest das neue Segment eingetragen und das alte gelöscht werden.

Dieser Algorithmus läßt sich in endlich vielen Schritten ausführen, da eine Lösung nach Satz 4.2 existiert. Abbildung 4.13 stellt solch einen Vorgang dar.

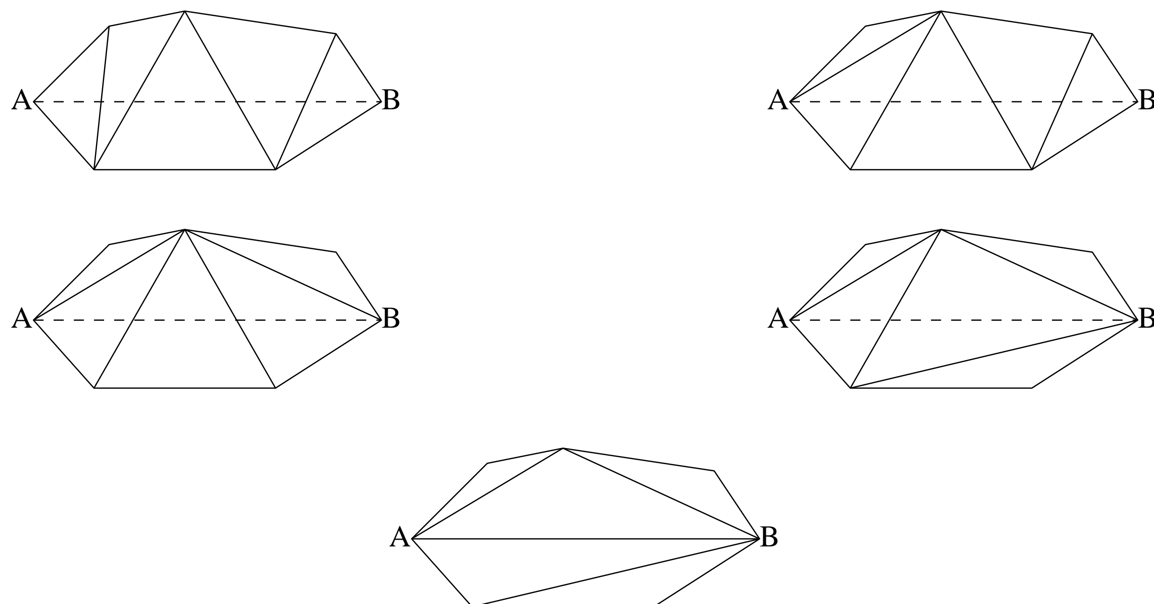

Abbildung 4.13: Erzwingen einer Nebenbedingung durch Edge-Swapping.

4.7 Die Erzeugung der leeren Triangulierung

Wie in Bemerkung 4.2 beschrieben wird das gegebene Gebiet in ein Rechteck eingebettet und eine Triangulierung aus zwei Elementen erzeugt. In diese Triangulierung werden dann sukzessive alle gegebenen Punkte mittels des Delaunay-Kerns eingefügt. Nachdem alle Punkte in der Triangulierung enthalten sind, wird durch die in Abschnitt 4.6 vorgestellten Techniken der Rand des gewünschten Gebietes hergestellt. Die Dreiecke außerhalb des Gebietes können dann erkannt und entfernt werden. Die nach diesem Schritt erhaltene Triangulierung wird die *leere Triangulierung* genannt. Dieser Begriff wurde so gewählt, da im inneren des Gebietes noch keine Punkte eingefügt wurden, das Gebiet also noch "leer" ist.
Die äußeren Elemente können mit dem folgenden Algorithmus durch Einfärben erkannt werden. Bezeichne dabei $\mathcal{T}_{\text{Box}}$ die Triangulierung des Gebietes und des umgebenden Rechtecks.

1. Weise allen Elementen von $\mathcal{T}_{\text{Box}}$ den Wert -1 zu und initialisiere $c = 0$.

2. Suche ein Element aus $\mathcal{T}_{\text{Box}}$, das einen Randpunkt von $\mathcal{T}_{\text{Box}}$ als Knoten hat. Dieses Element bekommt den Wert c zugewiesen.

3. Besuche die Nachbarn des aktuellen Elements.

- Falls das erreichte Element einen Wert ungleich -1 hat, gehe zu (3)
- Falls das erreichte Element ohne ein Rand-Segment zu überqueren erreicht wurde, weise diesem den Wert c zu und gehe zu (3)
- Falls das überquerte Segment zum Rand gehört, gehe zu (3)

4. Setze $c := c + 1$ und gehe zu (3), solange, bis kein Element mit dem Wert -1 existiert.

Der Algorithmus sucht zusammenhängende Komponenten des Gebietes, und weist allen Komponenten Werte zu, gerade falls außerhalb, ungerade falls innerhalb. Falls der Rand nicht geschlossen ist, haben am Ende alle Elemente den Wert 0.

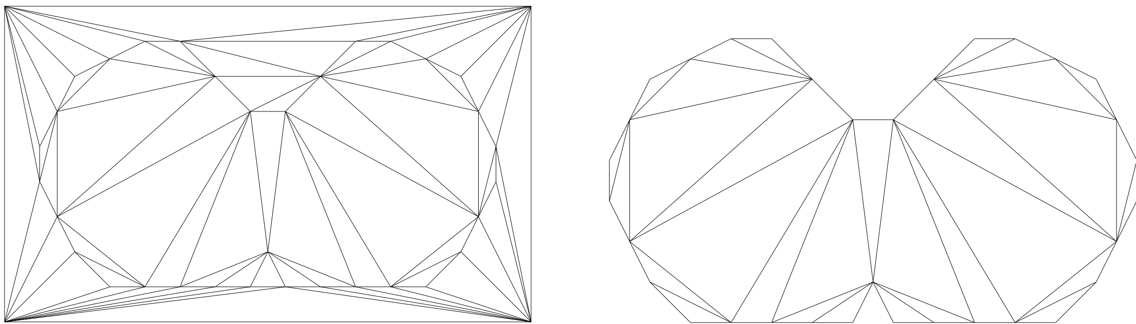

Abbildung 4.14: Leere Triangulierung

Das Entfernen der unnötigen Dreiecke findet jedoch noch nicht zu diesem Zeitpunkt statt, da dadurch die Konvexität der Triangulierung verloren ginge. Statt dessen werden die äußeren Dreiecke erst dann entfernt, nachdem alle Punkte innerhalb des Gebietes generiert wurden.

4.8 Generierung der inneren Punkte

Eine der nützlichsten Eigenschaften dieser Triangulierungs-Generierungsmethode ist, daß zuerst die leere Triangulierung, und erst dann die inneren Punkte erzeugt werden. Das hat den Vorteil, daß eine Triangulierung bekannt ist, die durch Einfügen von Punkten verbessert werden soll. So kann man leicht feststellen, ob ein neu einzufügender Punkt zu nahe an einem bereits existierenden liegt.

4.8.1 Erzeugung entlang der Kanten

Die inneren Punkte werden in diesem Fall mit dem Ziel möglichst gleichlanger Kanten auf die bereits existierenden Kanten gelegt. Dabei werden zuerst alle Punkte berechnet, und dann einer nach dem andern mit dem Delaunay-Kern in die Triangulierung eingefügt. Dieser Vorgang wird so lange wiederholt, bis keine Kante in der Triangulierung mehr unterteilt werden kann.

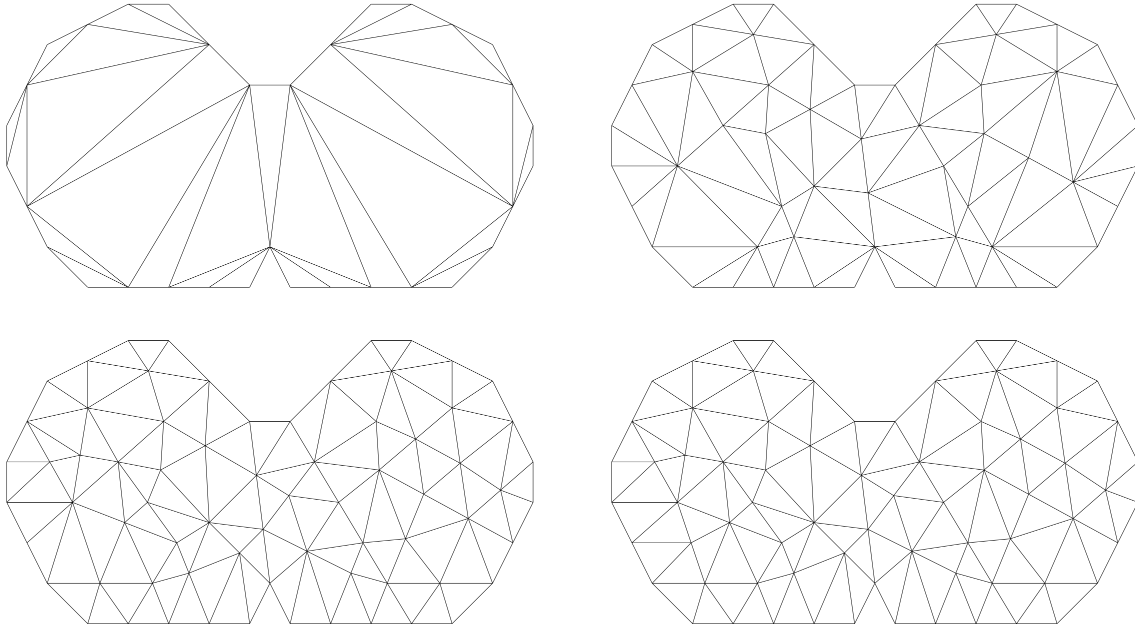

Abbildung 4.15: Einfügen von Punkten entlang der Kanten

Der hier vorliegende klassische Fall mit möglichst gleichlangen Kanten erlaubt es, eine arithmetische Punktverteilung auf den Kanten vorzunehmen. Jedem Punkt in der Triangulierung $\mathcal{T}_l$ (nach dem iterativen Schritt l, $\mathcal{T}_0$ ist die leere Triangulierung) wird ein Wert h_{loc} — die Schrittweite — zugewiesen. Dieser Wert gibt die gewünschte Länge der von diesem Punkt ausgehenden Kanten an. Er kann durch die mittlere Länge seiner bereits existierenden Kanten initialisiert werden.
Sei S mit den Endpunkten A und B ein Segment aus $\mathcal{T}_l$, des weiteren

- $h(0) = h_{loc}(A)$ die mit A verbundene Schrittweite,
- $h(n+1) = h_{loc}(B)$ die mit B verbundene Schrittweite.

$n+1$ ist die Anzahl der zu erzeugenden Punkte. Man kann also eine Folge α_i

$$\begin{cases} \alpha_0 &= \quad h(0) + r \\ \alpha_n &= h(n+1) - r \\ \alpha_i &= \operatorname{dist}(P_i, P_{i+1}) \end{cases} \tag{4.4}$$

definieren, wobei die α_i die Positionen der neuen Punkte auf der Kante über ihre Abstände zueinander angeben. $\operatorname{dist}(P_1, P_2)$ ist der euklidische Abstand zweier Punkte P_1 und P_2, r gibt die Dichte der Verteilung an.
Zur Bestimmung von (4.4) muß das System

$$\begin{cases} \sum_{i=0}^{n} \alpha_i &= \operatorname{dist}(A, B) \\ \alpha_{i+1} = \alpha_i + r & \end{cases}$$

gelöst werden; hier gibt d die Länge des Segments S an. Daraus ergibt sich

$$n = \frac{2\text{dist}(A, B)}{h(0) + h(n+1)} - 1,$$

und

$$r = \frac{h(n+1) - h(0)}{n+2}.$$

Da n ganzzahlig sein muß, wird n zuerst gerundet und dann daraus r berechnet. Die Schrittweite der neuen Punkte ist r. Die so erhaltenen Punkte werden nicht sofort in die Triangulierung eingefügt, sondern erst nachdem alle berechnet wurden, gefiltert und dann mittels des Delaunay-Kerns eingebunden. Filtern heißt in diesem Zusammenhang, daß kein Punkt zu nah an einem bereits in der Triangulierung befindlichen oder einem anderen neuen liegen darf. Solche Punkte werden einfach nicht eingefügt.

4.9 Der Algorithmus im Überblick

1. Vorbereitung:
   - Einschluß des Gebietes in ein Rechteck.
   - Generierung einer Triangulierung aus zwei Elementen des Rechtecks.
2. Konstruktion der Triangulierung innerhalb des Rechtecks:
   - Einfügen der vorgegebenen Punkte.
3. Generierung der inneren Punkte:
   - Suche nach nicht vorhandenen, aber gewünschten Segmenten.
   - Erzwingen dieser Segmente.
   - Suche nach den Zusammenhangskomponenten des Gebietes.
   - Untersuchung der inneren Kanten.
   - Einfügen von Knoten entlang dieser Kanten mittels des Delaunay-Kerns.
   - Solange Punkte eingefügt wurden, gehe zu (3)
4. Definition des Gebietes:
   - Entfernen aller Elemente außerhalb des Gebietes.

4.10 Alternative Methoden

Der Delaunay-Kern ist nicht der einzige Weg, eine Delaunay-Triangulierung zu erhalten. Der Sweeping-Algorithmus und eine Divide- and Conquer-Technik werden in [18] vorgestellt. Hier soll auf diese Verfahren nicht weiter eingegangen werden.

Kapitel 5

Verbesserung der Qualität einer Triangulierung im $\mathbb{R}^2$

Im allgemeinen ist die Qualität einer Triangulierung nach der Generierung nicht optimal. Es existieren jedoch verschiedene Strategien, Triangulierungen bezüglich eines vorgegebenen Qualitätskriteriums zu verbessern. Eine Auswahl dieser Strategien, die auch unter dem Namen Glättungsverfahren bekannt sind, sollen nun vorgestellt werden.
Im Rahmen der dieser Arbeit zugrundeliegenden Implementierung wurden der klassische baryzentrische Glätter und der Edge-Swap-Operator realisiert.

5.1 Anforderungen an einen Glättungs-Operator

Definition 5.1 (zulässiger Glättungsoperator)
Ein Glättungsoperator heißt zulässig, falls eine konforme Triangulierung nach Anwendung dieses Operators wieder konform ist und die resultierende Triangulierung in seiner Qualität verbessert wurde (d.h. das schlechteste Element derjenigen, auf die sich der Operator auswirkt, ist nach der Glättung besser als das schlechteste Element vor der Anwendung).

Ein unzulässiger Operator könnte beispielsweise sich überschneidende Segmente erzeugen.

5.2 Topologische Operatoren

Topologische Operatoren verändern die Zusammenhänge einer Triangulierung, beispielsweise Nachbarschaftsinformationen, indem Objekte der Triangulierung entfernt oder hinzugefügt werden.

5.2.1 Edge-Swapping

Edge-Swapping wurde in Kapitel 4.5 bereits zur Einhaltung der Nebenbedingungen einer Delaunay-Triangulierung benutzt. Der Operator kann auch zur Verbesserung der Qualität einer Triangulierung verwendet werden.
Der Glättungsprozeß durchläuft sämtliche inneren Segmente der Triangulierung und prüft, ob das aktuelle Segment herumgeklappt werden kann. Ist dies der Fall, muß getestet werden, ob die beiden resultierenden Dreiecke bezüglich eines gegebenen Qualitätskriteriums eine Verbesserung zum alten Zustand darstellen. Dann kann das Segment geswapt werden, und das nächste untersucht werden.

5.2.2 Splitten von Segmenten

Auf einem Segment S, das P_1 und P_2 verbindet, wird ein neuer Punkt P eingefügt und S durch Segmente P_1P und PP_2 ersetzt. Die beiden Dreiecke, die S als Segment haben, werden durch vier neue ersetzt.

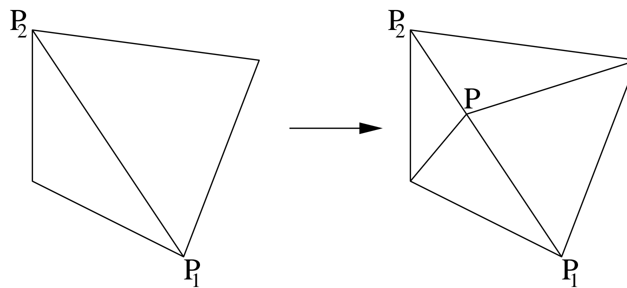

Abbildung 5.1: Splitten des Segments P_1P_2

Die Position von P kann bezüglich eines beliebigen Qualitätsmaßes gewählt werden oder so, daß S halbiert wird.

5.2.3 Entfernen von Segmenten

Sei S ein Segment, das zwei Punkte P_1 und P_2 verbindet. Dieses Segment soll durch nur einen Punkt P ersetzt werden. Einfache Möglichkeiten für die Lage von P stellen P_1, P_2 oder die Mitte von S dar. Abbildung 5.2 zeigt diese drei Möglichkeiten.

5.3 Geometrische Operatoren

Geometrische Operatoren zeichnen sich dadurch aus, daß Nachbarschaften in der Triangulierung erhalten bleiben. Sie arbeiten im wesentlichen mit Verschiebung von Knotenpunkten.

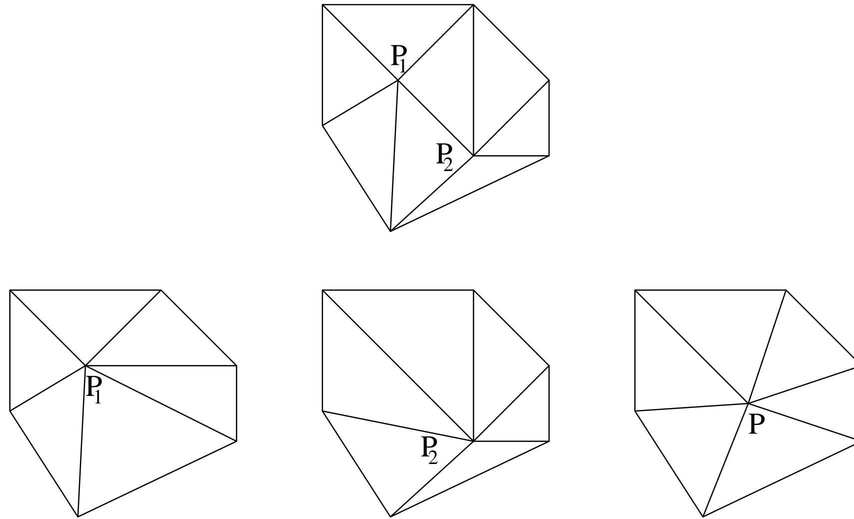

Abbildung 5.2: Entfernen des Segments P_1P_2

Das Verschieben eines Knotens beeinflußt nur den Knoten selbst und die Größen der Simplizes höherer Dimensionen, die aus diesem Knoten aufgebaut sind. So wird die Länge einer Strecke verändert, wenn einer ihrer Endpunkte verschoben wird.

Der Operator versucht, einen Knoten P' optimal innerhalb des Patchs B_P des Ausgangspunktes P zu positionieren. Unabhängig von der verwendeten Methode kann P' über einen Parameter $\omega \in [0,1]$ einer Relaxation unterzogen werden. Das bedeutet, daß ein zusätzlicher Punkt $\overline{P'}$ eingeführt wird, der als $\overline{P'} = (1-\omega)P + \omega P'$ definiert ist. Statt P' wird dann $\overline{P'}$ als Zielposition für P verwendet. Dieses Vorgehen kann dabei helfen, den optimalen Punkt $\overline{P'}$ im Inneren von B_P zu halten.

5.3.1 Klassisches baryzentrisches Glätten

Seien B_P der Patch von P und P_j die b von P verschiedenen Knoten aus B_P. Dann soll P im Schwerpunkt von B_P positioniert werden:

$$P' = \frac{1}{b}\sum_{j=1}^{b} P_j.$$

Da B_P i.a. nicht konvex ist, muß sichergestellt werden, daß P' im Inneren von B_P liegt.

Bei diesem Vorgang — der auch als Laplace-Glätter bekannt ist — wird der Unterschied der Längen der Strecken, die von P ausgehen, minimiert. Dieses Verfahren ist dann geeignet, wenn keine Größenvorgaben an die einzelnen Elemente der Triangulierung gestellt werden. Andernfalls müssen Gewichte eingeführt werden.

5.3.2 Gewichtetes baryzentrisches Glätten

Seien P, B_P, P_j und b definiert wie in Abschnitt 5.3.1. Jedem Punkt P_j sei ein Gewicht α_j zugeordnet. Dann ist

$$P' = \frac{\sum_{j=1}^{b} \alpha_j P_j}{\sum_{j=1}^{b} \alpha_j}$$

eine weitere Möglichkeit für baryzentrisches Glätten. Für die Wahl der α_j stehen viele Varianten zur Verfügung. Falls für jeden Punkt eine Wunschgröße vorgegeben ist, wäre $\alpha = \frac{1}{\overline{h_j}^2}$ mit $\overline{h_j}$ als dem Mittel der Größenvorgaben von P und P_j denkbar.

5.4 Globale Anwendung eines lokalen Glätters

Die bisher vorgestellten Verfahren sind in ihrer Auswirkung auf einen lokalen Bereich beschränkt. Um die gesamte Triangulierung zu verbessern ist es nötig, die Operatoren wiederholt auf alle in Frage kommenden Grundobjekte anzuwenden. Die Wiederholungen werden durch Auswirkungen der Glättung von Nachbarobjekten notwendig. Beispielsweise wird der Patch eines Punktes P durch Edge-Swapping eines seiner Segmente verändert, was eine neue Situation beim Verschieben von Knoten bedeutet. Damit die Wiederholungen nicht ausarten, kann die Anzahl der Durchläufe beschränkt werden.
Ein lokaler Operator wird zuerst simuliert, und ausgehend von den Ergebnissen der Simulation wird entschieden, ob er auch wirklich ausgeführt werden soll.

5.5 Verbesserung im $\mathbb{R}^3$

Von den vorgestellten Verfahren kommen lediglich die geometrischen für eine Verbesserung von Triangulierungen im $\mathbb{R}^3$ in Frage. Eine ausführliche Darstellung von Verfahren kann [18] entnommen werden.

Teil II

Verfeinerung einer Triangulierung

Kapitel 6

Grundlagen der Verfeinerung

Ziel der numerischen Lösung von partiellen Differentialgleichungen ist es, Lösungen innerhalb vorgegebener Fehlertoleranzen zu finden. Triangulierungen, die durch die in den bisherigen Kapiteln vorgestellten Verfahren generiert wurden, sind lediglich als Ausgangstriangulierungen zu verstehen und führen im allgemeinen zu Lösungen, die diese Bedingung nicht erfüllen. Um innerhalb dieser Toleranzen zu bleiben, kann entweder eine global feinere Triangulierung erzeugt oder die Ausgangstriangulierung adaptiv verfeinert werden. Die erste dieser beiden Methoden ist bezüglich Rechenzeit und Speicherbedarf sehr aufwendig.
Der zweite Teil der Arbeit behandelt die adaptive Verfeinerung einzelner Simplizes einer Triangulierung. Zur Auswahl dieser Simplizes existieren unterschiedliche Fehlerschätzer [20].
Dieses einführende Kapitel stellt einige grundlegende Begriffe vor, die im folgenden zur Verfeinerung einer Triangulierung benötigt werden. Kapitel 7 stellt eine reguläre Verfeinerungsstrategie im $\mathbb{R}^2$ vor, in Kapitel 8 wird ein Teil davon für den $\mathbb{R}^n$ verallgemeinert. Kapitel 9 geht auf die Implementierung solch eines Verfahrens ein. Die Ausführungen von Teil 2 beruhen im wesentlichen auf [9].

6.1 Grundlagen

Definition 6.1 (dyadische Verfeinerung eines Simplexes)
Sei T ein nichtentartetes Simplex im $\mathbb{R}^n$. Eine Triangulierung $\mathcal{S}(T)$ heißt dyadische Verfeinerung von T, wenn $\mathcal{S}(T)$ aus mindestens zwei Elementen besteht und alle Eckpunkte der Simplizes $T' \in \mathcal{S}(T)$ entweder mit Eckpunkten oder Kantenmittelpunkten von T übereinstimmen. Außerdem seien die Eckpunkte eines jeden $T' \in \mathcal{S}(T)$ paarweise verschieden. Dann heißt T Vater der Simplizes $T' \in \mathcal{S}(T)$. Die T' werden als Söhne von T, bzw. Brüder der anderen Simplizes aus $\mathcal{S}(T)$ bezeichnet.

Der Fall der trivialen Triangulierung $\mathcal{S}(T) = \{T\}$ wurde in Definition 6.1 ausgeschlossen, um die Vater-Sohn-Beziehung sinnvoll definieren zu können. Ansonsten

könnte der Vater mit seinem Sohn übereinstimmen.

Bemerkung 6.1
Im folgenden wird jede Verfeinerung als dyadisch vorausgesetzt. □

Durch die Bedingung an die Eckpunkte der Söhne läßt sich deren Volumen abschätzen:

Lemma 6.1
Sei T ein nichtentartetes Simplex im $I\!R^n$ und $\mathcal{S}(T)$ eine Verfeinerung davon. Dann gilt für jeden Sohn $T' \subset \mathcal{S}(T)$ die Abschätzung

$$vol_{T'} \geq \frac{vol_T}{2^n}.$$

Beweis: Der Beweis arbeitet auf dem Referenzelement $T = \left[0, e^{(1)}, \ldots, e^{(n)}\right]$ mit den Standardeinheitsvektoren e^i, $i = 1, \ldots, n$ des $I\!R^n$. Aus Lemma 1.1 erhält man $\mathrm{vol}_T = \frac{1}{n!}$.
Sei $T' = \left[x^{(0)}, \ldots, x^{(n)}\right]$ ein beliebiger Sohn von T, für den nach Definition $\mathrm{vol}_{T'} > 0$ gilt. Laut Konstruktion liegen die Koordinaten der Eck- und Kantenmittelpunkte von T in der Menge $\left\{0, \frac{1}{2}, 1\right\}$, folglich gilt dasselbe für die Eckpunkte von T'.
Sei nun $\tilde{T}' := \left[2x^{(0)}, \ldots, 2x^{(n)}\right]$. Da $\tilde{T}'$ aus T' durch Skalierung mit dem Faktor 2 hervorgeht, gilt $\mathrm{vol}_{\tilde{T}'} = 2^n \mathrm{vol}_{T'} > 0$. Außerdem besitzen die Eckpunkte von $\tilde{T}'$ nach Konstruktion ganzzahlige Koordinaten. Hieraus folgt mit Lemma 1.1 $\mathrm{vol}_{\tilde{T}'} \geq \frac{1}{n!}$, da die Determinante der ganzzahligen Matrix $B_{\tilde{T}'}$ ebenfalls ganzzahlig ist. Somit gilt die Abschätzung $\mathrm{vol}_{T'} \geq \frac{2^{-n}}{n!} = \frac{\mathrm{vol}_T}{2^n}$, und die Behauptung ist für das Referenzelement T bewiesen. Für ein beliebiges, nichtentartetes Simplex im $I\!R^n$ folgt die Behauptung durch affine Transformation (Lemma 1.6). □

Aus Lemma 6.1 folgt, daß jedes verfeinerte Simplex $T \subset I\!R^n$ höchstens 2^n Söhne besitzt, die in diesem Fall alle das Volumen $\frac{\mathrm{vol}_T}{2^n}$ haben. Solche maximalen Verfeinerungen spielen aufgrund des identischen Volumens aller Söhne eine wichtige Rolle bei der Formulierung eines Verfeinerungsalgorithmus.

