Gittererzeugung

Norbert Köckler

WS 2004/5

Inhaltsverzeichnis

Vorwort

Dieses Skriptum ist entstanden für eine Vorlesung, die gemeinsam von Friedhelm Neugebauer und mir im WS 1996/97 für Informatik- und Mathematik-Studenten gehalten wurde. Es war auch Grundlage für eine Projektgruppe der AG Monien und für ein Fortgeschrittenenpraktikum bei mir; und es ist in den letzten Jahren von Diplomanden beider Arbeitsgruppen mehrfach als Grundlage ihrer Arbeit benutzt worden.

Im WS 1998/99 habe ich diese Vorlesung allein gehalten, weil Herr Neugebauer aus den Diensten der Uni geschieden ist. Inzwischen habe ich die Ergebnisse der Diplomarbeiten zu kleinen Teilen in dieses Skriptum eingearbeitet. Natürlich werden die Diplomanden an den entsprechenden Stellen auch erwähnt.

Im Laufe des Semesters können Unterschiede zwischen Skript und Vorlesung auftreten, auf die ich dann aufmerksam machen werde.

Es ist mir ein Bedürfnis, Herrn Neugebauer auch hier nochmal für die gute und fruchtbare Zusammenarbeit zu danken.

Norbert Köckler — Paderborn, im August 2004

Kapitel 1

Einleitung

Bevor eine PDE mit einer FEM oder einem Differenzenverfahren gelöst werden kann, muß auf dem Grundgebiet Ω ein Gitter vorhanden sein, das für die FEM die geometrischen Elemente, für ein Differenzenverfahren die inneren Punkte und deren Nachbarn definiert.

Die notwendigen Schritte zur Lösung einer partiellen Differentialgleichung (PDE) wie

$$\begin{array}{lrl} \text{PDE:} & Lu = f & \text{in } \Omega \\ \text{RB:} & u = g & \text{in } \partial\Omega \end{array} \tag{1.1}$$

wollen wir noch einmal näher ansehen:

(1) Geometrische Konfiguration mit Diskretisierung und Gittererzeugung.

(2) Numerische Lösung der PDE.

Wir wollen uns nur mit der Gittererzeugung beschäftigen. Für die Erzeugung eines Gitters kommen mehrere Basismethoden in Frage:

(1) Erzeugung von Hand, d.h. Eingabe einer Koordinatenliste.

(2) Erzeugung einer groben Gitterstruktur per Hand, Verfeinerung automatisch uniform oder adaptiv mit entsprechenden Verfeinerungsalgorithmen.

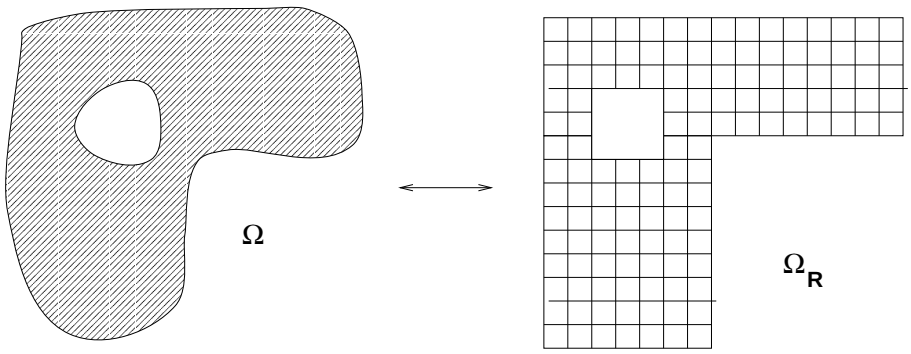
(3) Erzeugung eines **strukturierten** Gitters automatisch mit einem entsprechenden 'mathematischen' Algorithmus. Oft ist dies eine Transformation von einem Standardgitter auf das zu erzeugende *physikalische* Gitter. Auch hier sind mehrere uniforme oder adaptive Verfeinerungsstufen möglich.

(4) Erzeugung eines **unstrukturierten** Gitters automatisch mit einem entsprechenden 'Informatik'-Algorithmus. Auch hier sind mehrere uniforme oder adaptive Verfeinerungsstufen möglich.

In allen Softwarepaketen zur Lösung partieller Differentialgleichungen sind heute Methoden zur Gittererzeugung und/oder Gitterverfeinerung enthalten.

Bei den Transformationsverfahren wird das Gesamtschema zur Lösung der PDE komplizierter, da das Gitter durch Transformation auf ein einfacheres Gebiet erzeugt wird. Damit muß aber auch die PDE transformiert werden und kann danach in der Regel nur noch iterativ gelöst werden. Damit kommen wir zu folgendem Lösungsschema:

(1) Geometrische Konfiguration mit Diskretisierung und Gittererzeugung:



(2) Gittererzeugung mittels Transformation, algebraisch oder iterativ:

(a) Erzeugung eines Startgitters (algebraische Methoden)

(b) Iterative Verbesserung.

(3) Transformation der PDE auf Ω_R

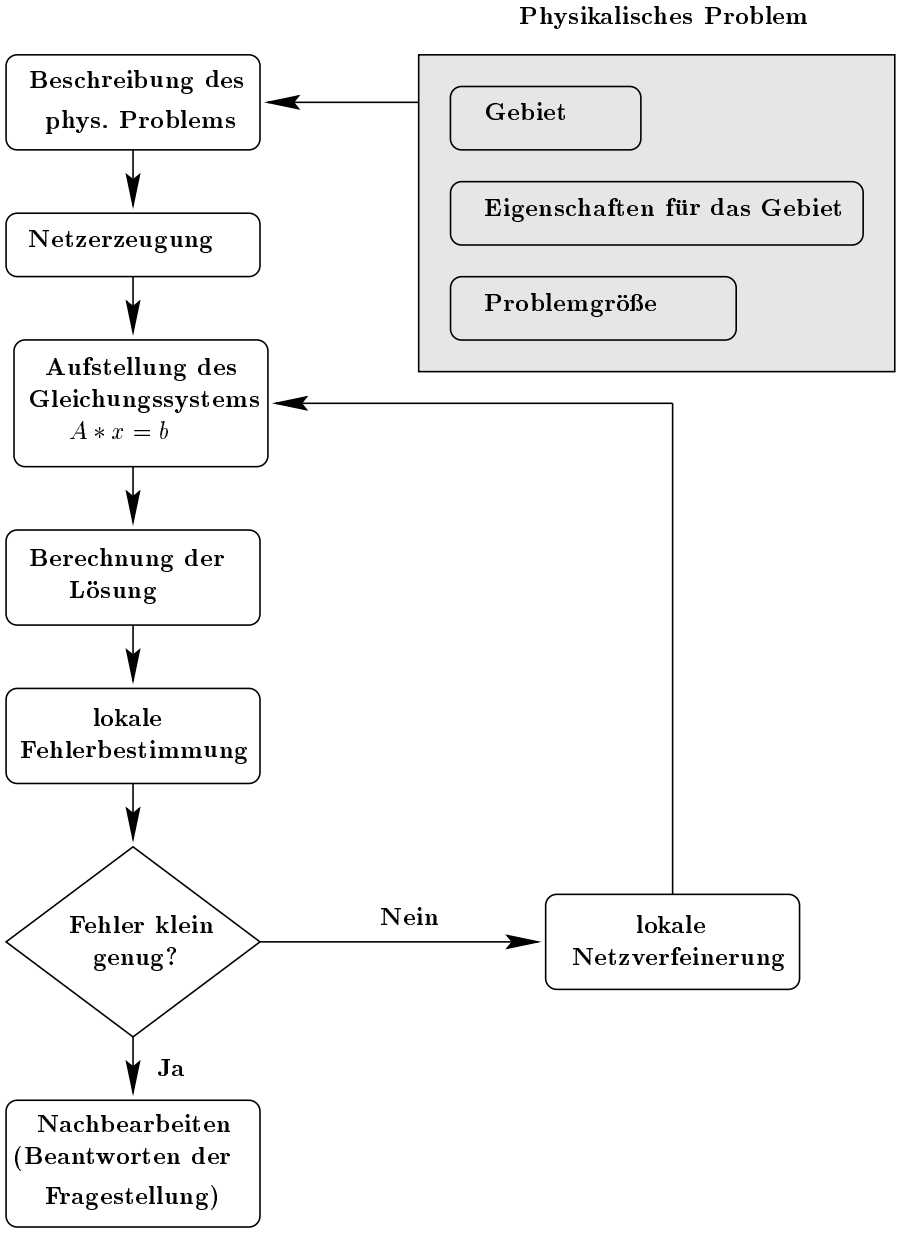
$$\left.\begin{array}{rcl} Lu = f & \text{in} & \Omega \\ u = g & \text{auf} & \partial\Omega \end{array}\right\} \rightarrow \left\{\begin{array}{rcl} \tilde{L}(\tilde{u}) = \tilde{f} & \text{in} & \Omega_R \\ \tilde{u} = \tilde{g} & \text{auf} & \partial\Omega_R \end{array}\right. \tag{1.2}$$

(Selbst wenn L linear war, kann und wird normalerweise $\tilde{L}$ nicht-linear sein, aber nur schwach nicht-linear, sozusagen quasi-linear.)

(4) Numerische Lösung der PDE (1.2) mit einer möglichst einfachen und schnellen Methode, die aber jetzt iterativ sein muß, weil PDE (1.2) nichtlinear ist.

Bei den unstrukturierten Methoden, die oft zur Erzeugung von Finite-Elemente-Netzen (z.B. Triangulierung) benutzt werden, ergibt sich ein ähnlicher Ablauf, aber ohne, daß die PDE nochmal transformiert werden muß:

Ein einfaches FEM–System



Schema: Friedhelm Neugebauer

Wir wollen in diesem Kapitel zunächst graphisch orientiert einen Eindruck von den Möglichkeiten und Zielen der Gittererzeugung bekommen. Dazu habe ich Beispiele aus verschiedenen Vorträgen und Diplomarbeiten zusammengestellt. Stellvertretend nenne ich die Namen von Friedhelm Neugebauer und Jörg-Udo Aden.

Anschließend wollen wir uns mit den Methoden näher beschäftigen.

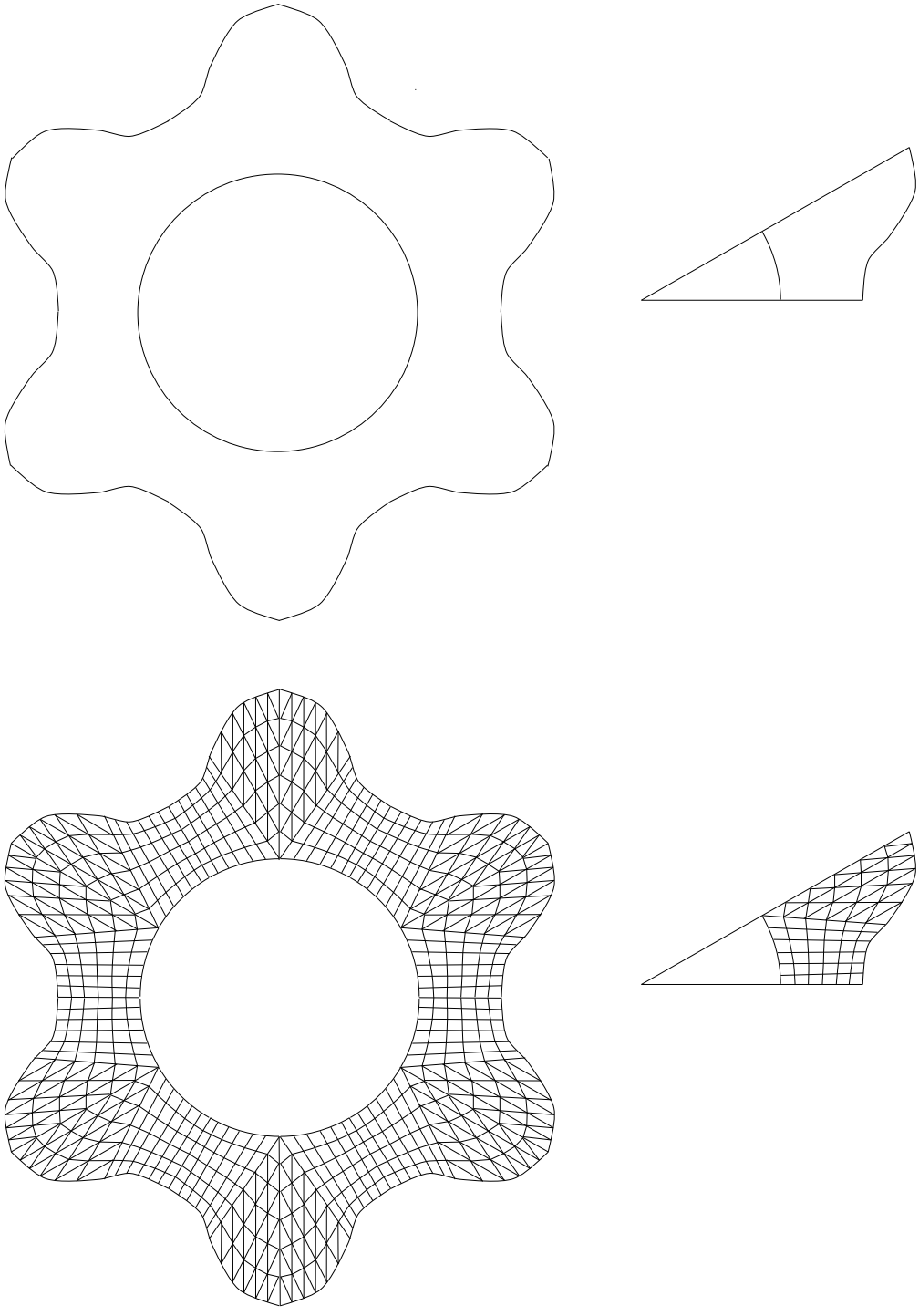
Bei einigen Verfahren werde ich auch auf die Parallelisierbarkeit der Algorithmen eingehen.

1.1 Ein Überblick

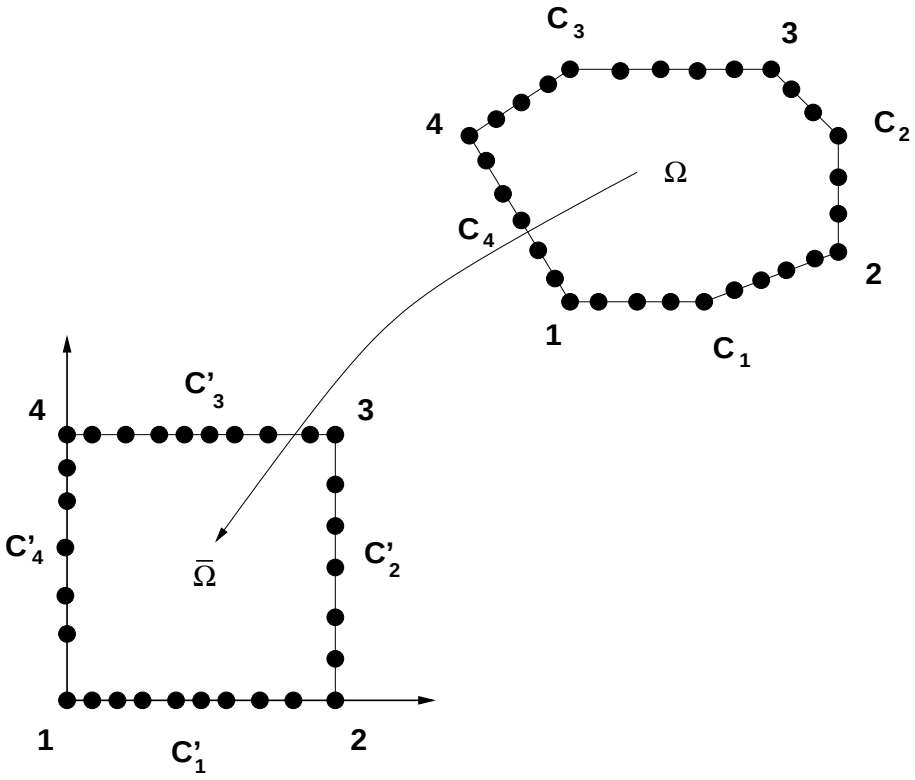
In diesem ersten Abschnitt wollen wir einen Überblick über die wesentlichen Verfahren bekommen. Dies soll überwiegend optisch geschehen, um einen Eindruck von den verschiedenen Möglichkeiten der Gittererzeugung zu gewinnen.

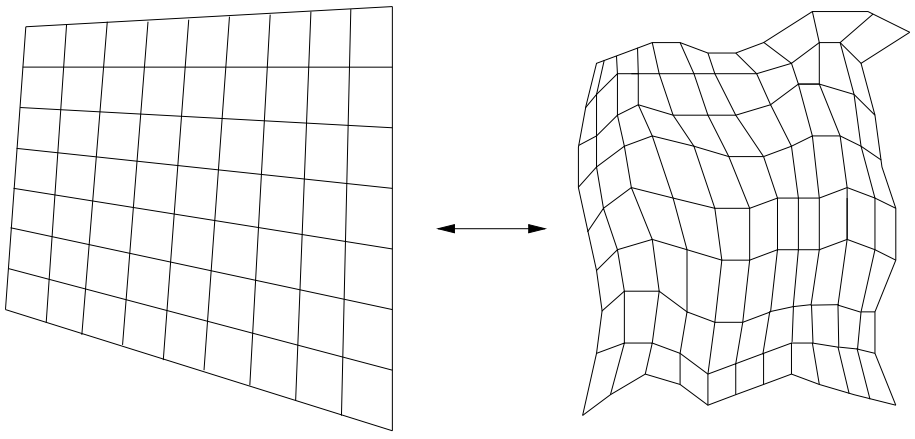
Der zweite Abschnitt zeigt Beispielgitter, schließt also mit der Absicht der optischen Eingewöhnung in unser Thema direkt an den ersten Abschnitt an.

1.1.1 Manuelle Aufteilung des Gebietes und Einsetzen von definierten Mustern

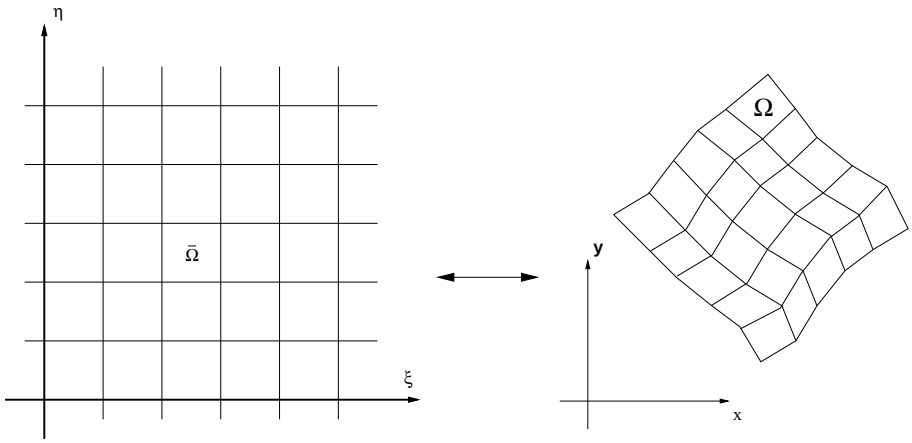


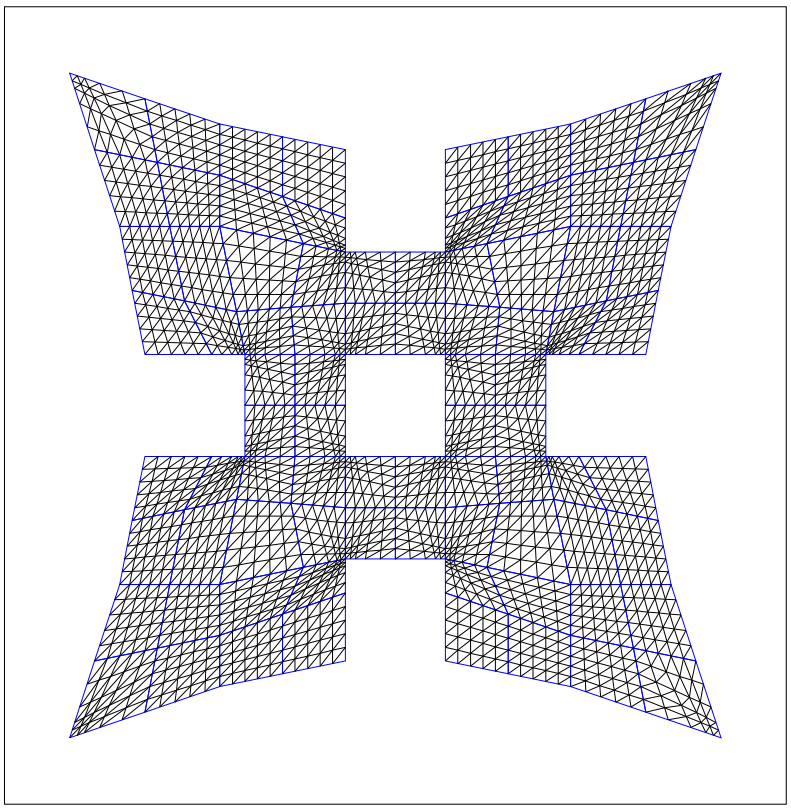
1.1.2 Transformation eines einfach zu generierenden oder bekannten Netzes in das gewünschte Gebiet



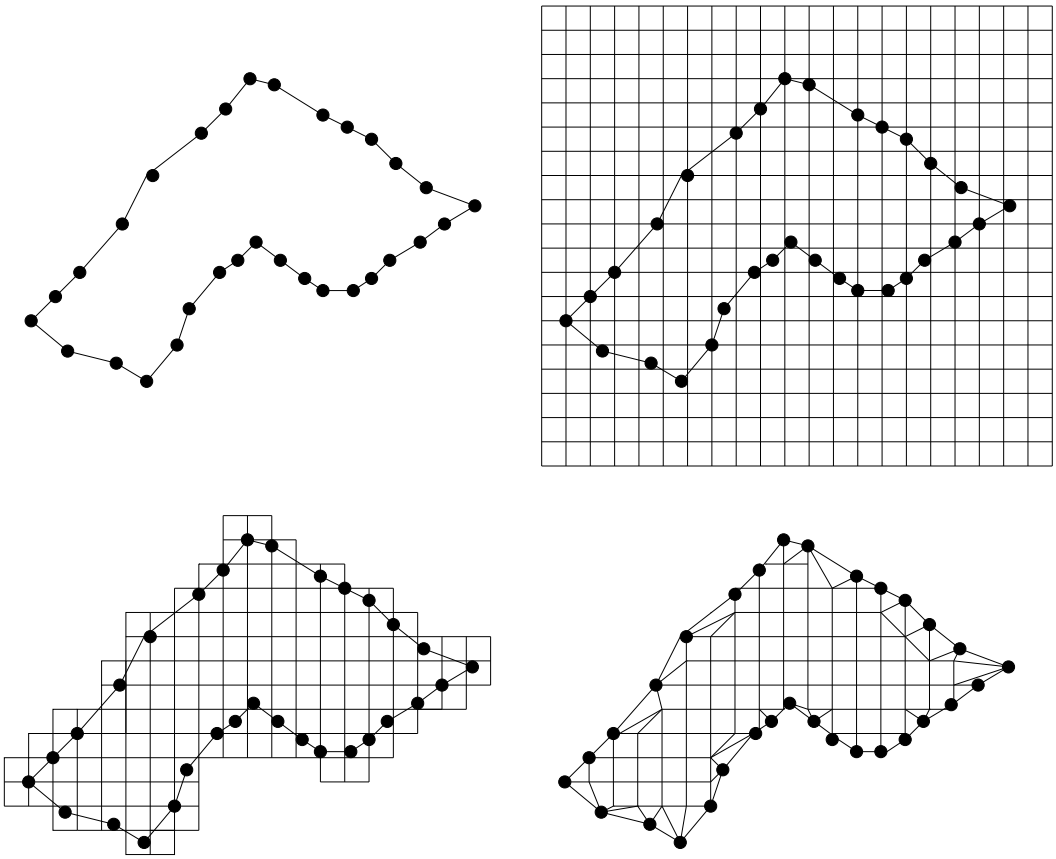


1.1.3 Transformationen mit partiellen Differentialgleichungen

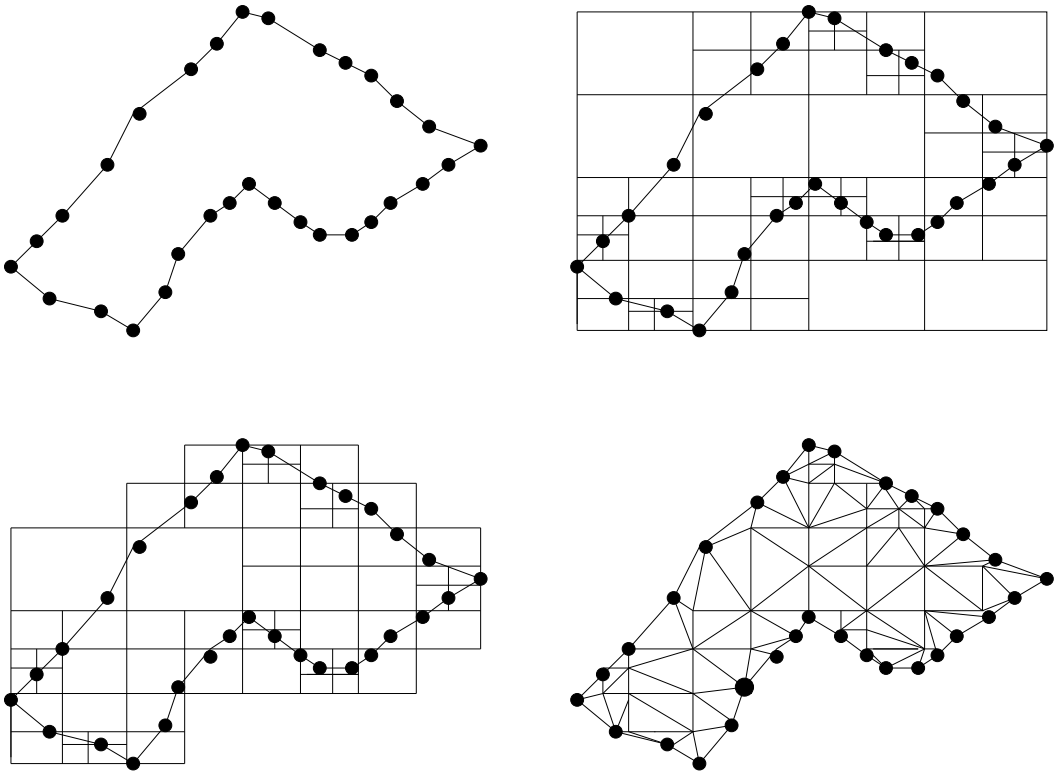




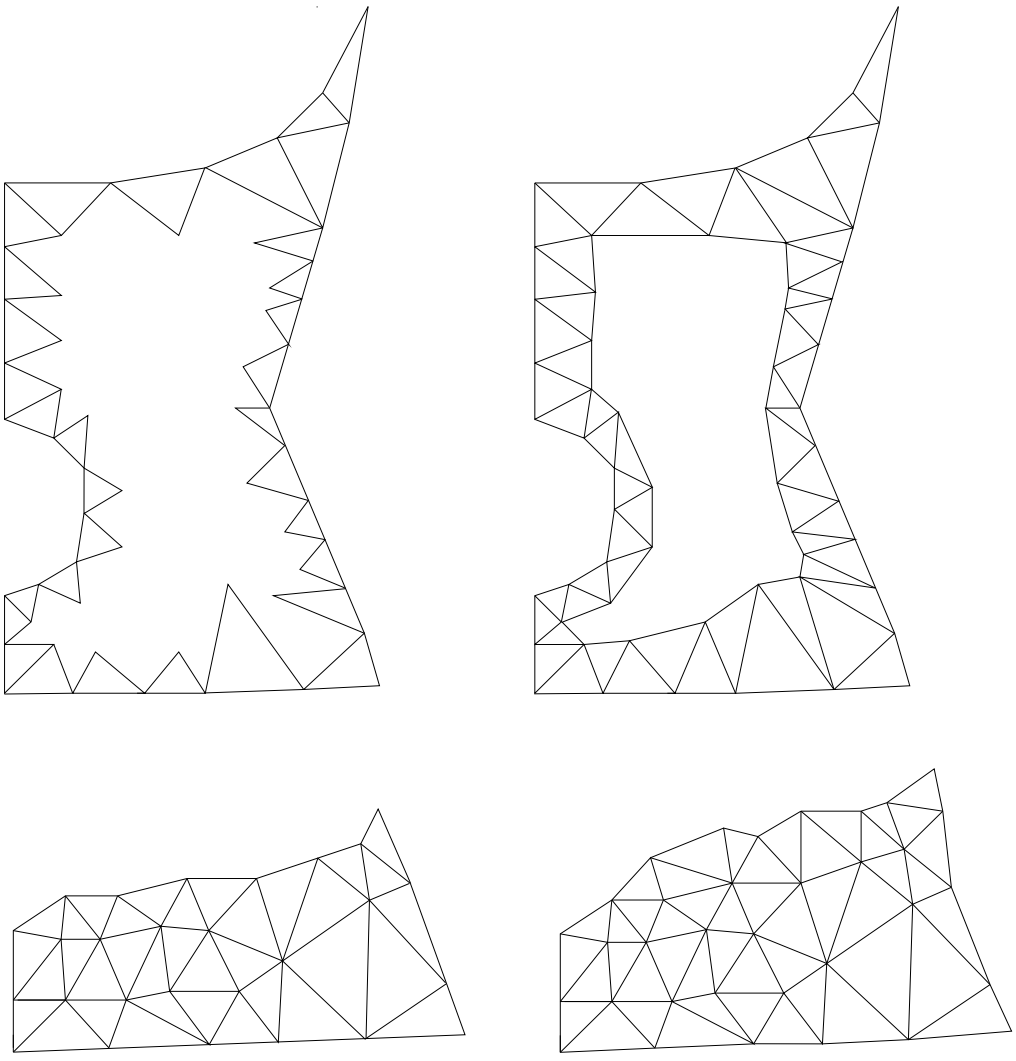
1.1.4 Gitterüberlagerung und Deformierung



1.1.5 Gebietszerlegung (Domain Decomposition) (Quadtree, Octree)



1.1.6 Schrittweise Gebietsauffüllung (Advancing Front)



1.1.7 Greedy Triangulierung

Definition 1.1.1 Sei S eine Menge von Punkten in der Ebene. Die Greedy Triangulierung $GT = GT(S)$ entsteht durch folgenden Algorithmus:

> Sei L die Menge aller Verbindungsgeraden in S.
>
> Sei GT die leere Menge.
>
> **while** $L \neq \emptyset$ **do**
>
> Bestimme die kürzeste Kante $w \in L$;
>
> $GT \rightarrow GT \cup \{w\}$;
>
> $L \rightarrow L - \{w\} - \{m \in L \,|\, w$ schneidet m 'mittig'$\}$;
>
> **od**;

'Mittiges' Schneiden heißt: w schneidet m in einem inneren Punkt.

Ein Beispiel für diesen Algorithmus sehen wir in Abbildung 1.1.

1.1.8 Delauney Triangulierung

Definition 1.1.2 DT: Sei S eine Menge von Punkten in der Ebene. Eine Triangulierung T ist Delauney Triangulierung (DT), wenn für jede Kante e von T ein Kreis C existiert, der folgende Eigenschaften erfüllt:

(1) Die Endpunkte von e liegen auf C.

(2) Kein anderer Knoten von S liegt im Inneren von C.

Definition 1.1.3 CDT: Sei G ein planarer Graph. Eine Triangulierung T heißt *beschränkte (constrained) Delauney Triangulierung* (CDT), wenn jede Kante von G eine Kante in T ist und für jede der verbleibenden Kanten e der Triangulierung ein Kreis C existiert, der folgende Eigenschaften erfüllt:

(1) Die Endpunkte von e liegen auf C.

(2) Wenn ein Knoten v von G in diesem Kreis liegt, dann kann dieser Knoten von mindestens einem Endpunkt von e nicht gesehen werden, d.h. die Verbindung dieses Endpunktes mit v schneidet eine andere Kante von G.

Ein Beispiel für eine Delauney Triangulierung sehen wir in Abbildung 1.2.

Eine Delaunay Triangulierung existiert zu jeder Punktmenge. Die Delaunay Triangulierung ist das Dual des Voronoi-Diagramms (VD) (siehe unten und optisch in Abbildung 1.2. Sie läßt sich am einfachsten über das VD bestimmen. Die Verbindung aller benachbarter Zellen des VD erzeugt die Delaunay Triangulierung.

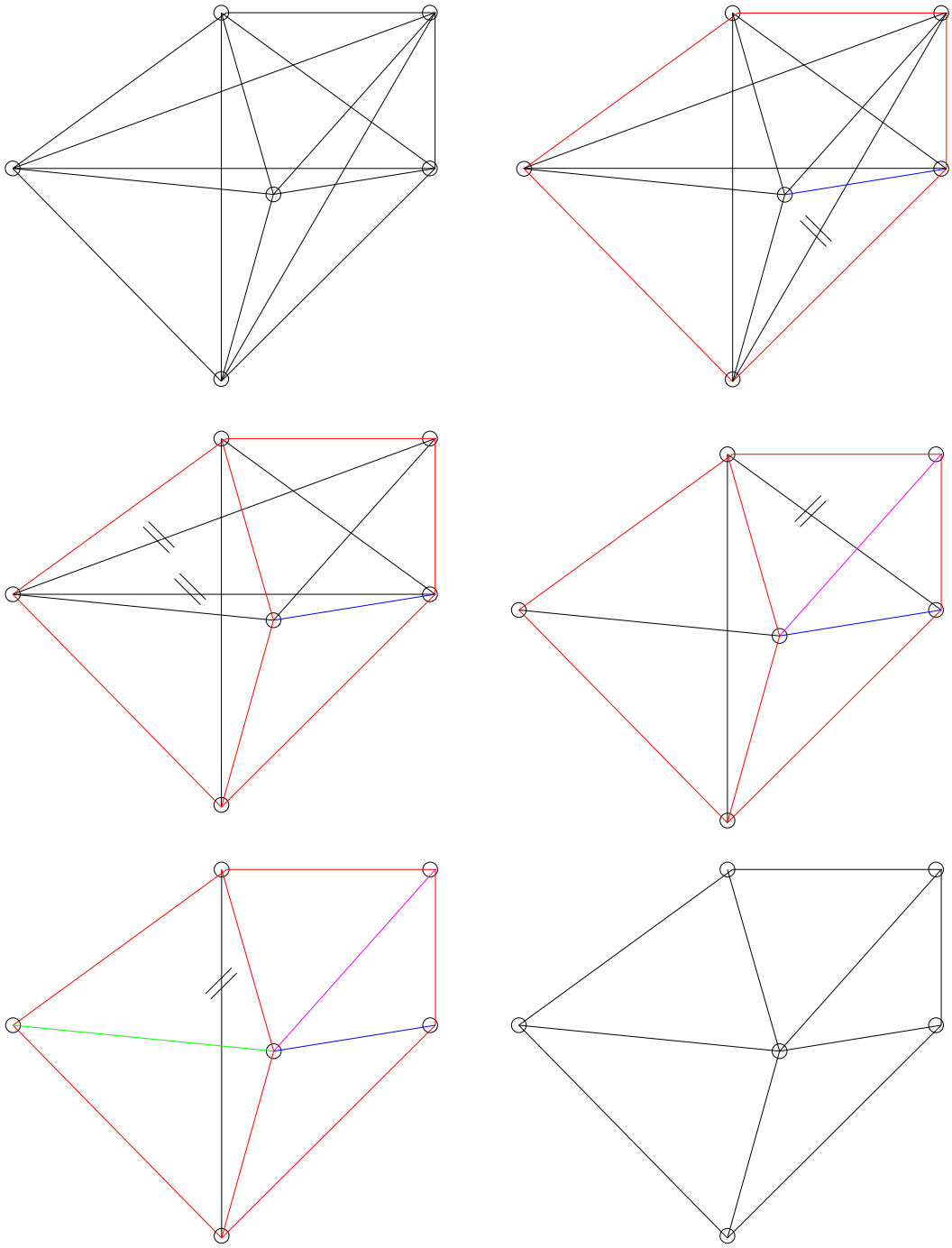


Abbildung 1.1: Beispiel für die Greedy-Triangulierung einer 6-Punkte-Menge

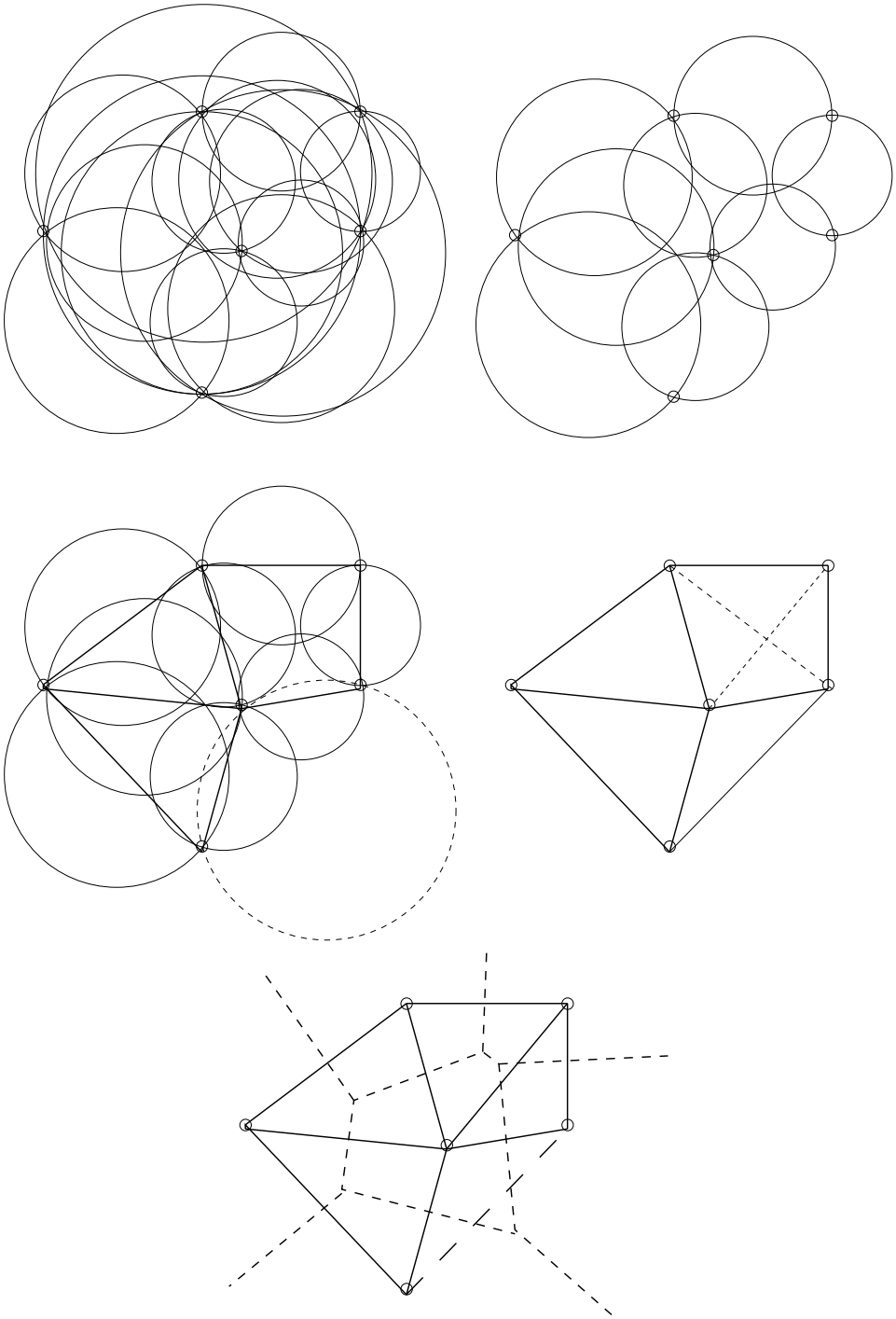
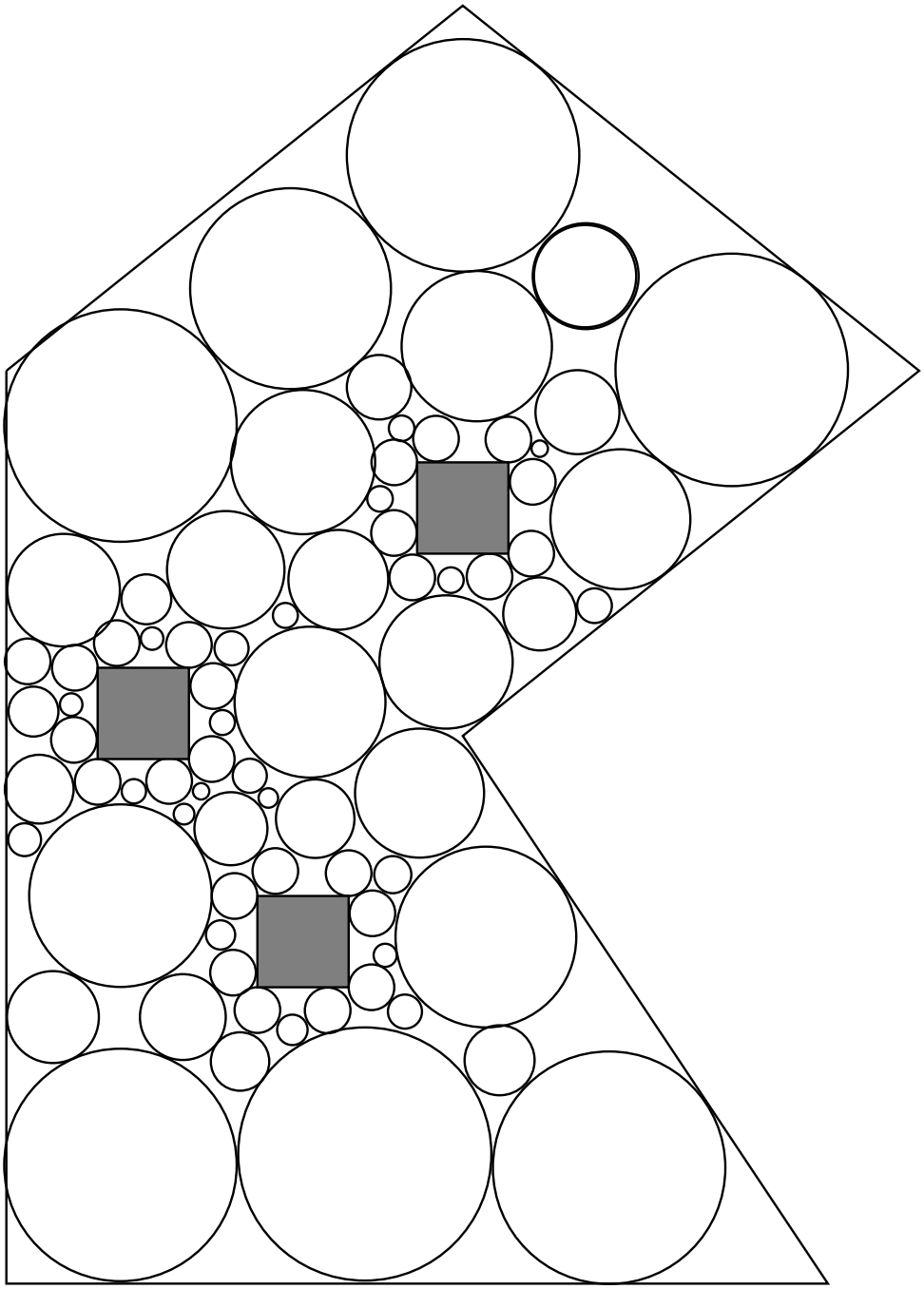


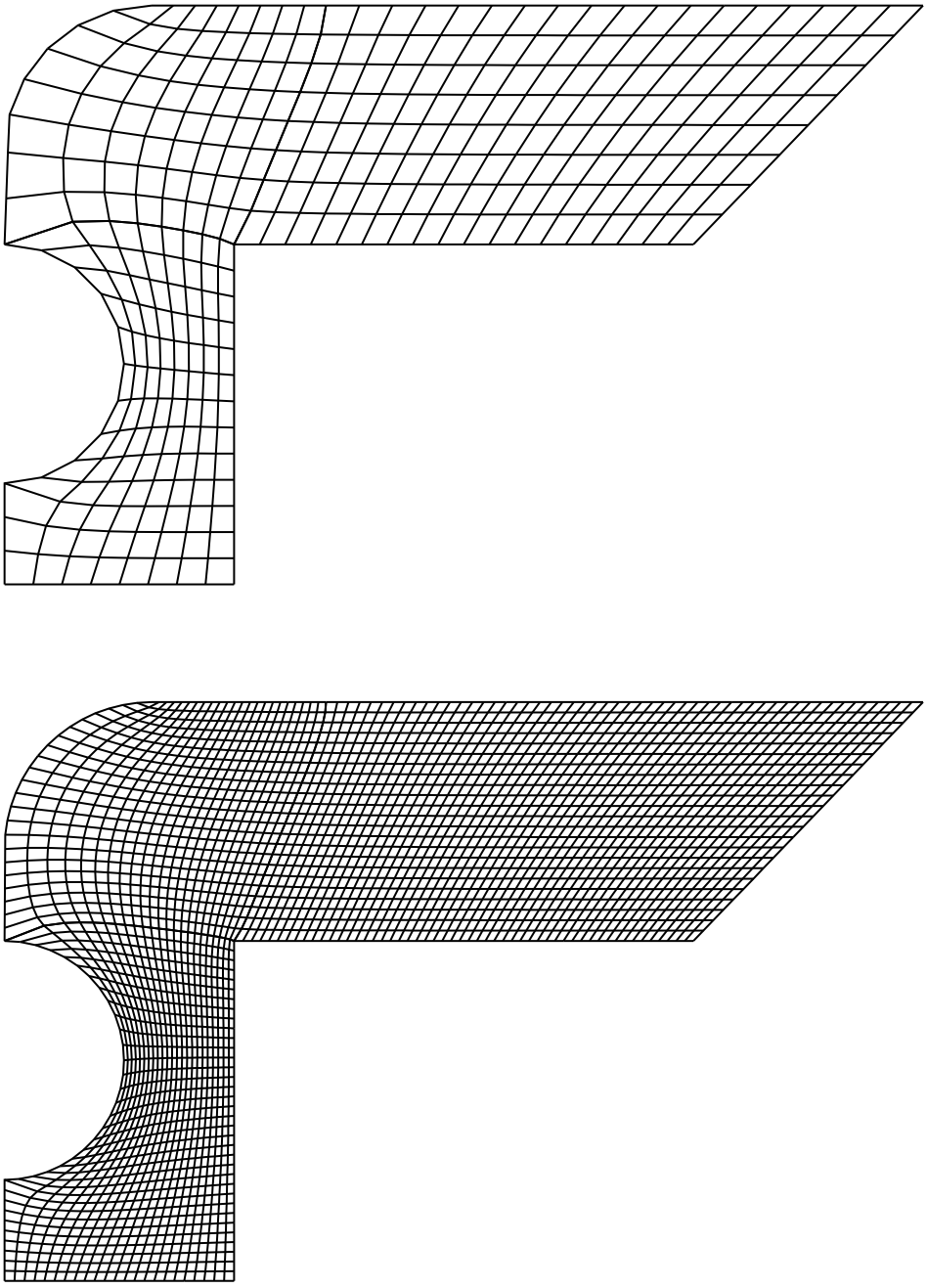
Abbildung 1.2: Delauney Triangulierung und Voronoi-Diagramm für eine 6-Punkte-Menge

1.1.9 Disc Packing

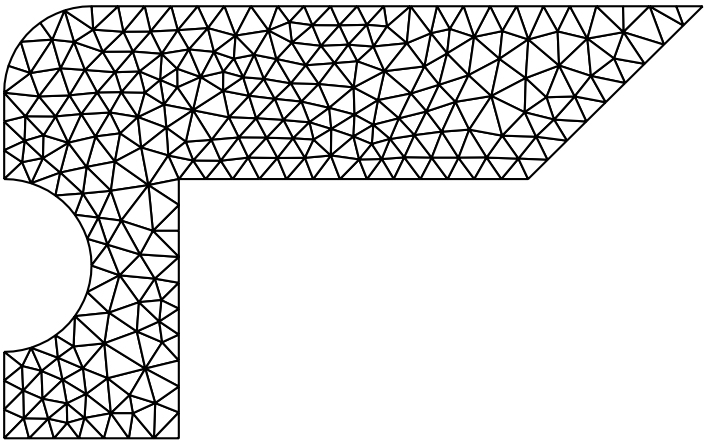


1.2 Beispielgitter

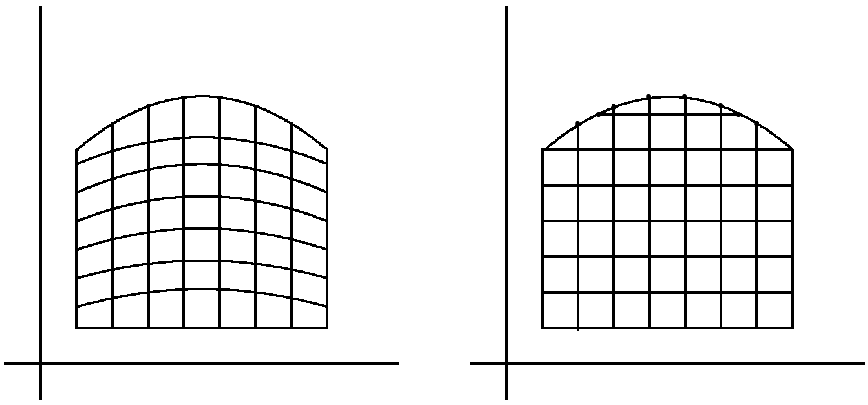
Zwei Transformationsgitter



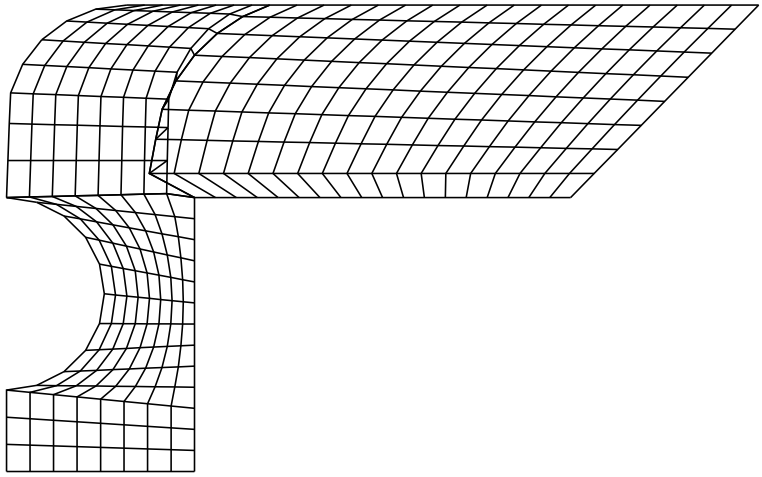
Ein Dreiecksnetz von PLTMG



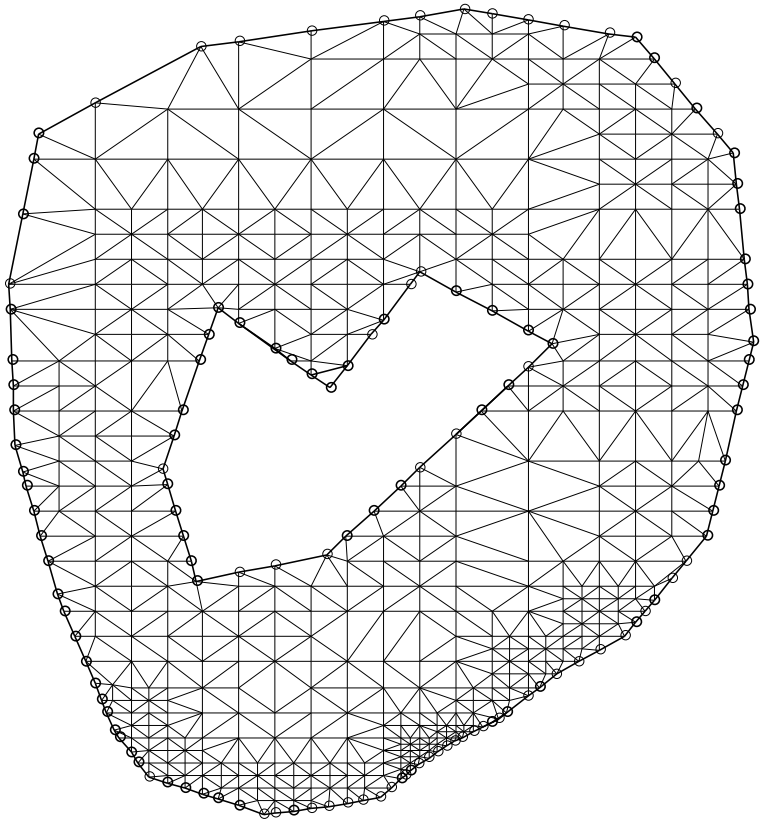
Randkonform vs. Nicht-randkonform



Transfinite Interpolation: Gitter mit Falten



Quadtree



Dreidimensionale Oberflächen

Wir wollen uns einige zusätzliche Kopien aus industriellen Anwendungen ansehen. Sie wurden von professioneller (teurer!) Software erzeugt.

Kapitel 2

Strukturierte Gittererzeugung, Transformationsmethoden

2.1 Geometrische Möglichkeiten für randkonforme Koordinatensysteme

Beispiel 2.1.1 Ein einfacher Kreisring:

Das Gebiet Ω ist ein kreisförmiges Gebiet (siehe Zeichnung 2.1), das mit zylindrischen Koordinatenlinien überdeckt ist.

Dabei ist Ω gegeben durch

$$\Omega = \{(r, \varphi) | r_1 < r < r_2, 0 < \varphi \leq 2\pi\}$$

mit den beiden Koordinaten (r, φ) und der bekannten Beziehung zwischen kartesischen und Polar-Koordinaten, die man auch als Transformation und inverse Transformation interpretieren kann:

$$\left.\begin{array}{l} x(r,\varphi) = r\cos\ \varphi \\ y(r,\varphi) = r\sin\ \varphi \end{array}\right\} \longleftrightarrow \left\{\begin{array}{l} r(x,y) = \sqrt{x^2+y^2} \\ \varphi(x,y) = \arctan y/x \end{array}\right. .$$

Ω läßt sich im physikalischen (r, φ)-Raum als Rechteck $[r_1, r_2] \times [0, 2\pi]$ darstellen, wobei der Rand $\partial\Omega$ gegeben ist durch zwei Geraden $r = r_1$ und $r = r_2$.

Der nicht-triviale Teil dieses Beispiels läßt sich wie folgt beschreiben:

Im (r, φ)-Raum wird das Gebiet Ω durch seinen Rand $\partial\Omega$ nicht vollständig eingeschlossen. Das liegt daran, daß es sich um ein mehrfach zusammenhängendes Gebiet handelt, bei der wir einen sog. Verzweigungsschnitt *branch-cut* angewendet haben, um zu einem einfach zusammenhängenden Rechteck zu gelangen.

Eine andere Transformation für den Kreisring wäre:

$$(r_1, r_2) \times (0, 2\pi] \quad \longrightarrow \quad (0, 1)^2 \ \text{ mit}$$

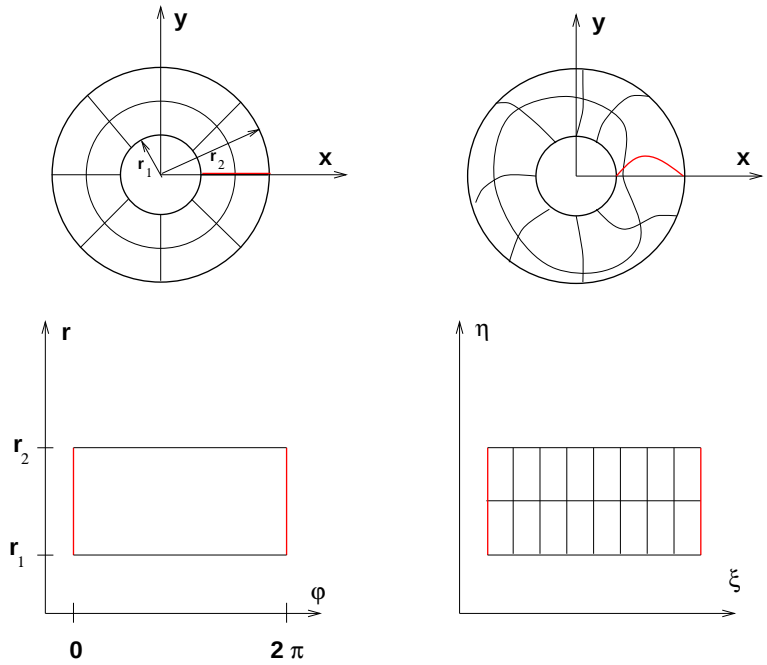


Abbildung 2.1: Der Kreisring als branch-cut-Beispiel

$$\begin{aligned} \xi &:= \frac{\varphi}{2\pi} \\ \eta &:= \frac{r - r_1}{r_2 - r_1} \quad \text{oder} \\ x(\xi,\eta) &= [r_1 + (r_2 - r_1)\eta] \cos(2\pi\xi) \\ y(\xi,\eta) &= [r_1 + (r_2 - r_1)\eta] \sin(2\pi\xi) \\ & \xi, \eta \in (0,1) \end{aligned}$$

Beispiel 2.1.2 Wir können diese Transformation verallgemeinern:

$$\begin{aligned} x(r,\varphi) &= r(\eta)\ \cos(\varphi(\xi)) \\ y(r,\varphi) &= r(\eta)\ \sin(\varphi(\xi)). \end{aligned} \tag{2.1}$$

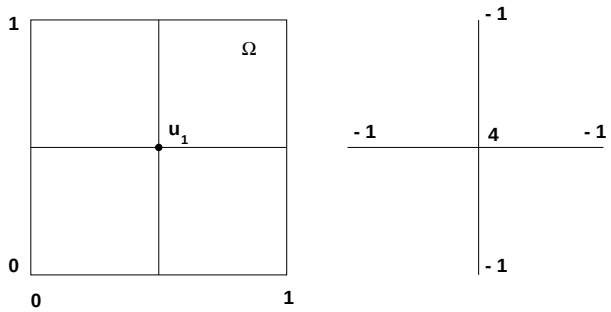
Für diese Möglichkeit sehen wir in Zeichnung 2.1 rechts ein Beispiel.

Beispiel 2.1.3 Das Branch-Cut-Prinzip bei einem PDE-Beispiel

Sehen wir uns zur Verdeutlichung des Branch-Cut-Prinzips noch einmal die einfachste PDE mit der einfachsten Diskretisierung an:

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega = (0,1) \times (0,1) \\ u &= 0 \text{ auf } \partial\Omega \end{aligned} \tag{2.2}$$

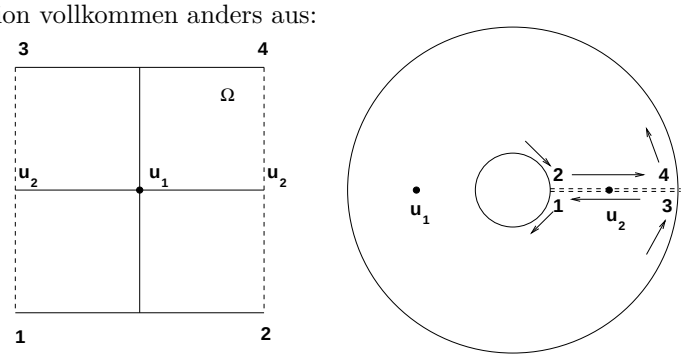
Diese PDE wollen wir mit $h = 0.5$ äquidistant diskretisieren, so daß nur ein innerer Punkt entsteht:



Wir erhalten eine lineare Gleichung:

$$\begin{aligned} 4u_1 &= h^2 f(0.5, 0.5) \\ u_1 &= \frac{1}{16} f(0.5, 0.5) \end{aligned}$$

Ist das Einheitsquadrat aus einer Branch-Cut-Transformation entstanden, dann sieht diese einfache Situation vollkommen anders aus:



In diesem Fall gibt es zwei unbekannte Punkte, so daß man zwei lineare Gleichungen erhält:

$$\begin{aligned} 4u_1 - 2u_2 &= h^2 f(0.5, 0.5) \\ -2u_1 + 4u_2 &= h^2 f(1, 0.5) \end{aligned}$$

Dabei wurde die Transformation der PDE nicht berücksichtigt!
Dieses Beispiel hat folgende Prinzipien aufgezeigt:

(1) In speziellen Fällen sind krummlinige Koordinaten geeignet, randkonforme Gitter zu erzeugen. Während die eine Koordinate längs eines Randes konstant bleibt, variiert die andere monoton (wachsend oder fallend).

(2) Branch-Cuts sind geeignet für mehrfach zusammenhängende und für kompliziertere Gebiete.

Wir wollen unser Beispiel noch einmal verallgemeinern:

$$\begin{aligned} \Omega &:= \{(x,y) | r_1^2 < x^2 + y^2 < r_2^2\} \\ \partial\Omega &= C_1 \cup C_2 : \\ C_1 &:= \{(x,y) | x^2 + y^2 = r_1^2\} \\ C_2 &:= \{(x,y) | x^2 + y^2 = r_2^2\} \end{aligned}$$

Bestimme (Finde) zwei Abbildungen $\xi(x,y), \eta(x,y)$, die für $(x,y) \in \overline{\Omega} := \Omega \cup \partial\Omega$ definiert sind, sodaß

(1) $\eta = 0$ für $x^2 + y^2 = r_1^2$
$\eta = 1$ für $x^2 + y^2 = r_2^2$,

(2) ξ auf $[0,1]$ monoton variiert auf C_1 und C_2 mit demselben Richtungssinn,

(3) gewisse Restriktionen erfüllt sind, s.u..

ABER: Wir benötigen die inverse Transformation:

Finde

$$x(\xi,\eta),\ y(\xi,\eta) \text{ für } (\xi,\eta) \in \Omega_R,\ \Omega_R = (0,1)^2$$

sodaß

z.B.	$x^2(\xi,0) + y^2(\xi,0) = r_1^2$	$\eta = 0$
	$x^2(\xi,1) + y^2(\xi,1) = r_2^2$	$\eta = 1$
Periodizität:	$x(1+\xi,\eta) = x(\xi,\eta)$	$\xi = 0,1$
	$y(1+\xi,\eta) = y(\xi,\eta)$	
und	$(x(\xi,\eta), y(\xi,\eta)) \in \Omega$	für $(\xi,\eta) \in \Omega_R$

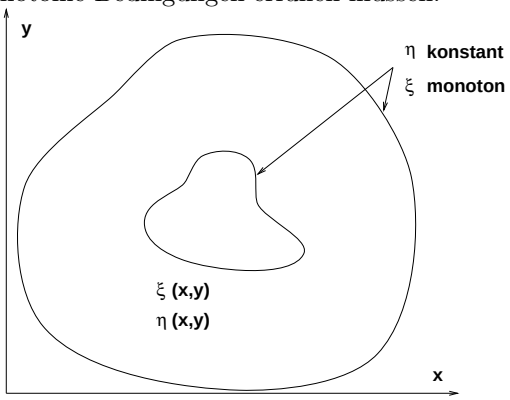
gilt, denn dann kann man ein Gitter in Ω erzeugen, das dem Quadratgitter in Ω_R entspricht.

Das hier am Beispiel geschilderte Konzept kann übertragen werden auf die Transformation von beliebigen Gebieten.

Wenn wir die Transformationen weiter verallgemeinern, erhalten wir folgende Gleichungen:

$$\left.\begin{array}{l} r(\xi,\eta) \\ \varphi(\xi,\eta) \end{array}\right\} \longleftrightarrow \left\{\begin{array}{l} x(\xi,\eta) \\ y(\xi,\eta) \end{array}\right. \qquad \xi,\eta \in (0,1).$$

Jetzt können Punkte in $r-$ oder $\varphi-$ Richtung sehr allgemein variieren, auch wenn die Funktionen weiterhin Monotonie-Bedingungen erfüllen müssen:



Aber für diesen allgemeinen Fall müssen wir offensichtlich zwei Restriktionen einführen.

Notwendige Restriktionen

(1) Krummlinige Koordinatenlinien derselben Koordinate dürfen sich nicht schneiden.

(2) Zwei Linien unterschiedlicher Koordinaten sollen sich genau einmal schneiden.

Bei Verletzung dieser Restriktionen sprechen wir von gefaltetem Gitter.

Wir sollten zur Kenntnis nehmen, daß wir Polar-Koordinaten (r, φ) nur deshalb gewählt haben, weil wir diese gut kennen. Im allgemeinen wird man kein solches Koordinatensystem verwenden können.

Zusammenfassend kommen wir zu den folgenden Transformationsschritten:

(1) Finde eine glatte, gleichmäßige Transformation

$$\Omega \to \Omega_R$$

die eine Reihe "vernünftige" Bedingungen erfüllt.

(2) Bestimme die Inverse dieser Transformation:

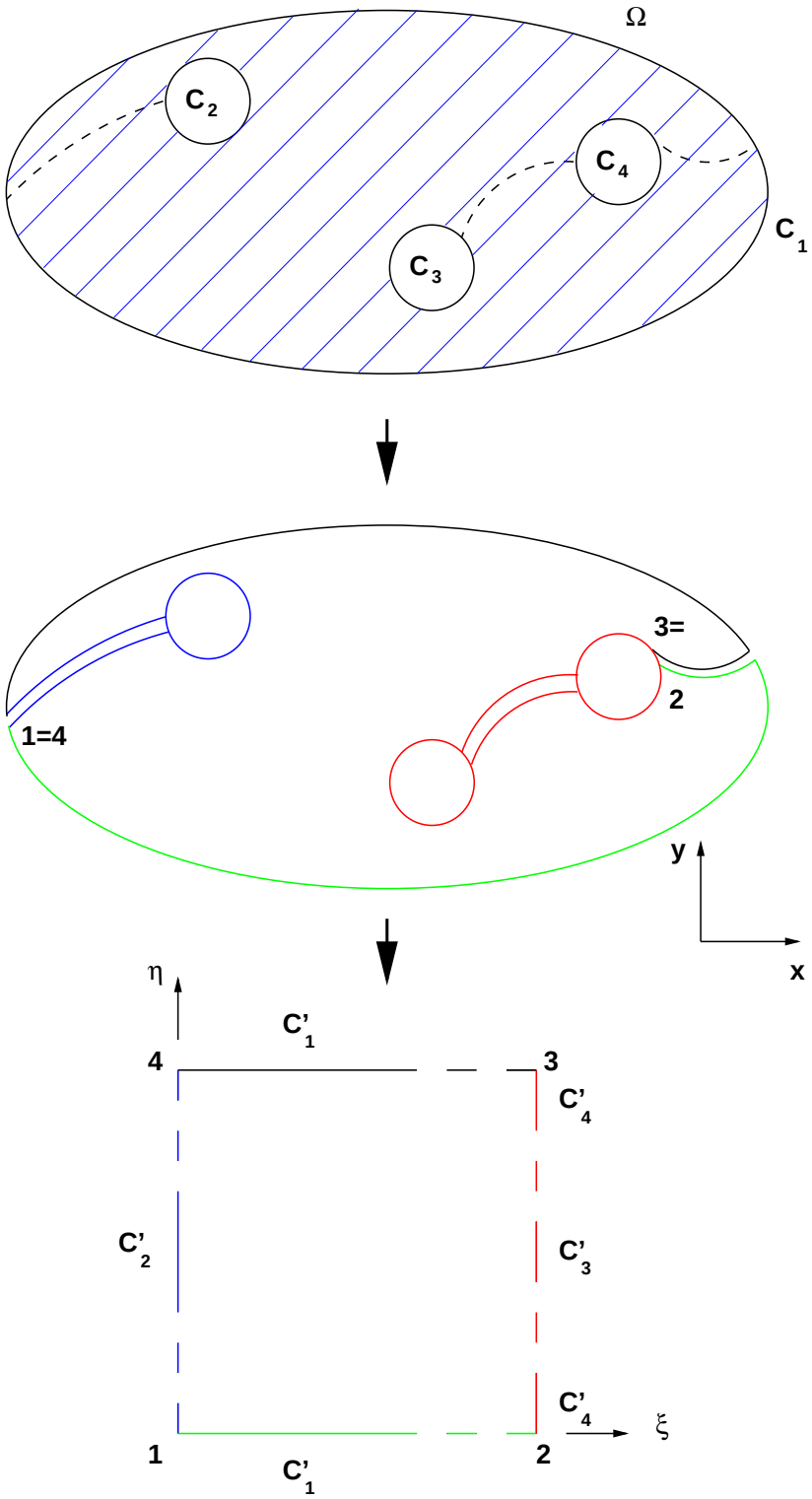
$$\Omega_R \to \Omega$$

(3) Erzeuge ein Gitter auf Ω durch Berechnung aller Punkte $(x_i, y_i) \in \Omega$, die Abbildung der inversen Transformation von Punkten $(\xi_i, \eta_i) \in \Omega_R$ eines quadratischen Gitters sind. Mit

$$\begin{array}{l} \tilde{\xi}_k := k\,h,\ k = 1, \ldots, n-1, \\ \tilde{\eta}_j := j\,h,\ j = 1, \ldots, n-1, \end{array} \quad n = \frac{1}{h},$$

wird $(\xi_i, \eta_i) := (\tilde{\xi}_k, \tilde{\eta}_j)$ (alle Kombinationen) $i = 1, \ldots, (n-1)^2$ ein quadratisches Gitter.

Beispiel 2.1.4 Zum Abschluß dieses Abschnitts wollen wir uns noch ein mehrfach zusammenhängendes Gebiet mit entsprechenden branch cuts ansehen:


Ω
C2
C4
C1
C3
3=
2
1=4
y
x
η
C'1
4
3
C'4
C'2
C'3
C'4
ξ
1
2
C'1

2.2 Algebraische Gittererzeugung I: Transfinite Interpolation

Um die inneren Gitterpunkte

$$(x_i, y_i) \in \Omega$$

zu bestimmen, wenden wir Interpolationsmethoden und randabgleichende Ansätze an. Dazu stellen wir die Region $\Omega \subset \mathbb{R}^2$ als Kurvenfamilien $(x(\xi,\eta), y(\xi,\eta))$ dar, die von den beiden Parametern ξ und η abhängen:

$$\overline{\Omega} = \{x(\xi,\eta), y(\xi,\eta) | 0 \leq \xi, \eta \leq 1\}$$

Dies ist äquivalent zu der Transformation des physikalischen Gebietes auf das Einheitsquadrates (i.a. auf irgendein Rechteck).

1. Schritt: Definiere für $\partial\Omega$ vier "Kanten" und vier "Ecken" und zerlege $\partial\Omega$ dementsprechend:

$$\partial\Omega = \underbrace{C_1}_{\text{"unten"}} \cup \underbrace{C_2}_{\text{"links"}} \cup \underbrace{C_3}_{\text{"oben"}} \cup \underbrace{C_4}_{\text{"rechts"}}$$

mit

$$\begin{aligned} C_1 \cap C_2 &= \big(x(0,0), y(0,0)\big) \\ C_2 \cap C_3 &= \big(x(0,1), y(0,1)\big) \\ C_3 \cap C_4 &= \big(x(1,1), y(1,1)\big) \\ C_4 \cap C_1 &= \big(x(1,0), y(1,0)\big), \end{aligned}$$

2. Schritt: Randdarstellung

Die einzelnen Teile des Randes sind wie folgt gegeben:

entweder

als Gitter-Randpunkte $(x_i, y_i) \in \partial\Omega$, z.B. $(x_i, y_i) = \big(x(\xi_i, 0), y(\xi_i, 0)\big)$ für C_1 $(0 \leq \xi_i \leq 1)$.