Definition 6.2 (reguläre Verfeinerung)
Eine Verfeinerung $\mathcal{S}(T)$ eines Simplexes $T \subset I\!R^n$ heißt regulär, wenn $\mathcal{S}(T)$ aus genau 2^n Söhnen von T besteht. Andernfalls heißt $\mathcal{S}(T)$ irregulär.

Bisher wurden lediglich Verfeinerungen einzelner Simplizes betrachtet. Diese werden nun zu Verfeinerungen von Triangulierungen — also auf polyedrischen Gebieten — zusammengefaßt. Dabei gelten bisher unverfeinerte Simplizes als regulär.

Definition 6.3 (Verfeinerung einer Triangulierung)
Seien $\mathcal{T}, \mathcal{T}'$, $\mathcal{T} \neq \mathcal{T}'$ Triangulierungen eines polyedrischen Gebietes $\Omega \subset I\!R^n$. Dann heißt $\mathcal{T}'$ Verfeinerung von $\mathcal{T}$, wenn für jedes Element $T \in \mathcal{T}$ entweder $T \in \mathcal{T}'$ gilt oder eine Teilmenge $\mathcal{S}(T) \subset \mathcal{T}'$ existiert, die Verfeinerung von T ist. Im Fall $T \in \mathcal{T} \cap \mathcal{T}'$ heißt T beim Übergang von $\mathcal{T}$ zu $\mathcal{T}'$ unverfeinert.

Definition 6.4 (reguläre Verfeinerung von Familien von Triangulierungen)
Sei $\Omega \subset I\!R^n$ ein polyedrisches Gebiet und $\mathcal{F} = (\mathcal{T}_0, \ldots, \mathcal{T}_J)$ eine Familie von Triangulierungen von Ω.
Eine Triangulierung $\mathcal{T}_i \in \mathcal{F}$ wird als Level i von $\mathcal{F}$ bezeichnet.
$\mathcal{F}$ heißt regulär, falls Level $i+1$ $(i = 0, \ldots, J-1)$ eine Verfeinerung von Level i ist.
Als Verfeinerung eines irregulären Simplexes $T \in \mathcal{T}_i$ ist nur eine Verfeinerung des letzten regulären Vorfahren $T' \in \mathcal{T}_j$ $(j < i)$ von T erlaubt.

Bemerkung 6.2
Im folgenden wird für eine Familie von Triangulierungen vorausgesetzt, daß sie regulär verfeinert ist. □

Um irreguläre Elemente weiter verfeinern zu können, müssen sie durch reguläre ersetzt werden. Das Entartungsmaß für eine Triangulierung aus Definition 1.14 soll nun auf Familien von Triangulierungen ausgedehnt werden.

Definition 6.5
Sei $\mathcal{F} = (\mathcal{T}_0, \ldots, \mathcal{T}_J)$ eine Familie von Triangulierungen. Die Größe

$$\mathcal{Q}_{\mathcal{F}} := \max_{\mathcal{T} \in \mathcal{F}} \mathcal{Q}_{\mathcal{T}}$$

heißt Qualität oder Entartungsmaß von $\mathcal{F}$. Gilt

$$\mathcal{Q}_{\mathcal{F}} < \infty,$$

so heißt $\mathcal{F}$ stabil.

Definition 6.6
Sei $\Omega \subset I\!R^n$ ein polyedrisches Gebiet und $\mathcal{T}$ eine Triangulierung von Ω. Dann heißt eine Verfeinerungsstrategie regulär, wenn dadurch eine regulär verfeinerte Familie von Triangulierungen erzeugt wird.

Kapitel 7

Verfeinerung bestehender Triangulierungen im $\mathbb{R}^2$

Ziel dieses Kapitels ist es, eine Verfeinerungsstrategie für Familien von Triangulierungen im $\mathbb{R}^2$ zu präsentieren.

7.1 Rot-Grün–Verfeinerung in zwei Dimensionen

Ein *Rot-Grün–Verfeinerungsalgorithmus* ist aus drei Bestandteilen zusammengesetzt:

1. einer stabilen, *regulären* Verfeinerungsstrategie für die vom Fehlerschätzer vorgesehenen Verfeinerungen,
2. zusätzlichen, *irregulären* Verfeinerungsregeln zur Konformitätserhaltung,
3. einem *globalen* Algorithmus, der reguläre und irreguläre Verfeinerungen so kombiniert, daß die Konformitäts- und Stabilitätsbedingungen gleichzeitig erfüllt werden.

Eine stabile, reguläre Verfeinerungsstrategie zerlegt jedes nichtentartete Simplex $T \subset \mathbb{R}^n$ in 2^n Teilsimplizes, so daß die sukzessive Anwendung dieser Strategie eine stabile Familie konformer Triangulierungen von T liefert. Diese reguläre Strategie wird auch als rote Verfeinerung bezeichnet, die resultierenden Teilsimplizes als rote Simplizes. Die rote Verfeinerung wird auf diejenigen Simplizes angewendet, die vom Fehlerschätzer markiert wurden. Sie werden jedoch auch zum Abschluß der Triangulierung $\mathcal{T}$ benötigt. Die Konformität wird jedoch endgültig erst wieder mit irregulären oder grünen Verfeinerungen erreicht.
Sowohl die roten als auch die grünen Verfeinerungsregeln beschreiben lediglich die Verfeinerung einzelner Simplizes. Die Aufgabe des globalen Algorithmus ist es, die roten und grünen Markierungen so zu kombinieren, daß die resultierende Triangulierung $\mathcal{T}$ wieder konform und stabil ist. Dabei dürfen irreguläre Elemente nicht weiter

verfeinert werden, sondern müssen zuerst durch reguläre ersetzt werden. Die so erzeugten Simplizes dürfen weiter verfeinert werden (Abbildung 7.4 und 7.5). Durch dieses Vorgehen ist die Stabilität gewährleistet.

Zwei sehr beliebte Regeln für den 2-dimensionalen Fall wurden von R. E. Bank in dem Code PLTMG[8] vorgestellt. Sie lassen sich wie folgt beschreiben: Bei einer roten Verfeinerung wird ein gegebenes Dreieck T durch Verbindung seiner Kantenmittelpunkte in vier Teildreiecke zerlegt, die alle zueinander ähnlich sind (Abbildung 7.1). Bank verwendet nur einfache Bisektionen als irreguläre Verfeinerungen. Diese Verfeinerungen dienen ausschließlich dem grünen Abschluß und werden auf Dreiecke angewandt, die selbst unverfeinert sind und genau ein rot verfeinertes Nachbar-Dreieck besitzen (Abbildung 7.2). Reguläre Dreiecke, die mindestens zwei rote Nachbarn haben, oder irreguläre Dreiecke, die mindestens einen roten Nachbarn haben, werden rot verfeinert (Abbildung 7.3). Durch die reguläre Verfeinerung von Elementen mit zwei roten Nachbarn kann es zu Domino-Effekten bei der Erstellung des Abschlusses kommen.

Es ist denkbar, die irregulären Regeln von Bank um eine weitere grüne Verfeinerung zu erweitern[9].

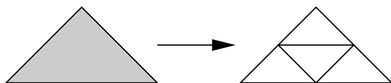

Abbildung 7.1: Rote Verfeinerung

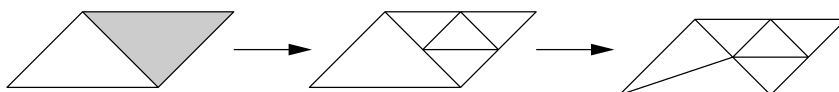

Abbildung 7.2: Abschluß bei einem roten Nachbarn

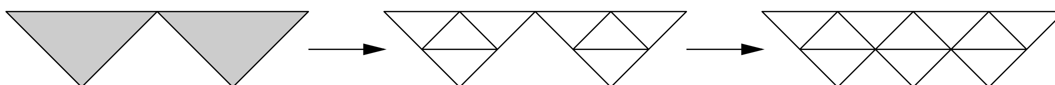

Abbildung 7.3: Abschluß bei zwei roten Nachbarn

Wie zuvor gesehen, dürfen irreguläre Simplizes selbst nicht weiter verfeinert werden. Grün verfeinerte Elemente, die zur Verfeinerung vorgesehen sind, müssen durch reguläre ersetzt werden, die dann weiter verfeinert werden können (Abbildung 7.4). Das gleiche gilt, wenn ein reguläres Nachbardreieck verfeinert werden soll (Abbildung 7.5).

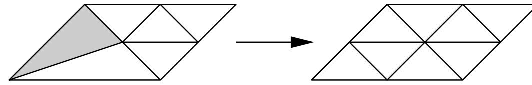

Abbildung 7.4: Direkte Ersetzung irregulärer Verfeinerungen

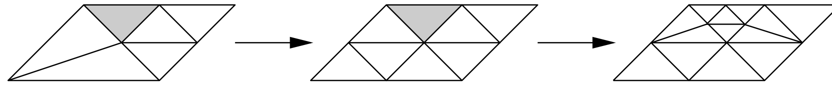

Abbildung 7.5: Indirekte Ersetzung irregulärer Verfeinerungen

Der globale Algorithmus, der die regulären Verfeinerungen mit dem grünen Abschluß kombiniert, wird in Kapitel 9 vorgestellt.

7.2 Weitere Verfahren im $\mathbb{R}^2$

Neben der oben vorgestellten Rot-Grün-Verfeinerungsstrategie existieren noch einige weitere, wie die Bisektionsverfahren von M. C. Rivara[16] und W. F. Mitchell [15]. Eine bewertende Diskussion über diese Verfahren wird in [9] geführt.

Kapitel 8

Der Algorithmus von Freudenthal

In dem vorigen Kapitel wurde ein Beispiel für eine stabile Verfeinerungsstrategie in zwei Raumdimensionen vorgestellt. Es stellt sich jedoch die Frage, wie solche Verfahren in höheren Dimensionen aussehen. Ein Verfahren zur Verallgemeinerung der regulären (roten) Verfeinerung wurde von H. FREUDENTHAL[11] vorgeschlagen. Im folgenden wird immer auf dem n-dimensionalen Einheitswürfel gearbeitet. Die so erhaltenen Ergebnisse lassen sich über affine Transformationen auf allgemeine Simplizes übertragen. Die Idee des Verfahrens ist in Abbildung 8.1 veranschaulicht.

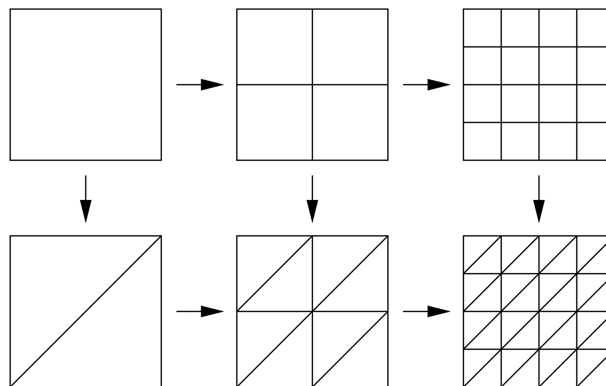

Abbildung 8.1: Triangulierungen des Einheitsquadrats

8.1 Die Kuhn-Triangulierung

Definition 8.1
Seien $C = [0,1]^n$ der Einheitswürfel und $e^{(1)}, \ldots, e^{(n)}$ die Standard-Einheitsvektoren des $\mathbb{R}^n$. Weiter sei Π_n die Menge aller Permutationen der Zahlen $\{1, \ldots, n\}$.
Für $\pi \in \Pi_n$ sei $T_\pi \subset C$ das Simplex

$$T_\pi = \left[x_\pi^{(0)}, \ldots, x_\pi^{(n)}\right]$$

mit den Eckpunkten

$$x_\pi^{(0)} = (0,0,\dots,0)^T, \qquad x_\pi^{(j)} = x_\pi^{(j-1)} + e^{(\pi(j))}, \qquad 1 \le j \le n. \tag{8.1}$$

Dann wird die Menge $\mathcal{K}(C) = \{T_\pi \mid \pi \in \Pi_n\}$ als Kuhn-Triangulierung von C bezeichnet.

Die Kuhn-Triangulierung besteht aus genau $n!$ Simplizes T_π, die alle das gleiche Volumen $\mathrm{vol}_{T_\pi} = \frac{1}{n!}$ besitzen. Alle Simplizes $T_\pi \in \mathcal{K}(C)$ haben die Eckpunkte $x_\pi^{(0)} = (0,0,\dots,0)^T$ und $x_\pi^{(n)} = (1,1,\dots,1)^T$ und damit auch die dazwischen liegende Kante. Abbildung 8.2 zeigt die Kuhn-Triangulierung in zwei Dimensionen.

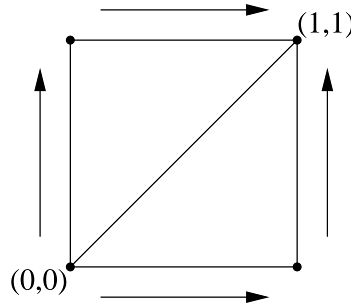

Abbildung 8.2: Triangulierung des Einheitsquadrates

Lemma 8.1
Sei $C = [0,1]^n$ der Einheitswürfel im $\mathbb{R}^n$.
Dann gilt für die Simplizes T_π der Kuhn-Triangulierung $\mathcal{K}(C)$ die Darstellung

$$T_\pi = \tilde{T}_\pi := \left\{ x \in C \;\middle|\; 0 \le x_{\pi(n)} \le \cdots \le x_{\pi(1)} \le 1 \right\}, \qquad \pi \in \Pi_n. \tag{8.2}$$

Beweis: Sei $\pi \in \Pi_n$ beliebig und zunächst $x \in T_\pi$. Mit Hilfe der baryzentrischen Koordinaten λ von x bzgl. T_π folgt

$$x = \sum_{j=0}^{n} \lambda_j x_\pi^{(j)} = \sum_{j=0}^{n} \lambda_j \sum_{k=1}^{j} e^{(\pi(k))} = \sum_{k=1}^{n} e^{(\pi(k))} \sum_{j=k}^{n} \lambda_j. \tag{8.3}$$

Da die $e^{(\pi(k))}$ die Einheitsvektoren sind, gilt

$$x_{\pi(k)} = \sum_{j=k}^{n} \lambda_j, \qquad 1 \le k \le n.$$

Wegen $x \in T_\pi$ gilt $\lambda_j \ge 0$ und $\sum_{j=0}^{n} \lambda_j = 1$ und somit folgt für $0 \le j \le n$

$$0 \le x_{\pi(n)} \le \cdots \le x_{\pi(1)} \le 1 \tag{8.4}$$

und somit $x \in \tilde{T}_\pi$. Damit ist $T_\pi \subset \tilde{T}_\pi$ gezeigt.
Zum Beweis der Umkehrung nehmen wir an, es sei $x \in \tilde{T}_\pi$, d.h. für die Komponenten von x gelte (8.4). Dann setzt man

$$\begin{aligned} \lambda_0 &:= 1 - x_{\pi(1)}, \\ \lambda_j &:= x_{\pi(j)} - x_{\pi(j+1)}, \qquad 1 \le j \le n-1, \\ \lambda_n &:= x_{\pi(n)}. \end{aligned}$$

Damit gilt $\sum_{j=0}^n \lambda_j = 1$. Aus (8.4) folgt weiter $\lambda_j \ge 0$ für $0 \le j \le n$. Darüber hinaus gilt

$$\sum_{j=k}^n \lambda_j = x_{\pi(k)}, \qquad 1 \le k \le n,$$

und damit durch Umkehrung von (8.3), daß die Zahlen $\lambda_0, \dots, \lambda_n$ die baryzentrischen Koordinaten von x bzgl. T_π sind. Wegen $\lambda_j \ge 0$ für $0 \le j \le n$ gehört x zu T_π.
Hiermit ist $T = \tilde{T}$ gezeigt. □

Lemma 8.2
Sei $C = [0,1]^n$ und $T = \left[x_\pi^{(0)}, \dots, x_\pi^{(n)}\right]$ eines der Simplizes in $\mathcal{K}(C)$. Weiter sei S ein (l)-Randsimplex von T_π, $0 \le l \le n$, d.h. es existieren $l+1$ Indizes $0 \le i_0 < \dots < i_l \le n$ mit $S = \left[x_\pi^{(i_0)}, \dots, x_\pi^{(i_l)}\right]$.
Dann besitzt S die Darstellung

$$S = \tilde{S} := \left\{x \in T_\pi \;\middle|\; x_{\pi(k)} = x_{\pi(k+1)} \textit{ für } 0 \le k \le n, k \notin \{i_0, \dots, i_l\}\right\}, \tag{8.5}$$

wobei zur Vereinfachung der Notation $x_{\pi(0)} := 1$ und $x_{\pi(n+1)} := 0$ gesetzt wurde.

Beweis: Sei $\pi \in \Pi_n$ und $S = \left[x_\pi^{(i_0)}, \dots, x_\pi^{(i_l)}\right]$ ein (l)-Randsimplex von T_π mit $0 \le l \le n$ und $0 \le i_0 < \dots < i_l \le n$.
Sei zunächst $x \in S$. Für die baryzentrischen Koordinaten λ von x bzgl. T_π gilt

$$\lambda_j = 0, \qquad 0 \le j \le n,\ j \ne i_0, \dots i_l. \tag{8.6}$$

Wie im Beweis von Lemma 8.1 gilt

$$x_{\pi(k)} = \sum_{j=k}^n \lambda_j, \qquad 1 \le k \le n.$$

Wegen $x_{\pi(0)} := 1$ und $x_{\pi(n+1)} := 0$ gilt diese Darstellung auch für $k = 0$ bzw. $k = n+1$. Aus (8.6) folgt daher

$$x_{\pi(k)} = x_{\pi(k+1)}, \qquad 0 \le k \le n,\ k \ne i_0, \dots, i_l,$$

und somit $x \in \tilde{S}$. Da $x \in S$ beliebig war, ist die Inklusion $S \subset \tilde{S}$ bewiesen.
Sei nun $x \in \tilde{S}$ beliebig. Analog zum Beweis von Lemma 8.1 sind die Koordinaten λ von x bzgl. T_π gegeben durch

$$\lambda_j := x_{\pi(j)} - x_{\pi(j+1)}, \qquad 0 \leq j \leq n.$$

Aus $x \in \tilde{S}$ folgt $\lambda_j = 0$ für $j \neq i_0, \ldots, i_l$. Folglich gilt $x \in \left[x_\pi^{(i_0)}, \ldots, x_\pi^{(i_l)}\right]$ und somit $x \in S$.
Damit ist $S = \tilde{S}$ bewiesen. □

Lemma 8.3 (Konformität der Kuhn-Triangulierung)
Sei $C = [0,1]^n$. Dann ist die Kuhn-Triangulierung $\mathcal{K}(C)$ eine konforme Triangulierung von C.

Beweis: Zuerst ist die Frage zu klären, ob es sich bei $\mathcal{K}(C)$ tatsächlich um eine Triangulierung von C handelt. Alle Elemente $T_\pi \in \mathcal{K}(C)$ besitzen das Volumen $\mathrm{vol}_{T_\pi} = \frac{1}{n!} > 0$. Des weiteren existiert für jeden Punkt $x \in C$ eine Permutation $\pi \in \Pi_n$ mit $0 \leq x_{\pi(n)} \leq \cdots \leq x_{\pi(1)} \leq 1$. Aus Lemma 8.1 folgt somit $C = \cup\{T_\pi \mid \pi \in \Pi_n\}$. Da für das Innere $\overset{\circ}{T}_\pi$ die Darstellung

$$\overset{\circ}{T}_\pi = \left\{x \in C \;\middle|\; 0 < x_{\pi(n)} < \cdots < x_{\pi(1)} < 1\right\}$$

gilt, können sich zwei verschiedene Simplizes $T_\pi, T_{\pi'} \in \mathcal{K}(C)$ höchstens am Rande schneiden. Somit erfüllt $\mathcal{K}(C)$ alle Voraussetzungen aus Definition 1.12 an eine Triangulierung.
Jetzt muß noch die Konformität von $\mathcal{K}(C)$ gezeigt werden. Seien dazu $\pi, \pi' \in \Pi_n$ mit $\pi \neq \pi'$ beliebig. Zu zeigen ist, daß $T_\pi \cap T_{\pi'}$ ein gemeinsames, niederdimensionales Randsimplex von T_π und $T_{\pi'}$ ist. Dazu zeigt man die Identität

$$T_\pi \cap T_{\pi'} = S := \mathcal{C}onv\left\{x_\pi^{(j)} \;\middle|\; x_\pi^{(j)} = x_{\pi'}^{(j)}; 0 \leq j \leq n\right\}, \tag{8.7}$$

woraus die Behauptung folgt.
Die Inklusion $S \subset T_\pi \cap T_{\pi'}$ ist klar, da S Teilmenge sowohl von T_π als auch von $T_{\pi'}$ ist. Es bleibt also noch die Umkehrung $T_\pi \cap T_{\pi'} \subset S$ zu zeigen.
Sei dazu $x \in T_\pi \cap T_{\pi'}$ beliebig. Wegen Lemma 8.1 gilt

$$1 \geq x_{\pi(1)} \geq \cdots \geq x_{\pi(n)} \geq 0, \tag{8.8}$$
$$1 \geq x_{\pi'(1)} \geq \cdots \geq x_{\pi'(n)} \geq 0. \tag{8.9}$$

Nun wird $i_0 := 0$ gesetzt und die Indizes i_ν, $\nu = 1, 2, \ldots$, durch

$$i_\nu := \min\{k > i_{\nu-1} \mid \{\pi(i_{\nu-1}+1), \ldots, \pi(k)\} = \{\pi'(i_{\nu-1}+1), \ldots, \pi'(k)\}\} \tag{8.10}$$

solange, bis für ein $l > 0$ schließlich $i_l = n$ gilt. Nun behaupten wir, daß mit den so definierten Indizes $0 = i_0 < i_1 < \cdots < i_l = n$ die Identität

$$x_{\pi(i_{\nu-1}+1)} = \cdots = x_{\pi(i_\nu)}, \qquad 1 \le \nu \le l, \tag{8.11}$$

gilt. Eine entsprechende Aussage gilt in diesem Fall natürlich auch für die Permutationen π'. Zum Beweis dieser Behauptung genügt es, den Fall $\nu = 1$ zu betrachten. Die übrigen Fälle folgen analog. Nimmt man also an, (8.11) sei für $\nu = 1$ falsch. Dann existiert ein Index i mit $1 < i \le i_1$ und

$$x_{\pi(1)} \ge \cdots \ge x_{\pi(i-1)} > x_{\pi(i)} \ge \cdots \ge x_{\pi(n)}.$$

für $1 \le j \le i-1$ und $i \le k \le i_1$ setzen wir $j' := (\pi')^{-1}(\pi(j))$ bzw. $k' := (\pi')^{-1}(\pi(k))$. Dann gilt $1 \le j', k' \le i_1$ und $\pi(j) = \pi'(j')$ bzw. $\pi(k) = \pi'(k')$. Aus (8.9) und

$$x_{\pi'(j')} = x_{\pi(j)} > x_{\pi(k)} \ge x_{\pi'(k')}$$

folgt $j' < k'$ für jedes solche Paar (j,k). Durch Variation von k' erhält man

$$j' < k' = (\pi')^{-1}(\pi(k)), \qquad i \le k \le i_1.$$

Da für unterschiedliche k auch die Indizes k' verschieden sind, gilt

$$j' = (\pi')^{-1}(\pi(j)) \le i-1, \qquad 1 \le j \le i-1,$$

und folglich

$$\{\pi(1), \ldots, \pi(i-1)\} = \{\pi'(1), \ldots, \pi'(i-1)\},$$

im Widerspruch zur Definition von i_1. Die Identität (8.11) ist somit bewiesen.
Aus (8.11) folgt aber mit Lemma 8.5 $x \in \left[x_\pi^{(i_0)}, \ldots, x_\pi^{(i_l)}\right]$. Da die entsprechende Aussage auch für π' gilt, folgt analog $x \in \left[x_{\pi'}^{(i_0)}, \ldots, x_{\pi'}^{(i_l)}\right]$. Wegen

$$x_\pi^{(i_k)} = \sum_{\nu=1}^{k} \left(\sum_{j=i_{\nu-1}+1}^{i_\nu} e^{(\pi(j))} \right) = \sum_{\nu=1}^{k} \left(\sum_{j=i_{\nu-1}+1}^{i_\nu} e^{(\pi'(j))} \right) = x_{\pi'}^{(i_k)}$$

für $0 \le k \le l$ gilt dann auch $x \in S$ und die Konformität von $\mathcal{K}(C)$ ist bewiesen. □

8.2 Verfeinerung der Kuhn-Triangulierung

Die bisherigen Ausführungen stellen eine konforme Triangulierung des n-dimensionalen Einheitswürfels $C = [0,1]^n$ bereit. Ziel ist es jedoch, eine stabile, konforme Verfeinerungsstrategie darzustellen. Dazu soll C in 2^n Teilwürfel zerlegt, und die

Kuhn-Triangulierungen der Teilwürfel gebildet werden. Diese Triangulierungen werden in $\mathcal{K}(C)$ zusammengesetzt und bilden eine konforme Triangulierung von C.
Im folgenden bezeichne $x := \{0, \frac{1}{2}\}^n$ einen Vektor im $I\!R^n$, dessen Einträge entweder den Wert 0 oder $\frac{1}{2}$ haben.
Sei nun $\mathcal{T}_0 := \mathcal{K}(C) = \{T_\pi \mid \pi \in \Pi_n\}$ die Kuhn-Triangulierung von C. C wird in 2^n Teilwürfel $C_v := v + \frac{1}{2}C$, $v \in \{0, \frac{1}{2}\}^n$ der Kantenlänge $\frac{1}{2}$ zerlegt, so daß gilt $C = \cup C_v$, $v \in \{0, \frac{1}{2}\}^n$. Analog zu Definition 1.16 sei $v + qC$ das Bild von C unter der affinen Transformation $F : x \mapsto v + qx$. Die zugehörige Kuhn-Triangulierung von C_v, $v \in \{0, \frac{1}{2}\}^n$, $\pi \in \Pi$ ergibt sich aus der entsprechenden Transformation der Kuhn-Triangulierung von C (vgl. Definition 1.18):

$$\mathcal{K}(C_v) := v + \frac{1}{2}\mathcal{K}(C) = \left\{ v + \frac{1}{2}T_\pi \mid \pi \in \Pi_n \right\}, \qquad v \in \{0, \frac{1}{2}\}^n.$$

Mit der Bezeichnung

$$T_{v,\pi} := v + \frac{1}{2}T_\pi, \qquad v \in \{0, \frac{1}{2}\}^n, \ \pi \in \Pi_n$$

kann die Kuhn-Triangulierung der Teilwürfel C_v als

$$\mathcal{K}(C_v) = \{T_{v,\pi} \mid \pi \in \Pi_n\}, \qquad v \in \{0, \frac{1}{2}\}^n$$

dargestellt werden.
Durch Vereinigung der Kuhn-Triangulierungen $\mathcal{K}(C_v)$ erhält man die Triangulierung

$$\mathcal{T}_1 := \bigcup_{\substack{\pi \in \Pi_n \\ v \in \{0, \frac{1}{2}\}}} \mathcal{K}(C_v) = \left\{ T_{v,\pi} \;\middle|\; v \in \{0, \frac{1}{2}\}^n, \pi \in \Pi_n \right\}. \tag{8.12}$$

Nun muß noch gezeigt werden, daß $\mathcal{T}_1$

- eine Verfeinerung von $\mathcal{T}_0$ ist,
- eine konforme Triangulierung ist,
- mittels dieses Verfahrens beliebig tief verfeinert werden kann. Dieser Punkt wird in der vorliegenden Arbeit nicht behandelt. Eine ausführliche Darstellung ist in [9] gegeben.