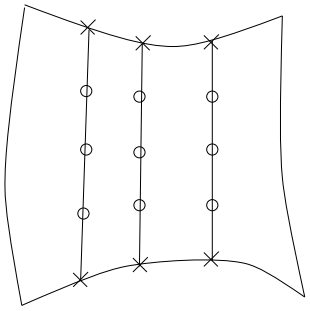
oder

als Randfunktionen, ebenfalls gegeben als Funktionen der Parameter (ξ, η):

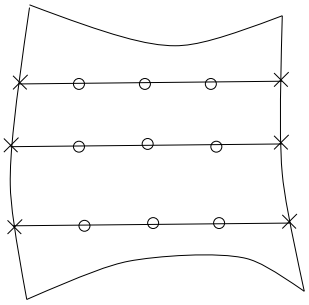
$$\begin{aligned} C_1 = &\quad \{(x_1(\xi), y_1(\xi)) | 0 \leq \xi \leq 1\} \quad \eta = 0 \\ C_2 = &\quad \{(x_2(\eta), y_2(\eta)) | 0 \leq \eta \leq 1\} \quad \xi = 0 \\ C_3 = &\quad \{(x_3(\xi), y_3(\xi)) | 0 \leq \xi \leq 1\} \quad \eta = 1 \\ C_4 = &\quad \{(x_4(\eta), y_4(\eta)) | 0 \leq \eta \leq 1\} \quad \xi = 1 \end{aligned}$$

Zunächst werden wir uns drei Interpolationsmethoden näher ansehen, die direkte Anwendungen der Lagrange-Interpolation sind:

Vertikale Interpolation
zwischen den Gitterpunkten des unteren und des oberen Teils des Bereichs $C_1 - C_3$ (hier linear):



Horizontale Interpolation
entsprechend:



Die Gitterpunkte sind definiert durch äquidistante Punkte der Parametern ξ und η.

Transfinite Interpolation besteht aus einer Kombination von vertikaler Interpolation und horizontaler Korrektur. Diese Korrektur ist ein zusätzlicher Term, der gebildet wird aus der Differenz zwischen den Differenzen zwischen den Rändern des physikalischen Bereichs und den Differenzen der Eckpunkte.

Lagrange-Interpolation ist definiert durch

$$\begin{aligned} x(\xi) &= \sum_{n=0}^{N} \Phi_n(\xi) x_n \\ \text{mit } \Phi_n(\xi) &= \prod_{i=0, i\neq n}^{N} \frac{\xi - \xi_i}{\xi_n - \xi_i}, \quad \xi_i = ih, \quad h = \frac{1}{n}. \end{aligned} \tag{2.3}$$

(Hier haben wir die Darstellung auf die x-ξ-Koordinaten beschränkt.)

Bei linearer Interpolation haben wir nur zwei Punkte – z.B. $\xi_0 = 0$ auf C_2 und $\xi_1 = 1$ auf C_4 bei festem η –, gegeben und es ist

$$\begin{aligned} x(\xi, \eta) &= x_0\,(1 - \xi) + x_1\,\xi \\ y(\xi, \eta) &= y_0\,(1 - \xi) + y_1\,\xi \end{aligned}$$

Die vertikale Interpolation mit $\eta_0 = 0$, $\eta_1 = 1$ und den Bereichsfunktionen

$$\big(x_1(\xi), y_1(\xi)\big) \text{ auf } C_1 \qquad \text{und} \qquad \big(x_3(\xi), y_3(\xi)\big) \text{ auf } C_3$$

führt zu

$$\begin{aligned} x_v(\xi, \eta) &= (1 - \eta)\, x_1(\xi) + \eta\, x_3(\xi) \\ y_v(\xi, \eta) &= (1 - \eta)\, y_1(\xi) + \eta\, y_3(\xi) \end{aligned} \tag{2.4}$$

Nun nehmen wir $k + 1$ Punkte auf C_1 und C_3

$$\begin{aligned} (x_{1i}, y_{1i}) &= \big(x_1(\xi_i), y_1(\xi_i)\big) \\ (x_{3i}, y_{3i}) &= \big(x_3(\xi_i), y_3(\xi_i)\big) \end{aligned}$$

mit $\xi_i = i\,h$, $h = \frac{1}{k}$, $i = 0, \ldots, k$, und $m+1$ Werte $\eta_j = j\,\tau$, $\tau = \frac{1}{m}$ auf C_2 und C_4.
So erhalten wir $(m-1)\,(k-1)$ innere Gitterpunkte:

$$\begin{array}{rcll} x(\xi_i,\eta_j) & = & (1-\eta_j)\,x_{1i} + \eta_j\,x_{3i} & \\ y(\xi_i,\eta_j) & = & (1-\eta_j)\,y_{1i} + \eta_j\,y_{3i} & \\ & & i = 1,\ldots,k-1, \qquad j = 1,\ldots,m-1. & \end{array} \tag{2.5}$$

Horizontale Interpolation ist entsprechend gegeben durch:

$$\begin{array}{rcl} x_h(\xi,\eta) & = & (1-\xi)x_2(\eta) + \xi\,x_4(\eta) \\ y_h(\xi,\eta) & = & (1-\xi)y_2(\eta) + \xi\,y_4(\eta) \end{array} \tag{2.6}$$

Die transfinite Interpolation ist dann definiert als

$$\begin{array}{rcl} x(\xi,\eta) & = & x_\nu(\xi,\eta) + (1-\xi)\,(x_2(\eta) - x_\nu(0,\eta)) + \xi\,(x_4(\eta) - x_\nu(1,\eta)) \\ y(\xi,\eta) & = & y_\nu(\xi,\eta)) + (1-\xi)\,(y_2(\eta) - y_\nu(0,\eta) + \xi\,(y_4(\eta) - y_\nu(1,\eta)) \end{array} \tag{2.7}$$

bzw. bei Zusammenschluß aller Terme (hier nur x, y entsprechend)

$$\begin{array}{rcl} x(\xi,\eta) & = & (1-\eta)x_1(\xi) + \eta\,x_3(\xi) + (1-\xi)x_2(\eta) + \xi\,x_4(\eta) \\ & & -\,\{\xi\,\eta\,x_3(1) + \xi\,(1-\eta)\,x_1(1) \\ & & +\eta\,(1-\xi)\,x_3(0) + (1-\xi)(1-\eta)\,x_1(0)\} \end{array} \tag{2.8}$$

Beispiel 2.2.1 Im folgenden betrachten wir ein einfaches Beispiel mit nur einem zu berechnenden inneren Gitterpunkt $\big(x(\frac{1}{2},\frac{1}{2}), y(\frac{1}{2},\frac{1}{2})\big)$.

Ω ist ein fast rechteckiges Gebiet $(0,8)\times(0,6)$ mit nur zwei krummlinigen Rändern C_1 und C_2. Die Ränder sind gegeben durch die folgende Punkte:

$$\begin{array}{lccc} C_1: & (0,0) & (4,1) & (8,0) \\ C_2: & (0,0) & (-2,2) & (0,6) \\ C_3: & (0,6) & (4,6) & (8,6) \\ C_4: & (8,0) & (8,3) & (8,6) \end{array}$$

Vertikale Interpolation (2.4) führt zu

$$\left.\begin{array}{l} x_\nu(\frac{1}{2},\frac{1}{2}) = \frac{1}{2}\,4 + \frac{1}{2}\,4 = 4 \\ y_\nu(\frac{1}{2},\frac{1}{2}) = \frac{1}{2}\,1 + \frac{1}{2}\,6 = 3.5 \end{array}\right\} \rightarrow (4, 3.5).$$

Horizontale Interpolation (2.6) liefert die Werte

$$\left.\begin{array}{l} x_h(\frac{1}{2},\frac{1}{2}) = \frac{1}{2}(-2) + \frac{1}{2}\,8 = 3 \\ x_h(\frac{1}{2},\frac{1}{2}) = \frac{1}{2}\,2 + \frac{1}{2}\,3 = 2.5 \end{array}\right\} \rightarrow (3, 2.5),$$

die sich von denen der vertikalen Interpolation deutlich unterscheiden.

Um die horizontale Korrektur der vertikalen Interpolation berechnen zu können, benötigt man zusätzlich die Werte der vertikalen Interpolation für die Ränder C_2 und C_4:

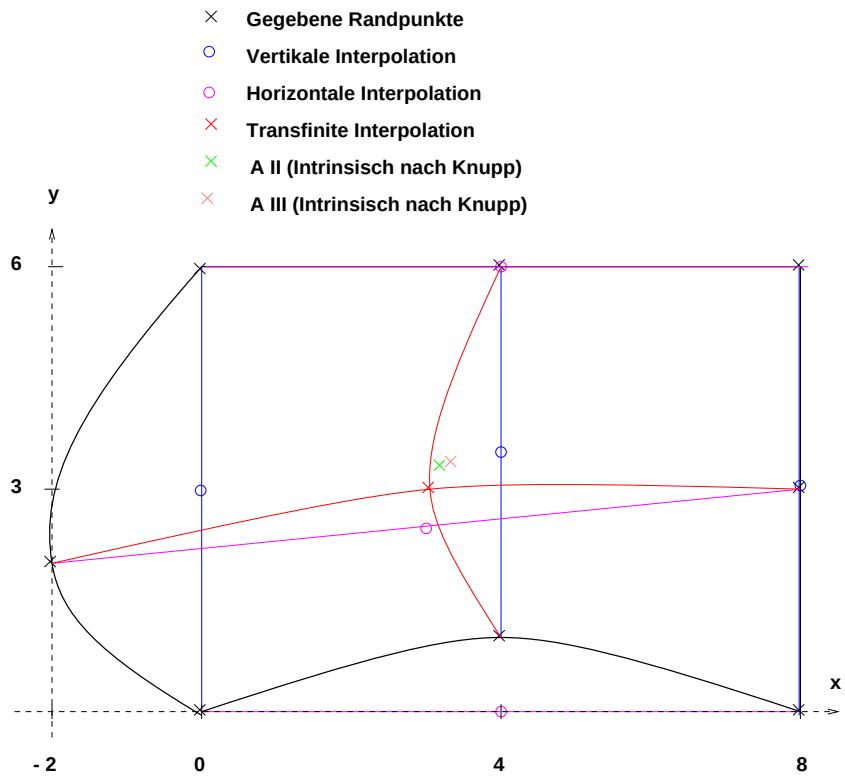
$$\left.\begin{aligned} x_\nu(0,\tfrac{1}{2}) &= \tfrac{1}{2}\,0 + \tfrac{1}{2}\,0 = 0 \\ y_\nu(0,\tfrac{1}{2}) &= \tfrac{1}{2}\,0 + \tfrac{1}{2}\,6 = 3 \end{aligned}\right\} \quad \rightarrow \quad (0,3),$$

$$\left.\begin{aligned} x_\nu(1,\tfrac{1}{2}) &= \tfrac{1}{2}\,8 + \tfrac{1}{2}\,8 = 8 \\ y_\nu(1,\tfrac{1}{2}) &= \tfrac{1}{2}\,0 + \tfrac{1}{2}\,6 = 3 \end{aligned}\right\} \quad \rightarrow \quad (8,3).$$

Mit diesen Werten kann die transfinite Interpolation (2.7) wie folgt berechnet werden:

$$\left.\begin{aligned} x(\tfrac{1}{2},\tfrac{1}{2}) &= 4 + \tfrac{1}{2}(-2-0) + \tfrac{1}{2}(8-8) = 3 \\ y(\tfrac{1}{2},\tfrac{1}{2}) &= 3.5 + \tfrac{1}{2}(2-3) + \tfrac{1}{2}(3-3) = 3 \end{aligned}\right\} \rightarrow (3,3).$$

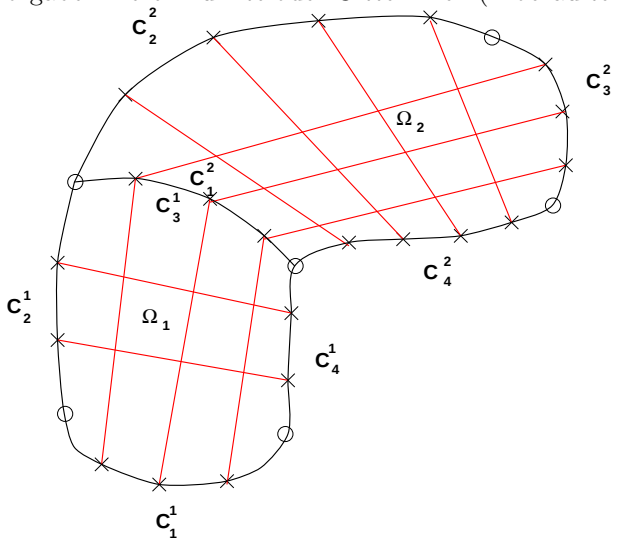
Mit Hilfe von (2.8) hätten diese Werte auch direkt ermittelt werden können.



Spezialfälle:

(1) Rekonstruktion fehlender Randteile durch Interpolation.

(2) Zusätzliche Vorgabe innerer Punkte oder Gitterlinien (Blockaufteilung des Gebietes)



Dabei sind die Teilgebiets- oder Block-Eckpunkte durch Kreise und die vorgegebenen Randpunkte durch Kreuze gekennzeichnet.

Wie leicht zu erkennen ist, sind transfinite Interpolationskurven stetig und Rand-konform. Aber die entstehenden Gitter können gefaltet sein. Das ist für die Lösung einer PDE nicht akzeptabel, deshalb kommen solche Methoden nur zur Erzeugung von Startgittern für bessere Methoden wie "Gittererzeugung mit Differentialgleichungen" in Betracht, da solche iterativen Methoden eine Startnäherung benötigen.

Einige Mathematiker haben versucht, bessere Gitter mit Hilfe allgemeinerer algebraischer Methoden zu erzeugen. Die intrinsische (wesentliche) algebraische Gittererzeugung wurde von P. M. Krupp eingeführt, siehe Castillo, [18] (Kap. 6). Zwei dieser Methoden werden wir in Abschnitt 2.4 näher betrachten.

2.3 Gittererzeugung mit Differentialgleichungen

Wir weisen noch einmal auf die wichtigsten Eigenschaften eines guten Gitters hin:

Randkonforme Gitter: Die Gitterlinien, die Randpunkte verbinden, verlaufen auch innerhalb der Randlinie.

Automatische Erzeugung: So wenig menschliche Arbeit und Eingriffe in die Gittererzeugung wie möglich!

Robust: Stetige Abhängigkeit des erzeugten Gitters von der Geometrie des Gebietes. Oder: Wenn es ein zugehöriges kontinuierliches Problem gibt, dann soll dieses korrekt gestellt sein.

Sinnvoll: Keine allgemeine mathematische Voraussetzung! Wieder kann man sagen: Wenn die Gittererzeugung die Diskretisierung einer kontinuierlichen Transformation ist, dann läßt sich 'sinnvoll' definieren als:

Die Transformation soll ein-eindeutig sein.

Faltenlos: Das Gitter darf keine Falten haben, d.h.:

- Gitterlinien in die gleiche Richtung sollen sich nicht schneiden!
- Gitterlinien zu verschiedenen Richtungen sollen sich genau einmal schneiden!

2.3.1 Differentialgeometrische Grundlagen

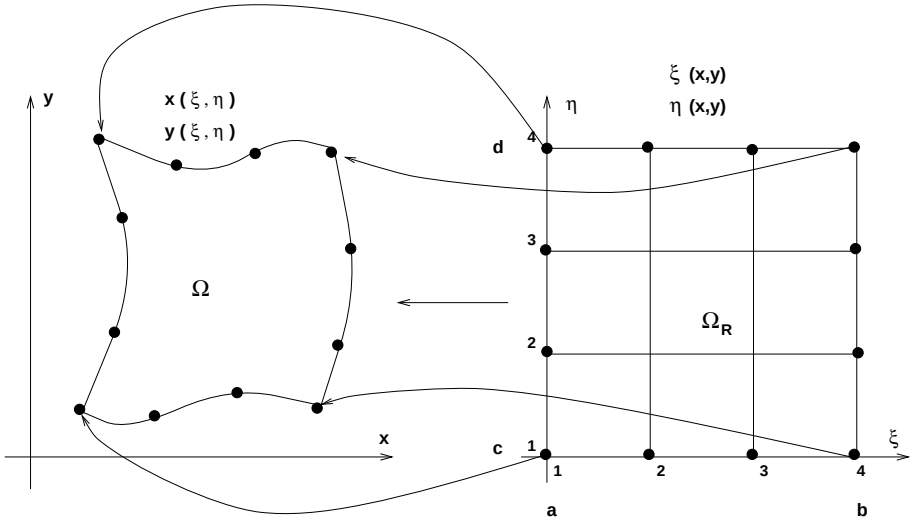


Abbildung 2.2: **Randkonforme Transformation von Ω_R auf Ω**

Die Grundidee der hier betrachteten Methoden geht von einer Transformation des physikalischen Gebietes auf ein Rechteck aus. Dabei werden die Randkurven des physikalischen

Gebietes eineindeutig auf die vier Seiten des Rechtecks transformiert. Das Gitter im physikalischen Gebiet wird dann durch die Inverse derselben Transformation, angewendet auf ein quadratisches Gitter im Rechteck, erzeugt. Die Verteilung der Gitterpunkte im physikalischen Gebiet hängt damit ausschließlich von der gewählten Transformation und der dafür festzulegenden Randpunktezuordnung ab. Es wird vorausgesetzt, daß die Abbildung ein-eindeutig ist. Außerdem soll der Rand mit abgebildet werden und als Rand erhalten bleiben. Das ergibt für solche Transformationsmethoden den Namen "boundary conforming", also etwa "Rand-erhaltend".

Die geometrischen Grundlagen solcher Transformationen bilden krummlinige Koordinatensysteme mit entsprechenden Basisvektoren, die Bildung der zugehörigen Differentialelemente und die zugehörige Analysis wie der Divergenz-Satz. Einzelheiten findet man in [78].

Seien (x, y) die kartesischen Koordinaten des physikalischen Gebietes, die gleichzeitig die abhängigen Koordinaten

$$(x(\xi,\eta), y(\xi,\eta)) \ , \quad \mathfrak{x} = \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.9}$$

der Abbildung $(\xi, \eta) \to (x, y) : \ \Omega_R \to \Omega$ sind. Seien (ξ, η) die kartesischen Koordinaten im Rechteck $[a, b] \times [c, d]$. Die krummlinigen Koordinaten sind dann die Kurven-Familien

$$\begin{array}{ll} (x(s,\eta), y(s,\eta)), & a \le s \le b \\ (x(\xi,t), y(\xi,t)), & c \le t \le d \end{array} \tag{2.10}$$

und wegen der Randerhaltungseigenschaften sind die vier Ränder gegeben als

$$\begin{array}{ll} (x(\xi,c), y(\xi,c)), & a \le \xi \le b, \\ (x(\xi,d), y(\xi,d)), & a \le \xi \le b, \\ (x(a,\eta), y(a,\eta)), & c \le \eta \le d, \\ (x(b,\eta), y(b,\eta)), & c \le \eta \le d. \end{array} \tag{2.11}$$

Üblicherweise setzt man die stetige Differenzierbarkeit der Transformationsabbildung voraus. Dann kann man die Ein-eindeutigkeits-Eigenschaft mit Hilfe der Jacobi-Determinante darstellen (oder überprüfen). Sie darf im gesamten Gebiet nicht verschwinden, sollte sogar aus Stabilitätsgründen gleichmäßig nach unten und oben beschränkt sein. Die Jacobi-Matrix ist ja gerade

$$M = \begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix} \tag{2.12}$$

und die Jacobi-Determinante dementsprechend

$$J = x_\xi y_\eta - x_\eta y_\xi. \tag{2.13}$$

Dadurch, daß die Jacobi-Determinante nirgends verschwindet, existiert überall lokal eine inverse Abbildung

$$\Omega \to \Omega_R : \quad (\xi(x,y), \eta(x,y)). \tag{2.14}$$

Sind $\tilde{M}$ und $\tilde{J}$ Jacobi-Matrix und -Determinante dieser inversen Transformation, dann ergibt die Kettenregel die folgenden Beziehungen:

$$M\tilde{M} = I, \quad \text{oder} \tag{2.15}$$

$$x_\xi = \eta_y/\tilde{J}, \ \ x_\eta = -\xi_y/\tilde{J}, \ \ y_\xi = -\eta_x/\tilde{J}, \ \ y_\eta = \xi_x/\tilde{J} \tag{2.16}$$

2.3.2 Transformationskoeffizienten und Differentialelemente

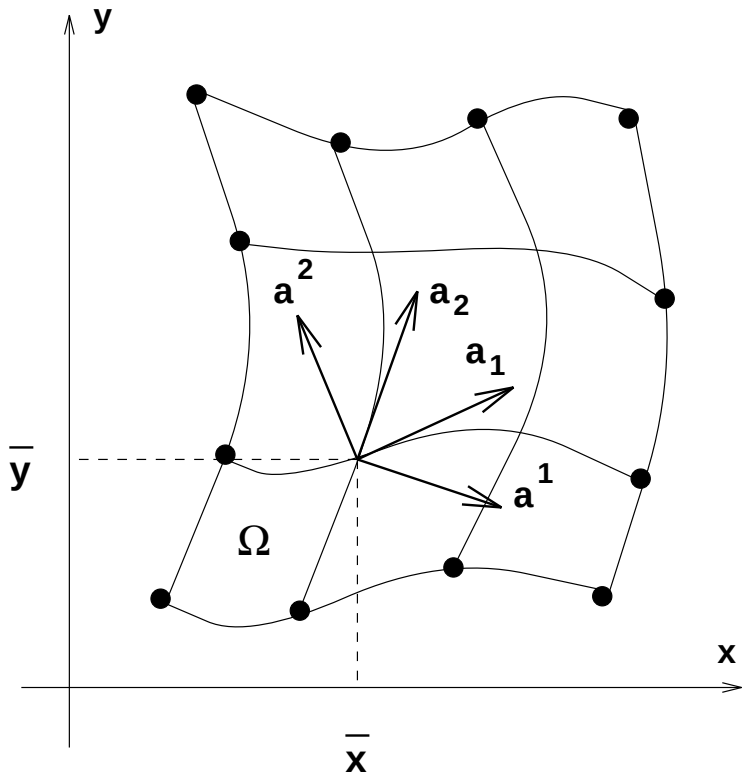


Abbildung 2.3: **Basisvektoren**

Wir wollen nur kurz die ko- und kontravarianten Basisvektoren, einige Differentialelemente und Folgerungen aus dem Divergenz-Satz angeben:

Die kovarianten Basisvektoren a_i sind die Tangenten an die Koordinatenlinien, auf denen ξ bzw. η laufen. Sie werden repräsentiert durch die Spalten der Funktionalmatrix

$$M = \left(\frac{\partial(x,y)}{\partial(\xi,\eta)} \right), \quad \text{also im } \mathbb{R}^2 \tag{2.17}$$

$$a_1 = \begin{pmatrix} x_\xi \\ y_\xi \end{pmatrix}, \quad a_2 = \begin{pmatrix} x_\eta \\ y_\eta \end{pmatrix} \tag{2.18}$$

Die kontravarianten Basisvektoren a^i sind entsprechend definiert. Sie sind die Normalen zu den Koordinatenlinien, auf denen ξ bzw. η konstant sind.

Als Transformationskoeffizienten benötigen wir die Elemente der ko- und kontravarianten Maßtensoren

$$g_{ij} := a_i \cdot a_j \quad \text{oder} \quad g := M^T M, \quad \text{und damit} \quad J = \sqrt{\det\, g} \tag{2.19}$$

g^{ij}, $\tilde{J}$ entsprechend. Die Jacobi-Determinante J, also die Wurzel aus der Determinante von g, gibt gerade die Zellfläche eines differentiellen Volumenelementes an. Im kontinuierlichen Fall handelt es sich um eine Abbildung

$$\Omega_R \to \Omega.$$

Für die numerische Gittererzeugung werden nur die Werte dieser Abbildung in allen Gitterpunkten festgelegt und daraus näherungsweise mit Differenzenausdrücken die Transformationskoeffizienten berechnet.

Als Differentialelemente mit dx und dy, aufgefaßt als 'kleine Stücke', bekommen wir:

- Differentialinkrement
$$d\mathfrak{x} = d\xi a_1 + d\eta a_2 \tag{2.20}$$
- Inkrement der Bogenlänge s:
$$(ds)^2 = |d\mathfrak{x}|^2 = g_{11}d\xi d\xi + 2g_{12}d\xi d\eta + g_{22}d\eta d\eta \tag{2.21}$$
- Inkrement der Bogenlänge auf der ξ-Koordinatenlinie
$$ds_\xi = \sqrt{g_{11}}d\xi, \tag{2.22}$$
ds_η entsprechend.
- Inkrement einer Zellfläche
$$dS_\xi = \sqrt{g}d\xi d\eta, \tag{2.23}$$

Aus dem Gauß'schen Integralsatz lassen sich Transformationsformeln für den Gradienten, die Rotation und den Laplace-Operator herleiten.

Berechnung der Transformationskoeffizienten

Es werden zunächst die transformierten Gitterwerte

$$x(\xi,\eta) \quad \text{und} \quad y(\xi,\eta) \tag{2.24}$$

berechnet, dann an allen Gitterpunkten des Rechteckgitter die Ableitungen

$$x_\xi,\ y_\xi,\ x_\eta,\ y_\eta,\ x_{\xi\xi},\ y_{\xi\xi},\ x_{\eta\eta},\ y_{\eta\eta} \tag{2.25}$$

mit Differenzenausdrücken zweiter Ordnung auf einem 5-Punkte-Stern. Aus ihnen können dann die Werte der Maßtensoren und der Laplace-Operatoren

$$g_{ij},\ g^{ij}, \quad \text{und} \quad P := \xi_{xx} + \xi_{yy},\ Q := \eta_{xx} + \eta_{yy}, \tag{2.26}$$

berechnet werden.

Wir haben hauptsächlich die folgenden Transformationen und ihre Erweiterungen benutzt:

- Erzeugung eines Startgitters mit algebraischen Methoden: Für die Gittererzeugung mit partiellen Differentialgleichungen benötigt man ein Startgitter, das mit Interpolationsmethoden leicht zu berechnen ist. Die Qualität dieser Gitter reicht aber meistens nicht aus. Auch die verbesserten Methoden von Gilding, [31], oder Knupp, [41], die sogenannte *intrinsic* algebraic grid-generation, unterliegen zu vielen Einschränkungen für allgemeine Gebiete.

- Iterative Verbesserung des Gitters durch harmonische Transformation des physikalischen Gebietes auf ein Rechteck, d.h. iterative Lösung eines quasi-linearen elliptischen Differentialgleichungssystems, s.u. Die linearisierte PDE bleibt elliptisch und muß nur in einem Rechteck gelöst werden.

- Einführung von Kontrollfunktionen zur Verbesserung der Gitterstruktur, d.h. Übergang von der homogenen Laplace- zur inhomogenen Poisson-Gleichung; die rechten Seiten stellen gerade die Kontrollfunktionen dar.

- Aufteilung des Gebietes in Blöcke – mit offensichtlichen Vorteilen sowohl für eine gleichmäßige Punkteverteilung als auch für die Parallelisierung.

Diese Punkte sind erweiterbar durch:

- Lokale Koordinatensysteme.

- Gittererzeugung bei mehrfach zusammenhängenden Gebieten wie etwa im Beispiel zur geometrischen Konfiguration in Kapitel 1 auf Seite 4.

- Adaptive Gittergenerierung: Während der iterativen Lösung der PDE werden laufend die Gradienten mitberechnet und die Gitterlinien in der Nähe stark variierender Gradienten konzentriert. Das führt zu höherem Aufwand, der durch schnellere Konvergenz und höhere Genauigkeit der Lösung ausgeglichen werden kann. Besonders interessant ist die adaptive Gittergenerierung bei instationären Problemen.

- Parabolische oder hyperbolische Systeme zur Gittergenerierung, z.B. bei unbeschränkten Gebieten.

Das erzeugte Gitter sollte die oben genannten Bedingungen erfüllen. Damit wird erreicht, daß ein Punkt im Rechteck genau einem Punkt im physikalischen Gebiet zugeordnet werden kann. Man muß aber beachten, daß diese Bedingungen für die Diskretisierung einer kontinuierlichen Transformation nicht erfüllt sein muß, auch wenn sie für die Original-Transformation gelten.

2.3.3 Zwei einfache Beispiele analytischer Transformationen

Für das Beispiel links in Abbildung 2.4 gilt:

$$\begin{aligned} x(\xi,\eta) &= \xi + \frac{1}{2}\eta - \frac{3}{2} \\ y(\xi,\eta) &= \frac{1}{2}\xi + \eta - \frac{3}{2} \end{aligned}$$

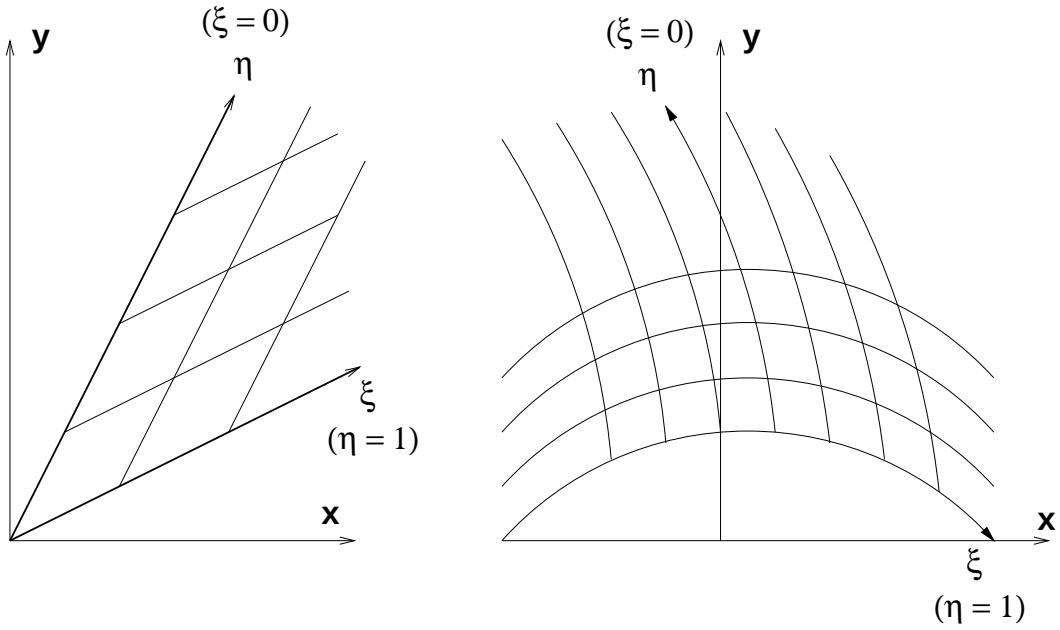


Abbildung 2.4: Linear und quadratisch krummlinige Koordinaten

sowie

$$\begin{aligned} \xi(x,y) &= \frac{4}{3}x - \frac{2}{3}y + 1 \\ \eta(x,y) &= -\frac{2}{3}x + \frac{4}{3}y + 1 \end{aligned}$$

Somit ergeben sich:

$$\underline{a}_1 = \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}, \underline{a}_2 = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix}, g_{ij} = \begin{pmatrix} \frac{5}{4} & 1 \\ 1 & \frac{5}{4} \end{pmatrix}, \sqrt{g} = \frac{3}{4}$$

$$\underline{a}^1 = \begin{pmatrix} \frac{4}{3} \\ -\frac{2}{3} \end{pmatrix}, \underline{a}^2 = \begin{pmatrix} -\frac{2}{3} \\ \frac{4}{3} \end{pmatrix}, g^{ij} = \begin{pmatrix} \frac{20}{9} & -\frac{16}{9} \\ -\frac{16}{9} & \frac{20}{9} \end{pmatrix}, \sqrt{det(g^{ij})} = \frac{4}{3}$$

Für die quadratischen krummlinigen Koordinaten (Abbildung 2.4 rechts) gilt:

$$\begin{aligned} x(\xi,\eta) &= \xi - \frac{1}{16}\eta^2 + \frac{1}{8}\eta + \frac{15}{16} \\ y(\xi,\eta) &= -\frac{2}{25}\xi^2 + \eta + 1 \end{aligned}$$

Somit ergeben sich:

$$\underline{a}_1 = \begin{pmatrix} 1 \\ -\frac{4}{25}\xi \end{pmatrix}, \underline{a}_2 = \begin{pmatrix} -\frac{1}{8}\eta + \frac{1}{8} \\ 1 \end{pmatrix}, g_{ij} = \begin{pmatrix} \frac{16}{625}\xi^2 + 1 & -\frac{4}{25}\xi - \frac{1}{8}(\eta - 1) \\ -\frac{4}{25}\xi - \frac{1}{8}(\eta - 1) & \frac{1}{64}(\eta - 1)^2 + 1 \end{pmatrix}$$

$$\sqrt{g} = 1 - \tfrac{1}{50}\xi(\eta - 1)$$

$$\underline{a}^1 = \frac{1}{\sqrt{g}} \begin{pmatrix} 1 \\ \frac{1}{8}\eta - \frac{1}{8} \end{pmatrix}, \underline{a}^2 = \frac{1}{\sqrt{g}} \begin{pmatrix} \frac{4}{25}\xi \\ 1 \end{pmatrix},$$
$$g^{ij} = \frac{1}{g} \begin{pmatrix} \frac{1}{64}(\eta-1)^2 + 1 & \frac{4}{25}\xi + \frac{1}{8}(\eta-1) \\ \frac{4}{25}\xi + \frac{1}{8}(\eta-1) & \frac{16}{625}\xi^2 + 1 \end{pmatrix}$$

Es sei jedoch an dieser Stelle nochmals darauf hingewiesen, daß die Transformation in der Praxis nicht analytisch gegeben ist, sondern alle Werte mit Differenzenausdrücken gebildet werden.

2.3.4 Harmonische Transformation

Die einfachste Methode, ein glattes Gitter mit Differentialgleichungen zu erzeugen, ist die Anwendung der harmonischen Transformation

$$\begin{aligned} \xi_{xx} + \xi_{yy} &= 0 && \text{in } \Omega, \\ \eta_{xx} + \eta_{yy} &= 0 && \\ \xi &= G && \text{auf } \partial\Omega. \\ \eta &= H && \end{aligned}$$

Diese Abbildung $\Omega \to \Omega_R$ erfüllt die gewünschten Bedingungen:

(1) Alle Punkte von Ω werden in Ω_R abgebildet.

(2) Die Gradienten $\nabla\xi$ und $\nabla\eta$ verschwinden nirgends in Ω.

(3) Die Jacobi-Determinante der Abbildung

$$\tilde{J} = \xi_x \eta_y - \xi_y \eta_x$$

verschwindet nirgends in Ω.

Lösen wollen wir aber die inverse Abbildung auf dem Rechteck Ω_R. Das führt zu dem nichtlinearen System

$$\begin{aligned} g_{22}\, \mathfrak{x}_{\xi\xi} + g_{11}\, \mathfrak{x}_{\eta\eta} - 2g_{12}\, \mathfrak{x}_{\xi\eta} = 0 \qquad & \text{in} \quad \Omega_R \\ \mathfrak{x} = \begin{pmatrix} \tilde{G} \\ \tilde{H} \end{pmatrix} \qquad & \text{auf} \quad \partial\Omega_R \end{aligned}$$

g_{ij} sind die Transformationskoeffizienten, die im diskretisierten Fall für jeden Gitterpunkt berechnet werden. Ist Ω_R ein Rechteck, so sind

$$g_{11} = x_\xi^2 + y_\xi^2, \;\; g_{22} = x_\eta^2 + y_\eta^2, \;\; g_{12} = x_\xi x_\eta + y_\xi y_\eta$$

In dem entstehenden Gitter kann die Verteilung der Gitterpunkte nicht beeinflußt werden. So wird in der Nähe konvexer Ränder z.B.

$$\eta_{xx} > 0 \;\Rightarrow\; \eta_{yy} < 0$$

gelten. Das bedeutet, daß dort die Gitterlinien enger zusammen, und entsprechend bei konkaven Rändern weiter auseinander liegen werden, mit der Gefahr der Faltenbildung des Gitters aufgrund der Diskretisierung und der Fehlerverstärkung in diesem Bereich bei Lösung der PDE.

2.3.5 Kontrollfunktionen

Um den Verlauf der Gitterlinien beeinflussen zu können, lösen wir statt der Laplace- die Poisson-Gleichung

$$\begin{aligned} \xi_{xx} + \xi_{yy} &= f^1 \\ \eta_{xx} + \eta_{yy} &= f^2 \end{aligned} \qquad in\,\Omega,$$

Die Funktionen f^i nennt man Kontrollfunktionen. Das System für die inverse Abbildung ist dann

$$g_{22}\,\mathfrak{x}_{\xi\xi} + g_{11}\,\mathfrak{x}_{\eta\eta} - 2g_{12}\,\mathfrak{x}_{\xi\eta} + g(P\mathfrak{x}_{\xi} + Q\mathfrak{x}_{\eta}) = 0 \quad \text{in} \quad \Omega_R$$

mit $g := \det(g_{ij})$, $P := \xi_{xx} + \xi_{yy}$ und $Q := \eta_{xx} + \eta_{yy}$.
Ein einfacheres System bekommt man für

$$\begin{aligned} \xi_{xx} + \xi_{yy} &= g^{11} f^1 \\ \eta_{xx} + \eta_{yy} &= g^{22} f^2 \end{aligned} \qquad in\,\Omega,$$

und zwar:

$$g_{22}\,(\,\mathfrak{x}_{\xi\xi} + P\mathfrak{x}_{\xi}) + g_{11}\,(\,\mathfrak{x}_{\eta\eta} + Q\mathfrak{x}_{\eta}) - 2g_{12}\,\mathfrak{x}_{\xi\eta} = 0 \quad \text{in} \quad \Omega_R$$

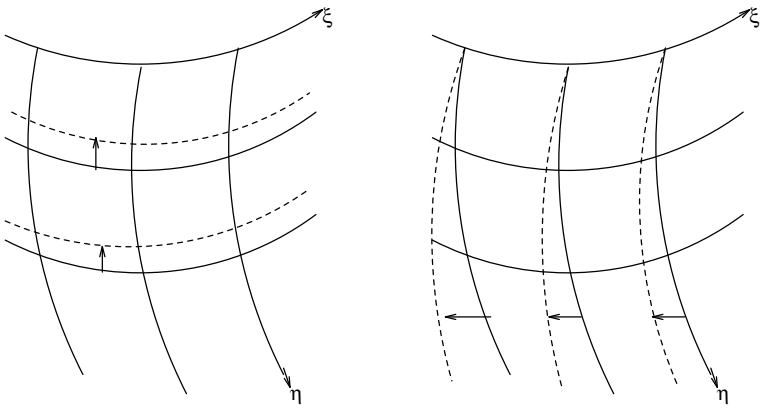


Abbildung 2.5: **Wirkung negativer Kontrollfunktionen**

Die Möglichkeiten sind aber weit vielfältiger, wie diese Aufstellung zeigen soll:

(1) Konstante Kontrollfunktionen (siehe Abbildung 2.5)

(2) Übertragung der Randpunkteverteilung ins Innere

(3) Übertragung einer vorgegebenen Linienverteilungsfunktion

(4) Konzentration von Gitterlinien in der Nähe vorgegebener Linien oder Punkte (siehe Abbildung 2.6), z.B.

$$\begin{aligned} f^1 \;:=\; & -a\,\operatorname{sign}(\xi - \bar{\xi})\,\exp(-c|\xi - \bar{\xi}|) \\ & -b\,\operatorname{sign}(\xi - \bar{\xi})\,\exp\left(-d\sqrt{(\xi - \bar{\xi}) + (\eta - \bar{\eta})}\right) \end{aligned}$$

(5) Diese vier Möglichkeiten können in einzelnen Punkten oder Linien additiv überlagert eingesetzt werden, sodaß eine differenzierte Einflußmöglichkeit auf das resultierende Gitter im physikalischen Raum entsteht.

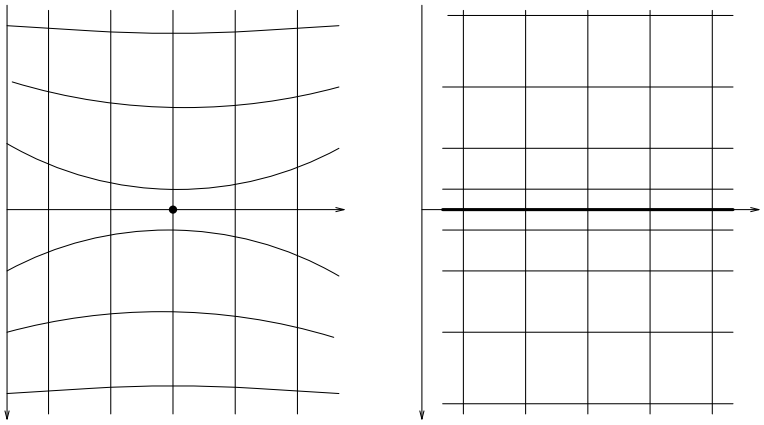


Abbildung 2.6: **Konzentration von Gitterlinien um Punkt und Linie**

Die kontinuierlichen Transformationen werden auf dem Quadratgitter des Rechtecks mit Hilfe des zugehörigen 5-Punkte-Sterns angenähert berechnet. Ihre Ausführung und die anschließende Lösung der partiellen Differentialgleichungen erfolgen in sieben Schritten:

(1) Aufteilung des Randes des physikalischen Gebietes in vier Seiten durch Definition von vier 'Eckpunkten'.

(2) Festlegung der Randpunktzahlen $imax$ und $jmax$ und damit der Gesamtpunktezahl $imax \cdot jmax$ und der Zuordnung der Randpunkte (diskret oder mittels Funktionen).

(3) Erzeugung eines Startgitters mit algebraischen Methoden. Diskrete Berechnung der Transformations-Koeffizienten in jedem inneren Gitterpunkt.

(4) Erzeugung des Gitters im physikalischen Gebiet mit Hilfe der inversen Transformationen, d.h. durch iterative Lösung des quasi-linearen elliptischen PDE-Systems. Dabei werden nach jedem Schritt die Transformations-Koeffizienten in jedem inneren Gitterpunkt diskret berechnet.

(5) Transformation der zu lösenden PDE mit diesen Koeffizienten.

(6) Lösung der transformierten PDE auf dem Rechteck Ω_R mit Quadratgitter zur Schrittweite $h = 1$.

(7) Rücktransformation der Lösung $\tilde{u}(\xi, \eta) \rightarrow u(x, y)$.

2.3.6 Parallelisierung durch Gebietszerlegung

Dies ist die natürlichste Form der Parallelisierung, die zusätzlich den Vorteil hat, daß als anschließendes Lösungsverfahren für die PDE am günstigsten eine Gebietszerlegungsmethode

mit denselben Teilgebieten gewählt werden kann. Das erhöht die Effizienz des Gesamtverfahrens, da es den größten Kommunikationsanteil – die Ausgabe des erzeugten Punktegitters aller Teilgebiete durch das Prozessorennetzwerk zurück zum Host-Rechner und die eventuell notwendige Umnumerierung der Gitterpunkte – vermeidet.

Für die Aufteilung des Gitters in Blöcke stellen wir folgende Ziele auf:

- Wenige Punkte auf inneren Blockrändern (Kommunikation), oder

$$\frac{\#\text{ Gitterpunkte}}{\#\text{ Blockschnittpunkte}} \quad \text{groß}$$

- # Gitterpunkte/Block etwa gleich für alle Blöcke
- Rechtecke möglichst quadratisch
- Keine Konvergenz-störende Blockaufteilung, z.B. innere Blöcke
- Anfangsblockaufteilung günstig für automatische Blockverkleinerung

Einige dieser Ziele können nur durch eine geschickte Aufteilung des Gebietes in Blöcke bzw. Teilgebiete durch den Benutzer selbst erreicht werden, andere werden durch die automatische Blockverfeinerung des Algorithmus erledigt:

(1) **Anfangszerlegung** durch den Benutzer
Festlegung, ob innere Schnittlinien fest oder frei
Anzahl Gitterpunkte, Prozessoren und Architektur

Automatisch:

(2) Prüfung, ob für weitere Zerlegung genügend Prozessoren vorhanden sind.

(3) Prüfung, ob vertikale oder horizontale Zerlegung günstiger.

(4) Prüfung, ob Zerlegung sinnvoll (bez. Lastausgleich, Speed-up).

(5) Zurück zu Punkt 2.

(6) Zerlegung durchführen, Variablen umsetzen, Prozessoren mit Daten beschicken.

Wir wollen diese Prinzipien an einem Beispiel verdeutlichen. Dazu nehmen wir das schon bekannte Gebiet, das wir in naheliegender Weise in drei Blöcke aufteilen. Eine automatische Verfeinerung in 6 und 12 Blöcke ist angegeben.

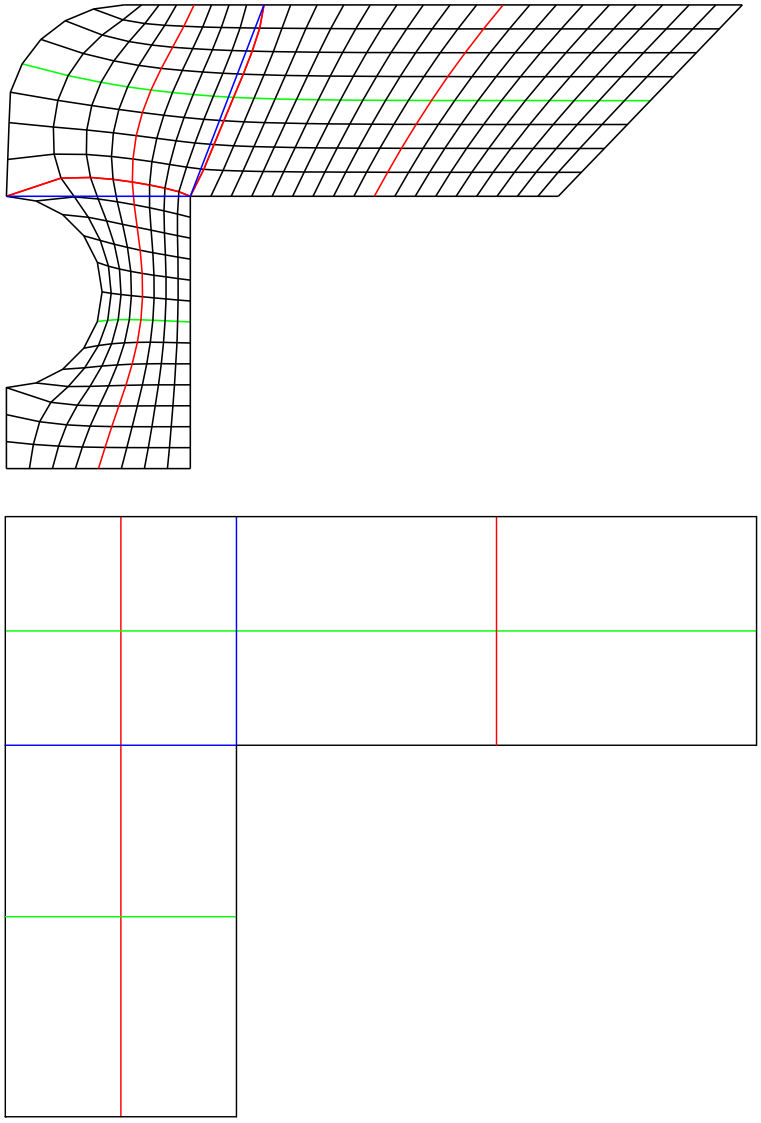


Abbildung 2.7: **Gitter in zwölf Blöcken**

2.3.7 Prozessorennetzwerk

Transputer sind voneinander unabhängige Prozessoren, die über einen eigenen Speicherplatz von 1, 4 oder 16 MByte verfügen, und 4 I/O-Kanäle besitzen. Sie können also mit bis zu 4 anderen Transputern (Nachbarn) verknüpft werden, und sie verkehren mit diesen über Nachrichten (asynchrone message-passing-Architektur).

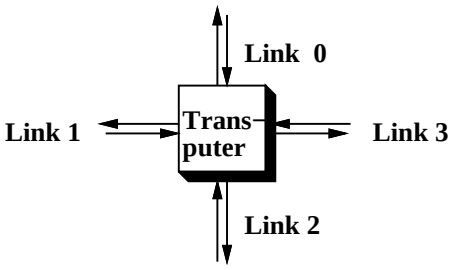


Abbildung 2.8: **Transputer**

Forderungen an das Netzwerk:

Unser Ziel war, ein Netzwerk zu benutzen, das den notwendigen Datenaustausch auf möglichst kurzen Wegen erledigt, andererseits aber skalierbar bleibt, d.h. daß es bei Vergrößerung der Anzahl Prozessoren nicht seine Struktur ändern muß wegen der Beschränkung auf 4 Links. Diese Forderung führt zum de Bruijn-Netz, das folgendermaßen definiert ist:
Die Prozessorenzahl sei

$$N = 2^k$$

Die Prozessoren werden durchnumeriert wie

$$0, 1, \ldots, N-1.$$

Der Prozessor mit der Binärzahldarstellung

$$P = \alpha_1\, \alpha_2\, \ldots\, \alpha_k$$

besitzt Kanten zu den vier Prozessoren

$$\begin{aligned}
&\alpha_2\, \ldots\, \alpha_k\, \alpha_1\\
&\alpha_2\, \ldots\, \alpha_k\, \bar{\alpha}_1\\
&\alpha_k\, \alpha_1\, \ldots\, \alpha_{k-1}\\
&\bar{\alpha}_k\, \alpha_1\, \ldots\, \alpha_{k-1}
\end{aligned}$$

Dabei ist $\bar{\alpha}_i$ das zu α_i komplementierte Bit. k ist die Dimension des Netzes, sein Grad ist immer gleich 4. Dabei gibt es allerdings überflüssige Doppelverbindungen, und P_0 und P_{N-1} sind immer doppelt mit sich selbst verbunden. Das kann ausgenutzt werden zur Einbindung eines Host-Prozessors.

Für $k = 3$ bekommen wir die Kanten

von	000	zu:	000	001	000	100
von	001	zu:	010	011	100	000
von	010	zu:	100	101	001	101
von	011	zu:	110	111	101	001
von	100	zu:	001	000	010	110
von	101	zu:	011	010	110	010
von	110	zu:	101	100	011	111
von	111	zu:	111	110	111	011

oder schön symmetrisch im Bild

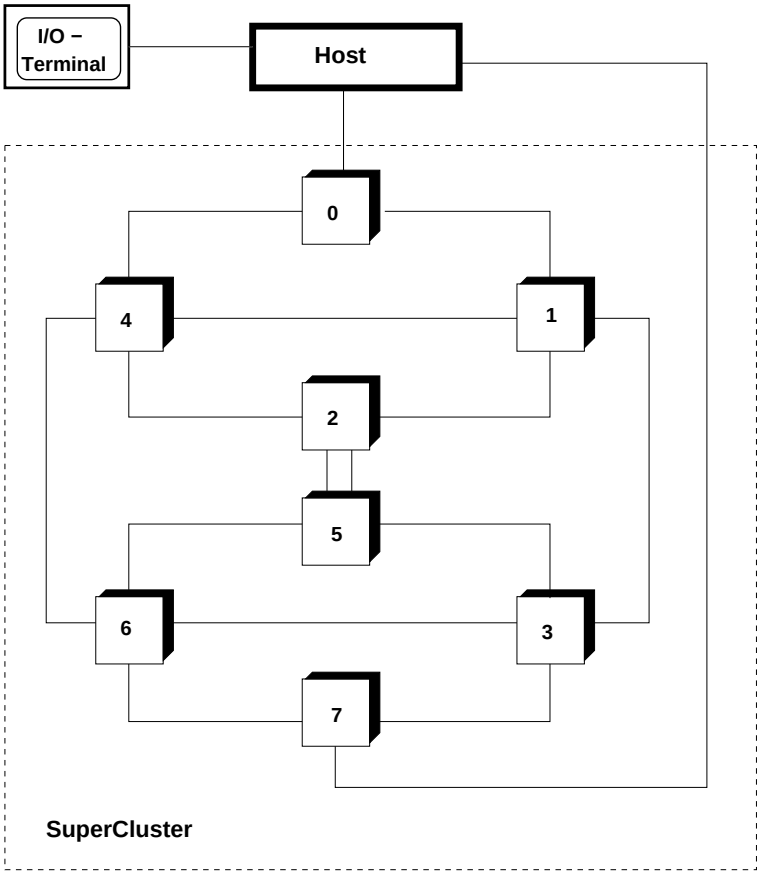


Abbildung 2.9: **De Bruijn-Netz**

Andere in Frage kommende Prozessorennetze wären gewesen:

- **Zweidimensionales Gitter**: Abbildung der Blockstruktur mit weniger als vier Nachbarblöcken problematisch.
- **Ring**: zu lange Wege.
- **Hypercube**: Nicht skalierbar, mit Transputern nur bis zur Dimension 4, also bis zu $2^4 = 16$ Transputern zu verwirklichen.
- **Cube connected Cycles**: Nur Grad 3.

2.3.8 Anwendungen

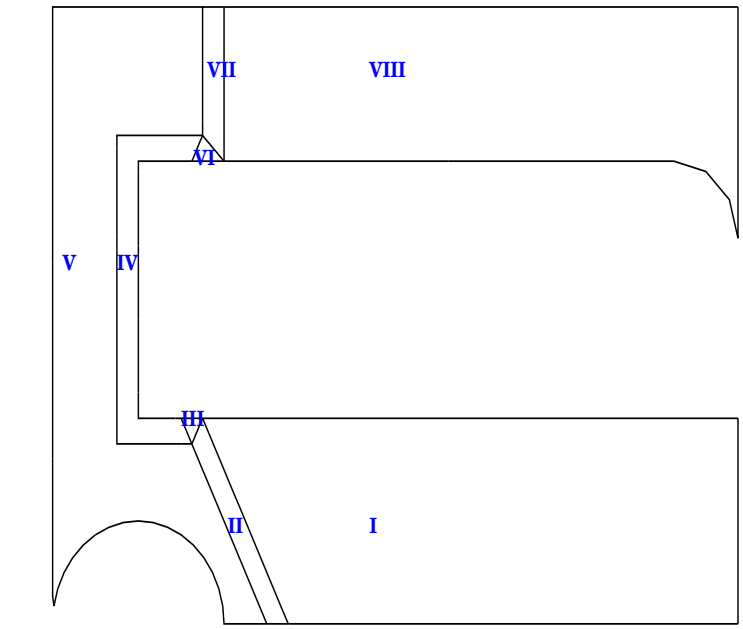
Als Anwendung wollen wir ein Wärmeleitungsproblem aus einer praktischen Anwendung ansehen.

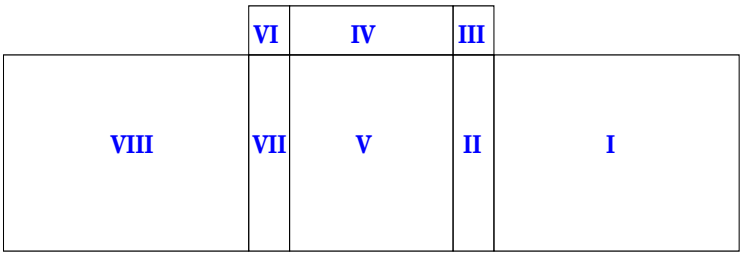
$$\begin{aligned} \frac{\partial}{\partial x}\left(K_1\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_2\frac{\partial u}{\partial y}\right)+Q &= 0 \quad \text{in } \Omega \\ u &= g \quad \text{auf } \partial\Omega \end{aligned}$$

mit

$$\begin{aligned} K_1 &= \text{ Wärmeleitzahl in } x\text{-Richtung} \\ K_2 &= \text{ Wärmeleitzahl in } y\text{-Richtung} \end{aligned}$$

Die partielle Differentialgleichungen hat hier unstetige Koeffizienten – die Material-abhängigen Wärmeleitzahlen. Dementsprechend bekommt man acht Blöcke, die zusammen mit Ihrer Anordnung für die Transformation zur Gittererzeugung auf der Abbildung 15 dargestellt sind. Abbildung 16 zeigt das entstandene Gitter mit etwa 3000 Punkten. Bei diesen Problemen werden Temperaturen berechnet. Die Ingenieure interessieren sich aber mehr für Spannungen, die aus den Temperaturen unter anderem durch Differenzenbildung auf demselben Gitter berechnet werden. Da sich dabei der Fehler erheblich verstärken kann, ist hier auch für Ingenieurprobleme eine hohe relative Genauigkeit zu fordern. Das führt schnell zu Problemen mit einigen hunderttausend Unbekannten – es geht schließlich nicht nur um die Zeichengenauigkeit. Für solche 'real world'-Probleme wird der Einsatz eines großen Parallelrechners erst wirklich notwendig.


VII
VIII
VI
V
IV
III
II
I


VI
IV
III
VIII
VII
V
II
I

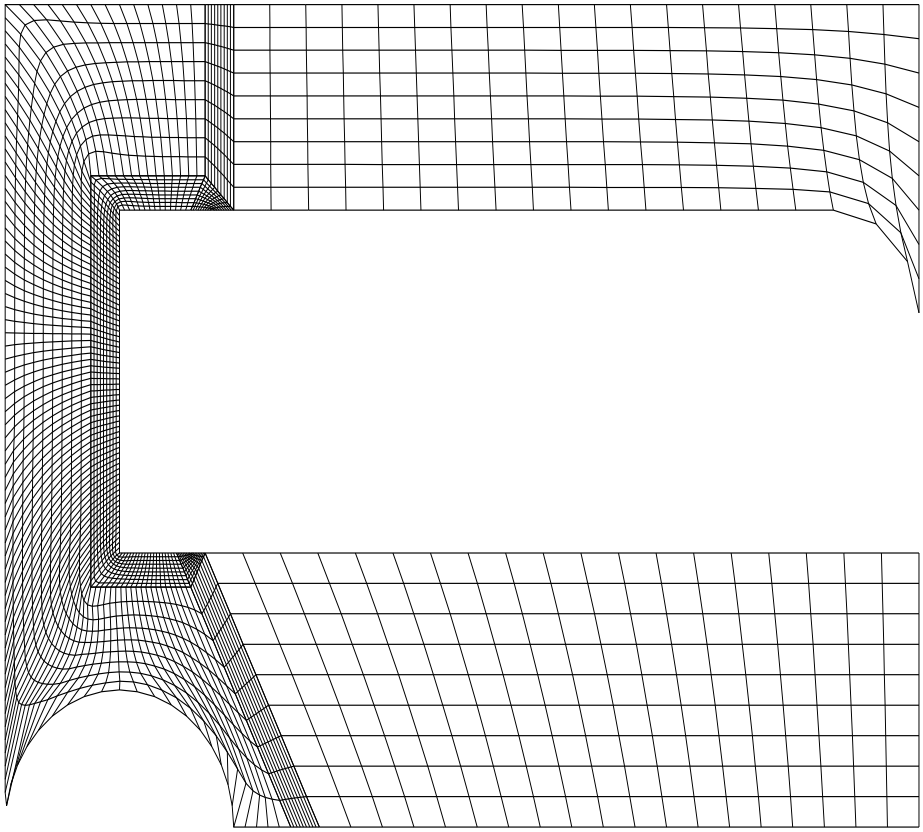
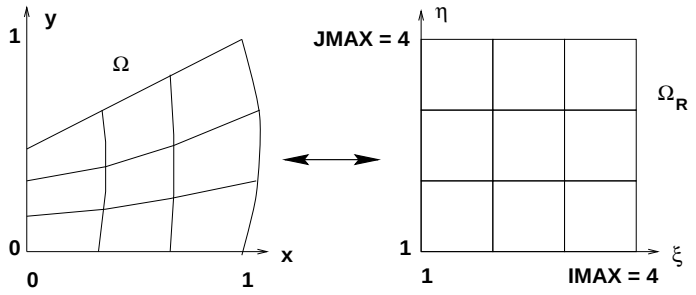


Abbildung 2.10: **Hochofen – Blockaufteilung und Gitter**

Beispiel 2.3.1 Ein einfaches Beispiel



(1) Festlegung der Punkteanzahlen IMAX und JMAX

(2) Konfiguration des physikalischen Gebietes:
Zuordnung der Randpunkte

(3) Startnäherung durch transfinite Interpolation

(4) Iterativ:

(a) Berechnung der notwendigen Transformationskoeffizienten.

(b) Lösung der Transformations-PDEs

(1) **Festlegung der gewünschten Punktzahl**

Für das zweidimensionale Gebiet mit 4×4 Punkten wird IMAX $= 4$ und JMAX $= 4$ eingegeben. Die Numerierung von ξ und η läuft von $1 \ldots$ IMAX bzw. $1 \ldots$ JMAX.

(2) **Aufstellung einer Konfiguration für das transformierte Gebiet Ω**

Zunächst erfolgt die Eingabe der Randpunkte, in diesem Beispiel als Funktionen:
Unterer/oberer Rand:

$$\begin{pmatrix} x(\xi,1) \\ y(\xi,1) \end{pmatrix} = \begin{pmatrix} K_1 \\ 0 \end{pmatrix} \text{ und } \begin{pmatrix} x(\xi,\ \text{JMAX}) \\ y(\xi,\ \text{JMAX}) \end{pmatrix} = \begin{pmatrix} K_1 \\ \frac{1}{2} + \frac{1}{2}K_1 \end{pmatrix}$$
$$\text{mit} \qquad (2.27)$$
$$K_1 = \frac{\xi - 1}{\text{IMAX} - 1}$$

Linker/rechter Rand:

$$\begin{pmatrix} x(1,\eta) \\ y(1,\eta) \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{2}K_2 \end{pmatrix} \text{ und } \begin{pmatrix} x(\ \text{IMAX},\eta) \\ y(\ \text{IMAX},\eta) \end{pmatrix} = \begin{pmatrix} 1 + 0.4\,K_2(1 - K_2) \\ K_2 \end{pmatrix}$$
$$\text{mit} \qquad (2.28)$$
$$K_2 = \frac{\eta - 1}{\text{JMAX} - 1}$$

In diesem Fall ergibt sich also $K_1 = \frac{1}{3}(\xi - 1), K_2 = \frac{1}{3}(\eta - 1)$.

Somit werden folgende Randpunkte vom Programm berechnet, hier angenähert ausgegeben:

unterer Rand:

(ξ, η)	$(1,1)$	$(2,1)$	$(3,1)$	$(4,1)$
x	0.0	0.333333	0.666667	1.0
y	0.0	0.0	0.0	0.0

oberer Rand:

(ξ, η)	$(1,4)$	$(2,4)$	$(3,4)$	$(4,4)$
x	0.0	0.333333	0.666667	1.0
y	0.5	0.666667	0.833333	1.0

linker/rechter Rand:

(ξ, η)	$(1,2)$	$(1,3)$	$(4,2)$	$(4,3)$
x	0.0	0.0	1.08889	1.08889
y	0.166667	0.333333	0.333333	0.666667

Die Transformationskoeffizienten (nach transfiniter Interpolation)

	$(\xi = 2, \eta = 2)$	$(\xi = 2, \eta = 3)$	$(\xi = 3, \eta = 2)$	$(\xi = 3, \eta = 3)$
g_{11}	0.134829	0.144088	0.134829	0.144088
g_{22}	0.049602	0.049602	0.078039	0.078038
g_{12}	0.017723	0.019314	0.026186	0.020109

(3) **Generierung der Gitterpunkte**

Das Ziel ist es, mit dem oben beschriebenen Verfahren zu jedem Gitterpunkt $(\xi, \eta), \xi = 2, \ldots,$ IMAX $-1, \eta = 2, \ldots,$ JMAX-1 im Rechenraum Ω_R die Koordinaten des ihm entsprechenden Punkts im physikalischen Gebiet zu bestimmen. Die in Punkt 2 berechneten Randpunkte werden dabei (in diesem Beispiel) nicht mehr verändert.

Die Generierung soll mit einer elliptischen PDE ohne Kontrollfunktionen (Laplace-System) erfolgen, um eine gleichmäßige Verteilung der Gitterpunkte ohne Sonderbedingungen zu erreichen. Das Generierungssystem

$$\begin{aligned} \xi_{xx} + \xi_{yy} &= 0 \\ \eta_{xx} + \eta_{yy} &= 0 \end{aligned}$$

wird zunächst transformiert, um die Generierung im Rechenraum auf rechteckigem Gebiet Ω_R mit quadratischem Gitter und Einheitsschrittweite ausführen zu können:

$$g_{22}\mathfrak{x}_{\xi\xi} + g_{11}\mathfrak{x}_{\eta\eta} - 2g_{12}\mathfrak{x}_{\xi\eta} = 0 \text{ mit } \mathfrak{x} = \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.29}$$

Zur Lösung dieser PDE werden die Randbedingungen (2.27) und (2.28) benutzt.

Dann erfolgt die Berechnung der Startwerte für die inneren Punkte mit transfiniter linearer Lagrange-Interpolation, das heißt, einer vertikalen Interpolation schließt sich eine horizontale Korrektur an.

	Spalte $\xi = 2$		Spalte $\xi = 3$		
(ξ, η)	$(2,2)$	$(2,3)$	$(3,2)$	$(3,3)$	
x	0.362963	0.362963	0.725926	0.725926	Interpolation
y	0.222222	0.444444	0.277778	0.555556	

	Spalte $\xi = 2$		Spalte $\xi = 3$		
(ξ, η)	$(2,2)$	$(2,3)$	$(3,2)$	$(3,3)$	
x	0.338236	0.339523	0.689219	0.692987	DGL
y	0.212355	0.266214	0.431785	0.543009	

(4) **Abschließende Berechnung der Transformationskoeffizienten**

Die Transformationskoeffizienten an jedem inneren Punkt können nun (abermals mit Differenzenausdrücken) aus den je 16 $x-$ und $y-$Werten endgültig bestimmt werden:

	Spalte $\xi = 2$		Spalte $\xi = 3$	
(ξ, η)	$(2,2)$	$(2,3)$	$(3,2)$	$(3,3)$
g_{11}	0.121231	0.131047	0.144527	0.154181
g_{22}	0.0466191	0.0516054	0.0738875	0.0805329
g_{12}	0.011812	0.0229642	0.0213615	0.0290776

2.4 Algebraische Gittererzeugung II: Intrinsische Methoden

Diese Gittererzeugung ist durch algebraische Formeln definiert, die die Randfunktionen mit den Ecken einschließt, aber keinen Gebrauch von Ansatzfunktionen wie die Lagrange-Interpolation macht.

2.4.1 Randerhaltender Ansatz AII (nach KNUPP)

Komplexe Randdarstellung ($i := \sqrt{-1}$)

$$\begin{array}{rcl} C_1 : z_1(\xi) & := & x_1(\xi) + iy_1(\xi) \\ C_2 : z_2(\eta) & := & x_2(\eta) + iy_2(\eta) \\ C_3 : z_3(\xi) & := & x_3(\xi) + iy_3(\xi) \\ C_4 : z_4(\eta) & := & x_4(\eta) + iy_4(\eta) \end{array} \tag{2.30}$$

Auch hier genügt für die praktischen Rechnungen die Vorgabe genügend vieler diskreter Randwerte.

Das Gitter ist beschrieben durch

$$z(\xi, \eta) = x(\xi, \eta) + iy(\xi, \eta) \tag{2.31}$$

mit dem speziellen Ansatz:

$$z(\xi, \eta) = f(\eta)z_1(\xi) + g(\eta)z_3(\xi), \quad f, g : [0, 1] \rightarrow \mathbb{C}. \tag{2.32}$$

Die Randwerte auf C_2 und C_4 führen auf ein lineares Gleichungssystem für f und g:

$$(z(0, \eta) =) f(\eta)z_1(0) + g(\eta)z_3(0) = z_2(\eta)$$

$$(z(1, \eta) =) f(\eta)z_1(1) + g(\eta)z_3(1) = z_4(\eta).$$

Mit der Determinante

$$D_{II} = z_1(0)z_3(1) - z_1(1)z_3(0)$$

ergibt sich die Lösung:

$$\begin{array}{l} f(\eta) = \left| \begin{array}{cc} z_2(\eta) & z_3(0) \\ z_4(\eta) & z_3(1) \end{array} \right| / D_{II} \\ g(\eta) = \left| \begin{array}{cc} z_1(0) & z_2(\eta) \\ z_1(1) & z_4(\eta) \end{array} \right| / D_{II} \end{array} \tag{2.33}$$

Einsetzen in (2.32) liefert das Gitter.

$$z(\xi, \eta) = \frac{1}{D_{II}} \{ z_3(1)z_2(\eta)z_1(\xi) - z_3(0)z_4(\eta)z_1(\xi) + z_1(0)z_4(\eta)z_3(\xi) - z_1(1)z_2(\eta)z_3(\xi) \} \tag{2.34}$$

Für das einfache Beispiel 2.2.1 erhalten wir die Werte:

$$D_{II} = \underbrace{0}_{z_1(0)} - \underbrace{(8)}_{z_1(1)} \underbrace{6i}_{z_3(0)} = -48i$$

und mit (2.32) und (2.33):

$$\begin{aligned} -48i\, z(\frac{1}{2},\frac{1}{2}) &= \begin{vmatrix} z_2(\frac{1}{2}) & z_3(0) \\ z_4(\frac{1}{2}) & z_3(1) \end{vmatrix} z_1(\frac{1}{2}) \\ &+ \begin{vmatrix} z_1(0) & z_2(\frac{1}{2}) \\ z_1(1) & z_4(\frac{1}{2}) \end{vmatrix} z_3(\frac{1}{2}) \end{aligned}$$

Mit dem Taschenrechner erhält man folgende Ergebnisse:

$$-48i \left(x(\frac{1}{2},\frac{1}{2}) + iy(\frac{1}{2},\frac{1}{2}) \right) = 164 - 154i$$

$$\Rightarrow x(\frac{1}{2},\frac{1}{2}) = \frac{-154}{-48} = 3.21$$

$$y(\frac{1}{2},\frac{1}{2}) = \frac{164}{48} = 3.42$$

Bei Verwendung von (2.34) gelangt man zu denselben Ergebnissen.

2.4.2 Randerhaltender Ansatz A III (nach KNUPP)

Diese Alternative beginnt mit einem Separationsansatz, der A II ähnelt und auch ähnliche Gitter erzeugt. Der Hauptgrund für die zusätzliche Einführung dieser Alternative ist der, daß sie leicht auf dreidimensionale Oberflächen ausgeweitet werden kann.

Separationsansatz

$$\begin{aligned} x(\xi,\eta) &= a_1(\eta)x_1(\xi) + b_1(\eta)y_1(\xi) + a_3(\eta)x_3(\xi) + b_3(\eta)y_3(\xi) \\ y(\xi,\eta) &= a_1(\eta)y_1(\xi) + b_1(\eta)x_1(\xi) + a_3(\eta)y_3(\xi) + b_3(\eta)x_3(\xi) \end{aligned} \tag{2.35}$$

Dieser Ansatz unterscheidet sich an zwei Stellen von dem reellen Analogon von A II und zwar bei den Vorzeichen von b_1 und b_3. Wie bei A II erhalten wir ein lineares 4×4-Gleichungssystem mit den Unbekannten a_1, b_1, a_3, b_3, der Determinanten

$$D_{III} = \begin{vmatrix} x_1(0) & y_1(0) & x_3(0) & y_3(0) \\ y_1(0) & x_1(0) & y_3(0) & x_3(0) \\ x_1(1) & y_1(1) & x_3(1) & y_3(1) \\ y_1(1) & x_1(1) & y_3(1) & x_3(1) \end{vmatrix} \tag{2.36}$$

und der Lösung

$$a_1(\eta) = \begin{vmatrix} x_2(\eta) & y_1(0) & x_3(0) & y_3(0) \\ y_2(\eta) & x_1(0) & y_3(0) & x_3(0) \\ x_4(\eta) & y_1(1) & x_3(1) & y_3(1) \\ y_4(\eta) & x_1(1) & y_3(1) & x_3(1) \end{vmatrix} / D_{III},$$

$b_1(\eta), a_3(\eta), b_3(\eta)$ entsprechend nach der Cramerschen Regel.

Für unser einfaches Beispiel 2.2.1 erhalten wir die folgenden Werte:

$$D_{III} = \begin{vmatrix} 0 & 0 & 0 & 6 \\ 0 & 0 & 6 & 0 \\ 8 & 0 & 8 & 6 \\ 0 & 8 & 6 & 8 \end{vmatrix} = -2304$$

$$a_1(\frac{1}{2}) = \begin{vmatrix} -2 & 0 & 0 & 6 \\ 2 & 0 & 6 & 0 \\ 8 & 0 & 8 & 6 \\ 3 & 8 & 6 & 8 \end{vmatrix} / D_{III} = +8 \cdot (-264)/(-2304) = 0.92$$

$$b_1(\frac{1}{2}) = \begin{vmatrix} 0 & -2 & 0 & 6 \\ 0 & 2 & 6 & 0 \\ 8 & 8 & 8 & 6 \\ 0 & 3 & 6 & 8 \end{vmatrix} / D_{III} = 8 \cdot (-132)/(-2304) = 0.46$$

$$a_3(\frac{1}{2}) = \begin{vmatrix} 0 & 0 & -2 & 6 \\ 0 & 0 & 2 & 0 \\ 8 & 0 & 8 & 6 \\ 0 & 8 & 3 & 8 \end{vmatrix} / D_{III} = 8 \cdot (-96)/(-2304) = 0.33$$

$$b_3(\frac{1}{2}) = \begin{vmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & 6 & 2 \\ 8 & 0 & 8 & 8 \\ 0 & 8 & 6 & 3 \end{vmatrix} / D_{III} = 8 \cdot 96/(-2304) = -0.33$$

was mit (2.35) zu folgendem Ergebnis führt:

$$x(\frac{1}{2}, \frac{1}{2}) = 0.92 \cdot 4 + 0.46 \cdot 1 + 0.33 \cdot 4 + (-0.33) \cdot 6$$

$$y(\frac{1}{2}, \frac{1}{2}) = 0.92 \cdot 1 + 0.46 \cdot 4 + 0.33 \cdot 6 - 0.33 \cdot 4.$$

Daraus ergibt sich:

$$(\overline{x}, \overline{y}) = (3.48, 3.42)$$

was sich leicht von A II unterscheidet.

Im Knupp lassen sich zwei andere intrinsische Methoden finden, die mathematisch miteinander verglichen werden.

Ein Nachteil von A II / A III ist der folgende:

Ein physikalisches Gebiet, das von einer realistischen Anwendung stammt, ist meistens unabhängig von einem Koordinatensystem. Daher sollte die numerische Gittererzeugung unabhängig sein von den folgenden Operationen:

(1) Translation: $\tilde{z}(\xi, \eta) = z(\xi, \eta) + z_0$

(2) Rotation: $\tilde{\tilde{z}}(\xi, \eta) = e^{i\varphi} z(\xi, \eta)$

(3) Streckung: $\hat{z}(\xi, \eta) = r\, z(\xi, \eta)$

(4) Spiegelung: $z'(\xi, \eta) = \overline{z(\xi, \eta)}$

Alle Interpolationsmethoden führen zu geometrisch kongruenten Gittern für Gebiete, die sich nur durch diese vier Bewegungen unterscheiden. A II dagegen ist nur unabhängig von Drehung und Streckung, jedoch nicht von der Translation, A III ist weder unabhängig von der Translation noch von der Rotation.

Invarianz - Eigenschaften

Alle betrachteten algebraischen Gittererzeugungsmethoden sind unabhängig von (3) und (4). Interpolationsmethoden sind von allen starren Bewegungen (1) - (4) unabhängig

Invarianz-Tabelle		
	Translation	Rotation
Interpolation	invariant	invariant
A II	nicht i.	invariant
A III	nicht i.	nicht i.
A II_∞	invariant	invariant

Dieses ist einerseits schlecht, kann aber andererseits genutzt werden, um unterschiedliche Gitter durch folgende Verfahren zu erzeugen (hier nur für A II):

2.4.3 Der Ansatz A II mit einem shift-Parameter

(1) Gegeben sind z_1, z_2, z_3, z_4 sowie ein komplexer Parameter $\zeta \in \mathbb{C}$.
Ersetze die gegebenen Randfunktionen durch

$$\left.\begin{array}{rcl} \tilde{z}_1(\xi) & = & z_1(\xi) + \zeta \\ \tilde{z}_2(\eta) & = & z_2(\eta) + \zeta \\ \tilde{z}_3(\xi) & = & z_3(\xi) + \zeta \\ \tilde{z}_4(\eta) & = & z_4(\eta) + \zeta \end{array}\right\} \zeta \in \mathbb{C} \tag{2.37}$$

(2) Erzeuge mit den Randfunktionen $\tilde{z}_i, i = 1, \ldots, 4$, mit A II ein Gitter

$$\tilde{z}(\xi, \eta; \zeta)$$

(3) Schiebe zurück

$$z(\xi, \eta) = \tilde{z}(\zeta, \eta; \zeta) - \zeta$$

Interessant ist der Fall $\zeta \to \infty$: A II_∞, denn dieses Gitter ist unabhängig von den Bewegungen (1) bis (4).

Auf die mathematischen Eigenschaften werden wir nicht näher eingehen. Stattdessen werden wir abschließend einige Beispiele betrachten.

2.4.4 Längenminimierung durch shift-Parameter Verallgemeinerter Ansatz

$$\begin{aligned} x(\xi,\eta) &= g(\eta)x_1(\xi) + f(\eta)x_3(\xi) + k(\xi)x_2(\eta) + h(\xi)x_4(\eta) - e(\xi,\eta) \\ y(\xi,\eta) &= \hat{g}(\eta)y_1(\xi) + \hat{f}(\eta)y_3(\xi) + \hat{k}(\xi)y_2(\eta) + \hat{h}(\xi)y_4(\eta) - \hat{e}(\xi,\eta) \end{aligned}$$

Für diesen Ansatz ergibt eine Verschiebung $(x,y) \to (x+p, y+q)$ Gitterfunktionen

$$x^R(\xi,\eta) = x(\xi,\eta) - p[1 - f(\eta) - g(\eta)][1 - h(\xi) - k(\xi)]$$

$$y^R(\xi,\eta) = y(\xi,\eta) - q[1 - \hat{f}(\eta) - \hat{g}(\eta)][1 - \hat{h}(\xi) - \hat{k}(\xi)]$$

Die shift-Parameter nutzt man aus, um ein Funktional zu minimieren, z.B. die Längen

$$l_{ij}^2 := (x_{ij}^R - x_{i-1,j}^R)^2 + (y_{ij}^R - y_{i-1,j}^R)^2 \text{ und } \tilde{l}_{ij}^2 := (x_{ij}^R - x_{i,j-1}^R)^2 + (y_{ij}^R - y_{i,j-1}^R)^2$$

über alle zu erzeugenden inneren Punkte:

$$\min F(p,q) = \sum_{j=2}^{M}\sum_{i=2}^{N+1} l_{ji}^2 + \sum_{j=2}^{M+1}\sum_{i=2}^{N} \tilde{l}_{ij}^2.$$

Man zeigt leicht, daß für die Differenzen gilt:

$$\begin{aligned} x_{ij}^R - x_{i-1,j}^R &= (x_{ij} - x_{i-1,j}) + pA_{ij} \\ y_{ij}^R - y_{i-1,j}^R &= (y_{ij} - y_{i-1,j}) + qB_{ij} \\ x_{ij}^R - x_{i,j-1}^R &= (x_{ij} - x_{i,j-1}) + pC_{ij} \\ y_{ij}^R - y_{i,j-1}^R &= (y_{ij} - y_{i,j-1}) + qD_{ij} \end{aligned}$$

$$\begin{aligned} \text{mit } A_{ij} &= [1 - f(\eta j) - g(\eta j)][(h(\xi_i) - h(\xi_{i-1})) + (k(y_i) - k(\xi_{i-1}))] \\ B_{ij} &= [1 - \hat{f}(\eta_i) - \hat{g}(\eta_i)][(\hat{h}(\xi_i) - \hat{h}(\xi_{i-1})) + (\hat{k}(\xi_i) - \hat{k}(\xi_{i-1}))] \\ C_{ij} &= [1 - h(\xi_i) - k(\xi_i)][(f(y_i) - f(y_{j-1})) + (g(\eta_i) - g(y_{j-1})] \\ D_{ij} &= [1 - \hat{h}(\xi_i) - \hat{k}(\xi_i)][(\hat{f}(y_j) - \hat{f}(y_{j-1})) + (g(\eta_j) - \hat{g}(\eta_{j-1}))] \end{aligned}$$

Man bekommt so eine entkoppelte Lösung für p, q:

$$p = -(\sum\sum A_{ij}(x_{ij} - x_{i-1,j}) + \sum\sum C_{ij}(x_{ij} - x_{i,j-1}))/\Lambda_1$$

$$q = -(\sum\sum B_{ij}(y_{i,j} - y_{i-1,j}) + \sum\sum D_{ij}(y_{ij} - y_{i,j-1})/\Lambda_2$$

mit

$$\Lambda_1 = \sum\sum (A_{ij}^2 + C_{ij}^2)$$

$$\Lambda_2 = \sum\sum (B_{ij}^2 + D_{ij}^2)$$

Insgesamt ist dies ein schneller nicht-iterativer Gittererzeugungsalgorithmus, der die Gesamtlänge der Zellkanten minimiert.

2.4.5 Existenz der Gitter

Nach Definition von vier Eckpunkten auf $\partial\Omega$ kann mit der transfiniten Interpolation immer ein Gitter erzeugt werden. Damit ist nicht gesagt, daß dieses Gitter in Ω bleibt und faltenlos ist.

Für die Methode A II existiert ein Gitter, wenn die Determinante $D_{II} \neq 0$ ist. Es ist schnell nachzurechnen, wann dies nicht der Fall ist:

Lemma 2.4.1 $D_{II} = 0$ genau dann, wenn

$$\begin{aligned} z_1(0) \neq 0 \quad &\text{und} \quad w_3 = \frac{w_1 w_2}{z_1(0)} + w_1 + w_2 \\ \text{oder } z_1(0) = 0 \quad &\text{und} \quad w_1 = 0 \text{ oder } w_2 = 0. \end{aligned}$$

Dabei sind

$$w_1 := z_3(0) - z_1(0), w_2 := z_1(1) - z_1(0), w_3 := z_3(1) - z_1(0).$$

Das Lemma zeigt einen generellen Nachteil der intrinsischen Methoden:

Die Existenz eines Gitters hängt für A II nicht nur vom Gebiet, sondern auch von seiner Lokalisierung im $\mathbb{R}^2$ ab.

Diese Vermutung legt ja auch schon die Invarianz-Tabelle nahe.

D_{II} kann also für viele Gebiete Null werden, wenn ich sie ungeschickt genug in die Ebene lege, wird aber für die meisten Gebiete ungleich Null sein.

Für A III gelten ähnliche Verhältnisse.

2.4.6 Vergleich der algebraischen Methoden

(1) Transfinite Interpolation (T I), A II und A III produzieren alle die Identität für das Einheitsquadrat:

$$x(\xi, \eta) = \xi, \qquad y(\xi, \eta) = \eta$$

(2) Die von T I, A II und A III erzeugten Gitter sind identisch, wenn zwei gegenüberliegende Seiten Geraden sind. Wir erhalten dann:

$$\begin{aligned} x(\xi, \eta) &= (1 - \xi) x_2(\eta) + \xi x_4(\eta), \\ y(\xi, \eta) &= (1 - \xi) y_2(\eta) + \xi y_4(\eta), \end{aligned}$$

also die einfache horizontale Lagrange-Interpolation, entsprechend die vertikale Lagrange-Interpolation, wenn die Seiten C_1 und C_3 Geraden sind.

(3) Die Methoden erzeugen verschiedene Gitter für Gebiete, die nur wenig vom Einheitsquadrat abweichen. Als Beispiel nehmen wir folgende Randkurven

$$\begin{aligned} x_1(\xi) &= \xi + \rho(\xi), \\ y_1(\xi) &= \tau(\xi), \\ x_3(\xi) &= \xi, \\ y_3(\xi) &= 1, \\ x_2(\eta) &= \tilde{\rho}(\eta), \\ y_2(\eta) &= \eta + \tilde{\tau}(\eta), \\ x_4(\eta) &= 1, \\ y_4(\eta) &= \eta \end{aligned}$$

wobei die Funktion $\rho, \tau, \tilde{\rho}$ und $\tilde{\tau}$ die Abweichung vom Einheitsquadrat bestimmen und

$$\rho(0) = \tau(0) = \rho(1) = \tau(1) = \tilde{\rho}(0) = \tilde{\tau}(0) = \tilde{\rho}(1) = \tilde{\tau}(1) = 0$$

erfüllen sollen. Für eine solche Region bekommen wir folgende Gitterformen:

T I :

$$\begin{aligned} x(\xi,\eta) &= \xi + (1-\eta)\rho(\xi) + (1-\xi)\tilde{\rho}(\eta), \\ y(\xi,\eta) &= \eta + (1-\eta)\tau(\xi) + (1-\xi)\tilde{\tau}(\eta). \end{aligned}$$

A II :

$$\begin{aligned} x(\xi,\eta) &= \xi + (1-\eta)\rho(\xi) + (1-\xi)\tilde{\rho}(\eta) \\ &\quad -\tau(\xi)\tilde{\rho}(\eta) + \tau(\xi)\tilde{\tau}(\eta) - \rho(\xi)\tilde{\rho}(\eta) - \rho(\xi)\tilde{\tau}(\eta), \\ y(\xi,\eta) &= \eta + (1-\eta)\tau(\xi) + (1-\xi)\tilde{\tau}(\eta) \\ &\quad -\tau(\xi)\tilde{\rho}(\eta) - \tau(\xi)\tilde{\tau}(\eta) + \rho(\xi)\tilde{\rho}(\eta) - \rho(\xi)\tilde{\tau}(\eta) \end{aligned}$$

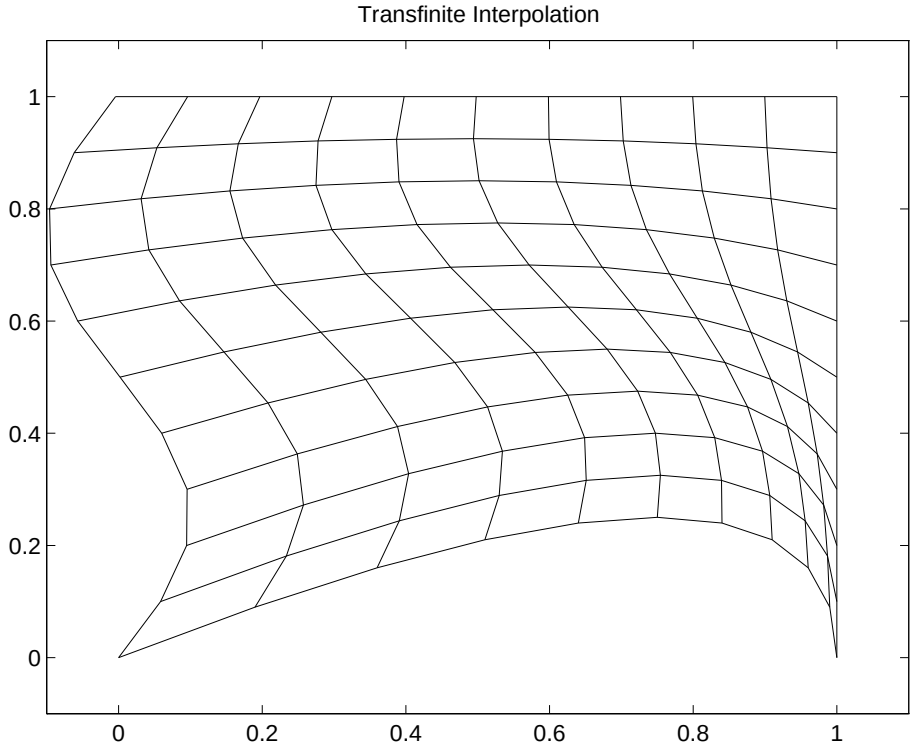
A III :

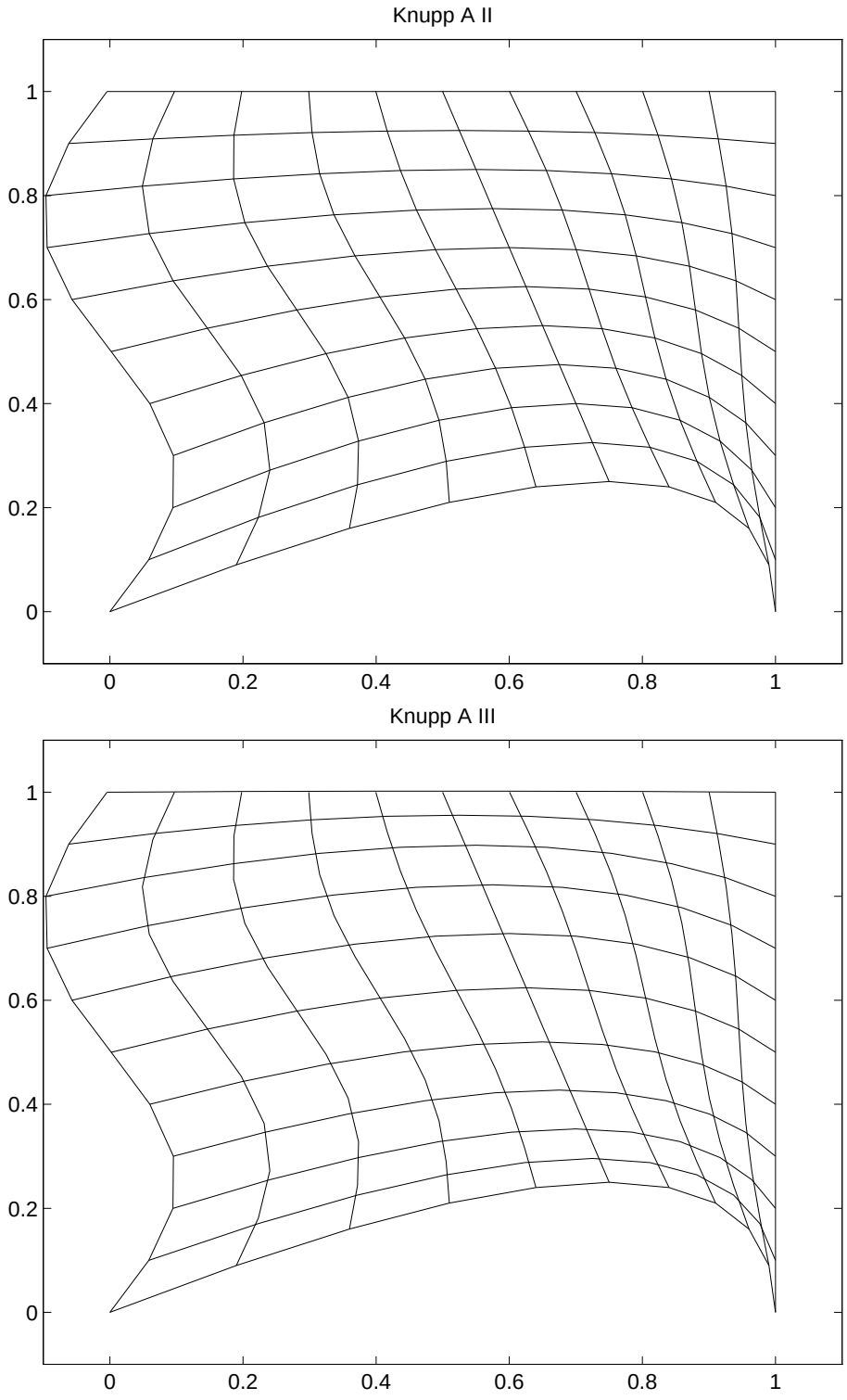
$$\begin{aligned} x(\xi,\eta) &= \xi + (1-\eta)\rho(\xi) + (1-\xi)\tilde{\rho}(\eta) \\ &\quad -\tau(\xi)\tilde{\rho}(\eta) - \tau(\xi)\tilde{\tau}(\eta) - \rho(\xi)\tilde{\rho}(\eta) - \rho(\xi)\tilde{\tau}(\eta), \\ y(\xi,\eta) &= \eta + (1-\eta)\tau(\xi) + (1-\xi)\tilde{\tau}(\eta) \\ &\quad -\tau(\xi)\tilde{\rho}(\eta) - \tau(\xi)\tilde{\tau}(\eta) - \rho(\xi)\tilde{\rho}(\eta) - \rho(\xi)\tilde{\tau}(\eta). \end{aligned}$$

Ein Beispiel für diesen letzten Fall, auf das Rechteck [0,8] x [0,6] verallgemeinert, haben wir in Beispiel 2.2.1 gesehen, wenn auch nur mit einem inneren Punkt.

Beispiel 2.4.2 Wir wollen ein weiteres Beispiel auf dem Einheitsquadrat mit folgenden 'Abweich'-Funktionen betrachten:

$$\begin{aligned} \tau(\xi) &= \xi\,(1-\xi) \\ \rho(\xi) &= \tau(\xi) \\ \tilde{\rho}(\eta) &= 0.1\,\sin(2\,\pi\,\eta) \\ \tilde{\tau}(\eta) &= 0 \end{aligned}$$




Knupp A II
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
Knupp A III
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

Kapitel 3

Kontinuierliche Variationsmethoden

Die klassische Variationsrechnung liefert die Theorie für ein Minimierungsproblem, dessen Lösung Euler-Lagrangegleichungen, im folgenden kurz E-L Gleichungen, genannt werden. Die Parameter der Gitterkontrolle ergeben sich durch die Minimierung von Funktionalen, das sind in den betrachteten Fällen Integrale. Diese Funktionale produzieren dann ein Erzeugendensystem in Form von E-L Gleichungen, die drei Eigenschaften des Gitters kontrollieren können:

(1) die Abstände zwischen benachbarten Gitterpunkten,

(2) die Flächen der Zellen,

(3) die Winkel zwischen den Gitterlinien.

Man unterscheidet zwischen kontinuierlichen und diskreten Variationsmethoden. Wir wollen uns hier nur mit kontinuierlichen Variationsmethoden beschäftigen. Mit Hilfe der Variationsmethoden kann man sehr gleichmäßige Gitter erzeugen, allerdings ist die Existenz und Eindeutigkeit einer Lösung ($\sim$ Gitter) nicht immer garantiert und es kann zu gefalteten Gittern kommen. Wir folgen im wesentlichen dem Buch von Knupp und Steinberg ([42]) und der Diplomarbeit von Frau Annette Froning gen. Havixbeck.

3.1 Einführungsbeispiel

Das folgende Beispiel dient der Einführung in die Variationsmethoden:
Es sei $n \in \mathbb{N}$, $1 \leq i \leq n$ und $\omega_i, \tau_i \in \mathbb{R}$ mit $\omega_i > 0$. Gesucht sind x_i, so dass

$$S = \sum_{i=1}^{n} \frac{(x_i - \tau_i)^2}{2\omega_i} \tag{3.1}$$

minimal wird unter der Bedingung

$$\sum_{i=1}^{n} x_i = 1 \tag{3.2}$$

Geometrisch gesehen ergibt (3.1) für festes S einen Ellipsoiden mit dem Radius $R = S^{\frac{1}{2}}$ und die Bedingung (3.2) eine Hyperebene. Das Minimum tritt auf, wenn der Ellipsoid die Hyperebene in genau einem Punkt trifft (vgl. Steinberg und Roache, 1986 [76]).

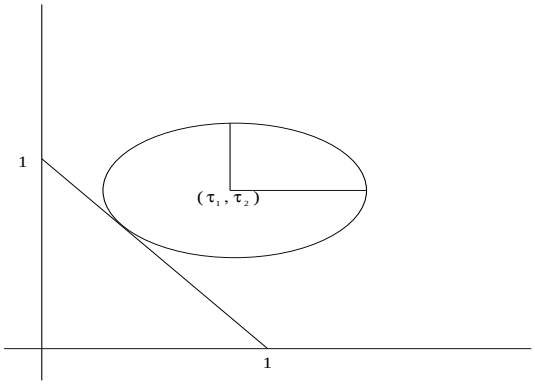


Abbildung 3.1: Geometrische Veranschauung des Beispiels

Das Problem hat ein eindeutiges Minimum und keine Maxima, da S positiv quadratisch ist. Die Lösung erhält man mit der Methode des *Lagrange-Multiplikators*: Setze

$$Q = S - \lambda(\sum_{i=1}^{n} x_i - 1) \tag{3.3}$$

Das Minimum ergibt sich mit

$$\frac{\partial Q}{\partial x_i} = \frac{(x_i - \tau_i)}{\omega_i} - \lambda = 0 \quad \Longrightarrow \quad x_i = \tau_i + \lambda\omega_i, \quad 1 \leq i \leq n. \tag{3.4}$$

Zur Bestimmung von λ muß man die Nebenbedingung einbeziehen. Seien

$$\tau := \sum_{i=1}^{n} \tau_i \quad \text{und} \quad \Omega := \sum_{i=1}^{n} \omega_i,$$

dann folgt aus (3.2) und (3.4)

$$1 = \tau + \lambda\,\Omega \quad \Longrightarrow \quad \lambda = \frac{1 - \tau}{\Omega}.$$

3.2 Eindimensionale Gittererzeugung

Dieses einfache Variationsproblem läßt sich in verallgemeinerter Form zur eindimensionalen Gittererzeugung benutzen. Dabei wollen wir auch noch vom diskreten zu einem kontinuierlichen Variationsproblem übergehen.

Ergänzt man die Variablen x_i um x_0 und ersetzt in (3.1) $(x_i - \tau_i)$ durch $(x_i - x_{i-1})$, dann bedeutet die Lösung (3.4), dass der Abstand zwischen x_i und x_{i-1} proportional zu ω_i ist (vgl. Castillo, 1991 [17]).

Dieses diskrete Problem wird nun auf den kontinuierlichen Fall übertragen. Dazu seien die x_i Werte einer Gitterabstandsfunktion $x(\xi)$ mit einer Referenzvariablen $\xi \in [0,1]$, und die ω_i seien Werte einer Gewichtsfunktion $\Phi(\xi)$. Das führt zu dem diskreten Problem

$$S = \sum_{i=1}^{n} \frac{(x_i - x_{i-1})^2}{2\Phi_{i-1/2}} \tag{3.5}$$

mit $\Phi_{i-1/2} = \Phi(\xi_{i-1/2})$, $\xi \in [0,1]$, und man erhält

$$x_i - x_{i-1} = \lambda \Phi_{i-1/2}, \qquad 1 \le i \le n, \tag{3.6}$$

d.h.: sind die Gitterabstände proportional zur gegebenen Gewichtsfunktion, dann ist die Lösung ein Extremum von S. Teilen durch $\Delta\xi$ ergibt:

$$\frac{S}{\Delta\xi} = \sum_{i=1}^{n} \left(\frac{x_i - x_{i-1}}{\Delta\xi} \right)^2 \frac{\Delta\xi}{2\Phi_{i-1/2}} \tag{3.7}$$

Ist $x(\xi)$ differenzierbar, dann gilt für $\Delta\xi \to 0$ für die rechte Seite

$$I[x] := \int_0^1 \frac{x_\xi^2(\xi)}{2\Phi(\xi)} d\xi \tag{3.8}$$

Problem: $I[x]$ über allen differenzierbaren x zu minimieren mit $x(0) = a$ und $x(1) = b$.

Dieses Problem wollen wir mit klassischer Variationsrechnung lösen. Dazu führen wir eine Funktion $c(\xi)$ ein mit $c(0) = c(1) = 0$. Dadurch erfüllt jede Funktion

$$x(\xi) + \varepsilon\, c(\xi) \tag{3.9}$$

die gleichen Randbedingungen wie $x(\xi)$. Die Funktionenschar (3.9) minimiert $I[x]$ für $\varepsilon = 0$. Sei also

$$F(\varepsilon) := \int_0^1 \frac{(x_\xi(\xi) + \varepsilon c_\xi(\xi))^2}{2\Phi(\xi)} d\xi \tag{3.10}$$

Damit $x(\xi)$ (3.10) minimiert, muß $F'(0) = 0$ sein:

$$\int_0^1 \frac{x_\xi(\xi)}{\Phi(\xi)} c_\xi(\xi) d\xi = 0. \tag{3.11}$$

Partielle Integration und die Beachtung der Randbedingungen für c führen zu

$$\int_0^1 \left(\frac{x_\xi(\xi)}{\Phi(\xi)} \right)_\xi c(\xi) d\xi = 0. \tag{3.12}$$

Damit dies für beliebige stetige Funktionen c zutrifft, muß

$$\left(\frac{x_\xi(\xi)}{\Phi(\xi)}\right)_\xi = 0 \tag{3.13}$$

Diese Differentialgleichung ist die E-L-Gleichung zum Variationsproblem (3.8). Das sich ergebende Gitter ist proportional zu Φ.

Beispiel 3.2.1 Glockenkurven-Verteilung

Wir nehmen vereinfachend für dieses Beispiel $\xi \in [-1, 1]$ und wählen

$$\Phi(\xi) = \exp(-a\,\xi^2).$$

Die Lösung der Differentialgleichung (3.13) ist dann

$$x(\xi) = \frac{\operatorname{erf}(\sqrt{a}\,\xi)}{\operatorname{erf}(\sqrt{a})}$$

mit der sogenannten Fehlerfunktion

$$\operatorname{erf}(\xi) = \frac{2}{\sqrt{\pi}} \int_0^\xi \exp(-t^2)\,dt.$$

Diese Verteilung mit 21 Punkten und für $a = 2.5$ sehen wir in der folgenden Zeichnung:

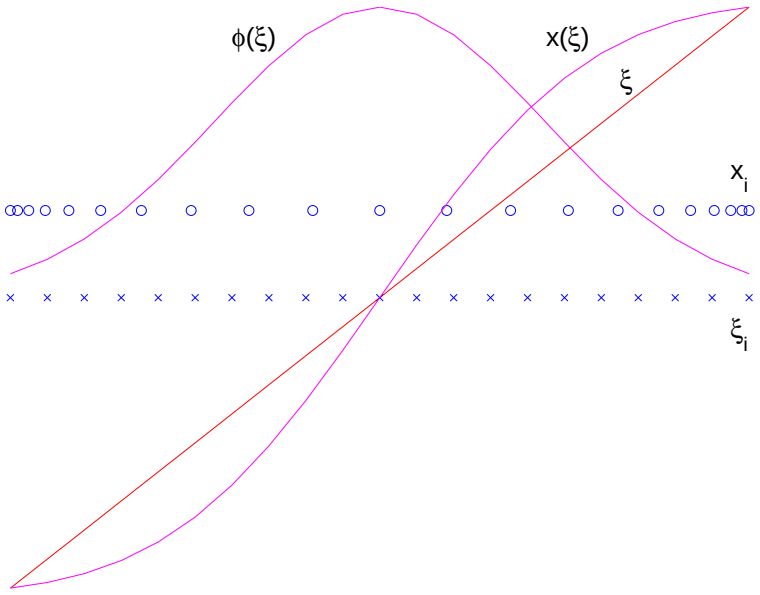


Abbildung 3.2: Gitter proportional zur Glockenkurve

3.3 Gewichtete Integrale

Gewichtete Variationsprinzipien sind nützlich, da sie gewichtete Gittererzeuger liefern, die Kontrolle im Innern erlauben.

Im folgenden Abschnitt wird zunächst das Prinzip der ersten und der zweiten Variation eines Funktionals beschrieben und die Bedingungen für ein Minimum gegeben. Es wird gezeigt, dass die E-L Gleichungen die Transformationen sind, die das Funktional minimieren. In den darauffolgenden Abschnitten wird diese Theorie auf das gewichtete Längenfunktional, das gewichtete Flächenfunktional, das Winkelfunktional und eine Kombination angewandt. Es schließt sich ein Abschnitt über das Problem der Faltung an, welches durch ein Modellbeispiel näher erklärt wird. Diesem folgt ein Abschnitt über weitere allgemeinere Funktionale. Außer den Grundtatsachen über Variationsrechnung, die wir im nächsten Unterabschnitt kurz einführen werden, brauchen wir auch wieder die differentialgeometrischen Grundlagen (Basisvektoren, Maßtensoren, ...) aus dem Abschnitt 2.3.

3.3.1 Variationsrechnung

Ein Funktional in der Variationsrechnung ist eine Menge von Vektoren von Funktionen; im Falle der Gittererzeugung eine Menge von C^2- Funktionen mit Dirichlet RB. Dazu sei

$$I[\mathbf{x}] = \int_0^1 \int_0^1 G(\xi, \eta, \mathbf{x}, \mathbf{x}_\xi, \mathbf{x}_\eta)\, d\xi d\eta \tag{3.14}$$

ein solches zu minimierendes Funktional, wobei $\mathbf{x}(\xi, \eta) = (x(\xi, \eta), y(\xi, \eta))$ ist.

Definition 3.3.1 a) Ist $I[\mathbf{x}]$ ein mehrdimensionales Funktional und $\mathbf{c}$ ein Vektor von Funktionen, so dass $\mathbf{x} + \varepsilon\mathbf{c}$ im Vektorraum liegt, dann lautet die erste Variation

$$D_{\mathbf{c}} I[\mathbf{x}] = \left. \frac{d}{d\varepsilon} I[\mathbf{x} + \varepsilon\mathbf{c}] \right|_{\varepsilon=0} \tag{3.15}$$

b) und die zweite Variation

$$D^2_{\mathbf{c}} I[\mathbf{x}] = \left. \frac{d}{d\varepsilon} D_{\mathbf{c}} I[\mathbf{x} + \varepsilon\mathbf{c}] \right|_{\varepsilon=0} \tag{3.16}$$

Satz 3.3.2 a) Eine notwendige Bedingung für die Existenz eines Minimums von $I[\mathbf{x}]$ an der Stelle $\mathbf{x}_0$ lautet:

$$D_{\mathbf{c}} I[\mathbf{x}_0] = 0 \quad \forall \mathbf{c}$$

b) Hinreichend für die Existenz eines Minimums an der Stelle $\mathbf{x}_0$ ist:

$$D^2_{\mathbf{c}} I[\mathbf{x}_0] > 0 \quad \forall \mathbf{c}$$

Dies führt zu einem Randwertproblem, dessen PDEs die E-L Gleichungen sind.

$\mathbf{c} = (c_1, c_2)$ ist eine glatte Funktion mit verschwindenen Randwerten: $\mathbf{c}(0)|_{\partial\Omega} = 0$. Diese RB garantieren, dass, falls das Minimum $\mathbf{x} = \mathbf{x}(\xi, \eta)$ Dirichlet-RB erfüllt, dies auch für $\mathbf{x}(\xi, \eta) + \varepsilon\mathbf{c}(\xi, \eta)$ gilt.

Satz 3.3.3 Eine notwendige Bedingung für $I[\mathbf{x}]$ für ein Extremum in $\mathbf{x}_0$ ist die Tatsache, dass $\mathbf{x}_0$ den E-L Gleichungen von I genügt.

3.3.2 Gewichtetes Längenfunktional

Zur Erinnerung:
$g = M^T M$ ist der Maß- oder Metriktensor (siehe Abschnitt 2.3.2) mit:
$g_{11} = \mathbf{x}_\xi \mathbf{x}_\xi$ ist das Quadrat des Tangentenvektors in ξ- Richtung,
$g_{22} = \mathbf{x}_\eta \mathbf{x}_\eta$ das des Tangentenvektors in η- Richtung.
$g_{12} = \mathbf{x}_\xi \mathbf{x}_\eta$ ist das innere Produkt zweier Tangentenvektoren.
$\sqrt{\det(g)} = j$ ist die Fläche des Parallelogramms, dessen Seiten $\mathbf{x}_\xi$ und $\mathbf{x}_\eta$ sind.

In der gewichteten Längenkontrolle ist ein Gitter gesucht, sodass $\sqrt{g_{11}}$ und $\sqrt{g_{22}}$ zu zwei gegebenen positiven Gewichtsfunktionen $\Phi = \Phi(\xi, \eta)$ und $\Psi = \Psi(\xi, \eta)$ proportional sind. Aus dieser Zielsetzung ergibt sich:

$$I_L = \frac{1}{2} \int_0^1 \int_0^1 \left(\frac{g_{11}}{\Phi} + \frac{g_{22}}{\Psi}\right) d\xi d\eta \tag{3.17}$$

Mit $\mathbf{c}(\xi, \eta) = (c_1(\xi, \eta), c_2(\xi, \eta))$ wie oben erhält man:

$$I[\mathbf{x} + \varepsilon \mathbf{c}] = \frac{1}{2} \int_0^1 \int_0^1 \frac{(\mathbf{x}_\xi + \varepsilon \mathbf{c}_\xi)^2}{\Phi} + \frac{(\mathbf{x}_\eta + \varepsilon \mathbf{c}_\eta)^2}{\Psi} d\xi d\eta \tag{3.18}$$

Wir gehen jetzt genauso vor wie im eindimensionalen Fall. Differenzieren nach ε ergibt

$$\frac{d}{d\varepsilon} I[\mathbf{x} + \varepsilon \mathbf{c}] = \frac{1}{2} \int_0^1 \int_0^1 \frac{2\,(\mathbf{x}_\xi + \varepsilon \mathbf{c}_\xi) \cdot \mathbf{c}_\xi}{\Phi} + \frac{2\,(\mathbf{x}_\eta + \varepsilon \mathbf{c}_\eta) \cdot \mathbf{c}_\eta}{\Psi}\, d\xi d\eta. \tag{3.19}$$

Die 1. Variation erhalten wir durch $\varepsilon = 0$:

$$D_c I_L[\mathbf{x}] = \int_0^1 \int_0^1 \left(\frac{\mathbf{x}_\xi \mathbf{c}_\xi}{\Phi} + \frac{\mathbf{x}_\eta \mathbf{c}_\eta}{\Psi}\right) d\xi d\eta \tag{3.20}$$

Partielle Integration ($\Omega_R = [0,1] \times [0,1]$):

$$D_c I_L[\mathbf{x}] = \frac{\mathbf{x}_\xi}{\Phi} \mathbf{c}\Big|_{\partial\Omega_R} + \frac{\mathbf{x}_\eta}{\Psi} \mathbf{c}\Big|_{\partial\Omega_R} - \int_0^1 \int_0^1 \left(\frac{\mathbf{x}_\xi}{\Phi}\right)_\xi \mathbf{c} + \left(\frac{\mathbf{x}_\eta}{\Psi}\right)_\eta \mathbf{c}\, d\xi d\eta \tag{3.21}$$

Mit $\mathbf{c} = 0$ auf $\partial\Omega_R$ folgt:

$$D_c I_L[\mathbf{x}] = -\int_0^1 \int_0^1 \mathbf{c} \cdot \left\{\left(\frac{\mathbf{x}_\xi}{\Phi}\right)_\xi + \left(\frac{\mathbf{x}_\eta}{\Psi}\right)_\eta\right\} d\xi d\eta \tag{3.22}$$

Um I_L zu minimieren, muss $D_c I_L$ für alle $\mathbf{c}$ gleich Null sein, womit sich die E-L Gleichungen für das gewichtete Längenfunktional ergeben:

$$\left(\frac{\mathbf{x}_\xi}{\Phi}\right)_\xi + \left(\frac{\mathbf{x}_\eta}{\Psi}\right)_\eta = 0. \tag{3.23}$$

Mit der Quotientenregel können diese anders geschrieben werden:

$$\frac{\mathbf{x}_{\xi\xi}\Phi - \mathbf{x}_\xi \Phi_\xi}{\Phi^2} + \frac{\mathbf{x}_{\eta\eta}\Psi - \mathbf{x}_\eta \Psi_\eta}{\Psi^2} = 0$$

$$\Rightarrow \tau_{11}\mathbf{x}_{\xi\xi} + \tau_{12}\mathbf{x}_{\xi\eta} + \tau_{22}\mathbf{x}_{\eta\eta} + S = 0 \tag{3.24}$$

mit

$$\begin{aligned} \tau_{11} &= \frac{1}{\Phi} \cdot I_2 \\ \tau_{12} &= 0 \cdot I_2 \\ \tau_{22} &= \frac{1}{\Psi} \cdot I_2 \end{aligned}$$

wobei $I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ und

$$S = -\frac{\Phi_\xi}{\Phi^2}\mathbf{x}_\xi - \frac{\Psi_\eta}{\Psi^2}\mathbf{x}_\eta$$

Mit $\Psi = \Phi = const.$ sind die E-L Gleichungen die ungekoppelten Laplacegleichungen

$$\mathbf{x}_{\xi\xi} + \mathbf{x}_{\eta\eta} = 0. \tag{3.25}$$

Dies stellt den Zusammenhang mit der Gittererzeugung mit elliptischen Differentialgleichungen her, siehe 2.3. Allerdings ergeben sich hier die Laplacegleichungen zur direkten Erzeugung des physikalischen Gebietes, also für die Transformation vom Rechengebiet Ω_R auf das physikalische Gebiet Ω, umgekehrt zu 2.3.

Die Lösung mit den üblichen zentralen Differenzen liefert mit $\Delta\xi = \Delta\eta$ folgende Formel mit Werten benachbarter Punkte:

$$\mathbf{x}_{\xi\xi} + \mathbf{x}_{\eta\eta} = \frac{1}{\Delta\xi^2}(\mathbf{x}_{i-1,j} + \mathbf{x}_{i+1,j} + \mathbf{x}_{i,j+1} + \mathbf{x}_{i,j-1} - 4\mathbf{x}_{i,j}). \tag{3.26}$$

$$\Longrightarrow \mathbf{x}_{i,j} = \frac{\mathbf{x}_{i-1,j} + \mathbf{x}_{i+1,j} + \mathbf{x}_{i,j+1} + \mathbf{x}_{i,j-1}}{4} \tag{3.27}$$

Diese Formel kann man durch den 5-Punkte-Differenzenstern darstellen: Die E-L Gleichun-

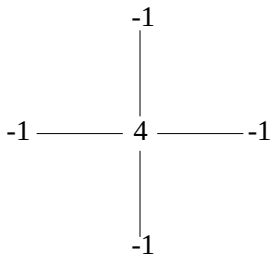


Abbildung 3.3: 5-Punktestern

gen sind linear und ungekoppelt und somit sind die Lösungen eindeutig und existieren. Diese Tatsache ermöglicht auch eine leichte numerische Lösung der Gleichungen zur Erzeugung der Gitter.

Sind die Gewichtsfunktionen nicht konstant, so bekommt man wegen $\Phi > 0$ und $\Psi > 0$ elliptische PDEs und kann deshalb glatte Gitter erwarten. Man muss jedoch bei diesem Funktional immer mit gefalteten Gittern rechnen wie wir im Abschnitt 3.4 sehen werden.

3.3.3 Gewichtetes Flächenfunktional

Die Proportionalität zwischen den Flächen der Zellen und einer gegebenen Gewichtsfunktion $\Phi = \Phi(\xi, \eta)$ ist das Ziel dieses Funktionals. Es ist bekannt, dass $\det(g) = j^2$ und dass $j = x_\xi y_\eta - x_\eta y_\xi$ proportional zur Fläche einer Zelle ist.

$$\text{Minimiere} \quad I_A[\mathbf{x}] = \frac{1}{2}\int_0^1\int_0^1 \frac{\det(g)}{\Phi} d\xi d\eta = \frac{1}{2}\int_0^1\int_0^1 \frac{j^2}{\Phi} d\xi d\eta \tag{3.28}$$

Mit derselben Variationsrechnung wie oben ergeben sich die folgenden E-L-Gleichungen:

$$\left(\frac{j\mathbf{x}_\eta}{\Phi}\right)_\xi - \left(\frac{j\mathbf{x}_\xi}{\Phi}\right)_\eta = 0 \tag{3.29}$$

Die Produktregel liefert:

$$\begin{aligned}
(\tfrac{j}{\Phi})_\xi \mathbf{x}_\eta - (\tfrac{j}{\Phi})_\eta \mathbf{x}_\xi &= 0 \\
\iff \left(\frac{j_\xi\Phi - j\Phi_\xi}{\Phi^2}\right)\mathbf{x}_\eta - \left(\frac{j_\eta\Phi - \Phi_\eta j}{\Phi^2}\right)\mathbf{x}_\xi &= 0 \\
\iff j_\xi x_\eta - j_\eta x_\xi - j\frac{\Phi_\xi}{\Phi}x_\eta + j\frac{\Phi_\eta}{\Phi}x_\xi &= 0 \\
j_\xi y_\eta - j_\eta y_\xi - j\frac{\Phi_\xi}{\Phi}y_\eta + j\frac{\Phi_\eta}{\Phi}y_\xi &= 0
\end{aligned}$$

Mit $j = x_\xi y_\eta - x_\eta y_\xi$ und

$$\begin{aligned}
j_\xi &= x_{\xi\xi}y_\eta + x_\xi y_{\eta\xi} - x_{\eta\xi}y_\xi - x_\eta y_{\xi\xi} \\
j_\eta &= x_{\xi\eta}y_\eta + x_\xi y_{\eta\eta} - x_{\eta\eta}y_\xi - x_\eta y_{\xi\eta}
\end{aligned}$$

lässt sich die E-L Gleichung folgendermaßen schreiben:

$$\tau_{11}\mathbf{x}_{\xi\xi} + \tau_{12}\mathbf{x}_{\xi\eta} + \tau_{22}\mathbf{x}_{\eta\eta} + S = 0 \tag{3.30}$$

$$\begin{aligned}
\text{mit} \quad \tau_{11} &= \begin{pmatrix} y_\eta^2 & -x_\eta y_\eta \\ -x_\eta y_\eta & x_\eta^2 \end{pmatrix} \\
\tau_{12} &= \begin{pmatrix} -2y_\xi y_\eta & x_\xi y_\eta - x_\eta y_\xi \\ x_\xi y_\eta + x_\eta y_\xi & -2x_\xi y_\eta \end{pmatrix} \\
\tau_{22} &= \begin{pmatrix} y_\xi^2 & -x_\xi y_\xi \\ -x_\xi y_\xi & x_\xi^2 \end{pmatrix} \\
S &= -\frac{j}{\Phi}\begin{pmatrix} y_\eta & -y_\xi \\ -x_\eta & x_\xi \end{pmatrix}\begin{pmatrix} \Phi_\xi \\ \Phi_\eta \end{pmatrix}
\end{aligned}$$

Hier sind die Matrizen τ_{ij} von den Gewichtsfunktionen unabhängig. Die E-L Gleichungen sind quasilinear, gekoppelt und nicht immer elliptisch. Die Lösungen müssen nicht existieren. Es gibt zur Zeit keine Aussage darüber, wann Lösungen existieren und ob sie eindeutig sind. Da die Gleichungen gekoppelt sind, sind sie numerisch schwieriger zu lösen als die E-L Gleichungen von I_L. Die fehlende Elliptizität ist der Grund für nichtglatte Gitter.

3.3.4 Winkelfunktional

Anders als beim Längen- und Flächenfunktional wird das Winkelfunktional nicht von einer Proportionalität abgeleitet. Sind die Gitterlinien orthogonal, so gilt g_{12}=0 mit der Maßtensorkomponente $g_{12} =\mid \mathbf{x}_\xi \mid \mid \mathbf{x}_\eta \mid \cos\Theta$, wo Θ der Winkel zwischen den beiden Tangentenvektoren der Gitterlinien ist. Um die Winkel der Zellen zu kontrollieren, wählt man deshalb

$$I_O = \frac{1}{2}\int_0^1\int_0^1 g_{12}^2 d\xi d\eta = \frac{1}{2}\int_0^1\int_0^1 (\mathbf{x}_\xi \cdot \mathbf{x}_\eta)^2 d\xi d\eta \tag{3.31}$$

Dies führt zu den E-L Gleichungen:

$$(g_{12}\mathbf{x}_\eta)_\xi + (g_{12}\mathbf{x}_\xi)_\eta = 0 \tag{3.32}$$

Mit $g_{12} = x_\xi x_\eta + y_\xi y_\eta$ ergibt sich:

$$\tau_{11}\mathbf{x}_{\xi\xi} + \tau_{12}\mathbf{x}_{\xi\eta} + \tau_{22}\mathbf{x}_{\eta\eta} + S = 0 \tag{3.33}$$

wobei

$$\begin{aligned}
\tau_{11} &= \begin{pmatrix} x_\eta^2 & x_\eta y_\eta \\ x_\eta y_\eta & y_\eta^2 \end{pmatrix} \\
\tau_{12} &= \begin{pmatrix} 4x_\xi y_\eta + 2y_\xi y_\eta & x_\xi y_\eta + x_\eta y_\xi \\ x_\xi y_\eta + x_\eta y_\xi & 4y_\xi y_\eta + 2x_\xi x_\eta \end{pmatrix} \\
\tau_{22} &= \begin{pmatrix} x_\xi^2 & x_\xi y_\xi \\ x_\xi y_\xi & y_\xi^2 \end{pmatrix} \\
S &= 0
\end{aligned}$$

Diese E-L Gleichungen sind ebenso wie die des Funktionals I_A quasilinear, gekoppelt und nicht immer elliptisch. Die Lösungen existieren oft nicht, da der Iterationsprozess nicht konvergiert. Diese Methode ist also nicht empfehlenswert.

3.3.5 Kombinationen

Um die gewünschten Eigenschaften zu optimieren, minimiert man gewichtete Kombinationen der drei betrachteten Funktionale. Sei

$$I_K[\mathbf{x}] := \omega_L I_L + \omega_A I_A + \omega_O I_O \tag{3.34}$$

mit

$$\omega_L + \omega_A + \omega_O = 1, \quad \omega_L, \omega_A, \omega_O \geq 0.$$

Jetzt wird versucht, diese Gewichtsparameter optimal zu wählen. Mit $\omega_L = 0$ und $\omega_A = \frac{1}{2}$ erhält man das sogenannte AO- Funktional:

$$I_{AO}[\mathbf{x}] = \frac{1}{2}\int_0^1\int_0^1 \frac{j^2 + g_{12}^2}{\Phi} d\xi d\eta = \frac{1}{2}\int_0^1\int_0^1 \frac{g_{11}g_{22}}{\Phi} d\xi d\eta \tag{3.35}$$

Die E-L Gleichungen haben die Form

$$\left(\frac{g_{22}\mathbf{x}_\xi}{\Phi}\right)_\xi + \left(\frac{g_{11}\mathbf{x}_\eta}{\Phi}\right)_\eta = 0 \tag{3.36}$$

Analog wie oben kann die folgende Gestalt erreicht werden:

$$\tau_{11}\mathbf{x}_{\xi\xi} + \tau_{12}\mathbf{x}_{\xi\eta} + \tau_{22}\mathbf{x}_{\eta\eta} + S = 0 \tag{3.37}$$

wobei

$$\begin{aligned}
\tau_{11} &= \begin{pmatrix} x_\eta^2 + y_\eta^2 & 0 \\ 0 & x_\eta^2 + y_\eta^2 \end{pmatrix} \\
\tau_{12} &= \begin{pmatrix} 4x_\xi x_\eta & 2(x_\xi y_\eta + x_\eta y_\xi) \\ 2(x_\xi y_\eta + x_\eta y_\xi) & 4y_\xi y_\eta \end{pmatrix} \\
\tau_{22} &= \begin{pmatrix} x_\xi^2 + y_\xi^2 & 0 \\ 0 & x_\xi^2 + y_\xi^2 \end{pmatrix} \\
S &= \frac{1}{\Phi}\begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix}\begin{pmatrix} g_{22}\Phi_\xi \\ g_{11}\Phi_\eta \end{pmatrix}
\end{aligned}$$

Diese Gleichungen sind quasilinear, gekoppelt und nicht elliptisch. Die ungewichteten Gitter sind glatt, meist ungefaltet, haben fast gleich große Zellen und die Gitterlinien sind nahezu orthogonal.

Die angegeben Formen der E-L-Gleichungen von I_L, I_A, I_O und I_{AO}

$$\tau_{11}\mathbf{x}_{\xi\xi} + \tau_{12}\mathbf{x}_{\xi\eta} + \tau_{22}\mathbf{x}_{\eta\eta} + S = 0$$

können auch in Matrixvektorform geschrieben werden:

$$\begin{aligned}
&\begin{pmatrix} A_{11} & B_{11} \\ B_{11} & C_{11} \end{pmatrix}\begin{pmatrix} x_{\xi\xi} \\ y_{\xi\xi} \end{pmatrix} + \\
&\begin{pmatrix} A_{12} & B_{12} \\ B_{12} & C_{12} \end{pmatrix}\begin{pmatrix} x_{\xi\eta} \\ y_{\xi\eta} \end{pmatrix} + \\
&\begin{pmatrix} A_{22} & B_{22} \\ B_{22} & C_{22} \end{pmatrix}\begin{pmatrix} x_{\eta\eta} \\ y_{\eta\eta} \end{pmatrix} + \\
&= \begin{pmatrix} S_1 \\ S_2 \end{pmatrix}
\end{aligned} \tag{3.38}$$

wobei die drei Koeffizientenmatrizen symmetrisch sind. A_{ij}, B_{ij}, C_{ij} und S hängen von den ersten Ableitungen, der Transformation, der Gewichtsfunktion und deren Ableitungen ab.

Die Gleichungen sind nichtlinear und gekoppelt und werden deshalb iterativ gelöst, z.B. mit der Picarditeration oder dem Newtonverfahren. Dazu benötigt man ein Startgitter. Dieses erhält man durch tranfinite Interpolation oder einen anderen einfachen Gittererzeuger.

Die Diskretisierung erfolgt mit zentralen Differenzen:

$$\begin{aligned} x_{\xi\xi} &\approx \frac{x_{i-1,j} - 2x_{i,j} + x_{i+1,j}}{\Delta\xi^2} \\ x_{\eta\eta} &\approx \frac{x_{i,j-1} - 2x_{i,j} + x_{i,j+1}}{\Delta\eta^2} \\ x_{\xi\eta} &\approx \frac{x_{i+1,j+1} - x_{i-1,j+1} - x_{i+1,j-1} + x_{i-1,j-1}}{4\Delta\xi\Delta\eta} \end{aligned}$$

Beispiel 3.3.4 Für die E-L Gleichungen des Längenfunktionals ergibt sich:

$$\begin{aligned} \frac{1}{\Phi} I_2 \mathbf{x}_{\xi\xi} + \frac{1}{\Psi} I_2 \mathbf{x}_{\eta\eta} &= -\frac{\Phi_\xi}{\Phi^2}\mathbf{x}_\xi - \frac{\Psi_\eta}{\Psi^2}\mathbf{x}_\eta \\ A_{11} = C_{11} &= \frac{1}{\Phi} \\ B_{11} = A_{12} = B_{12} = C_{12} = B_{22} &= 0 \\ A_{22} = C_{22} &= \frac{1}{\Psi} \end{aligned}$$

Somit gilt:

$$\begin{aligned} A_{11}x_{\xi\xi} + A_{22}x_{\eta\eta} &= \frac{\Phi_\xi}{\Phi^2}x_\xi - \frac{\Psi_\eta}{\Psi^2}x_\eta = S_1 \\ C_{11}y_{\xi\xi} + C_{22}y_{\eta\eta} &= \frac{\Phi_\xi}{\Phi^2}y_\xi - \frac{\Psi_\eta}{\Psi^2}y_\eta = S_2. \end{aligned}$$

Diskretisierung mit $h := \Delta\xi = \Delta\eta$ ergibt:

$$\begin{aligned} &\frac{A_{11}}{h^2}(x_{i-1,j} - 2x_{ij} + x_{i+1,j}) + \frac{A_{22}}{h^2}(x_{i,j-1} - 2x_{i,j} + x_{i,j+1}) \\ &= (\frac{\Phi_\xi}{\Phi^2}(x_{i+1,j} - x_{i-1,j}) + \frac{\Psi_\eta}{\Psi^2}(x_{i,j+1} - x_{i,j-1}))\frac{1}{2\,h} \\ \Leftrightarrow\ & x_{i-1,j}\frac{1}{\Phi} + x_{i+1,j}\frac{1}{\Phi} + x_{i,j-1}\frac{1}{\Psi} + x_{i,j+1}\frac{1}{\Psi} - 2x_{i,j}(\frac{1}{\Phi} + \frac{1}{\Psi}) \\ &= (\frac{\Phi_\xi}{\Phi^2}(x_{i+1,j} - x_{i-1,j}) + \frac{\Psi_\eta}{\Psi^2}(x_{i,j+1} - x_{i,j-1}))\frac{h}{2}, \end{aligned}$$

y analog.

Bemerkung 3.3.5 (Knupp/Steinberg): Generell gute Gitter erzeugt

$$I_{AO^2} = \int_0^1 \int_0^1 [g_{11}\, g_{22}]^2 \, d\xi d\eta.$$

3.4 Faltung

Zwei wichtige Eigenschaften eines 'guten' Gitters sind:

(1) Gitterlinien in die gleiche Richtung sollen sich nicht schneiden.

(2) Gitterlinien in verschiedene Richtungen sollen sich genau einmal schneiden.

Gilt eine dieser Eigenschaften nicht, so spricht man von *Faltung*.

Definition 3.4.1 Transformationen, bei denen die Jacobideterminante j in einem Punkt 0 wird, nennt man *gefaltet*.

Außerdem können gefaltete Gitter auch durch Diskretisierungsfehler entstehen.

Beispiel 3.4.2 Die bekannten Funktionale (und weitere Kombinationen) wollen wir jetzt auf ein Modellproblem anwenden. Dazu betrachten wir ein hufeisenförmiges Gebiet (C-Typ), das z.B. an einem Flugzeugflügel auftreten kann (vgl. Castillo, 1988 [16]). Die Ergebnisse können auch auf andere Geometrien übertragen werden.

Das Modell dieses Gebietes ist eine aus neun Punkten bestehende Vereinfachung mit acht Rand- und einem inneren Punkt, der bestimmt werden soll. Aus einem Gebiet zwischen zwei Ellipsen wird eines zwischen zwei Dreiecken. Das physikalische Gebiet wird nur durch

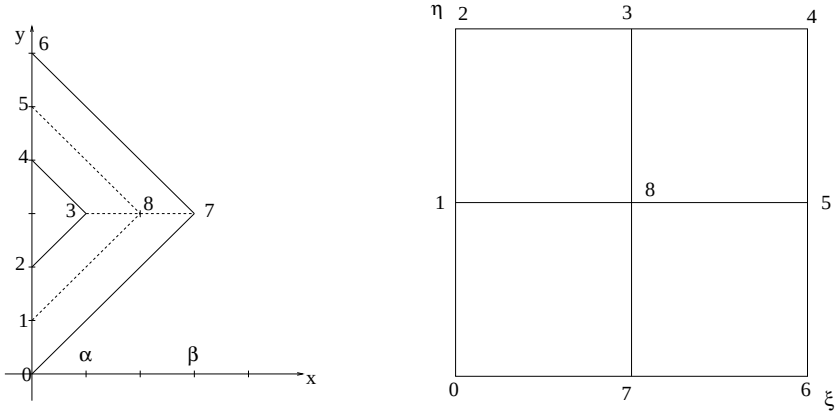


Abbildung 3.4: Modellgebiet

die Wahl von α und β mit $\alpha < \beta$ verändert. Unbekannt ist nur die x-Koordinate von Punkt 8. Das Gitter ist nicht gefaltet, wenn

$$\alpha < x_8 < \beta \tag{3.39}$$

gilt. Die Gitterlinien berühren sich, falls $x_8 = \alpha$ oder $x_8 = \beta$.

Längenfunktional

Das Differenzenverfahren liefert für das Längenfunktional mit konstanten Gewichtsfunktionen, vgl. (3.27):

$$\begin{aligned} 4(\alpha+\beta-4x_8) &= 0 \\ \Rightarrow \quad x_8 &= \frac{\alpha+\beta}{4} \end{aligned} \tag{3.40}$$

Unter der Bedingung (3.39) folgt:

$$\alpha < \frac{\alpha+\beta}{4} \quad \Rightarrow \quad \alpha < \frac{\beta}{3}.$$

D.h., ein Gitter, das mit den E-L Gleichungen von I_L erzeugt wird, faltet sich, sobald $\alpha > \frac{\beta}{3}$ und zwei Gitterlinien berühren sich, falls $\alpha = \frac{\beta}{3}$.

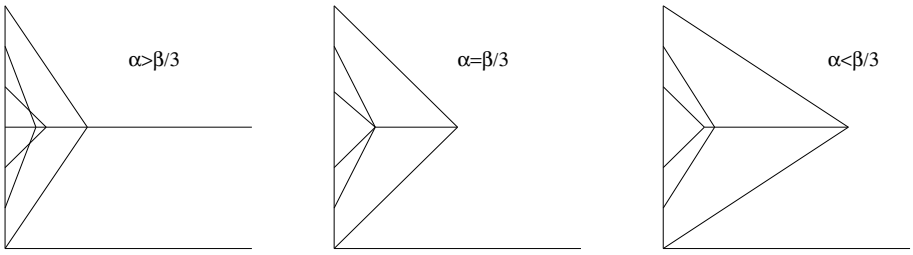


Abbildung 3.5: Längenfunktional

Flächenfunktional

Hier erhält man analog mit der Diskretisierung der Gleichung (3.30):

$$\begin{aligned} 64(\alpha - 2x_8 + \beta) + 16(\beta - \alpha) &= 0 \\ \Rightarrow \quad 64\alpha - 128x_8 + 64\beta + 16\beta - 16\alpha &= 0 \\ \Rightarrow \quad -8x_8 + 3\alpha + 5\beta &= 0 \\ \Rightarrow \quad x_8 = \frac{3\alpha + 5\beta}{8} & \end{aligned} \tag{3.41}$$

Unter der Bedingung (3.39) folgt:

$$\alpha < \frac{3\alpha + 5\beta}{8} \quad \Rightarrow \quad \underline{\alpha < \beta}$$

Dies war aber vorausgesetzt und ist somit immer gewährleistet. D.h.: in diesem Modell ergibt die Erzeugung durch die E-L Gleichungen von I_A ein faltenloses Gitter.

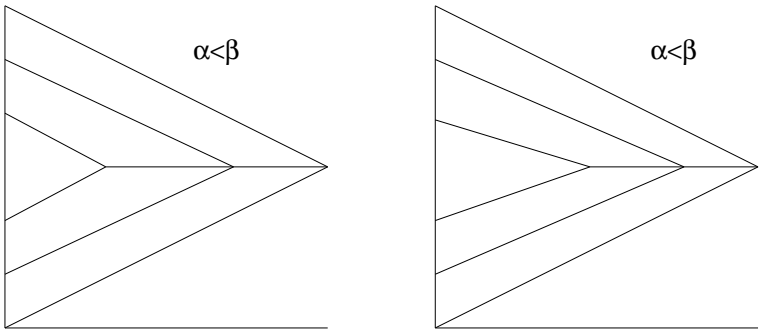


Abbildung 3.6: Flächenfunktional

Winkelfunktional

Das Differenzenverfahren für (3.33) liefert für das Winkelfunktional I_O:

$$(\alpha - \beta)^2(-8x_8) + 16(\beta - \alpha) = 0 \tag{3.42}$$

$$\Rightarrow \quad x_8 = \frac{2}{\beta - \alpha}$$

Mit der Bedingung (3.39) folgt:

$$\alpha < \frac{2}{\beta - \alpha} < \beta$$

Es ergeben sich folgende Gleichungen für die Berührungspunkte links bzw. rechts:

$$\alpha^2 - \alpha\beta + 2 = 0 \quad \text{bzw.} \quad \beta^2 - \alpha\beta - 2 = 0$$

wobei $\sqrt{8} \leq \beta$ für die Gleichung des linken Berührungspunktes gelten muss.

Die erste Gleichung liefert zwei Berührungspunkte (B_1 und B_2), die zweite Gleichung einen (B_3). Die folgenden Abbildungen zeigen die Veränderung der Gitterlinien bei festem β und Lösen der Gleichung nach α. Man sieht leicht: Ist $\alpha < B_1$, so liegt keine Faltung vor [a)]. In Abb. b) ist $\alpha = B_1$. Liegt α zwischen B_1 und B_2, so ist das Gitter gefaltet [c)]. Zwischen B_2 und B_3 liefert die Erzeugung durch I_O ein ungefaltetes Gitter [d),e),f)] und ist $\alpha > B_3$, so liegt wieder Faltung vor [g)].

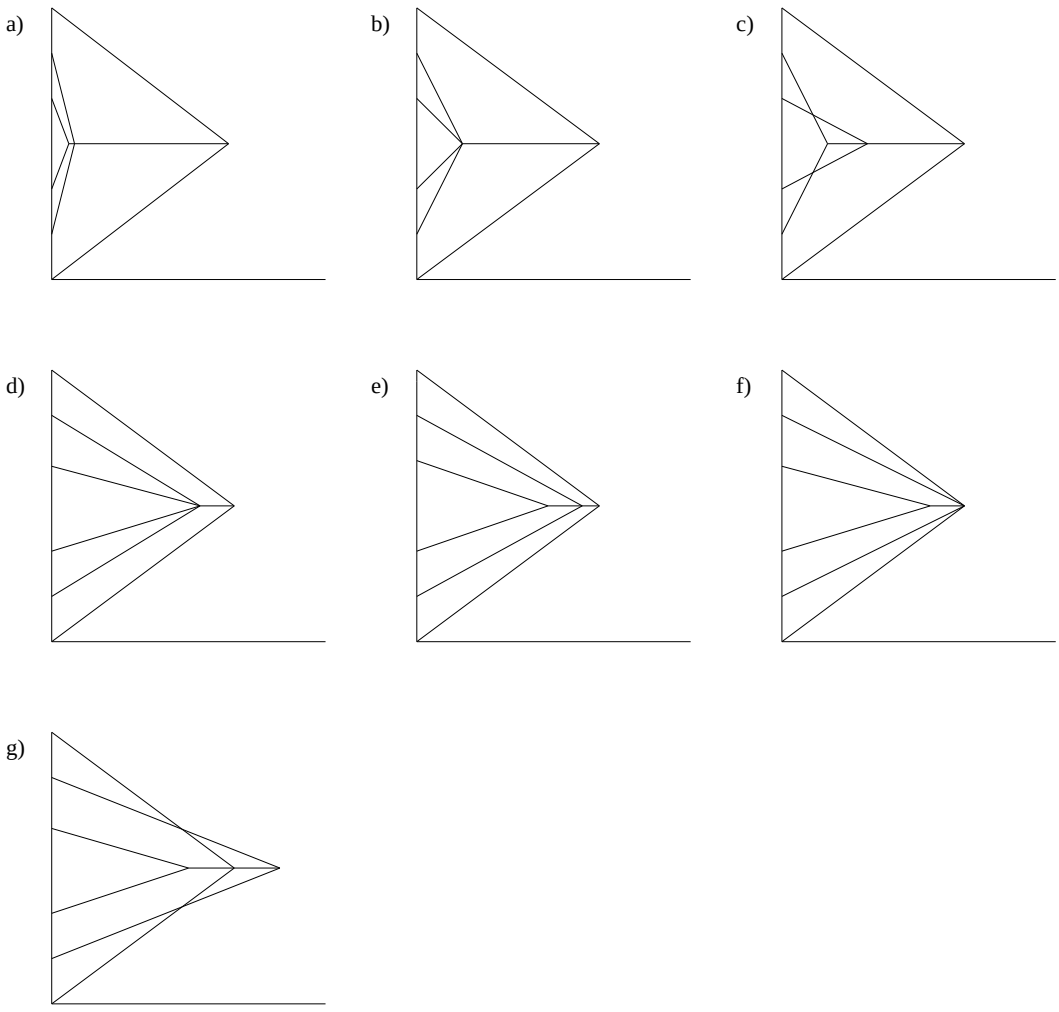


Abbildung 3.7: Winkelfunktional

3.4.1 Kontrolle mehrerer Eigenschaften

Es wird die Kombination aus I_L, I_A und I_O

$$I = aI_L + bI_A + cI_O$$

betrachtet, wobei wieder $a + b + c = 1$ und $a, b, c \geq 0$ gelte.

a, b und c sollen für dieses Problem so gefunden werden, dass dadurch die Eigenschaften des Gitters optimiert werden und insbesondere keine Faltung auftritt. Die diskretisierte E-L-Gleichung ist einfach die mit a, b, c gewichtete Summe der Gleichungen (3.40), (3.41) und (3.42):

$$4a(-4x_8 + \beta + \alpha) + 16b(-8x_8 + 5\beta + 3\alpha) - 8c(\beta - \alpha)(-(\beta - \alpha)x_8 + 2) = 0 \qquad (3.43)$$

In den folgenden Abschnitten werden die Kombinationen zweier bzw. aller drei Funktionale betrachtet und verglichen.

Länge und Fläche

Es ist $c = 0$. Zwei Gleichungen bekommen wir, wenn wir $a + b = 1$ berücksichtigen und in (3.43) $x_8 = \dfrac{\alpha + \beta}{2}$ einsetzen. Die so entstehenden zwei Gleichungen haben die Lösungen

$$a = \frac{4(\beta - \alpha)}{5\beta - 3\alpha}, \quad b = \frac{\beta + \alpha}{5\beta - 3\alpha}.$$

Sie sind für $\beta > \alpha$ immer lösbar und deshalb können a und b immer sinnvoll bestimmt werden. Falls $\alpha < \beta < \frac{5\alpha}{3}$, dann ist $a < b$. Man sieht also, dass das Flächenfunktional wichtiger als das Längenfunktional ist.

Die folgenden Abbildungen entstanden mit
a) a=0.667, b=0.333
b) a=0.727, b=0.273

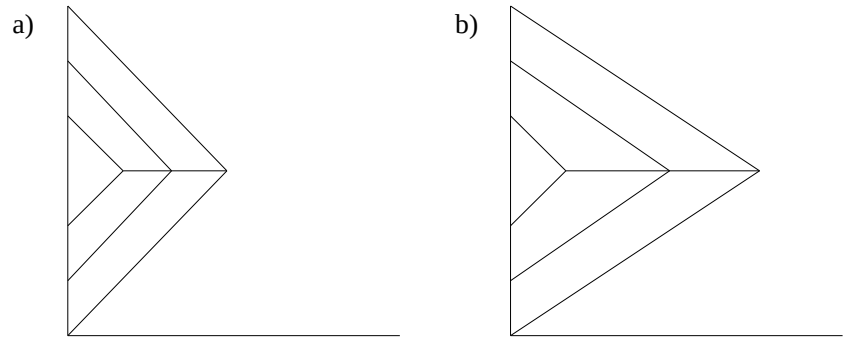


Abbildung 3.8: Länge und Fläche

Fläche und Winkel

Es ist $a = 0$ und wir setzen (w.o.)

$$b = 1 - \frac{4}{\beta^2 - \alpha^2}, \quad c = \frac{4}{\beta^2 - \alpha^2}.$$

Es gilt immer $c \geq 0$. $b \geq 0$ gilt nur im Falle $\beta^2 - \alpha^2 > 4$. Falls $\beta \geq \alpha + \sqrt{2}$, gilt $b \geq c$. D.h. das Flächenfunktional ist auch hier wichtiger als das Winkelfunktional. Die folgenden Abbildungen zeigen das Gitter mit
a) $b = 0.5$, $c = 0.5 \Longleftrightarrow \alpha = 1$, $\beta = 3$,
b) $b = 0.733$, $c = 0.267 \Longleftrightarrow \alpha = 1$, $\beta = 4$,
c) $b = 0.833$, $c = 0.167 \Longleftrightarrow \alpha = 1$, $\beta = 5$.

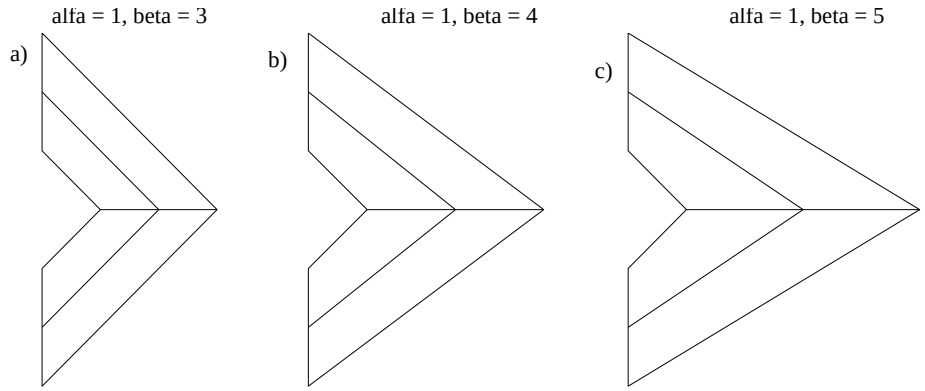


Abbildung 3.9: Fläche und Winkel

Länge und Winkel

Es ist $b = 0$ und wir setzen (w.o.)

$$c = \frac{\alpha + \beta}{5\beta - 3\alpha - (\alpha + \beta)(\beta - \alpha)^2},$$

$$a = 1 - \frac{\alpha + \beta}{5\beta - 3\alpha - (\alpha + \beta)(\beta - \alpha)^2}.$$

Um zu gewährleisten, dass $0 \leq a,\ c \leq 1$ erfüllt sind, muss gelten: $\beta^2 - \alpha^2 \leq 4$. Diese Einschränkung zeigt, dass diese Kombination nicht sehr sinnvoll ist. Somit wird wieder der Einfluss von I_A deutlich.
Trotzdem zwei Abbildungen:
a) a=0.004, c=0.996
b) a=0.06, c=0.94

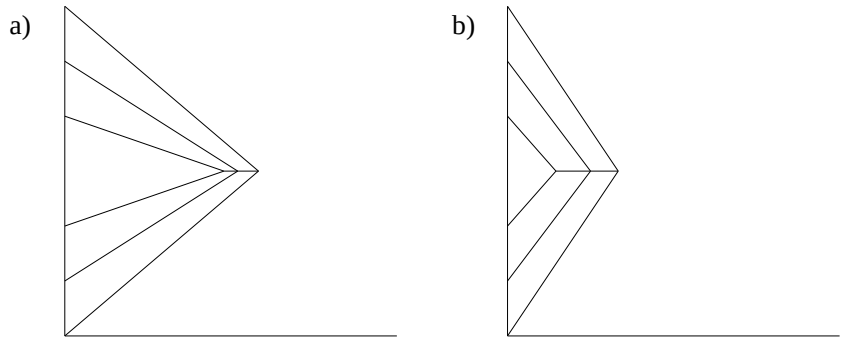


Abbildung 3.10: Länge und Winkel

Länge, Fläche und Winkel

Wir lassen $c = 1 - b - a$ als Parameter stehen und setzen

$$a = \frac{4(\beta - \alpha) - c(\alpha + \beta)(\beta - \alpha)^2}{5\beta - 3\alpha},$$

$$b = \frac{\beta - \alpha + c((\alpha + \beta)(\beta - \alpha)^2 - 5\beta - 3\alpha)}{5\beta - 3\alpha}.$$

Da $a \geq 0$ gelten soll, folgt:

$$c \leq \frac{4}{\beta^2 - \alpha^2}.$$

Mit $\beta^2 - \alpha^2 > 4$ gilt $0 \leq a, b, c \leq 1$. Um sinnvolle Gitter zu erhalten, muss b wieder das stärkere Gewicht tragen.