Lemma 8.4 (Verfeinerung der Kuhn-Triangulierung)
Sei $C = [0,1]^n$ und $\mathcal{T}_0$ die Kuhn-Triangulierung von C. Dann ist die durch (8.12) definierte Triangulierung $\mathcal{T}_1$ eine Verfeinerung von $\mathcal{T}_0$.

Beweis: Wenn gezeigt werden kann, daß ein beliebiges Element aus $\mathcal{T}_1$ einen Vorgänger aus $\mathcal{T}_0$ (Vorgänger bezeichnet ein Element aus $\mathcal{T}_0$, das das Element aus $\mathcal{T}_1$ enthält) besitzt, so ist die Behauptung bewiesen.

Seien $v \in \{0, \frac{1}{2}\}^n$ und $\pi \in \Pi_n$ beliebig. Gesucht wird eine Permutation $\hat{\pi} = \hat{\pi}(v, \pi)$ mit $T_{v,\pi} \subset T_{\hat{\pi}}$. Sei dazu $0 \leq k \leq n$ die Anzahl der Einträge v_i von v mit $v_i = \frac{1}{2}$. Dann existieren k Indizes $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit

$$1 \leq i_1 < \cdots < i_k \leq n, \qquad v_{\pi(i_1)} = \cdots = v_{\pi(i_k)} = \frac{1}{2}. \tag{8.13}$$

Die $n-k$ restlichen Indizes $i_{k+1}, \ldots, i_n \in \{1, \ldots, n\} \backslash \{i_1, \ldots, i_k\}$ können so angeordnet werden, daß

$$1 \leq i_{k+1} < \cdots < i_n \leq n, \qquad v_{\pi(i_{k+1})} = \cdots = v_{\pi(i_n)} = 0, \tag{8.14}$$

gilt. Hier und im folgenden werden in den Fällen $k = 0$ und $k = n$ diejenigen Gleichungen bzw. Ungleichungen übergangen, die keinen Sinn machen. Die Permutation $\hat{\pi}$ wird nun durch $\hat{\pi}(j) = \pi(i_j)$, $1 \leq j \leq n$ definiert. Nach (8.13) und (8.14) gilt also

$$v_{\hat{\pi}(1)} = \cdots = v_{\hat{\pi}(k)} = \frac{1}{2}, \qquad v_{\hat{\pi}(k+1)} = \cdots = v_{\hat{\pi}(n)} = 0.$$

Sei nun $x \in \frac{1}{2}T_\pi$; nach Lemma 8.1 ergibt sich $0 \leq x_{\pi(n)} \leq \cdots \leq x_{\pi(1)} \leq \frac{1}{2}$. Aus der Reihenfolge der Indizes i_j auf der linken Seite von (8.13) bzw. (8.14) folgt daher

$$0 \leq x_{\hat{\pi}(k)} \leq \cdots \leq x_{\hat{\pi}(1)} \quad \leq \frac{1}{2} \tag{8.15}$$

$$0 \leq x_{\hat{\pi}(n)} \leq \cdots \leq x_{\hat{\pi}(k+1)} \leq \frac{1}{2}. \tag{8.16}$$

Durch Addition von (8.15) mit $v_{\hat{\pi}(j)} = \frac{1}{2}$, $1 \leq j \leq k$ und (8.16) mit $v_{\hat{\pi}(j)} = 0$, $k+1 \leq j \leq n$ erhält man

$$\begin{aligned} 0 \leq v_{\hat{\pi}(n)} + x_{\hat{\pi}(n)} \leq \cdots \leq v_{\hat{\pi}(k+1)} + x_{\hat{\pi}(k+1)} &\leq \frac{1}{2} \\ \frac{1}{2} \leq v_{\hat{\pi}(k)} + x_{\hat{\pi}(k)} \leq \cdots \leq v_{\hat{\pi}(1)} + x_{\hat{\pi}(1)} \qquad &\leq 1 \end{aligned}$$

oder — damit gleichbedeutend —

$$v + x \in T_{\hat{\pi}}, \qquad x \in \frac{1}{2}T_\pi.$$

Also gilt $T_{v,\pi} \subset T_{\hat{\pi}} \in \mathcal{T}_0$. Da $T_{v,\pi} \in \mathcal{T}_1$ beliebig gewählt war und die Eckpunkte von $\mathcal{T}_1$ nach Konstruktion mit den Eck- und Kantenmittelpunkten von $\mathcal{T}_0$ übereinstimmen, ist $\mathcal{T}_1$ tatsächlich eine Verfeinerung von $\mathcal{T}_0$. $\square$

Lemma 8.5 (Konformität von $\mathcal{T}_1$)
Sei $C = [0,1]^n$ und $\mathcal{T}_0$ die Kuhn-Triangulierung von C.
Dann ist die durch (8.12) definierte Triangulierung konform.

Beweis: Seien $T_{v,\pi} = \left[x_{v,\pi}^{(0)}, \dots, x_{v,\pi}^{(n)}\right]$ und $T_{v',\pi'} = \left[x_{x',\pi'}^{(0)}, \dots, x_{x',\pi'}^{(n)}\right]$ zwei Elemente der Triangulierung $\mathcal{T}_1$ mit $v, v' \in \{0, \frac{1}{2}\}^n$ und $\pi, \pi' \in \Pi_n$. Da die Triangulierung $\mathcal{K}(C_v)$ wegen Lemma 8.3 und Lemma 1.7 konform ist, kann o.B.d.A. $v \neq v'$ angenommen werden. Wir verwenden die Darstellung

$$\begin{aligned} T_{v,\pi} &= \left\{ x + v \;\middle|\; \frac{1}{2} \geq x_{\pi(1)} \geq \cdots \geq x_{\pi(n)} \geq 0 \right\}, \\ T_{v',\pi'} &= \left\{ x + v' \;\middle|\; \frac{1}{2} \geq x_{\pi'(1)} \geq \cdots \geq x_{\pi'(n)} \geq 0 \right\}. \end{aligned}$$

Die entsprechenden Teilwürfel C_v und $C_{v'}$ sind gegeben durch

$$\begin{aligned} C_v &= \left\{ x + v \;\middle|\; 0 \leq x_k \leq \frac{1}{2},\ 1 \leq k \leq n \right\}, \\ C_{v'} &= \left\{ x + v' \;\middle|\; 0 \leq x_k \leq \frac{1}{2},\ 1 \leq k \leq n \right\}. \end{aligned}$$

Seien $i_1 < \cdots < i_l$ die Indizes in $\{1, \dots, n\}$ mit $v_{\pi(i_k)} = v'_{\pi(i_k)}$, $1 \leq k \leq l$. Entsprechend seien $j_1 < \cdots < j_{n-l}$ die Indizes, für die $v_{\pi(j_k)} \neq v'_{\pi(j_k)}$ gilt. Für jeden der Indizes j_k gilt also entweder $v_{\pi(j_k)} = 0$ und $v'_{\pi(j_k)} = \frac{1}{2}$ oder $v_{\pi(j_k)} = \frac{1}{2}$ und $v'_{\pi(j_k)} = 0$. Damit hat die gemeinsame (l)-Hyperfläche $H := C_v \cap C_{v'}$ von C_v bzw. $C_{v'}$ die Darstellung

$$H = \left\{ x + v \;\middle|\; 0 \leq x_{\pi(i_k)} \leq \frac{1}{2},\ 1 \leq k \leq l;\ x_{\pi(j_k)} + v_{\pi(j_k)} = \frac{1}{2},\ 1 \leq k \leq n - l \right\}.$$

Für den Schnitt von $T_{v,\pi}$ mit H gilt also

$$T_{v,\pi} \cap H = \left\{ x + v \;\middle|\; \frac{1}{2} \geq x_{\pi(1)} \geq \cdots \geq x_{\pi(n)} \geq 0;\ x_{\pi(j_k)} + v_{\pi(j_k)} = \frac{1}{2},\ 1 \leq k \leq n - l \right\}.$$

Unter der Voraussetzung, daß die entsprechenden Mengen nichtleer sind, setzt man

$$a := \max \left\{ j_k \;\middle|\; 1 \leq k \leq n - l;\ v_{\pi(j_k)} = 0 \right\}, \tag{8.17}$$

$$b := \min \left\{ j_k \;\middle|\; 1 \leq k \leq n - l;\ v_{\pi(j_k)} = \frac{1}{2} \right\}. \tag{8.18}$$

Falls die Menge auf der rechten Seite von (8.17) leer ist, so setzt man $a := 0$. Entsprechend sei $b := n + 1$, falls die rechte Seite von (8.18) nicht existiert.

Sei nun $x+v \in T_{v,\pi} \cap H$ und zunächst $a > 0$. Dann existiert ein Index $1 \leq k \leq n-l$ mit $a = j_k$, und für $1 \leq i \leq a$ gilt

$$\frac{1}{2} \geq x_{\pi(i)} \geq x_{\pi(a)} = x_{j_k} = x_{j_k} + v_{j_k} = \frac{1}{2}.$$

Also gilt $x_{\pi(i)} = \frac{1}{2}$ für $1 \leq i \leq a$. Für $a = 0$ ist die Aussage klar. Analog gilt $x_{\pi(i)} = 0$ für $b \leq i \leq n$. Daraus ergibt sich die Darstellung

$$T_{v,\pi} \cap H = \left\{ x+v \;\middle|\; \frac{1}{2} = x_{\pi(1)} = \cdots = x_{\pi(a)} \geq \cdots \geq x_{\pi(b)} = \cdots = x_{\pi(n)} = 0 \right\}. \tag{8.19}$$

Falls $b \leq a$ gilt $T_{v,\pi} \cap H = \emptyset$ und damit auch $T_{v,\pi} \cap T_{v',\pi'} = \emptyset$. Sei also o.B.d.A. $b > a$. In diesem Fall gehören alle Indizes zwischen a und b zu der Menge $\{i_1, \ldots, i_l\}$. a ist folglich der Index des ersten Eckpunktes von $T_{v,\pi}$, der in H liegt. b ist der erste Index nach a, für den dies nicht mehr gilt. Aus (8.19) folgt nämlich mit Lemma 8.5 die Darstellung

$$T_{v,\pi} \cap H = \left[x_{v,\pi}^{(a)}, \ldots, x_{v,\pi}^{(b-1)}\right],$$

d.h., es gilt $x_{v,\pi}^{(k)} \in H$ genau dann, wenn $a \leq k < b$ gilt. Eine ähnliche Darstellung gilt auch für $T_{v',\pi'} \cap H$. Seien dazu $i'_1 < \cdots < i'_l$ die Indizes mit $v_{\pi'(i'_k)} = v'_{\pi'(i'_k)}$ und entsprechend $j'_1 < \cdots < j'_{n-l}$ die Indizes mit $v_{\pi'(j'_k)} \neq v'_{\pi'(j'_k)}$. Man definiert

$$\begin{aligned} a' &:= \max\left\{ j'_k \;\middle|\; 1 \leq k \leq n-l;\; v'_{\pi'(j'_k)} = 0 \right\}, \\ b' &:= \min\left\{ j'_k \;\middle|\; 1 \leq k \leq n-l;\; v'_{\pi'(j'_k)} = \frac{1}{2} \right\}, \end{aligned}$$

falls die entsprechenden Mengen auf der rechten Seite nichtleer sind. Wie oben sei sonst $a' := 0$ bzw. $b' := n+1$. Dann gilt die Darstellung

$$T_{v',\pi'} \cap H = \left[x_{v',\pi'}^{(a')}, \ldots, x_{v',\pi'}^{(b'-1)}\right].$$

Sei nun $a_0 := 0$ und die Indizes s_ν, $\nu = 1, 2, \ldots$ definiert durch

$$s_\nu := \min\left\{ k > s_{\nu-1} \;\middle|\; \{\pi(i_{s_{\nu-1}+1}), \ldots, \pi(i_k)\} = \{\pi'(i'_{s_{\nu-1}+1}), \ldots, \pi'(i'_k)\} \right\},$$

und so lange, bis schließlich für ein $m > 0$ $i_{s_m} = i_l$ gilt. Wie im Beweis zu Lemma 8.3 kann man schließen, daß für jeden Punkt $x \in T_{v,\pi} \cap T_{v',\pi'}$ und für $1 \leq \nu \leq m$ die Identität

$$x_{\pi(i_{s_{\nu-1}+1})} = \cdots = x_{\pi(i_{s_\nu})} = x_{\pi'(i'_{s_{\nu-1}+1})} = \cdots = x_{\pi'(i'_{s_\nu})} \tag{8.20}$$

gilt. Im folgenden wird $i_0 := 0$, $x_{\pi(0)} := \frac{1}{2}$ und $s_{m+1} := l+1$, $i_{l+1} := n+1$, $x_{\pi(n+1)} := 0$ gesetzt. Dadurch sind die folgenden Indizes $0 \leq p, q \leq m+1$ wohldefiniert:

$$\begin{aligned} p &:= \min\left\{0 \leq \nu \leq m+1 \;\middle|\; i_{s_\nu} \geq a;\; i'_{s_\nu} \geq a'\right\}, \\ q &:= \max\left\{0 \leq \nu \leq m+1 \;\middle|\; i_{s_\nu} < b;\; i'_{s_\nu} < b'\right\}. \end{aligned}$$

Sei nun $x \in T_{v,\pi} \cap T_{v',\pi'}$ beliebig. Aus (8.19) und (8.20) schließt man $x_{\pi(i_k)} = \frac{1}{2}$ für alle Indizes $0 \leq k \leq s_p$. Umgekehrt gilt $x_{\pi(i_k)} = 0$ für $s_q < k \leq l+1$. Für die übrigen Indizes $s_p < k \leq s_q$ erhält man die Ungleichungskette

$$\begin{aligned} \frac{1}{2} = x_{\pi(i_{s_p})} &\geq x_{\pi(i_{s_p+1})} = \cdots = x_{\pi(i_{s_{p+1}})} \\ &\geq \cdots \\ &\geq x_{\pi(i_{s_{q-1}+1})} = \cdots = x_{\pi(i_{s_q})} \\ &\geq x_{\pi(i_{s_q+1})} = 0. \end{aligned} \tag{8.21}$$

Aus (8.21) folgt zunächst, daß $T_{v,\pi} \cap T_{v',\pi'}$ im Fall $q < p$ leer ist. Sei also zunächst $q \geq p$. In diesem Fall kann aus Lemma 8.5 geschlossen werden, daß x in dem Randsimplex $S := \left[x_{v,\pi}^{(i_{s_p})}, x_{v,\pi}^{(i_{s_{p+1}})}, \ldots, x_{v,\pi}^{(i_{s_q})}\right]$ von $T_{v,\pi}$ liegt. Da $x \in T_{v,\pi} \cap T_{v',\pi'}$ beliebig gewählt war, gilt

$$T_{v,\pi} \cap T_{v',\pi'} \subset S := \left[x_{v,\pi}^{(i_{s_p})}, x_{v,\pi}^{(i_{s_{p+1}})}, \ldots, x_{v,\pi}^{(i_{s_q})}\right].$$

Analog dazu zeigt man

$$T_{v,\pi} \cap T_{v',\pi'} \subset S' := \left[x_{v',\pi'}^{(i'_{s_p})}, x_{v',\pi'}^{(i'_{s_{p+1}})}, \ldots, x_{v',\pi'}^{(i'_{s_q})}\right].$$

Falls nun noch gezeigt wird, daß die Eckpunkte von S und S' identisch sind, so folgt $S = S' \subset T_{v,\pi} \cap T_{v',\pi'}$ und somit $S = T_{v,\pi} \cap T_{v',\pi'}$. In diesem Fall ist $T_{v,\pi} \cap T_{v',\pi'}$ ein gemeinsames Randsimplex von $T_{v,\pi}$ bzw. $T_{v',\pi'}$ und die Behauptung wäre bewiesen. Sei also $p \leq \nu \leq q$ beliebig. Für den Eckpunkt $x_{v,\pi}^{i_{s_\nu}}$ von $T_{v,\pi}$ gilt

$$x_{v,\pi}^{(i_{s_\nu})} = v + \frac{1}{2} x_{\pi}^{(i_{s_\nu})} = \sum_{i=1}^{n} v_{\pi(i)} e^{(\pi(i))} + \sum_{i=1}^{i_{s_\nu}} \frac{1}{2} e^{(\pi(i))}.$$

Nun werden die beiden Summen aufgespalten. Dann wird getrennt über die Indizes i_k bzw. j_k aufsummiert. Für die Indizes $j_k \leq i_{s_\nu}$ muß wegen $i_{s_\nu} < b$ die Ungleichung $j_k \leq a$ gelten. Damit erhält man

$$x_{v,\pi}^{(i_{s_\nu})} = \sum_{k=1}^{l} v_{\pi(i_k)} e^{(\pi(i_k))} + \sum_{k=1}^{n-l} v_{\pi(j_k)} e^{(\pi(j_k))} + \sum_{k=1}^{s_\nu} \frac{1}{2} e^{(\pi(i_k))} + \sum_{\substack{k=1 \\ j_k \leq a}}^{n-l} \frac{1}{2} e^{(\pi(j_k))}.$$

Wegen $v_{\pi(j_k)} = 0$ für $j_k \le a$ und $v_{\pi(j_k)} = \frac{1}{2}$ für $j_k > a$ gilt weiter

$$x_{v,\pi}^{(i_{s_\nu})} = \sum_{k=1}^{l} v_{\pi(i_k)} e^{(\pi(i_k))} + \sum_{k=1}^{n-l} \frac{1}{2} e^{(\pi(j_k))} + \sum_{k=1}^{s_\nu} \frac{1}{2} e^{(\pi(i_k))}.$$

Nun gelten für die Indizes i_k, j_k die Beziehungen

$$\{\pi(i_1), \dots, \pi(i_l)\} = \{\pi'(i'_1), \dots, \pi'(i'_l)\}$$

sowie

$$\{\pi(j_1), \dots, \pi(j_l)\} = \{\pi'(j'_1), \dots, \pi'(j'_l)\}\,.$$

Nach Definition der Zahlen s_ν gilt außerdem

$$\{\pi(i_1), \dots, \pi(i_{s_\nu})\} = \left\{\pi'(i'_1), \dots, \pi'(i'_{s_\nu})\right\}.$$

Mit $v_{\pi(i_k)} = v'_{\pi'(i'_k)}$ für $1 \le k < l$ erhält man

$$x_{v,\pi}^{(i_{s_\nu})} = \sum_{k=1}^{l} v'_{\pi'(i'_k)} e^{(\pi'(i'_k))} + \sum_{k=1}^{n-l} \frac{1}{2} e^{(\pi'(j'_k))} + \sum_{k=1}^{s_\nu} \frac{1}{2} e^{(\pi'(i'_k))} = x_{v',\pi'}^{(i'_{s_\nu})}.$$

Damit ist die Konformität der Triangulierung $\mathcal{T}_1$ bewiesen. □

Der Beweis zu Lemma 8.4 enthält bereits den Freudenthalschen Algorithmus. Geht man den Beweis "rückwärts" durch, läßt sich nämlich folgende Verfeinerungsstrategie für die einzelnen Simplizes T_π von $\mathcal{T}_0$ ablesen:

Bemerkung 8.1
Sei $C = [0,1]^n$, $\mathcal{T}_0$ die Kuhn-Triangulierung von C und $\mathcal{T}_1$ die durch (8.12) definierte Verfeinerung von $\mathcal{T}_0$. Für $\hat{\pi} \in \Pi_n$ sei $\mathcal{S}(T_{\hat{\pi}}) = \{T \in \mathcal{T}_1 \mid T \subset T_{\hat{\pi}}\}$ die durch $\mathcal{T}_1$ induzierte Verfeinerung von $T_{\hat{\pi}} \in \mathcal{T}_0$.
Dann besteht $\mathcal{S}(T_{\hat{\pi}})$ aus allen Elementen $T_{v,\pi} \in \mathcal{T}_1$ mit

$$v_{\hat{\pi}(1)} = \cdots = v_{\hat{\pi}(k)} = \frac{1}{2}, \qquad v_{\hat{\pi}(k+1)} = \cdots = v_{\hat{\pi}(n)} = 0,$$

(das war die rechte Seite von (8.13) und (8.14)) und

$$(\pi^{-1} \circ \hat{\pi})(1) < \cdots < (\pi^{-1} \circ \hat{\pi})(k), \qquad (\pi^{-1} \circ \hat{\pi})(k+1) < \cdots < (\pi^{-1} \circ \hat{\pi})(n),$$

für einen Index $0 \le k \le n$. Die letzte Gleichung kommt aus der linken Seite von (8.13) und (8.14) unter der Voraussetzung aus dem obigen Beweis, daß π mit $\hat{\pi}(j) = \pi(i_j)$ eine passende Permutation ist. □

Bemerkung 8.2
Sei $0 \leq k \leq n$ und $\hat{\pi} \in \Pi_n$. Dann gibt es genau $\binom{n}{k}$ Möglichkeiten, k Positionen in π für die Zahlen $\hat{\pi}(1), \ldots, \hat{\pi}(k)$ auszuwählen, also $\binom{n}{k}$ Möglichkeiten für π, der Bedingung (8.1) zu genügen. Die Gesamtzahl der Simplizes in $\mathcal{S}(T_{\hat{\pi}})$ beträgt damit $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ und $\mathcal{S}(T_{\hat{\pi}})$ genügt Definition 6.2 und ist regulär.
Die verfeinerten Elemente des Algorithmus können durch Rotation und Skalierung in ihre Vater-Elemente transformiert werden. Somit ergibt die Strategie eine stabile Familie sukzessiv verfeinerter Triangulierungen. □

Bemerkung 8.3 (Der Algorithmus von Freudenthal)
Sei $T = \left[x^{(0)}, \ldots, x^{(n)}\right]$ ein Simplex im $\mathbb{R}^n$. Dann ergibt folgender Algorithmus eine stabile, konforme Verfeinerung dieses Simplexes:

`Durchlaufe mit` $l \in [0, n]$:
  $v^{(0)} := \frac{1}{2}(x^{(0)} + x^{(k)})$
  `Durchlaufe alle` $\pi \in \Pi_n$:
    `Falls` $(\pi^{-1}(1) < \ldots < \pi^{-1}(k))$ `und` $(\pi^{-1}(k+1) < \ldots < \pi^{-1}(n))$:
      `Durchlaufe mit` $l \in [1, n]$:
        $v^{(l)} := v^{(l-1)} + \frac{1}{2}(x^{\pi(l))} - x^{(\pi(l)-1)})$
        $T_{k,\pi} := \left[v^{(0)}, \ldots, v^{(\pi)}\right]$ □

Bemerkung 8.4
Der Algorithmus von Freudenthal stellt lediglich eine rote Verfeinerung in beliebiger Raumdimension zur Verfügung. Entsprechende Regeln für den grünen Abschluß im $\mathbb{R}^3$ werden in [9] angegeben. □

Kapitel 9

Realisierung der Verfeinerung

In Kapitel 7 wurde ein stabiles Verfeinerungsverfahren für Triangulierungen im $\mathbb{R}^2$ eingeführt. Dieses Verfahren soll jetzt mit einem globalen Verfeinerungsalgorithmus zur Verfeinerung von Triangulierungen benutzt werden. Dabei ist die Konformität der Triangulierung, wie in den in Kapitel 7.1 behandelten Regeln beschrieben, stets zu erhalten.

9.1 Verfeinerung für mehr als zwei Raumdimensionen

In der aktuellen Implementierung sind lediglich zwei Raumdimensionen vorgesehen, die Erweiterung auf höhere Dimensionen ist jedoch bereits eingeplant. Das ist möglich, da die Kuhn-Triangulierung der Randsimplizes mit der Triangulierung der Simplizes übereinstimmt. Von der niedrigsten Dimension ausgehend, ist jedes Objekt selbst für seine Verfeinerung zuständig. Sollte beispielsweise auf drei Dimensionen ausgebaut werden, müssen diese Objekte ihre eigenen Regeln mitbringen, der Rest ist bereits vorhanden.

9.2 Markieren der Simplizes

Bei der Markierung (k)-Simplex werden seine $(k-1)$-Randsimplizes automatisch markiert. Diese Randsimplizes markieren wiederum ihre Randsimplizes. Durch diesen Mechanismus genügt es, die höchstdimensionalen Objekte — also die Elemente — zu markieren.
Der Markierungsvorgang bei den Elementen ist der komplexeste, da zwischen regulären und irregulären Elementen unterschieden werden muß. Die Regeln verbieten die irreguläre Verfeinerung bereits irregulärer Elemente. Daher muß darauf geachtet werden, daß nicht das irreguläre Element, sondern sein Vater markiert und verfeinert wird.

Folgende internen Markierungen stehen zusätzlich zu den auf Seite 19 angegebenen zur Verfügung:

- `none`: keine Verfeinerung,
- `done`: Element ist verfeinert und fertig,
- `rm`: Element soll entfernt werden.

9.3 Verfeinerung der Komponenten einer Triangulierung

Die Verfeinerung findet für jede Dimension einzeln statt. Das bedeutet, daß keine globalen Regeln zur Verfeinerung existieren, sondern daß jedes Objekt nur seine eigenen Regeln kennt.
Begonnen wird mit dem Aufruf an jedes Objekt der niedrigsten Dimension — also Knoten (siehe unten) — sich auf das neue Level zu verfeinern. Danach erfolgt die Aufforderung an die nächsthöhere Dimension sich zu verfeinern. Dies geschieht solange, bis die Raumdimension erreicht ist.
Jedes Simplex kann beim Übergang zum nächsten Level unverfeinert bleiben. Dann muß dieses Simplex jedoch der verfeinerten Triangulierung bekannt gemacht werden, indem es für diese registriert wird. Dieser Vorgang wird im folgenden auch als Kopieren auf das nächste Level bezeichnet.
Nach jedem Verfeinerungsvorgang wird die Markierung des ursprünglichen Simplex auf `done` gesetzt, die eines neu erzeugten ist `none`.