Zwei Beispiele mit
a) a=0.001, b=0.885, c=0.114
b) a=0.034, b=0.786, c=0.18

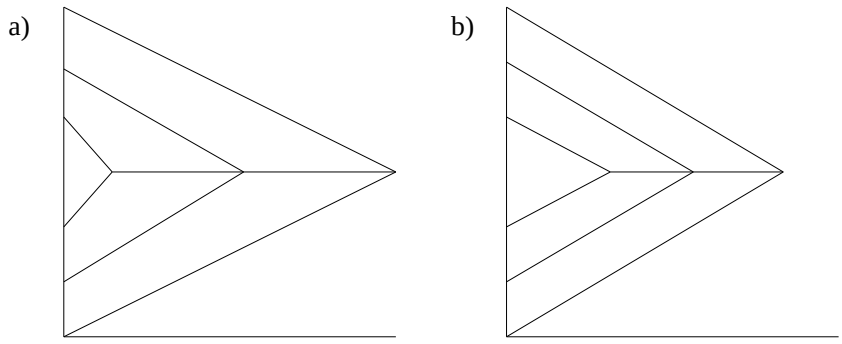


Abbildung 3.11: Länge, Fläche und Winkel

3.4.2 Zusammenfassung

Bei ausreichender Flächenkontrolle unter Hinzunahme von Längen- und Winkelkontrolle erhält man Gitter mit den gewünschten Eigenschaften. Die Kombinationen aus I_A und I_L erzeugen die geeignetesten Gitter; I_O liefert zusätzlich "Orthogonalität" im Innern, jedoch nicht in der Nähe des Randes.

Das Modellproblem ist zwar eine Einschränkung, es gibt jedoch eine gute Einsicht in die Methode. Hervorzuheben ist noch einmal die Tatsache, dass die Gitter, bei deren Erzeugung die Flächenkontrolle signifikant ist, nicht gefaltet sind.

3.5 Beispiele

Von den 18 Beispielgebieten, die Knupp und Steinberg zeigen, wollen wir uns einige ansehen und auf ihnen jeweils mit den drei Minimierungsfunktionalen

- I_L Längenkontrolle,
- I_A Flächenkontrolle und
- I_{AO} Flächen-Winkel-Kombination

Gitter zu erzeugen versuchen.

Wir werden dabei feststellen, dass erst Kombinationen der Funktionale gute Gitter liefern, wobei die Parameter ω_L, ω_A und ω_O für jedes Gebiet gesondert gefunden werden müssen.

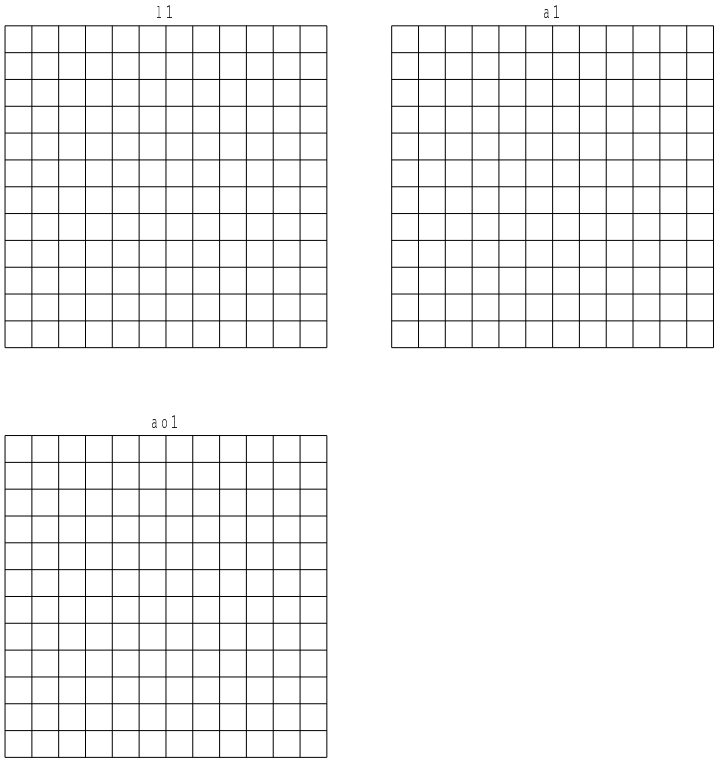


Abbildung 3.12: Länge, Fläche und AO für das Quadrat

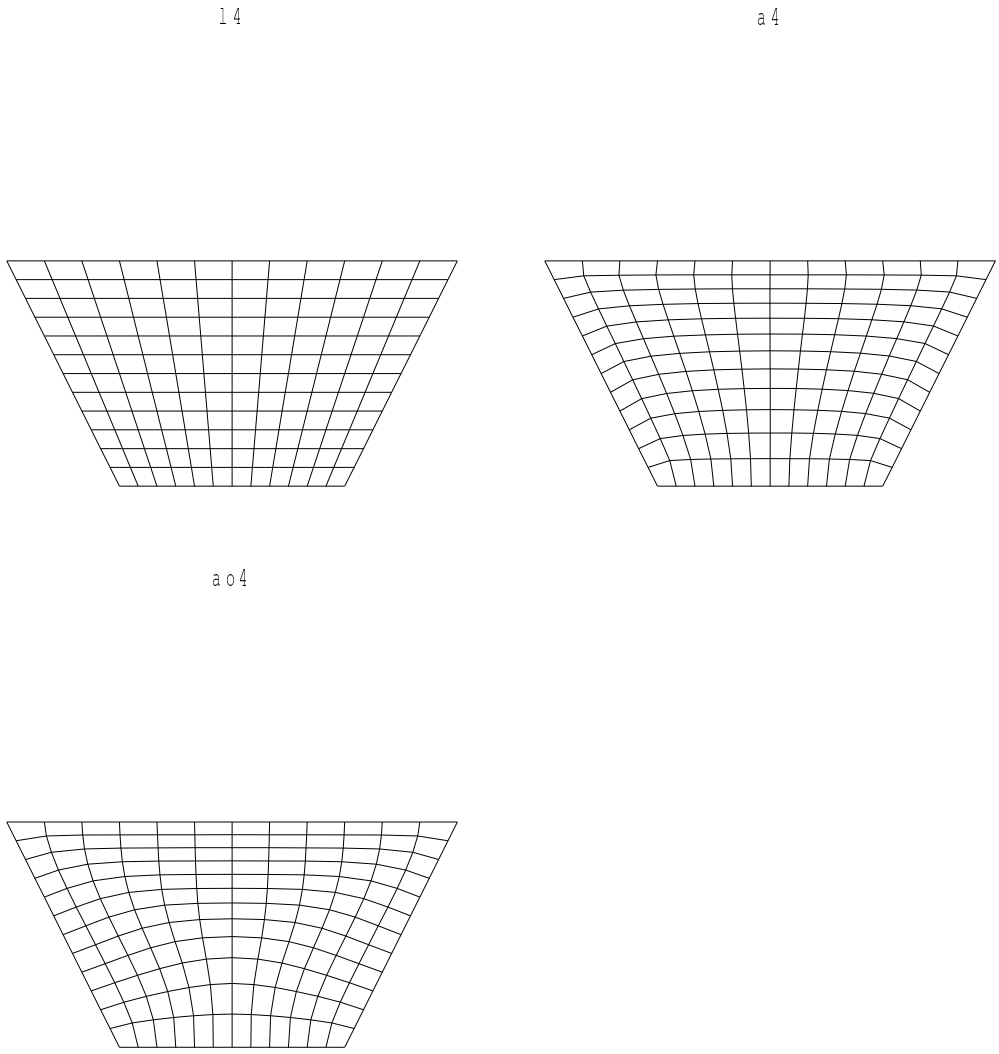


Abbildung 3.13: Länge, Fläche und AO für das Trapez

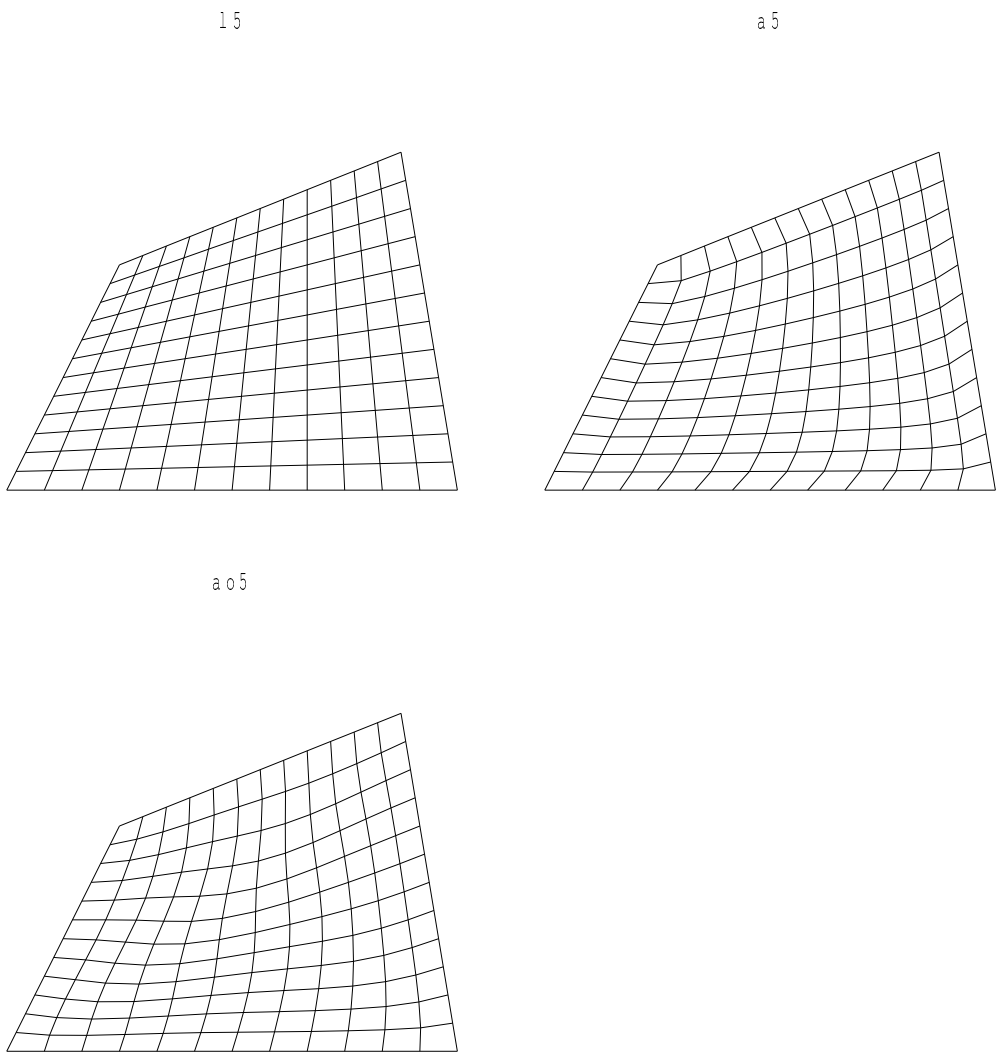


Abbildung 3.14: Länge, Fläche und AO für das Viereck

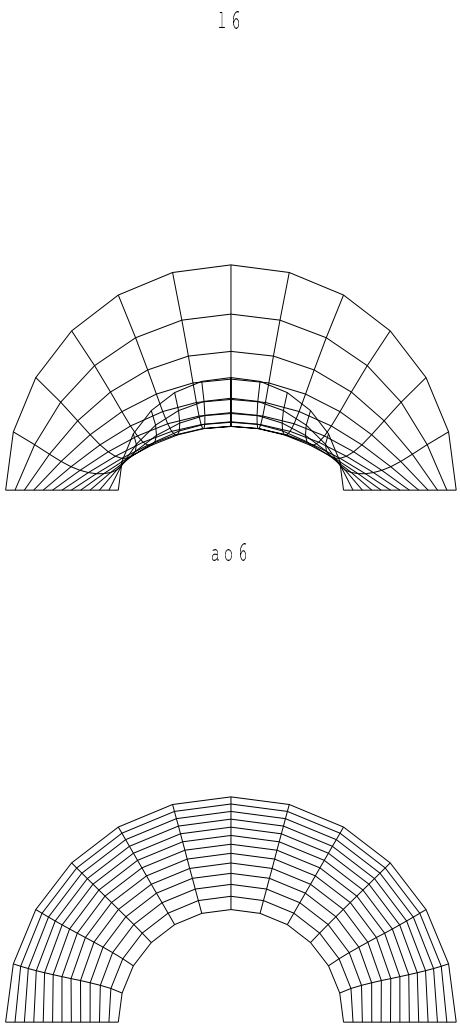


Abbildung 3.15: Länge und AO für das Hufeisen 1

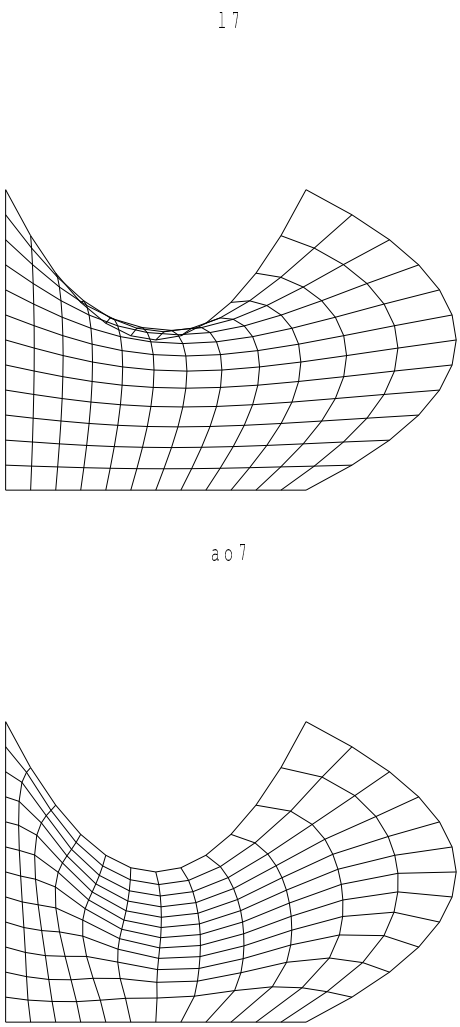


Abbildung 3.16: Länge und AO für den Schwan

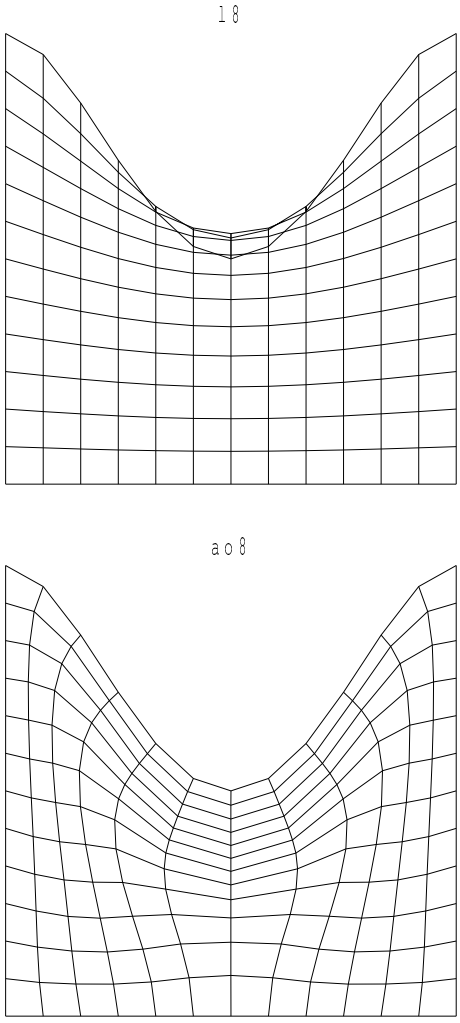


Abbildung 3.17: Länge und AO für das 'Dome'

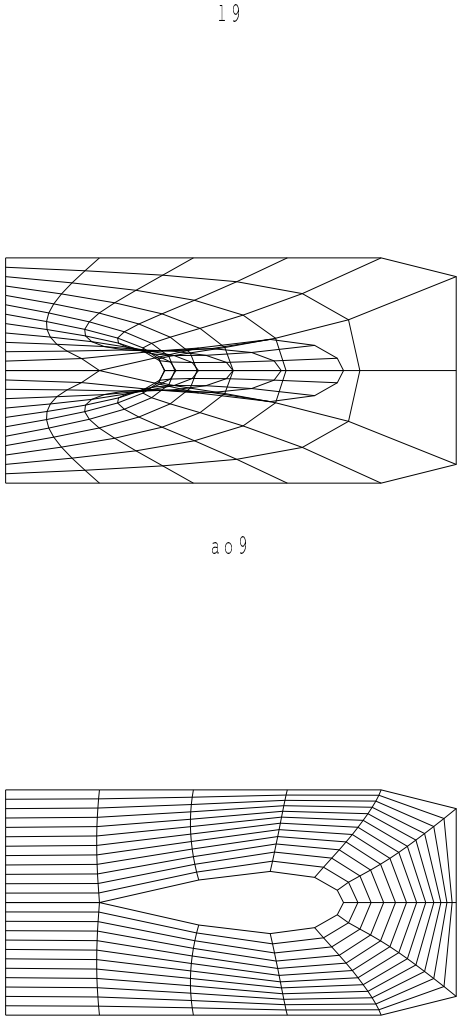
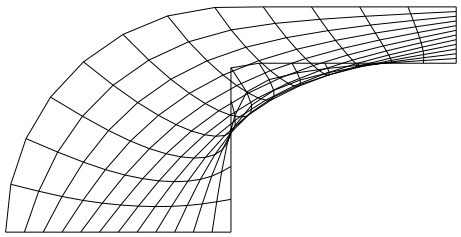


Abbildung 3.18: Länge und AO für das 'Airfoil 1'

l 1 3



a o l 3

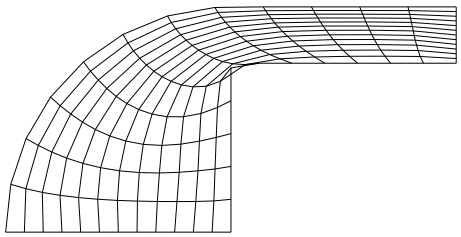


Abbildung 3.19: Länge und AO für den Pflug

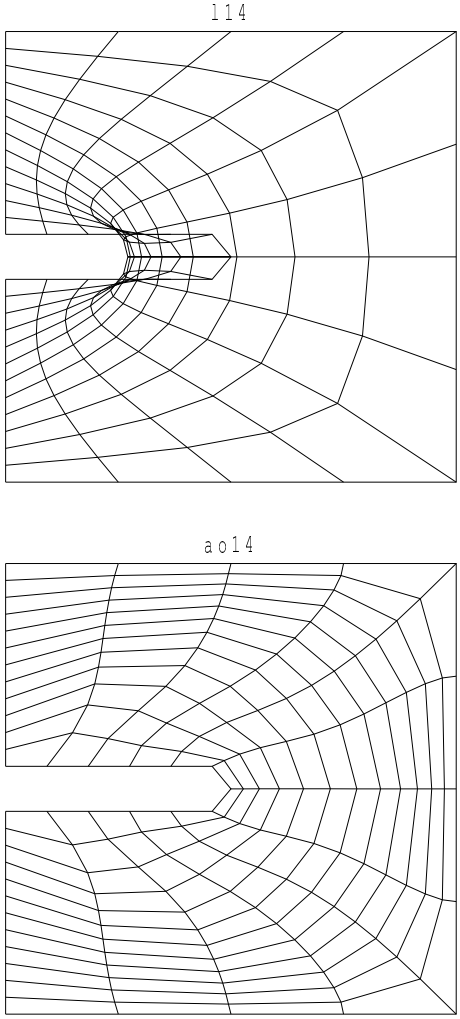


Abbildung 3.20: Länge und AO für das 'C'

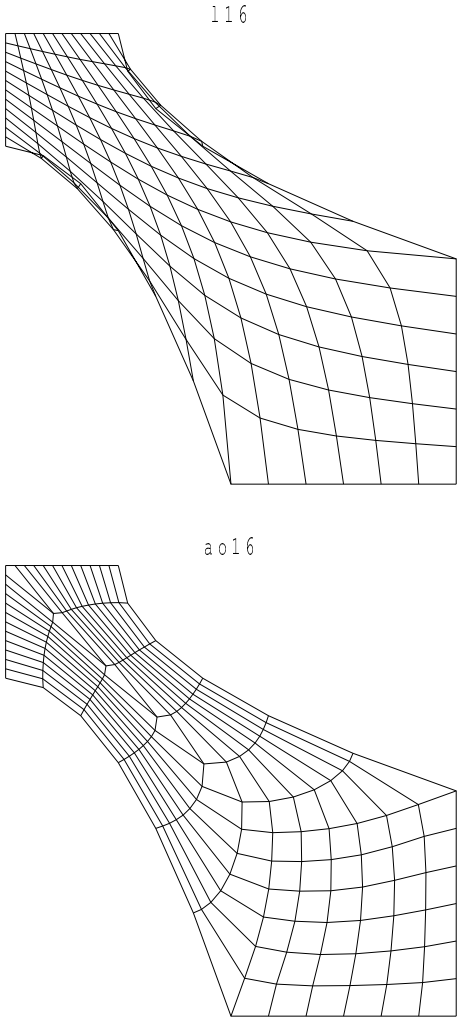


Abbildung 3.21: Länge und AO für die Krawattenform

3.6 Weitere Funktionale

Die bisher betrachteten Variationsprobleme in der Ebene wollen wir in allgemeiner Form schreiben. dazu gibt es drei Möglichkeiten.

(1) Minimiere

$$I_1[\mathbf{x}] = \int_0^1 \int_0^1 G(\xi, \eta, \mathbf{x}, \mathbf{x}_\xi, \mathbf{x}_\eta) d\xi d\eta \tag{3.44}$$

unter Beachtung von Randbedingungen zur Erhaltung der Randkonformität.

(2) Die Einführung einer Gewichtsfunktion im logischen Raum und die Beschränkung auf die Abhängigkeit des Integranden von ersten Ableitungen führt zu:

Minimiere unter Beachtung Dirichletscher Randbedingungen

$$I_2[\mathbf{x}] = \int_0^1 \int_0^1 \frac{\hat{G}(\mathbf{x}_\xi, \mathbf{x}_\eta)}{\Phi(\xi, \eta)} d\xi d\eta \tag{3.45}$$

Dies ist die Form, die wir für die speziellen vier Funktionale

I_L	Gleichung (3.17)
I_A	Gleichung (3.28)
I_O	Gleichung (3.31)
I_{AO}	Gleichung (3.35)

schon betrachtet haben.

(3) Soll die Gewichtsfunktion im physikalischen Raum definiert sein, so erhalten wir das Problem:

Minimiere unter Beachtung Dirichletscher Randbedingungen

$$I_3[\mathbf{x}] = \int_0^1 \int_0^1 \frac{\hat{G}(\mathbf{x}_\xi, \mathbf{x}_\eta)}{\omega^2(\mathbf{x})} d\xi d\eta \tag{3.46}$$

Wie wir gesehen haben, sind Funktionale von besonderem Interesse, die die Elemente des Maß- oder Metriktensors

$$g := M^T M \text{ mit } M = \left(\frac{\partial(x, y)}{\partial(\xi, \eta)} \right) \text{ und } j := \sqrt{\det g}$$

enthalten. Außerdem erscheint eine Gewichtsfunktion im physikalischen Raum sinnvoll. Das führt zu dem allgemeinen Funktional

$$I_{MT}[\mathbf{x}] = \int_0^1 \int_0^1 \frac{H(g_{11}, g_{12}, g_{22}, j)}{\omega^2(\mathbf{x})} d\xi d\eta, \tag{3.47}$$

wobei $H : I\!R^4 \to I\!R$ eine glatte positive Funktion sei.

Die folgende Tabelle aus [42] fasst die uns schon bekannten Funktionale der ungewichteten Variationsprinzipien zusammen, liefert die Form von H und die zugehörigen E-L Gleichungen:

	Symbol	H	E-L-Gleichungen
Länge	I_L	$g_{11}+g_{22}$	$\mathbf{x}_{\xi\xi}+\mathbf{x}_{\eta\eta}=0$
Fläche	I_A	j^2	$(j\mathbf{x}_\eta)_\xi-(j\mathbf{x}_\xi)_\eta=0$
Winkel	I_O	g_{12}^2	$\mathbf{x}_{\eta\xi}+\mathbf{x}_{\xi\eta}=0$
Fläche- Winkel	I_{AO}	$g_{11}g_{22}$	$(g_{22}\mathbf{x}_\xi)_\xi+(g_{11}\mathbf{x}_\eta)_\eta=0$

Unter Nutzung der Tensoranalysis können die E-L Gleichungen in Form des Divergenzoperators in Bezug auf logische oder physikalische Gewichtsfunktionen beschrieben werden. Wir wollen hier nur das Ergebnis für das allgemeine Funktional I_{MT}, (3.47), angeben. Dazu brauchen wir einige Tensoren, die wir zunächst definieren wollen:

$$C := j\,(M^{-1})^T = j\begin{pmatrix} y_\eta & -y_\xi \\ -x_\eta & x_\xi \end{pmatrix},$$

$$\tau := \begin{pmatrix} \hat{G}_{x_\xi} & \hat{G}_{x_\eta} \\ \hat{G}_{y_\xi} & \hat{G}_{y_\eta} \end{pmatrix} = \begin{pmatrix} H_{x_\xi} & H_{x_\eta} \\ H_{y_\xi} & H_{y_\eta} \end{pmatrix}$$

Die E-L-Gleichung zum Funktional I_{MT} läßt sich jetzt schreiben als

$$\frac{H}{j}\mathrm{div}_{(\xi,\eta)}\frac{C}{\omega^2}-\mathrm{div}_{(\xi,\eta)}\frac{\tau}{\omega^2}=0 \tag{3.48}$$

Sei

$$\mathbf{x}_\xi^\perp=(-y_\xi,x_\xi)^T,\quad \mathbf{x}_\eta^\perp=(-y_\eta,x_\eta)^T.$$

Dann gilt offensichtlich:

$$\frac{\partial g_{11}}{\partial \mathbf{x}_\xi}=2\mathbf{x}_\xi \quad \frac{\partial g_{12}}{\partial \mathbf{x}_\xi}=\mathbf{x}_\eta \quad \frac{\partial g_{22}}{\partial \mathbf{x}_\xi}=0 \quad \frac{\partial j}{\partial \mathbf{x}_\xi}=-\mathbf{x}_\eta^\perp$$

$$\frac{\partial g_{11}}{\partial \mathbf{x}_\eta}=0 \quad \frac{\partial g_{12}}{\partial \mathbf{x}_\eta}=\mathbf{x}_\xi \quad \frac{\partial g_{22}}{\partial \mathbf{x}_\eta}=2\mathbf{x}_\eta \quad \frac{\partial j}{\partial \mathbf{x}_\eta}=\mathbf{x}_\xi^\perp$$

und man sieht leicht, dass

$$\begin{aligned}\frac{\partial H}{\partial \mathbf{x}_\xi}&=2\mathbf{x}_\xi\frac{\partial H}{\partial g_{11}}+\mathbf{x}_\eta\frac{\partial H}{\partial g_{12}}-\mathbf{x}_\eta^\perp\frac{\partial H}{\partial j}\\ \frac{\partial H}{\partial \mathbf{x}_\eta}&=2\mathbf{x}_\eta\frac{\partial H}{\partial g_{22}}+\mathbf{x}_\eta\frac{\partial H}{\partial g_{12}}+\mathbf{x}_\xi^\perp\frac{\partial H}{\partial j}\end{aligned} \tag{3.49}$$

Lemma 3.6.1 Es seien die Tensoren B und $\hat{M}$ definiert als

$$\begin{aligned}(\hat{M})_{ij} &:= (1+\delta_{ij})\frac{\partial H}{\partial g_{ij}}\\ B &:= \hat{M}+j\frac{\partial H}{\partial j}\hat{G}^{-1}.\end{aligned}$$

Dann gilt

$$\tau = jB. \tag{3.50}$$

Gleichung (3.48) lässt sich damit umformen zu

$$\mathrm{div}_{(\xi,\eta)}\frac{(\nabla_{(\xi,\eta)}\mathbf{x})B}{\omega^2} = \frac{H}{j}\mathrm{div}_{(\xi,\eta)}\frac{C}{\omega^2} \tag{3.51}$$

Das allgemeine Variationsproblem I_{MT} (3.47) führt also zu der nichtlinearen E-L-Gleichung (3.51).

Die partiellen Ableitungen sowie die Form des Tensors B werden in der folgenden Tabelle aus [42] zusammengefasst.

	$\frac{\partial H}{\partial g_{11}}$	$\frac{\partial H}{g_{12}}$	$\frac{\partial H}{\partial g_{22}}$	$\frac{\partial H}{\partial j}$	B
Länge	1	0	1	0	$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$
Fläche	0	0	0	$2j$	$2j^2\begin{pmatrix} g_{22} & -g_{12} \\ -g_{12} & g_{11} \end{pmatrix}$
Winkel	0	$2g_{12}$	0	0	$\begin{pmatrix} 0 & 2g_{12} \\ 2g_{12} & 0 \end{pmatrix}$
Fläche- Winkel	g_{22}	0	g_{11}	0	$\begin{pmatrix} 2g_{22} & 0 \\ 0 & 2g_{11} \end{pmatrix}$

3.7 Numerische Implementierung

Die Lösung der nichtlinearen E-L Gleichung (3.51) erfolgt in diesem Abschnitt. Die Gleichung kann explizit folgendermaßen geschrieben werden:

$$\begin{aligned}
\left(\frac{B_{11}x_\xi + B_{12}x_\eta}{\omega^2}\right)_\xi + \left(\frac{B_{12}x_\xi + B_{22}x_\eta}{\omega^2}\right)_\eta &= \frac{H}{j}\left(\left(\frac{y_\eta}{\omega^2}\right)_\xi - \left(\frac{y_\xi}{\omega^2}\right)_\eta\right), \\
\left(\frac{B_{11}y_\xi + B_{12}y_\eta}{\omega^2}\right)_\xi + \left(\frac{B_{12}y_\xi + B_{22}y_\eta}{\omega^2}\right)_\eta &= \frac{H}{j}\left(\left(-\frac{x_\eta}{\omega^2}\right)_\xi + \left(\frac{x_\xi}{\omega^2}\right)_\eta\right).
\end{aligned} \tag{3.52}$$

Die einzelnen partiellen Ableitungen werden mit Hilfe der zentralen Differenzen approximiert, die allgemein lauten:

$$((\alpha x_\xi)_\xi)_{i,j} \approx \frac{(\alpha x_\xi)_{i+\frac{1}{2},j} - (\alpha x_\xi)_{i-\frac{1}{2},j}}{\Delta\xi} \tag{3.53}$$

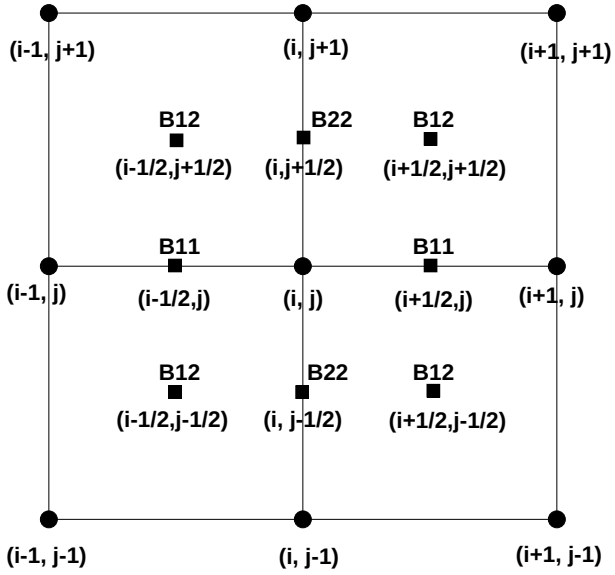
mit

$$(\alpha x_\xi)_{i+\frac{1}{2},j} \approx \alpha_{i+\frac{1}{2},j}\frac{x_{i+1,j} - x_{i,j}}{\Delta\xi} \tag{3.54}$$

Dies gilt analog für $i - \frac{1}{2}$-Werte und für die Ableitungswerte in η-Richtung unter Veränderung von j. Die gemischten partiellen Ableitungen werden durch die vier 'Halb'-Nachbarn approximiert, deren Durchschnitt genommen wird. Die rechte Seite wird entsprechend diskretisiert. Insgesamt entsteht eine komplexe nichtlineare Diskretisierungsformel, die die Werte

der Metriktensorkomponenten auf den Zwischenräumen der Gitterpunkte benutzt, aber die **x**-Werte nur auf den Gitterpunkten mit ganzzahligem Index. Dies soll die Zeichnung unten verdeutlichen.

Dieser Ansatz approximiert also (3.49). Eine Existenz- und Eindeutigkeitstheorie ist ein offenes Problem, da (3.51) nicht elliptisch ist. Der Iterationsprozess liefert für I_L, I_A und I_{AO} sinnvolle Ergebnisse, für I_O konvergiert er nicht. In [42] werden 11 verschiedene Realisierungen für die allgemeine Form des Integranden H im Funktional I_{MT}, (3.47), angegeben. Alle sind auch implementiert in Beispielprogrammen, die dem Buch auf Diskette beiliegen. Dort können Sie auf 18 Beispielgebiete angewendet werden, die auch noch vom Benutzer über Parameter verändert werden können.



Kapitel 4

Unstrukturierte Gittererzeugung

Für die unstrukturierte Gittererzeugung ist das Buch von *Du* und *Hwang* (eds.), [24], eine wichtige Grundlage. Es besteht aus acht Übersichtsartikeln von jeweils etwa 50 Seiten. Im folgenden verwenden wir insbesondere Darstellungen aus zwei dieser Artikel, dem von *Bern* und *Eppstein*, [8], und dem von *Fortune*, [27].

Für die Darstellung des Triangulierungsalgorithmus von *Ruppert*, seine Implementierung von *Shewchuk* und des Algorithmenvergleichs haben wir hauptsächlich deren Artikel [65], [69] und [70] mit ihren anschaulichen Beispiel-Graphiken zu Hilfe genommen.

Wir wollen in dieser Einleitung mit einigen grundsätzlichen Feststellungen beginnen, die teilweise eine Wiederholung und Anknüpfung an früher festgestelltes bedeuten.

Strukturierte Gitter haben folgende Vorteile gegenüber *unstrukturierten*:

(1) Sie sind einfacher.

(2) Sie sind besser geeignet für die gegenüber der FEM einfachere Differenzenmethode.

(3) Sie benötigen weniger Speicherplatz, weil die Koordinaten ihrer Gitterpunkte berechnet werden können, also nicht gespeichert werden müssen.

(4) Sie bieten mehr Kontrolle über die Größe und Gestalt der Elemente.

Der große Nachteil der strukturierten Gitter ist ihr Mangel an Flexibilität bei der Gittererzeugung für komplexe, sagen wir 'zerklüftete', Gebiete. Dieser Nachteil kann teilweise ausgeräumt werden durch Gebietszerlegung, einer Methode, die auch bei der verteilten numerischen Lösung von PDEs eine große Rolle spielt.

Die Tendenz in den letzten Jahren ging mehr zu den unstrukturierten Gittern. Dies gilt insbesondere für Arbeitsbereiche wie die CFD ('Computational Fluid Dynamics'), die sich z.B. mit der Berechnung der Umströmung von Flugkörpern und Maschinenteilen beschäftigt.

Auch hier werden oft die Gebiete oder Oberflächen aus vielen einzelnen Teilen zusammengesetzt, die dann zum Gesamtgebiet zusammengesetzt werden.

Die Anpassung an das Gebiet kann aber nicht allein gesehen werden, denn die Qualität der Gitter spielt bei der anschließenden Lösung der PDE eine große Rolle.

Unter Qualität versteht man die gekoppelte Erfüllung folgender Anforderung:

(1) An die Problemanforderung angepaßte *Elementgröße*, entweder gleichmäßig über das gesamte Gebiet oder adaptiv mit Steuerung durch Fehlerschätzer.

(2) Minimale *Elementezahl* bei Erfüllung sämtlicher Kriterien.

(3) *Elementeform:* Garantierte kleinste und/oder größte Winkel, maximaler Innenkreisradius bei beschränkter Maximalfläche, maximaler Aspect Ratio (kleinste Höhe/längste Seite).

(4) *Orientierung:* Ausrichtung der Elementeformen nach Anwendungsforderungen wie Strömungsrichtung.

Dementsprechend versteht man unter *Qualitäts-Netzgenerierung* (Quality mesh generation) Techniken, die für das quantitative Erfüllen einer oder mehrerer der genannten Anforderungen garantieren, also z.B.:

- Keine stumpfen Winkel!
- Ein garantierter Minimalwinkel! Ein Minimalwinkel α garantiert auch einen Aspect Ratio zwischen $|1/\sin(\alpha)|$ und $|2/\sin(\alpha)|$.

Unter *Qualitäts-Netzgenerierung* versteht man aber auch die Einhaltung einer oberen Schranke für die Anzahl der Zellen (Dreiecke) unter Einhaltung der quantitativen geometrischen Bedingungen. Ein Netz heißt *size-optimal*, wenn sich die Anzahl der erzeugten Zellen nur um einen konstanten Faktor von der minimal möglichen Anzahl Zellen irgendeiner Triangulierung, die dieselben Qualitätsmaße erfüllt, unterscheidet.

Zum Thema *strukturiert – unstrukturiert* noch ein Zitat von Henshaw, [36]:

> "Es ist schwer, auf unstrukturierten Gittern gute Performance zu erreichen. Man benötigt mehr Speicherplatz und gewisse schnelle Algorithmen wie die Mehrgittermethode oder implizite Methoden kann man kaum anwenden.
>
> ...
>
> Die Erzeugung von qualitativ hochwertigen Gittern für hyperbolische Probleme und die Berücksichtigung der Randschichten mit stark gestreckten Elementen bleibt ein aktives Forschungsfeld."

Beispiele, die zu sehr unterschiedlichen Gittern führen können:

(1) Zerlege ein gegebenes Gebiet (Polygon der Einfachheit halber) gleichmäßig in Elemente, d.h. jedes Element erfüllt eine Größenbedingung (z.B. Durchmesser nach oben und unten begrenzt).

(2) Zerlege ein gegebenes Gebiet (Polygon der Einfachheit halber) adaptiv in Elemente, d.h. die Lösung der PDE erfüllt in jedem Gitterpunkt eine Fehlerabschätzung.

(3) Trianguliere eine gegebene Punktmenge. Maximiere dabei den minimalen Winkel (Keine *Steiner-Punkte*!).

(4) Trianguliere eine Polygon mit n Ecken. Benutze keinen Winkel größer als 90 Grad und höchstens eine in n polynomiale Zahl von Steiner-Punkten.

Steiner-Punkte sind Gitterpunkte, die vom Algorithmus den gegebenen Punkten hinzugefügt werden.

Man kann zwischen *globaler* und *lokaler* Optimierung unterscheiden. Manchmal liefert eine lokal optimierte Triangulierung ein akzeptables globales Gitter.

Man kann ein Gitter iterativ erzeugen. Man beginnt mit der Triangulierung der gegebenen Punkte, die man mit der schrittweisen Hinzufügung von Steiner-Punkten lokal verbessert.

In der Forschung entsteht ein Konflikt zwischen dem strengen Beweis geometrischer Kriteria und heuristischen Algorithmen für die Praxis.

Bei zweidimensionalen Triangulierungen, auf die wir uns zunächst beschränken wollen, unterscheiden wir fünf geometrische Typen, die als Daten am Anfang der Triangulierung stehen können:

- Einfach zusammenhängende Polygone ('Simple polygon').
- Mehrfach zusammenhängende Polygone ('Polygon with holes').
- Punktmengen.
- Ebene Graphen, bestehend aus Ecken und nicht-schneidenden Kanten ('Planar straight line graph (PSLG)').
- Einfache Triangulierungen, die unter Einhaltung von Qualitätsvorgaben verfeinert werden.

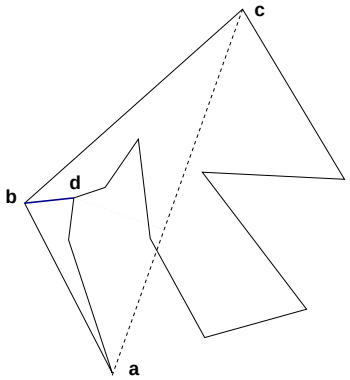
4.1 Delaunay-Triangulierungen ohne Steiner-Punkte

4.1.1 Existenz einer Triangulierung

Grundlage ist das

Lemma 4.1.1 Jedes einfach zusammenhängende Polygon mit mehr als drei Kanten hat eine Diagonale.

Beweis:
b sei eine Ecke mit minimaler x-Koordinate. ab und ac seien die in b endenden Kanten. Wenn ac keine Diagonale ist, d.h. wenn ac das Polygon schneidet, dann gibt es mindestens eine Ecke innerhalb des Dreiecks abc. Sei d die Ecke in abc, die den größten Abstand von ac hat. Die Strecke bd kann das Polygon nicht schneiden, sonst hätte die geschnittene Strecke einen Endpunkt, der weiter von ac entfernt ist als d. ■



Ein einfach zusammenhängendes Polygon kann in linearer Zeit trianguliert werden, die anderen Formen in $O(n \log n)$. Diese von Diagonalenzerteilung ausgehenden Triangulierungen können Starttriangulierung für Triangulierungen mit Optimierung geometrischer Eigenschaften sein.

Andere Fragen wie die, ob eine gegebene Menge von Dreiecken eine Triangulierung in unserem Sinne enthält, sind NP-komplex.

4.1.2 Voronoi-Diagramm und Delaunay-Triangulierung

Definition 4.1.2 (1) Sei S eine Menge von n verschiedenen Punkten im $\mathbb{R}^n$. Das *Voronoi-Diagramm* von S ist eine disjunkte Zerlegung des $\mathbb{R}^n$ in n Gebiete derart, dass jedem Punkt $s \in S$ die Teilmenge des $\mathbb{R}^n$ zugeordnet wird, deren Punkte näher an s liegen als an jedem anderen Punkt von S.

(2) Die *Delaunay-Triangulierung (DT)* von S besteht aus allen Verbindungslinien zwischen Punkten in benachbarten Gebieten des Voronoi-Diagramms, siehe Abbildung 4.1.

(3) Die Punktmenge S heißt *nicht degeneriert* (engl.: in general position), wenn nicht alle Punkte kollinear und nicht mehr als drei Punkte auf einem Kreis liegen. Letzteres ist gleichbedeutend damit, dass im Voronoi-Diagramm mindestens ein Knoten mit Grad > 3 existiert. Ein Gegenbeispiel sehen wir in Abbildung 4.2.

Lemma 4.1.3 Die Delaunay-Triangulierung von S ist die Triangulierung, bei der im Innern eines Umkreises eines Dreiecks kein Punkt von S liegt.

Zur graphischen Veranschaulichung dieses Lemmas siehe Beispiel 4.1.6.

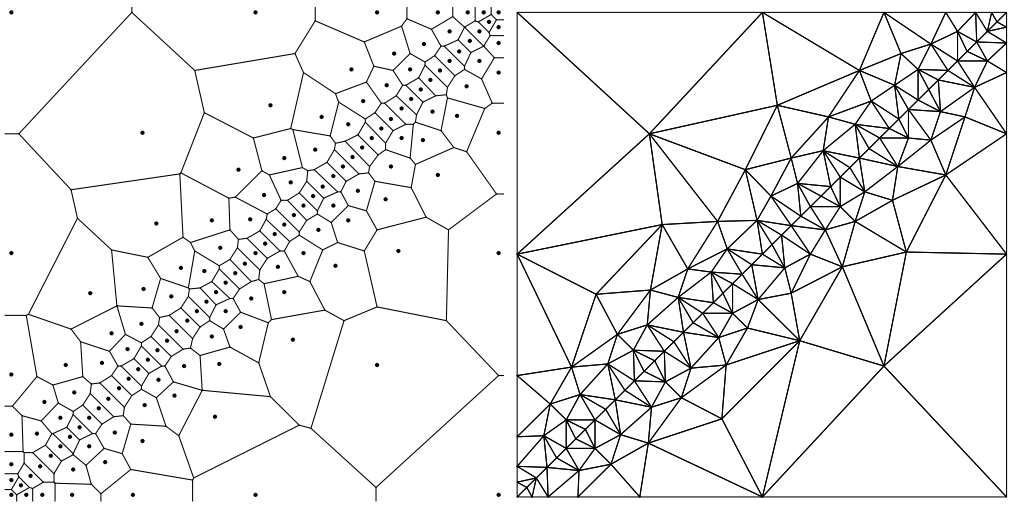


Abbildung 4.1: Voronoi-Diagramm und Delaunay-Triangulierung von 142 Punkten

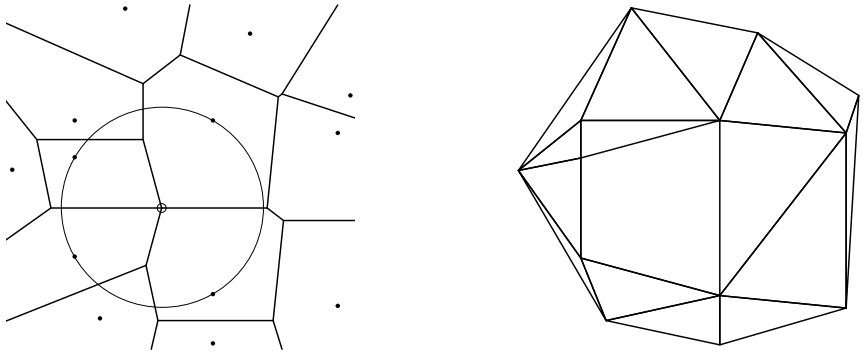


Abbildung 4.2: Degeneriertes Voronoi-Diagramm mit Delaunay-Triangulierung

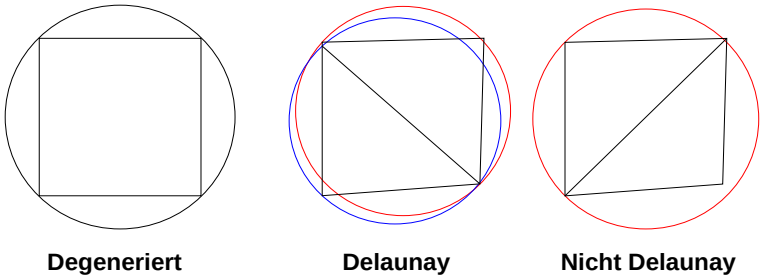
Definition 4.1.4 (1) Zwei benachbarte Dreiecke heissen *lokal Delaunay*, falls ihre vier Punkte die Bedingung des Lemmas 4.1.3 erfüllen.

(2) Eine Triangulierung heißt *lokal Delaunay*, falls jedes Paar benachbarter Dreiecke lokal Delaunay ist.

Lemma 4.1.5 Eine Triangulierung ist Delaunay genau dann, wenn sie lokal Delaunay ist.

Beweisskizze: Die eine Richtung ist trivial. Sei also die Triangulierung lokal Delaunay: Wähle ein Dreieck D und einen Punkt s, der nicht zu D gehört. Jetzt wird eine Folge von Dreiecken konstruiert, die D mit s verbinden. Je zwei von ihnen müssen lokal Delaunay sein und daraus wird gefolgert, dass s nicht im Umkreis von D liegen kann. Der Beweis benutzt die Tatsache, dass die DT die einzige Triangulierung mit leeren Umkreisen ist.
■

Beispiel 4.1.6 Zu den Lemmata 4.1.3 und 4.1.5 hier noch ein einfaches Beispiel, das die Eindeutigkeit der DT im Sinne dieses Lemmas demonstrieren soll:



Bemerkung 4.1.7 Es gibt einen interessanten Zusammenhang zwischen der Delaunay-Triangulierung und einem Paraboloid im $\mathbb{R}^3$. Für einen Punkt $(x_1, x_2) \in \mathbb{R}^2$ berechnen wir einen korrespondierenden Punkt im $\mathbb{R}^3$ mit der Abbildung (*'lifting map'*)

$$\lambda(x) = (x_1, x_2, x_1^2 + x_2^2). \tag{4.1}$$

Alle Punkte dieser Abbildung liegen auf dem Paraboloid

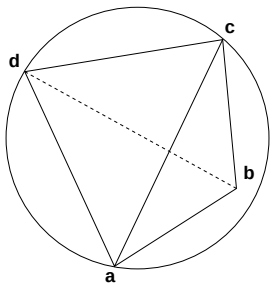
$$\Lambda = \{(x_1, x_2, z) \,\Big|\, z = x \cdot x, x \in \mathbb{R}^2\}. \tag{4.2}$$

Wenn H die konvexe Hülle von $\lambda(S)$ ist, dann ist die DT von S die Projektion der nach ihrer Höhe geordneten und verbundenden Dreiecke vom $\mathbb{R}^3$ (zurück) in den $\mathbb{R}^2$.

Definition 4.1.8 Eine Triangulierung heißt *konform*, wenn die Schnittmenge zweier Dreiecke entweder eine Ecke oder eine Kante oder die leere Menge ist.

Lemma 4.1.9 Unter allen konformen Triangulierungen einer nicht-degenerierten Punktmenge $S \in \mathbb{R}^2$ maximiert die Delaunay-Triangulierung den kleinsten Winkel jedes Dreiecks.

Beweisskizze: Wir nehmen an, dass die konforme Triangulierung, die den maximalen kleinsten Winkel liefert, nicht Delaunay ist. Dann folgt aus Lemma 4.1.5, dass sie auch nicht lokal Delaunay ist. Dann gibt es also zwei benachbarte Dreiecke *abc* und *acd*, von denen ein Eckpunkt des einen im Umkreis des anderen ist, sagen wir: *b* ist im Umkreis von *acd*. *abcd* ist notwendigerweise eine konvexes Viereck. Nun kann man mit leichten geometrischen Überlegungen zeigen, dass der kleinste Winkel der Dreiecke *abd* und *bcd* größer ist als kleinste Winkel der Dreiecke *abc* und *acd* und das führt unsere Annahme zum Widerspruch.



■

Wenn die Punktmenge degeneriert ist, entstehen in der Delaunay-'Triangulierung' polygonale Elemente, also z.B. Vierecke wie in Abbildung 4.2. Diese müssen durch Diagonalen trianguliert werden. Für diese zusätzlichen Triangulierungsschritte gibt es einfache Algorithmen, die die Optimalitätseigenschaft der DT erhalten.

Lemma 4.1.10 Unter allen konformen Triangulierungen einer nicht-degenerierten Punktmenge $S \in \mathbb{R}^2$ minimiert die Delaunay-Triangulierung den größten Innenkreis.

Beweisidee: Für diesen Beweis benutzt man die oben kurz dargestellte Beziehung der DT mit dem 3D-Paraboloid Λ.

■

Zur Gittererzeugung sind in der Regel keine Punktmengen, sondern geschlossene Polygone vorgegeben. Die DT der Punktmenge, die aus den Ecken des Polygons besteht, kann eine Gebietskante schneiden, wenn das Gebiet nicht konvex ist. Deshalb benötigen wir eine Verallgemeinerung der Definition. Hierzu gibt es mehrere Möglichkeiten:

- *Die konforme Delaunay-Triangulierung* (KDT), bei der die Punktmenge so vergrößert wird, dass die DT der vergrößerten Punktmenge das Polygon abdeckt. Auf konforme DTs kommen wir später zurück.
- Bei der *beschränkten Delaunay-Triangulierung* (CDT) wird die Definition der Delaunay-Triangulierung so verallgemeinert, dass gewisse Kanten (des Randes) in die DT gezwungen werden.

Definition 4.1.11 (1) Ein Punkt b heißt *sichtbar* von a, wenn die Kante ab keine Kante des Polygons schneidet.

(2) Die beschränkte Delaunay-Triangulierung enthält die Kante ab zwischen zwei Eingabe-Ecken genau dann, wenn a von b sichtbar ist, und wenn es einen Kreis gibt, der durch a und b geht und keinen Eingabe-Punkt c in seinem Inneren enthält, der von der Kante ab sichtbar ist.

Diese Definition der Sichtbarkeit *(visibility)* kann man auf PSLGs verallgemeinern, indem man das Wort Polygon durch Graph ersetzt. Dann ist eine Punktmenge ein Graph ohne Kanten, und die CDT ist mit der DT identisch.

Wie die DT ergibt die CDT eine konforme Triangulierung für eine nicht-degenerierte Eingabe (Polygon oder PLSG):

Lemma 4.1.12 Wenn keine vier Eingabe-Punkte einer PLSG auf einem gemeinsamen Kreis liegen, dann ist die CDT der PLSG eine Triangulierung.

Auf den nicht ganz kurzen Beweis wollen wir verzichten.

Um die CDT einer nicht degenerierten PLSG zu berechnen, gibt es mehrere Algorithmen. Wir wollen zunächst auf zwei geometrische Test eingehen, die Grundmodule der meisten Algorithmen sind. Dann wollen wir den Flip-Algorithmus beschreiben, der mit einem Aufwand von $O(n^2)$ nicht zu den schnellstmöglichen gehört. Dafür ist er einfach zu verstehen und zu programmieren, und ist auch schnell genug für mittelgroße Datenmengen. Anschließend erwähnen wir einige der $O(n \log n)$-Algorithmen. Dabei beschreibt die Komplexität immer den 'worst case'. Man sollte Algorithmen auch nach der Komplexität ihrer Datenstruktur und ihrer Implementation beurteilen. Außerdem kann es sein, dass die Komplexität geringer wird bei Zufallsdaten im Vergleich zu 'worst case'-Daten.

4.1.3 Geometrische Tests als Basis-Algorithmen

Der Orientierungstest

Seien $p_1 = (p_{11}, p_{12})^T$, $p_2 = (p_{21}, p_{22})^T$ und $p_3 = (p_{31}, p_{32})^T$ drei Punkte in der Ebene. Wir definieren ihre Orientierung als Vorzeichen der Determinante

$$D_1 := \begin{vmatrix} p_{11} & p_{12} & 1 \\ p_{21} & p_{22} & 1 \\ p_{31} & p_{32} & 1 \end{vmatrix}. \tag{4.3}$$

Man sieht leicht, dass

$D_1 > 0$, falls $\{p_1,\ p_2,\ p_3\}$ ein nichtleeres Dreieck bilden mit gegen den Uhrzeigersinn numerierten Ecken,
$D_1 < 0$, falls $\{p_1,\ p_2,\ p_3\}$ ein nichtleeres Dreieck bilden mit im Uhrzeigersinn numerierten Ecken,
$D_1 = 0$, falls $\{p_1,\ p_2,\ p_3\}$ kollinear sind.

Der Inkreis-Test

Für den zweiten geometrischen Test wollen wir annehmen, dass drei Punkte $\{p_2,\ p_3,\ p_4\}$ mathematisch orientiert und nicht kollinear sind, also $D_1(p_2,\ p_3,\ p_4) > 0$. Dann liegt der Punkt p_1 außerhalb des Umkreises von $\{p_2,\ p_3,\ p_4\}$, falls $\lambda(p_1)$ oberhalb der Hyperebene liegt, die von $\{\lambda(p_2),\ \lambda(p_3),\ \lambda(p_4)\}$ aufgespannt wird, mit λ nach (4.1), und das ist genau dann der Fall, wenn die Orientierung von $\{\lambda(p_1),\ \lambda(p_2),\ \lambda(p_3),\ \lambda(p_4)\}$ positiv ist oder

$$D_2 := \begin{vmatrix} p_{11} & p_{12} & p_{11}^2 + p_{12}^2 & 1 \\ p_{21} & p_{22} & p_{21}^2 + p_{22}^2 & 1 \\ p_{31} & p_{32} & p_{31}^2 + p_{32}^2 & 1 \\ p_{41} & p_{42} & p_{41}^2 + p_{42}^2 & 1 \end{vmatrix} > 0. \tag{4.4}$$

D_2 wird genau dann Null, wenn die Punkte $p_1,\ p_2,\ p_3,\ p_4$ auf einem Kreis liegen. Die Bestimmung des Vorzeichens von D_2 ist gerade der *Inkreis-Test*.

4.1.4 Eine Datenstruktur

Eine mögliche Datenstruktur ist folgendermaßen aufgebaut: Jedem Dreieck wird ein *Knoten* zugeordnet, der das Dreieck repräsentiert. Der Knoten enthält folgende Daten:

- Einen Indexvektor für die drei Eckpunkte.
- Einen Zeiger-Vektor, der auf die Nachbarknoten der gegenüberliegenden Kante zeigt. Randkanten sind durch Null-Zeiger gekennzeichnet.

4.1.5 Der Flip-Algorithmus

- Bestimme eine beliebige Triangulierung des gegebenen PSLG.
- Durchlaufe alle Kanten der Triangulierung, die nicht Eingabe-Kanten sind, und die nicht auf der konvexen Hülle des PSLG liegen:
  - Die beiden Dreiecke, zu denen die Kante e gehört, bilden ein Viereck Q_e. e ist eine Diagonale dieses Vierecks.
  - Wenn die Kante e nicht zur Delaunay-Triangulierung der vier beteiligten Punkte gehört, wird sie gegen die andere Diagonale des Vierecks ausgetauscht (*flip*).

Dieser Algorithmus nutzt offenbar Lemma 4.1.5 aus. Ohne Beweis wollen wir die folgenden Eigenschaften angeben:

Lemma 4.1.13
- Wenn keine Viereckdiagonalen mehr vertauscht werden müssen, ist die entstandene Triangulierung die CDT.
- Der Flip-Algorithmus ist nach $O(n^2)$ Vertauschungen beendet.
- Unter allen Triangulierungen eines PSLG
  (1) minimiert die CDT den größten Umkreis;
  (2) minimiert die CDT den größten Innenkreis;
  (3) maximiert die CDT den kleinsten Winkel der Triangulierung.

4.1.6 Chew's 'Teile-und-herrsche'-Algorithmus

Chew hat 1989 gezeigt, dass folgender Satz gilt:

Satz 4.1.14 Die CDT eines PSLG kann in $O(n \log n)$ Zeit konstruiert werden.

Beweis-Algorithmus-Skizze:
Chew sortiert die Menge der Eingabepunkte nach ihren x-Koordinaten und teilt diese Menge durch eine vertikale Linie in zwei Teilmengen mit jeweils maximal $[n/2]$ Punkten. Die CDT wird jetzt für jede Teilmenge berechnet und dann zusammen gemischt.

Um eine Komplexität von $O(n \log n)$ zu erreichen, teilt der Algorithmus einen vertikalen Streifen weiter mit Schnittkanten in Regionen. Die CDT wird nur für solche Regionen berechnet, die mindestens einen Punkt enthalten.

Das Zusammenmischen geschieht jetzt Regionen-weise. Das Mischen kann dadurch erfolgen, dass man Kreise um die Schnittkanten schlägt und dadurch auszutauschende Kanten bestimmt.

Es gibt Varianten, die die Komplexität dieses Algorithmus auf $O(n \log \log n)$ oder sogar auf $O(n)$ herunterdrücken, siehe Fortune und die dort zitierte Literatur.

Beispiel 4.1.15 Als Beispiel nehmen wir das Polygon, das durch folgende Punktmenge und ihre Reihenfolge definiert ist:

x	0.0	2.0	6.0	11.0	17.0	13.5	9.5	4.0
y	6.1	1.7	0.0	6.1	3.0	10.0	8.5	11.5

Die vertikale Teilung und CDT der beiden Halbmengen sieht man links in Abbildung 4.3. Am Umkreis des linken oberen Dreiecks sieht man schon die Problematik des Mischens, bei dem zwei Flips erforderlich sind, das Ergebnis sehen wir rechts in Abbildung 4.3.

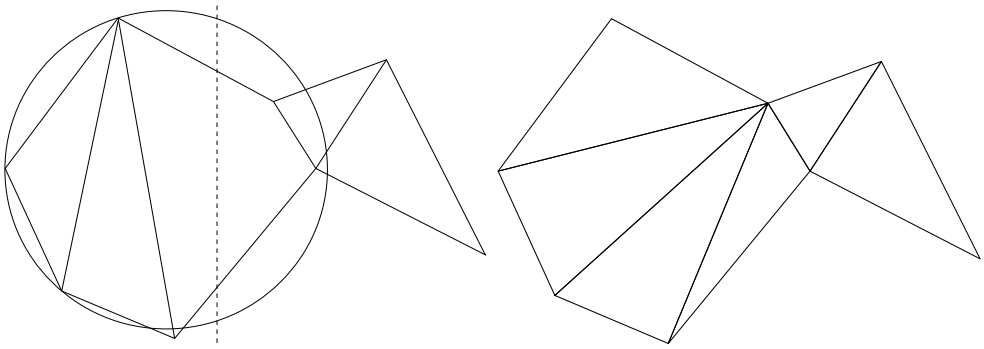


Abbildung 4.3: Chews Algorithmus

Zum Divide-and-Conquer-Algorithmus gibt es mehrere (strategische) Varianten mit unterschiedlichem Laufzeitverhalten. Sie werden von Shewchuk verglichen. In diesen Vergleich werden auch zwei andere Algorithmen einbezogen, die wir hier nur kurz erwähnen wollen:

- *Incremental insertion*: Die globale DT wird erzeugt durch sukzessive Hinzunahme von Punkten und jeweils lokaler DT. Dreiecke werden entsprechend der Umkreisbedingung (auf den neuen Punkt bezogen) ausgetauscht. Auf diese Weise entsteht nach und nach die gewünschte DT.

- *Plane sweep*: Eine Linie – die sweepline – wird über die Ebene geführt. Über den Abstand naher Umkreise von dieser Linie wird die DT bestimmt. Dieser Algorithmus benötigt nur $O(n \log n)$ Zeit und $O(n)$ Speicherplatz. Details und Beweise sind nicht trivial.

Shewchuk hat diese Algorithmen verglichen, er hat außerdem Versionen zweier Algorithmen in exakter Arithmetik entwickelt und diese in den Vergleich einbezogen. Der zusätzliche Aufwand für die exakte Arithmetik ist erstaunlich gering im Vergleich zu dem Vorteil, dass keine ungültige DT aufgrund von Rundungsfehlern auftreten kann. Ein Beispiel werden wir unten sehen.

Der Vergleich ist auch für verfeinerte (optimierte) DTs gültig, das sind DTs, die durch zusätzliche Steiner-Punkte in der Lage sind, Qualitätsbedingungen zu erfüllen.

4.2 Delaunay-Triangulierungen mit Steiner-Punkten

Die Ausgangssituation ist dieselbe wie im letzten Abschnitt. Ausgangspunkt ist ein gegebener PSLG, der in unseren Anwendungen meist ein einfach oder mehrfach zusammenhängendes Polygon ist. Die gesuchte Triangulierung muß auch hier Punkte und Kanten der Eingabe konform überdecken. Erlaubt ist allerdings eine Unterteilung der Kanten und die Hinzufügung von Punkten mit dem Zweck, die geometrische Qualität der DT uniform oder adaptiv zu verbessern oder geometrische Bedingung wie 'kleinster Winkel $>$ 20 Grad' zu erfüllen.

Die theoretische Beziehung zwischen der Anzahl der Steiner-Punkte und der Optimalität ist nicht ganz klar. Man kann den kleinsten Winkel beliebig nahe an 60 Grad bringen, kann dann aber die Anzahl der Steiner-Punkte nicht beschränken.

Deshalb wird man entweder die Anzahl Steiner-Punkte beschränken und unter dieser Bedingung ein geometrisches Maß optimieren, oder man wird ein geometrisches Maß quantitativ vorgeben und unter dieser Bedingung die Anzahl Steiner-Punkte minimieren. Beide Formulierungen führen zu fast unlösbaren Problemen für den allgemeinen Fall.

Sinnvoll erscheinen unter diesen Bedingungen Näherungsalgorithmen. Das sind Algorithmen, die ein bestimmtes Maß bis auf einen konstanten Faktor einhalten und dabei size-optimal sind. Der Vorteil von Approximationen ist darüberhinaus, dass eine gleichzeitige Erfüllung mehrerer geometrischer Bedingungen möglich wird.

4.2.1 Keine kleinen Winkel

Ohne Steiner-Punkte maximiert die DT den kleinsten Winkel. Und diese ist in $O(n \log n)$ berechenbar. Ein normales Beispiel ohne und mit Steiner-Punkten und verschiedenen Triangulierungen sehen wir in Abbildung 4.4 aus Ruppert. Mit Steiner-Punkten kann die Bedingung eines festen minimalen Winkels allerdings zu einem NP-komplexen Problem werden, wie man an Beispiel 4.2.2 von Shewchuk sieht. Es gilt nämlich:

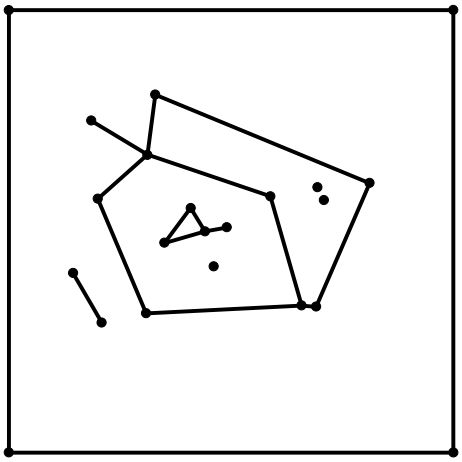
Lemma 4.2.1 Zu jeder Winkel-Bedingung $\varphi \geq \bar{\varphi} > 0$ gibt es ein PSLG P derart, dass es nicht möglich ist, P zu triangulieren ohne einen Steiner-Punkt zu erzeugen, an dem die Winkelbedingung verletzt ist.

Beweis: Als PSLG P geben wir zwei kollineare Kanten und einen Punkt, der die beiden Kanten im Verhältnis $b : a$ trennt, vor, siehe Abbildung 4.5. Damit die Triangulierung eines solchen PLSG mit Steiner-Punkten einen minimalen Winkel von $\bar{\varphi}$ haben kann, muß gelten

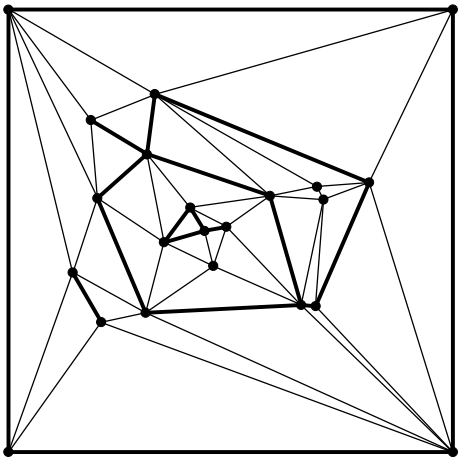
$$\frac{b}{a} \leq (2 \cos \bar{\varphi})^{180^0/\bar{\varphi}}. \tag{4.5}$$

Diese Ungleichung hat Mitchell bewiesen, siehe Shewchuk. Zu jedem Winkel $\bar{\varphi}$ gibt es also ein PSLG P in der angegebenen Form, der gegen Mitchells Bedingung verstößt. ■

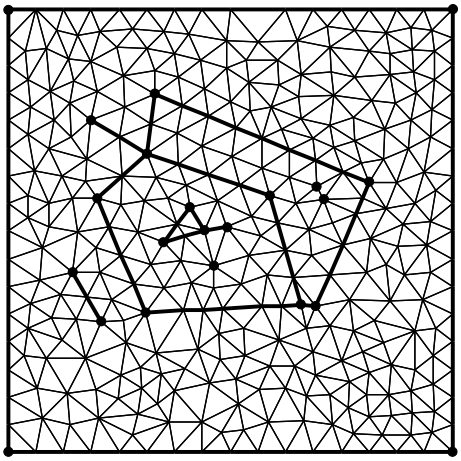
Abbildung 4.5 zeigt ein Beispiel mit $\bar{\varphi} = 30^0$, was zu einem maximalen Seitenverhältnis von 27 führt. Um eine Winkel-Bedingung einhalten zu können, darf also in der gesamten Triangulierung keine Seitenverhältnis entstehen, das gegen (4.5) verstößt. Daß ein solch verbotenes Seitenverhältnis auch durch einen ungünstigen Eingabewinkel zwangsläufig entstehen kann zeigt Beispiel 4.2.2.



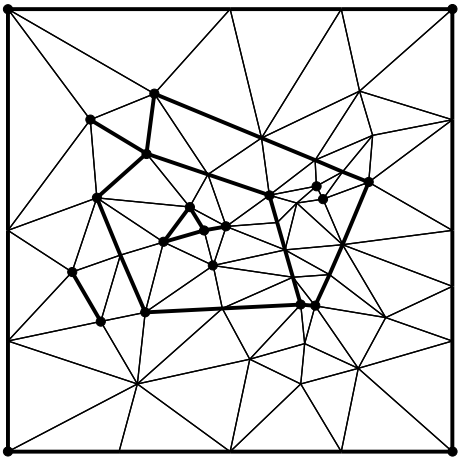
(a) PSLG in einer Box als Eingabe.



(b) Triangulierung ohne Steinerpunkte. Einige kleine Winkel sind unvermeidbar.



(c) Homogenes Netz mit einem kleinsten Winkel von 22.5 Grad.



(d) Delaunay−Verfeinerungsalgorithmus von Ruppert mit einem kleinsten Winkel von 20 Grad.

Abbildung 4.4: Verschiedene Triangulierungen einer Eingabe-PLSG

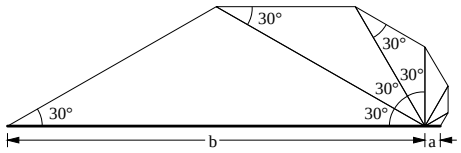


Abbildung 4.5: Erzeugung eines Fächers durch ein ungünstiges Kantenverhältnis

Beispiel 4.2.2 Eingabemenge ist ein PLSG mit einem kleinen Winkel α zwischen zwei Eingabekanten und einem Eingabepunkt p auf einer dieser Kanten, siehe Abbildung 4.6. Der kleine Winkel kann nicht verbessert werden, aber es soll kein weiterer kleiner Winkel entstehen, sondern für Steiner-Punkte die Winkelbedingung eingehalten werden. Das schmale Dreieck muß vom Punkt p aus trianguliert werden, was unter der Winkelbedingung zur Einfügung mindestens eines Steiner-Punktes q führt.

Es sei jetzt $a = |\overline{pq}|$ und $b = |\overline{op}|$. Um die Winkelbedingung mit einem Winkel $\bar{\varphi}$ einzuhalten, darf a nicht zu groß sein, es muß gelten:

$$\frac{a}{b} \leq \frac{\sin(\alpha)}{\sin(\bar{\varphi})} \left(\cos(\bar{\varphi} + \alpha) + \frac{\sin(\bar{\varphi} + \alpha)}{\tan(\bar{\varphi})} \right). \quad (4.6)$$

Wenn das Gebiet oberhalb der Eingabekante mit dem Punkt p Teil des zu triangulierenden Gebiets ist, tritt jetzt der Fächereffekt aus Abbildung 4.5 ein. Er macht die Einfügung eines Punktes r zwischen o und p notwendig.
r erzeugt jetzt näher an o dieselbe Situation wie vorher p. Diese Prozedur setzt sich unter entsprechenden geometrischen Bedingungen unendlich fort, wobei immer kleinere Dreiecke erzeugt werden, deren Zahl also unbeschränkt wächst.
Dieser Effekt ist unvermeidbar, wenn das Produkt der beiden oberen Schranken aus den Gleichungen (4.5) und (4.6) kleiner als 1 ist. Für $\bar{\varphi} = 30^0$ ist das der Fall, wenn $\alpha \leq 0.6^0$.