9.3.1 Knoten

Ein Knoten interpretiert alle Markierungen bis auf `done` als Verfeinerung auf das nächste Level. Ein Verfeinerungsaufruf an einen Knoten mit Markierung `done` wird ignoriert, alle anderen Aufrufe registrieren den Knoten in dem nächsten (oder dem angebenen) Level.

9.3.2 Segmente

Segmente können neben dem Kopieren — das analog zu dem Vorgang bei den Knoten abläuft — auf das nächste Level auch regulär verfeinert werden.
Durch den Verfeinerungsvorgang werden ein Knoten und zwei Segmente erzeugt. Der Knoten halbiert das ursprüngliche Segment. Die Segmente bestehen jeweils aus dem neuen Knoten und einem der beiden auf das aktuelle Level verfeinerten Knoten des Vater-Segments.
Die Markierungen `rm` und `done` werden ignoriert; `green` und `none` registrieren das Segment unverfeinert für das nächste Level. Durch `red` wird der reguläre Verfeinerungsvorgang ausgelöst.

9.3.3 Elemente

Das Verfeinern der Elemente ist der derzeit aufwendigste Teil des Gesamtvorgangs, da Elemente nicht nur regulär, sondern auch irregulär verfeinert werden können. Insbesondere müssen irregulär verfeinerte Elemente weiter verfeinert werden können, indem ihre irreguläre Verfeinerung zurückgenommen wird.
Das aktuelle Level werde mit k, das Level, auf das verfeinert werden soll mit $k+1$ bezeichnet.
Unverfeinerte Elemente werden wie die Knoten und Segmente für das Level $k+1$ registriert.

Die reguläre Verfeinerung

Es sollen nicht nur reguläre, sondern auch irreguläre Elemente verfeinert werden können. Da die irregulären Elemente selbst nicht verfeinert werden dürfen, müssen deren Eltern auf das Ziel-Level verfeinert werden. Folglich haben diese Elemente Nachfahren auf mehr als einem Level.
Ablauf der roten Verfeinerung eines regulären Elements E von Level k nach Level $k+1$:

1. Suche alle Knotenpunkte $P_{1,\ldots,3}$ von E.
2. Durchlaufe alle Punkte P_i, $i = 1, \ldots, 3$.
   (a) Suche die Segmente S_{i_1} und S_{i_2} auf Level $k+1$, die Kinder der Segmente von E auf Level k sind, die von Punkt P_i ausgehen.
   (b) Suche die Punkte P_{i_j} $j = 1, 2$ der Segmente S_{i_j}, die von P_i verschieden sind.
   (c) Erzeuge aus den Punkten P_{i_1} und P_{i_2} ein Segment S_{i+3} auf Level $k+1$.
   (d) Erzeuge ein Element aus den Segmenten S_{i_1}, S_{i_2} und S_{i+3} auf Level $k+1$.
3. Erzeuge ein Element S_4, S_5 und S_6 auf Level $k+1$.

Die irreguläre Verfeinerung

Der Fall einer wiederholten irregulären Verfeinerung eines Elements E wird im Vorfeld abgefangen.
Im folgenden wird vorausgesetzt, daß das Element E regulär ist und alle Segmente, aus denen E aufgebaut ist, bereits auf Level $k+1$ verfeinert sind. Das bedeutet, daß ein beliebiges dieser Segmente rot verfeinert ist, die anderen beiden unverfeinert für Level $k+1$ registriert wurden.
Ablauf der grünen Verfeinerung eines regulären Elements E von Level k nach Level $k+1$:

1. Suche das Segment S_1 von E, das auf Level $k+1$ rot verfeinert wurde, also zwei Nachfahren auf Level $k+1$ hat. S_{1_1} und S_{1_2} seien diese beiden Segmente.

2. Suche den gemeinsamen Punkt P_1 von S_{1_1} und S_{1_2}, sowie deren Randpunkte P_2 und P_3 (das sind die Eckpunkte von S_1). P_2 sei Randpunkt von S_{1_1}, P_3 von S_{1_2}.

3. Suche den dritten Eckpunkt von E, also den Punkt, der dem Segment S_1 gegenüberliegt.

4. Suche die Segmente S_2 und S_3 von E für die gilt, daß S_2 P_2 und P_4, S_3 P_3 und P_4 verbindet.

5. Erzeuge auf Level $k+1$ ein Segment S_4, das P_4 und P_1 verbindet.

6. Erzeuge auf Level $k+1$ zwei Elemente aus den Segmenten S_{1_1}, S_2, S_4 und S_{1_2}, S_3, S_4.

9.4 Der globale Algorithmus

Die vorgestellten Mechanismen zur Verfeinerung der Simplizes müssen noch so zusammengestellt werden, daß die resultierende Triangulierung konform ist. Dazu muß vor der globalen Verfeinerung der Abschluß berechnet werden.

9.4.1 Der rote Abschluß

Die Bestimmung des Abschlusses — unabhängig ob rot oder grün — wird in der jeweiligen Triangulierung durchgeführt. Mit der Berechnung des Abschlusses findet noch keine Verfeinerung statt; es werden lediglich Elemente mit roten Markierungen versehen.

Definition 9.1 (Hängender Knoten)
Ein Element, das auf das höchste Level verfeinert werden soll, hat einen hängenden Knoten, falls mindestens eines seiner Segmente bereits verfeinert ist oder zur Verfeinerung markiert ist.

Der rote Abschluß wird gebildet, indem so lange alle Elemente markiert werden, die noch nicht rot markiert sind und entweder regulär sind und mindestens zwei hängende Knoten haben oder irregulär sind und mindestens einen hängenden Knoten haben.
Durch die Markierung der Elemente werden auch deren Segmente markiert (wie oben beschrieben). Somit können sich die Markierungen fortpflanzen.

9.4.2 Der grüne Abschluß

Der grüne Abschluß dient dazu, die Triangulierung in einen konformen Zustand zu befördern. Die durch solch eine Verfeinerung erzeugten Elemente sind irregulär. Dieser Abschluß wird gebildet, nachdem der rote Abschluß auf dem nächsten Level erzeugt — aber noch nicht verfeinert — wurde. Es wird vorausgesetzt, daß jedes Element höchstens einen hängenden Knoten hat.
Der grüne Abschluß wird gebildet, indem alle Elemente, die nicht `red` markiert sind und einen hängenden Knoten haben, `green` markiert werden.

9.4.3 Der Algorithmus

1. Entferne alle irregulären Markierungen.
2. Markiere alle Knoten `red`.
3. Ersetze die `done`-Markierungen aller Segmente durch `none`. Dadurch wird sichergestellt, daß alle Segmente auf Level $k+1$ enthalten sein werden.
4. Verfeinere alle Knoten auf Level $k+1$.
5. Berechne den roten Abschluß (9.4.1).
6. Verfeinere alle Segmente auf Level $k+1$.
7. Verfeinere alle Elemente auf Level $k+1$.
8. Setze das aktuelle Level auf $k+1$
9. Solange durch den roten Abschluß (9.4.1) Elemente markiert werden:
   (a) Verfeinere alle `red` markierten Segmente innerhalb von Level $k+1$. Entferne das ursprüngliche Segment von Level $k+1$.
   (b) Verfeinere alle `red` markierten Elemente innerhalb von Level $k+1$. Entferne das ursprüngliche Element von Level $k+1$.
10. Berechne den grünen Abschluß (9.4.2).
11. Entferne alle Elemente, die `green` markiert sind aus dem Level $k+1$, verfeinere diese dann auf dieses Level.

Durch den Schritt (9b) können Elemente entstehen, die zu keinem Level gehören. Dies stellt jedoch kein Problem dar und muß nur dann beachtet werden, wenn ein Level gelöscht werden soll.

Kapitel 10

Ergebnisse

Abschließend sollen einige mit dem entwickelten Code **meshMan** erstellte Triangulierungen gezeigt werden. Zu den Triangulierungen werden die zugehörigen Qualitätsdiagramme angegeben. In diesen Diagrammen ist in x-Richtung der Kehrwert der Qualität aufgetragen. Der Kehrwert wurde gewählt, um eine Einschränkung auf das Intervall $(0, 1]$ zu erreichen. "Optimale" Simplizes haben folglich den Wert 1, entartete 0. Das verwendete Qualitätskriterium ist $\frac{1}{2}\frac{\rho_T}{h_T}$; ρ_T ist der Umkreis-, h_T der Inkreisradius des Simplexes T. Auf der y-Achse ist der prozentuale Anteil einer Qualitätsklasse abgetragen. Dadurch sind die Diagramme direkt vergleichbar.

10.1 Einfluß der Parameter auf die Generatoren

In diesem Abschnitt soll untersucht werden, wie stark sich die Generatoren zur Erzeugung von Ausgangstriangulierungen durch ihre Parameter steuern lassen bzw. in wieweit die Standardeinstellungen vernünftig gewählt sind.

10.1.1 Der 1D-Generator

Der einzige Parameter P des 1D-Generators bestimmt die Länge der zu erzeugenden Segmente. Für $P < 0$ wird zuerst das kürzeste Segment gesucht. Dessen Länge wird dann als Referenzlänge R bezeichnet. Alle Segmente, die länger als $-PR$ sind, werden in kürzere Segmente dieser Länge zerlegt. Ist $P > 0$, werden alle Segmente, die länger als P sind, auf Länge P verfeinert. Um die vorgegebenen Segmente vollständig zu überdecken, werden die Ergebnisse gerundet.
Der Parameter -1 bewirkt also die Aufteilung jedes Segments in Teilstücke, die alle die Länge des zuvor kürzesten Segments aufweisen; -2 würde Segmente mit der doppelten Länge des kürzesten erzeugen.
Eine ähnliche Länge aller Randsegmente ist im allgemeinen erstrebenswert, da damit die gleichmäßigsten Triangulierungen erzeugt werden können. Folglich ist standardmäßig ein Wert von -1 eingestellt.

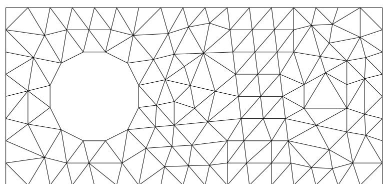

Abbildung 10.1: Parameter -1.0

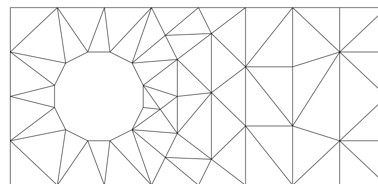

Abbildung 10.2: Parameter -2.0

Liegen jedoch sehr kurze und sehr lange Segmente in einer Geometrie vor, so kann ein Faktor von -1 einen erheblichen Rechenmehraufwand bedeuten. Dann ist es meistens sinnvoll, höhere Werte einzustellen (Abbildungen 10.3 und 10.4).

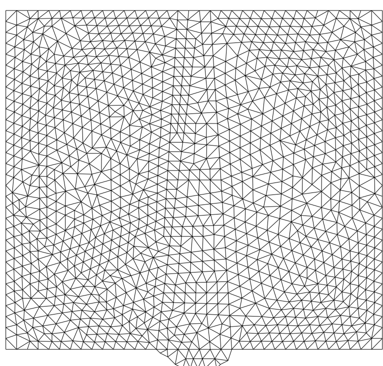

Abbildung 10.3: Parameter -2.0

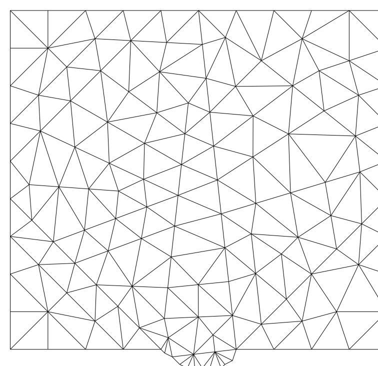

Abbildung 10.4: Parameter -7.0

Problematisch kann es auch dann werden, wenn die Geometrie aus langen Segmenten besteht und zusätzlich eine Verengung vorliegt. Solch ein Beispiel wird in den Abbildungen 10.5 und 10.6 vorgestellt.

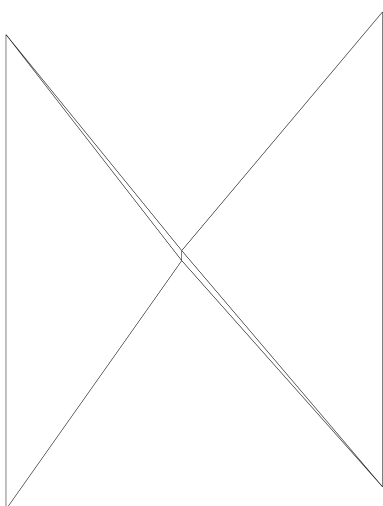

Abbildung 10.5: Parameter -1.0

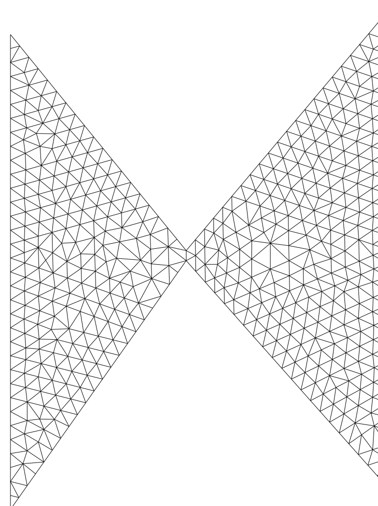

Abbildung 10.6: Parameter -0.05

10.1.2 Delaunay

Die Abbildungen 10.7 – 10.12 demonstrieren die Auswirkungen des minimalen Abstands zweier Punkte auf die Qualität der resultierenden Triangulierung. Der angegebene Parameter gibt den relativen Abstand an, den zwei Punkte innerhalb der Triangulierung einhalten müssen.

Bei den im Laufe der Implementierung untersuchten Geometrien haben sich die Parameter 0.7 und 0.8 als besonders geeignet erwiesen.

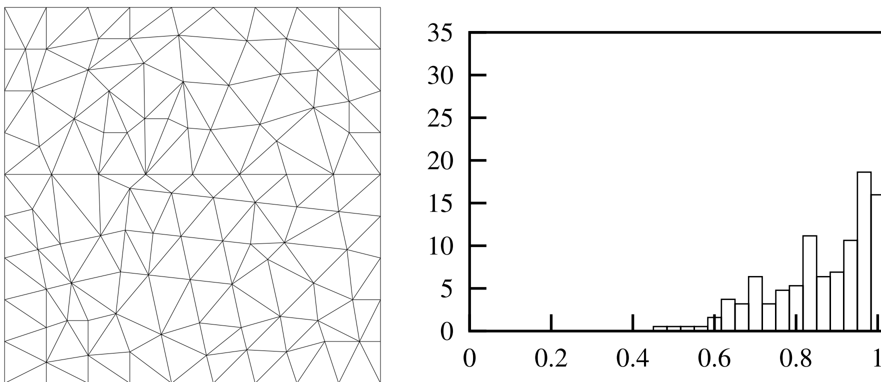

Abbildung 10.7: Parameter 0.5

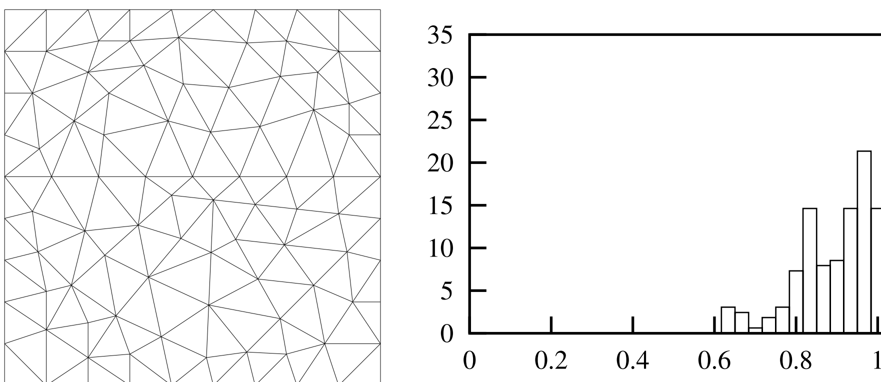

Abbildung 10.8: Parameter 0.6

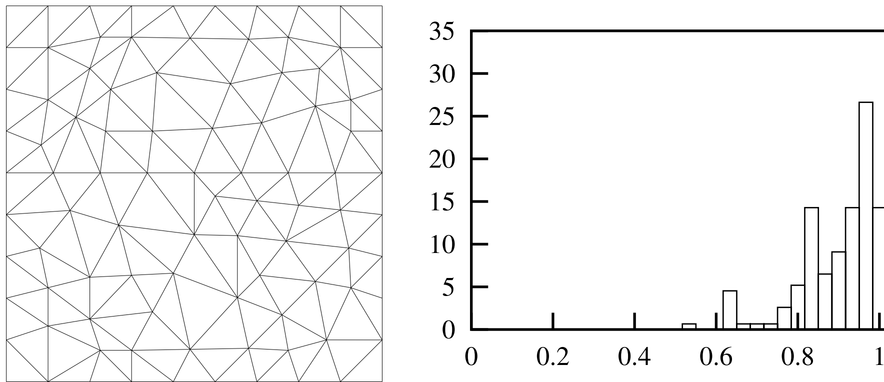

Abbildung 10.9: Parameter 0.7

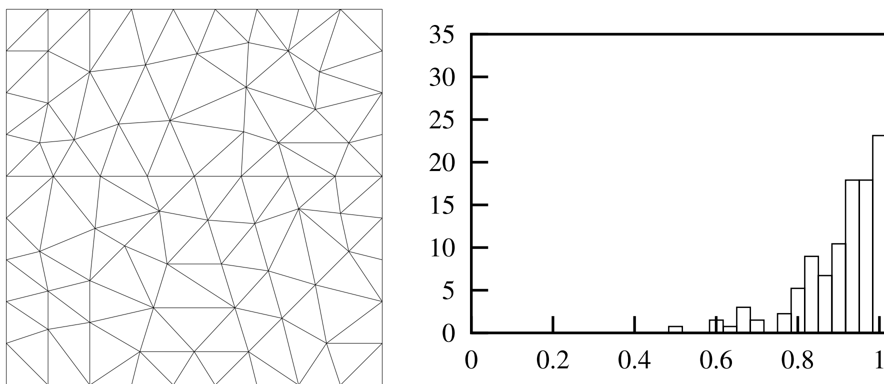

Abbildung 10.10: Parameter 0.8

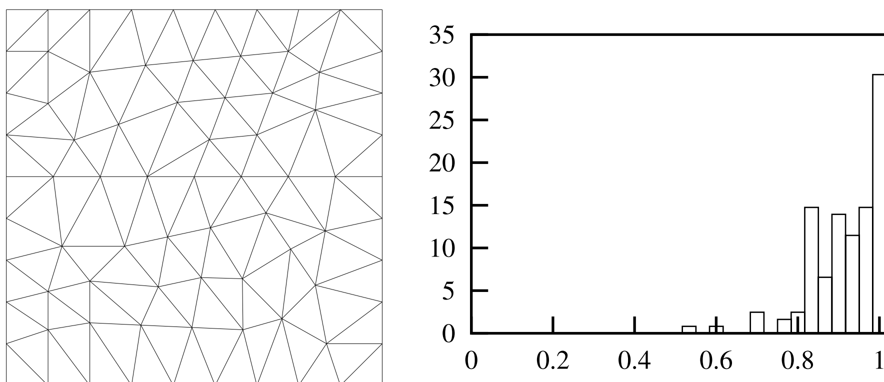

Abbildung 10.11: Parameter 0.9

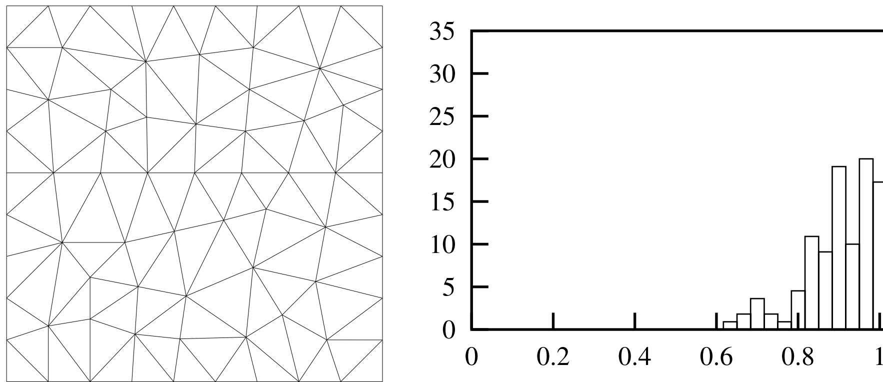

Abbildung 10.12: Parameter 1.0

10.1.3 Advancing-Front

Bei der Advancing-Front–Methode wird versucht, möglichst optimale Elemente an den Rand anzuhängen. Hierbei besteht jedoch das Problem des Zusammenwachsens der Front aus den unterschiedlichen Richtungen. Sucht man in einer vorgegebenen Umgebung um den optimalen zu erzeugenden Punkt nach bereits existierenden, so kann das Problem entschärft werden. Als Nebeneffekt wird erst durch diese Erweiterung eine Triangulierung von Gebieten mit Löchern möglich. In den folgenden Abbildungen (10.13 – 10.19) wird die Auswirkung der Größe dieser Umgebung auf die Qualität der Triangulierung gezeigt.

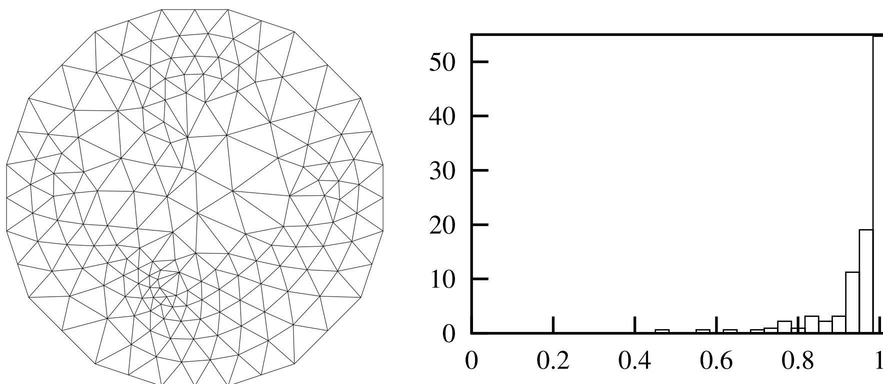

Abbildung 10.13: Parameter 0.3

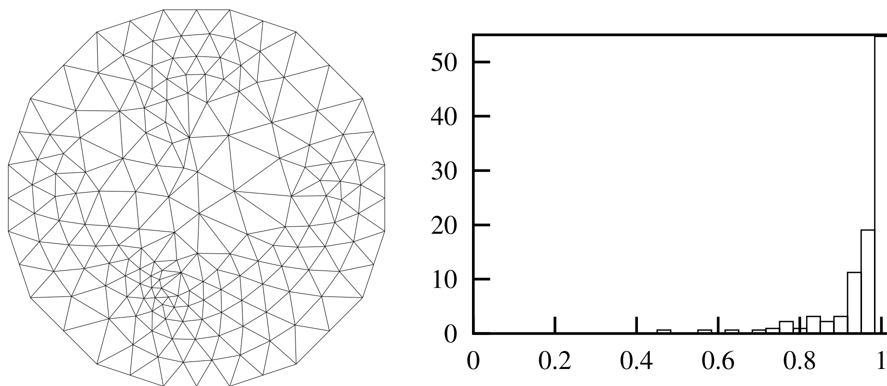

Abbildung 10.14: Parameter 0.4

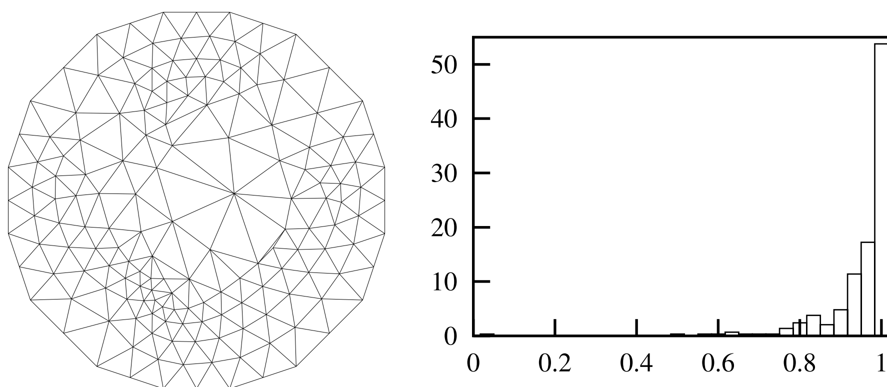

Abbildung 10.15: Parameter 0.5

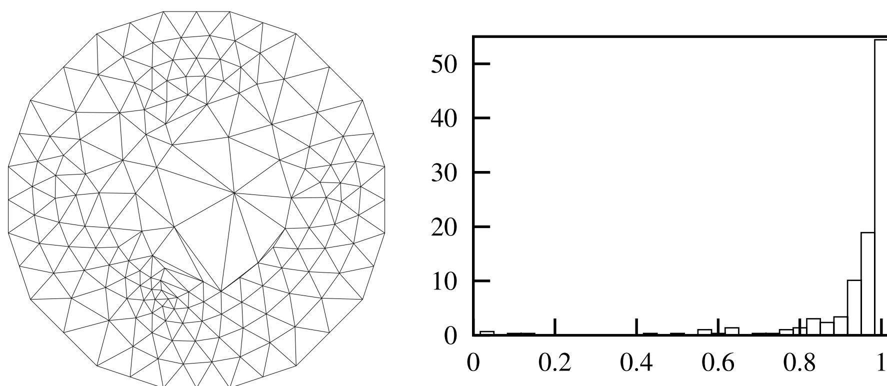

Abbildung 10.16: Parameter 0.6

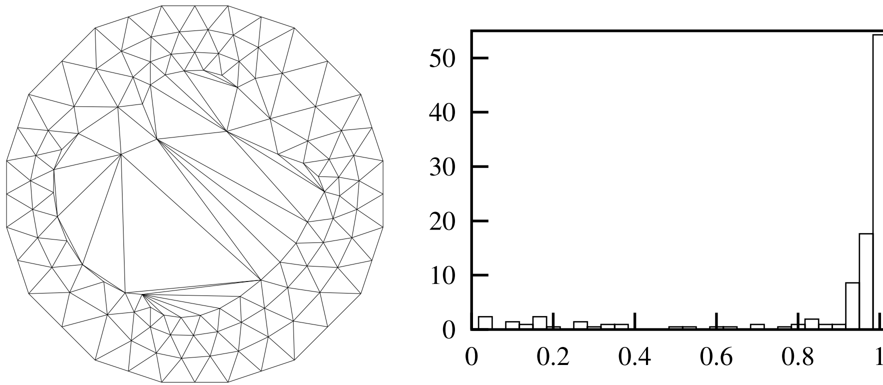

Abbildung 10.17: Parameter 0.7

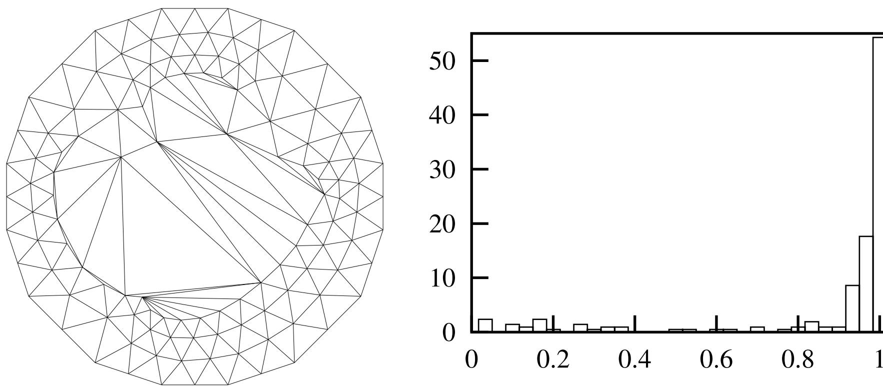

Abbildung 10.18: Parameter 0.8

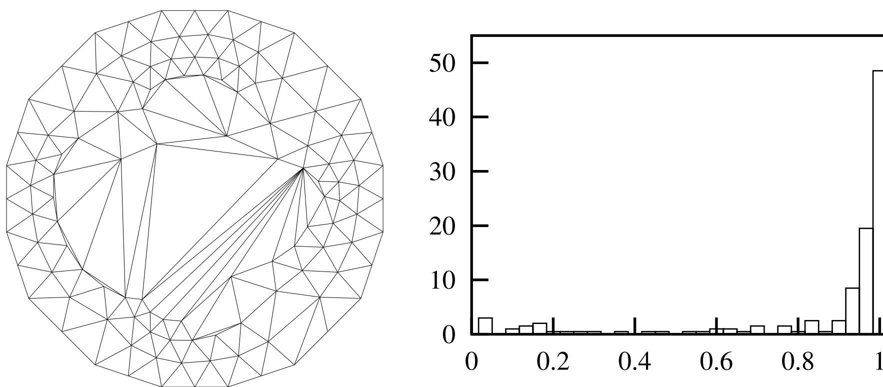

Abbildung 10.19: Parameter 0.9

Für die meisten Anwendungen hat sich der Parameter 0.6 bewährt.

10.2 Unterschiede zwischen den Generatoren

Die hier gezeigten Triangulierungen wurden alle mit den Standardparametern des jeweiligen 2D-Generators erzeugt. Um die Unterschiede besser verdeutlichen zu können, wurde die 1D-Verfeinerung oft feiner als unbedingt nötig gewählt.
Zur Erläuterung der Generatoren-Bezeichnung:

- delaunay
  Delaunay-Generator mit Punkte-Einfügung entlang der vorhandenen Kanten.
- afm1
  Advancing-Front-Generator, Auswahlkriterium: Minimiere den Abstand zum ursprünglichen Rand, dann die Länge des Segments.
- afm2
  Advancing-Front-Generator, Auswahlkriterium: Minimiere den Abstand zum ursprünglichen Rand, suche aus den verbleibenden Segmenten das mit dem geringsten Winkel zu einem seiner Nachbarsegmente.
- afm3
  Advancing-Front-Generator, Auswahlkriterium: Minimiere den Abstand zum ursprünglichen Rand, suche aus den verbleibenden Segmenten das mit dem geringsten Produkt aus seiner Länge mit dem kleinsten Winkel zu seinen Nachbarsegmenten.
- afm4
  Advancing-Front-Generator, Auswahlkriterium: Suche den kleinsten Winkel zwischen zwei Front-Segmenten, wähle eines dieser beiden Segmente.

Nachfolgend werden einige Triangulierungsergebnisse der verschiedenen Generatoren einschließlich einer Bewertung dargestellt.