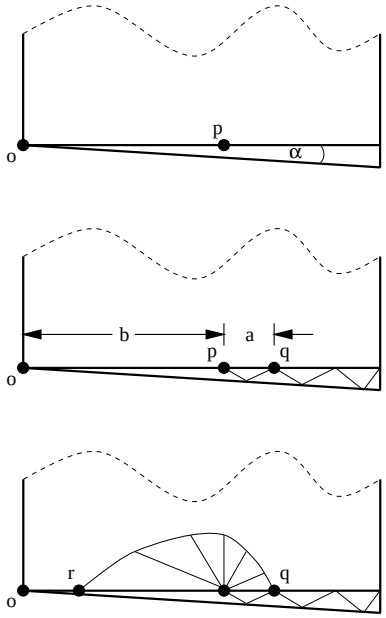


Abb. 4.6: # Dreiecke $\rightarrow \infty$

An den Beispielen sieht man, dass praktische Schwierigkeiten nur in Extremsituationen zu erwarten sind, und die Frage bleibt, was man mit sinnvollen Voraussetzungen erreichen kann. Chew hat 1989 einen solchen Algorithmus angegeben.

Voraussetzung 4.2.3 Gegeben sei ein PSLG oder ein Polygon. s sei die *input feature size*, das ist der kürzeste Abstand zwischen einem Punkt und einem anderen Punkt oder einer disjunkten Kante, der oder die von diesem Punkt aus sichtbar ist. Dabei gehören die Endpunkte von Eingabekanten auch zu den betrachteten Punkte. Für diese Größe s gelte:
Alle Eingabekanten sind nicht kürzer als s und nicht länger als $\sqrt{3}\, s$.

Diese Voraussetzung kann man immer durch Kantenteilungen erreichen.

Chews Algorithmus:

(1) Berechne die CDT der Eingabemenge.

(2) Wenn es ein Dreieck gibt, dessen Umkreisradius größer als s ist, dann füge den Umkreismittelpunkt der Menge der Steinerpunkten hinzu und berechne von der so erweiterten Menge die CDT.

Satz 4.2.4 Chews Algorithmus berechnet eine Steiner-Triangulierung, in der alle Winkel zwischen 30^0 und 120^0 liegen.

Beweisskizze: Es wird niemals ein Punkt hinzugefügt, der näher als s an einem schon vorhandenen Punkt liegt. Der Algorithmus bricht deshalb auch nach endlich vielen Schritten ab, da die CDT der Eingabemenge beschränkt ist. Am Ende haben alle Umkreise einen Maximalradius s, und alle Kanten haben eine Länge zwischen s und $2\,s$. Daraus folgt mit einfachen geometrischen Überlegungen der Satz. ■

Eine verbesserte Version dieses Algortihmus, die auch Chew selbst angab, berechnet eine uniforme Triangulierung in $O(n+k)$-Zeit, wenn k die Anzahl der Steiner-Punkte ist. Diese ist beschränkt durch die Komplexität einer Triangulierung, in der alle Dreiecke eine Seitenlänge der Größenordnung $O(s)$ haben. Das heißt, dass Chews Algorithmus zufriedenstellend für eine uniforme Triangulierung ist, was sich auch in Experimenten zeigt.

Beispiel 4.2.5 Wir wollen ein einfaches Beispiel betrachten. In Abbildung 4.7 sehen wir eine Folge von PSLG mit folgenden Eigenschaften (von links nach rechts):

(1) Die Bedingung $s_{max} \leq \sqrt{3}\, s_{min}$ ist verletzt.

(2) Die Bedingung $s_{max} \leq \sqrt{3}\, s_{min}$ ist nach Einfügen von vier Punkten erfüllt.

(3) CDT des verbesserten Eingabe-PSLG. Einfügung eines Steiner-Punktes wegen Umkreisradius größer s notwendig.

(4) Neue CDT.

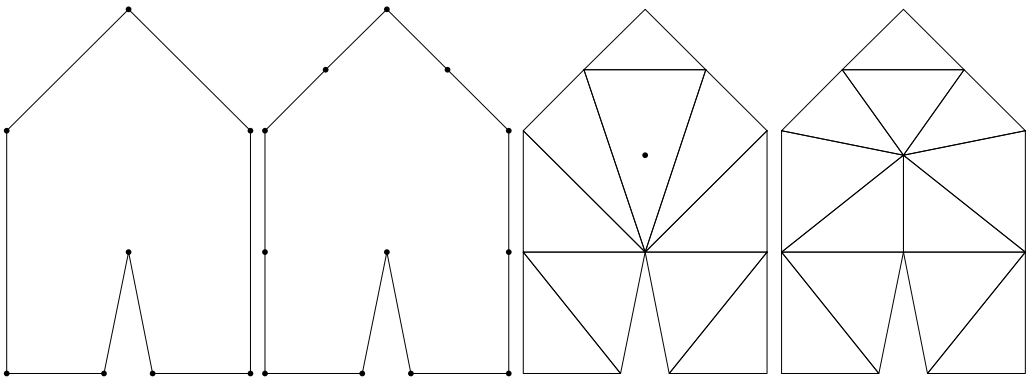


Abbildung 4.7: Das Haus des Nikolaus mit Steiner-Punkten

In vielen praktischen Situationen ist die schnelle Berechnung einer uniformen Triangulierung nicht ausreichend, weil adaptiv verfeinerte Triangulierungen mit starken Größenunterschieden bei den Dreiecken notwendig sind. Das trifft besonders auf das Gebiet CFD zu mit Problemen wie Schockwellen, umströmten Gebieten usw.

Bern, Eppstein und Gilbert gaben als erste für solche Triangulierungen Gitter optimaler Komplexität an, siehe [9]. Ihre 'beweisbar guten' Algorithmen arbeiten mit Quadtrees, auf die wir im nächsten Abschnitt zurückkommen werden.

Wir wollen hier noch den Delaunay-Verfeinerungs-Algorithmus von Ruppert in der Version von Shewchuk angeben, der auch sehr gute Gitter mit allen möglichen Bedingungen erzeugt, und der im Programm `triangle` verwendet wird. Dieses Programm wurde von Shewchuk entwickelt und ist Lizenz-frei zu erhalten. Wir folgen bei der Schilderung des Algorithmus deshalb auch Shewchuk und seinen anschaulichen Beispielen.

Der Algorithmus arbeitet mit DTs der Punkte von PSLGs. Um dabei keine Kanten des PSLG wegen der Delaunay-Eigenschaft zu verlieren, werden solche Kanten solange durch Einfügen von Mittelpunkten aufgeteilt, bis die DT der neuen Punktmenge die Eingabekante als Vereinigung mehrerer Kanten enthält. Dieser Teilalgorithmus ist *Lawsons incremental insertion algorithm*. Ein einfaches Beispiel sehen wir in Abbildung 4.8 von Shewchuk.

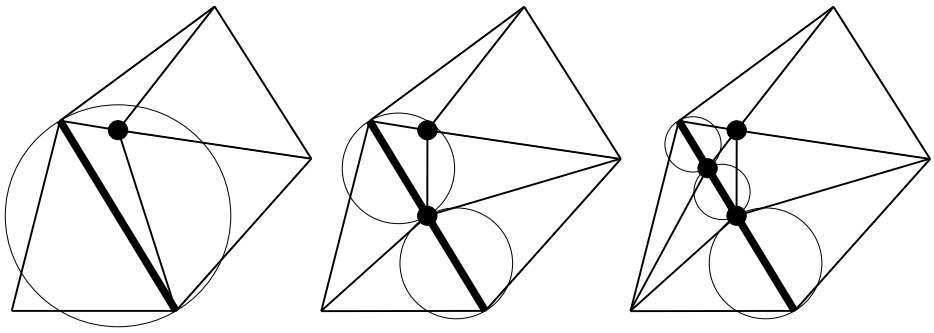


Abbildung 4.8: Teilung einer Eingabekante mit Erhalt der Delaunay-Eigenschaft

Rupperts Delaunay-Verfeinerungs-Algorithmus in der Triangle-Version

Eingabemenge ist ein PSLG, d.h. eine Menge aus Punkten und Kanten, wobei die Kantenendpunkte der Menge der Punkte hinzugerechnet werden. Die einzelnen Schritte werden anschließend durch das Gitarrenbeispiel von Shewchuk veranschaulicht.

(1) Bestimme die DT der Eingabepunkte.

(2) **Entweder:**
Erzwinge die Einfügung der Eingabekanten, die nicht in der DT der Eingabepunkte enthalten sind, durch Hinzunahme der Mittelpunkte solcher Kanten und Anwendung von Lawsons incremental insertion algorithm. So bleibt die Delaunay-Eigenschaft erhalten.

Oder:
Berechne die CDT der Eingabe-PSLG. Eingabekanten kommen dadurch in die Triangulierung, dass man alle Dreiecke, die solche Kanten schneidet, wegnimmt, und die Gebiete auf beiden Seiten der Eingabekanten neu trianguliert. Im Programm `triangle` hat der Benutzer die Wahl zwischen diesen beiden Alternativen.

(3) Jetzt werden alle Dreiecke aus konkaven Teilgebieten, die nicht zum Eingabegebiet gehören sollen, und aus Löchern bei mehrfach zusammenhängenden Gebieten entfernt.

`triangle` verwendet hier im Unterschied zu Ruppert einen 'Dreiecksfresser-Virus', der ausgehend von einem vom Benutzer einzugebenden Punkt mit 'depth-first-search' alle Dreiecke um diesen Punkt herum entfernt, bis er von einer Eingabekante aufgehalten wird.

Diese Vorgehensweise ist einfach zu implementieren, und sie ist effektiver als die übliche, geometrisch 'korrekte' Methode.

(4) Verfeinerung des Gitters durch Hinzufügen von Steiner-Punkten. Dabei wird wieder zusätzlich Lawsons incremental insertion algorithm angewendet, damit die Delaunay-Eigenschaft erhalten bleibt.

Die Punkteinfügung geschieht nach zwei Regeln:

- Der *Durchmesserkreis* einer Kante ist der eindeutige kleinste Kreis, der die Kante enthält. Eine Kante heißt 'encroached' (durchdrungen), wenn ihr Durchmesserkreis einen Punkt enthält. In diesem Fall wird ihr Mittelpunkt als Steiner-Punkt eingefügt, siehe Abbildung 4.8.
- Ein Dreieck heißt *schlecht (bad)*, wenn sein kleinster Winkel oder seine Fläche gegen die Nebenbedingung des Benutzers verstößt. Schlechte Dreiecke werden aufgeteilt durch Einfügen des Umkreis-Mittelpunktes als Steiner-Punkt. Die Delaunay-Eigenschaft garantiert dann, dass das schlechte Dreieck in der neuen Triangulierung nicht mehr enthalten ist. Wenn der neue Steiner-Punkt eine Kante durchdringt (encroached), dann wird er wieder entfernt und alle Kanten, die er durchdringt, werden unterteilt (halbiert, ...).

Die Bedingung *durchdrungen* hat also Vorrang vor der *schlechtes Dreieck*. Für beide Bedingung gibt es eine Queue verletzter Elemente. Die Queues werden abgearbeitet, wobei die Hinzufügung eines Steiner-Punktes neue Elemente an eine Queue anhängen kann.

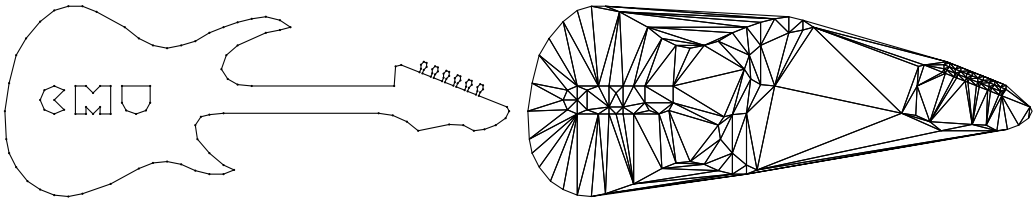


Abbildung 4.9: Die Eingabe-PSLG und ihre DT, die nicht alle Eingabekanten enthält.

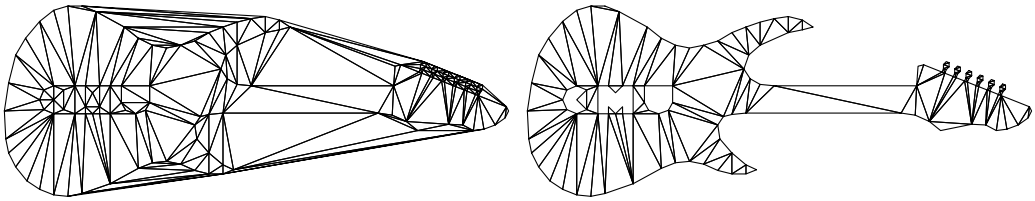


Abbildung 4.10: CDT der Eingabekanten und CDT nach Entfernung von Dreiecken in Löchern und konkaven Außenregionen

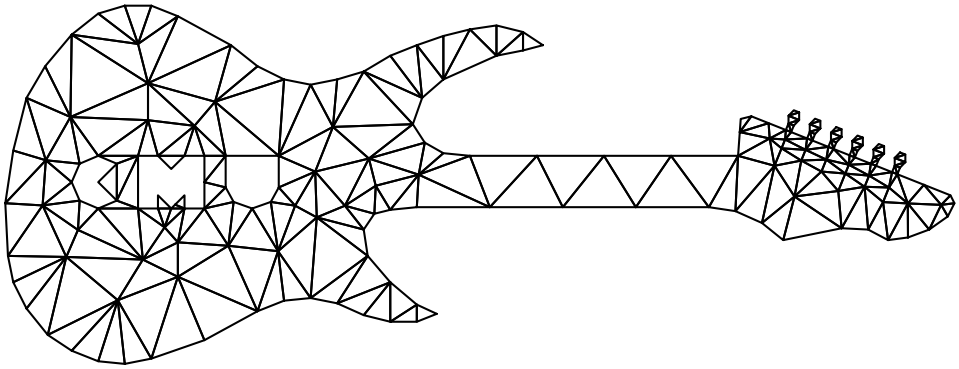
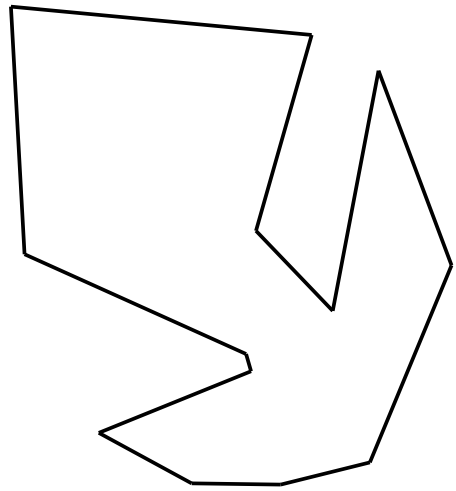


Abbildung 4.11: CDT mit einem kleinsten Winkel von 20 Grad.

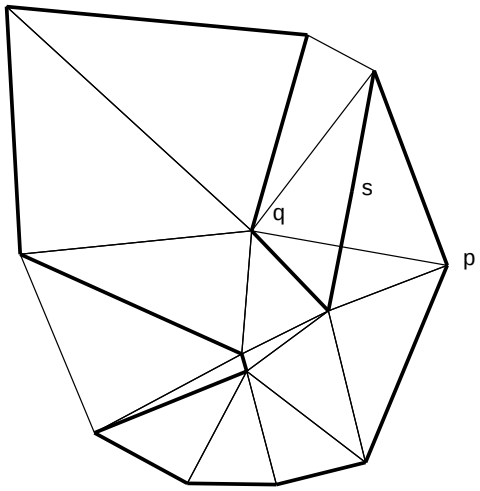
Zwei Beispiele sollen den Verfeinerung-Algorithmus erläutern. Rupperts Beispiel, Abbildungen 4.12 und 4.13, erläutert sich selbst. Shewchuks Beispiel besteht aus sechzehn Schritten, die in Abbildung 4.14 dargestellt sind. Die ersten beiden Bilder zeigen den Eingabe-PSLG und seine CDT. In jedem der folgenden zwölf Bilder wird ein Verfeinerungsschritt an dem im vorherigen Bild geschwärzten Element vollzogen.

Shewchuk geht noch auf Implementierungseinzelheiten und Alternativen ein. Darauf wollen wir verzichten bis auf die schon erwähnte robuste exakte Gleitpunktarithmetik. Für den Verzicht auf diese Eigenschaft wollen wir ein Beispiel aufgreifen, das die Wirbel der Gitarre mit und ohne exakte Arithmetik zeigt, siehe Abbildung 4.15.

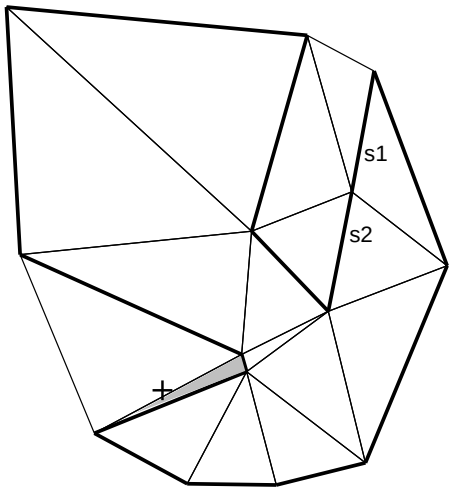
Ruppert geht in seinem langen Artikel ausführlich auf Details seines Algorithmus ein. Er beweist einige geometrische Eigenschaften der Verfeinerungsschritte (minimale Abstände etc.) und die Größen-Optimalität. Dabei stellt er selbst fest, dass die hierbei auftretenden



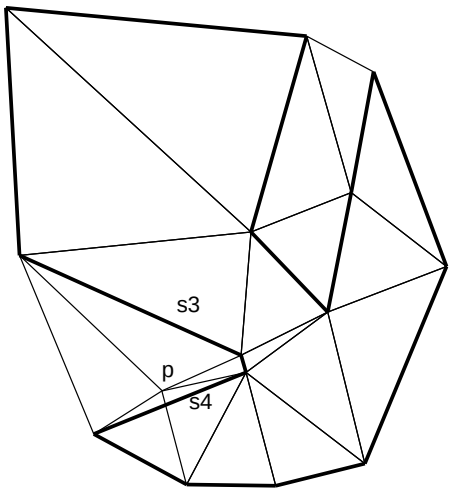
(a) Eingabe Polygon
(Hier ohne Rahmenquadrat).



(b) Delaunay Triangulierung der Ecken.
Die Kante s ist keine Delaunay Kante,
denn sie wird von der Kante pq geschnitten.

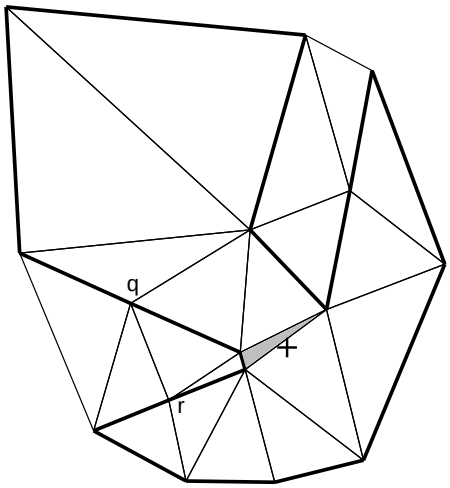


(c) Kante s wird halbiert zu s1 und s2.
Der kleinste Winkel ist 5.9 Grad.
Das Kreuz ist der Umkreismittelpunkt.

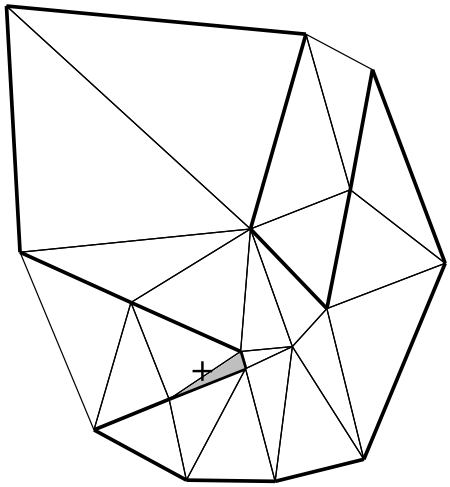


(d) Der Umkreismittelpunkt p wuerde
die Kanten s3 und s4 durchdringen.

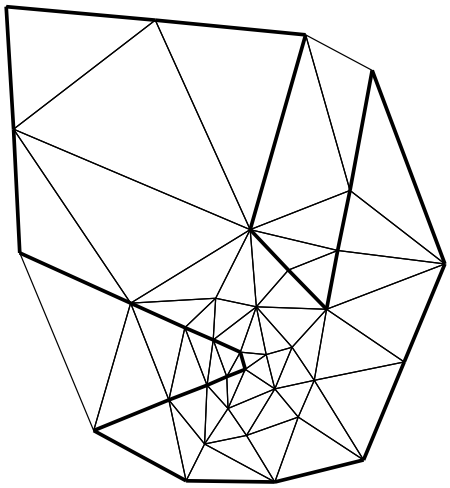
Abbildung 4.12: Rupperts Beispiel für seinen Algorithmus



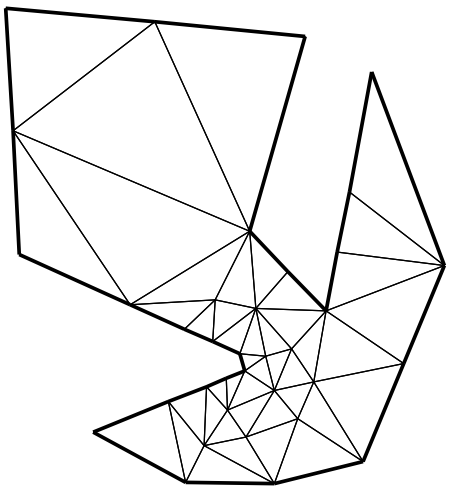
(e) 2 Kanten werden halbiert bei q und r.
Kleinster Winkel jetzt 9.8 Grad,
Das Dreieck wird aufgeteilt.



(f) Neuer kleinster Winkel 11.6 Grad.



(g) Resultat nach weiteren Schritten,
bis ein kleinster Winkel von
25 Grad erreicht ist.



(h) Dreiecke ausserhalb des Polygons
werden entfernt (optional).

Abbildung 4.13: Rupperts Beispiel, fortgesetzt

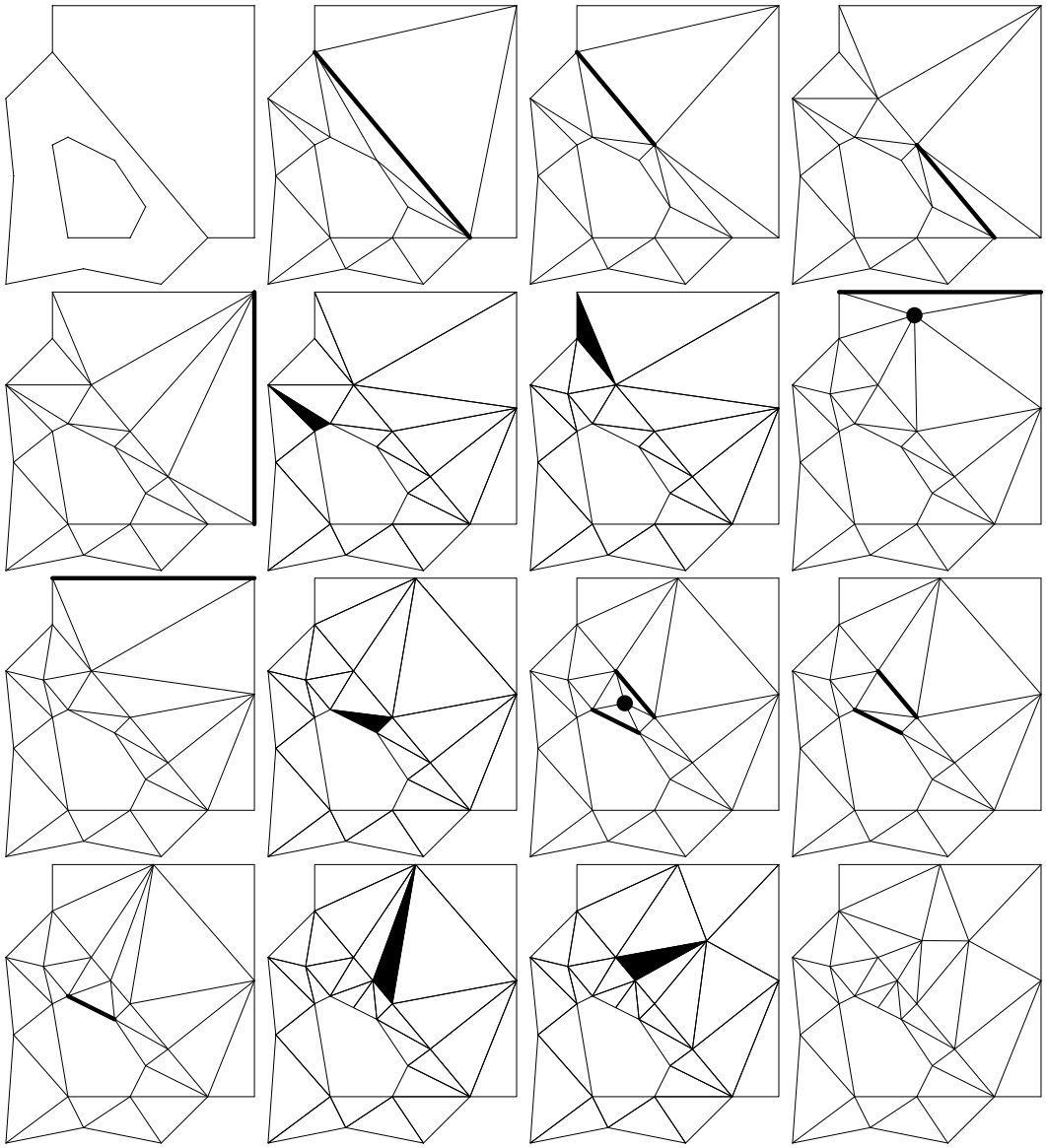


Abbildung 4.14: Shewchuks Beispiel zu Rupperts Verfeinerungalgorithmus

Konstanten viel zu groß sind und nach strengerer Beweistechnik rufen. Er stellt diesen Überlegungen Beispiele und Statistiken gegenüber.

Ruppert geht außerdem auf die Verallgemeinerung seiner Methode auf den 3D-Fall ein.

An Extrembeispielen im $\mathbb{R}^2$ stellt Ruppert die Methode der 'concentric shells' der verwendeten Methode des Einfügens von Kanten- bzw. Umkreis-Mittelpunkten gegenüber. Diese Methode wird auch in `triangle` verwendet. Ein Beispiel dafür sehen wir in Abbildung 4.16. Der Eingabe-PSLG besteht aus einem Bündel von Linien (in Fettdruck), die von einem Ur-

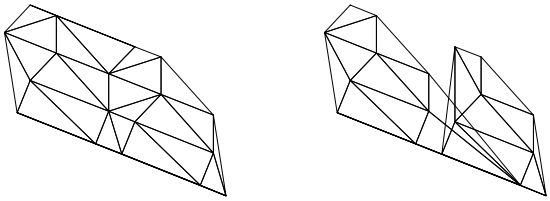


Abbildung 4.15: Exakte und normale Gleitpunktarithmetik im Vergleich

sprung in viele verschiedene Richtungen weisen ('spokes', Speichen), in einem Quadrat. Neben diesem Eingabe-PSLG ist seine DT, darunter die verfeinerte DT nach Ruppert und mit `triangle` bei einer Vorschrift, dass der minimale Winkel mindestens 20 Grad beträgt. Bei einer Winkel-Vorschrift von 25 Grad gibt `triangle` eine Fehlermeldung aus.

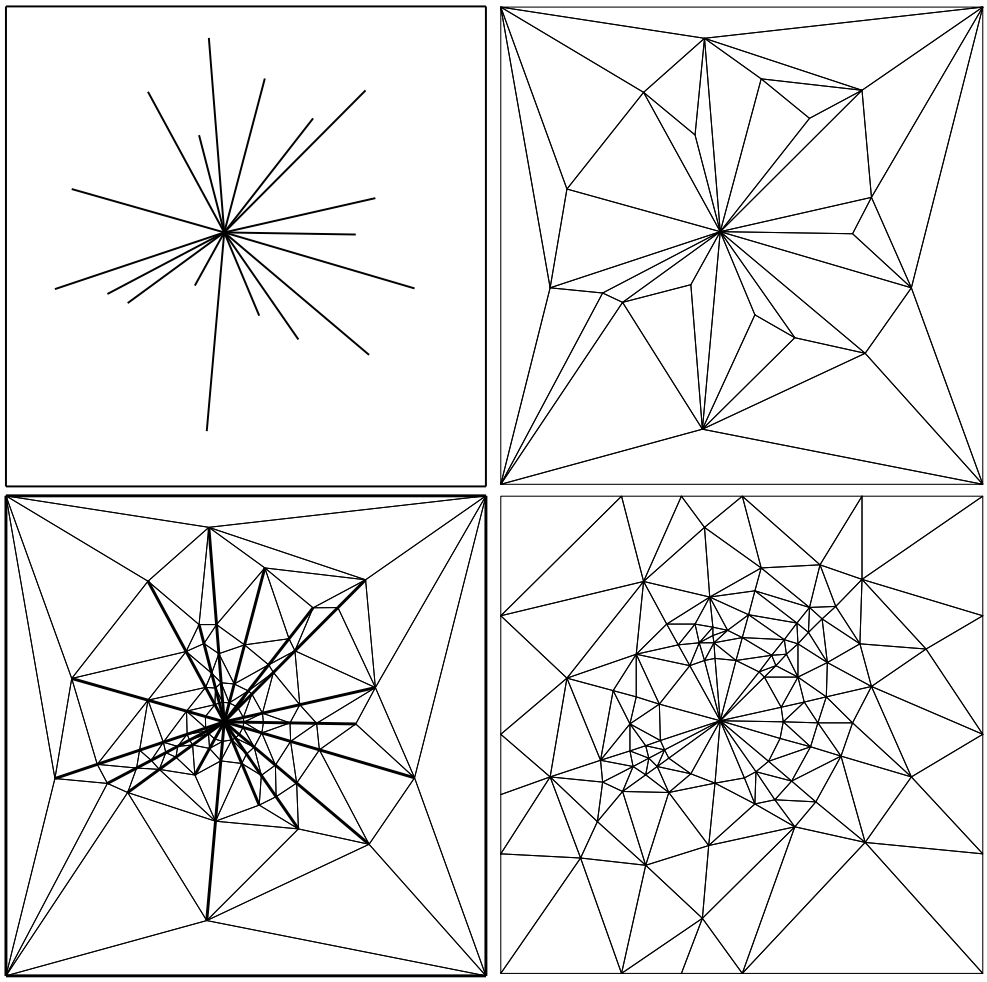


Abbildung 4.16: Die Methode der 'concentric shells' an einem extremen Beispiel

Bemerkung 4.2.6

Drei Bemerkungen zu Shewchuks Realisierung von Rupperts Algorithmus.

(1) Es ist nicht ganz einzusehen, dass erst eine DT berechnet wird, und erst bei Nicht-Respektierung einer Kante eine CDT. Vielleicht kommt man aber damit einer DT näher, die eine CDT ja nicht immer liefert.

(2) Im Gegensatz zu Ruppert verfeinert Shewchuks Programm `triangle` erst nach dem Entfernen der Dreiecke aus Löchern und Außengebieten. Das verhindert unnötige Verfeinerung in Gebieten, die anschließend entfernt werden. Den Unterschied sieht man an den Beispielen von Ruppert (Abbildungen 4.12 und 4.13) und Shewchuk (Abbildung 4.14).

(3) Es ist nicht leicht zu sehen, warum die exakte Arithmetik so wichtig ist. Der Grund liegt in der Berechnung der Umkreis-Mittelpunkte, die durch Rundungsfehler gerade eben außerhalb des Gitters liegen können mit Folgen wie in Abbildung 4.15.

4.2.2 Keine stumpfen Winkel*

Am wichtigsten für die FEM ist sicher das Vermeiden kleiner Winkel. Aber auch das Vermeiden stumpfer Winkel ist aus mehreren Gründen wichtig, denn dann gilt:

(1) Der Umkreismittelpunkt ist im Dreieck enthalten.

(2) Das ebene Dual einer solchen Triangulierung ist einfach mit Mittelsenkrechten konstruierbar.

(3) Bei der finite Volumen-Methode benutzt man gern duale Netze, weil dann Strömungsberechnungen einfacher werden.

(4) Es ergeben sich gewisse Optimalitäts-Eigenschaften, wie wir gleich sehen werden.

Wir betrachten das Problem 'keine stumpfen Winkel', weil für das Problem 'keine großen Winkel' bei Grenzwinkeln deutlich unter 90 Grad schnell eine große Menge von Steiner-Punkten und entsprechende Komplexität entstehen kann.

Satz 4.2.7 Jede Triangulierung, in der kein stumpfes Dreieck enthalten ist, ist die Delaunay-Triangulierung seiner Ecken.

Der Beweis benutzt die lokale Eigenschaft der DT, indem er geteilte Vierecke betrachtet. Daß die Umkehrung nicht gilt, kann man ebenso leicht an einem Viereck mit zwei benachbarten stumpfen Winkeln sehen.

Satz 4.2.8 Für jede Menge mit n Punkten gibt es eine nicht-stumpfe Steiner-Triangulierung der Komplexität $O(n)$.

*Die beiden Überschriften der Unterabschnitte dieses Abschnitts sind irreführend, denn auch im nächsten Abschnitt "Quadtrees" werden wir garantierte kleinste Winkel bekommen und Algorithmen zum Vermeiden von stumpfen Winkeln kennenlernen.

Der Beweis benutzt den Quadtree-Algorithmus, den wir im nächsten Abschnitt behandeln werden.

Satz 4.2.9 Ein (mehrfach zusammenhängendes) Polygon mit n Ecken kann nicht-stumpf trianguliert werden in $O(n^2)$ Zeit.

Den entsprechenden Algorithmus haben Bern, Dobkin und Eppstein entwickelt. Er ist im Übersichtsartikel von Bern und Eppstein beschrieben, Details findet man in [7]. Wir wollen auf seine Beschreibung verzichten.

Die Komplexität $O(n^2)$ kann nicht aufrecht erhalten werden, wenn man das Innere und Äußere eine Polygons triangulieren will. Dieses offene Problem tritt auch auf, wenn man ein in Teilgebiete aufgeteiltes Polygon so triangulieren will, dass die Punkte auf den Schnittkanten von beiden Seiten übereinstimmen sollen. Das ist wichtig für die FEM bei Gebietszerlegungen sowohl aus algorithmischen Gründen (Parallelisierung) als auch aus physikalischen Gründen (Sprungstellen in Parameterfunktionen).

4.3 Quadtree

Quadtree-Algorithmen bestehen aus der Einteilung einer graphischen Eingabemenge in Baum-artig hierarchisch geordnete Standardobjekte, meist Quadrate. Es gibt Quadtree-Algorithmen etwa seit 1983. Sie werden aber im Detail sehr unterschiedlich definiert.

Auf der einen Seite stehen einfache Unterteilungen einer Punktmenge in jeweils um den Faktor 2 kleiner werdende Quadrate, die gewisse Regularitätsbedingungen erfüllen, aber weder in Dreiecke eingeteilt noch verbogen werden, um sich einem Gebiet anzupassen. Dies wird dann ausgenutzt, um die Lösung der PDE möglichst einfach mit Differenzenverfahren erhalten zu können. Dadurch wird auch der Einsatz von Mehrgittermethoden ermöglicht bzw. vereinfacht.

Auf der anderen Seite stehen Definitionen von Datenstrukturen, die die einfache obige Konstruktion ebenso beinhalten wie die entsprechend Baum-artig organisierte Hierarchie von Dreiecken oder Sechsecken oder anderen geometrischen Formen.

Wir wollen uns auf eine Algorithmen-Gruppe konzentrieren, die Quadrate als Grundelemente benutzt, damit nach gewissen Regeln eine Punktmenge, eine Kantenmenge oder ein Polygon überdeckt und sie schließlich so verbiegt, daß sie mit den gegebenen Punkten, Kanten oder dem Polygon zusammenfallen. Die Algorithmen stammen von Bern, Eppstein und Gilbert, [9]. Sie beweisen für verschiedene Versionen das Einhalten gewisser Minimalwinkel und Aspektverhältnisse. Eine Version erzeugt eine nicht-stumpfe Triangulierung.

4.3.1 Beweisbar gute Triangulierungen

Wir fassen zunächst die Ergebnisse von Bern, Eppstein und Gilbert kurz zusammen, um diese dann teilweise im Einzelnen vorzustellen:

(1) Triangulierung einer Punktmenge im $\mathbb{R}^2$ mit garantiertem Minimalwinkel und maximalem Aspektverhältnis[†]. Der Algorithmus ist size-optimal.

(2) Nicht-stumpfe Triangulierung einer Punktmenge im $\mathbb{R}^2$. Dieser Algorithmus ist nur eine konstruktive Verfeinerung des ersten.

(3) Triangulierung einer Menge von sich nicht schneidenden Kanten im $\mathbb{R}^2$ mit garantiertem Minimalwinkel.

(4) Triangulierung eines Polygons im $\mathbb{R}^2$ mit garantiertem Minimalwinkel. Hier wird eine untere Schranke für die Eingabewinkel vorausgesetzt. Es wird allerdings auch ein spezieller Teilalgorithmus für spitze innere Winkel angegeben.

Außerdem geben die Autoren Algorithmen für 'Triangulierungen' im $\mathbb{R}^d$ an. Alle Algorithmen habe eine Laufzeit $O(n \log n+k)$, wenn n die Eingabegröße ist und k die Ausgabegröße. Für letztere können jeweils auch die Größenordnungen angegeben werden.

[†]Jetzt: (längste Seite)/(kleinste Höhe)!

4.3.2 Triangulierung von Punktmengen

Im folgenden sei $A(a, b, c)$ das Aspektverhältnis eines Dreiecks mit den Eckpunkten a, b, c. $R(a, b, c)$ sei das Verhältnis der längsten zur kürzesten Seite des Dreiecks. Es gilt offenbar $A(a, b, c) > R(a, b, c)$, wobei $R(a, b, c)$ sehr viel kleiner als $A(a, b, c)$ sein kann.

Weiter sei $|\mathcal{T}|$ die Anzahl Ecken einer Triangulierung $\mathcal{T}$ und $A(\mathcal{T})$ das größte Aspektverhältnis in dieser Triangulierung, $R(\mathcal{T})$ entsprechend. Mit $\mathcal{QT}(X)$ bezeichnen wir eine Quadtree-Triangulierung von X. Sie ist immer auch algorithmisch definiert. $\mathcal{DT}(X)$ bezeichnet die Delaunay-Triangulierung von X.

Satz 4.3.1 Zu einer gegebenen Punktmenge X im $\mathbf{R}^2$ kann man eine Triangulierung $\mathcal{QT}(X)$ angeben derart, daß jeder Punkt in X eine Ecke in $\mathcal{QT}(X)$ ist und daß

$$A(\mathcal{QT}(X)) \leq 4. \tag{4.7}$$

Für jede Triangulierung $\mathcal{T}$, die auch alle Punkte aus X enthält, gilt:

$$|\mathcal{QT}(X)| = O(|\mathcal{T}| \log A(\mathcal{T})). \tag{4.8}$$

Beweisidee: Der Beweis folgt i.w. aus der Konstruktion des Algorithmus und Überlegungen zur $\mathcal{DT}(X)$. ■

Das Aspektverhältnis kann sogar auf 2 reduziert werden, wie man später an dem nichtstumpfen Algorithmus sehen kann. Dazu ist allerdings eine größere Zahl von Dreiecken notwendig, aber nur um einen konstanten Faktor mehr Dreiecke.

Dasselbe gilt umgekehrt: Ein größeres Aspektverhältnis führt nur zu einer linearen Reduzierung der Anzahl Dreiecke. Außerdem kann man eine scharfe Abschätzung gegenüber der DT angeben:

$$|\mathcal{QT}(X)| = O(n \log A(\mathcal{DT}(X))) \tag{4.9}$$

Definition 4.3.2 (1) Der *Quadtree* besteht aus Quadraten mit Kanten in den Koordinatenrichtungen und verbogenen Quadraten. Diese geometrischen Grundelemente nennen wir auch *Zellen* oder *Boxen*. Die Zellen des Quadtree können als *Baum* angeordnet werden. Jede Zelle ist entweder ein *Blatt* des Baumes, oder sie ist in vier Zellen *geteilt*. Die *Wurzel* des Baumes enthält alle Punkte aus X.

(2) Eine Zelle q kann höchstens vier *Nachbarn* haben, das sind Zellen gleicher Größe, die eine Kante mit q gemeinsam haben.

(3) Eine Zelle q kann acht *erweiterte Nachbarn* haben, das sind Zellen gleicher Größe, die eine Ecke mit q gemeinsam haben.

(4) Eine Zelle q heißt *überfüllt*, falls eine der folgenden Bedingungen zutrifft:

- C1. q hat die Seitenlänge l, enthält einen Punkt $x \in X$, und es gibt einen anderen Punkt in X, dessen Abstand zu x kleiner als $2\sqrt{2}\,l$ ist. (Das ist die doppelte Länge der Diagonale von q.)
- C2. q enthält einen Punkt $x \in X$, und einer der erweiterten Nachbarn von q ist geteilt.

(5) Der Quadtree heißt *balanciert*, wenn jede Kante einer ungeteilten Zelle höchstens eine zusätzliche Ecke enthält. (Diese Definition wird später verschärft!)

Der Algorithmus *A*4

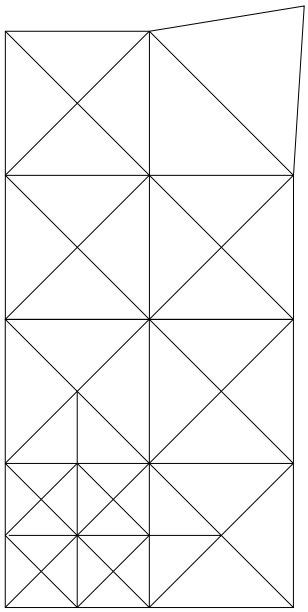
(1) Sei $\tilde{q}$ ein kleinstes Quadrat, das alle Punkte von X enthält. Dann sei das Wurzel-Quadrat q_0 doppelt so groß wie $\tilde{q}$ und mit $\tilde{q}$ konzentrisch. Setze $Q := \{q_0\}$.

(2) Solange es überfüllte Zellen $q \in Q$ gibt:

- Teile q.
- Teile alle erweiterten Nachbarn von q so, daß jedes q mit einem Punkt $x \in q$ acht erweiterte Nachbarn hat.
- Teile alle Zellen, die nicht balanciert sind.
- Füge alle neuen Zellen in den Baum Q ein.

(3) Verbiege die Zellen, die einen Punkt $x \in X$ enthalten:
Wenn y die Ecke ist, die x am nächsten liegt, wird in Q y durch x ersetzt.

(4) Trianguliere den entstanden Quadtree:
Unverbogene Zellen ohne geteilte Nachbarn werden durch Einfügen des Mittelpunktes in vier rechtwinklige, gleichschenklige Dreiecke geteilt.
Bei unverbogenen Zellen mit geteilten Nachbarn wird pro Kantenmittelpunkt ein Dreiecke geteilt.
Verbogene Zellen werden längs der Diagonale geteilt, die das kleinere Aspektverhältnis ergibt. Geteilte Nachbarn gibt es in diesem Fall ja nicht.
(Die Autoren ändern diesen Algorithmus für ihre Beispiele leicht ab, definieren aber nicht genau, wie!) In der neben stehenden Zeichnung sind diese Situationen verdeutlicht.



Geht man jetzt alle vorkommenden Fälle durch und beachtet, daß eine verbogene Zelle immer von acht Nachbarn gleicher Original-Seitenlänge umrahmt ist, dann rechnet man leicht nach, daß Gleichung (4.7) erfüllt ist:

Im rechtwinkligen, gleichschenkligen Dreieck ist das Aspektverhältnis genau 2, in den verbogenen Zellen kann sich die längste Seite höchstens um den Faktor $\sqrt{2.5}$ verlängern, und die zugehörige Höhe etwas weniger als halbieren. Das ergibt einen Faktor kleiner als 3.2. ■

4.3.3 Quadtrees ohne stumpfe Winkel

Die Vermeidung von stumpfen Winkeln erreichen die Autoren durch Hinzufügen von Steiner-Punkten in zwei Schritten. Zunächst setzen sie voraus, daß sich kein Punkt zu nah am Rand einer Zelle befindet. Dann kann durch Einfügen von drei Steiner-Punkten pro Eingabepunkt eine nicht-stumpfe Triangulierung mit maximalem Aspektverhältnis 2 erzeugt werden.

Im zweiten Schritt wird das Gitter in der Nähe von Punkten, die zu nah am Rand ihrer Zelle liegen, so verschoben, daß die randferne Situation hergestellt ist und der erste Algorithmusschritt angewendet werden kann. Dabei wird die Verschiebung auch so definiert, daß kein stumpfer Winkel entsteht und das Aspektverhältnis kleiner gleich 2 bleibt.

Definition 4.3.3 (1) Zu einem Punkt $x \in X$ heißen die vier Zellen, die die am nächsten bei x liegende Ecke enthalten, *Umgebungszellen* bzw. *x umgebende Zellen.*

(2) Ein Punkt $x \in X$ liegt *zentral* in seiner Zelle q, wenn er auch in der zu q konzentrischen Zelle mit halber Seitenlänge liegt.

Der Algorithmus soll als Variante von $A4$ verstanden werden, die Bedingungen des Algorithmus $A4$ also einhalten. Deshalb ist jede Umgebungszelle genau so groß wie q. Also liegt x auch zentral in der Vereinigung der Umgebungszellen, wenn x nur zentral in q liegt.

Der Algorithmus $A2$-zentral

Es wird vorausgesetzt, daß jeder Punkt $x \in X$ zentral in seiner Zelle q liegt.

Für $x \in X$ und seine Umgebungszellen wird folgendes durchgeführt:

(1) Es werden drei Steiner-Punkte wie folgt definiert:

- Aus den Umgebungszellen, die senkrecht zu q liegen, werden die Mittelpunkte zusätzlich genommen.
- Aus der Umgebungszelle, die diagonal zu q liegt, wird der Mittelpunkt der Diagonale vom Zellenmittelpunkt zum Mittelpunkt der Vereinigung der Umgebungszellen genommen,

(2) x und die drei Steiner-Punkte werden miteinander verbunden mit Ausnahme der beiden Mittelpunkte der senkrecht liegenden Umgebungszellen. Außerdem wird jeder Punkt mit den drei nächsten Eckpunkten verbunden.

(3) Die inneren Kanten der Umgebungszellen werden entfernt.

Die Situation ist in Abbildung 4.17 verdeutlicht: Die gepunkteten Quadrate definieren die zentrale Lage in der oberen rechten Zelle bzw. die Lage bezüglich der Umgebungszellen. Die gestrichelten Linien sind die inneren Zellenkanten, die am Ende des Algorithmus entfernt werden, durchgezogene Linien sind gültige Zellenkanten. Es ist jetzt wieder mit Fallbetrachtungen leicht zu sehen, daß die Triangulierung keine stumpfen Winkel enthalten kann, und daß das Aspektverhältnis kleiner gleich 2 ist.

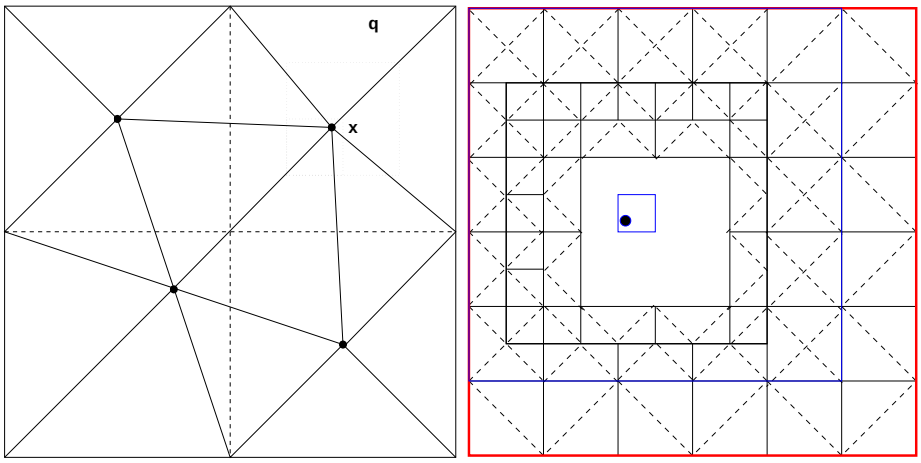


Abbildung 4.17: Triangulierung mit zentralen Punkten und Gitterverschiebung

Der Algorithmus $A2$-shift

Nach dem Algorithmus $A4$ liegt jeder Punkt $x \in X$ in der zentralen Zelle eines 3×3-Gitters von Zellen gleicher Kantenlänge l. Jede dieser Zellen wird geteilt. Durch diese Teilung können weitere Teilungen erforderlich werden. Diese werden ausgeführt. Jetzt liegt der Punkt $x \in X$ in der zentralen Zelle eines 5×5-Gitters von Zellen gleicher Kantenlänge. Die inneren neun Zellen dieses Gitters werden nochmal geteilt, sie haben jetzt die Seitenlänge $l/4$. Die äußeren 16 Zellen werden jetzt so in Dreiecke und Quadrate geteilt, daß die Außen kanten ungeteilt bleiben, aber daß x in der zentralen Zelle eines 7×7-Gitters liegt.

Jetzt gibt es vier Möglichkeiten, die Quadrate dieses Gitters wieder in ein Gitter mit der Seitenlänge $l/2$ umzuformen. Bei einer dieser Möglichkeiten liegt x zentral in seiner Zelle. Zu dieser Möglichkeit bestimmen wir die Umgebungszellen und entfernen ihre inneren Kanten. Das ändert nichts an den Außenkanten des ursprünglichen 3×3-Gitters. Die Umgebungszellen werden jetzt wie im Algorithmus $A2$-zentral trianguliert, die restlichen Zellen mit gleichschenkligen, rechtwinkligen Dreiecken.

Abbildung 4.17 zeigt rechts diese Konstruktion. Dabei sind die gleichschenkligen, rechtwinkligen Dreiecken gestrichelt gezeichnet, während in der Vereinigung der Umgebungszellen nur die zentrale $l/4$-Zelle zu sehen ist, dort ist verkleinert eine der Abbildung 4.17-links entsprechende Triangulierung einzusetzen. Die Gesamtzeichnung ist gleich dem ursprünglichen 3×3-Gitter. Das 7×7-Gitter ist verstärkt eingezeichnet.

Aus der Konstruktion wird der folgende Satz unmittelbar klar.

Satz 4.3.4 Für jede Punktmenge X gibt es eine Triangulierung ohne stumpfen Winkel, die die Punkte von X als Ecken enthält, und die ein maximales Aspektverhältnis von 2 hat.

Sie ist nur um einen konstanten Faktor größer als die Quadtree-Triangulierung nach Algorithmus $A4$.

Wenn man die Bedingung über das Aspektverhältnis fallen läßt, kann man auch einen ähnlichen Algorithmus konstruieren, der nur $O(|X|)$ Punkte bzw. Dreiecke benötigt. Eine

superlineare Menge von Dreiecken tritt beim vorgestellten Algorithmus nur auf, wenn man sogenannte Cluster von Punkten hat, also eine hohe Punktdichte in einem relativ kleinen Teilgebiet. Für diese Situation geben die Autoren einen speziellen Algorithmus an, der diese Teilgebiete gesondert behandelt, dort aber auf ein begrenztes Aspektverhältnis verzichtet, und damit auf $O(|X|)$ Dreiecke kommt. Details hierzu wollen wir nicht angeben.

4.3.4 Triangulierung von Polygonen

Sei P ein Polygon mit Rand ∂P.

Das Polygon kann mehrfach zusammenhängend sein, und die 'Löcher' können auch degeneriert sein, d.h. leeres Inneres haben. Dies ist eine sinnvolle Erweiterung für den Fall, daß die Koeffizienten einer zu lösenden PDE Funktionen mit Sprungstellen sind, also z.B. Wärmeleitungskoeffizienten bei Gebieten, die aus mehreren Materialen bestehen. In solchen Fällen muß eine innere Grenzlinie eingehalten werden. Punkte auf solchen Grenzlinien werden automatisch verdoppelt, wenn man diese Grenzlinie als degeneriertes Loch behandelt. Aus Gründen der einfacheren Parallelisierbarkeit kann die Einführung fester Innenkanten bzw. degenerierter Löcher auch sinnvoll erscheinen.

Definition 4.3.5 (1) Seien $x, y \in \partial P$. Dann liegt y *entfernt* von x, falls gilt:

x, y liegen auf verschiedenen Zusammenhangskomponenten von ∂P oder jeder Weg von x nach y auf ∂P enthält mindestens zwei Knoten. Dabei werden Anfangs- und Endknoten mitgezählt, d.h. die zwei Endpunkte einer Kante liegen entfernt voneinander.

(2) Seien $x, y \in \partial P$. Dann heißt y *nächster entfernter Nachbar* von x, falls gilt:

y ist der nächste Punkt, der entfernt von x liegt. (Geodätische Distanz = kürzester Weg in P, nicht nur auf ∂P.)

(3) Eine Quadtreezelle q mit der Seitenlänge l heißt *überfüllt* falls eine der folgenden Bedingungen erfüllt ist:

(a) q enthält einen Punkt $x \in \partial P$ und der nächste entfernte Nachbar liegt näher als $2\sqrt{2}\,l$ am Punkt x.

(b) q enthält einen Punkt $x \in X$ und einer der erweiterten Nachbarn von q wurde geteilt.

(4) Der Schnittpunkt des Polygons mit dem Rand einer Zelle heißt *q-Knoten*.

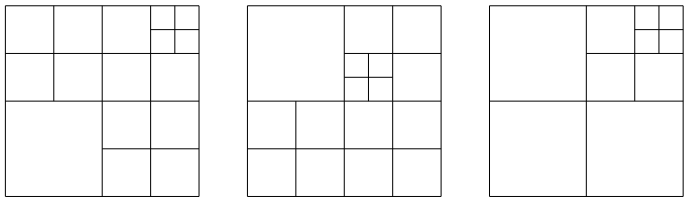
(5) Der Quadtree heißt *balanciert*, falls gilt:

(a) Jede zu einer Zelle q benachbarte Zelle hat entweder die halbe, die gleiche oder die doppelte Seitenlänge wie q.

(b) Die Seitenlängen von zwei zu einer Zelle q benachbarten Zellen sind gleich oder sie unterscheiden sich nur um den Faktor 2.

Voraussetzung 4.3.6 Abschätzungsbedingung

Jede Zelle, die einen Knoten des Polygons enthält, muß von 24 Zellen gleicher Größe umgeben sein.



balanciert nicht balanciert nicht balanciert

Der Algorithmus *AP*

(1) Berechne eine das Polygon umgebende Zelle.

(2) Solange eine überfüllte Zelle existiert, teile diese in vier gleichgroße Zellen auf.

(3) Balanciere den Quadtree.

(4) Verschiebe Quadtreeknoten auf die entsprechenden Polygonknoten.

(5) Verschiebe Quadtreeknoten auf die entsprechenden $q-$Knoten.

(6) Trianguliere die entstandenen Vierecke.

Schwierigkeiten können bei diesem Algorithmus auftreten, wenn eine Zelle Punkte mehrerer Zusammenhangskomponenten enthält. Dann wird diese Zelle sozusagen für jede Zusammenhangskomponente einmal verbogen. Genauer legen wir folgende Verbiege-Regeln fest. Eine kleines Beispiel für die etwas komplizierten Regeln sehen wir am Rand daneben.

Es sei B_y die Vereinigung der drei oder vier Zellen, deren Kanten y als Knoten (Ecke oder q-Knoten) enthalten.

(1) Jeder Punkt $x \in X$ und jeder q-Knoten wählen ihre nächste Quadtree-Ecke. Jede so gewählte Ecke y kann in bis zu vier Kopien geteilt werden. Wir biegen eine Kopie von y zu jedem $x \in X$, das y gewählt hat, und zum nächsten q-Knoten, der y in einer Zusammenhangskomponente von $P \cup B_y$ gewählt hat, die kein $x \in X$ enthält.

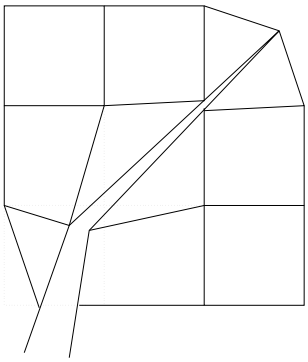
Wenn y nur von q-Knoten gewählt wird, wird es noch nicht verbogen.

(2) Jeder verbleibende q-Knoten wählt jetzt die nächste Quadtree-Ecke, die noch nicht verbogen wurde. Wir biegen eine Kopie von y zu jeder Kante von ∂P, die einen q-Knoten enthält, der y gewählt hat. Dabei wird – mit einer Ausnahme – vertikal verbogen, wenn die Kante eine Steigung zwischen -1 und 1 hat, sonst horizontal. Die Ausnahme ist, daß eine Ecke, die schon verbogen ist und nur einmal gewählt wird, ihrem Wähler direkt zugebogen wird.

(3) Für geteilte Kanten gibt es zwei Regeln:

(a) Wenn die beiden Endpunkte einer geteilten Seite beide zu einer Kante s von ∂P gebogen werden, dann wird auch der Mittelpunkt von s zu dieser Kante gebogen.

(b) Wenn eine geteilte Kante von einer Kante s von ∂P geschnitten wird, dann werden beide Endpunkte der geschnittenen Kante zu s gebogen.

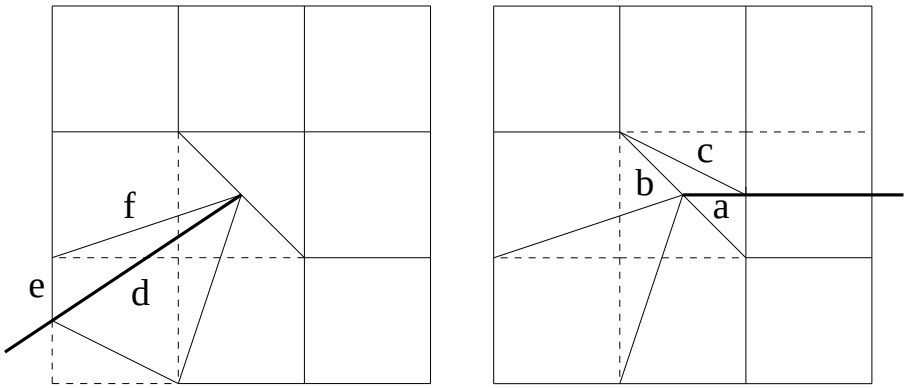


Nach Beendigung aller notwendigen Verbiegungen werden Punkte und Kanten des Quadtree, die zum Komplement von P gehören, entfernt.

Schließlich werden alle Vierecke durch Diagonalen trianguliert. Dabei werden Randstücke von P gewählt, wo das notwendig ist. Ansonsten wird die Diagonale gewählt, die das kleinere Aspektverhältnis liefert.

Satz 4.3.7 Die entstandene Triangulierung $\mathcal{QT}(P)$ trianguliert P, hat ein Aspektverhältnis kleiner gleich 5 und ihr kleinster Winkel ist größer oder gleich 15.26 Grad.

Beweis: Die Autoren geben 18.4 Grad als minimalen Winkel an, der folgenden Zeichnung kann man aber entnehmen, daß das maximale Aspektverhältnis und der minimale Winkel in zwei verschiedenen Extremsituationen auftreten:



Sei l die Seitenlänge der betrachteten unverbogenen Zelle. Mit $a = \frac{1}{2}l$, $b = \frac{\sqrt{2}}{2}l$ und $c = \frac{\sqrt{5}}{2}l$ ergibt sich der Winkel α zu $18,4^o$ und das maximale Aspektverhältnis zu 5.

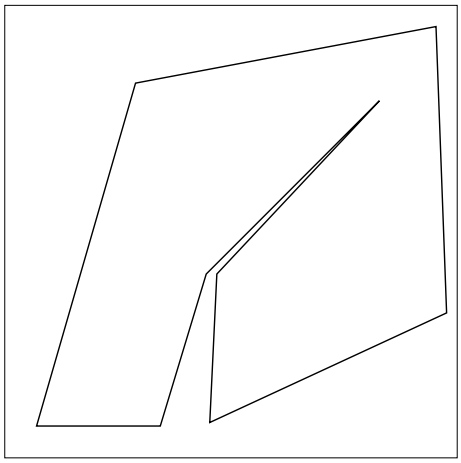
Mit $d = \frac{\sqrt{13}}{2}l$, $e = \frac{1}{2}l$ und $f = \frac{\sqrt{10}}{2}l$ ergibt sich der minimale Winkel α zu 15.26^o und das Aspektverhältnis zu 4.33.

Es sind jetzt drei weitere Falluntersuchungen notwendig, auf die wir nicht eingehen wollen. ■

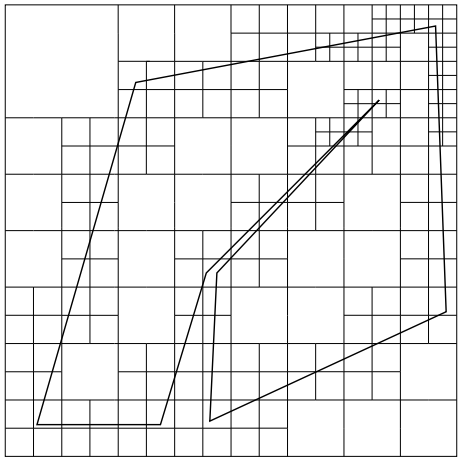
Der Algorithmus setzt voraus, daß es keine sehr spitzen Innenwinkel gibt. (> 11.5 Grad für das Aspektverhältnis von 5, > 15.26 Grad für einen minimalen Winkel von 15.26 Grad.)

Für kleinere Innenwinkel wird ein spezieller Algorithmus angegeben. Bevor wir diesen Algorithmus angeben, wollen wir uns zunächst ein Beispiel ansehen:

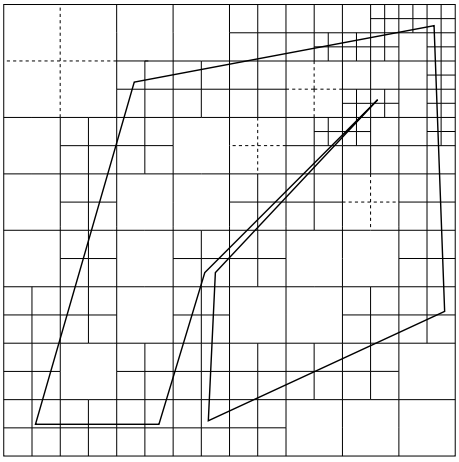
Beispiel 4.3.8



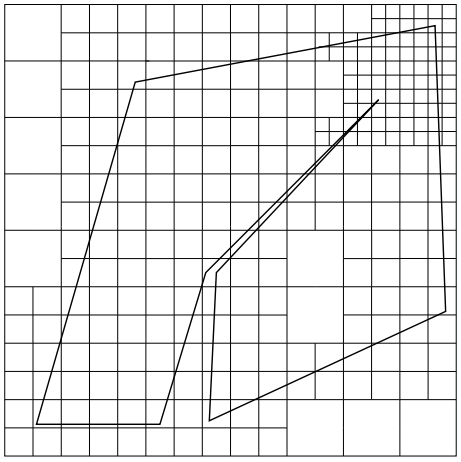
Berechne eine das Polygon umgebende Zelle.



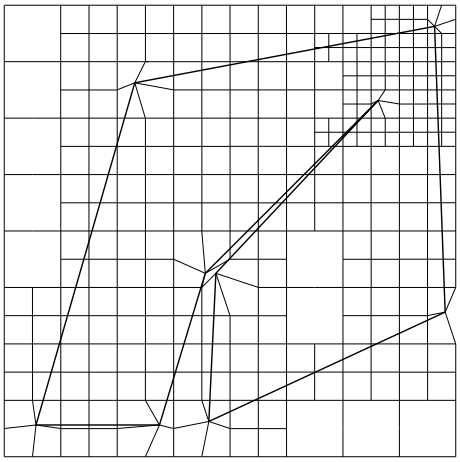
Solange eine überfüllte Zelle existiert, teile diese in vier gleichgroße Zellen auf.



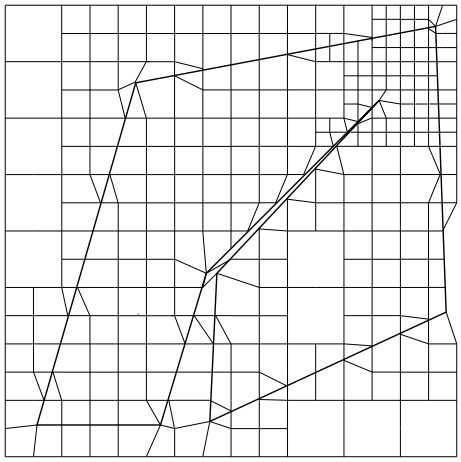
Balanciere den Quadtree.



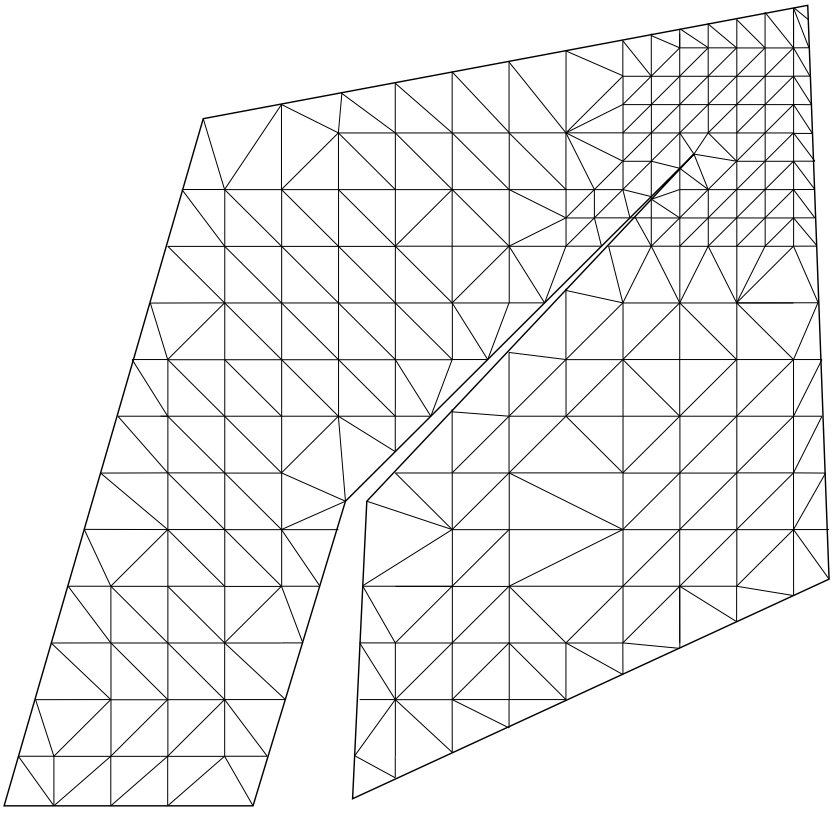
Erfülle die Abschätzungsbedingung.



Verschiebe Quadtreeknoten auf die entsprechenden Polygonknoten.



Verschiebe Quadtreeknoten auf die entsprechenden q-Knoten.



Trianguliere die entstandenen Vierecke.

Algorithmus zur speziellen Behandlung spitzer Winkel

Die Voraussetzung 'keine spitzen Eingabewinkel' kann man fallenlassen, wenn diese gesondert behandelt werden. Und da die Behandlung von Sonderfällen in vielen der automatischen geometrischen Algorithmen eine Rolle spielt, wollen wir diesen speziellen Teilalgorithmus abschließend angeben.

(1) Sei v die Spitze eines spitzen Winkels von P. Wir behandeln zunächst P mit dem Quadtree-Algorithmus wie oben. Danach wird v in einem Blatt des Quadtree-Baumes liegen, das acht erweiterte Nachbarn derselben Größe hat. Wir schneiden das größte gleichschenklige Dreieck I aus, dessen Schenkel Teile von ∂P sind, und das vollständig in der Vereinigung dieser neun Zellen liegt. Das ergibt eine neue *Schnittkante*, die wir P hinzufügen.

(2) Das gleiche tun wir mit jedem spitzen Winkel von P. Dann bekommen wir ein Restpolygon $\tilde{P}$, das keine spitzen Winkel mehr hat und wie oben behandelt wird (mit kleinen Ausnahmen, auf die wir nicht eingehen wollen, siehe aber unten). Wir müssen jetzt die abgeschnittenen Dreiecke so unterteilen, daß eine konforme Gesamttriangulierung entsteht.

(3) Die dem spitzen Winkel gegenüberliegende Seite s sei durch eine Anzahl von Kanten in Ecken $v_1, \ldots, v_m$ aufgeteilt. Wir haben $\tilde{P}$ so trianguliert (das war die Ausnahme oben), daß die Kanten $\overline{v_i\, v_{i+1}}$ alle dieselbe Länge l haben bis eventuell auf $\overline{v_1\, v_2}$, $\overline{v_2\, v_3}$, $\overline{v_{m-2}\, v_{m-1}}$ und $\overline{v_{m-1}\, v_m}$, und daß die Länge dieser Kanten im Bereich $[l/2,\ 3\,l/2]$ liegt.

(4) **Wenn** s aus einer einzelnen Kante besteht, sind wir fertig.

Sonst fassen wir die Knoten auf s in überlappende Dreiergruppen zusammen:

$$G_1 := \{v_1, v_2, v_3\},\ \ G_2 := \{v_3, v_4, v_5\},\ \ \ldots.$$

Die letzte Gruppe kann dabei nur einen oder zwei Knoten enthalten. Wir ziehen in I eine Parallele e im Abstand l zu s, siehe Abbildung 4.18.

(5) Für jede Gruppe G_i, außer der ersten und letzten, fügen wir einen Knoten u_i auf der Parallelen e ein, der die Projektion des mittleren Punktes von G_i auf e ist. Diese Knoten haben untereinander den Abstand $2\,l$. Außerdem definieren wir den Anfangs- und Endpunkt von e als neue Knotenpunkte.

(6) Wenn die erste Kante $\overline{u_1\, u_2}$ auf e länger als $3\,l$ ist, wird im Abstand $2\,l$ vo u_2 ein zusätzlicher Punkt eingefügt. Entsprechend wird die letzte Kante behandelt.

(7) Das Trapez zwischen s und e wird trianguliert, indem die Punkte der Gruppe G_i mit dem zugeordneten Knotenpunkt u_i verbunden werden, wieder außer der ersten und letzten. Die Randvierecke werden so trianguliert, daß sich das bessere Aspektverhältnis ergibt.

(8) Jetzt setzen wir $s := e$, und $l := 2\,l$ und fahren bei (4) fort.

Mit diesem Algorithmus haben wir das spitze Dreieck I rekursiv trianguliert.

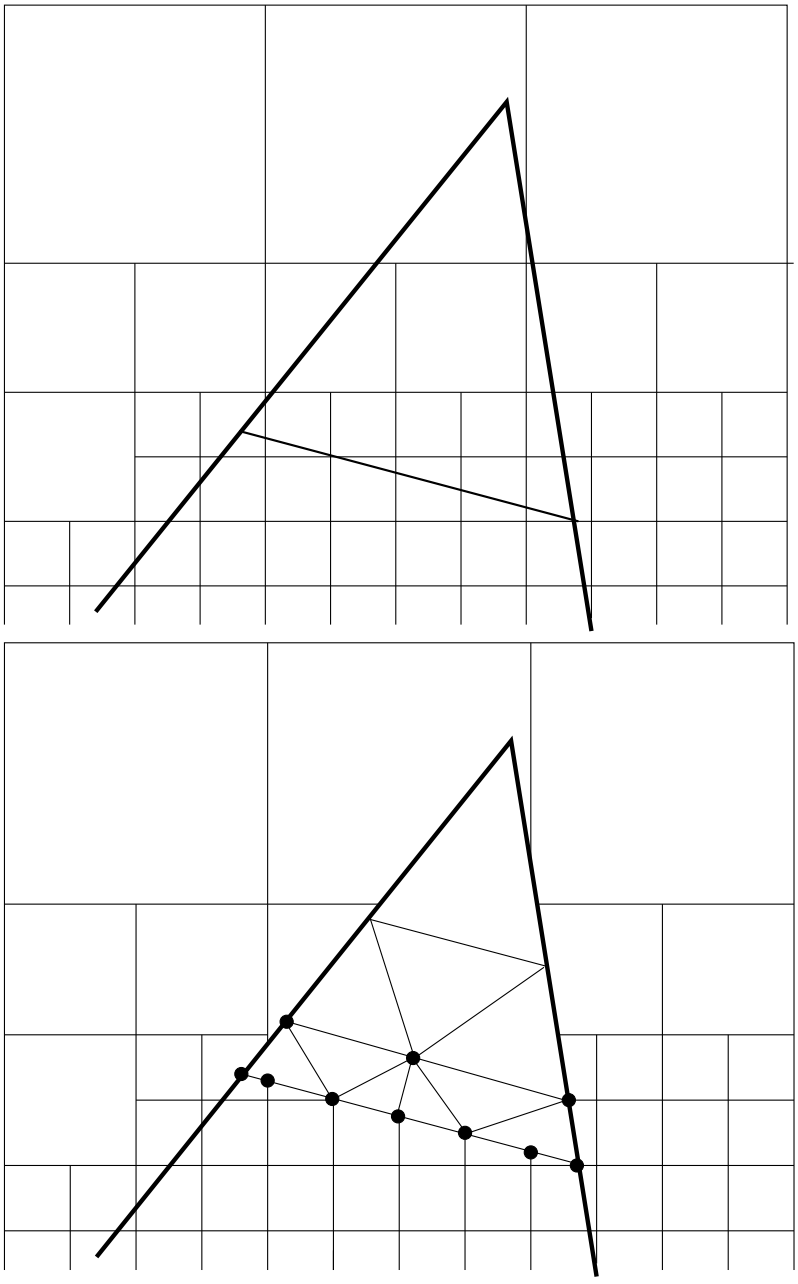


Abbildung 4.18: Triangulierung eines spitzen Winkels

4.4 Ein disc-packing Verfahren

M. Bern, S. Mitchel und J. Ruppert beschreiben in [10] ein disc-packing Verfahren zur Triangulierung von polygonal berandeten zweidimensionalen Gebieten ("Linear-size Nonobtuse Triangulation of Polygons"). Zunächst wird das gegebene Polygon mit sich nicht überlappenden Kreisen gefüllt. Mit Hilfe der Kreismittelpunkte und der Berührungspunkte wird dann eine Triangulierung, welche ausschließlich aus rechtwinkligen Dreiecken besteht, konstruiert. Dieses Verfahren hat für ein Polygon P mit n Knoten eine sequentielle Laufzeit von $O(n \log(n))$, falls P keine Löcher enthält, und eine Laufzeit von $O(n \log(n)^2)$ für ein Polygon mit Löchern, wobei $O(n)$ Dreiecke zur Triangulierung benötigt werden.

Definition 4.4.1 Verallgemeinertes Voronoi-Diagramm (GVD)

Seien A_1, A_2, $\ldots A_k$ zusammenhängende Teilmengen des $\mathbb{R}^d$ (Punkte, Kanten, Polygone) der Dimension $\leq d-1$.

Sei weiter für einen Punkt $p \in \mathbb{R}^d$ dist$(p,\, A_j)$ der kürzeste Abstand des Punktes p von der Menge A_j.

Dann ist das *Dominanz-Gebiet* von A_j über A_k definiert als

$$\mathrm{Dom}\,(A_j,\, A_k) := \left\{ p \in \mathbb{R}^d \,\middle|\, \mathrm{dist}\,(p,\, A_j) \leq \mathrm{dist}\,(p,\, A_k) \right\} \tag{4.10}$$

und damit das Voronoi-Gebiet von A_j als

$$V(A_j) := \cap_{k \neq j} \mathrm{Dom}\,(A_j,\, A_k). \tag{4.11}$$

Die Aufteilung des $\mathbb{R}^d$ in die Voronoi-Gebiete $V(A_1)$, $V(A_2)$, $\ldots V(A_k)$ heißt verallgemeinertes Voronoi-Diagramm (GVD). Das GVD wird zum VD, wenn alle A_i Punkte sind. Die Ränder der Gebiete $V(A_j)$ heißen Voronoi-Ränder. Sind die A_i geometrische Primitive wie Punkte, Linien, Polygone oder Splines, dann sind die Voronoi-Ränder Teile algebraischer Kurven oder Oberflächen.

Ein Beispiel eines GVD zu einer Menge aus Punkten, einer Kante, einem Polygon (Dreieck) und einer Bézier-Spline-Kurve (alle weiß) sehen wir in Abb. 4.19, nach [84].

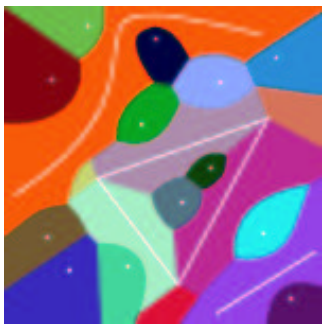


Abbildung 4.19: Verallgemeinertes Voronoi-Diagramm

Der Algorithmus kann kurz wie folgt beschrieben werden:

Algorithmus

(1) Fülle das Polygon mit Kreisen geeigneter Größe

  (a) Platziere Kreise in/an den Ecken
  (b) Verbinde die Löcher mit dem Rand
  (c) Fülle die Restflächen mit Kreisen

(2) Trianguliere die Restflächen

Im folgenden werden die einzelnen Schritte etwas genauer beschrieben.

Die Vorbereitungen

Fülle das Polygon mit Kreisen geeigneter Größe.

(1) Platzierung von Kreisen in/an den Ecken (siehe Abb. 4.20)

Bedingung 4.4.2 Kreise dürfen sich nicht schneiden.

Bedingung 4.4.3 Kreise liegen innerhalb des Polygons.

a) konvexe Ecke
Platziere einen Kreis so in die Ecke, daß er die beiden Eckseiten tangiert. Dabei entsteht eine dreiseitige Restfläche in der Ecke des Polygons.

b) konkave Ecke
Platziere zwei Kreise gleicher Größe an der konkaven Ecke, so daß jeder Kreis eine Seite und die Winkelhalbierende tangiert. Dabei entsteht eine vierseitige Restfläche an der Ecke des Polygons.

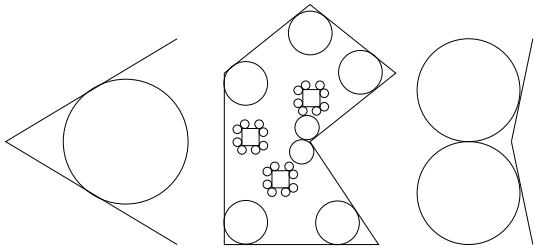


Abbildung 4.20: Einzufügende Kreise und die erzeugten Restflächen

Es stellt sich hier die Frage nach der Größe der einzufügenden Kreise. Die Kreisgröße ist beliebig, solange die beiden angegebenen Bedingungen erfüllt sind, wobei die Größe der hier eingefügten Kreise Einfluß auf die genaue Anzahl der erzeugten Dreieckselemte hat. Ein geeignetes Hilfsmittel zur Bestimmung der Kreise ist das verallgemeinerte Voronoi Diagram

(GVD). Werden alle Kanten aus dem GVD entfernt, welche zu dem Rand des "Polygons" gehören, so entsteht ein Baum, und jeder Knoten des Baumes mit höchstens zwei Blättern ist als Kreismittelpunkt des gesuchten Kreises geeignet.

Nachdem alle Ecken mit Kreisen ausgefüllt sind, müssen vorhandene Löcher mit dem Polygon verbunden werden, um eine einzige Zusammenhangskomponente zu erzeugen. Zu diesem Zweck wird die nachfolgende Frage beantwortet.

Verbindung der Löcher mit dem Rand.

Frage F: Welches Objekt wird als erstes von einem (expandierenden) Kreis berührt, der die durch den Punkt p verlaufende Vertikale in dem Punkt p tangiert und links der Vertikalen liegt? (siehe Abb. 4.21)

(1) Die Objektmenge O besteht anfangs aus allen Objekten, die den Rand des Polygons bilden, und aus den Kreisen, die den Rand berühren.

(2) Alle ein Loch berührenden Kreise werden als zu dem Loch zugehörig betrachtet.

(3) ALL sei die Menge aller Kanten die ein Loch beschreiben.
Solange $ALL \neq \emptyset$

> Wähle einen am weitesten links liegenden Punkt p innerhalb von ALL und beantworte die Frage F für diesen Punkt.
>
> Füge den so gefundenen Kreis und alle zu dem betrachteten Loch gehörenden Kanten in die Menge O ein, wobei diese Kanten aus der Menge ALL zu entfernen sind.

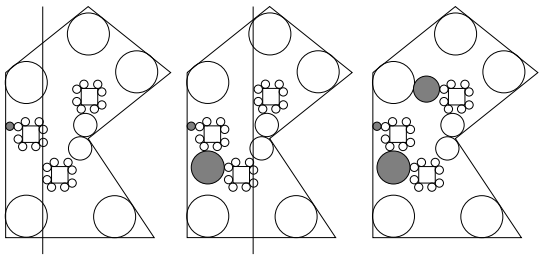


Abbildung 4.21: Aufgrund der Löcher einzufügende Kreise

Das Polygon ist durch das Einfügen der Kreise in eine Menge von Restflächen zerfallen, die jede mindestens 3 Seiten (Kanten) besitzt. Als nächstes werden diese Restflächen durch das Einfügen von weiteren Kreisen solange unterteilt, bis jede Restfläche maximal 4 Seiten hat.

Erzeugung von drei bzw. vierseitigen Restflächen

Solange es eine Restfläche mit $n \geq 5$ Seiten gibt, wähle einen Kreis, der drei Seiten der Restfläche tangiert, welche nicht alle miteinander verbunden sind, und füge diesen Kreis ein (siehe Abb. 4.22).

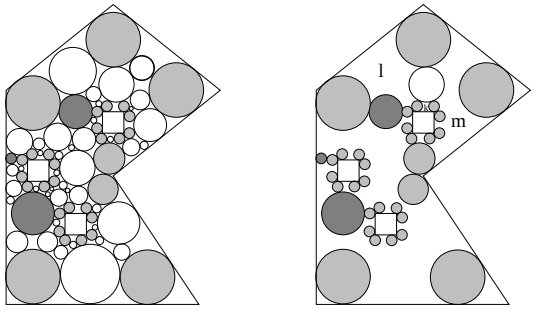


Abbildung 4.22: Eine Restfläche darf maximal vier Seiten besitzen.

Lemma 4.4.4 Ein n-seitiges Polygon läßt sich mit $O(n)$ Kreisen in drei und vierseitige Restflächen zerlegen.

Beweis:

Sei P eine polygonal berandete Fläche mit $n \geq 5$ Kanten. Das GVD von P enthält in diesem Fall mindestens einen Knoten v, der von 3 Kanten die Distanz r besitzt, wobei die Kanten nicht alle miteinander verbunden sind (siehe Abb. 4.22). Ein Kreis C mit Mittelpunkt v und Radius r teilt P in $h \geq 3$ (O.B.d.A. $h = 3$) Restflächen mit k, l, m Seiten ($3 \leq k \leq l \leq m < n$) und es gilt $k+l+m-6 = n$. Sei $d(n)$ die minimale Anzahl von Kreisen, die benötigt wird, um P in drei und vierseitige Restflächen zu zerlegen.

Behauptung: $d(n) \leq n - 4$

IA: $n = 3, n = 4$. Es gilt $d(3) = d(4) = 0$.

IV: Für $i < n$ gelte die Behauptung.

(1) $k = 3$

Da die drei Berührungskanten nicht miteinander verbunden sind, folgt direkt $l \geq 4$.

IS: $d(n) \leq 1 + d(k) + d(l) + d(m) \leq 1 + 0 + (l-4) + (m-4) = (l+m) - 7 = (n+3) - 7 = n - 4$

(2) $k > 3$

IS: $d(n) \leq 1+d(k)+d(l)+d(m) \leq 1+(k-4)+(l-4)+(m-4) = (k+l+m)-11 = (n+6)-11 = n-5$ ∎

Nachdem das Polygon mit Kreisen gefüllt ist, wird jetzt unter mit Hilfe dieser Kreise eine Triangulierung angegeben.

Die Triangulierung

Bezeichnung 4.4.5 erweiterte Restfläche

1. Füge die Kreismittelpunkte und die Berührungspunkte als Steinerpunkte ein.
2. Füge die Verbindung des Kreismittelpunktes mit allen Berührungspunkten, die der Kreis besitzt, als neue Kanten ein.

Die so entstandenen Partitionen des Polygons werden als erweiterte Restflächen bezeichnet (siehe Abb. 4.23).

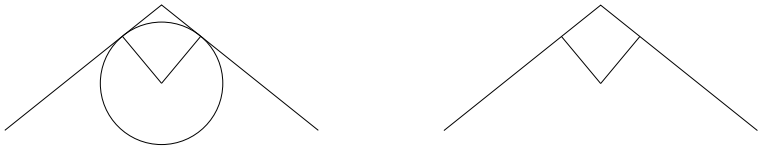


Abbildung 4.23: Eine erweiterte vierseitige Restfläche.

Lemma 4.4.6 Eine dreiseitige Restfläche kann mit maximal 6 rechtwinkligen Dreiecken trianguliert werden.

Lemma 4.4.7 Eine vierseitige Restfläche kann mit maximal 28 rechtwinkligen Dreiecken trianguliert werden.

Der Beweis von beiden Lemmata erfolgt durch die Angabe der Triangulierungen.

Dreiseitige Restfläche mit Knotenpunkt des Polygons

Eine erweiterte vierseitige Restfläche wird in zwei rechtwinklige Dreiecke zerlegt, indem der Knotenpunkt des Polygons und des Kreismittelpunktes miteinander verbunden werden (siehe Abb. 4.24).

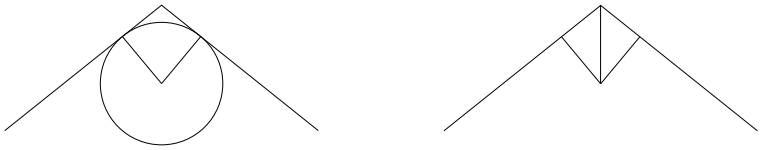


Abbildung 4.24: Triangulierung einer dreiseitigen Restfläche mit Knotenpunkt.

Vierseitige Restfläche mit Knotenpunkt des Polygons

Eine erweiterte fünfseitige Restfläche wird in vier rechtwinklige Dreiecke zerlegt, indem der Knotenpunkt des Polygons mit den beiden Kreismittelpunkten und dem Berührungspunkt der Kreise verbunden wird (siehe Abb. 4.25).

Dreiseitige Restfläche mit einer Polygonseite

Die erweiterte Restfläche läßt sich in vier rechtwinklige Dreiecke zerlegen, indem der Schnittpunkt der Kreistangente in dem Berührungspunkt der beiden Kreise mit der Polygonseite

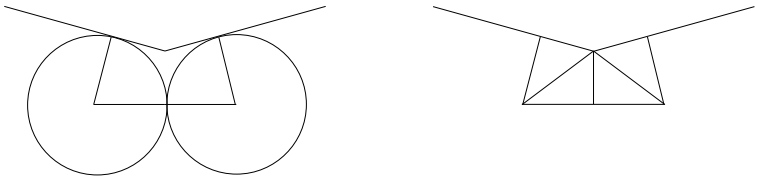


Abbildung 4.25: Triangulierung einer vierseitigen Restfläche mit Knotenpunkt.

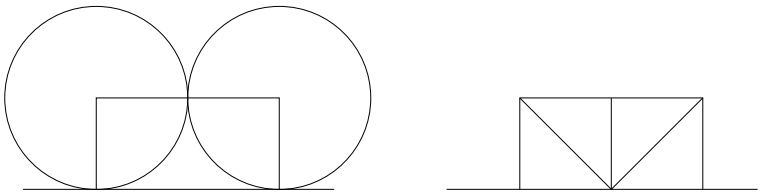


Abbildung 4.26: Triangulierung einer dreiseitigen Restfläche mit einer Polygonseite.

als Steinerpunkt eingefügt und mit den beiden Kreismittelpunkten und dem Berührungspunkt verbunden wird (siehe Abb. 4.26).

Die Powerfunktion

Für einen Kreis C mit dem Radius r und dem Mittelpunkt (x_m, y_m) ist die Powerfunktion wie folgt definiert: $f(x,y) := (x-x_m)^2 + (y-y_m)^2 - r^2$

Seien jetzt C_1, C_2 zwei Kreise mit den Radien r_1, r_2 und den Mittelpunkten (x_1, y_1), (x_2, y_2). Falls sich die Kreise C_1, C_2 im Punkt b berühren, sei T die Kreistangente im Punkt b. Falls sich die Kreise C_1, C_2 in den Punkten s_1, s_2 schneiden, sei T die Gerade, auf der die zugehörige Sekante liegt, und b der Schnittpunkt von T mit der Geraden, die durch die Kreismittelpunkte bestimmt ist. Wir werden zeigen, daß die Powerfunktion der beiden Kreise für die Punkte der Tangente (Sekante) identisch ist. Mit dieser Aussage ist dann die Gültigkeit der nachfolgenden Triangulierung offensichtlich.

Behauptung: $\forall (x,y) \in T : f^{C_1}(x,y) = f^{C_2}(x,y)$

Beweis:
Sei l die Distanz von b zu (x,y).

$$\begin{aligned} f^{C_1}(x,y) &= (x-x_1)^2 + (y-y_1)^2 - r_1^2 \\ &= l_1^2 - r_1^2 \\ &= l^2 \\ &= l_2^2 - r_2^2 \\ &= (x-x_2)^2 + (y-y_2)^2 - r_2^2 \\ &= f^{C_2}(x,y) \end{aligned}$$

■

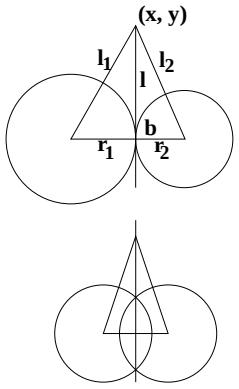


Abbildung 4.27: Die Powerfunktion.

Dreiseitige Restfläche ohne Polygonseite

Die erweiterte Restfläche läßt sich in sechs rechtwinklige Dreiecke zerlegen, indem der Schnittpunkt der drei Kreistangenten in den Berührungspunkten als Steinerpunkt eingefügt und mit den Kreismittelpunkten und den Berührungspunkten verbunden wird (siehe Abb. 4.28).

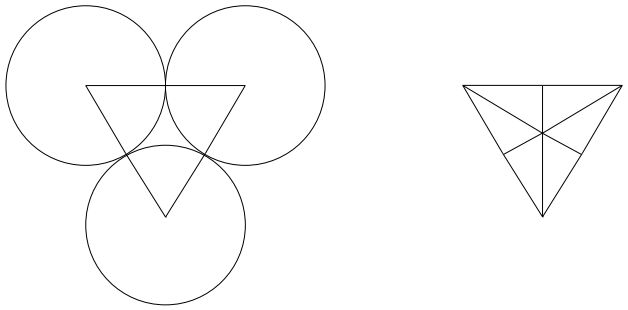


Abbildung 4.28: Triangulierung einer dreiseitigen Restfläche ohne Polygonseite.

Vierseitige Restfläche ohne Polygonseite

Satz 4.4.8 Die vier Berührungspunkte liegen auf einem Kreis (siehe Abb. 4.33).

Beweisskizze:

Die gemeisamen Tangenten zweier gegenüberliegender Kreise schneiden sich (siehe Abb. 4.29).

Transformiere den Schnittpunkt ins unendliche.

Die beiden Kreise sind nach der Transformation gleich groß, und der Bisektor dieser Kreise ist eine Gerade.

Die Bisektoren der Berührungspunkte schneiden sich im Kreismittelpunkt des gesuchten Kreises.

Rücktransformation liefert den Kreis. ■

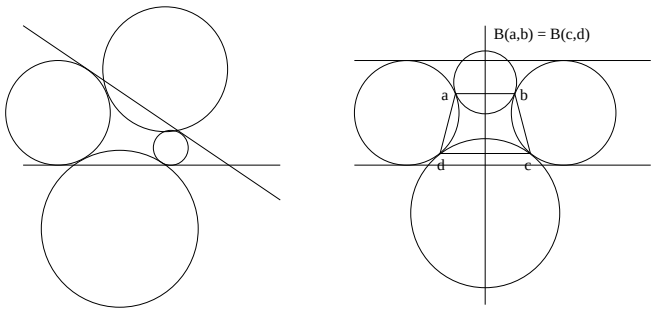


Abbildung 4.29: Die vier Berührungspunkte liegen auf einem Kreis.

Bezeichnung 4.4.9 Zentrierte bzw. unzentrierte Restfläche

Liegt der Mittelpunkt des Kreises, auf dem die vier Berührungspunkte liegen, innerhalb der konvexen Hülle der Restfläche, so wird diese als zentriert und ansonsten als unzentriert bezeichnet.

Eine Seite der Restfläche ist größer als π mal Radius

Sei diese Seite ein Teilstück des Kreises C_4, und seien C_1, C_3 die beiden Kreise, die C_4 berühren. Durch das Einfügen von einem neuen Kreis C_*, der C_2 und C_4 tangiert, wobei die Mittelpunkte aller drei Kreise auf einer Gerade liegen, kann es zu folgen zwei Situationen kommen:

(1) Der Kreis C_* schneidet keinen der anderen Kreise (siehe Abb. 4.30).

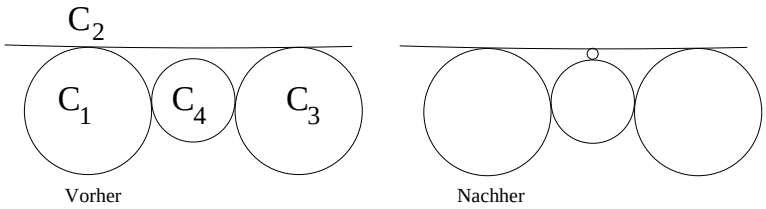


Abbildung 4.30: Reduzierung auf eine Bogenlänge kleiner als π.

In diesem Fall werden die neuen Restflächen wie alle anderen Restflächen behandelt.

(2) Der Kreis C_* schneidet einen der anderen Kreise (siehe Abb. 4.31).

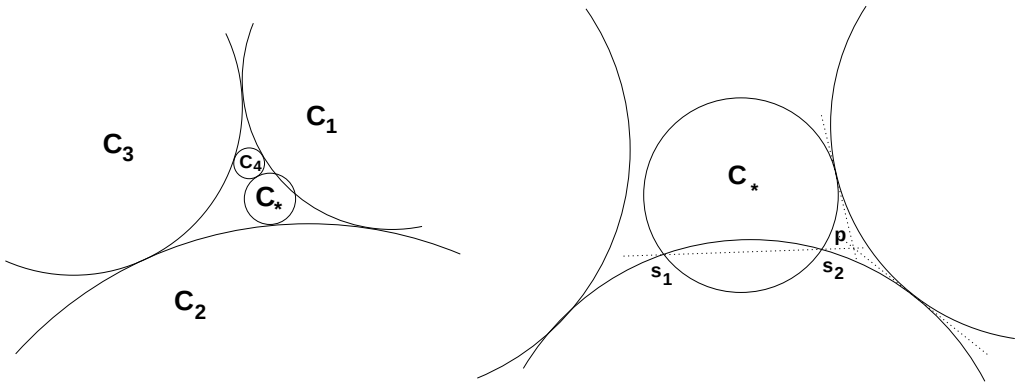


Abbildung 4.31: Der Kreis C_* schneidet einen der anderen Kreise.

Seien s_1, s_2 die Schnittpunkte von C_* und C_1. Sei p der Schnittpunkt der beiden Kreistangenten in den Berührungspunkten von C_4, C_1 und C_*, C_4 bzw. C_1, C_2 und C_*, C_2.

Behauptung: s_1, s_2 und p liegen auf einer Geraden.

Beweis:

Sei $f(x, y) = (x-x_c)^2+(y-y_c)^2-r^2$ die Powerfunktion der Kreise gegeben. Dabei ist (x_c, y_c) der Kreismittelpunkt und r der Kreisradius. Die Powerfunktion zweier sich tangierender Kreise ist gleich für alle Punkte der gemeinsamen Tangente. Für zwei sich schneidende Kreise ist sie gleich für alle Punkte der Geraden, die durch die Schnittpunkte bestimmt ist. Der Punkt p ist genau der Punkt, an dem die drei Powerfunktionen gleich sind. ■

Die Behauptung ermöglicht die in Abbildung 4.32 angegebene Triangulierung.

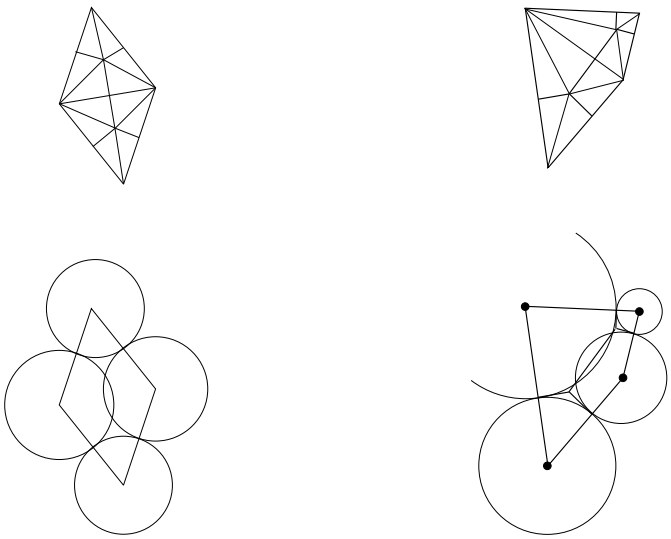


Abbildung 4.32: Zerlegung in zwölf rechtwinklige Dreiecke.

Die zentrierte Restfläche

Die erweiterte Restfläche läßt sich in sechszehn rechtwinklige Dreiecke zerlegen (siehe Abb. 4.33). Füge den Kreismittelpunkt, auf dem die vier Berührungspunkte liegen, als Steinerpunkt ein und verbinde diese Punkte mit den Berührungspunkten und den Kreismittelpunkten. Verbinde benachbarte Berührungspunkte miteinander und füge die entstehenden Schnittpunkte als Steinerpunkte ein. An diesen Schnittpunkten liegen die rechten Winkel der Dreiecke.

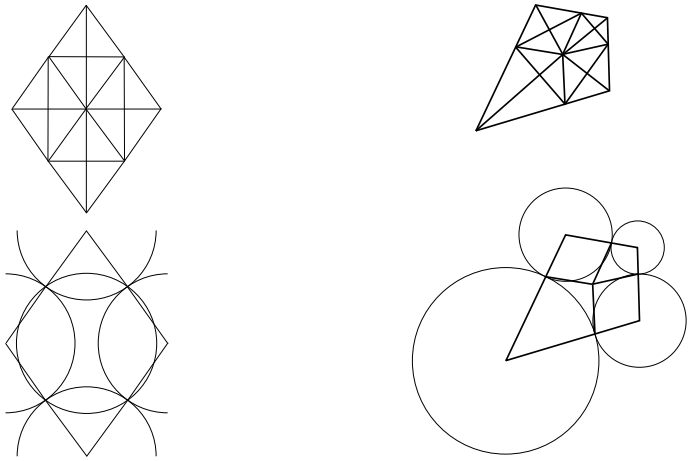


Abbildung 4.33: Triangulierung einer zentrierten Restfläche.

Die unzentrierte Restfläche

Sei $t_{i,j}$ der Berührungspunkt der Kreise C_i und C_j, $i, j \in \{*, 1, 2, 3, 4\}$.

Satz 4.4.10 Es gibt einen Kreis C_* derart das gilt:

(1) Der Kreis C_* berührt die Kreise C_1, C_3 in den Punkten $t_{*,1}$ und $t_{*,3}$.

(2) Die Punkte $t_{4,1}, t_{*,1}$, $t_{*,3}$ und $t_{3,4}$ liegen auf einem Kreis mit dem Mittelpunkt C_l

(3) Der Mittelpunkt C_l liegt auf der konvexen Hülle der vier Punkte.

(4) Die Punkte $t_{1,2}, t_{*,1}$, $t_{*,3}$ und $t_{2,3}$ liegen auf einem Kreis mit dem Mittelpunkt C_r

(5) Der Mittelpunkt C_r liegt auf der konvexen Hülle der vier Punkte.

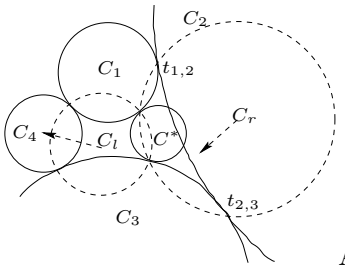


Abbildung 4.34: Auf der Suche nach dem richtigen Kreis C_*.

In diesem Fall kann die Restfläche in achtundzwanzig rechtwinklige Dreiecke zerlegt werden. Seien s_1, s_2 die Schnittpunkte der Kreise $''C_l, C_r''$. Es werden die Steinerpunkte C_l, C_r, s_1, s_2 eingefügt und miteinader verbunden. Der Steinerpunkt C_l wird mit den Berührpunkten $t_{4,1}, t_{4,3}$ und der Steinerpunkt C_r mit den Punkten $t_{1,2}, t_{2,3}$ verbunden. s_1 bzw. s_2 wird mit dem Kreismittelpunkt von C_1 bzw. C_3 verbunden. Die so entstandenen Vierecke werden durch Einfügen der Diagonalen in rechtwinklige Dreiecke zerlegt wobei der Schnittpunkt als Steinerpunkt eingefügt werden muß (siehe Abb. 4.35).

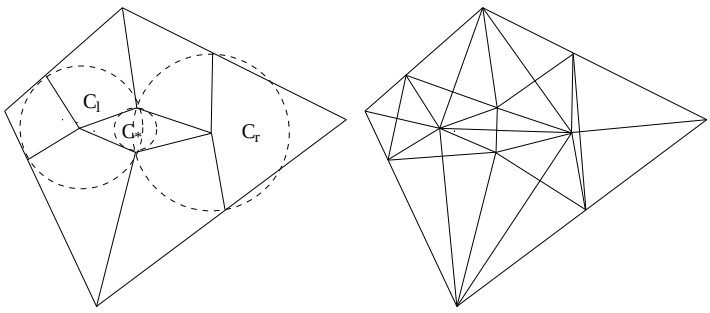


Abbildung 4.35: Die Triangulierung einer unzentrierten Restfläche.

Der Aufwand

	sequentiell
1. Fülle das Polygon mit Kreisen geeigneter Größe	
a) Platzierung von Kreisen an den Ecken	$O(n \log(n))$
b) Verbindung der Löcher mit dem Rand	$O(n \log^2(n))$
c) Auffüllen der Restflächen durch Kreise	$O(n \log(n))$
2. Trianguliere die Restflächen	$O(n)$
	$\bar{O}(n \log^2(n))$

	parallel mit $O(n^2)$ Prozessoren:
1. Fülle das Polygon mit Kreisen geeigneter Größe	
a) Platzierung von Kreisen an den Ecken	$O(\log(n))$
b) Verbindung der Löcher mit dem Rand	$O(\log^3(n))$
c) Auffüllen der Restflächen durch Kreise	$O(\log^3(n))$
2. Trianguliere die Restflächen	$O(1)$
	$\bar{O}(\log^3(n))$

Satz 4.4.11 Eine polygonal berandete Fläche ohne Löcher die n Knoten besitzt kann mit $O(n)$ rechtwinkligen Dreiecken in $O(n \log(n))$ Zeit trianguliert werden.

Satz 4.4.12 Eine polygonal berandete Fläche mit Löchern die n Knoten besitzt kann mit $O(n)$ rechtwinkligen Dreiecken in $O(n \log^2(n))$ Zeit trianguliert werden.

Satz 4.4.13 Eine polygonal berandete Fläche mit Löchern die n Knoten besitzt kann mit $O(n)$ rechtwinkligen Dreiecken parallel mit $O(n^2)$ Prozessoren in $O(\log^3(n))$ Zeit trianguliert werden.

Die Gültigkeit der Sätze folgt direkt aus den vorherigen Angaben.

4.5 Eine Oberflächentriangulierung

Dieser Algorithmus von L. P. Chew beschreibt ein Verfahren zur Triangulierung von zweidimensionalen Gebieten bzw. Oberflächen, welche durch ein Polygon approximiert werden und wurde in dem Paper "Guaranteed-Quality Mesh Generation for Curved Surfaces" vorgestellt [19]. Die erzeugte Triangulierung besteht aus Dreieckselementen mit Winkeln zwischen 30° und 120° Grad und alle Elemente erfüllen ein vom Benutzer definiertes Größenkriterium.

Das Verfahren kann kurz wie folgt beschrieben werden:

Zuerst wird die verallgemeinerte Delaunay Triangulierung (CDT) der Eingabe erzeugt und anschließend werden Elemente, die nicht die vorgegebenen Gütekriterien erfüllen, bearbeitet, indem Steinerpunkte eingefügt werden.

Zunächst erfolgt nun eine genaue Beschreibung des Algorithmus für polygonal berandete zweidimensionale Gebiete. Damit sich dieses Verfahren auch auf gekrümmte Oberflächen anwenden lässt, wird anschließend die Definition der CDT erweitert und der leicht veränderte Algorithmus angegeben.

Im Folgenden wird von einem gegebenen Eingangsgraphen $G(V, E)$ mit der Knotenmenge V und der Kantenmenge E ausgegangen, welcher das zu triangulierende Gebiet beschreibt. Die Knoten und die Kanten werden als Eingangsobjekte bezeichnet. Der Abstand zweier Objekte $[dist(v, w)]$ ist als der minimale euklidische Abstand definiert. Zwei Punkte (v, w) werden als sichtbar bezeichnet falls das durch die beiden Punkte gebildete Segment keine Eingangskante, ausgenommen in den Endpunkten, schneidet. Eine geschnittene Eingangskante wird als blockierendes Segment bezeichnet. Der sichtbare Abstand zweier Punkte $[dist_s(v, w)]$ ist gleich dem euklidischen Abstand falls die Punkte sichtbar sind und ansonsten unendlich. Der Algorithmus unterscheidet zwei verschiedene Knotentypen. Die Menge der S-Knoten besteht aus allen Steinerpunkten, die auf einer Eingangskante liegen, und aus der vorgegebenen Knotenmenge V. Die Menge der C-Knoten wird von allen Steinerpunkten gebildet, die nicht auf einer Eingangskante liegen.

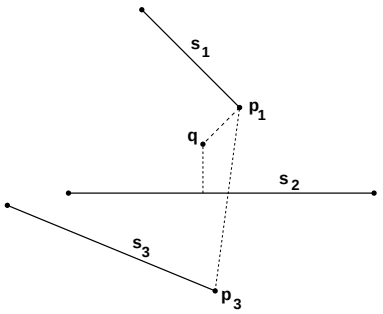


Abbildung 4.36: Ein Eingabeobjekt, die Sichtbarkeit und die Distanz

Die Punkte p_1 und p_2 sind nicht sichtbar, da das durch die Punkte beschriebene Segment (gestrichelt) das Eingabesegment s_2 schneidet. Damit ist s_2 das blockierende Segment. Die Distanz des Punktes q zu den Segmenten s_1, s_2 ist durch die Länge der gestrichelt eingezeichneten Segmente gegeben.

Für eine einfachere und übersichtliche Algorithmenbeschreibung werden folgende Operationen definiert:

(1) $Build(V, E)$

$Build(V, E)$ erzeugt die CDT des gegebenen Graphen.

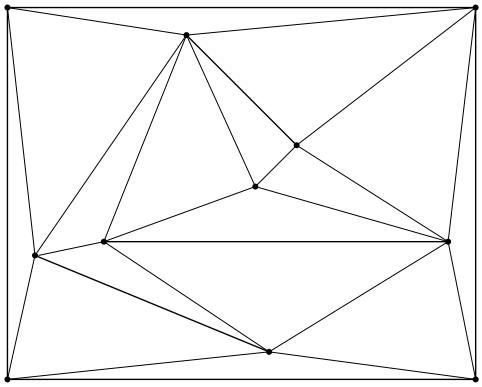


Abbildung 4.37: Die CDT des Eingabeobjektes

(2) $Insert(k)$

$Insert(k)$ fügt den Knoten k in die CDT ein und erzeugt wieder eine gültige CDT.

(3) $Delete(k)$

$Delete(k)$ entfernt den Knoten k aus der CDT und erzeugt wieder eine gültige CDT.

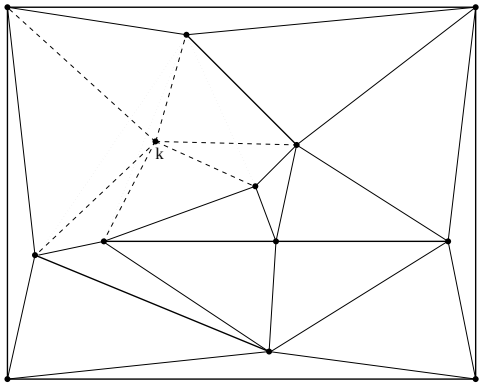


Abbildung 4.38: Die Triangulierung nach dem Einfügen bzw. Löschen eines Knotens

Durch das Einfügen von k werden die gestrichelten Kanten eingefügt. Alle Kanten die geschnitten werden (gepunktet) sind zu entfernen. Eine Löschung von k stellt die vorherige Triangulierung wieder her.

(4) $Split(e, x)$

$Split(e, x)$ unterteilt die Kante e in $x \in \{2, 3\}$ gleichlange Teilkanten. Die alte Kante e wird dabei durch die neuen Kanten ersetzt und eine gültige CDT erzeugt.

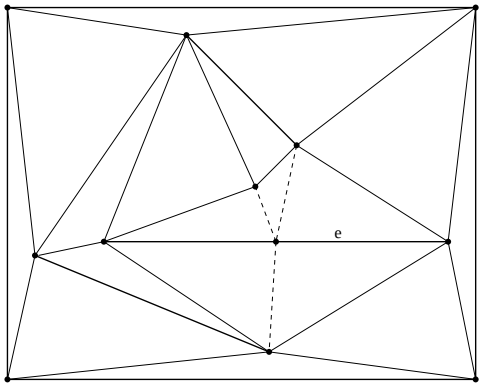


Abbildung 4.39: Die Unterteilung einer Kante

(5) $Travel(s, d)$

$Travel(s, d)$ bestimmt das Dreieckselement in dem der Knoten d liegt falls die beiden Knoten s, d sichtbar sind. Andernfalls wird die erste Kante $e \in E$ bestimmt welche die Sichtbarkeit von s blockiert.

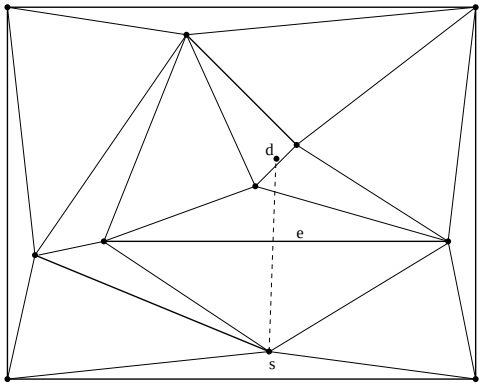


Abbildung 4.40: Bestimmung einer blockierenden Kante

Definition 4.5.1 well-shaped

Ein Dreieck t heißt well-shaped, falls alle Winkel zwischen 30° und 120° Grad liegen.

Definition 4.5.2 well-sized

Ein Dreieck t heißt well-sized, falls es ein gegebenes Qualitätskriterium f erfüllt.

Das Qualitätskriterium f ist ein beliebiges vom Benutzer vorgegebenes Maß welches jedoch folgende Bedingung erfüllen muss:

Bedingung 4.5.3 Qualitätsbedingung

Das Qualitätskriterium f ist nur gültig falls gilt:

Es existiert ein $\delta > 0$ und alle well-shaped Dreiecke mit Umkreisradius $r \leq \delta$ erfüllen f.

Definition 4.5.4 δ_{max}, l_{min}, h

Sei δ_{max} der größte δ-Wert für den das Qualitätskriterium erfüllt werden kann.

Sei l_{min} der kleinste Abstand $\neq 0$ zweier Eingabeobjekte.

$h := minimum(l_{min}, \delta_{max})$

Beispiele für f

(1) Der Umkreis jedes Dreiecks muss kleiner als $x \in \mathbb{R}^+$ sein.

(2) Jede Kante der Triangulierung die auf einer Eingangskante liegt muss eine Länge kleiner als $x \in \mathbb{R}^+$ haben.

Algorithmus A

Sei T die Menge der Dreieckselemente der CDT und $U \subset T$ seien die Elemente, welche nicht well-shaped oder well-sized sind. Sei $u \in U$, p_u ein Eckpunkt von u, $C(u)$ der (kleinste) Umkreis von u, $C_r(u)$ der Radius des Umkreises, $C_m(u)$ der Mittelpunkt des Umkreises und $C_r(U) := maximum\{C_r(u) | u \in U\}$.

(1) $Build(V, E)$.

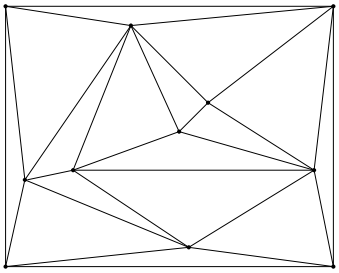
(2) Falls alle Dreiecke well-shaped und well-sized sind dann halte.

(3) Bestimme $C_r(U)$ und ein Element $u \in U$ mit $C_r(u) = C_r(U)$.

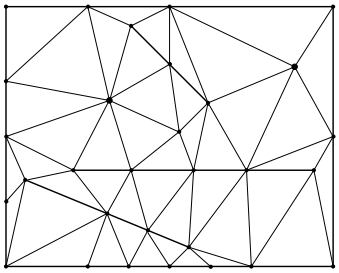
(4) $Travel(p_u, C_r(u))$.

(5) Es wurde ein Dreieck ermittelt das $C_m(u)$ enthält:
$Insert(C_m(u))$;
goto 2.

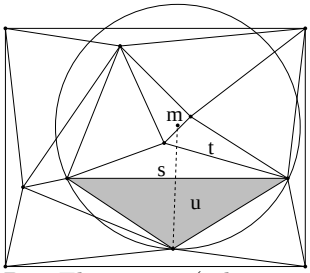
(6) Es wurde eine blockierende Kante e ermittelt:
Sei l die Länge der Kante e.
$Split(e, 3)$, falls $2 * \sqrt{3} * h \leq l \leq 4 * h$;
$Split(e, 2)$ sonst.
Sei l_{neu} die Länge der erzeugten Kanten und M die Menge der C-Knoten mit einem sichtbaren Abstand zu einem neuen S-Knoten der kleiner als l_{neu} ist.
$Delete(k)\ \forall\ k \in M$;
goto 2.

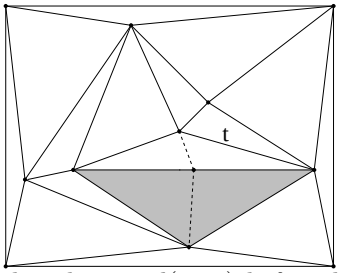


$Build(V, E)$ erzeugt die initiale CDT.

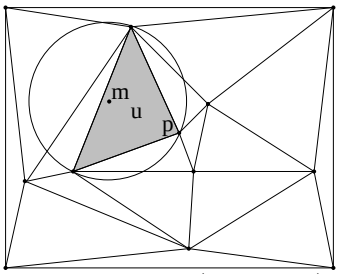


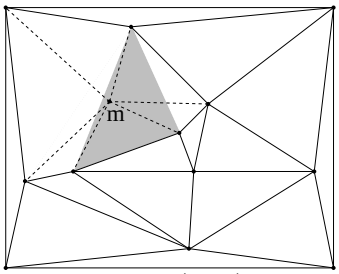
Die erzeugte abschließende Tringulierung.



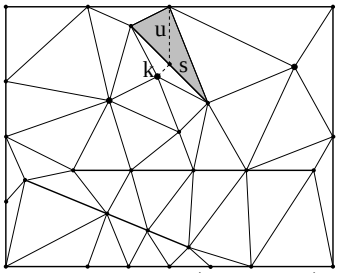


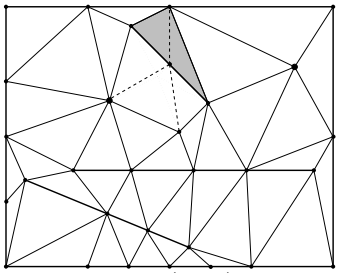
Das Element u (schattiert) ist nicht well sized und $Travel(m, p)$ liefert das blockierende Segment s. Dadurch wird mit $Split(m, 2)$ das Segment s halbiert. Die entstandene Triangulierung ist keine CDT. Daher wird die Kante t geflippt.





Das Element u (schattiert) ist nicht well sized und $Travel(m, p)$ liefert das Element u. Dadurch wird mit $Insert(m)$ der Mittelpunkt m eingefügt.





Das Element u (schattiert) ist nicht well sized und $Travel(m, p)$ liefert das blockierende Segment s. Dadurch wird mit $Split(m, 2)$ das Segment s halbiert. Jedoch liegt hier der S_Knoten k näher an dem Segment s als die halbe Seitenlänge von s erlaubt. Daher wird der Knoten k wird mit $Delete(k)$ aus der Triangulierung entfernt.

Abbildung 4.41: Der Ablauf des Algorithmus an Hand eines Beispiels

Lemma 4.5.5

Der Radius $C_r(u)$ ist größer oder gleich der Länge der kürzesten Seite von u, falls u einen Winkel kleiner oder gleich 30° besitzt.

Beweis:

Sei α der kleinste Winkel und a die kürzeste Seite von u. Damit gilt:

$0 < \alpha \leq 30° \Rightarrow 0.5 \geq \sin\alpha \Rightarrow \frac{1}{2*\sin\alpha} \geq 1 \Rightarrow C_r(u) = \frac{a}{2*\sin\alpha} \geq a$ ■

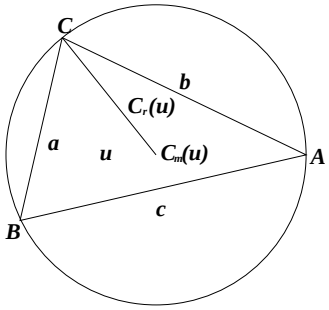


Abbildung 4.42: $C_r(u) \geq$ der Länge der kürzesten Seite

Lemma 4.5.6

Vor Ausführung von Schritt 2 im Algorithmus gilt immer $dist_s(v, w) \geq h \; \forall v, w \in V, v \neq w$.

Beweis:

IA: Nach $Build(V, E)$ gilt Lemma 4.5.6 da es sich um eine CDT handelt.

IV: Nach i Schleifendurchläufen gelte Lemma 4.5.6.

IS: (a) Im $i+1-$ten Durchlauf wird $m := C_m(u)$ eingefügt:
Es gilt: $dist_s(m, v) \geq C_r(u) \; \forall v \in V_{CDT}$.
Behauptung: $C_r(u) \geq h$.
Beweis:

i. u ist nicht well-shaped.
Nach Lemma 4.5.5 gilt $C_r(u) \geq$ kürzeste Seite von $u \geq h$.

ii. u ist nicht well-sized aber well-shaped.
Aus $C_r(u) < h$ und der Definition von dem Qualitätskriterium f folgt direkt u ist well-sized. Folglich gilt $C_r(u) \geq h$. ■

(b) Im $(i+1)-$ten Durchlauf wird die Kante e geteilt:
Sei l die Länge der Kante e.

i. $l > 4 * h$
$\Rightarrow \frac{l}{2} > 2 * h$

ii. $2 * \sqrt{3} * h \leq l \leq 4 * h$
$\Rightarrow h < \frac{2*\sqrt{3}}{3} * h \leq \frac{l}{3} \leq \frac{4}{3} * h \leq \sqrt{3} * h$

iii. $2 * h \leq l \leq 2 * \sqrt{3} * h$
$\Rightarrow h \leq \frac{l}{2} \leq \sqrt{3} * h$

iv. $h \leq l \leq \sqrt{3} * h$
Behauptung: e wird nicht geteilt.
Beweisskizze:
Sei d die Länge von e, und sei t das Dreick mit den Knoten a, b, c, welches die Teilung von e verursacht. Da $C_r(t)$ und die Knoten von t auf verschiedenen Seiten von e liegen, müssen die Knoten von t innerhalb des Halbkreis C_0 über der Kante e liegen. Gleichzeitig kann aber keiner der Knotenpunkte innerhalb der Kreise C_1, C_2 vom Radius h liegen, deren Mittelpunkte die Endpunkte der Kante e sind, da nach Induktionsvoraussetzung zwei Knoten einen Abstand größer als h haben. Die Restfläche $R = C_0 - C_1 - C_2$ hat jedoch einen Durchmesser kleiner als h, womit nur ein Knotenpunkt innerhalb dieser Fläche liegen kann. Daher seien o.B.d.A. a, b Endpunkte von e. Für jeden Punkt $v \in R$ gilt jedoch: Das Dreieck t' mit den Knoten a, b, v ist well-sized und hat eine Umkreisradius kleiner als h und ist somit auch well-shaped. Damit gehört t nicht zur Menge U und kann keine Teilung der Kante e bewirken. ■

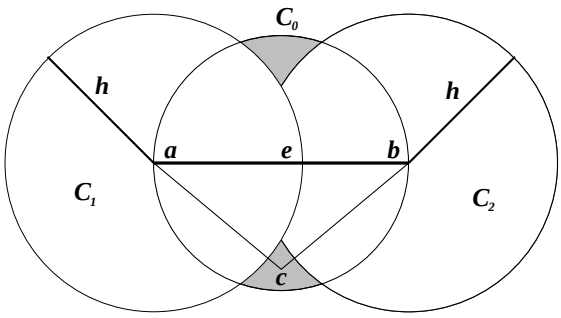


Abbildung 4.43: Der Knoten c liegt innerhalb der schattierten Region

Satz 4.5.7 Der Algorithmus hält.

Beweis:

Folgt direkt aus Lemma 4.5.6. ■

Eigenschaften des Algorithmus.

(1) Alle vom Benutzer vorgegebenen Knoten und Kanten sind in der Triangulierung enthalten.

(2) Das erzeugte Netz ist konform.

(3) Alle erzeugten Elemente besitzen Winkel zwischen 30° und 120°, ausgenommen kleinerer Eingabewinkel.

(4) Der Benutzer kann die Elementgröße durch das Qualitätskriterium kontrollieren.

Um diesen Algorithmus auf gekrümmte Flächen anwenden zu können ist es erforderlich sowohl die CDT einer gekrümmten Fläche zu definieren als auch den Umkreis (und dessen Mittelpunkt) eines Dreieckselementes dieser Fläche. Beide Definitionen erfordern eine Kreisdefinition für eine gekrümmte Fläche, aber wie kann der Kreis durch drei Punkte auf einer gekrümmten Fläche definiert werden?

In der Ebene ist ein Kreis durch drei Punkte eindeutig definiert und im dreidimensionalen Raum existiert eine unendliche Menge von Kugeln auf denen die Punkte liegen, wobei die Mittelpunkte eine Geraden bilden. Diese Eigenschaft wird für die Definition eines Kreises in der gekrümmten Fläche ausgenutzt.

Definition 4.5.8 Kreis in der gekrümmten Fläche F

Seien drei Punkte p, q, o auf F gegeben und sei g die Gerade im Raum mit gleichem Abstand von diesen drei Punkten.

Seien $S = \{s1, .., s_n\}$ die Schnittpunkte von g und F und $m \in S$ sei der nächstgelegene Schnittpunkt, d. h. $dist_g(m, p) \leq dist_g(s, p)\ \forall s \in S$.

m wird dann als der Mittelpunkt des Umkreises definiert, und der Kreis selber wird als der Schnitt der entsprechenden Kugel mit der Fläche definiert.

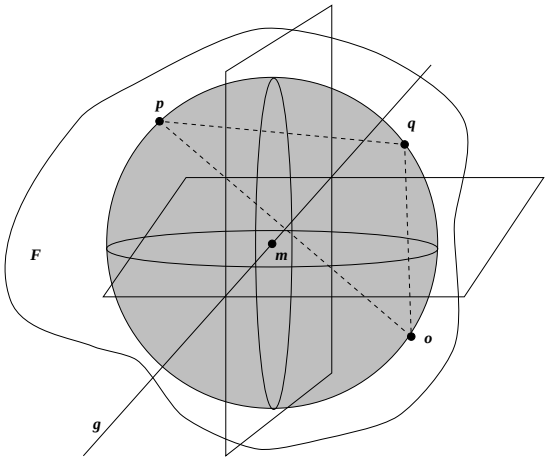


Abbildung 4.44: Kreis einer gekrümmten Fläche

Eigenschaften der Kreisdefinition

+ Es ist einfach festzustellen, ob ein gegebener Punkt innerhalb oder außerhalb eines Kreises liegt.
+ Der Mittelpunkt des Kreises liegt immer auf der gekrümmten Fläche.
+ In den meisten Darstellungen (z.B. Parameter-) ist es relativ einfach möglich den gesuchten Kreismittelpunkt zu bestimmen.
– Die Gerade g kann F mehr als einmal schneiden.
– Es gibt keinen eindeutigen Schnittpunkt.

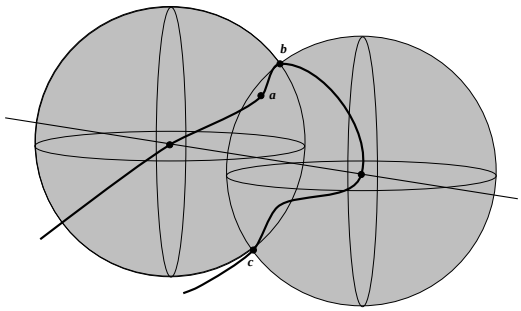


Abbildung 4.45: Der Kreismittelpunkt ist nicht eindeutig

– Die Gerade g muss F nicht schneiden.

Dass es sich hier um eine sinnvolle Kreisdefinition handelt, geht aus den nachfolgenden Betrachtungen hervor. Sie zeigen, dass die negativen Eigenschaften bei den betrachteten Flächen entweder nicht auftreten oder vermieden werden können.

Definition 4.5.9 Konsistenz

Seien t_1, t_2 zwei benachbarte Dreiecke mit den Knoten a, b, c und a, b, d.

Zwei benachbarte Dreiecke heißen konsistent falls gilt:

$$d \in C(t_1) \Rightarrow a \in C(t_2)$$

Eine Triangulierung heißt konsistent, wenn alle benachbarten Dreiecke der Triangulierung konsistent sind.

Satz 4.5.10 Jede 2D Triangulierung ist konsistent.

Beweis:

Seien t_1, t_2 beliebige benachbarte Dreiecke mit den Knoten a, b, c und b, c, d. Weiter gelte $d \in C(t_1)$. Sei G die Gerade mit $dist(x, b) = dist(x, c)\ \forall x \in G$.

$r = C_r(t_1)$ teilt G in zwei Teile A, B, für die offensichtlich folgende Aussagen gelten:

(1) $\forall x \in A - \{r\} : dist(x, a) < dist(x, b)$

(2) $\forall x \in B - \{r\} : dist(x, a) > dist(x, b)$

(3) $\forall x \in B : dist(x, d) < dist(x, b)$

$\Rightarrow C_r(t_2) \in A$

$\Rightarrow dist(C_r(t_2), a) < dist(C_r(t_2), b)$

$\Rightarrow a \in C(t_2)$ ■

Satz 4.5.11

Jede $2\frac{1}{2}$ D Triangulierung ist konsistent, falls sich die Normalenvektoren höchstens um $\frac{\pi}{2}$ unterscheiden.

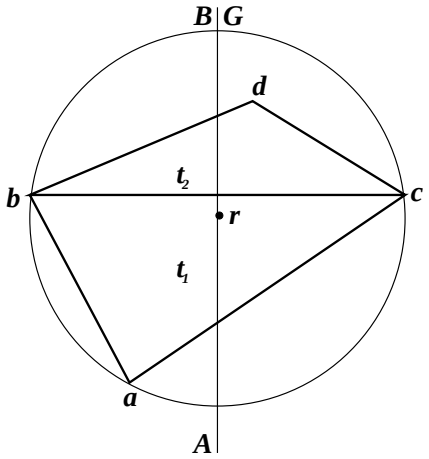


Abbildung 4.46: Konsistenz $2D$

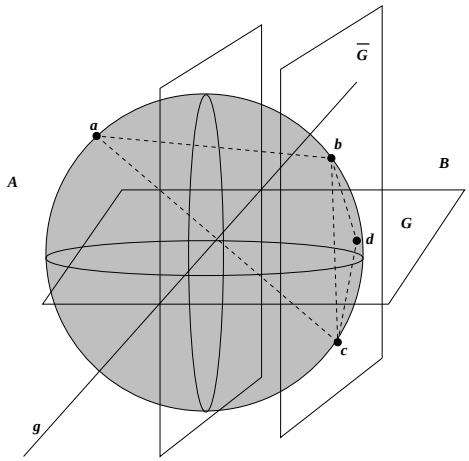


Abbildung 4.47: Konsistenz $2\frac{1}{2}$ D

Beweisskizze:

Seien t_1, t_2 beliebige benachbarte Dreiecke mit den Knoten a, b, c und b, c, d, und es gelte $d \in C(t_1)$, wobei $C(t_1)$ hier die Kugel beschreibt. Weiter sei G die Fläche mit $dist(x, b) = dist(x, c)\ \forall x \in G$, und $\overline{G}$ sei die Fläche senkrecht zu G, auf der die Punkte b, c liegen. $\overline{G}$ teilt G in zwei Teilflächen und den Raum in zwei Teilräume A, B mit $a \in A, d \in B$. Wird der Kugelmittelpunkt nun in Richtung des Punktes d verschoben, wobei er in der Fläche G und innerhalb von $C(t_1)$ liegt, so vergrößert sich die Halbkugel im Raum B und d bleibt innerhalb der Kugel liegen. Andererseits vergrößert sich die Kugel im Raum A, wenn der Mittelpunkt in die andere Richtung verschoben wird, und a bleibt innerhalb der Kugel liegen. Eine wichtige Bedingung ist hier, dass a und d in den verschiedenen Teilräumen liegen, was durch die Einschränkung der Normalenvektorn gegeben ist, und dass der Kugelmittelpunkt nur innerhalb der gegebenen Umkugel verschoben wird. ■

Aufgrund der Einschränkung der Normalenvektorn und der Kreisdefinition für gekrümmte Flächen ist der gesuchte Kreismittelpunkt nun eindeutig definiert. Für die Triangulierung ist es notwendig, dass die anfangs konstruierte Triangulierung konsistent ist, was eine feine Anfangstriangulierung in stark gekrümmten Gebieten erfordert. Die CDT für gekrümmte Flächen ist ebenfalls durch die Kreisdefinition definiert. (Innerhalb eines Kreises, auf dem drei Knoten liegen, existiert kein weiterer Knoten.) Die Konsistenzeigenschaft der Triangulierung stellt sicher, dass unabhängig von der Wahl des Umkreises der Edgeflippingalgorithmus die gleiche Triangulierung erzeugt.

Algorithmus B

(1) $Build(V, E)$ für gekrümmte Flächen.

(2) Falls alle Dreiecke well-shaped und well-sized sind, dann halte.

(3) Bestimme $C_r(U)$ und ein Element $u \in U$ mit $C_r(u) = C_r(U)$.

(4) $Travel(p_u, C_r(u))$.

(5) Es wurde ein Dreieck ermittelt, das $C_m(u)$ enthält:

$Insert(C_m(u))$;

goto 2;

(6) Es wurde eine blockierende Kante e ermittelt:

Sei l die Länge der Kante e.

$Split(e, 3)$, falls $2 * \sqrt{3} * h \leq l \leq 4 * h$;

$Split(e, 2)$ sonst.

Sei l_{neu} die Länge der erzeugten Kanten und M die Menge der C-Knoten mit einem sichtbaren Abstand zu einem neuen S-Knoten, der kleiner als l_{neu} ist.

$Delete(k)\ \forall\ k \in M$.

Goto 2 .

In der Praxis ergeben sich bei Anwendung des Algorithmus hauptsächlich zwei Probleme. Das erste ist die Konstruktion eines geeigneten Eingangsnetzes beim Aufruf von $Build$ und das zweite ist die Bestimmung der Kreismittelpunkte. Hier kommen oft iterative Techniken (z.B. Newton's Methode) zum Einsatz, die dann einen großen Teil der Berechnungszeit bilden.

4.6 Eine 3 D Advancing-Front/Delaunay Triangulierung

Dieser Algorithmus von P. Frey, H. Borouchaki und P.-L. George beschreibt ein Verfahren zur Erzeugung von Tetraeder Netzen und wurde in dem Paper "Delaunay Tetrahedralization using an Advancing-Front Approach" vorgestellt [28]. Ausgehend von einer Anfangs konstruierten verallgemeinerten Delaunay Triangulierung (CDT) werden nach einem Advancing-Front Ansatz Steinerpunkte gesucht und nach der Delaunay Methode eingefügt, so daß ein gegebenes Größenkriterium für die Elemente erfüllt wird.

Im folgenden sei ein beliebiges zu triangulierendes dreidimensionales Gebiet $\Omega \subset \mathbf{R}^3$ gegeben. F seien die Flächen die den Rand von Ω beschreiben, V die Knoten dieser Flächen und B sei ein Quader der Ω enthält. Mit $T(\Omega), T(V,F), T(B), T(B,V)$ und $T(B,V,F)$ wird die Triangulierung (CDT) der angegebenen Gebiete bezeichnet. Somit läßt sich der Algorithmus wie folgt beschreiben:

Algorithmus

(1) Konstruiere die CDT $T(V,F)$
   (a) Berechne eine Boundingbox B von Ω
   (b) Konstruiere die CDT $T(B)$
   (c) Konstruiere die CDT $T(B,V)$ (Einfügen der Knoten V.)
   (d) Konstruiere die CDT $T(B,V,F)$ (Anpassung falls F geschnitten wird.)
   (e) Bestimme das Teilnetz $T(V,F)$

(2) Konstruiere die CDT $T(\Omega)$
   (a) Initialisiere $T(\Omega)$ mit $T(V,F)$
   (b) Initialisiere die Front $\mathbf{F}$ mit F
   (c) Solange $\mathbf{F}$ nicht leer ist führe die nächsten Schritte aus
   (d) Berechne die Menge S der optimalen Steinerpunkte bzgl. $\mathbf{F}$
   (e) Entferne alle Punkte aus S die zu nahe an einem Knotenpunkt aus Ω liegen
   (f) Füge S in die CDT $T(\Omega)$ ein
   (g) Identifiziere die neue Front $\mathbf{F}$

(3) Optimiere $T(\Omega)$

Auf die einzelnen Punkte wird im folgenden wieder etwas genauer betrachtet.

Die initiale Triangulierung

Durch die Berechnung einer Boundingbox B, bestehend aus 8 Knoten und 6 Flächen die rechtwinklig aufeinander stehen, die das gegebene Gebiet enthält ist zunächst ein konvexes zu Triangulierendes Gebiet gegeben welches in 5 Tetraeder zerlegt wird (siehe Abb. 4.48).

Damit ist $T(B)$ gegeben und die Knotenpunkte V können unter Verwendung des Delaunaykerns (siehe Der Delaunaykern) eingefügt werden womit das Netz $T(B,V)$ berechnet ist. Das Netz $T(B,V)$ muß nun noch die Flächen F berücksichtigen. Falls der Benutzer

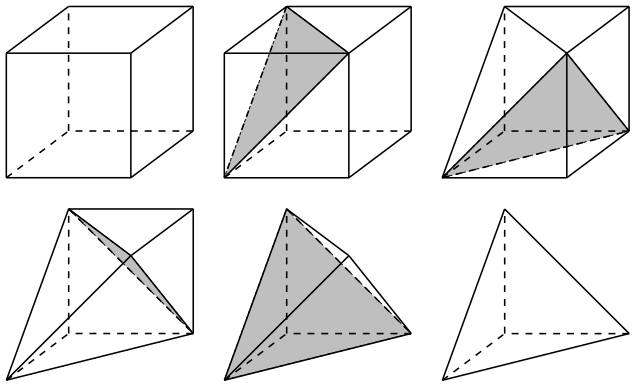


Abbildung 4.48: Zerlegung eines Quaders in 5 Tetraeder

Steinerpunkte auf gegebenen Flächen erlaubt werden geschnittene Flächen trianguliert, so daß diese Triangulierung in dem neu zu berechnenden Netz $T(B, V, F)$ enthalten ist. Sind keine Steinerpunkte auf den Flächen zulässig wir über eine Heuristik ein einzufügender Steinerpunkt bestimmt um die Fläche in die CDT zu integrieren. Diese sehr aufwendige Bestimmung der Steinerpunkte sollte jedoch vermieden werden, indem eine Unterteilung der Flächen zugelassen wird. Nach der Berechnung der Triangulierung $T(B, V, F)$ gilt es das Netz $T(V, F)$ zu bestimmen. Ausgehend vom Rand der Boundingbox und den adjazenten Tetraedern kann leicht entschieden werden ob ein Tetraeder innerhalb des Gebietes Ω liegt oder nicht.

Die Berechnung der Steinerpunkte

Um die Menge der optimalen Steinerpunkte zu berechnen, die ideale Tetraeder bzgl. des vorgegebenen Raumes erzeugen ist es notwendig den Begriff des idealen Tetraeders zu definieren.

Der vorgegebene Kontrollraum KR in dem die zu konstruierenden Tetraeder liegen ist zunächst durch Ω gegeben und liegt durch die Triangulierung $T(V, F)$ in diskretisierter Form vor. Der Kontrollraum wird mit einer kontinuierlichen Feldgröße ausgestattet die ebenfalls durch $T(V, F)$ diskretisiert wird. Um die diskreten Feldgrößen an den Knotenpunkten V bestimmen zu können wird die assozierte Feldgröße $h(v)$ an einem Kontenpunkt v wie folgt definiert:

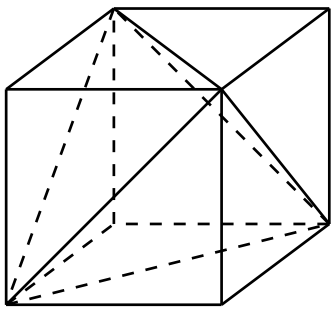


Abbildung 4.49: Ein Würfel als Gebiet Ω

Definition 4.6.1 Diskrete Feldgröße

Sei $v \in V$ beliebig, M die Menge der Kanten mit Endpunkt v, $a \in M$ mit $a = (u, v)$ und u, v, w seien Eckpunkte eines Dreiecks aus der Triangulierung, damit liegt der Punkt w gegenüber der Kante a. Weiter sei $d_{min} := minimum\{dist(u, v)|a \in M\}$ und $d_{max} := maximum\{dist(u, v)|a \in M\}$.

(1) v ist ein Knoten einer Fläche aus F

$$h(v) := \frac{d_{min}(v) + d_{max}(v)}{2}$$

(2) v ist ein Steinerpunkt

$$h(v) := \frac{\sum_{a \in M} \frac{1}{dist_{(u,v)}} \cdot h(w)}{\sum_{a} \frac{1}{dist_{(u,v)}}}$$

Damit ist die diskrete Feldgröße für alle Knotenpunkte wohl definiert. Für jeden beliebigen Punkt $q \in \Omega$ läßt sich die Definition wie folgt erweitern:

Definition 4.6.2 Feldgröße

Sei $q \in \Omega$ gegeben und sei K der Tetraeder der q enthält und die Knoten (v_1, v_2, v_3, v_4) besitzt. Weiter seien $(\alpha_i)_{1 \leq i \leq 4)}$ die baryzentrischen Koordinaten von q in K. Durch eine arithmethische Interpolation ergibt sich die assozierte Feldgröße:

$$h(q) := \sum_{i=1}^{4} \alpha_i \cdot h(v_i)$$

Definition 4.6.3 normalisierte Kantenlänge $l(u, v)$

Sei (u, v) eine gegebene Kante. Unter der Annahme das die Kante keine Tetraeder schneidet wird die normalisierte Kantenlänge definiert als:

$$l(u, v) := dist(u, v) \cdot \int_0^1 \frac{1}{h(t)} dt$$

Für eine arithmetische Interpolation ergibt sich

$$h(t) = h(u) + [h(v) - h(u)] \cdot t$$

Folglich gilt:

$$l(u, v) = \frac{dist(u, v)}{h(v) - h(u)} \cdot \ln(\frac{h(v)}{h(u)})$$

und für eine Kante (u, v) die m verschiedene Schnittpunkte $u = s_1, .., s_m = v$ mit Tetraedern besitzt ergibt sich damit:

$$l(u, v) = \sum_{i=1}^{m-1} l(s_i, s_{i+1})$$

Definition 4.6.4 optimaler Tetraeder

Sei $f = (u, v, w)$ die Frontfläche für die ein Tetraeder gebildet werden soll und sei p der optimale Steinerpunkt. Der durch die Punkte u, v, w, p definierte Tetraeder heißt optimal genau dann wenn

$$\forall q \in \{u, v, w\} : l(p, q) = 1$$

gilt.

Es wird nun ein Algorithmus zur Bestimmung von p angegeben.

Algorithmus

(1) Berechne einen Punkt q, so daß q auf der Geraden liegt die senkrecht zu f und durch das Baryzentrum verläuft.

(2) Berechne $h(q)$

(3) Berechne die Punkte u', v', w' mit $l(u, u') = l(v, v') = l(w, w') = 1$ und u', v', w' liegen auf den entsprechenden Geraden $(u, q), (v, q), (w, q)$

(4) Berechne das gewichtete Baryzentrum q' von u', v', w'

(5) $q := q + \alpha \cdot \vec{qq'}$ mit dem Relaxionskoeffizient α

(6) goto 2

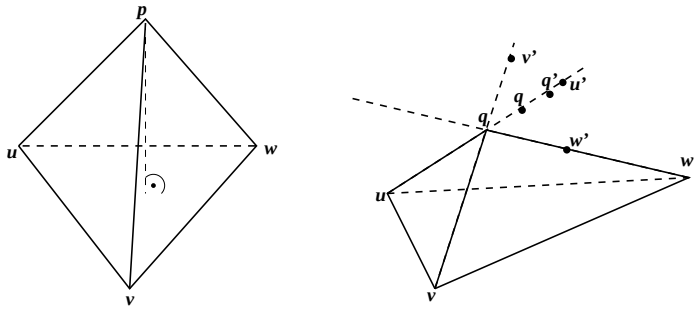


Abbildung 4.50: Berechnung der Steinerpunkte

In der Praxis wird mit $\alpha = 0.1$ und 5 Iterationsschritten eine sehr gute Näherung für einen pseudo-optimalen Punkt berechnet. Auf diese Art wird für jede Fläche ein (pseudo) optimaler Steinerpunkt berechnet und so die Menge S der einzufügenden Steinerpunkte gebildet.

Löschen von Steinerpunkten aus der Menge S

Da die Steinerpunkte alle unabhängig voneinander berechnet werden ist noch zu prüfen ob diese Punkte zu dicht beieinander liegen. Ein Punkt p aus der Menge S wird gelöscht falls es einen Knotenpunkt $q \in V \cup S, q \neq p$ gibt mit $l(p,q) < \frac{1}{\sqrt{2}}$.

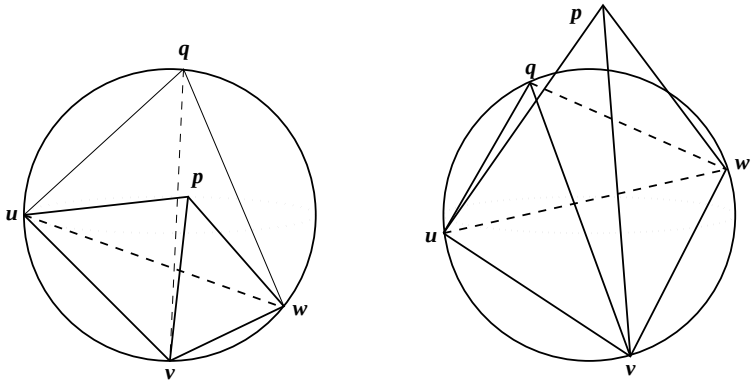


Abbildung 4.51: Löschen von Punkten. Bild rechts: p wird gelöscht

Der Delaunaykern: Einfügen eines Steinerpunktes

Definition 4.6.5 3 D Delaunay Triangulierung einer Punktmenge

Sei $V = \{p_i | p_i \in \mathbb{R}^3, i \in \mathbb{N}_{>3}\}$ gegeben. $T(V)$ sei eine Triangulierung von V im $\mathbb{R}^3$, K ein Tetraeder der Triangulierung und $C(K)$ die kleinste umschreibende Kugel von K.