10.2.1 square2x2.net

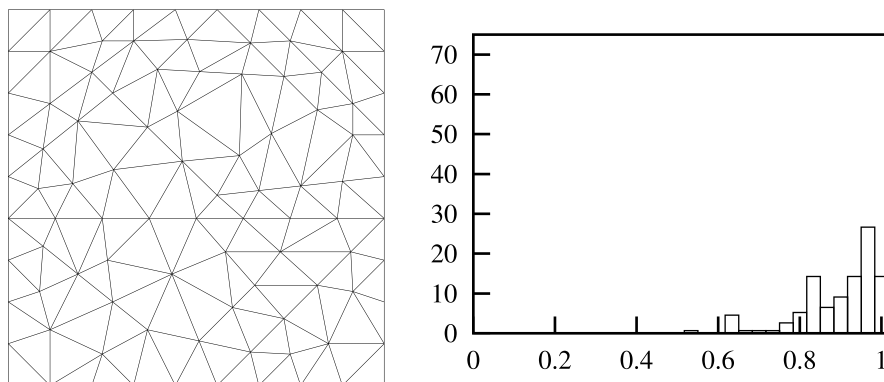

Abbildung 10.20: delaunay

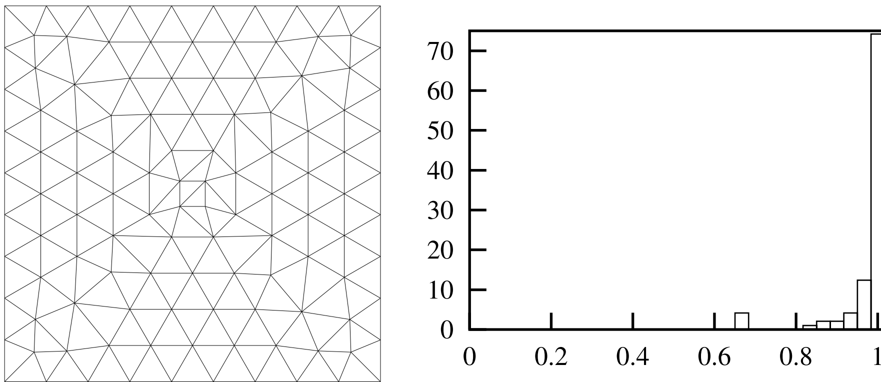

Abbildung 10.21: afm1

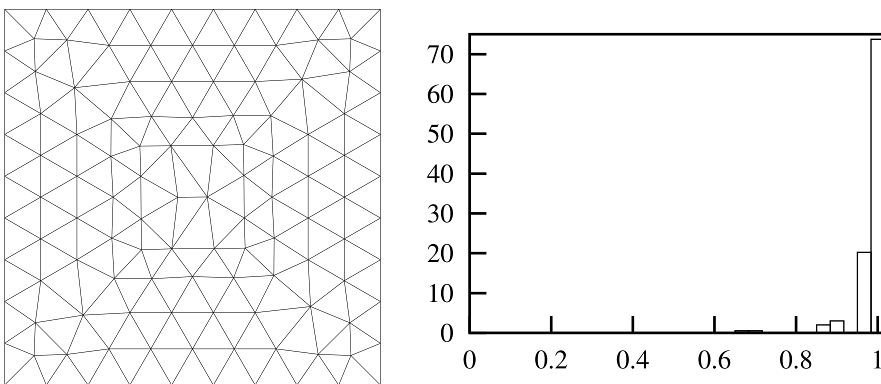

Abbildung 10.22: afm2

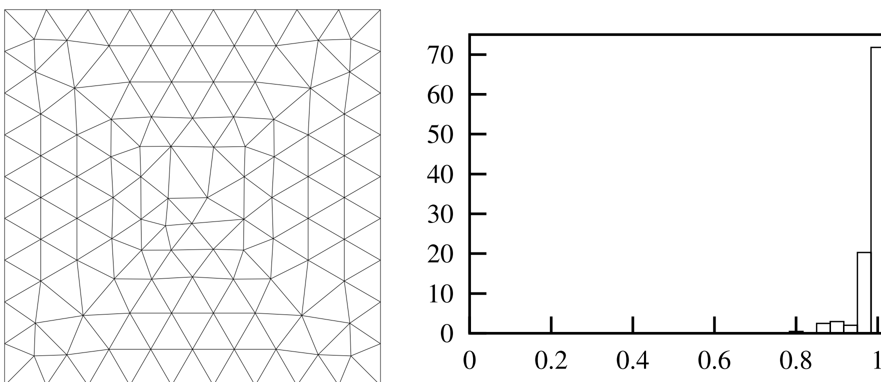

Abbildung 10.23: afm3

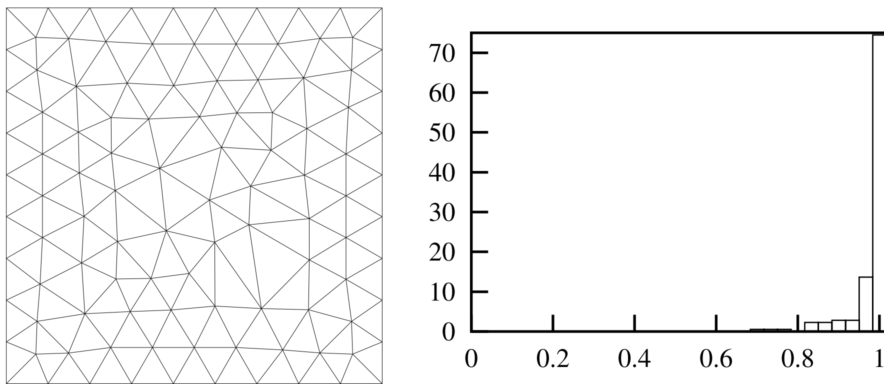

Abbildung 10.24: afm4

10.2.2 hai.net

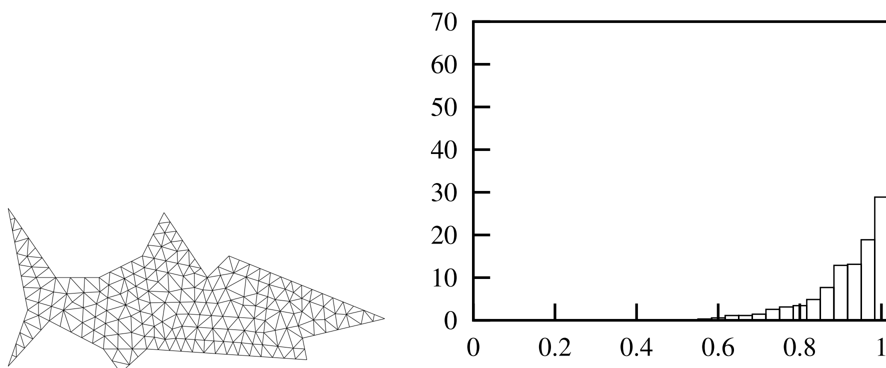

Abbildung 10.25: delaunay

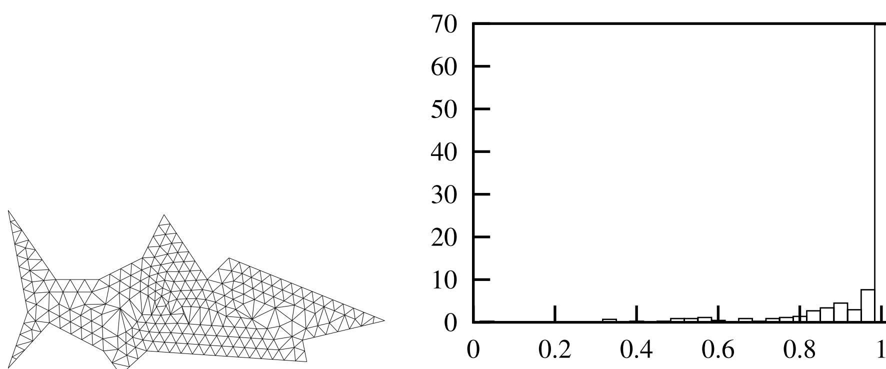

Abbildung 10.26: afm1

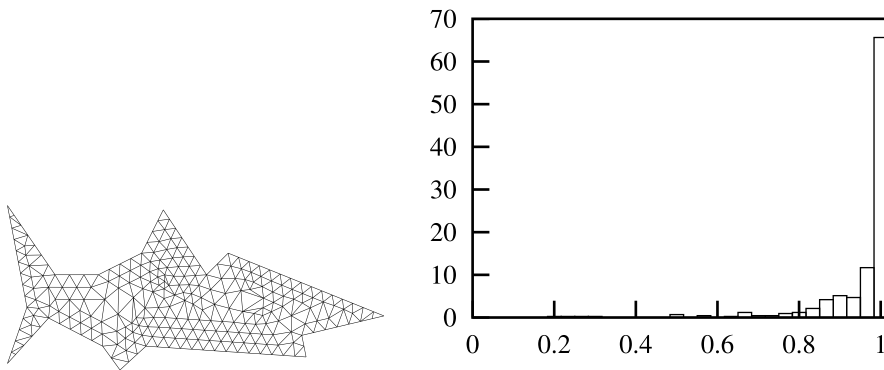

Abbildung 10.27: afm2

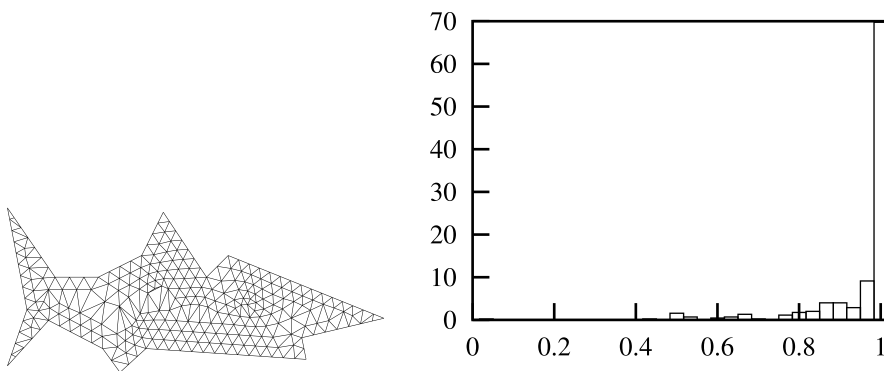

Abbildung 10.28: afm3

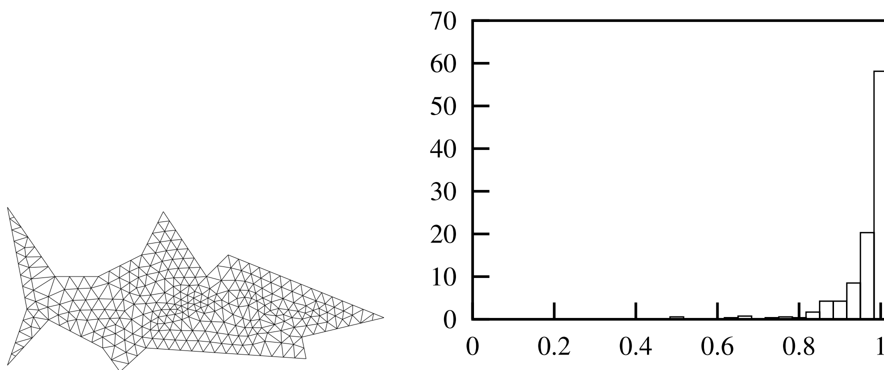

Abbildung 10.29: afm4

10.2.3 magd4.net

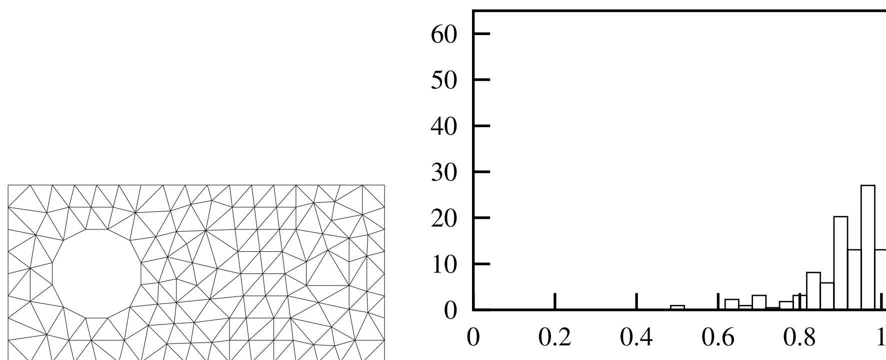

Abbildung 10.30: delaunay

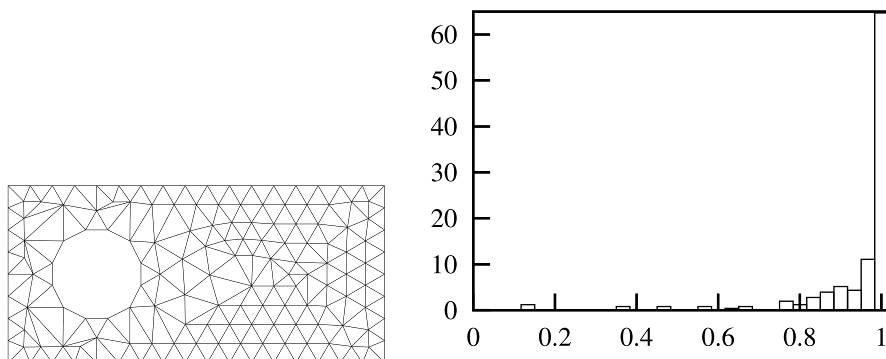

Abbildung 10.31: afm1

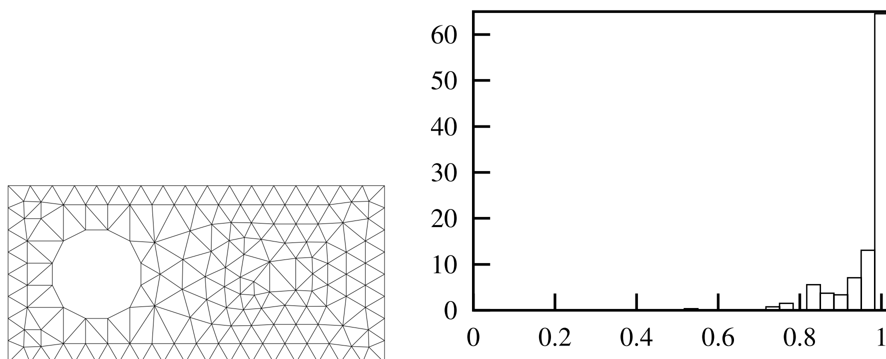

Abbildung 10.32: afm2

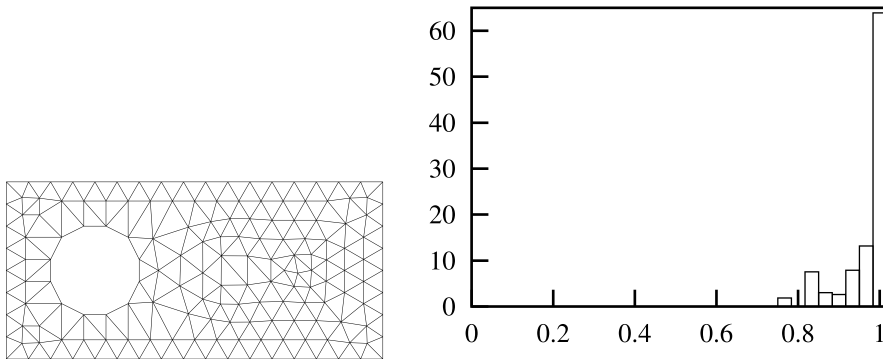

Abbildung 10.33: afm3

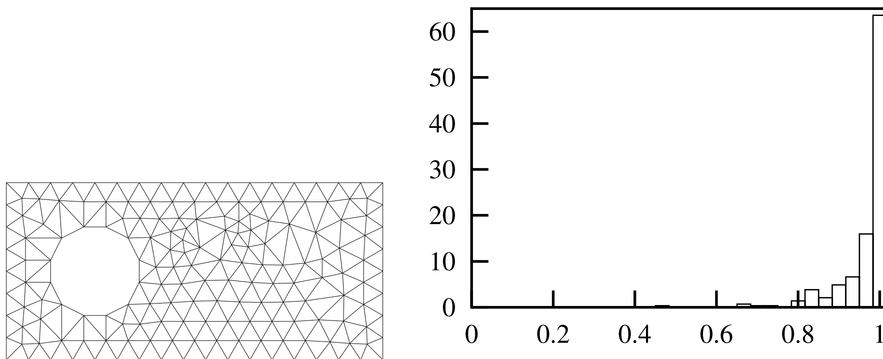

Abbildung 10.34: afm4

10.2.4 kompl2.net

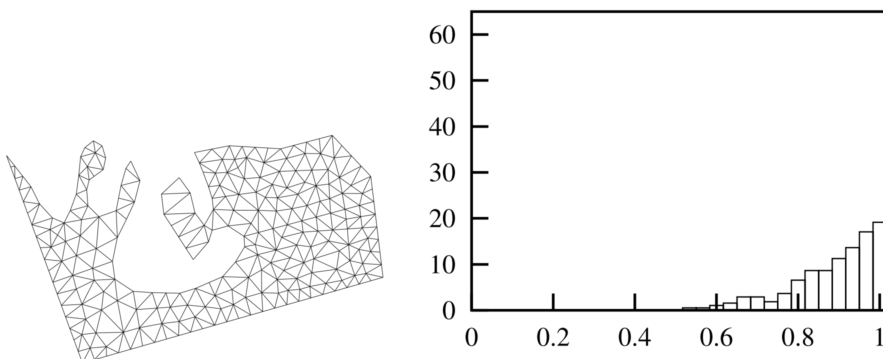

Abbildung 10.35: delaunay

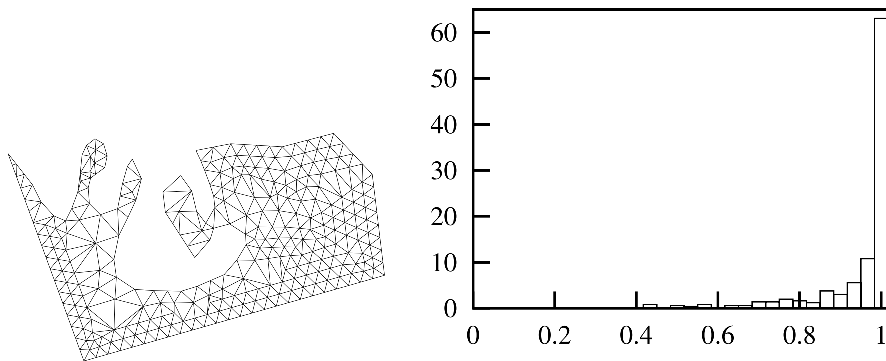

Abbildung 10.36: afm1

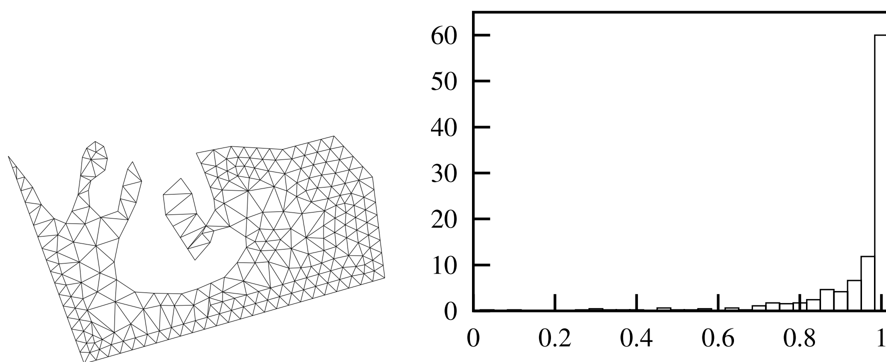

Abbildung 10.37: afm2

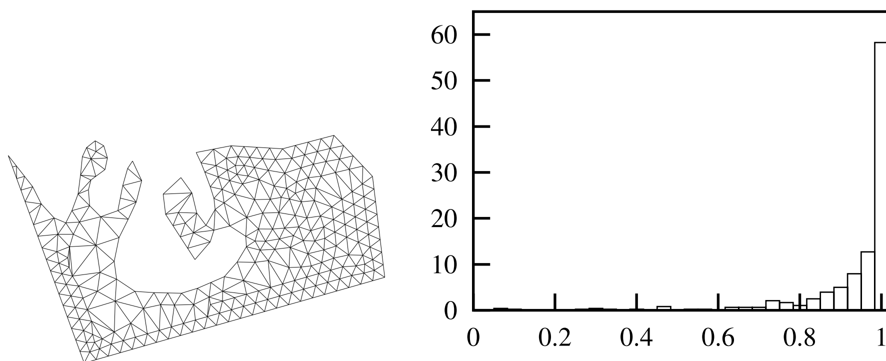

Abbildung 10.38: afm3

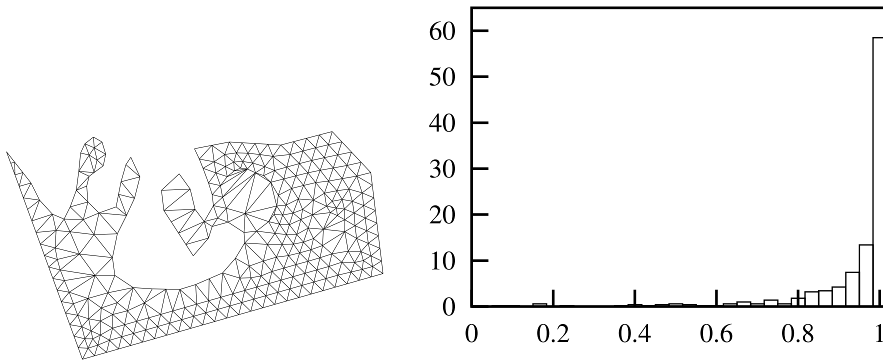

Abbildung 10.39: afm4

10.2.5 schluessel3a.net

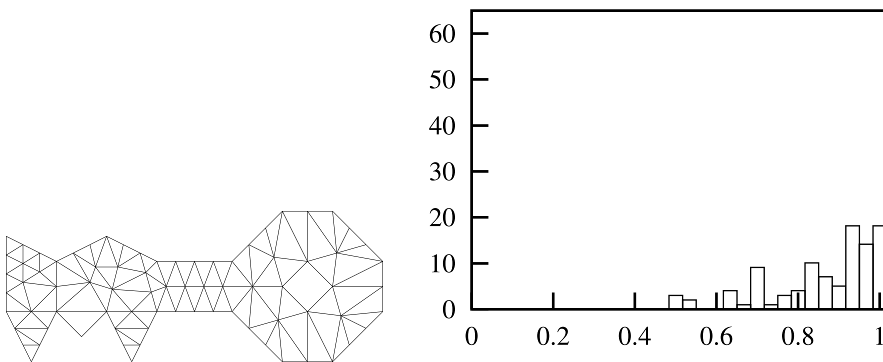

Abbildung 10.40: delaunay

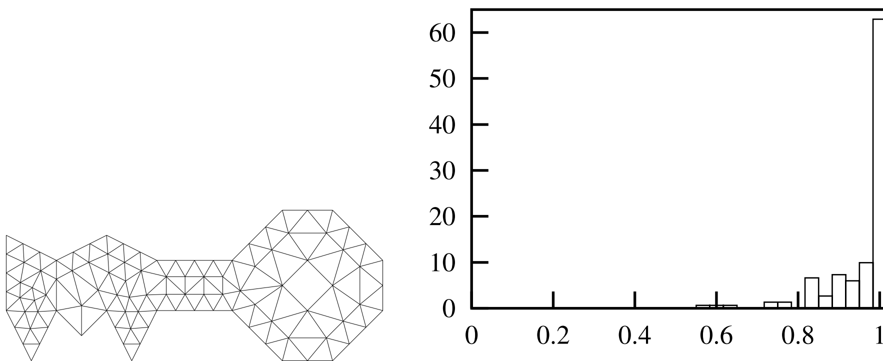

Abbildung 10.41: afm1

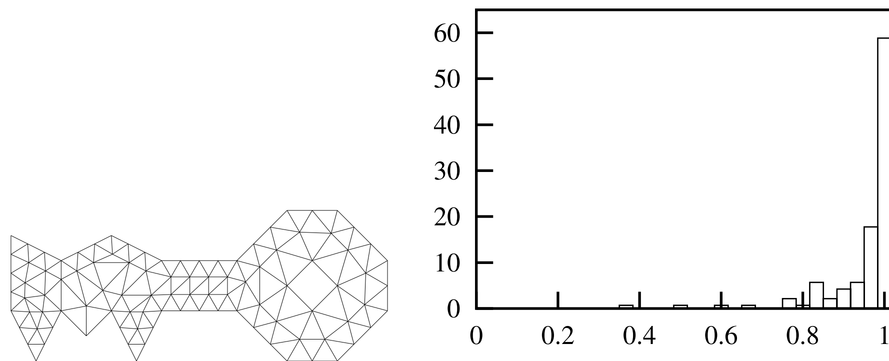

Abbildung 10.42: afm2

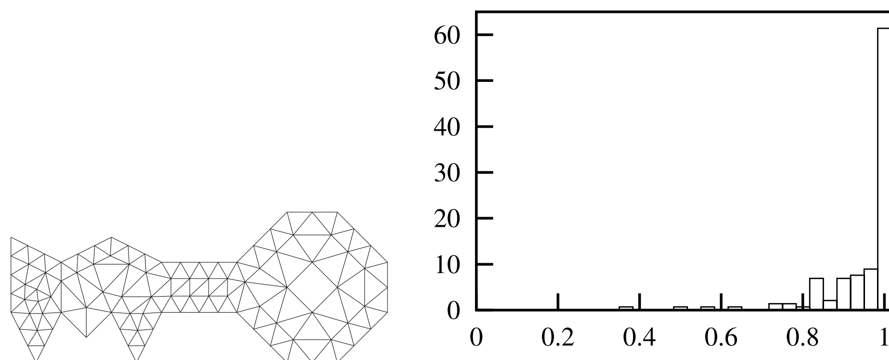

Abbildung 10.43: afm3

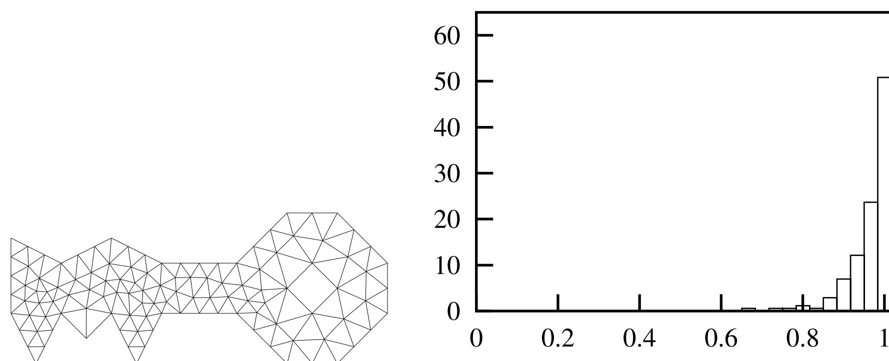

Abbildung 10.44: afm4

10.2.6 Bewertung

In der vorliegenden Implementierung liegt kein Generator vor, der alle Gebiete optimal behandelt. Die Advancing-Front-Generatoren bilden in Randnähe zwar zu-

meist optimale Simplizes, tendieren allerdings zu lokalen Schwächen, die durch das Zusammenwachsen der Front hervorgerufen werden und sich in extrem schlechten Elementen äußern können.

Der Delaunay-Generator ist in der Spitzenqualität den AFM-Generatoren klar unterlegen, durch die gewählte Strategie zum Einfügen der inneren Punkte werden jedoch Elemente erzeugt, die von der Fläche her relativ nahe beieinander liegen.

10.3 Auswirkungen der Glätter

Die bisher demonstrierten Triangulierungsgeneratoren kümmern sich in erster Linie um die Fertigstellung einer Triangulierung innerhalb der gegebenen Geometrie. Auch wenn dabei schon einige Strategien zur Erreichung einer guten Qualität eingebaut sind, können die meisten Triangulierungen durch sogenannte Glättungsverfahren drastisch verbessert werden.

Es wurde abwechselnd der Edge-Swap- und der Laplace-Glätter angewendet. Das Verfahren wurde so lange wiederholt, bis sich keine Änderungen mehr zeigten.

Die nachfolgend gezeigten Abbildungen 10.45–10.69 sind das Glättungsergebnis der Abbildungen 10.20–10.44.