$T(V) = DT(V)$ heißt Delaunay Triangulierung falls für alle K der Triangulierung gilt: $C(K)$ enthält keinen Knoten $v \in V$.

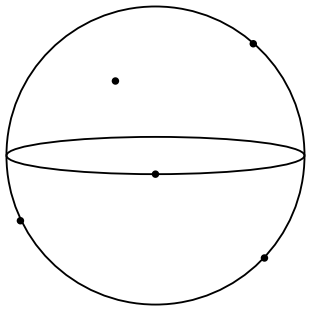


Abbildung 4.52: Die kleinste umschreibende Kugel enthält einen Knoten

Definition 4.6.6 Sternförmig

Sei G ein beliebiges Gebiet und p ein Punkt von G. G heißt sternförmig bzgl. p falls

$$\forall q \in G : (p,q) \cap G = (p,q)$$

gilt, das heißt p und q sind sichtbar.

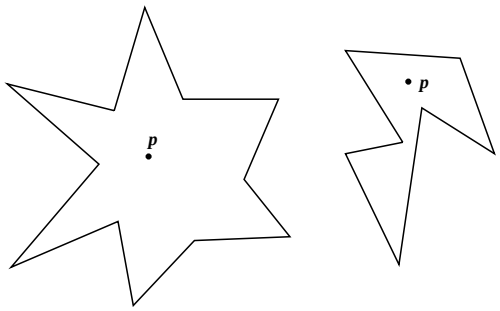


Abbildung 4.53: sternförmiges (links) und nicht sternförmiges Polygon (rechts)

G heißt sternförmig, falls ein Punkt p existiert für den gilt: G ist sternförmig bzgl. p.

Gegeben sei $DT(V)$ und ein Punkt $x \in \mathbb{R}^3, x \notin V$. G sei die Mege der Tetraeder K mit $x \in C(K)$ und V' seien die Knoten aus V in G inklusive des neuen Knotens x.

Offensichtlich gilt für jede Delaunay Triangulierung

(1) G ist sternförmig bzgl. x

(2) $DT(V \cup x) = (DT(V) - G) \cup DT(V')$

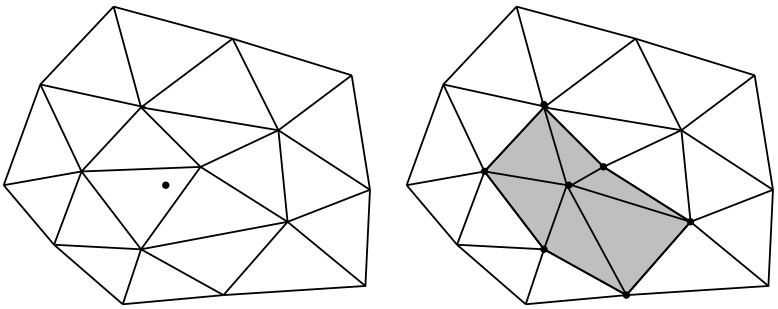


Abbildung 4.54: $DT(V \cup x) = (DT(V) - G) \cup DT(V')$

Diese Eigenschaft wird für die Integration eines Steinerpunktes p in die gegebene Triangulierung verwendet. Leider können die Definitionen nicht direkt übernommen werden und bei der Anpassung bleibt auch die Delaunay Bedingung nicht erhalten, was sich allerdings nicht vermeiden läßt.

Sei G ein sternförmiges Gebiet bzgl. des einzufügenden Steinerpunktes, wobei G aus Tetraedern der gegebenen Triangulierung gebildet wird. Entferne G aus der Triangulierung und füge die neue Triangulierung des Gebietes G, in die der Steinerpunkt integriert ist ein.

G wird dabei wie folgt erzeugt:

(1) $G_1 := \{K \in T(\Omega) | p \in K\}$

(2) $G_2 := \{K \in T(\Omega) | \exists K' \in G_1 : K \cap K' = \emptyset, p \in C(K)\}$

(3) $G_2 := G_2 - \{K \in G_2 | \exists v \in V : v \in K, G \cap v \neq \emptyset, \delta G \cap v = \emptyset\}$

(4) Solange G nicht sternförmig bzgl. p ist entferne die entsprechenden Tetraeder aus G. Diese Tetraeder liegen in der Mege G_2 da G_1 sternförmig bzgl. p ist.

Damit ist G zwar sternförmig aber es werden nicht alle Tetraeder berücksichtigt deren umschreibende Kugel den eingefügten Steinerpunkt enthält. In diesem Punkt weicht die konstruierte Triangulierung von der Delaunay Definition ab.

Die Identifikation der Front F

Die Front $\mathbf{F_i}$ trennt zwei verschiedene Arten von Tetraedern der Triangulierung, die Menge der gültigen (A_i) und der ungültigen (U_i) Tetraeder. Der Index i gibt dabei an welche Front gerade bestimmt wird. Die Front $\mathbf{F_0}$ wird von den gegebenen Flächen F des Gebietes Ω gebildet und die Menge A_0 wird als die Menge der Tetraeder definiert, die außerhalb von Ω liegen und an $\mathbf{F_0}$ grenzen. Die Menge U_0 wird entsprechend von den Tetraedern innerhlab von Ω gebildet die an $\mathbf{F_0}$ grenzen. Die Menge A_0 gehört dabei nicht zu der gesuchte Triangulierung $T(\Omega)$.

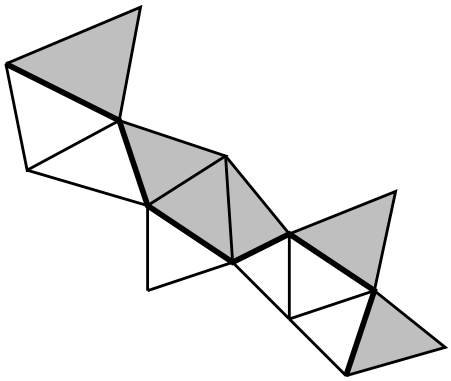


Abbildung 4.55: Teil einer initialen Front mit A_0 grau und U_0 weiß

Definition 4.6.7 Gültiger Tetraeder

Der Tetraeder K wird als gültiger Tetraeder definiert genau dann wenn

$$\forall a \in K : l(a) < \sqrt{2}$$

gilt, wobei a eine Kante von K ist und $l(a)$ die normalisierte Kantenlänge.

Definition 4.6.8 Gültiger Tetraeder bzgl. $\mathbf{F_{i+1}}$

Der Tetraeder K wird als gültiger Tetraeder bzgl. $\mathbf{F_{i+1}}$ definiert falls K ein gültiger Tetraeder ist oder $K \in U_i$ gilt.

Definition 4.6.9 Ungültiger Tetraeder

Ein Tetraeder K wird als ungültiger Tetraeder bezeichnet falls es sich bei K nicht um einen gültigen Tetraeder handelt.

Es wird nun ein Algorithmus zur Bestimmung der Mengen A_{i+1}, U_{i+1} der Front $\mathbf{F_{i+1}}$ und des Kontrollraumes KR_{i+1} angegeben, wobei die initialen Mengen wie oben beschrieben gegeben sind.

Algorithmus

(1) $A_{i+1} := A_i$

(2) do

(3) Überprüfe alle Tetraeder K des Kontrollraumes auf ihre Gültigkeit bzgl. $\mathbf{F_{i+1}}$ und auf ihre Nachbarschaft zu einen Tetraeder aus A_{i+1}. Füge K zu der Menge A_{i+1} hinzu falls K gültig ist und an A_{i+1} grenzt.

(4) solange sich A_{i+1} ändert.

(5) $A_{i+1} := A_{i+1} - A_i$

(6) $KR_{i+1} := KR_i$ ohne das durch A_{i+1} beschriebene Gebiet.

(7) U_{i+1} := Die Menge der Tetraeder aus KR_{i+1} die an einen Tetraeder aus A_{i+1} grenzen.

(8) $\mathbf{F_{i+1}}$:= Die Menge der Flächen die sowohl zu einem Tetraeder aus A_{i+1} als auch aus U_{i+1} gehören.

Die Netzoptimierung

Da während der Konstruktion des Netzes immer wieder ungültige Tetraeder bzgl. der Front als gültig definiert werden ist eine Netzoptimierung notwendig, um die Qualität des Netzes die gerade von diesen Elementen bestimmt wird zu verbessern.

Definition 4.6.10 Qualität einer Kante

Sei a eine Kante eines Tetraeders. Die Qualität einer Kante wird definiert als

$$Q_e(a) := \begin{cases} l(a) & \text{falls } l(a) \geq 1 \\ \frac{1}{l(a)} & \text{sonst} \end{cases}$$

Definition 4.6.11 Qualität eines Tetraeders

Sei K ein Tetrader der Triangulierung. Die Qualität von K wird definiert als

$$Q_t(K) := \alpha \cdot \frac{h_{max}}{r}$$

mit h_{max} als Länge der längsten Kante von K und r als Radius der größten Innenkugel von K. $\alpha = \frac{6}{\sqrt{12}}$ ist dabei ein Normalisierungsfaktor um die Qualität eines gleichseitigen Tetraeders auf den Wert 1 zu setzen.

Diese Qualitätsdefinitionen liefern Werte zwischen 1, für den optimalen Tetraeder und ∞ für den schlechtesten Tetraeder. Ziel ist es nun Q zu minimieren.

(1) Netzkorrektur durch löschen von Kanten.

Sei $(v, w) = a$ eine Kante der Triangulierung, l_{min} die untere Schranke der normalisierten Kantenlänge und l_{max} die obere Schranke.

Falls $l(v, w) < l_{min}$ verschiebe v nach w. Damit wird die Kante a und alle Tetraeder mit dieser Kante gelöscht.

Falls $l(v, w) > l_{max}$ wird das Gebiet $G := \{K | a \subset K\}$ neu Trianguliert.

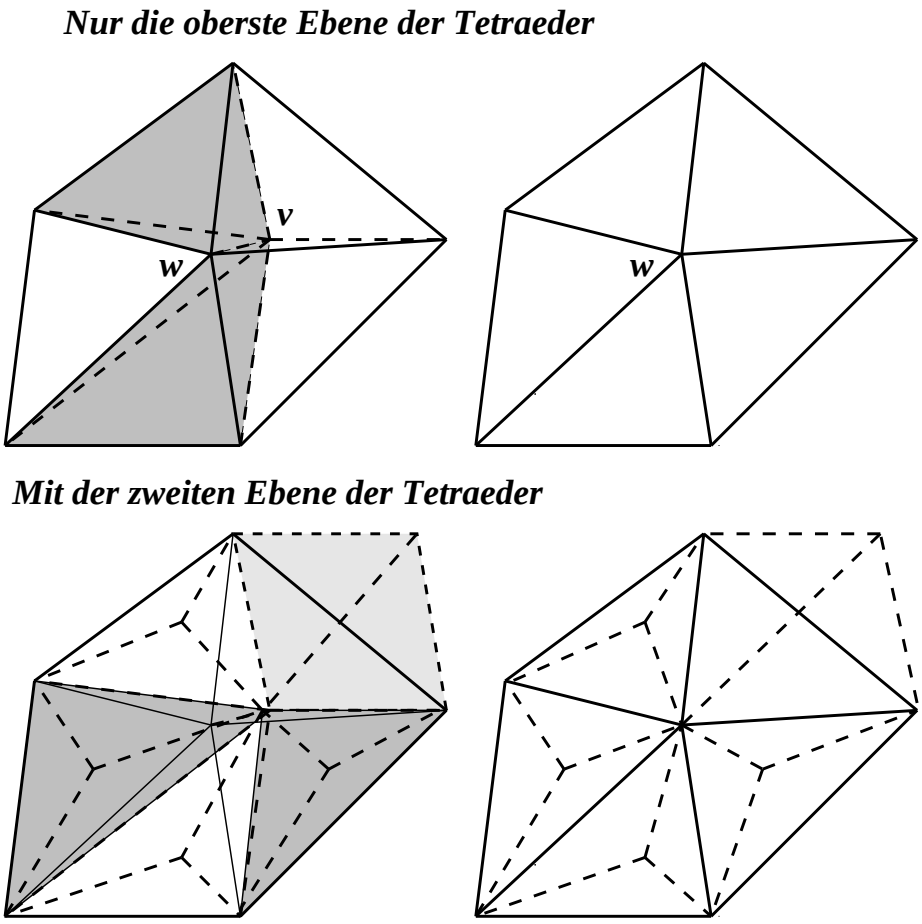


Abbildung 4.56: Netzkorrektur durch entfernen einer Kante

(2) Netzkorrektur durch Verschiebung von Knoten.

Sei K ein Tetraeder schlechter Qualität und p ein Knoten von K.
Weiter sei $G := \{K | p \subset K\}$.

Berechne einen optimalen Punkt q und verschiebe p nach

$$q' := p + \omega \cdot \vec{pq}$$

falls $Q_t(K') < Q_t(K) \ \forall K' \in G$ gilt.

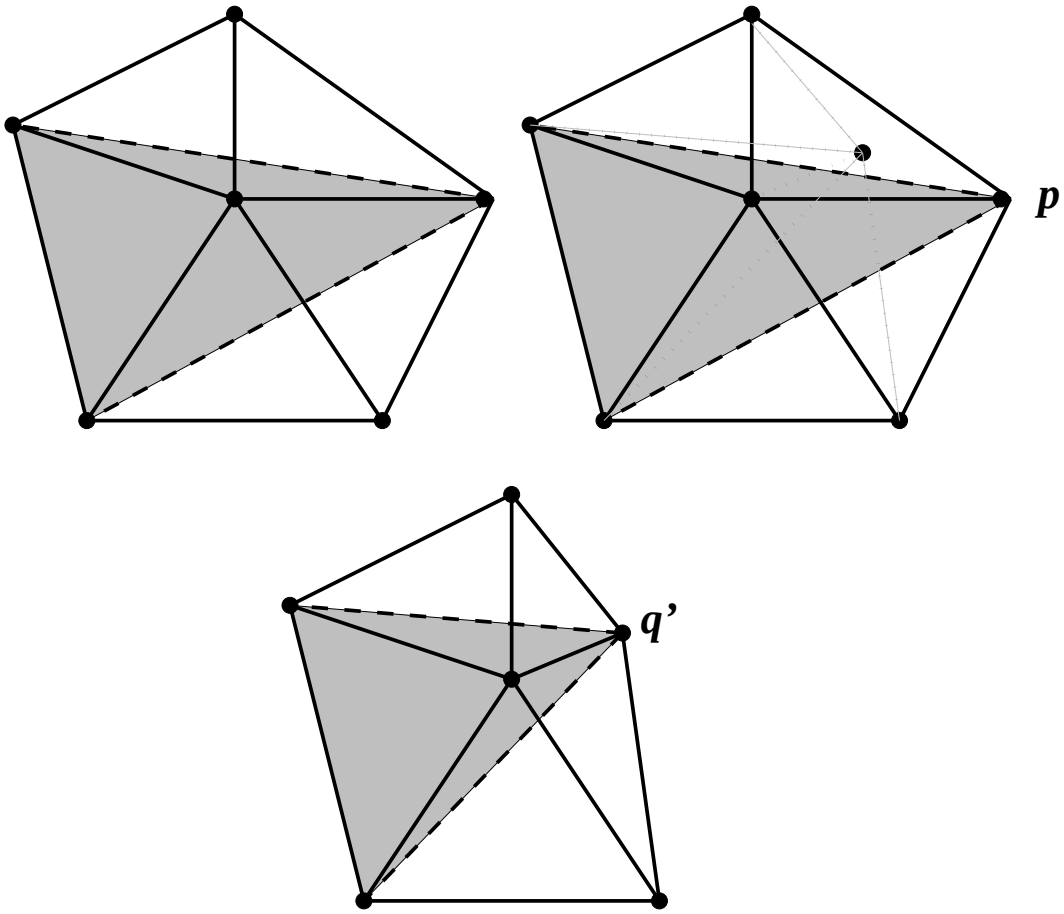


Abbildung 4.57: Netzkorrektur durch Verschiebung eines Knotens

Durch diese Netzoptimierung wird im letzten Schritt des Algorithmus die Qualität der erzeugten Triangulierung verbessert.

4.7 Eine Octree basierte 3 D Triangulierung

Der erste Algorithmus zur Triangulierung von dreidimensionalen Objekten, welcher sowohl eine Ausage über die Qualität der erzeugten Tetraeder als auch über die Größe des erzeugten Netzes macht, stammt von S. Mitchell und S. Vavasis [53]. Der in dem Paper "Quality Mesh Generation in Three Dimensions" vorgestellte Algorithmus ist eine Erweiterung des zweidimensionalen quadtree basierten Verfahrens von M. Bern, D. Eppstein und J. Gilbert [9]. Der hier vorgestellte octree basierte Algorithmus garantiert einen, bis auf eine Konstante optimalen Aspect Ratio und wiederum bis auf eine Konstante optimale Netzgröße mit einer Laufzeit von $O(\gamma \cdot n \cdot \log(n))$.

Im folgenden sei ein (fast) beliebiges Polytop P gegeben welches die Bedingungen

(1) P ist zusammenhängend

(2) P schneidet sich nicht selbst

(3) Eine Fläche trennt eindeutig das Innere des Polytops vom Äußeren

(4) Jede Kante des Polytops gehört zu genau zwei Flächen

(5) Zu jedem Punkt p des Polytops existiert eine Epsilonumgebung ϵ, so daß die Umgebung $\epsilon - p$ zusammenhängend ist

erfüllt.

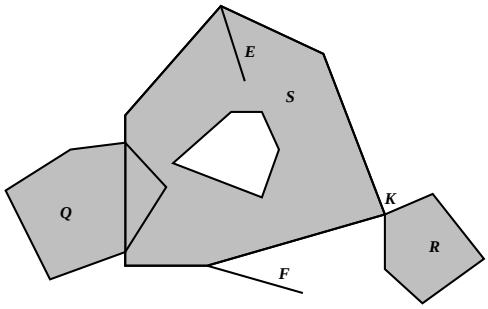


Abbildung 4.58: Ein 2D Beispiel für nicht erlaubte Polytope

Das Polygon S ist durch das gegebene Loch nicht zusammenhängend, die Polygone S und Q schneiden sich, auf beiden Seiten der Kante E liegt das Innere des Polygons, auf beiden Seiten der Kante F liegt das Äußere des Polygons und die Polygone P und R besitzen nur einen gemeinsamen Punkt K womit die Epsilonumgebung ohne K nicht mehr zusammenhängend ist.

Definition 4.7.1 Box

Eine Box B beschreibt das Volumen eines Quaders.

(1) $h_x(B), h_y(B)$ und $h_z(B)$ gibt die Seitenlänge des Quaders in x, y oder z-Richtung an.

(2) $h(B)$ gibt die Seitenlänge des Quaders in dem Fall $h_x(B) = h_y(B) = h_z(B)$ an.

Definition 4.7.2 Erweiterte Box (siehe Abb. 4.59)

Sei B eine Box.

Die erweiterte Box $ex(B)$ besitzt die fünffache Größe und liegt konzentrisch um B.

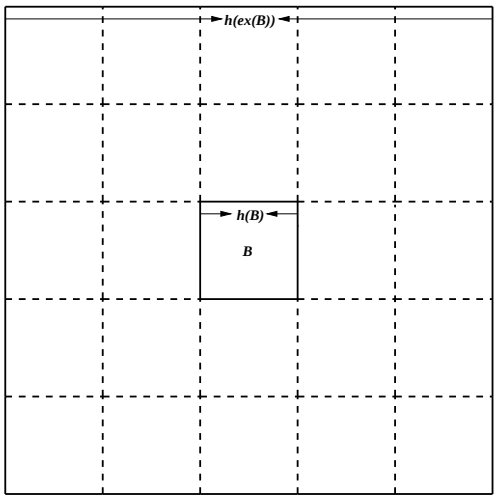


Abbildung 4.59: B und $ex(B)$

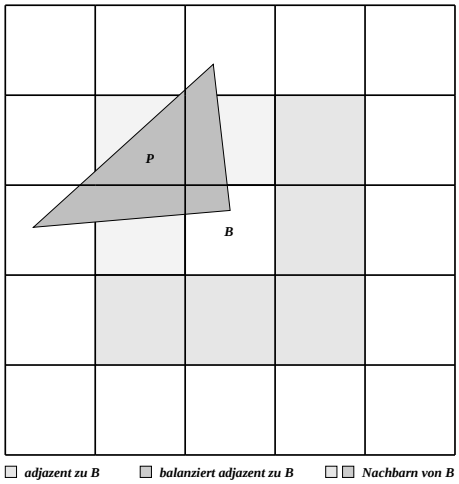


Abbildung 4.60: Nachbarn, adjazent und balanziert adjazent.

Definition 4.7.3 Nachbar (siehe Abb. 4.60)

Seien B_1, B_2 zwei Boxen. B_1 und B_2 sind benachbart falls gilt:

$1: \quad B_1 \not\subset B_2 \quad 2: \quad B_2 \not\subset B_1 \quad 3: \quad B_1 \cap B_2 \neq \emptyset$

Definition 4.7.4 Adjazent (siehe Abb. 4.60)

Seien B_1, B_2 zwei Boxen. B_1 und B_2 sind adjazent falls gilt:

(1) B_1 und B_2 sind benachbart

(2) $P \cap B_1 \cap B_2 \neq \emptyset$

Definition 4.7.5 Balanziert Adjazent (siehe Abb. 4.60)
Seien B_1, B_2 zwei Boxen. B_1 und B_2 sind balanziert adjazent falls gilt:

(1) B_1 und B_2 sind benachbart

(2) $(P \cap ex(B_1) \cap B_2 \neq \emptyset) \vee (P \cap ex(B_2) \cap B_1 \neq \emptyset)$

Definition 4.7.6 Knotenkegel (vertex cone)

Sei v ein Knoten und seien $F_1, F_2, ..., F_l$ alle zu v inzidenten Objekte (Kanten oder Flächen) des Polyeders P.

Die Objektmenge $VC(v) = \{v, F_1, F_2, ..., F_l\}$ heißt Knotenkegel und v wird als Spitze des Knotenkegels bezeichnet (siehe Abb. 4.61).

Definition 4.7.7 Knotenkegel einer Box (vertex cone of a box)

Sei $VC(v)$ ein Knotenkegel. $VC(v) \cap B$ heißt Knotenkegel der Box B falls gilt:

(1) Der Knoten v liegt innerhalb der Box B

(2) Die erweiterte Box $ex(B)$ enthält ausschließlich Objekte die auch zum Knotenkegel gehören

Definition 4.7.8 Knotenüberfüllt (vertex crowded)

Eine Box B heißt knotenüberfüllt falls gilt:

(1) B enthält einen Knoten v

(2) $VC(v)$ ist kein Knotenkegel der Box B

Bezeichnung 4.7.9 Knotenbox
B wird als Knotenbox bezeichnet, falls es einen Knotenkegel der Box B gibt.

Definition 4.7.10 Kantenkegel (edge cone)

Sei e eine Kante und seien F_1, F_2 die zu e inzidenten Objekte (Flächen) des Polytops P.

Die Objektmenge $EC(e) = \{e, F_1, F_2\}$ heißt Kantenkegel und e wird als Spitze des Kantenkegels bezeichnet (siehe Abb. 4.61).

Definition 4.7.11 Kantenkegel einer Box (edge cone of a box)

Sei $EC(e)$ ein Kantenkegel. $EC(e) \cap B$ heißt Kantenkegel der Box B falls gilt:

(1) B enthält eine Kante e

(2) Die erweiterte Box $ex(b)$ enthält ausschließlich Objekte die auch zum dem Kantenkegel $EC(e)$ gehören

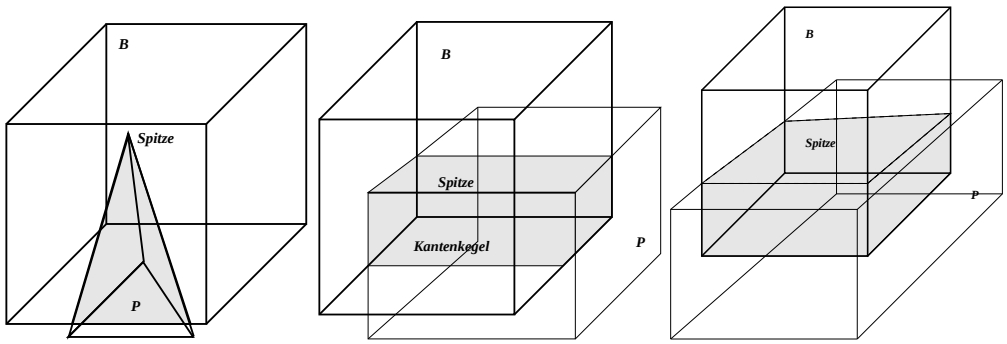


Abbildung 4.61: Ein Knotenkegel,ein Kantenkegel und ein Flächenkegel

Definition 4.7.12 Kantenüberfüllt (edge crowded)

Eine Box B heißt kantenüberfüllt falls gilt:

(1) B enthält eine Kante e

(2) $EC(e) \cap B$ ist kein Kantenkegel der Box B

Bezeichnung 4.7.13 Kantenbox

B wird als Kantenbox bezeichnet, falls es einen Kantenkegel der Box B gibt.

Definition 4.7.14 Flächenkegel einer Box (facet cone of a box)

Sei f eine Fläche des Polyeders. $f \cap B$ heißt Flächenkegel der Box B falls gilt:

(1) $f \cap b \neq \emptyset$

(2) Die erweiterte Box $ex(b)$ enthält kein weiteres Objekt außer der Fläche f

Aus Konsistezgründen wird f als die Spitze des Flächenkegels bezeichnet (siehe Abb. 4.61).

Definition 4.7.15 Flächenüberfüllt (facet crowded)

Eine Box B heißt flächenüberfüllt falls gilt:

(1) B enthält eine Fläche f

(2) $f \cap B$ ist kein Flächenkegel der Box B

Bezeichnung 4.7.16 Flächenbox

B wird als Flächenbox bezeichnet, falls es einen Flächenkegel der Box B gibt.

Der generierte Octree soll eine möglichst einfach zu triangulierende Vorzerlegung des Gebietes erzeugen, wobei die Triangulierung einen möglichst kleinen Aspect Ratio aufweisen sollte. Aus diesem Grund dürfen sich benachbarte Boxen in ihrer Größe nicht zu stark unterscheiden, Knoten müssen zentriert innerhalb einer Box liegen und in einer Box sollte auch nur ein zusammenhängendes Teilstück des Polytops liegen. Damit ergibt sich ein Balanzierungs-, ein Zentrierungs- und ein Duplizierungskriterium, welches während der Octreegenerierung zu berücksichtigen ist.

Bedingung 4.7.17 Balanzierungkriterium

Nicht geschützte balanziert adjazente Boxen unterscheiden sich in der Grösse maximal um den Faktor 2.

Bedingung 4.7.18 Zentrierungkriterium

Die Distanz eines Knotens zum Rand einer Box B beträgt mindestens $\frac{h(B)}{2}$

Bedingung 4.7.19 Duplizierungskriterium

Jede Box des Octree enthält genau eine Zusammenhangskomponente des gegebenen Polytops. Bei mehreren Zhk's erhält jede Zhk ein eigenes Duplikat der Box. Auf die duplizierten Boxen wird hier nicht näher eingegangen. Für das Verständnis des Algorithmus ist zu beachten, daß sich duplizierte Boxen gegenseitig nicht beeinflussen, d. h. sie lösen gegenseitig keine weitere Aufteilung der Box aus.

Mit dem gegebenen Polytop P kann der Algorithmus grob folgendermaßen beschrieben werden:

Der Algorithmus

(1) Initialisiere den Octree als eine einzige Box die das Polytop enthält

(2) Knotenphase

  (a) Aufteilung einer Box (I)
  Solange eine knotenüberfüllte Box existiert teile die Box auf, falls nötig dupliziere die Box und erfülle die Balanzierungsbedingung nach jeder Aufteilung.

  (b) Aufteilung einer Box (II)
  Jede zu einer Knotenbox E balanziert adjazente Box B, mit $h(E) < h(B)$ wird Aufgeteilt und nach jeder Aufteilung wird die Balanzierungbedingung überprüft.
  Jede Knotenbox ist nun von 26 gleichgroßen Boxen umgeben.

  (c) Zentrierung der Knoten
  Jede Knotenbox wird mit 7 benachbarten Boxen so zu einer Box der Größe $2 * h(E)$ kombiniert, daß der Knoten einen Mindestabstand von $h(E)/2$ zum Rand der Box besitzt.

  (d) Schützen der Knotenboxen
  Jede Knotenbox wird nach der Zentrierung vor einer weiteren Aufteilung geschützt.

(3) Kantenphase

  (a) Aufteilung einer Box (I)
  Solange eine nicht geschützte, kantenüberfüllte Box existiert teile die Box auf, falls nötig dupliziere die Box und erfülle die Balanzierungsbedingung nach jeder Aufteilung.

  (b) Aufteilung der Box (II)
  Jede zu einer Kantenbox balanziert adjazente Box und die Kantenbox selber werden genau einmal aufgeteilt. Dabei wird die Balanzierungbedingung überprüft.

  (c) Schützen der Kantenboxen
  Jede Kantenbox und die zu ihr balanziert adjazenten Boxen werden vor einer weiteren Aufteilung geschützt.

(4) Flächenphase

  (a) Aufteilung einer Box (I)
  Solange eine nicht geschützte, flächenüberfüllte Box existiert teile die Box auf, falls nötig dupliziere die Box und erfülle die Balanzierungsbedingung nach jeder Aufteilung.

(b) Aufteilung der Box (II)
Jede zu einer Flächenbox balanziert adjazente Box und die Flächenbox selber werden genau einmal aufgeteilt. Dabei wird die Balanzierungbedingung überprüft.

(5) Verschiebe Knoten einer Box falls das Polytop zu dicht am Rand der Box liegt.

(6) Trianguliere die Oberflächen jeder Zellen (2D Triangulierung)

(7) Trianguliere das Volumen jeder Zelle (3D Triangulierung)

(8) Entferne das Gebiet außerhalb des Polytops

Die einzelnen Schritte zur Generierung des Octrees sollten mit der obigen Algorithmenbeschreibung verständlich sein und werden daher nur nochmal anhand eines Beispiels in einer 2D Darstellung verdeutlicht (siehe Abb. 4.62).

Nach der Konstrucktion des Octrees (nach Schritt (4)) ist jede Box entweder leer, liegt vollständig im Polytop, enthält genau einen Knoten, eine Kante, eine Fläche oder zwei Flächen des Polytops. In der Knotenphase des Algorithmus wurde eine Zentrierung vorgenommen, um einen Mindestabstand des Knotens vom Rand der Box zu gewährleisten. Um einen Mindestabstand der Kanten und Flächen zu garantieren werden die Eckknoten der bisherigen Würfel verschoben, falls dies erforderlich sein sollte. Da hier die Form des Würfels verloren geht, kann dieser Schritt erst nach der Octreegenerierung durchgeführt werden.

Verschiebung von Octreeknoten

Definition 4.7.20 Minimale Distanze

(1) Minimale Kantendistanze

Sei k eine Kante des Polytops und e eine Kante der Box die k schneidet.

$$h := \begin{cases} h(e)/2 & \text{falls die Kante } e \text{ geteilt ist} \\ h(e) & \text{sonst} \end{cases}$$

Die Kante e liegt zu nahe an der Kante k, falls $dist(e,k) < h/8$ gilt.

(2) Minimale Flächendistanze

Sei f eine Fläche des Polytops und v ein Knoten der Box die f schneidet und B die kleinste Box mit Knoten v. Der Knoten v liegt zu nahe an der Fläche f falls $dist(v,f) < h(B) \ / \ 16$ gilt.

Eine Kante oder ein Knoten wird wie folgt verschoben, falls eine minimale Distanz unterschritten wird.

(1) Die Kante e liege zu nahe an der Kante k

Verschiebe die Kante e um die Distanze $h/8$, wobei die Verschiebung orthogonal zu e und zu k vorgenommen wird und sich die Kante von k entfernt. Sind die beiden Kanten parallel erfolgt die Verschiebung orthogonal zu e und coplanar zu k.

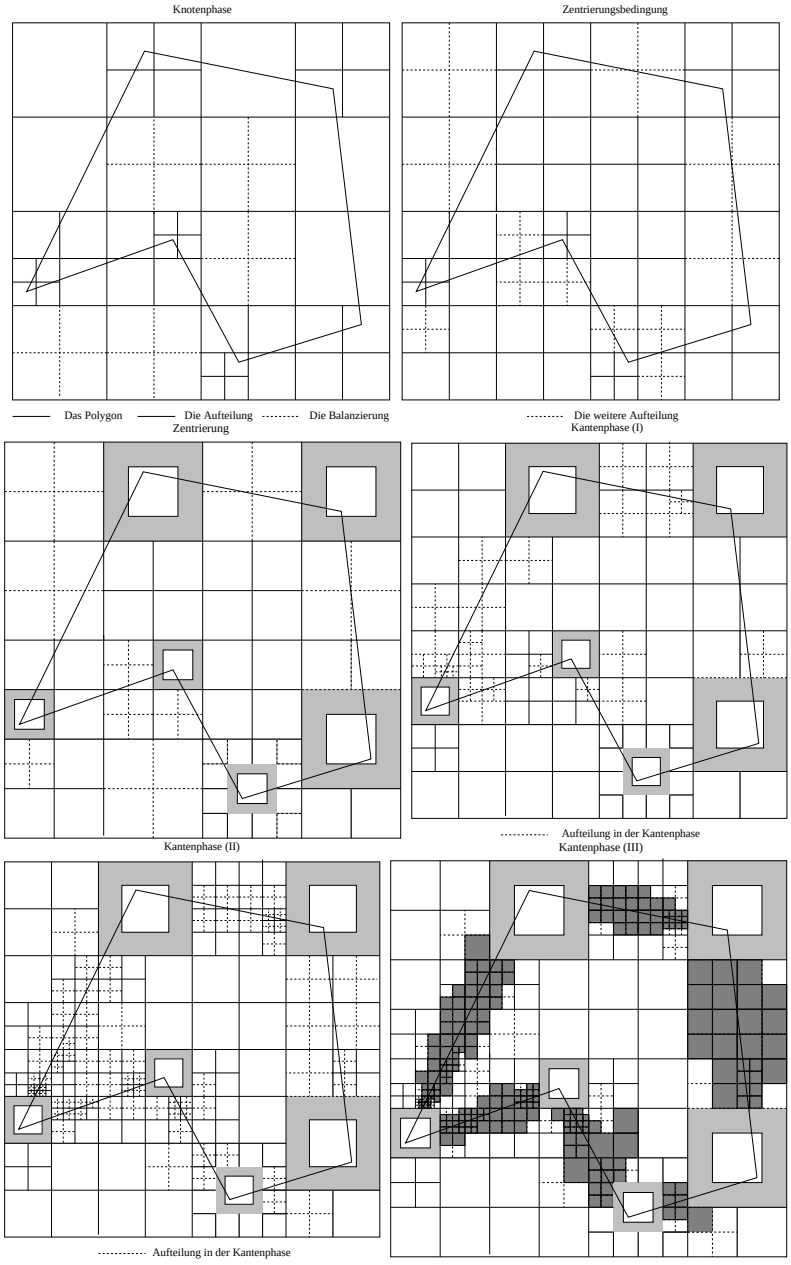


Abbildung 4.62: Eine 2 D Übersicht der Octreegenerierung

(2) Der Knoten v liegt zu nahe an der Fläche f

Verschiebe den Knoten v um die Distanze $h/16$ orthogonal zu f.

Nachdem durch die Verschiebung der Octreeknoten ein Mindestabstand des Polytops zu dem nicht geschnittenen Rand der Boxen garantiert ist wird die Triangulierung des Polytops vorgenommen. Zunächst werden die Randflächen der Boxen trianguliert. Zu beachten ist dabei, daß durch die Verschiebung der Octreeknoten die Knoten einer Randfläche der Box nicht mehr in einer Ebene liegen müssen, daher wird beim Schnitt des Polytops mit einer Randfläche immer die Randfläche vor einer Verschiebung betrachtet. Die 2D Triangulierung der Oberfläche der Boxen bildet anschließend die Grundlage für die endgültige 3D Triangulierung, wobei selbstverständlich nur der Bereich innerhalb des Polytops trianguliert wird.

2D Triangulierung

Sei B eine Box und Q eine Fläche (Quadrat) von B. Sei m_q der Mittelpunkt von Q. Sei m_s der Mittelpunkt der Schnittkante von Q mit dem Polytop, falls die Box von einer Fläche des Polytops geschnitten wird. Wird die Box von zwei Fläche des Polytops geschnitten so sei m_s der Mittelpunkt der näher an m_q liegt bzw. der Schnittpunkt der gemeinsamen Kante mit Q.

Die Schnittpunkte einer Fläche mit dem Rand von Q werden immer als Steinerpunkte eingefügt und sind als Knotenpunkte zu betrachten.

(1) Q liegt außerhalb des Polytops

Hier ist keine Triangulierung erforderlich.

(2) Q liegt innerhalb des Polytops

m_q wird als Steinerpunkt eingefügt und mit allen Knotenpunkten von Q, also auch den Knotenpunkten die auf dem Rand liegen, verbunden.

(3) Q wird von genau einer Fläche des Polytops geschnitten.

(a) $dist(m_s, m_q) \leq \frac{h(B)}{4}$

m_s wird als Steinerpunkt eingefügt und mit allen Knotenpunkten des (eventuell verzerrten) Quadrats Q die innerhalb des Polytops liegen verbunden.

(b) $dist(m_s, m_q) > \frac{h(B)}{4}$

Die Verbindung von m_q mit allen Knotenpunkten des (eventuell verzerrten) Quadrats Q zerlegt die Fläche des Polytops innerhalb von Q in Dreiecke und Vierecke. Die durch diese Drei- und Vierecke definierten Steinerpunkte und Kanten werden eingefügt und die Vierecke werden durch die kürzeste Diagonale trianguliert.

(4) Q wird von zwei Flächen des Polytops geschnitten

(a) Q wird von einer Kante des Polytops geschnitten.

Die Verbindung von m_s mit allen Knotenpunkten des (eventuell verzerrten) Quadrats Q erzeugt die gewünschte Triangulierung.

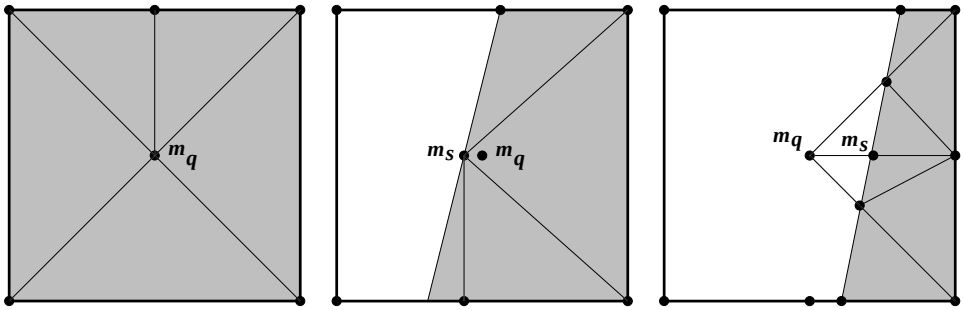


Abbildung 4.63: 2D Triangulierung im Fall 2 und 3

(b) $dist(m_s, m_q) \leq \frac{h(B)}{4}$
Wie 3.(a).

(c) $dist(m_s, m_q) > \frac{h(B)}{4}$
Wie 3.(b).

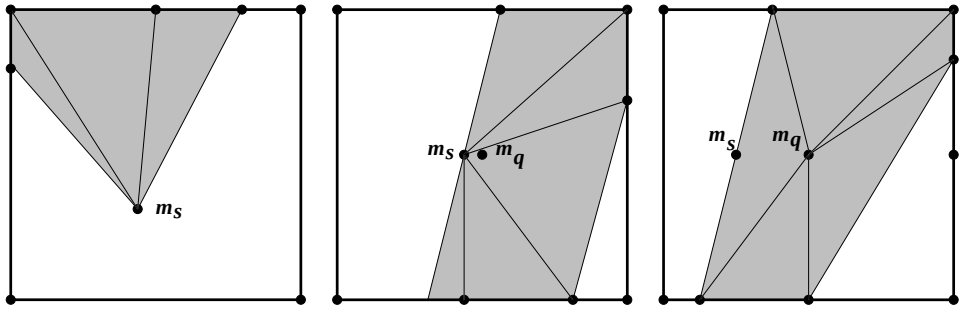


Abbildung 4.64: 2D Triangulierung im Fall 4

3D Triangulierung

Sei B eine Box und die 2D Triangulierung der Oberfläche innerhalb des Polytops sei bekannt.

(1) $B \cap P = \emptyset$

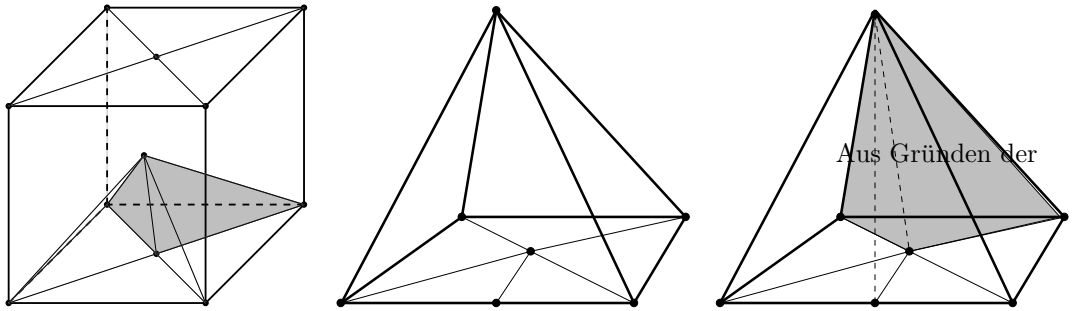
Da die Box außerhalb des Polytops liegt ist eine Triangulierung nicht erforderlich.

(2) $B \cap P = B$

Da die Box vollständig innerhalb des Polytops liegt wird der Mittelpunkt von B als Steinerpunkt eingefügt und mit allen Knotenpunkten der Oberflächentriangulierung verbunden.

(3) $B \cap P$ enthält einen Knotenkegel mit der Spitze v.

Die Spitze v des Knotenkegels bildet hier den vierten Punkt für die Tetraeder und wird mit allen Knotenpunkte der Oberflächentriangulierung verbunden.



Sichtbarkeit wurde nur die Oberflächentriangulierung des oberen und unteren Randes und die Tetraeder mit dem unteren Rand eingezeichnet.

Abbildung 4.65: 3D Triangulierung im Fall 2 und im Fall 3

(4) $B \cap P$ enthält einen Kantenkegel mit der Spitze e

Die Spitze des Kantenkegels ist ein Geradenstück und der Mittelpunkt v dieses Segments wird als Steinerpunkt eingefügt. Eine Verbindung von v mit allen Knotenpunkte der Oberflächentriangulierung erzeugt die gewünscht Triangulierung der Box.

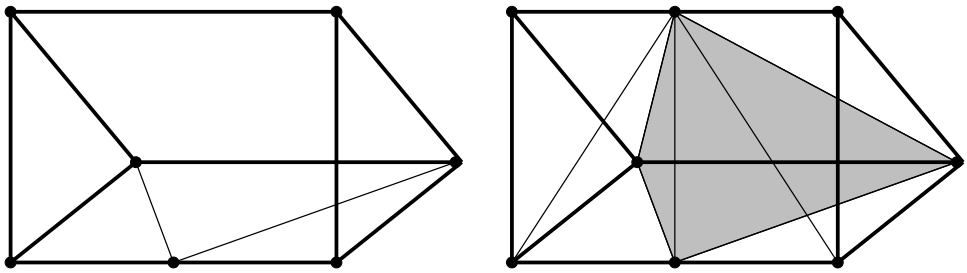


Abbildung 4.66: 3D Triangulierung im Fall 4

(5) $B \cap P$ enthält einen Flächenkegel mit der Spitze f.

Die Spitze des Flächenkegels ist hier die Schnittfläche der Box mit einer Fläche des Polytops. Sei m der Mittelpunkt von B und l der Punkt auf der Fläche f mit minimalem Abstand zu m. Der Punkt v mit

$$v := \begin{cases} l & \text{falls } dist(m,l) \leq \frac{h(B)}{4} \\ m & \text{sonst} \end{cases}$$

dient als Bezugzpunkt für die Triangulierung.

(a) v liegt im inneren des Polytops

Die Fläche f ist noch nicht trianguliert, daher wird zunächst eine Triangulierung von f erzeugt und anschließend wird v mit allen Knotenpunkte der Oberflächentriangulierung verbunden. v ist natürlich als Steinerpunkt einzufügen.

(b) v liegt auf einer Fläche des Poltops

Wiederum wird v als Steinerpunkt eingefügt und mit allen Knotenpunkte der Oberflächentriangulierung verbunden. Damit wird automatisch auch f triangulliert.

(c) v liegt außerhalb des Polytops

In diesem Fall wird v nicht als Steinerpunkt eingefügt. Die Verbindung von v mit allen Knotenpunkten der Oberflächentriangulierung erzeugt jedoch eine Zerlegung des Gebietes in Tetraeder, welche durch f in Tetraeder und Prismen aufgeteilt wird. Die dadurch entstandenen Knoten auf f und die Kanten innerhalb von P werden in die Triangulierung aufgenommen. Jetzt werden die Prismen in Tetraeder zerlegt. Die Prismen bestehen aus einer dreieckigen Grundfläche, die auf der Oberfläche von B liegt und einer dreieckigen Fläche auf dem Flächenkegel f. Dabei können diese beiden Dreiecksflächen keinen, einen oder zwei gemeinsame Knotenpunkte besitzen.

i. Zwei gemeinsame Knotenpunkte
In diesem Fall handelt es sich schon um einen Tetraeder.

ii. Einen gemeinsamen Knotenpunkt p
Durch einfügen einer Dreiecksfläche die durch p, q, r bestimmt ist, wobei q, r Knoten verschiedener Dreiecksflächen sind und $q \neq p \neq r$ gilt, zerfällt das Prisma in zwei Tetraeder.

iii. Kein gemeinsamer Knotenpunkt
Seien p_1, p_2, p_3 die Knotenpunkte der oberen und q_1, q_2, q_3 die Knotenpunkte der unteren Dreiecksfläche, wobei p_i mit q_i durch eine Kante verbunden sind. Durch einfügen der beiden Dreiecksflächen p_1, p_2, q_3 und q_3, q_1, p_2 zerfällt das Prisma in drei Tetraeder.

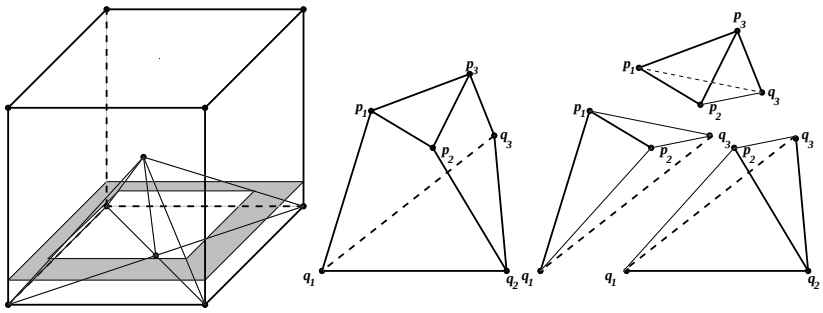


Abbildung 4.67: Zerfall in Prismen und die Triangulierung.

(6) $B \cap P$ enthält zwei "Flächenkegel" mit den Spitzen f, k

Die Spitzen der Flächenkegel sind hier die Schnittflächen der Box mit einer Fläche des Polytops. Sei m der Mittelpunkt von B und l der Punkt auf der Fläche f oder k mit minimalem Abstand zu m. Der Punkt v mit

$$v := \begin{cases} l & \text{falls } dist(m,l) \leq \frac{h(B)}{4} \\ m & \text{sonst} \end{cases}$$

dient als Bezugspunkt für die Triangulierung.

Für den Punkt v gibt es wieder verschiedene Möglichkeiten:

(a) v liegt auf f
In diesem Fall wird zuerst die Fläche k trianguliert.

(b) v liegt auf k
In diesem Fall wird zuerst die Fläche f trianguliert.

(c) v liegt innerhalb von P
In diesem Fall werden beide Flächen f, k trianguliert.

(d) v liegt außerhalb von P
In diesem Fall wird die von v aus unsichtbare Fläche trianguliert.

Die weiter Zerlegung wird wie zuvor unter 5 beschrieben vorgenommen. v wird mit allen Knotenpunkten der Oberflächentriangulierung verbunden, wodurch wiederum eine Zerlegung in Tetraeder und Prismen entsteht die in Tetraeder zerlegt werden.

Optimalität der Triangulierung

Definition 4.7.21 Aspect Ratio

Sei T die Triangulierung von P, t ein beliebiger Tetraeder aus T, $U(t)$ sei die kleinste umschreibende Kugel von t und $I(t)$ sei die größte in t liegende Kugel.

$$AR(t) := \frac{U(t)}{I(t)}$$

$$AR(T) := maximum\{AR(t)|t \in T\}$$

Da bei der Erzeugung der Triangulierung für jeden Knotenpunkt ein Mindestabstand vom Rand des Polytops garantiert wird ist es möglich das maximale Aspect Ratio abzuschätzen. Hier wird nur das Ergebnis dieser Abschätzung angegeben. Für den Beweis sei auf das Paper [53] verwiesen.

Satz 4.7.22 Aspect Ratio der Triangulierung

Sei T die vom Algorithmus berechnete Triangulierung und α der kleinste Winkel innerhalb des Polytops. Es existiert eine von T und α unabhängige Konstante c mit

$$AR(T) \leq \frac{c}{\alpha}.$$

Satz 4.7.23 Optimalität des Aspect Ratio

Sei T die vom Algorithmus berechnete Triangulierung und O die Triangulierung mit minimalem Aspect Ratio. Es existiert eine von der Triangulierung unabhängige Konstante c mit

$$AR(T) \leq c \cdot AR(O).$$

Satz 4.7.24 Optimalität der Größe

Sei T die vom Algorithmus berechnete Triangulierung, $|T|$ die Anzahl der Tetraeder in T, O die Triangulierung mit minimaler Anzahl von Tetraedern wobei $AR(T) \geq AR(O)$ gilt, $|O|$ die Anzahl der Tetraeder in O und α der kleinste Winkel innerhalb des Polytops. Es existiert eine von α und vom Aspect Ratio abhängige Konstante c mit

$$|T| \leq c \cdot |O|.$$

Laufzeit des Algoritmus

Satz 4.7.25 Laufzeit

Sei n die Anzahl der Knoten des Polytops und γ die Anzahl der erzeugten Tetraeder, dann besitzt der Algorithmus eine Laufzeit von

$$O(\gamma \cdot n \cdot \log(n)).$$

Kapitel 5

Adaptive Finite Elemente Methoden

Die meisten der bisher betrachteten Methoden haben homogene Gitter erzeugt, das sind Gitter, bei denen alle Elementeflächen oder -höhen derart nach oben und unten beschränkt sind, dass von etwa gleich großen Elementen gesprochen werden kann. Für viele Anwendungen ist die Erzeugung eines homogenen Gitters (und dessen gleichmäßige Verfeinerung) nicht hinreichend. Um z.B. partielle Differentialgleichungen (PDE) hinreichend genau zu lösen, sollte in Regionen, in denen die Lösung stark variiert, oder gar in der Nähe von Singularitäten das Gitter feinmaschiger sein als an anderen Stellen. Ein Beispiel ist die Berechnung der Wärmeverteilung in einem Gebiet, das aus unterschiedlichen Materialien zusammengesetzt ist, siehe das Beispiel in Abschnitt 2.3.8. Die unterschiedliche Maschenweite soll idealerweise automatisch erzeugt werden, gesteuert über Fehlerschätzer. Ein solches Gitter heißt *adaptiv*.

Die adaptive Netzgenerierung ist also nicht gut von der Anwendung zu trennen. Es sollen deshalb in diesem Kapitel die Lösung elliptischer PDE mit der Finite Elemente Methode (FEM) betrachtet werden. Dabei werden schwache Kenntnisse über PDE und FEM vorausgesetzt.

Die adaptive Lösung einer PDE setzt sich etwa aus folgenden Schritten zusammen:

(1) Erzeugung eines groben Gitters.

(2) FEM-Lösung.

(3) Lokale Fehlerschätzung.

(4) Verfeinerung der Elemente, für die der Fehlerschätzer einen zu großen Wert liefert.

Die Punkte (2)–(4) werden iteriert, bis der Fehlerschätzer in allen Elementen einen hinreichend kleinen Wert liefert.

In diesem Kapitel sollen zunächst eindimensionale Probleme betrachtet werden, d.h. die adaptive Lösung von Randwertaufgaben bei gewöhnlichen Differentialgleichungen. Die Vorgehensweise wird auf höhere Dimensionen übertragen. Dabei kommt es zu einer Reihe von

Schwierigkeiten. Die Darstellung dieser Abschnitte orientiert sich an den entsprenden Abschnitten des Buches *Numerik partieller Differentialgleichungen* von Großmann und Roos, und an den bahnbrechenden Arbeiten von Rheinboldt u.a.; stellvertretend sei genannt: *Adaptive Mesh Refinement Processes for Finite Element Solutions*, Int. J. Num. Meth. Eng. 17 (1981) 649–662.

Danach soll zu einer mehr Praxis-orientierten Sichtweise gewechselt werden. Man könnte sagen, es werden mehr heuristische Fehlerschätzer entwickelt, wie sie in den Programmpaketen PLTMG und KASKADE verwendet werden. Ein zusätzlicher Gesichtspunkt ist dabei die Verwendung hierarchischer Basen für die verfeinerten finiten Elemente. Die Darstellung orientiert sich dann i.w. an den Arbeiten der Entwickler von PLTMG (Randolph E. Bank: *Hierarchical Bases and the FEM*, Acta Numerica 1997) und KASKADE (Deuflhard, P., Leinen, P. und Yserentant, H.: *Concepts of an Adaptive Hierarchical Finite Element Code*, Impact Comp. Science and Eng. 1 (1989) 3–35). Numerische Experimente, die KASKADE und PLTMG vergleichen, findet man in: Erdmann, B., Roitzsch, R., Bornemann, F.: KASKADE – Numerical Experiments, TR 91-1, Konrad-Zuse-Zentrum, Berlin, 1991.

5.1 Fehlerschätzer nach Babuška und Rheinboldt

Für die Methode der finiten Elemente können Fehlerabschätzungen der Form

$$\|u - u_h\|_r \leq Ch^p \|u\|_s, \text{ mit } r = 0, 1 \text{ und } s = 1, 2$$

bewiesen werden. Sie besitzen mehrere Nachteile:

- Die Konstante C ist im allgemeinen nicht bekannt, nur für einfache Elementtypen kann man C explizit angeben.
- Die Normen $\|\cdot\|_s$ der exakten Lösung sind nicht bekannt.
- Bei einem Polynom-Ansatz vom Grad k setzen optimale Konvergenzraten in der H^1-Norm voraus, dass $u \in H^{k+1}$ ist; dies ist oft eine unrealistische Annahme.

Babuška und Rheinboldt haben 1978 erstmalig eine Idee publiziert, die auf einem gegebenen Gitter aus einer Schätzung des Fehlers lokaler Natur, d.h. einer Schätzung, in der über lokale Fehler summiert wird, auf notwendige Gitterverfeinerungen in bestimmten Regionen schließt. Diese Methode besitzt eine große Bedeutung für **adaptive Finite-Element-Methoden**, bei denen die Lage der Gitterpunkte automatisch von der Methode so fixiert werden, dass der Fehler der Näherungslösung eine vorgegebene Toleranz nicht überschreitet.

5.1.1 Fehlerschätzer und -indikatoren im eindimensionalen Fall

Wegen der beträchtlich größeren Schwierigkeiten im mehrdimensionalen Fall soll der Fehlerschätzer zunächst im eindimensionalen Fall an der Modellaufgabe

$$-u'' = g, \quad u(0) = u(1) = 0$$

in der schwachen Formulierung

$$B(u,v) := \int_0^1 u'v'dx = (g,v) \text{ für alle } v \in H_0^1(0,1) \tag{5.1}$$

untersucht werden.

Es sei u_h die Lösung des diskreten Problems bei Verwendung linearer finiter Elemente mit den Knoten $x_i,\ i = 0,\ldots,m$; die ψ_i seien die den Knoten zugeordneten stückweise linearen, stetigen Basisfunktionen*. Wir nutzen als Norm die zur H^1-Norm äquivalente Energie-Norm

$$\|u\|^2 := \|u\|_E^2 = \int_0^1 u'^2 dx = B(u,u) =: B(u)$$

im Raum $H_0^1(0,1)$.

Lemma 5.1.1 Es gibt Konstanten D_1, D_2 mit

$$D_1\eta \le \|u-u_h\| \le D_2\eta, \tag{5.2}$$

wobei

$$\eta^2 = \sum_{i=0}^m \eta_i^2 \text{ und } \eta_i := \sup_{v\in H_0^1} \frac{|B(u-u_h,\psi_i v)|}{\|\psi_i v\|} \tag{5.3}$$

bezeichnen.

Beweis: Wir beweisen die Abschätzung nach oben, der andere Teil verläuft ähnlich. Aus

$$B(u-u_h,v_h) = 0 \text{ für alle } v_h \in S_h$$

folgt (siehe auch NumPDE)

$$\|u-u_h\|^2 = \inf_{v_h\in S^h} |B(u-u_h, u-u_h-v_h)|.$$

Zur Abkürzung setzen wir $e := u-u_h$ und schätzen weiter ab. Da die $\{\psi_k\}$ eine Zerlegung der Eins bilden [d.h. $\sum_{i=0}^m \psi_i(x) = 1\ \forall\, x \in (0,1)$], folgt

$$\|e\|^2 = \inf_{v_h\in S^h} |B(e, \sum_{i=0}^m \psi_i(e-v_h))| \le \inf_{v_h\in S^h} \sum_{i=0}^m |B(e,\psi_i(e-v_h))|,$$

also

$$\|e\|^2 \le \inf_{v_h\in S^h} \sum_{i=0}^m \frac{|B(e,\psi_i(e-v_h))|}{\|\psi_i(e-v_h)\|} \|\psi_i(e-v_h)\|.$$

Die Definition von η_i und die Schwarzsche Ungleichung liefern

$$\|e\|^2 \le \sum_{i=0}^m \eta_i \inf_{v_h\in S^h} \|\psi_i(e-v_h)\| \le \eta \inf_{v_h\in S^h} (\sum_{i=0}^m \|\psi_i(e-v_h)\|^2)^{\frac{1}{2}}.$$

*Aus Beweis-technischen Gründen nehmen wir die den Randknoten zugeordneten Basisfunktionen mit hinzu trotz der Null-Randwerte.

Wählt man für v_h etwa die Interpolierende von e, so erhält man

$$\|e\|^2 \leq \eta D_2 \|e\|,$$

also einen Teil der Behauptung (5.2)

$$\|e\| \leq D_2 \eta. \qquad \blacksquare$$

Allgemein heißt eine Größe η, die (5.2) erfüllt, **Fehlerschätzer**. Da die η_i lokal definiert sind, heißt η_i **lokaler Fehlerschätzer**. Ein Fehlerschätzer heißt zudem **asymptotisch exakt**, falls gilt $\lim_{h\to 0} \eta/\|u - u_h\| = 1$.

Wichtig an einem Fehlerschätzer ist aber auch seine *einfache praktische Berechenbarkeit*, und damit sieht es bei den obigen η_i noch nicht so gut aus.

Lemma 5.1.2 Es gilt (für $i = 1, \ldots, m$)

$$\eta_i^2 = \int_{x_{i-1}}^{x_i} ((z_i - u_h)')^2, \tag{5.4}$$

wobei z die lokale Randwertaufgabe

$$-z_i'' = g, \quad z_i|_{x_{i-1}} = u_h|_{x_{i-1}}, \quad z_i|_{x_i} = u_h|_{x_i} \tag{5.5}$$

löst.

Beweis: Zunächst gilt für $i = 1, \ldots, m$

$$\eta_i = \sup_{w \in H_0^1(x_{i-1}, x_i)} \frac{|B(e, w)|}{\|w\|}.$$

Andererseits haben wir für die Einschränkung der Bilinearform auf das Intervall (x_{i-1}, x_i)

$$\begin{aligned} B(e,w) &= \int_{x_{i-1}}^{x_i} e'w'dx = -\int_{x_{i-1}}^{x_i} u''w\,dx - \int_{x_{i-1}}^{x_i} u_h'w'dx \\ &= \int_{x_{i-1}}^{x_i} gw\,dx - \int_{x_{i-1}}^{x_i} u_h'w'dx. \end{aligned}$$

Aus der Definition von z_i folgt

$$B(e,w) = \int_{x_{i-1}}^{x_i} (z_i - u_h)'w'dx.$$

Also gilt

$$|B(e,w)| \leq \Big(\int_{x_{i-1}}^{x_i} ((z_i - u_h)')^2 dx\Big)^{\frac{1}{2}} \|w\|.$$

Daraus folgt unmittelbar die Behauptung. $\blacksquare$

Mit Lemma 5.1.2 steht eine im eindimensionalen Fall fast (in Abhängigkeit vom gegebenen Differentialoperator) berechenbare Größe zur Verfügung. Es wird nun noch etwas weiter umgeformt. Es sei $\zeta_i := z_i - u_h$. Dann genügt η_i der Beziehung

$$\eta_i^2 = \int_{x_{i-1}}^{x_i} (\zeta_i')^2 dx, \tag{5.6}$$

und ζ_i löst die lokale Randwertaufgabe

$$-\zeta_i'' = r \quad \text{mit} \quad \zeta_i\Big|_{x_{i-1}} = \zeta_i\Big|_{x_i} = 0. \tag{5.7}$$

Das **Residuum** $r = g - Lu_h$(L ist der gegebene Differentialausdruck, in unserem Fall $Lv = -v''$) der Näherungslösung wird dabei stückweise berechnet. Löst man die Randwertaufgabe (5.7) für (pro Intervall) konstantes r und berechnet dann η_i, so ergibt sich

$$\eta_i^2 = \frac{r^2 h_i^3}{12} \text{ mit } h_i = x_i - x_{i-1}.$$

Allgemeiner zeigen Babuška und Rheinboldt im eindimensionalen Fall für homogene Gitter unter Glattheitsvoraussetzungen an u die asymptotische Exaktheit des gegenüber (5.6), (5.7) noch einmal vereinfachten **Residuum-Typ-Schätzers**

$$\eta_i^2 = \frac{h_i^2}{12 a_i} \int_{x_{i-1}}^{x_i} r^2 dx \tag{5.8}$$

für Probleme der Form $-(au')' + bu = g$ (mit $a_i = a(x_i)$).

Unter gewissen Voraussetzungen kann man auch die asymptotische Exaktheit eines inhomogenen Gitters zeigen (Babuska, Kellogg, Pitkaranta), sogar, wenn u gewisse Singularitäten hat wie beim folgenden Beispiel.

Beispiel 5.1.3 (Rheinboldt)

$$-u'' = \frac{3}{16} x^{-5/4}, \quad u(0) = u(1) = 0$$

hat die Lösung

$$u(x) = x^{3/4} - x.$$

Im Fall linearer Ansatzfunktionen ψ_i ist das Residuum gleich der rechten Seite:

$$r_i(x) = \frac{3}{16} x^{-5/4} \text{ in } [x_{i-1}, x_i], \quad i = 1, \ldots, m.$$

Die Lösungen ζ_i von (5.7) sind damit

$$\zeta_i(x) = x^{3/4} - h_i \left[x_{i-1}^{3/4} (x_i - x) + x_i^{3/4} (x - x_{i-1})\right].$$

Mit $\delta_i := x_{i-1}/h_i$ wird der Fehlerschätzer (5.6) zu

$$\eta_i^2 = \int_{x_{i-1}}^{x_i} (\zeta_i')^2 dx = \frac{1}{8} \Phi(\delta_i)\, h_i^{1/2}$$

mit

$$\Phi(\delta_i) := 9\,[(1+\delta_i)^{1/2} - \delta_i^{1/2}] - 8\,[(1+\delta_i)^{3/4} - \delta_i^{3/4}]^2 \ .$$

Es ist leicht zu sehen, dass Φ monoton fallend ist für $\delta_i \geq 0$. Deshalb gilt

$$\eta_i \leq \frac{1}{\sqrt{8}}\, h_i^{1/4} \ .$$

Rheinboldt zeigt noch, dass es für alle Intervalle, für die

$$\delta_i \geq \delta := \left(\frac{6\sqrt{6}}{16\,\pi}\right)^{4/5}$$

gilt, eine kleinere Schranke gibt:

$$\eta_i \leq \frac{1}{\sqrt{8}}\, \delta^{5/4} x_{i-1}^{-5/4} h_i^{3/2} \ .$$

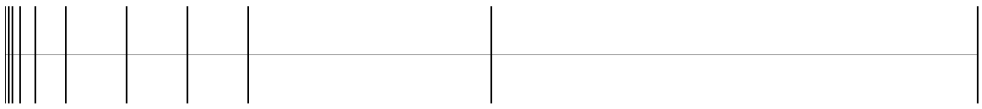
Eine untere Schranke ergibt sich auch aus der Monotonie der Funktion Φ:

$$\eta_i \geq \frac{1}{\sqrt{8}}\, \Phi(\delta)\, h_i^{1/4} \quad \text{für } \delta_i \leq \delta.$$

Rheinboldt beweist basierend auf diesen Abschätzungen ein Konvergenzresultat für Randsingularitäten, das sich allerdings auf Singularitäten im Innern des Intervalls nicht übertragen lässt.

Werden diese Fehlerschätzer angewendet, so ergibt sich für das Intervall $[0,1]$ mit der Singularität bei $x = 0$ für die grobe Schranke $\eta_i \leq 0.1$ das folgende Gitter:

$$x = [0,\ 1/256,\ 1/128,\ 1/64,\ 1/32,\ 1/16,\ 1/8,\ 3/16,\ 1/4,\ 1/2,\ 1]$$



5.1.2 Fehlerschätzer im zweidimensionalen Fall

Ohne Beweise auszuführen, soll die Situation jetzt auf den zweidimensionalen Fall übertragen werden. Es sei

$$B(u,v) := \int_\Omega (\nabla u \nabla v + (b\nabla u + cu)v)dx$$

mit $u, v \in H_0^1(\Omega)$ die der schwachen Formulierung eines Dirichletproblems zugeordnete Bilinearform. Diskretisiert werde mit linearen finiten Elementen. Es seien ψ_i die dem Knoten P_i zugeordnete Basisfunktion, E_i der Träger von ψ_i und $a_i(\cdot,\cdot)$ die Einschränkung der gegebenen Bilinearform auf E_i. Dann gelten sinngemäß Lemma 5.1.1 und Lemma 5.1.2, d.h., definiert man

$$\eta_i^2 = \|u_i - u_h\|^2, \tag{5.9}$$

wobei u_i das lokale Randwertproblem

$$\begin{aligned} a_i(u_i, v) &= (f, v) \text{ für alle } v \in H_0^1(E_i) \\ \text{mit } u_i|_{\partial E_i} &= u_h|_{\partial E_i} \end{aligned} \tag{5.10}$$

löst, so ist η_i ein lokaler Fehlerschätzer. Im Gegensatz zum eindimensionalen Fall gelingt aber jetzt der Übergang zu einem praktisch berechenbaren Schätzer (wie etwa (5.8)) nicht!

Wir betrachten den Spezialfall des Laplace-Operators mit $B(u, v) := (\nabla u, \nabla v)$. K_{ij} mit $j \in \Lambda_i$ sei die Menge aller zu E_i gehörenden Dreiecke, Γ_{ij} seien die inneren Kanten benachbarter Dreiecke. Nun gilt für

$$e_i = u_i - u_h$$

die Beziehung $e_i \in H_0^1(E_i)$, ferner

$$a_i(e_i, v) = (f, v) - a_i(u_h, v). \tag{5.11}$$

Daraus folgt

$$a_i(e_i, v) = \sum_{j \in \Lambda_i} [((f, v) + (\Delta u_h, v))_{K_{ij}} - \int_{\partial K_{ij}} \frac{\partial u_h}{\partial n} v \, ds].$$

Dabei ist $r = f + \Delta u_h$ wieder das Residuum der Näherungslösung. Führt man nun den Sprung $[\frac{\partial u_h}{\partial n}]$ der Normalenableitung auf Γ_{ij} ein und wendet die Schwarzsche Ungleichung sowie die Friedrichssche Ungleichung

$$(\int_{E_i} e_i^2 dx)^{\frac{1}{2}} \leq \gamma_i (\int_{E_i} (\nabla e_i)^2 dx)^{\frac{1}{2}}$$

und einen Spursatz vom Typ

$$(\int_{\partial K_{ij}} e_i^2 dx)^{\frac{1}{2}} \leq \delta_{ij} (\int_{K_{ij}} (\nabla e_i)^2 dx)^{\frac{1}{2}}$$

an, so erhält man die Abschätzung

$$\|u - u_h\| \leq D_2^* \eta \quad \text{mit} \quad \eta^2 = \sum \eta_i^2$$

und

$$\eta_i^* = \gamma_i (\int_{E_i} r^2 dx)^{\frac{1}{2}} + \max_{j \in \Lambda_i} \delta_{ij} (\int_{\Gamma_{ij}} [\frac{\partial u_h}{\partial n}]^2 ds)^{\frac{1}{2}}. \tag{5.12}$$

Da offen ist, ob auch eine entsprechende Abschätzung $D_1^* \eta \leq \|u - u_h\|$ nach unten gilt, heißt (5.12) lokaler **Verfeinerungsindikator**. Praktisch ist die Arbeit mit diesem Verfeinerungsindikator deshalb diffizil, weil man die Konstanten γ_i und δ_{ij} schwer in den Griff bekommt. Abschätzungen zeigen das asymptotische Verhalten $\gamma_i = O(h_i)$ und $\delta_{ij} = O(h_i^{\frac{1}{2}})$.

Für bilineare Elemente und spezielle Netze zeigten Babuška und Miller in einem äußerst anspruchsvollen Beweis die asymptotische Exaktheit des Schätzers

$$\lambda_i^2 = h_i^2 \int_{E_i} r^2 dx + h_i \sum_j \int_{\Gamma_{ij}} [\frac{\partial u_h}{\partial n}]^2 ds.$$

Voreilig zogen einige Autoren den Schluss, dass damit in jedem Fall eine theoretische Absicherung dafür vorliege, dass (5.12) asymptotisch exakter Fehlerschätzer sei. In Wirklichkeit ist dies für Dreieckszerlegungen und entsprechende finite Elemente ein offenes Problem.

Man erhält einen praktikablen Fehlerschätzer durch die Anwendung der naheliegenden Idee, das lokale Randwertproblem (5.11) nicht exakt, sondern näherungsweise zu lösen. Im eindimensionalen Fall wurde diese Idee für die Rheinboldt-Abschätzung (5.8) auch schon angewandt. Im zweidimensionalen Fall hilft das Einfrieren der Koeffizienten allein nicht, die resultierenden Probleme lassen sich nicht exakt lösen. Ein Ausweg besteht darin, die lokalen Randwertprobleme mit einem Verfahren höherer Ordnung anzugehen. Bank und Weiser lieferten eine theoretische Begründung für ein derartiges Vorgehen, benutzen dabei allerdings eine schwer nachprüfbare Superapproximationsvoraussetzung. Im Fall linearer finiter Elemente wird dabei vorgeschlagen, eine Näherungslösung $\overline{e}_i$ folgendermaßen zu bestimmen: Es sei K_h^d der Raum der stückweise quadratischen Funktionen, die in den Eckpunkten verschwinden. Als Freiheitsgrade werden in jedem Dreieck die Werte in den Seitenmittelpunkten gewählt. Durch die getrennte Vorgabe der Parameter über jedem Dreieck ist keine globale Stetigkeit gesichert. Die Ermittlung von $\overline{e}_i$ als Näherungslösung von (5.11) in diesem Raum zerfällt nun in unabhängige Teilaufgaben über den einzelnen Dreiecken. Es sind somit lediglich Gleichungssysteme vom Format 3×3 zu lösen. Dieser Schätzer fand Eingang in das weit verbreitete Programmpaket PLTMG [57]. Auch bei dem auf hierarchischen Basen beruhenden System KASKADE werden ähnliche Ideen zur Fehlerschätzung genutzt.