10.3.1 square2x2.net

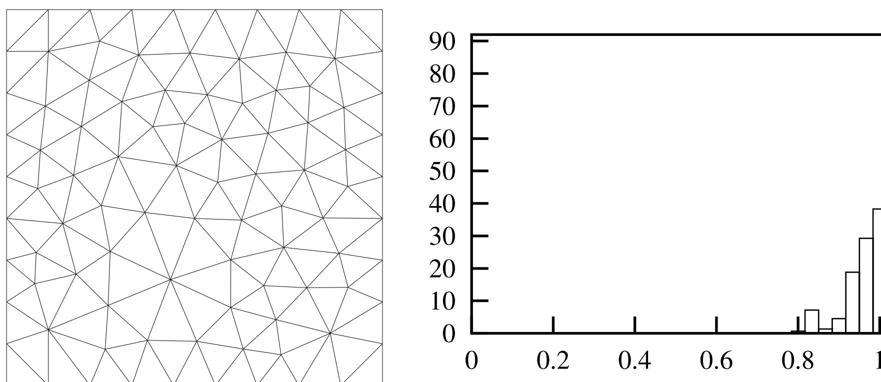

Abbildung 10.45: delaunay

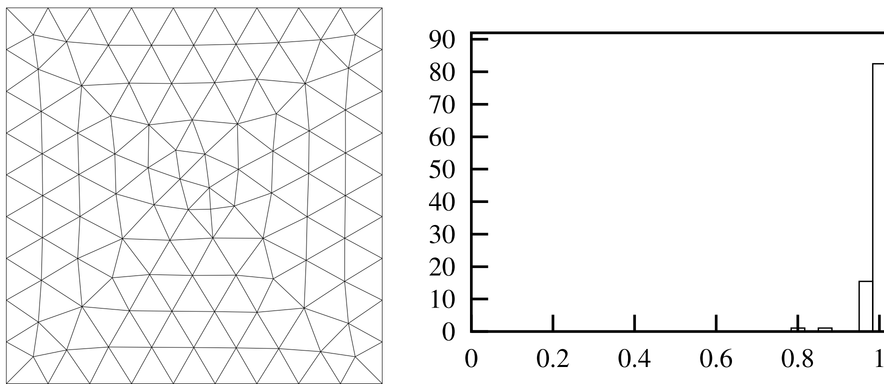

Abbildung 10.46: afm1

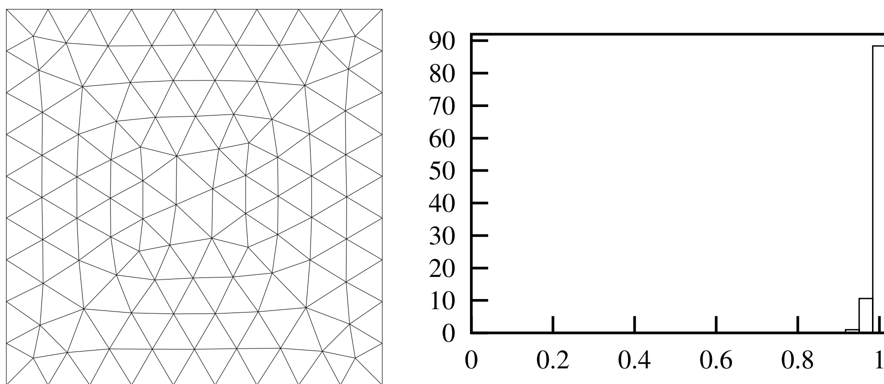

Abbildung 10.47: afm2

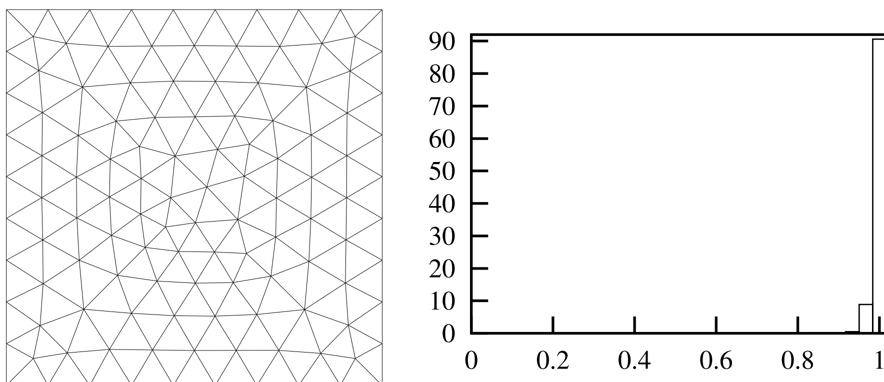

Abbildung 10.48: afm3

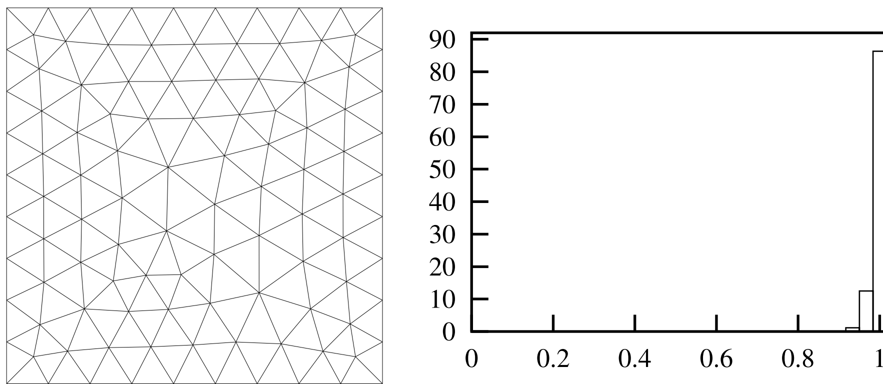

Abbildung 10.49: afm4

10.3.2 hai.net

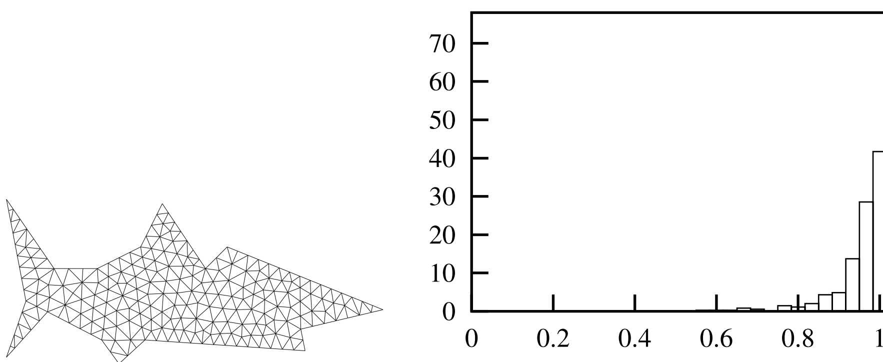

Abbildung 10.50: delaunay

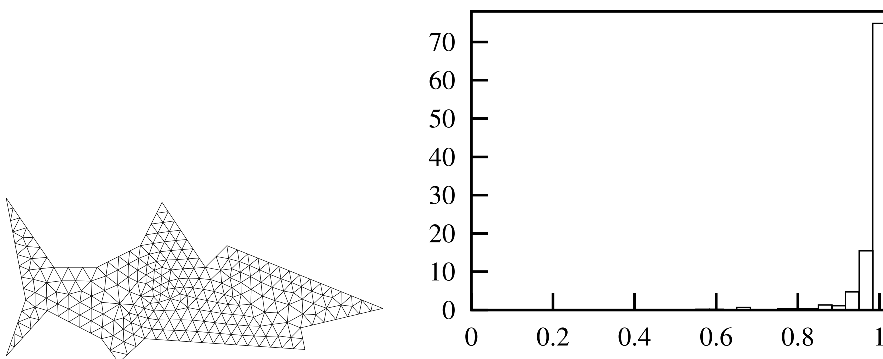

Abbildung 10.51: afm1

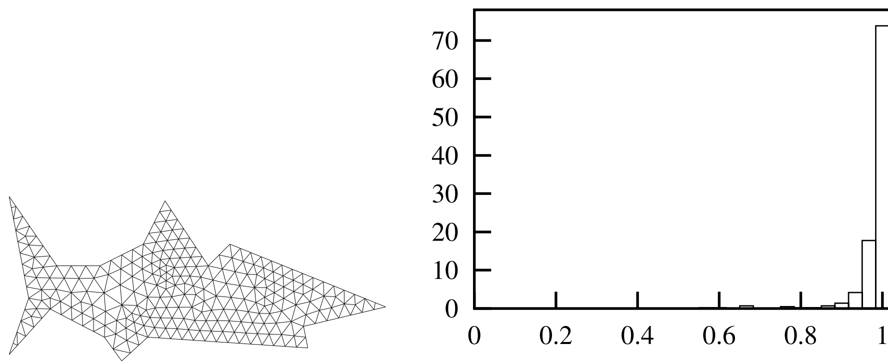

Abbildung 10.52: afm2

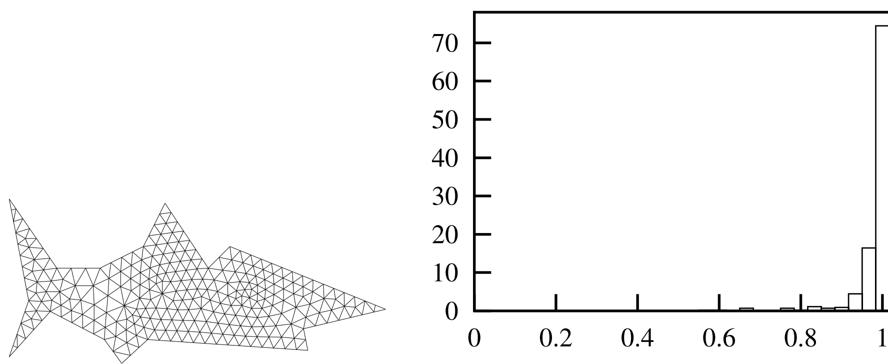

Abbildung 10.53: afm3

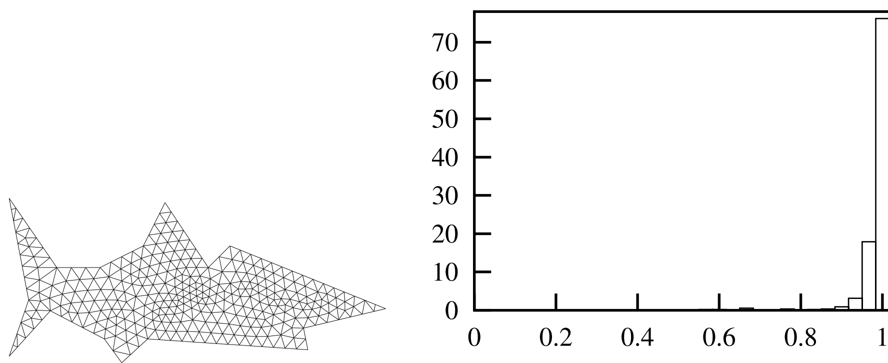

Abbildung 10.54: afm4

10.3.3 magd4.net

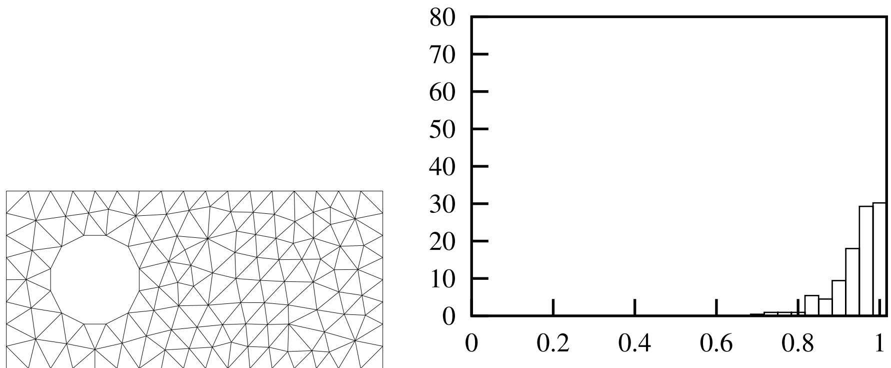

Abbildung 10.55: delaunay

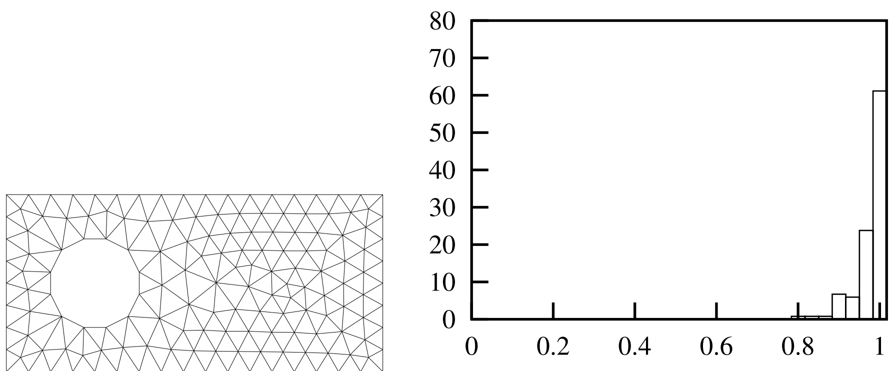

Abbildung 10.56: afm1

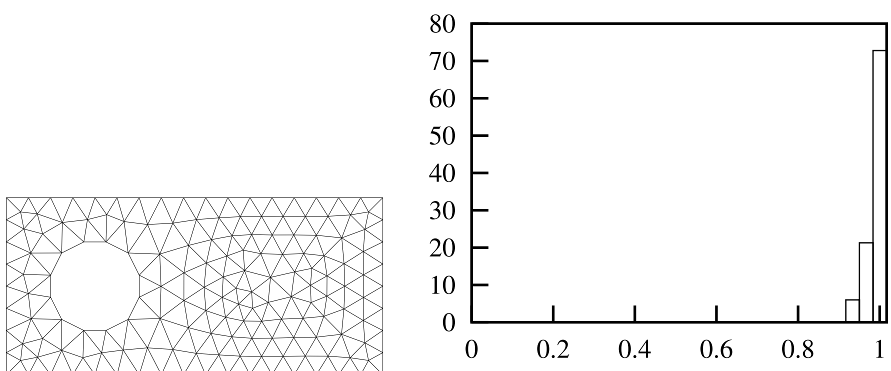

Abbildung 10.57: afm2

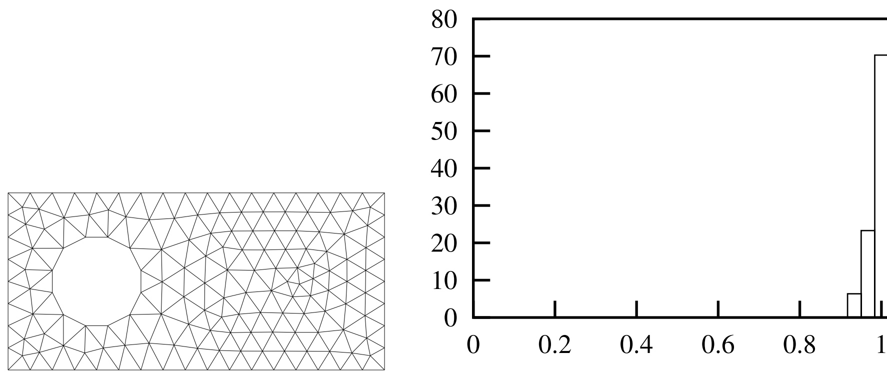

Abbildung 10.58: afm3

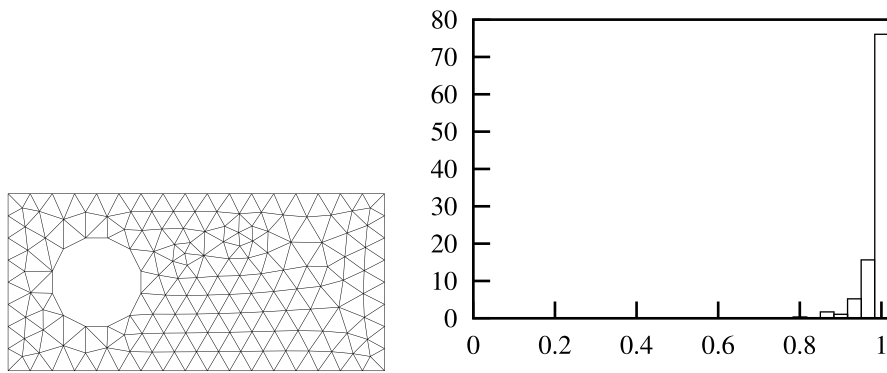

Abbildung 10.59: afm4

10.3.4 kompl2.net

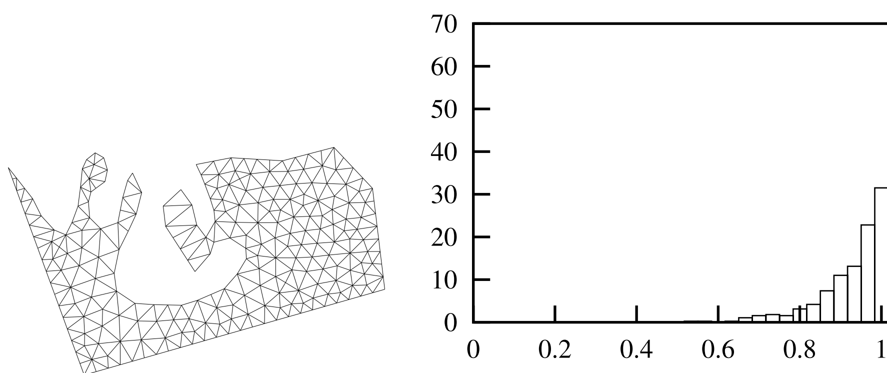

Abbildung 10.60: delaunay

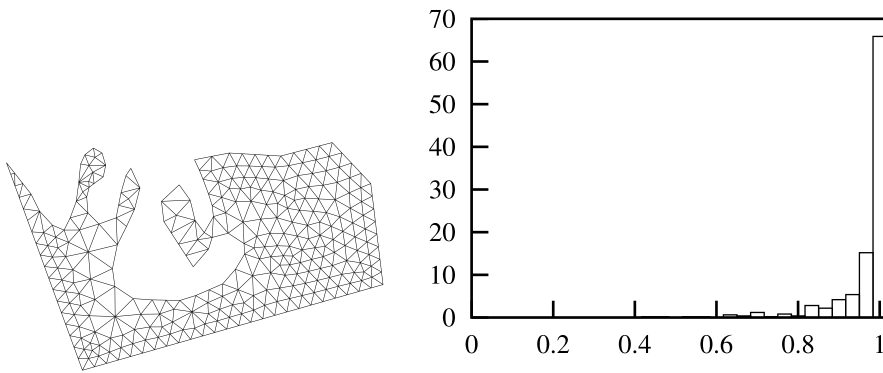

Abbildung 10.61: afm1

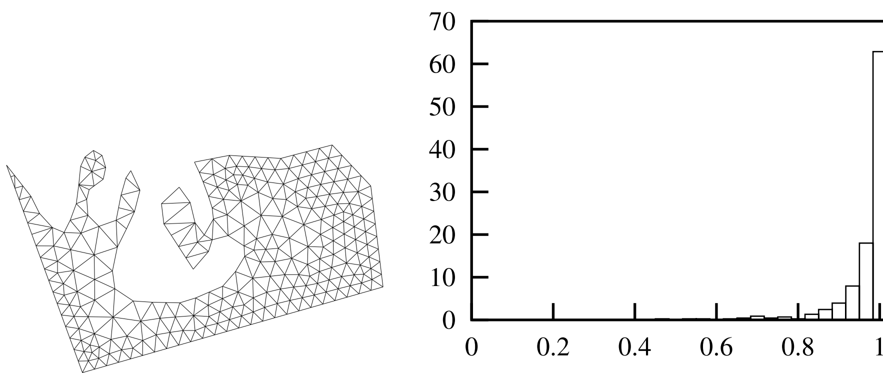

Abbildung 10.62: afm2

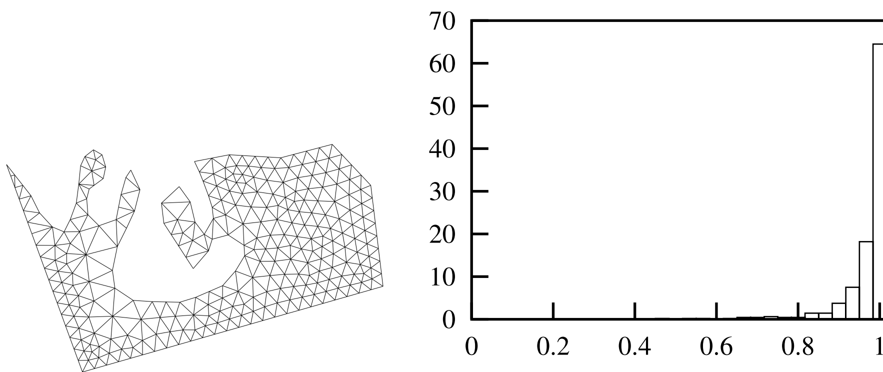

Abbildung 10.63: afm3

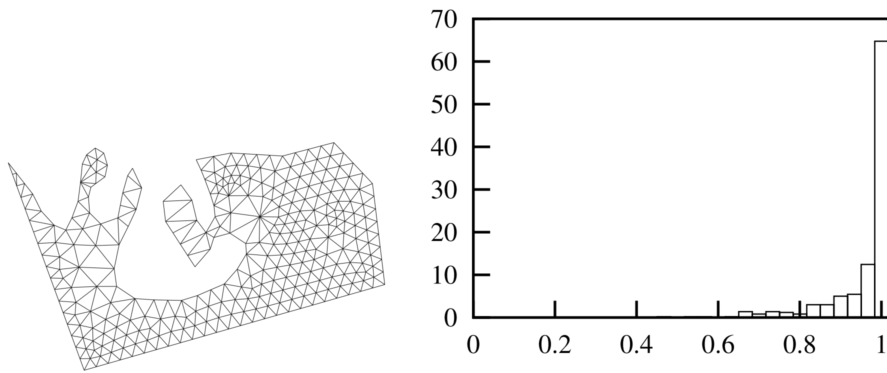

Abbildung 10.64: afm4

10.3.5 schluessel3a.net

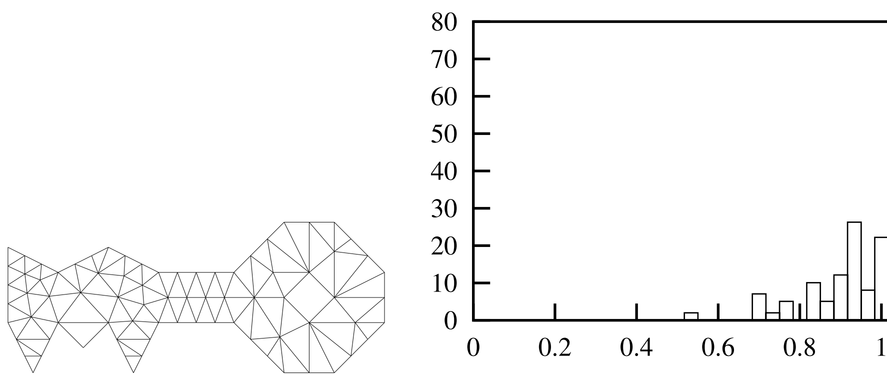

Abbildung 10.65: delaunay

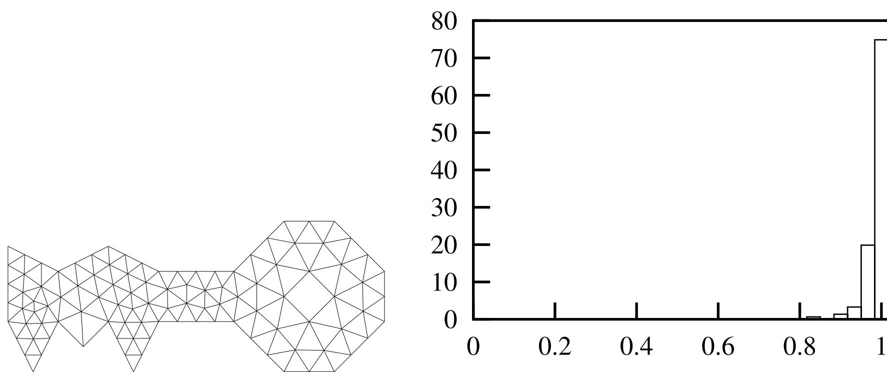

Abbildung 10.66: afm1

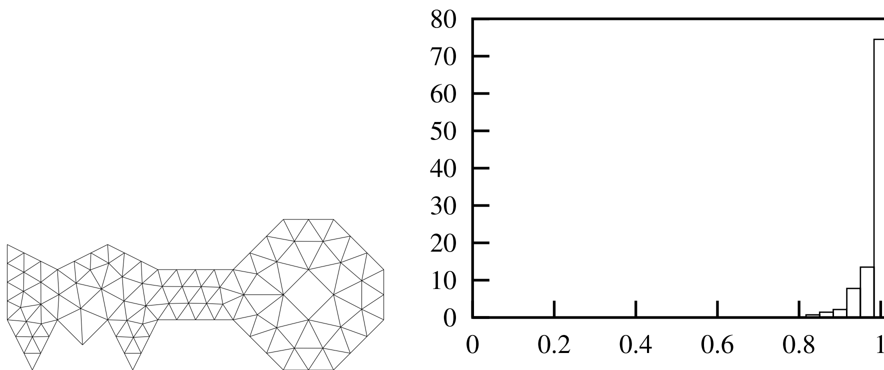

Abbildung 10.67: afm2

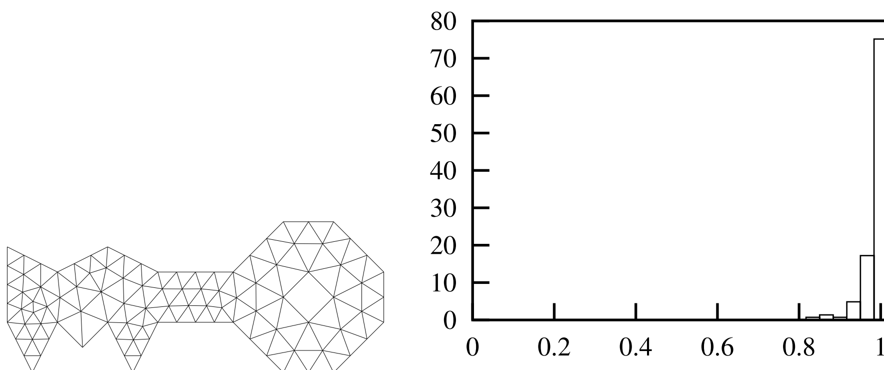

Abbildung 10.68: afm3

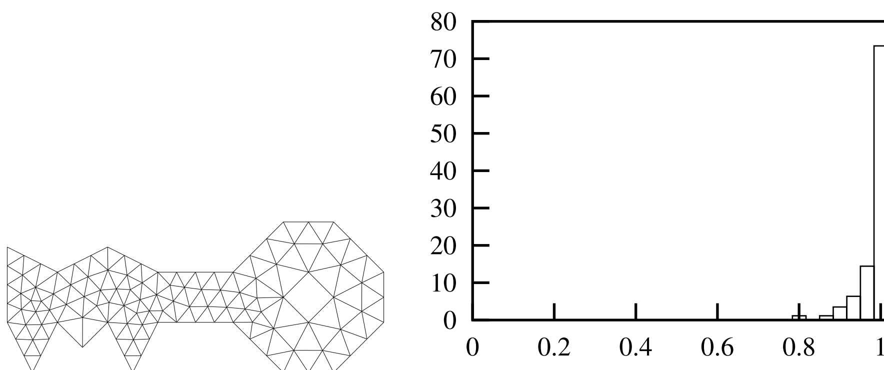

Abbildung 10.69: afm4

10.3.6 Bewertung

Die Qualität der erhaltenen Triangulierungen ist durch Einsatz der Glättungsverfahren stark verbessert. Der Rechenaufwand war in allen hier gezeigten Fällen klein

im Vergleich zur Generierung der Triangulierungen.

10.4 Die Verfeinerung

Im folgenden werden Triangulierungen mit zunehmender lokaler Verfeinerung am Beispiel der Lösungsfunktion $f = 8\pi^2 \sin(2\pi x_1)\sin(2\pi x_2)$ dargestellt. Für jedes Level werden sowohl die Triangulierung als auch die Höhenlinien der ermittelten Lösung angegeben. Es wurde ein einfacher Fehlerschätzer, implementiert von E. Stafilarakis, eingesetzt.

In der ersten Bildfolge wurde eine Delaunay-Ausgangstriangulierung verwendet. Der 1D-Parameter wurde auf 0.25 gesetzt.

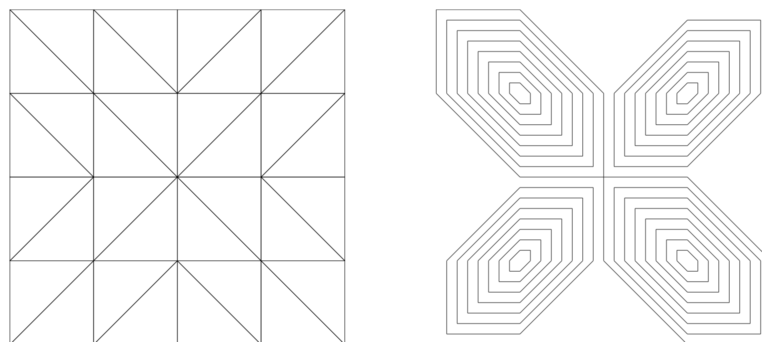

Abbildung 10.70: Level 1

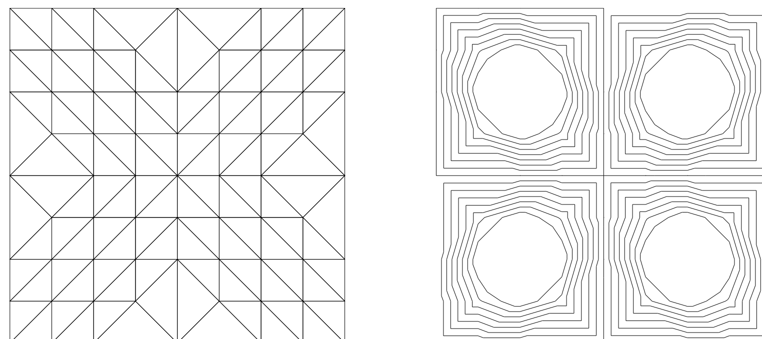

Abbildung 10.71: Level 2

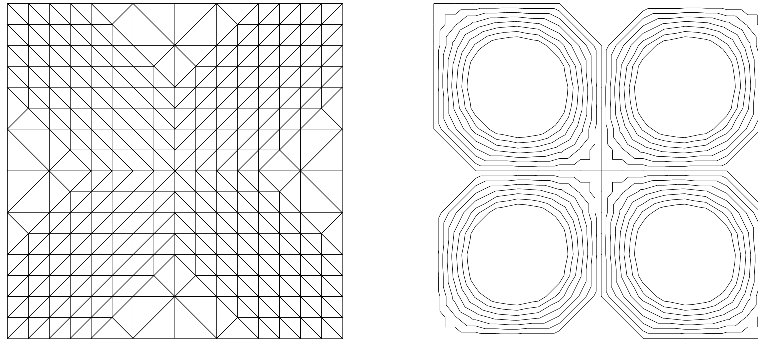

Abbildung 10.72: Level 3

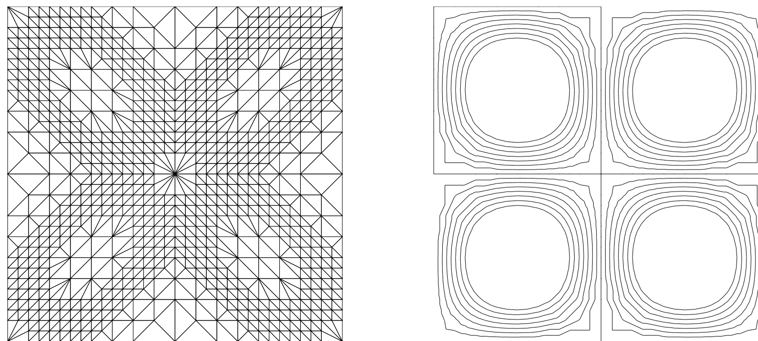

Abbildung 10.73: Level 4

Für die zweite Bildfolge wurde der 1D-Parameter auf 0.1 reduziert. Dadurch ergab sich eine unstrukturierte Ausgangstriangulierung, die anschließend mit Edge-Swapping und Laplace geglättet wurde. Die resultierende Triangulierung wurde einmal global verfeinert. Da dabei keine Lösung berechnet wurde, wird dieses Level hier nicht dargestellt.

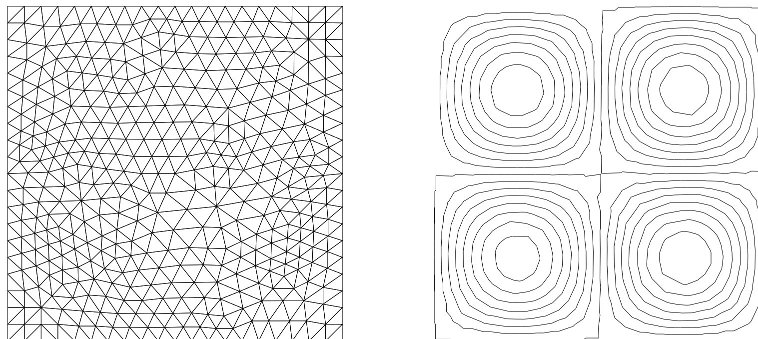

Abbildung 10.74: Level 2

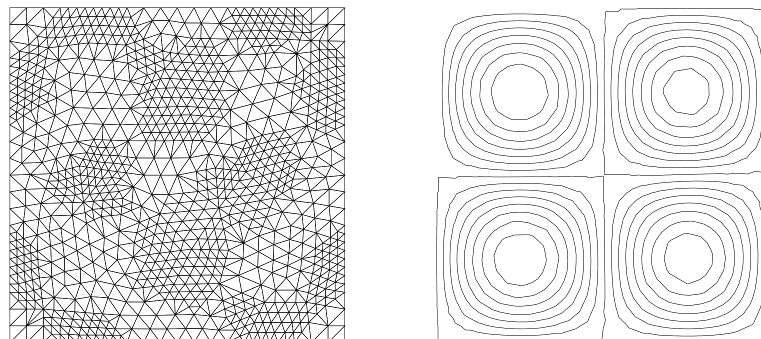

Abbildung 10.75: Level 3

Bei beiden Ausgangstriangulierungen zeichnet sich eine deutliche Korrelation zwischen der Triangulierung und dem Verlauf der Lösung ab.

10.5 Fazit

Bei den im Rahmen dieser Arbeit untersuchten Gebieten fanden sich stets Parameter, für die eine Ausgangstriangulierung gebildet werden konnte. Jedoch kann keine Aussage getroffen werden, welcher Generator der beste ist. Delaunay ist zu empfehlen, wenn es auf Elemente mit möglichst geringen Unterschieden in bezug auf die Volumina ankommt. Im Vergleich zu AFM benötigt er allerdings etwa 30% mehr Rechenzeit. Die AFM-Generatoren haben trotz des Schnappens manchmal Probleme beim Zusammenwachsen der Fronten, erzeugen dabei aber sehr gleichförmige Simplizes.
Beide Glättungsverfahren haben ihre Berechtigung und spielen ihre Stärken vor allem in der Zusammenarbeit aus. In der vorliegenden Arbeit wurden sie immer abwechselnd nacheinander eingesetzt. Dieses Verfahren führt zu sehr guten Ergebnissen.
Die Verfeinerung arbeitet sehr zuverlässig. Ein wirkungsvoller Einsatz setzt aber einen größeren Rahmen in Form eines Finite-Elemente-Paketes voraus, der im derzeitigen Stadium der Software-Entwicklung unserer Arbeitsgruppe erst in Ansätzen vorhanden ist.

Anhang A

Programmierhandbuch

Dieses Manual gibt einen Einblick in die Programmierung mit **meshMan**. **meshMan** besteht aus den in der Einleitung aufgeführten Bibliotheken zur Erzeugung hierarchisch verfeinerter Triangulierungen für Finite-Elemente-Probleme. Aus der Randbeschreibung läßt sich automatisch eine Triangulierung generieren, die durch verschiedene Glättungsverfahren qualitativ verbessert werden kann. Als Verfeinerungsstrategie wird die Rot-Grün-Strategie eingesetzt, die beliebig tiefe stabile Verfeinerungen erlaubt. Viele Routinen und Klassen von **meshMan** sind auch für andere Aufgaben einsetzbar; deshalb wurde das Paket in mehrere Bibliotheken unterteilt. Dadurch ergibt sich ein Einsatzbereich, der weit über die in dieser Arbeit angesprochenen Themen hinausgeht.
In den Codebeispielen dieses Manuals werden Zeilen mit bisher unbekannten Befehlen und Aufrufen über Indizes am rechten Rand markiert; diese Indizes beginnen für jedes Programm mit 1. Zu lange Zeilen — beispielsweise `for`-Schleifen — werden bei Bedarf umgebrochen; ihr Index steht dann in der letzten Zeile, ist aber für die gesamte Anweisung gültig.
Benötigte Headerdateien werden in den Codefragmenten immer angegeben; zuerst diejenigen, die für die neuen Anweisungen wichtig sind, dann nach einer Leerzeile die bereits bekannten. Es existieren zusätzlich Sammel-Includedateien (*`bibliothekname`*`all.h`) die sämtliche Header einer Bibliothek einbinden.
Durch die Aufteilung müssen beim Linken des Programms die Abhängigkeiten der Bibliotheken untereinander — wie sie in Anhang A.7 dargestellt sind — beachtet werden.
Die Diskette zu dieser Arbeit enthält zusätzlich eine komplette Referenzdokumentation.

A.1 Die Klasse `indexSetC`

Die Templateklasse `indexSetC` wurde als ungeordnete indizierte Menge für **meshMan** entwickelt. Es handelt sich dabei um einen erweiterten Vektor mit Routinen zum schnellen Löschen und Einfügen von Elementen. Durch die Erweiterung können

keine Aussagen über die Ordnung der Elemente gemacht werden; solch eine Ordnung ist für **meshMan** jedoch nicht von Bedeutung.

```
#include <indexset.hh>

// ...

indexSetC<base::indexT> set;

set.reserve(5);                                                  (1)
for (base::indexT i = 0; i < 5; ++i)
     set.append(i);                                              (2)

for (indexSetC<base::indexT>::iteratorConst it = set.beginConst();
      it != set.endConst();
      ++it)                                                      (3)
     cout << *it << ' ';                                         (4)
cout << endl;

set.unset(3);                                                    (5)
set.unset(set.begin());                                          (6)
for (indexSetC<base::indexT>::iteratorConst it = set.beginConst();
      it != set.endConst();
      ++it)
     cout << *it << ' ';
cout << endl;

set.append(10);                                                  (7)
for (indexSetC<base::indexT>::iteratorConst it = set.beginConst();
      it != set.endConst();
      ++it)
     cout << *it << ' ';

// ...
```

Der Aufruf (1) reserviert so viel Speicher für die Menge, daß mindestens 5 freie Elemente verfügbar sind. So ist garantiert, daß für die in (2) eingefügten Elemente die Größe der Menge nicht geändert werden muß. Der Zugriff auf die Elemente kann über Iteratoren (3) oder direkt über die Indizes der Elemente erfolgen. (4) dereferenziert den Iterator und gibt das entsprechende Element aus.
Elemente können entweder durch Angabe ihres Indexes (5) oder mittels eines Iterators (6) aus der Menge entfernt werden. Die Positionen der gelöschten Objekte

werden so gespeichert, daß ein neu einzufügendes Objekt auf den ersten freien Platz gesetzt wird (7).
Die Ausgabe des obigen Programms lautet also:

```
0 1 2 3 4
1 2 4
10 1 2 4
```

A.2 Die Geometrie-Bibliothek

Die Geometrie-Bibliothek stellt Funktionen für einfache Geometriefunktionen um eine allgemeine Punkt-Klasse zur Verfügung. Einige der Funktionen arbeiten derzeit lediglich im 2-dimensionalen Fall. Höhere Dimensionen waren für den vorliegenden Code nicht nötig, hätten für die Implementierung jedoch einen hohen Aufwand benötigt.

A.2.1 Punkte

Ein n-dimensionaler Punkt wird durch n Koordinaten repräsentiert; er kann also auch als Vektor aufgefaßt werden. Auf die Koordinaten kann einzeln über den Operator `[]` zugegriffen werden.
Für die Interpretation als Ortsvektor enthält die Klasse mehrere arithmetische Verknüpfungen und einen Vergleichsoperator.

```
#include <point.h>

using namespace geom;

// ...

pointC v1(2), v2(2), v3(2);                                (1)

v1[0] = 1.0;                                               (2)
v1[1] = 1.0;

v2[0] = 5.0;
v2[1] = 5.0;

v3 = v1 + v2;                                              (3)
cout << v3 << endl;                                        (4)

v3 = v3 * 2;                                               (5)
```

```
cout << v3 << endl;

cout << "v1 und v2 sind "
     << v1 == v2 ? "gleich" : "verschieden" << endl;            (6)

// ...
```

(1) erzeugt drei Punkte/Vektoren in zwei Dimensionen. Die Koordinaten von Punkten werden in (2) gesetzt. Vektoren können nach (3) und (5) mittels Operatoren verknüpft werden. Der Vergleichsoperator (6) arbeitet mit der in `base.h` definierten Konstanten `::base::eps` als Toleranz. Operator (4) gibt Punkte/Vektoren aus.
Die Ausgabe des obigen Programms lautet (ohne genaue Formulierung des Ausgabeoperators):

```
Dimension 2, 6.0 6.0
Dimension 2, 12.0 12.0
v1 und v2 sind verschieden
```

A.2.2 Zusätzliche Routinen für Punkte

Die bisher vorgestellten Operationen werden von der Klasse `::geom::pointC` bereitgestellt. Die Bibliothek enthält jedoch noch weitere Funktionen. Die folgenden beziehen sich auf zwei oder drei Punkte oder ihre Interpretation als Vektor.

```
#include <geom.h>

#include <point.h>

using namespace geom;

pointC p1(2), p2(2), p3(2);

p1[0] = 0.0;
p1[1] = 0.0;
p2[0] = 1.0;
p2[1] = 0.0;
p3 = p1;

cout << "p1, p2, p3 liegen "
     << onLine(p1, p2, p3) ? "" : "nicht"
     << "auf einer Geraden" << endl;                            (1)
cout << "<p1,p2> = " << scalar(p1, p2) << endl;                 (2)
```

```
cout << "Abstand(p1,p2) = " << dist(p1, p2) << endl;             (3)
```

(1) prüft, ob die drei gegebenen 2-dimensionalen Punkte auf einer gemeinsamen Geraden liegen. Die anderen Funktionen dieses Beispiels arbeiten hingegen in beliebigen Dimensionen; (2) berechnet das Skalarprodukt zweier Vektoren, (3) den euklidischen Abstand der Punkte.

A.2.3 Routinen für Strecken und Geraden

Die Geometrie-Bibliothek kann auch Aussagen über Strecken- und Geradenbeziehungen im 2-dimensionalen Raum machen.

```
#include <geom.h>

#include <point.h>

using namespace geom;

// ...

pointC p1(2), p2(2), p3(2), p4(2);

p1[0] = 0.0;
p1[1] = 0.0;
p2[0] = 1.0;
p2[1] = 0.0;
p3[0] = 2.0;
p3[1] = 1.0;
p4[0] = 0.0;
p4[1] = 0.0;

cout << "Der Mittelpunkt der Strecke p1, p2 liegt bei"
     << medium(p1, p2) << endl;                                  (1)
cout << "Eine Normale auf die Gerade p1, p2 lautet"
     << normal(p1, p2) << endl;                                  (2)
cout << "Die Geraden p1, p2 und p3, p4 schneiden sich im Punkt"
     << lineIntersect(p1, p2, p3, p4) << endl;                   (3)
cout << "Der Winkel zwischen den Strecken p1, p2 und p2, p3 ist"
     << angle(p1, p2, p3) << endl;                               (4)

// ...
```

(1) findet den Punkt, der die Strecke `p1, p2` halbiert; (2) berechnet einen Normalenvektor der Geraden. Der Schnittpunkt zweier Geraden wird durch (3) ermittelt. Der Winkel zweier sich im Punkt `p2` schneidenden Geraden wird in (4) berechnet; der gefundene Winkel ist der von `p2, p1` zu `p2, p3` gegen den Uhrzeigersinn gemessen.

A.2.4 Routinen für Dreiecke

Auch die Routinen für Dreiecke sind auf den 2-dimensionalen Raum beschränkt.

```
#include<geom.h>

#include<point.h>

using namespace geom;

// ...

pointC p1(2), p2(2), p3(2);

p1[0] = 0.0;
p1[1] = 0.0;
p2[0] = 1.0;
p2[1] = 0.0;
p3[0] = 2.0;
p3[1] = 1.0;

cout << "Dreieck p1, p2, p3" << endl;
cout << "Mittelpunkt des Umkreises: "
     << cCenter(p1, p2, p3) << endl;                            (1)
cout << "Radius des Umkreises: "
     << cRadius(p1, p2, p3) << endl;                            (2)
cout << "Radius des Inkreises: "
     << iRadius(p1, p2, p3) << endl;                            (3)

// ...
```

Der Umkreis des Dreiecks, das durch die Punkte `p1, p2, p3` gegeben ist, wird in (1) berechnet. Die Aufrufe für Radius von Umkreis und Inkreis erfolgen analog ((2) und (3)).

A.3 Die **meshMan**-Bibliothek

Die Bibliothek `libmeshman.a` ist der Kern des **meshMan**-Pakets. Sie stellt zum einen die Datenstruktur für Triangulierungen, zum anderen Routinen zur Glättung und Verfeinerung bereit.

A.3.1 Die Typen `::base::indexT` und `::base::floatT`

Innerhalb des gesamten Paketes wird ausschließlich mit den zwei elementaren Typen `::base::indexT` für vorzeichenbehaftete ganzzahlige Operationen und `::base::floatT` für Fließkommazahlen gearbeitet. Sie sind in `base.h` definiert; dort sind auch Konstanten — `::base::`*Typ*`max` — enthalten, die die größten Werte für diese Typen angeben. Die Definitionen dieser Typen liegen in `base.h`.

A.3.2 Die Datenstruktur

Die Datenstruktur ist so aufgebaut, daß Hierarchien von Triangulierungen möglichst einfach dimensionsunabhängig gespeichert werden können. Dieser Aufbau ist in den vorigen Kapiteln dieser Arbeit detailliert beschrieben; hier soll nur ein kleiner Überblick gegeben werden.

Die Objekte einer Triangulierung

Als Objekte einer Triangulierung stehen zur Verfügung:

- `::net::nodeC` — Knoten oder Knotenpunkt (0-dimensional)
  Ein Knotenpunkt kann aus einem Punkt erzeugt werden. Er enthält neben dessen Koordinaten eine Liste der von ihm ausgehenden Segmente.

- `::net::segmentC` — Segment (1-dimensional)
  Ein Segment besteht aus einer Strecke, definiert durch zwei Knotenpunkte. Es enthält als Zusatzinformation eine Liste derjenigen Elemente, die es eingrenzt.

- `::net::elementC` — Element (2-dimensional)
  Als Elemente der Triangulierung wurden in der vorliegenden Arbeit Dreiecke gewählt. Sie sind aus Segmenten aufgebaut. Andere Polygone sind möglich.

Diese Folge von Objekten läßt sich auf höhere Raumdimensionen erweitern.

Eine Triangulierung

Eine 2-dimensionale Triangulierung besteht aus Knoten, Segmenten und Elementen, die durch Indizes in einer Hierarchie bekannt sind.

Die Hierarchie

Die Hierarchie ist zentraler Speicher für eine Folge von Triangulierungen und deren Komponenten. Der Zugriff auf eines der Objekte erfolgt durch Angabe der Art des Objektes und seines Indexes.

A.3.3 Einfügen von Objekten in die Hierarchie

Die im vorigen Abschnitt vorgestellten Objekte sind isoliert betrachtet relativ nutzlos. Um wirkungsvoll mit ihnen arbeiten zu können, müssen sie in eine zuvor initialisierte Hierarchie eingefügt werden. In dieser Hierarchie kann dann aus den Objekten eine Triangulierung zusammengesetzt werden.
Das folgende Beispiel erzeugt eine Triangulierung eines Quadrats aus zwei Dreieckselementen, indem es die nötigen Objekte in die Hierarchie einfügt.

```
#include <hierarch.h>
#include <point.h>

#include <indexSet.h>
#include <base.h>

using namespace net;
using namespace geom;
using namespace base;

// ...

hierarchC hier(2);                                                     (1)
pointC p(hier.dim());                                                  (2)
indexSetC<indexT> obj;
indexT pid[4], sid[5];

// erzeuge die Knotenpunkte der Triangulierung
p[0] = 0.0;
p[1] = 0.0;
pid[0] = hier.append(p);                                               (3)
p[0] = 1.0;
p[1] = 0.0;
pid[1] = hier.append(p);
p[0] = 1.0;
p[1] = 1.0;
pid[2] = hier.append(p);
p[0] = 0.0;
p[1] = 1.0;
```

```
pid[3] = hier.append(p);

// erzeuge die Segmente der Triangulierung
obj.clear();
obj.append(pid[0]);                                    (4)
obj.append(pid[1]);                                    (5)
sid[0] = hier.append(1, obj);                          (6)
obj.clear();
obj.append(pid[1]);
obj.append(pid[2]);
sid[1] = hier.append(1, obj);
obj.clear();
obj.append(pid[2]);
obj.append(pid[3]);
sid[2] = hier.append(1, obj);
obj.clear();
obj.append(pid[3]);
obj.append(pid[0]);
sid[3] = hier.append(1, obj);
obj.clear();
obj.append(pid[2]);
obj.append(pid[0]);
sid[4] = hier.append(1, obj);

// erzeuge die Elemente der Triangulierung
obj.clear();
obj.append(sid[0]);                                    (7)
obj.append(sid[1]);
obj.append(sid[4]);
hier.append(2, obj);                                   (8)
obj.clear();
obj.append(sid[2]);
obj.append(sid[3]);
obj.append(sid[4]);
hier.append(2, obj);

// ...
```

Durch (1) wird eine 2-dimensionale Hierarchie erzeugt. Sie kann Objekte aufnehmen, deren Dimension höchstens 2 beträgt — also 2-dimensionale Punkte, Segmente und Elemente.
(2) erzeugt einen Punkt, der mit der Dimension der Hierarchie initialisiert wird. Die-

ser Punkt wird — nachdem seine Koordinaten gesetzt wurden — durch (3) in die Hierarchie eingefügt, wobei eine für diese Hierarchie eindeutige Knoten-Id zurückgeliefert wird. Mittels dieser Id kann jederzeit auf diesen Knoten zurückgegriffen werden.
Segmente werden durch die Id's ihrer Knotenpunkte beschrieben. Die Id's der Knoten werden in ein `indexSetC`–Objekt geschrieben ((4) und (5)). Dieses `indexSetC`–Objekt wird dann in (6) unter Angabe der Dimension des gewünschten Objekts in die Hierarchie geschrieben. Dadurch wird wie bei den Punkten eine eindeutige Segment-Id geliefert.
Die Erzeugung aller k-dimensionalen ($1 \leq k \leq n$, n ist die Dimension der Hierarchie) Objekte erfolgt auf die gleiche Art und Weise wie bei den Segmenten. (7) und (8) illustrieren diesen Vorgang für die Elemente ($k = 2$). Die Id's der Elemente sind in dem Fall der 2-dimensionalen Hierarchie nicht von Bedeutung, da aus ihnen keine weiteren Objekte aufgebaut werden.
Die Hierarchie enthält neben den oben vorgestellten Objekten zusätzlich ein Netz oder Level. Jedes der eingefügten Simplizes wird beim Einfügen in die Hierarchie automatisch für das aktuelle Netz registriert. Neue Levels können nur durch Verfeinern des aktuellen Netzes erzeugt werden.

A.3.4 Der Zugriff auf Objekte einer Triangulierung

Die Objekte der Hierarchie können über ihre Id erreicht werden. Dieser Abschnitt zeigt das genaue Vorgehen hierzu, indem alle Segmente eines Levels ausgegeben werden.

```
#include <hierarch.h>
#include <net.h>
#include &lt;segment.h&gt;

using namespace net;

// ...

hierarchC h(2);

// ...
// erzeuge ein Netz in der Hierarchie h
// ...

for (hierarchC::netIterConstC nit = h.netBeginConst();
     nit != h.netEndConst();
     ++nit) {                                                    (1)
     cout << "Segmente von Level " << *nit << endl;             (2)
```

```
    for (netC::segIterConstC sit = h.getNet(*nit).segBeginConst();
         sit != h.getNet(*nit).segEndConst();
         ++sit)                                                     (3)
        cout << h.getSeg(*sit) << endl;                             (4)
}

// ...
```

Die Hierarchie stellt Iteratoren (1) für alle enthaltenen Netze zur Verfügung. Die Objekte der Netze können mit den Objekt-Iteratoren der Netze (3) durchlaufen werden.
Alle Iteratoren liefern lediglich die Indizes der Objekte (2) innerhalb der Hierarchie. Die eigentlichen Objekte werden mit dem entsprechenden `get`-Aufruf (4) erreicht.
Weitere Informationen zu den einzelnen Elementfunktionen der Objekte können dem Referenzmanual entnommen werden.

A.3.5 Nachbarschaftsbeziehungen

Keines der Objekte speichert selbst irgendwelche Nachbarschaftsverhältnisse zu anderen gleichartigen Objekten. Jedem Objekt sind nur seine höher- bzw. niederdimensionalen "Nachbarn" bekannt.
Elemente müssen die Segmente, aus denen sie aufgebaut sind, befragen, um ihre Nachbarn zu erfahren. Wie dies genau geregelt ist, zeigt das folgende Codefragment:

```
#include <hierarch.h>
#include <net.h>
#include <element.h>
#include &lt;segment.h&gt;

using namespace net;

// ...

hierarchC h(2);

// ...
// erzeuge ein Netz in der Hierarchie h, so daß ein Element
// mit Index 3 auf Level 0 existiert.
// ...

// erfrage alle Nachbarn von Element 3 auf Level 0:

elementC &elm = h.getElm(3);                                      (1)
```

```
for (elementC::segIterConstC sit = elm.segBeginConst();
     sit != elm.segEndConst();
     ++sit)
     for (  segmentC::elmIterConstC eit
          = h.getSeg(*sit).elmBeginConst();
          eit != h.getSeg(*sit).elmEndConst();
          ++eit)                                             (2)
              if (   *eit!= elm.id()
                  && h.getNet(0).testElm(*eit))               (3)
                   cout << "Die Elemente "
                        << elm.id()
                        << " und "
                        << *eit
                        << " sind benachbart"
                        << endl;

// ...
```

Das Erfragen der Nachbarschaftsinformationen ist also wenig mehr als der Zugriff auf die niederdimensionalen Nachbarn und deren Daten.
Jedes Objekt stellt Iteratoren (2) für den Zugriff auf die registrierten Objekte zur Verfügung.
Ansonsten soll hierzu nur noch (1) als Erzeugung einer Referenz eines Objektes und (3) als Test, ob ein Objekt zu einem gegebenen Level gehört, hervorgehoben werden.

A.3.6 Löschen von Objekten

Objekte werden zum einen aus Netzen, zum anderen aus der Hierarchie gelöscht. Aus einem Netz können sie entfernt werden falls in diesem Level kein höherdimensionales Objekt aus diesem Objekt aufgebaut ist. Aus der Hierarchie kann es entfernt werden, sofern es in keinem Netz enthalten ist. Erst dann wird der Speicher dieses Objektes freigegeben.
Der Löschvorgang sieht für alle Objekte der Hierarchie identisch aus.

```
#include <hierarch.h>
#include <point.h>
#include &lt;segment.h&gt;
#include <element.h>

using namespace net;

// ...
```

```
hierarchC hier;

// ...
// Dieses Beispiel setzt eine Hierarchie voraus, die ein Netz
// aus Punkten, Segmenten und Elementen enthält.
// Dabei existiere von jedem Objekt eines mit Id 1.
// Zusätzlich sei Element 1 aus Segment 1 aufgebaut,
// Segment 1 habe Knoten 1 als Rand.
// ...

hier.getElm(1).unNet(0);                                    (1)
hier.unElm(1);                                              (2)

hier.getNod(1).unNet(0);                                    (3)

// ...
```

(1) entfernt ein Element aus Level 0, (2) löscht es komplett aus der Hierarchie.
(3) löst hingegen einen Fehler aus, da (nach Voraussetzung) Segment 1 den gewünschten Knoten benötigt.

A.3.7 Verbessern der Triangulierungsqualität

Die Qualität einer Triangulierung wird mit sogenannten Glättungsverfahren verbessert. **meshMan** implementiert zwei Verfahren:

- `qualitySwap`
  Das gemeinsame Segment zweier Elemente, deren Vereinigung konvex ist, wird "herumgeklappt".

- `laplace`
  Ein Knoten wird im Schwerpunkt seines Patches plaziert, sofern dadurch die Qualität des schlechtesten Elements innerhalb des Patches verbessert wird und der Knoten die Grenzen seines Patches nicht verläßt.

Die Glättungsverfahren werden in Kapitel 5 ausführlich beschrieben.
Beide Verfahren können nur auf unverfeinerte Level 0-Triangulierung angewendet werden.

```
#include <nethelper.h>

#include <hierarch.h>
#include <net.h>
```

```
#include <element.h>
#include &lt;segment.h&gt;
#include <node.h>

using namespace net;

// ...

hierarchC h(2);

// ...
// erzeuge ein Netz in der Hierarchie h
// ...

for (netC::segIterConstC sit = h.getNet().segBeginConst();
     sit != h.getNet().segEndConst();
     ++sit)
     if (qualitySwap(h.getSeg(*sit)))
          cout << *sit
               << " wurde gekippt"
               << endl;

for (netC::nodIterConstC nit = h.getNet().nodBeginConst();
     nit != h.getNet().nodEndConst();
     ++nit)
     if (laplace(h.getNod(*nit)))
          cout << *sit
               << " wurde verschoben"
               << endl;

// ...
```

A.3.8 Verfeinern einer Triangulierung

Für Multigrid-Verfahren werden Folgen von Triangulierungen benötigt. Diese Folgen entstehen durch adaptive Verfeinerungsstrategien, deren Aufruf nun demonstriert werden soll. In dem Beispiel werden alle Elemente, deren Fläche größer als 1 ist, verfeinert. Nach dem Verfeinerungsvorgang ist natürlich noch nicht garantiert, daß die Fläche aller Elemente kleiner als 1 ist.

```
#include <hierarch.h>
#include <net.h>
```

```
#include <element.h>

using namespace net;

// ...

hierarchC h(2);

// ...
// erzeuge ein Netz in der Hierarchie h
// ...

for (netC::elmIterConstC eit = h.getNet().elmBeginConst();
     eit != h.getNet().elmEndConst();
     ++eit)
     if (h.getElm(*eit).size() > 1)
          h.getElm(*eit).mark(red);                                   (1)
h.refine();                                                           (2)

// ...
```

Die zu verfeinernden Elemente müssen zuerst einzeln markiert (1) werden. (2) berechnet den Abschluß, erzeugt ein neues Level und verfeinert alle Elemente bezüglich ihrer Markierung.

A.3.9 Spezielle Funktionen

meshMan stellt einige häufig benutzte Routinen zur Verfügung:

- `quality`
  Berechnet das Qualitätsmaß eines Elements.
- `elmNodes`
  Findet die Knotenpunkte eines Elements.
- `oppositeNodes`
  Findet die Knoten eines Elements, die nicht zu einem gegebenen Segment gehören.
- `elmsBorder`
  Sucht den Rand der Vereinigung der gegebenen Elemente.
- `patch`
  Liefert alle Elemente eines Levels, die an den gegebenen Knoten angrenzen.

- `wheel`
  Findet alle Segmente, die von einem gegebenen Knoten auf einem Level ausgehen.

A.4 Die Generator-Bibliothek

Die Generatoren automatisieren die Erstellung einer Anfangstriangulierung.
Für jede Raumdimension existieren eigene Generatoren; ein 1-dimensionaler Generator zerlegt Segmente in kürzere, ein 2-dimensionaler zerlegt gegebene Elemente in kleinere.

A.4.1 Die Basisklasse der Generatoren

Die **meshMan**-Generatoren sind alle von `::net::meshGeneratorC` abgeleitet. Diese abstrakte Basisklasse enthält folgende Funktionen:

- `identify`
  Rückgabe eines ID-String der den Generator in Textform beschreibt.
- `dim`
  Angabe der Dimension des Generators.
- `makeMesh`
  Aufruf zur Generierung einer Triangulierung; Parameter ist die Hierarchie, in der die Triangulierung erzeugt werden soll. Das Gitter wird auf Level 1 generiert, und die entsprechenden Vater-Sohn-Beziehungen zu den Segmenten auf Level 0 gesetzt. Durch diesen Schritt können die Randsegmente ohne Umnumerierung leicht identifiziert werden.

A.4.2 Die Generatoren im Detail

In **meshMan** sind folgende Generatoren enthalten:

- `::net::linear1dC` (1-dimensional)
  Dieser Generator zerlegt Strecken in kürzere, so daß keine davon länger als ein vorgegebener Wert ist.
- `::net::delaunay2dC` (2-dimensional)
  Hiermit wird ein 2-dimensionale Triangulierung mittels des Delaunay-Verfahrens erzeugt.
- `::net::afm2dC` (2-dimensional)
  afm2dC realisiert ein Advancing-Front-Verfahren mit drei Regeln. Die Klasse selbst ist abstrakt. Von ihr wurden weitere Spezialversionen abgeleitet, die sich durch das Auswahlverfahren des Basissegments unterscheiden (siehe Kapitel 3).

  - ::net::afm2dLengthC
  - ::net::afm2dAngleC
  - ::net::afm2dAngleLengthC
  - ::net::afm2dAngl2C

  Diese Generatoren haben durch die Funktion `snap` einen gemeinsamen Konfigurationsmechanismus. Der damit gesetzte Wert definiert, in welcher Entfernung von neu zu erzeugenden Punkten nach bereits existierenden gesucht werden soll.

Genauere Angaben zur Implementierung der Generatoren kann den Kapiteln 3 und 4 entnommen werden. Die Konfigurationsmethoden sind in dem Referenzhandbuch beschrieben.
Die Handhabung der Generatoren gestaltet sich sehr einfach:

```
#include <linear1d.h>
#include <delaunay2d.h>
#include <afm2dlength.h>

#include <hierarch.h>

using namespace net;

// ...

hierarchC hier(2);
linear1dC lin1d;                                          (1)
delaunay2dC del2d;
afm2dLengthC afm2d;

// ...
// einlesen des Randes des gewünschten Gebiets
// ...

cout << lin1d.identify() << endl;                         (2)
cout << lin1d.dim();                                      (3)

lin1d.edgeLength(1.0);                                    (4)
lin1d.makeMesh(hier);                                     (5)

del2d.edgeLength(1.0);                                    (6)
del2d.minDistance(0.9);                                   (7)
del2d.makeMesh(hier);                                     (8)
```

```
afm2d.makeMesh(hier);                                          (9)

// ...
```

(1) erzeugt eine Instanz des 1D-Generators. Den Generatoren wird die gewünschte Hierarchie erst als Parameter des Generator-Aufrufs (5) übergeben. Dadurch sind die Generatoren an keine Hierarchie gebunden. (2) und (3) identifizieren den Generator und seine Dimension. Die beiden anderen hier erzeugten Generatoren stellen die gleichen Funktionen zur Verfügung. Durch (4), (6) und (7) werden die Generatoren konfiguriert. (5) und (8) erzeugen schließlich die Triangulierung.
Der Aufruf (9) erzeugt hingegen einen Fehler, da in der Hierarchie `hier` bereits eine Triangulierung erzeugt wurde.

A.4.3 Der Meshcontainer

Der Meshcontainer vereinfacht den Umgang mit den einzelnen Generatoren. Er löscht alle Levels bis auf Level 0 aus der angegebenen Hierarchie. Dann werden die einzelnen Generatoren in der richtigen Reihenfolge aufgerufen (beginnend bei Dimension 1). Da die Hierarchie in den Anfangszustand mit der reinen Randbeschreibung des Gebietes versetzt wird, sind wiederholte Generierungsversuche möglich.

```
#include<meshcontainer.h>

#include<linear1d.h>
#include<delaunay2d.h>
#include<afm2dlength.h>

#include<hierarch.h>

using namespace net;

// ...

hierarchC hier(2);
linear1dC lin1d;
delaunay2dC del2d;
afm2dLengthC afm2d;
meshContainerC cont(hier.dim());                               (1)

// ...
// lies den Rand des gewünschten Gebietes ein
// ...
```

```
cont.reg(&lin1d);                                                    (2)
cont.reg(&del2d);                                                    (3)
cont.mesh();                                                         (4)

cont.reg(&afm2d);                                                    (5)
cont.mesh(&hier);                                                    (6)

// ...
```

Der Meshcontainer ist an eine bestimmte Raumdimension gebunden, die dem Konstruktor (1) übergeben wirden. Danach kann die Geometrie-Beschreibung in die Hierarchie eingelesen werden. (2) und (3) registrieren Generatoren bei dem Meshcontainer. Diese werden durch die `dim`-Funktion richtig eingeordnet und sortiert. Die Generatoren können jederzeit (5) ausgetauscht werden. Sobald für jede Dimension eine Registrierung vorliegt, kann die Generierung (4) angestoßen werden.
Da die Generierung in der Hierarchie auf Level 1 geschieht und die Randbeschreibung in Level 0 unverändert erhalten bleibt, ist der Aufruf (6) erlaubt und erzeugt keinen Fehler wie im letzten Abschnitt.

A.5 Die IO-Bibliothek

Die Export-Bibliothek `libmeshio.a` implementiert mehrere Export-Filter. Durch diese Filter kann **meshMan** dazu gebracht werden, mit externen Programmen zusammenzuarbeiten.

A.5.1 Der Netparser

Die Klasse `netparserC` wurde von E. Stafilarakis implementiert und freundlicherweise zur Verfügung gestellt. Die Programmierschnittstelle war während der Fertigstellung dieses Manuals noch ständigen Änderungen unterworfen, so daß auf eine Beschreibung verzichtet wird.

A.5.2 Xfig

Xfig ist ein Vektor-orientiertes Graphikprogramm, das auf den meisten UNIX-Workstations installiert ist. Es kann die Triangulierung darstellen und weiterverarbeiten. Die meisten Abbildungen von Triangulierungen in dieser Arbeit wurden mit Hilfe dieses Programms nach Postscript konvertiert.
Das ausgegebene Format entspricht der Xfig-Version 3.2.

```
#include<xfig.h>
```

```
#include<hierarch.h>

using namespace net;

// ...


hierarchC hier(2);

// ...
// fülle die Hierarchie und erzeuge ein Netz
// ...

xfig(&hier, "outfile.fig");                                   (1)
xfig(&hier, 3, "outfile.fig");                                (2)

// ...
```

(1) exportiert das aktuelle Level, (2) Level 3.

A.5.3 jShow

jShow ist ein Tcl/Tk-Skript, das 2D-Triangulierungen darstellen kann. Durch seine leichte Anpaßbarkeit an spezielle Situationen eignet es sich gut zur Fehlersuche in Generatoren. Dieser Filter erzeugt ein spezielles Format für dieses Skript. *jShow* beinhaltet einen Exportfilter nach Postscript.
Der Aufruf der Funktion `jshow` erfolgt analog zu dem des Xfig-Filters.

A.5.4 PNS/netCDF

Für die Zusammenarbeit mit PNS bietet **meshMan** einen Filter zur Ausgabe in das netCDF-Format an. Das Ausgabeformat numeriert die Elemente und Segmente um. Um die Adaptivität von **meshMan** nutzen zu können, müssen die Elemente mit denjenigen aus **meshMan** identifiziert werden können.

```
#include<pnscdf.h>

#include<hierarch.h>
#include<element.h>
#include<base.h>

// ...
```

```
hierarchC hier(2);
indexT i, meshmanid;
pnscdfC pns;

// ...
// erzeuge eine Triangulierung in der Hierarchie
// ...

pns.cdf(&hier, 0, "level.cdf");                                  (1)
cin >> i;
meshmanid = pns.elm(i);                                          (2)
hier.getElm(meshmanid).mark();
hier.refine();
pns.clean();                                                     (3)

// ...
```

Die Anweisung (1) schreibt Level 0 der Hierarchie `hier` im PNS-netCDF-Format in die Datei "level.cdf". Diese Datei kann nun von PNS geladen und bearbeitet werden. Soll ein Element verfeinert werden, wird die PNS-Id wieder eingelesen (im Beispiel via der Standard-Eingabe). (2) bildet diese Id auf die **meshMan**-Id ab. Über diese Id kann das Element in **meshMan** angesprochen und verfeinert werden. (3) löscht die PNS-Id's.

A.6 Ausnahmebehandlung

Alle Exceptions die in den oben vorgestellten Bibliotheken verwendet werden, basieren auf der Basisklasse `::base::errorC`, definiert in `base.h`. Aus den ausgeworfenen Exceptions kann der Name der Funktion und eine Begründung des Vorgangs abgelesen werden. Die Namensgebung der abgeleiteten Klassen ermöglicht weitere Rückschlüsse auf den Grund der Exception.

```
#include<netexceptions.h>
#include<hierarch.h>
#include<point.h>

using namespace net;
using namespace geom;
using namespace base;

// ...
```

```
hierarchC hier(2);
pointC p(hier.dim());

p[0] = 0.0;
p[1] = 0.0;
hier.append(p);
try {
    hier.append(p);                                              (1)
}
catch (distUnderrunC e) {                                        (2)
    cout << "distUnderrunC wurde abgefangen:" << endl;
    cout << "Funktion: " << e.function() << endl;               (3)
    cout << "Meldung: " << e.message() << endl;                 (4)
}

// ...
```

(1) versucht einen Punkt ein zweites Mal in die Hierarchie einzufügen. Da dies nicht erlaubt ist, wird die von `::base::errorC` abgeleitete Exception `::net::distUnderrunC` ausgeworfen. (3) gibt den Namen der Funktion aus, die die Exception ausgeworfen hat. Falls diese Funktion das entsprechende Feld nicht initialisiert hat, lautet die Ausgabe "`unknown function`". Die entsprechende Fehlermeldung wird in (4) ausgegeben. Der Default-String lautet "`unknown message`". Die Ausgabe des Programms lautet also (abgesehen von den genauen Exception-Meldungen):

```
distUnderrunC wurde abgefangen:
Funktion: ::net::hierarchC::append(.)
Meldung: Dieser Punkt liegt zu nahe an einem anderen
```

A.7 Abhängigkeiten der Bibliotheken

Die folgende Tabelle A.1 zeigt die Abhängigkeiten der Bibliotheken. Enthalten sind auch diejenigen Bibliotheken, die nicht zum **meshMan**-Paket gehören:

- m
  Ansi-C Mathematik-Bibliothek

- netcdf
  Maschinenunabhängiges Datenformat von Unidata [6]

	geom	**meshMan**	meshIO	meshCreator	m	netcdf
geom	X				X	
meshMan	X	X			X	
meshIO	X	X	X		X	X
meshCreator	X	X		X	X	

Tabelle A.1: Abhängigkeiten der Bibliotheken

Literaturverzeichnis

[1] *AGM^{3D}Homepage*. http://elc2.igpm.rwth-aachen.de/bey/agm.html.

[2] *Delaundo*. http://www.cerfacs.fr/m̃uller/delaundo.html# delaundo.

[3] *EasyMesh*. http://www-dinma.univ.trieste.it/ñirftc/research/easymesh/.

[4] *Mesh Generation & Grid Generation on the Web*. http://www-users.informatik.rwth-aachen.de/r̃oberts/meshgeneration.html.

[5] *Meshing Research Corner*. http://www.andrew.cmu.edu/user/sowen/mesh.html.

[6] *netCDF Homepage*. http://www.unidata.ucar.edu/packages/netcdf.

[7] *Triangle: A Two-Dimensional Quality Mesh Generator*. http://www.cs.cmu.edu/q̃uake/triangle.html.

[8] R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Diferential Equations, Users's Guide 7.0*, volume 15 of Frontiers in Applied Mathematics. SIAM, Philadelphia, 1994.

[9] J. Bey. *Finite-Volumen- und Mehrgitter- Verfahren für elliptische Randwertprobleme*. B. G. Teubner, 1998.

[10] P. G. Ciarlet. *Basic error estimates for elliptic problems*, volume II: Finite Element Methods (Part 1) of *Handbook of Numerical Analysis*. 1991.

[11] H. Freudenthal. Simplizialzerlegungen von beschränkter Flachheit. *Annals of Mathematics*, 43:580–582, 1942.

[12] P. L. George. *Automatic Mesh Generation*. John Wiley & Sons, 1991.

[13] N. Josuttis. *Die C++-Standardbibliothek*. Addison-Wesley, 1996.

[14] M. Koecher. *Lineare Algebra und Analytische Geometrie*, volume 2 of Grundwissen Mathematik. Springer, Berlin, Heidelberg, 1983.

[15] W. F. Mitchell. Adaptive refinement for arbitrary finite-element spaces with hierarchical basis. *J. Comput. Appl. Math.*, 36:65–78, 1991.

[16] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal of Numerical Methods in Engineering*, 20:745–756, 1984.

[17] J. Schöberl. *NETGEN — An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules.* Universität Linz, ftp://ftp.numa.uni-linz.ac.at/pub/software/.

[18] P. L. George und H. Borouchaki. *Delaunay Triangulation and Meshing.* Hermès, 1998.

[19] Ch. Großmann und H.-G. Roos. *Numerik partieller Differentialgleichungen.* B. G. Teubner, 1994.

[20] R. Verfürth. *A Review of A Posteriori Error Estimator and Adaptive Mesh-Refinement Techniques.* Wiley Teubner, 1996.

Danksagung

Allen, die zum Gelingen dieser Arbeit beigetragen haben, danke ich herzlich!

Herrn Prof. Dr. G. Lube danke ich für die interessante Themenstellung. Durch sein Interesse und seine ständige Diskussionsbereitschaft hat er den Fortgang dieser Arbeit in vorbildlicher Weise unterstützt.

Mein besonderer Dank gilt Herrn Dipl. math. A. Priessnitz, der durch zahlreiche Diskussionsbeiträge beigetragen hat, die auftretenden Probleme zu bewältigen.

Herrn E. Stafilarakis danke ich für die Integrierung seines Netzparsers in den Programmcode.

Herrn H. Schilling danke ich für das ausführliche Testen des Programmcodes.

Allen Mitgliedern der CFD-Gruppe danke ich für die angenehme Arbeitsatmosphäre.

Last but not least möchte ich mich bei meinen Eltern und meiner Freundin Janna bedanken.