5.2 Hierarchische Finite Elemente Methoden (PLTMG, KASKADE)

Soll eine Programm erstellt werden, das Fehlerschätzer und -indikatoren benutzt, um ein gegebenes grobes Gitter adaptiv zu verfeinern, dann ist eines der Ziele *Einfachheit*. Dazu gehört u.a. die Lokalität der Fehlerindikatoren, die zur adaptiven Verfeinerung benutzt werden. Dazu gehört auch, dass die globale Steifigkeitsmatrix möglichst nirgendwo im Rechner akkumuliert werden muss, d.h. dass auch die Berechnung der FEM-Lösung mit lokalen Daten auskommt. Damit diese Einfachheit durchführbar ist, müssen die drei wesentlichen Komponenten der Rechnung zusammen betrachtet werden:

- Der iterative Löser
- Der lokale Fehlerschätzer
- Die lokale Netzverfeinerung

Wir wollen die Grundidee von PLTMG und KASKADE zunächst eindimensional verstehen. Dann wird das Verfahen, das KASKADE anwendet, im Einzelnen erläutert. Schließlich soll an Beispielen die Wirkung der Methode gesehen werden. Bank liefert mehr theoretischen Hintergrund für seine ähnliche Idee. Darauf muss leider verzichtet werden. Die Verfeinerungstechniken in den beiden Programmpaketen sollen nur kurz erwähnt werden, weil im folgenden Abschnitt eine allgemeine Einführung in verschiedene Algorithmen zur Verfeinerung im $\mathbb{R}^2$ und $\mathbb{R}^3$ dargestellt wird.

5.2.1 Grundlagen

Sei $\Omega \subset \mathbb{R}^2$ ein beschränktes Polygon mit dem Rand $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$. Wir betrachten das (gegenüber der angegebenen Literatur vereinfachte) elliptische Randwertproblem

$$\left.\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{auf } \partial\Omega_1 \\ \vec{n} \cdot \nabla u &= 0 \quad \text{auf } \partial\Omega_2 \end{aligned}\right\} \tag{5.13}$$

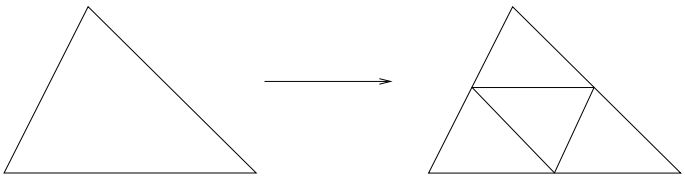
Dabei ist $\vec{n}$ der Vektor in Richtung der äußeren Normale. Wir wollen voraussetzen, dass das Problem eine eindeutige Lösung besitzt ($\partial\Omega_1 \neq \emptyset$) und definieren wieder die schwache Formulierung

$$B(u, v) = (f, v) \quad \forall v \in H \tag{5.14}$$

mit der Bilinearform

$$B(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dx \tag{5.15}$$

$\bar{\Omega}$ sei mit einer groben Triangulierung $\mathcal{T}_0$ überdeckt. Verfeinerungen $\mathcal{T}_1$, $\mathcal{T}_2$, ..., von $\mathcal{T}_0$ seien zunächst als homogen vorausgesetzt, d.h. jedes Dreieck wird in 4 kongruente Dreiecke zerlegt:



Jeder Triangulierung wird der Raum der stückweise linearen Polynome zugeordnet:

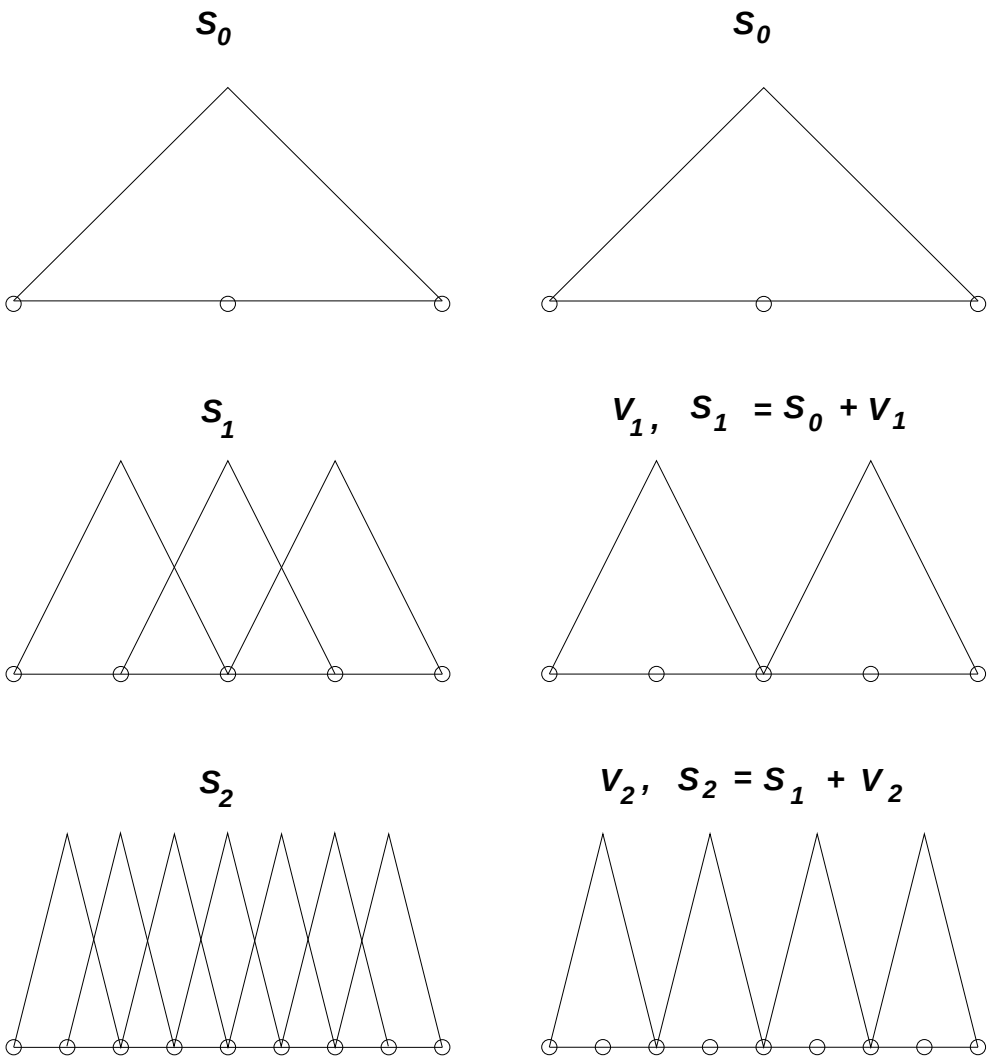
$$\mathcal{T}_i : \mathcal{S}_i \quad \text{mit} \quad \dim(\mathcal{S}_i) = n_i.$$

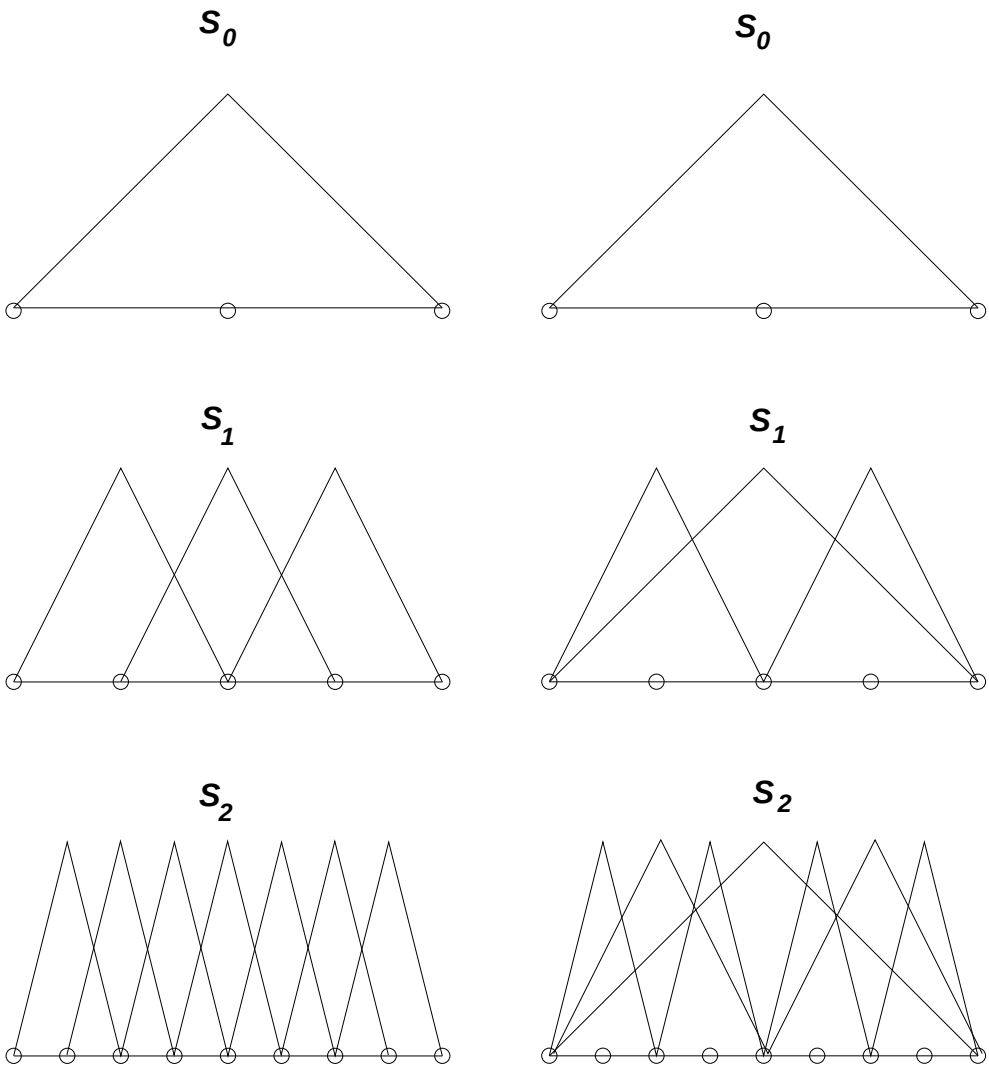
Die Räume $\mathcal{S}_i$ können jetzt verschieden durch Basen repräsentiert werden. Einerseits durch ihre Knotenbasen, d.h. durch die einem inneren Punkt zugeordneten Basiselemente. Andererseits durch *hierarchische Basen*:

$$\mathcal{S}_{i+1} = \mathcal{S}_i \oplus \mathcal{V}_i.$$

Der Unterschied soll im nächsten Abschnitt an Basen im $\mathbb{R}^1$ klargemacht werden.

5.2.2 Hierarchische und Knotenbasen im $\mathbb{R}^1$




S0
S0
S1
S1
S2
S2

5.2.3 Lösung der linearen Gleichungssysteme

Die Lösung der schwachen Formulierung (5.14) führt mit einer Knotenbasis $\{\psi_1, \ldots, \psi_{n_i}\}$ in $\mathcal{S}_i$ zu einem großen, dünn besetzten linearen Gleichungssystem

$$B(\sum_{k=1}^{n_i} \alpha_k \psi_k, \psi_j) = (f, \psi_j), \quad j = 1, \ldots, n_i, \tag{5.16}$$

oder

$$A_i u_i = b_i \tag{5.17}$$

mit $u_i = (\alpha_1, \ldots, \alpha_{n_i})^T$ und der Lösung

$$u_h(x) = \sum_{k=1}^{n_i} \alpha_k \psi_k. \tag{5.18}$$

Die hierarchische Basis führt auf das Gleichungssystem

$$B(\sum_{k=1}^{n_i} \beta_k \phi_k, \phi_j) = (f, \phi_j), \quad j = 1, \ldots, n_i, \tag{5.19}$$

oder

$$\bar{A}_i \bar{u}_i = \bar{b}_i. \tag{5.20}$$

mit $\bar{u}_i = (\beta_1, \ldots, \beta_{n_i})^T$ und der Lösung

$$\bar{u}_h(x) = \sum_{k=1}^{n_i} \beta_k \phi_k = u_h(x). \tag{5.21}$$

Dies ist natürlich dieselbe Lösung, aber die Gleichungssysteme, die sie liefern, sind unterschiedlich. Sie genügen folgenden Beziehungen:

$$u_i = S_i \bar{u}_i, \quad \bar{b}_i = S_i^T b_i, \quad \bar{A}_i = S_i^T A_i S_i. \tag{5.22}$$

Setzt man eine natürliche Ordnung der Basisfunktionen voraus, dann kann man eine symmetrisch positiv-definite $n_i \times n_i$-Matrix definieren:

$$\bar{D}_i := \begin{pmatrix} \bar{A}_0 & 0 \\ 0 & D_i \end{pmatrix} =: L_i L_i^T. \tag{5.23}$$

Dabei enthält die Diagonalmatrix D_i diejenigen Einträge von $\bar{A}_i$, die nicht in $\bar{A}_0$ enthalten sind. Für die Matrix

$$\hat{A}_i := L_i^{-1} \bar{A}_i L_i^{-T} = L_i^{-1} S_i^T A_i S_i L_i^{-T}, \tag{5.24}$$

hat Yserentant gezeigt, dass ihre Konditionszahl

$$\kappa(\hat{A}_i) \leq C_1 \, (i+1)^2 \tag{5.25}$$

erfüllt, wobei die Konstante C_1 unabhängig von i beschränkt ist. Im Vergleich müssen wir die Konditionszahl zur Matrix der Knotenbasis sehen:

$$\kappa(A_i) \leq C_2 \, 4^i. \tag{5.26}$$

Daher liegt nahe, die Matrix

$$S_i^{-T} \bar{D}_i S_i^{-1} \tag{5.27}$$

als Vorkonditionierer für das lineare System (5.17) zu nehmen. Das bedeutet, dass das lineare System

$$\hat{A}_i \hat{u}_i = \hat{b}_i, \quad \hat{b}_i := L_i^{-1} S_i^T b_i \tag{5.28}$$

mit dem cg-Verfahren effizient gelöst wird. Die Knotenbasislösung erhält man dann als

$$u_i = S_i L_i^{-T} \hat{u}_i. \tag{5.29}$$

Deuflhard, Leinen und Yserentant untersuchen die Beziehungen zwischen Konditionszahlen, Aufwand, erreichbarer Genauigkeit und vorkonditioniertem cg-Verfahren genauer und kommen damit zu der *Kaskaden-Iteration*, die ein spezielles Multilevel-Verfahren darstellt:

Statt (5.17) für ein $i = j$ zu lösen, wird eine Kaskade von linearen Gleichungssystemen wie folgt gelöst:

Auf dem gröbsten Gitter $\mathcal{T}_0$ wird das Gleichungssystem $A_0 u_0 = b_0$ mit dem Cholesky-Verfahren gelöst.

Auf jeder Stufe (Level) $i > 0$ werden Approximationen $\tilde{u}_i$ von u_i durch einige vorkonditionierte cg-Schritte berechnet. Als Startwert gilt dabei $u_i^{(0)} = \tilde{u}_{i-1}$ (Interpolation auf das feinere Gitter).

Mit vernünftig gewählten Abbruchkriterien kommt man auf einen Rechenaufwand von $O(n \operatorname{ld} n)$ (log dualis), wenn $n = n_j$ die feinste Verfeinerungsstufe ist.

5.2.4 Fehlerschätzung

Um die Kaskadeniteration des letzten Abschnitts implementieren zu können, braucht man einen guten Fehlerschätzer. Den braucht man natürlich auch für nicht-homogene Verfeinerungen der Triangulierungen, also für ein adaptives Verfahren.

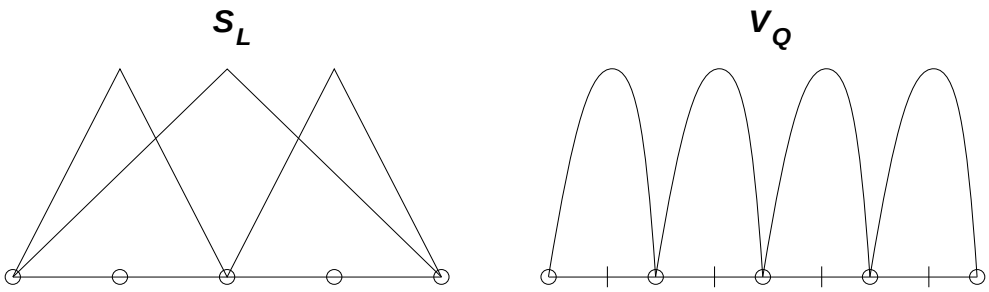
Bank und Weiser haben für das Paket PLTMG einen zuverlässigen und effizienten Fehlerschätzer entwickelt, der *Dreiecks-orientiert* ist. Für KASKADE wurde ein einfacherer Fehlerschätzer entwickelt, der für hierarchische Basen besonders geeignet ist und der *Kanten-orientiert* ist. Der soll hier vorgestellt werden.

Sei $\mathcal{T}$ die Triangulierung, auf die der Fehlerschätzer angewendet werden soll. Seien $\mathcal{S}_L$ und $\mathcal{S}_Q$ die Räume der stückweise linearen bzw. quadratischen finite-Elemente-Funktionen für $\mathcal{T}$. Der Fehlerschätzer soll dann den Abstand zwischen einer Näherungslösung in $\mathcal{S}_L$ und der exakten Lösung in $\mathcal{S}_Q$ schätzen. $\mathcal{S}_Q$ kann in die direkte Summe

$$\mathcal{S}_Q = \mathcal{S}_L \oplus \mathcal{V}_Q$$

zerlegt werden. Die Funktionen in $\mathcal{V}_Q$ verschwinden in den Ecken der Dreiecke von $\mathcal{T}$ und können charakterisiert werden durch ihre Werte in den Mittelpunkten der Dreieckskanten.

Nun sei der Raum $\mathcal{S}_L$ durch eine hierarchische Basis definiert. Dann ist die hierarchische Basis von $\mathcal{S}_Q$ auf natürliche Weise definiert durch die Hinzufügung der Knotenbasisfunktionen von $\mathcal{V}_Q$. Für den eindimensionalen Fall demonstriert die folgende Zeichnung diese Vorgehensweise:



Der Koeffizientenvektor jeder Funktion in $\mathcal{S}_Q$ kann in seine Anteile in $\mathcal{S}_L$ und in $\mathcal{V}_Q$ zerlegt werden:

$$v = \begin{pmatrix} v_L \\ v_Q \end{pmatrix} \tag{5.30}$$

Entsprechend können die Gleichungssysteme der FEM-Lösungen in $\mathcal{S}_L$ und in $\mathcal{S}_Q$ aufgebaut werden:

$$A_{LL} u_L = b_L \quad \text{in } \mathcal{S}_L \tag{5.31}$$

$$\begin{pmatrix} A_{LL} & A_{LQ} \\ A_{QL} & A_{QQ} \end{pmatrix} \begin{pmatrix} u_L^* \\ u_Q^* \end{pmatrix} = \begin{pmatrix} b_L \\ b_Q \end{pmatrix} \quad \text{oder kurz: } Au^* = b \quad \text{in } \mathcal{S}_Q \tag{5.32}$$

Nehmen wir jetzt an, dass eine Näherungslösung $\tilde{u}_L$ von (5.31) vorliegt. In der hierarchischen Basis von $\mathcal{S}_Q$ repräsentiert der Koeffizientenvektor $(\tilde{u}_L, 0)^T$ dieselbe Funktion. Der Defekt

$$\begin{pmatrix} d_L \\ d_Q \end{pmatrix} := \begin{pmatrix} u_L^* \\ u_Q^* \end{pmatrix} - \begin{pmatrix} \tilde{u}_L \\ 0 \end{pmatrix} \tag{5.33}$$

stellt gerade die Differenz zwischen der exakten Lösung in $\mathcal{S}_Q$ und der Näherungslösung $\tilde{u}_L$ in $\mathcal{S}_L$ dar. Er genügt der Gleichung

$$\begin{pmatrix} A_{LL} & A_{LQ} \\ A_{QL} & A_{QQ} \end{pmatrix} \begin{pmatrix} d_L \\ d_Q \end{pmatrix} = \begin{pmatrix} r_L \\ r_Q \end{pmatrix} \tag{5.34}$$

mit

$$r_L := b_L - A_{LL}\,\tilde{u}_L, \tag{5.35}$$

$$r_Q := b_Q - A_{QL}\,\tilde{u}_L. \tag{5.36}$$

Die Lösung von (5.34) ist zu aufwändig. Sie wird deshalb durch ein viel einfacheres Gleichungssystem

$$B\tilde{d} = r \tag{5.37}$$

approximiert. Bei Verwendung hierarchischer Basen ist

$$B = \begin{pmatrix} B_{LL} & B_{LQ} \\ B_{QL} & B_{QQ} \end{pmatrix} = \begin{pmatrix} D_{LL} & 0 \\ 0 & D_{QQ} \end{pmatrix} \tag{5.38}$$

eine natürliche Wahl. Dabei ist D_{LL} die schon in (5.23) definierte 'fast-diagonale' Matrix, und D_{QQ} ist einfach die Diagonale von A_{QQ}. $\tilde{d}_L$ und $\tilde{d}_Q$ können jetzt getrennt berechnet werden aus den Gleichungssystemen

$$D_{LL}\tilde{d}_L = r_L, \quad D_{QQ}\tilde{d}_Q = r_Q. \tag{5.39}$$

Das Ersetzen von d durch $\tilde{d}$ bei der Fehlerschätzung bedeutet, dass statt der optimalen globalen Korrektur in $\mathcal{S}_Q$ lokale eindimensionale Korrekturen in Richtung der hierarchischen Basisfunktionen berechnet werden, die $\mathcal{S}_Q$ aufspannen. Ist $\tilde{u}_L$ die exakte Lösung in $\mathcal{S}_L$, dann entspricht diese Vorgehensweise dem Lösen von eindimensionalen Dirichletproblemen auf den Kanten der Triangulierung $\mathcal{T}$. Randkanten werden entsprechend dem Ausgangsproblem berücksichtigt.

Ein geeignetes Maß für die Größe des Fehlers ist seine Energienorm:

$$|A^{1/2}d|^2 = (d, Ad), \tag{5.40}$$

wenn $(\cdot,\cdot)$ das euklidische innere Produkt und $|\cdot|$ die zugehörige Norm bedeuten. Dieser Wert wird approximiert durch

$$|B^{1/2}\tilde{d}|^2 = |D_{LL}^{1/2}\tilde{d}_L|^2 + |D_{QQ}^{1/2}\tilde{d}_Q|^2. \tag{5.41}$$

Dabei ist

$$|D_{LL}^{1/2}\tilde{d}_L|^2 = (\tilde{d}_L, D_{LL}\tilde{d}_L) = (D_{LL}^{-1}r_L, r_L) = |D_{LL}^{-1/2}r_L|^2, \tag{5.42}$$

$$|D_{QQ}^{1/2}\tilde{d}_Q|^2 = (\tilde{d}_Q, D_{QQ}\tilde{d}_Q) = (D_{QQ}^{-1}r_Q, r_Q) = |D_{QQ}^{-1/2}r_Q|^2. \tag{5.43}$$

Jetzt muss untersucht werden, ob (5.41) eine genügend gute Approximation für (5.40) ist. Hier soll nur das Ergebnis für den Fall, dass $\tilde{u}_L$ exakte Lösung in $\mathcal{S}_L$ ist, wiedergegeben werden:

$$\frac{1}{\sqrt{\gamma_1}}\,|A^{1/2}d| \le |B^{1/2}\tilde{d}| \le 2|A^{1/2}d|. \tag{5.44}$$

Dabei hängt γ_1 nur von der lokalen Elliptizität des Randwertproblems und der Dreiecksform (kleinster Winkel) ab. Wie effektiv dieser Schätzer ist, werden wir an einem Beispiel sehen.

5.2.5 Die Netzverfeinerungs-Strategie

Es soll der Schritt von $i = 0$ auf $i = 1$ betrachtet werden. Die Lösung u_0 auf $\mathcal{T}_0$ wird mit einem exakten Löser berechnet, z.B. mit dem Cholesky-Verfahren.

Dann werden die Komponenten des Fehlerschätzers

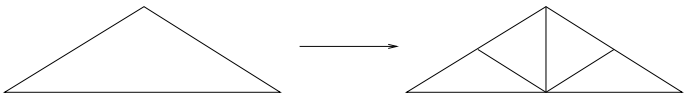
$$D_{QQ}^{1/2}\tilde{d}_Q = D_{QQ}^{-1/2}r_Q \tag{5.45}$$

für jede Kante in $\mathcal{T}_0$ berechnet. Dann wird eine Fehlerschwelle Θ berechnet als

$$\Theta := \sigma\bar{m} \quad \text{mit} \quad \sigma := 0.95. \tag{5.46}$$

Dabei ist $\bar{m}$ der Mittelwert der Quadrate der Komponenten des gewichteten Residuums (5.45).

Jetzt wird jedes Dreieck, das eine 'markierte' Kante besitzt, regulär verfeinert ('rot'), d.h. wie in 5.2.1 gezeigt. Eine Ausnahme stellen stumpfwinklige Dreiecke dar, bei denen der stumpfe Winkel geteilt wird:



Die Triangulierung wird dann konform vervollständigt durch weitere rote und grüne Verfeinerung einzelner Dreiecke, siehe auch Abschnitt 5.3.

5.2.6 Beispiel

KASKADE benutzt in der Regel drei Eingabedateien zur Definition eines Beispiels. Wir wollen das Beispiel *Corner* gründlich durchgehen. Die drei Eingabedateien sind

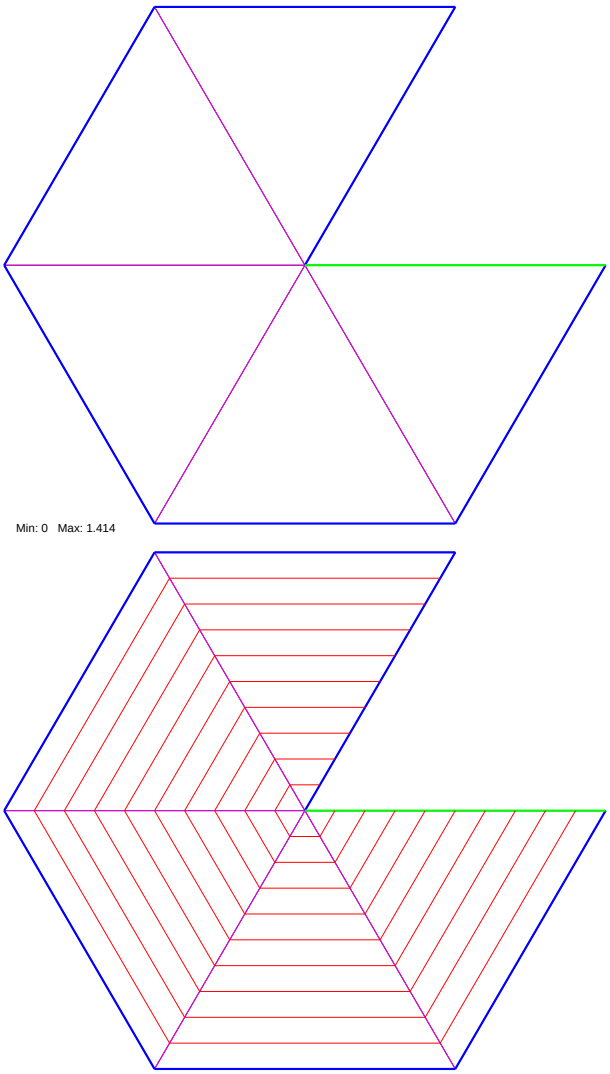
```
#  corner.geo

Points: maxIndex 8
 2  1.000000e+00 1.7320000e+00
 3 -1.000000e+00 1.7320000e+00
 4 -2.000000e+00 0.000000e+00
 5 -1.000000e+00 -1.7320000e+00
 6  1.000000e+00 -1.7320000e+00
 7  2.000000e+00 0.000000e+00
 8  0.000000e+00 0.000000e+00
END

Elements:
8 2 3 1
4 8 3 1
8 4 5 1
8 5 6 1
8 6 7 1
END

Boundary:
2 8 d 1
2 3 d 1
3 4 d 1
4 5 d 1
5 6 d 1
6 7 d 1
7 8 n 1
END

# corner.mat
Parameters:
width   1
height  1
end

Boundary:
1  D 10.0
1  N  0.0
end

Materials:
isotropic
```

```
1  e 1  s 0  end
end

Factors:
end
```

```
# corner.ex
# -------------- read options -------------
file=corner.geo
matFile=corner.mat
# ----------------------------------------
#spaceDim=2

DirichletBCs=RootOfRBCs
```

Jetzt wird die graphische Benutzeroberfläche von KASKADE aufgerufen und das Beispiel `corner` geladen. Nach 'Start' erscheinen die Ergebnisse der ersten Triangulierung und ein Graphik-Fenster mit der Triangulierung und zugehörigen Isolinien:

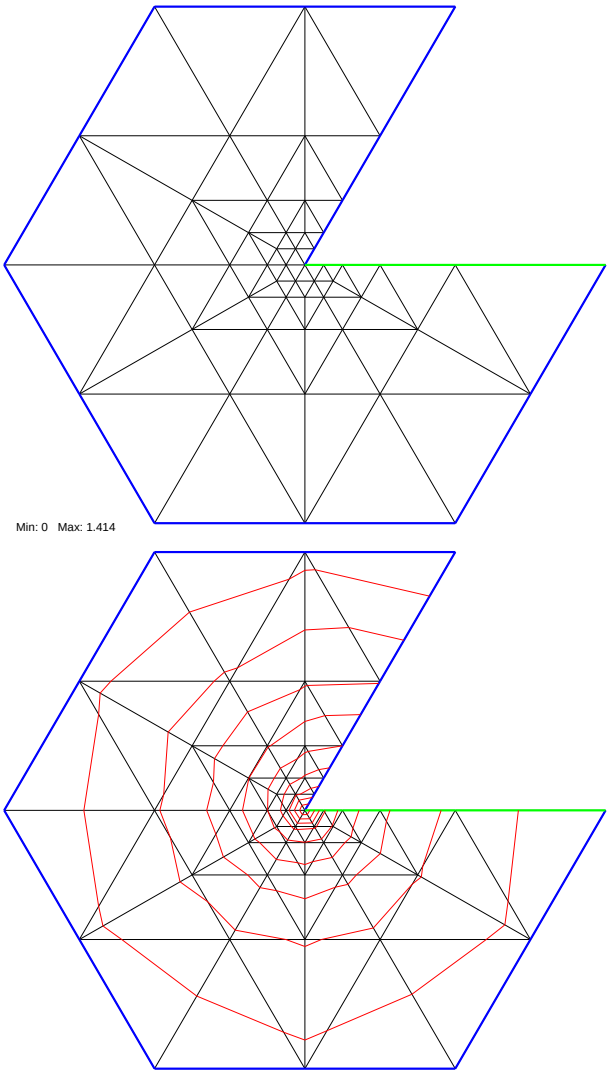
```
  Command File kaskade.init read

  Command File /runge/user/norbert/KASK/KASKADE.3.1.1/zgui/gui.ex read

  Command overwritten:  dirichletbcs = constdirichletbcs
dirichletbcs = rootofrbcs  <-- used value
  Command overwritten:  file = unit-2d.geo
file = corner.geo  <-- used value

 Triangulation:    points     edges triangles   Depth=0  Steps=0
                        7        11         5

Direct solution by LU-Factorization

    MA28 Decomposition: FLAG = 0 N = 7 Entries = 7 (0.000816 MB)
    U = 0.10  NSearch = 1  irncp=0  icncp=0  fill-in: 0 (f=0.00)
    MINICN=7 (0.25 LICN=28)  MINIRN=7 (0.50 LIRN=14)
    dropTol = 1e-08 (absTol = 5.77e-09)   NDROP = 0
  F=2.88672  FRes/F=0

 RefStep level  nodes      Energy Error(%)  cpu(sec)
       0     0      7     2.88672       20      0.00
 ------------------------------------------------------------------------------
```



Nach vier Verfeinerungsschritten ergibt sich

```
-------------------------------------------------------------------------

   Refine:        points     edges triangles   Depth=4  Steps=4
      previous        40        99        60
         total        51       130        80   Space : 0.041632 MB
         ratio      1.27      1.31      1.33   Blocks: 4

Linear Solver: CG -- SGS ML Preconditioner
         iter  r/b <   0.0001   ratio(r) <ratio(r)>   |b|=4.35
            1          0.0177     0.0835    0.0835
            3        5.99e-05     0.0338    0.0656
 F=1.17746  FRes/F=3.7e-05
```

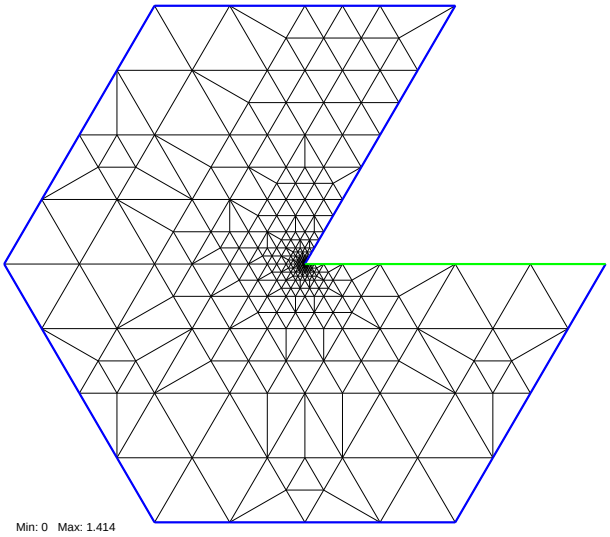
```
RefStep level  nodes      Energy Error(%)   ratio    dE(%)  cpu(sec)
      4     4     51     1.17746     12.5    0.73    -13.4      0.19
-------------------------------------------------------------------------
```

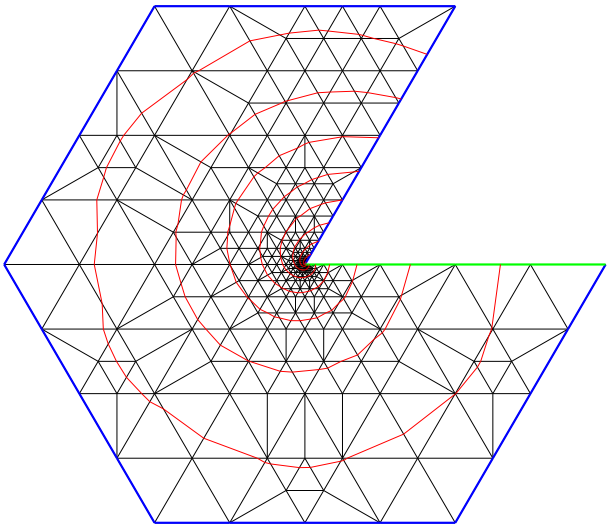


Nach 13 Schritten bekommen wir

```
-------------------------------------------------------------------------

   Refine:        points     edges triangles   Depth=13  Steps=13
      previous       295       831       537
         total       365      1035       671   Space : 0.174128 MB
         ratio      1.24      1.25      1.25   Blocks: 16
```

```
Linear Solver: CG -- SGS ML Preconditioner
          iter  r/b <   0.0001   ratio(r) <ratio(r)>   |b|=10
             1         0.00135     0.0584    0.0584
             4        2.98e-05      0.168      0.19
  F=0.829765  FRes/F=0.00026

 RefStep level  nodes      Energy Error(%)   ratio    dE(%)  cpu(sec)
      13    13    365    0.829765     2.44    0.78    -1.09      2.31
 ------------------------------------------------------------------------------
```





5.3 Algorithmen zur Netzverfeinerung

In diesem Kapitel werden verschiedene Algorithmen zur Verfeinerung von FE-Netzen vorgestellt. Die einzelnen Algorithmen wurden in verschiedenen Papern präsentiert, in denen oft auch eine Aussage über die Qualität der Verfeinerung zu finden ist. Eine recht gute Übersicht gibt das Paper "Adaptive Refinement of Unstructured Finite-Element Meshes" von T. Jones und P. Plassmann [39]. Die Verfeinerungsalgorithmen gehen davon aus, dass in einem gegebenen FE-Netz durch einen Fehlerschätzer oder Fehlerindikator Elemente ermittelt wurden, die in Subelemente aufgeteilt werden sollen. Bei der Verfeinerung wird darauf geachtet, dass wieder ein konformes FE-Netz entsteht. Es ist offensichtlich, dass ein nicht konformes Netz viel einfacher zu erzeugen ist, während die Erzeugung eines konformen Netzes auch die Unterteilung von Elementen zur Folge hat, die eigentlich nicht verfeinert werden müssten. Die Netzverfeinerung ist notwendig, da es fast unmöglich ist, vor der FEM-Berechnung zu entscheiden, wie fein das generierte FE-Netz sein muss, um ein akzeptables Ergebniss zu erhalten. Desweiteren soll ein möglichst kleines FE-Netz bzgl. der Elementeanzahl erzeugt werden, da diese einen entscheidenden Einfluss auf die Berechnungszeit besitzt. Aus diesem Grund startet eine FEM-Berechnung auf einem möglichst groben Netz, welches dann in den durch den Fehlerschätzer ermittelten Gebieten verfeinert wird.

Im folgenden wird ein initiales Netz M_0, bestehend aus Dreieckselementen, Tetraedern oder Hexaedern als gegeben vorausgesetzt. Der grobe Ablauf einer FEM-Berechnung kann dann wie folgt beschrieben werden:

1: $i = 0$
2: Berechne eine Lösung der PDE auf M_i
3: Schätze den Fehler für jedes Element
4: Markiere die Elemente mit zu großem Fehler
5: Falls markierte Elemnente existieren
6: Berechne M_{i+1} durch Verfeinerung der Elemente
7: $i = i + 1$
8: Wiederhole ab 2:

Die durch den Fehlerschätzer im i-ten Schleifendurchlauf markierten Elemente bilden in den folgenden Betrachtungen die Menge T_i.

5.3.1 2D Netzverfeinerung

Einfügen eines Steinerpunktes

Der wohl einfachste Verfeinerungsansatz besteht darin, den Mittelpunkt des zu verfeinernden Elementes als einen neuen Steinerpunkt einzufügen und mit den drei Eckpunkten zu verbinden. Dieser Ansatz hat den Vorteil, dass direkt ein konformes verfeinertes Netz entsteht, und sich die Verfeinerung auf die markierten Elemente beschränkt. Der große Nachteil besteht jedoch in der drastischen Verkleinerung der auftretenden Winkel bei einer mehrmaligen Verfeinerung, da jeder Verfeinerungsschritt die Winkel der markierten Elemente halbiert. Damit wird sehr schnell ein Netz erzeugt, das sich nicht mehr für FEM-Berechnungen eignet.

Algorithmus 0

1: for all $t \in T_i$ do
2: Füge den Mittelpunkt von t ein
3: Verbinde den Mittelpunkt mit den Knoten von t

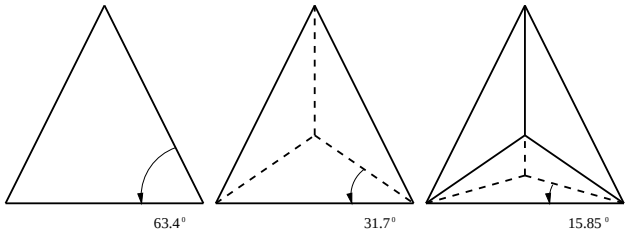


Abbildung 5.1: Einfügen des Mittelpunktes und die Verkleinerung der Winkel

Bisektion über die längste Kante

Dieser Algorithmus von M.C. Rivara [59] teilt immer die längste Kante des zu verfeinernden Elements. Der Mittelpunkt der längsten Kante wird als neuer Steinerpunkt eingefügt und mit dem gegenüberliegenden Knoten verbunden. Die durch die Unterteilung erzeugten nicht konformen Elemente werden ebenfalls verfeinert, bis wieder ein konformes Netz entstanden ist.

Algorithmus 1

1: while ($T_i \neq \emptyset$) do
2: Teile alle $t \in T_i$ über die längste Kante
3: T_i = Menge der nicht konformen Dreiecke

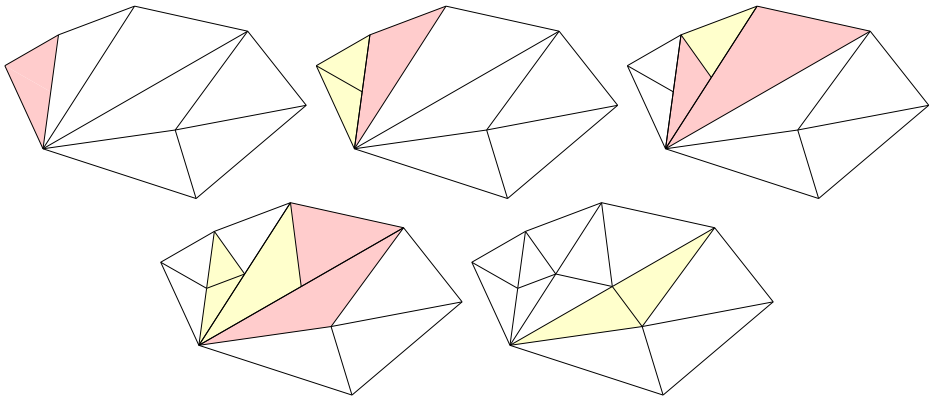
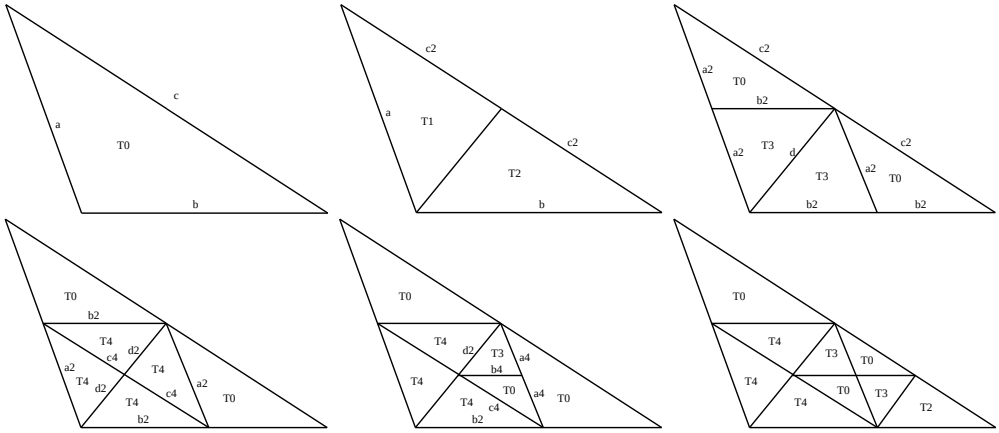


Abbildung 5.2: Bisektion über die längste Kante

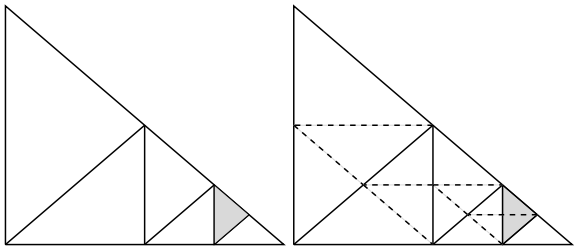
Der große Vorteil dieser Verfeinerung liegt in der Beschränkung der Verkleinerung der gegebenen Winkel. Es wird grundsätzlich der größte Winkel eines Elementes unterteilt, wobei hier nicht unbedingt eine Halbierung des Winkels vorgenommen wird. Eine mehrfache Verfeinerung eines Elementes erzeugt dabei fast immer eine Folge von vier verschiedenen Elementen, die bei Fogeunterteilungen immer wieder auftreten. Diese Folge wird genau dann erzeugt wenn die längste Kante gegenüber dem zuletzt eingefügten Knoten liegt.



Eine mehrfache Verfeinerung kann die Elemente $T0$ bis $T4$ erzeugen. Die Seitenlängen sind mit a, b, c, d bezeichnet, wobei $a2, a4, b2, b4, c2, c4$ und $d2$ sich auf die halbe bzw. auf ein viertel der Ausgangslänge bezieht.

Abbildung 5.3: Es entstehen maximal vier verschiedene Elemente

Eine Nachteil ist jedoch, dass durch die Herstellung der Konformität im schlechtesten Fall $O(n)$ Elemente verfeinert werden, falls das Netz aus n Elementen besteht.

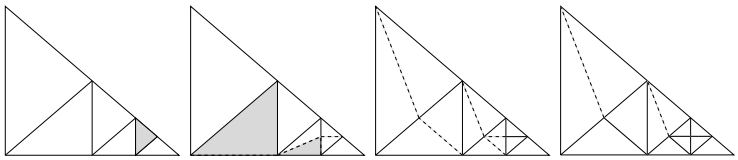


Vor der Verfeinerung besteht das Objekt aus 6 und nacher aus 15 Elementen. Das schattierte Element wurde als zu verfeinern markiert, und es wurde genau ein Element nicht verfeinert.

Abbildung 5.4: Ein worst case Beispiel

Bisektion über die längste Kante mit Abschlusskante

Durch eine Modifikation des vorherigen Algorithmus wird eine im Mittel verbesserte Laufzeit erreicht. Der Unterschied besteht darin, dass nur die markierten Elemente über die längste Kante geteilt werden müssen, und bei den folgenden Verfeinerungen eine Abschlusskante zum Knoten gegenüber der nicht konformen Seite gezogen wird, falls nur eine unterteilte Seite in dem Element existiert. Sind mindestens zwei Seiten unterteilt, so wird zunächst eine Bisektion über die längste Kante vorgenommen. Wird bei weiteren Verfeinerungsschritten dann ein Element ausgewählt, welches eine Abschlusskante besitzt, so wird diese zunächst wieder entfernt, um eine drastische Verschlechterung der Winkel zu vermeiden.



Zu verfeinernte Elemente sind schattiert, und die neuen Kanten sind gestrichelt. Im letzten Bild sind nur die Abschlusskanten gestrichelt.

Abbildung 5.5: Das Verhalten des worst case Beispiels bei Abschlusskanten

Algorithmus 2

1: while $T_i \neq \emptyset$ do
2: wähle $t \in T_i$
3: falls t eine Abschlusskante besitzt die zu den Elementen t, t' gehört
4: entferne t, t' aus T_i
5: füge $t \cup t'$ zu T hinzu
6: sonst
7: füge t zu T hinzu
8: Teile alle $t \in T$ über die längste Kante
9: T = Menge der nicht konformen Dreiecke
10: while $T \neq \emptyset$ do
11: wähle $t \in T$
12: falls t genau eine geteilte Kante e besitzt und e nicht die längste Kante von t ist,
13: teile t über die Kante e
14: markiere die neue Kante als Abschlusskante
15: sonst
16: teile t über die längste Kante
17: T = Menge der nicht konformen Dreiecke

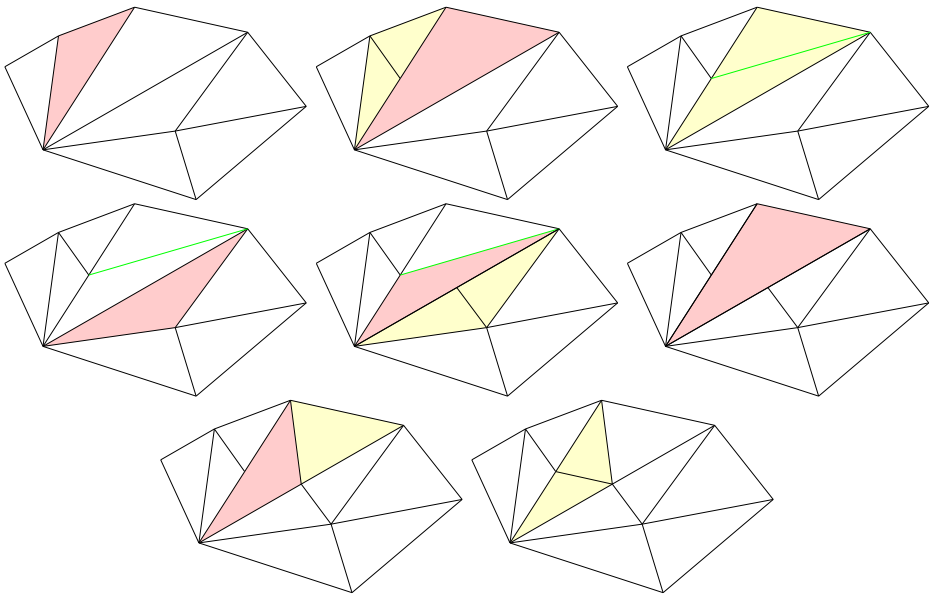


Abbildung 5.6: Bisektion über die längste Kante mit Abschlusskante

Bisektion über die längste Kante mit Rückverfolgung

M.-C. Rivara hat eine Variante ihres Bisektionsalgorithmus' vorgeschlagen, der eine einfache und schnelle Implementierung ermöglicht [61]. Ausgangspunkt ist die Überlegung, dass es für das entstehende Netz unerheblich ist, in welcher Reihenfolge eine Verfeinerung über die längste Kante erfolgt.

Für jedes zu verfeinernde Dreieck wird der *Longest Side Propagation Path* (LSPP) berechnet. Der LSPP ist die Folge $(d_0, d_1, \ldots d_n)$ von Dreiecken, so dass d_i für d_{i-1} das über die längste Kante benachbarte Element ist, für $i = 1 \ldots n$.

Der LSPP wird vom Ende zum Anfang, also rückwärts, abgebaut, d. h. das jeweils letzte Paar von Dreiecken wird über die gemeinsame längste Kante geteilt und der LSPP aktualisiert.

Algorithmus 3

```
1: while T_i ≠ ∅ do
2:     wähle ein t ∈ T_i
3:     while t ist ungeteilt do
4:         berechne/aktualisiere LSPP(t)
5:         t* sei letztes Element von LSPP(t)
6:         falls t* ist Randelement
7:             teile t*
8:         sonst
9:             teile das letzte Paar von Elementen in LSPP(T)
```

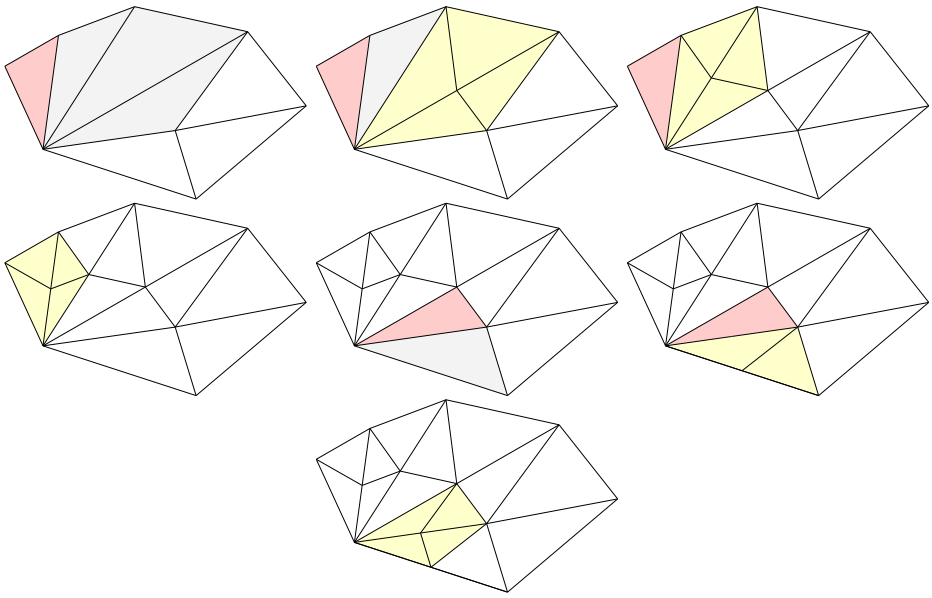


Abbildung 5.7: Bisektion über die längste Kante mit Rückverfolgung

Bisektion anhand des neuesten Knotens

Dieser Algorithmus von W. F. Mitchel ist dem Verfahren von M.-C. Rivara sehr ähnlich und erzeugt in der Tat oft die gleiche Verfeinerung [52]. Initial wird in dem Netz M_0 der Knoten gegenüber der längsten Kante als neuester Knoten markiert. Das zu verfeinernde Element wird durch eine Kante in zwei Dreiecke unterteilt, wobei der neueste Knoten und der Mittelpunkt der gegenüberliegenden Kante die Teilungskante bestimmen. Der neueste Knoten der entstandenen Dreiecke ist damit durch den neuen Steinerpunkt (Mittelpunkt) gegeben.

Algorithmus 4

1: while ($T_i \neq \emptyset$)
2: Teile alle $t \in T_i$ über die Kante gegenüber dem neuesten Knoten
3: markiere den neuesten Knoten
4: T_i = Menge der nicht konformen Dreiecke

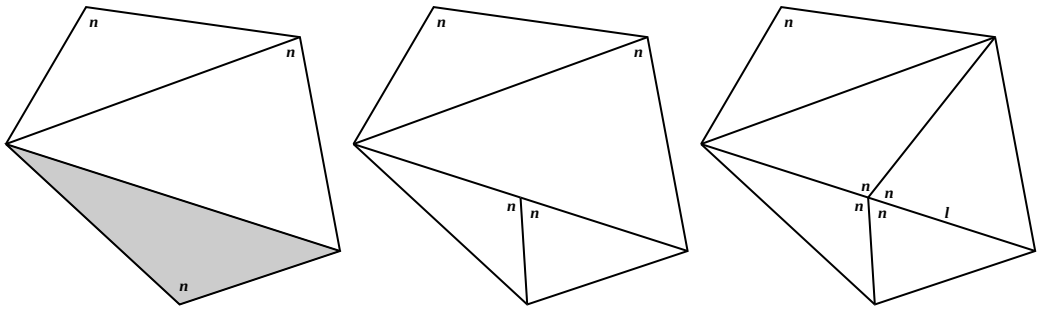


Abbildung 5.8: Bisektion anhand des neuesten Knotens

Reguläre Verfeinerung

Dieser Algorithmus stammt von E. Bank und wird in dem Softwarepacket PLTMG eingesetzt [57]. Die reguläre Verfeinerung unterteilt zu verfeinernde Elemente in vier neue Dreiecke, indem die Seiten halbiert werden. Der Mittelpunkt jeder Seite bildet einen neuen Steinerpunkt, und benachbarte Steinerpunkte werden über eine Kante verbunden. Dadurch entstehen vier identische Dreiecke die kongruent zum Ausgangsdreieck sind. Jedes nicht konforme Dreieck welches mindestens zwei geteilte Seiten besitzt wird anschliessend ebenfalls regulär verfeinert. Abschliessend werden die nicht konformen Dreiecke, welche jetzt genau eine geteilte Seite besitzen, durch eine Abschlusskante in zwei Dreiecke zerlegt. Diese Abschlusskante ist temporär ('grün') und wird bei einer erneuten Verfeinerung wieder aufgehoben.

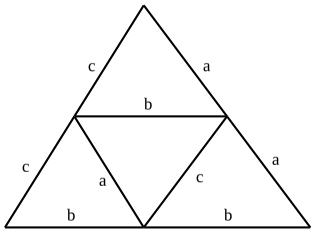


Abbildung 5.9: Die reguläre Zerlegung eines Dreiecks

Algorithmus 5

1: while ($T_i \neq \emptyset$)
2: Teile alle $t \in T_i$ in vier gleiche Dreiecke, wobei eine vorhandene Abschlusskante vorher entfernt wird.
3: T_i = alle Dreiecke mit mindestens zwei geteilten Kanten
4: T_i = Menge der nicht konformen Dreiecke
5: Teile alle $t \in T_i$ über die geteilte Kante.

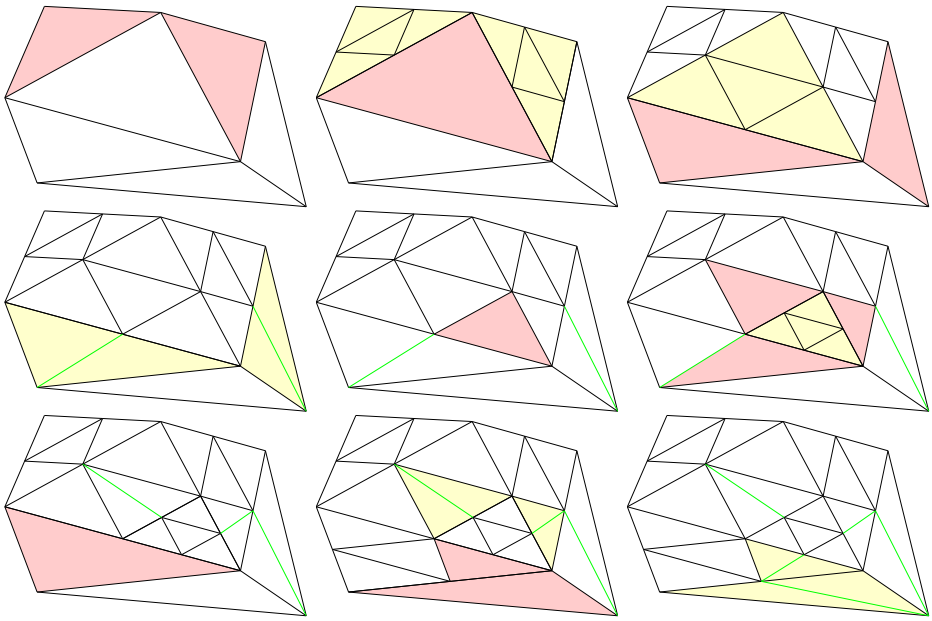


Abbildung 5.10: Die reguläre Verfeinerung

Ausnutzung der Quadtree basierten Netzgenerierung

Dieser Algorithmus kann nur in Zusammenhang mit einer Quadtree basierten Netzgenerierung verwendet werden. Die Verfeinerung baut auf der Netzgenerierung auf und benötigt die während der Netzgenerierung erzeugte Quadtree Datenstruktur. Alle Quads, in denen sich ein zu verfeinerdes Element befindet, werden zu einer weiteren Unterteilung veranlasst, womit sich automatisch eine Verfeinerung der betrachteten Elemente bei der Erzeugung der Triangulierung aus dem bestehenden Quadtree ergibt.

Algorithmus 6

1: Markiere alle Quads in denen ein $t \in T_i$ liegt
2: Teile alle markierten Quads
3: Erfülle die Quadtreebedingungen (Balanzierungsbedingung)
4: Trianguliere die geteilten Quads und dessen Nachbarn neu

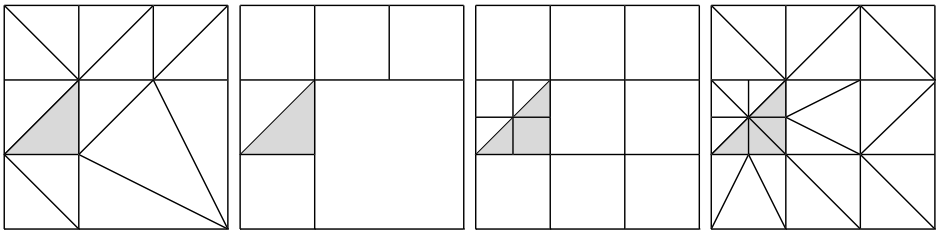


Abbildung 5.11: Die Ausnutzung der Quadtree-Struktur

5.3.2 3D Netzverfeinerung

Bisektion über die längste Kante

Dieser Algorithmus von M.-C. Rivara und C. Levin ist eine Erweiterung des 2-dimensionalen Bisektionsalgorithmus von Rivara [62]. Die längste Kante des zu verfeinernden Elementes wird halbiert indem der Mittelpunkt der Kante als Steinerpunkt aufgenommen wird. Der Tetraeder zerfällt in zwei neue Elemente durch das Einfügen der Fläche, die durch den Steinerpunkt und die beiden gegenüberliegenden Knotenpunkte definiert ist. Die dadurch endstehenden nicht konformen Tetraeder werden anschließend ebenfalls verfeinert, bis ein konformes Netz entstanden ist. Eine offene Frage ist hier immer noch, ob durch diese Verfeinerung eine begrenzte Verschlechterung der Winkel bzw. des Aspect Ratio gegeben ist oder nicht. Es wird nur eine begrenzte Verschlechterung vermutet, da alle Experimente darauf hinweisen.

Algorithmus 7

1: $while(T_i \neq \emptyset)$ do
2: Teile alle $t \in T_i$ über die längste Kante
3: T_i = Menge der nicht konformen Tetraeder

Bisektion über Verfeinerungskanten

Dieser Algorithmus von E. Bänsch und der äquivalente Algorithmus von A. Liu und B. Joe konstruieren auch eine Verfeinerung durch Bisektion, wobei nicht unbedingt die längste

Kante als Bisektionskante ausgewählt wird [45, 6]. Der Vorteil dieser Methode ist, dass die Verfeinerung eine Reihe von äquivalenten Tetraedern erzeugt. Anfangs wird die längste Kante von jeder Dreiecksfläche als Verfeinerungskante markiert. Diese Markierung ist eindeutig bzgl. jeder Dreiecksfläche unter der Voraussetzung, dass gleichlange Kanten nach einer Ordnung markiert werden. Danach gibt es in jedem Tetraeder mindestens eine Kante, die von zwei Dreiecksflächen als Verfeinerungskante markiert wurde. Diese Kanten sind die Bisektionskanten. Ein zu verfeinerndes Element wird jetzt über die Bisektionskante geteilt. In den beiden geteilten Dreiecksflächen wird die Kante als Verfeinerungskante markiert, die schon vor der Verfeinerung im Netz existierte. Die nicht geteilten Dreiecksflächen behalten ihre Markierung, und die neue Dreiecksfläche erhält eine Markierungen in Abhängigkeit von der Entstehung der Tetraeder.

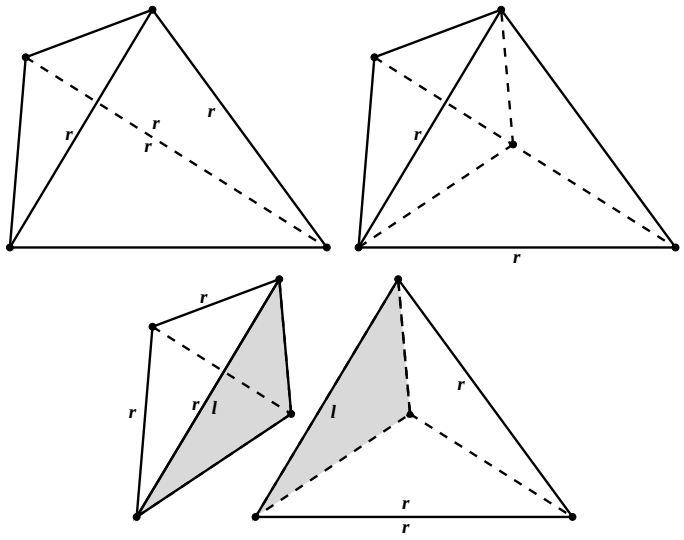


Abbildung 5.12: Teilung über die Bisektionkante

Die Verfeinerungskanten sind mit einem r markiert. Nach der Teilung über die Bisektionskante sind nur die beiden Markierungen eingetragen, die erhalten bleiben. In den neuen einzeln dargestellten Tetraedern sind wieder alle Markierungen eingetragen, wobei die Markierung l von der Entstehung der Tetraeder abhängt.

Es werden drei Typen von Tetraedern unterschieden, die roten, die schwarzen und die weißen Tetraeder.

Bezeichnung 5.3.1 roter , schwarzer, weißer Tetraeder

Sei $e = (v_0, v_1)$ die Bisektionskante des Tetraeders und $k = (v_2, v_3)$ die Kante, welche nicht mit e verbunden ist. Weiter sei $k \neq l_1 \neq l_2 \neq k$ d.h. $l_1 \in \{(v_1, v_2), (v_2, v_3)\}$ und $l_2 \in \{(v_0, v_2), (v_0, v_3)\}$ die beiden anderen Verfeinerungskanten des Tetraeders.

Falls l_1 und l_2 einen gemeinsamen Endpunkt besitzen, wird der Tetraeder als schwarzer – und andernfalls als roter Tetraeder bezeichnet. Bei allen nicht erfassten Tetraeder handelt es sich um weiße Tetraeder.

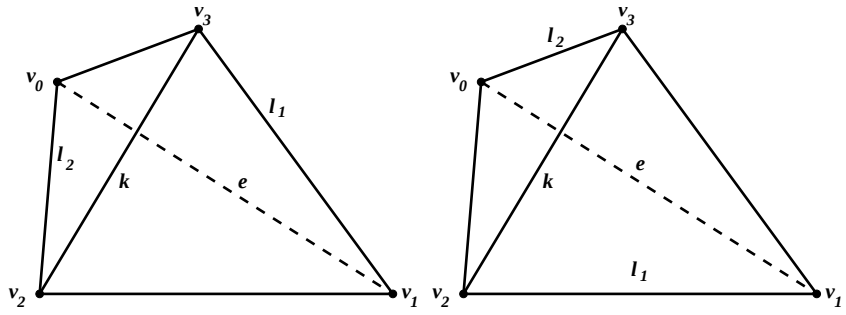


Abbildung 5.13: Rote Tetraeder

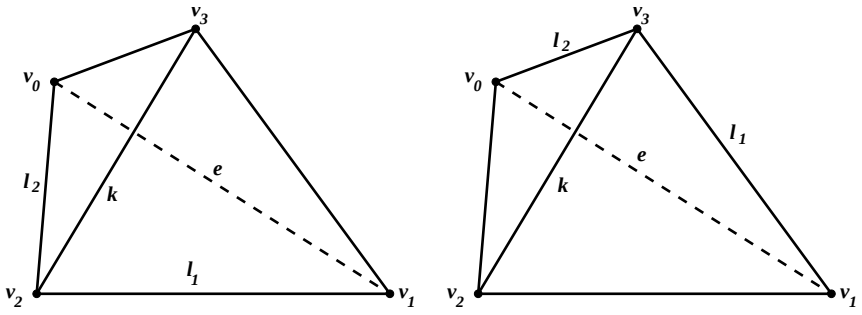


Abbildung 5.14: Schwarze Tetraeder

Sei t der zu verfeinernde Tetraeder und T der Tetraeder, durch dessen Teilung t entstanden ist. Es wird folgende Kante als Verfeinerungskante der neuen Dreieksfläche markiert, wobei v_{01} der eingefügte Steinerpunkt ist:

(1) T existiert nicht $\Rightarrow$ markiere (v_2, v_3)

(2) T ist weiß $\Rightarrow$ markiere (v_2, v_3)

(3) t ist rot $\Rightarrow$ markiere (v_2, v_3)

(4) t ist schwarz und T ist rot $\Rightarrow$ markiere (v_2, v_3)

(5) t ist schwarz und T ist schwarz $\Rightarrow$ markiere (v_2, v_{01}) oder (v_3, v_{01})

Alle auf diese Weise markierten Tetraeder sind entweder schwarz oder rot. Nach der Markierung der neuen Tetraeder werden alle nicht konformen Tetraeder verfeinert.

Algorithmus 8

1: $while(T_i \neq \emptyset)$ do
2: Teile alle $t \in T_i$ über die Bisektionskante
3: Bestimme die neuen Verfeinerungskanten
3: T_i = Menge der nicht konformen Tetraeder

Ausnutzung der Octree basierten Netzgenerierung

Dieser Algorithmus kann nur in Zusammenhang mit einer Octree basierten Netzgenerierung verwendet werden. Die Verfeinerung baut auf der Netzgenerierung auf und benötigt die während der Netzgenerierung erzeugte Octree Datenstruktur. Alle Quader, in denen sich ein zu verfeinerdes Element befindet, werden zu einer weiteren Unterteilung veranlasst, womit sich automatisch eine Verfeinerung der betrachteten Elemente bei der Erzeugung der Triangulierung aus dem bestehenden Octree ergibt.

Algorithmus 9

1: Markiere alle Quader in denen ein $t \in T_i$ liegt
2: Teile alle markierten Quader
3: Erfülle die Octreebedingungen (Balanzierungsbedingung)
4: Trianguliere die geteilten und dessen Nachbarn neu

Literaturverzeichnis

[1] **Aden J. U. and Köckler N.** (1995). Boundary-Conforming Numerical Grid Generation on Processor Networks, *Parallel Algorithmus and Applications* **5**, 251–268.

[2] **Amsden A. A. and Hirt C. W.** (1973). A Simple Scheme for Generating General Curvilinear Grids, *J. Comp. Physics* **11**, 348–359.

[3] **Anderson J. D.** (1995). *Computational Fluid Dynamics*, McGraw-Hill, New York, TLO 3108.

[4] **Arcilla A. S., Häuser J., Eiseman P. R. and Thompson J. F.** (1991). *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields,* North-Holland, New York.

[5] **Babuška I., Flaherty J. E., Henshaw W. D., Hopcroft J., Oliger J. and Tezduyar T.** (1995). *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations,* Springer, New York.

[6] **Bänsch E.** (1991). Local mesh refinement in 2 and 3 dimensions, *Impact of Computing in Science and ngineering, Vol.3*, 181-191

[7] **Bern M., Dobkin and Eppstein D.** (1992). Triangulating Polygons without Large Angles, *Proceedings 8th Computational Geometry*, 222–231, TTQ 92-0984.

[8] **Bern M. and Eppstein D.** (1992). Mesh Generation and Optimal Triangulation, in [24], 23–90.

Provably Good Mesh Generation, *Proc. 31st Symp. on Foundations of Computer Science*, 231–241, TTQ 90–0447–1.

[9] **Bern M., Eppstein D. and Gilbert J.** (1990). Provably Good Mesh Generation, *Proc. 31st Symp. on Foundations of Computer Science*, 231–241, TTQ 90–0447–1.

Linear-size Nonobtuse Triangulation of Polygons, *Proceedings 10th Computational Geometry* 221–229, TTQ 94-390.

[10] **Bern M., Mitchel S. and Ruppert J.** (1994). Linear-size Nonobtuse Triangulation of Polygons, *Proceedings 10th Computational Geometry* 221–229, TTQ 94-390.

[11] **Blacker T. D.** (1991). Paving: A New Approach to Automated Quadrilateral Mesh Generation, *International Journal For Numerical Methods in Engineering* **32**, 811–847.

[12] **Bramble J. H., Pasciak J. E. and Schatz A. H.** (1986). A Construction of Preconditioners for Elliptic Problems by Substructuring I, *Math. Comp.* **47**, 103–134.

[13] **Bramble J. H., Pasciak J. E. and Schatz A. H.** (1987). A Construction of Preconditioners for Elliptic Problems by Substructuring II, *Math. Comp.* **49**, 1–16.

[14] **Bramble J. H., Pasciak J. E. and Schatz A. H.** (1988). A Construction of Preconditioners for Elliptic Problems by Substructuring III, *Math. Comp.* **51**, 415–430.

[15] **Bramble J. H., Pasciak J. E. and Schatz A. H.** (1989). A Construction of Preconditioners for Elliptic Problems by Substructuring IV, *Math. Comp.* **53**, 1–24.

[16] **Castillo J. E.** (1988). Parameter estimation in variational grid generation. *Applied Mathematics and Computation* **28**, 155 ff.

[17] **Castillo J. E.** (1991). A discret variational grid generation method. *SIAM Journal Comp. Stat. Comp.* **12**, 454–468.

[18] **Castillo J. E.** (ed.) (1991). *Mathematical Aspects of Numerical Grid Generation,* SIAM, Philadelphia, TAY 881089.

Guaranteed-Quality Mesh Generation for Curved Surfaces, *Proc. 9th Computational Geometry*, 274-280

[19] **Chew L. P.** (1993). Guaranteed-Quality Mesh Generation for Curved Surfaces, *Proc. 9th Computational Geometry*, 274-280

[20] **Chazelle B. and Shouraboura N.** (1994). Bounds on the Size of Tetrahedralizations, *Proceedings 10th Computational Geometry*, 231–239, TTQ 94-390.

[21] **Dickerson M., Drysdale R., McElfrish S. and Welzl E.** (1994). Fast Greedy Triangulation Algorithms, *Proceedings 10th Computational Geometry*, 211–220, TTQ 94-390.

[22] **Diekmann R., Dralle U., Neugebauer F. and Römke T.** (1996). PadFEM: A Portable Parallel FEM-Tool, *Proc. HPCN '96*, Springer LNCS, to appear.

[23] **Diekmann R., Meyer D., Monien B.** (1995). Parallel Decomposition of Unstructured FEM-Meshes, *Proc. IRREGULAR '95*, Springer LNCS **980**, 199–215.

[24] **Du D.-Z. and Hwang F.** (eds.) (1992). *Computing in Euclidean Geometry*, World Scientific, Singapore, TGA 2267.

[25] **Eiseman P. R.** (1987). Adaptive Grid Generation. *Computer Meth. in Appl. Mech. and Eng.* **64**, 321–376.

[26] **Flaherty J. E., Paslow P. J., Shephard M. S. and Vasilakis J. D. (eds.)** (1989). *Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia.

[27] **Fortune S.** (1992). Voronoi Diagrams and Delaunay Triangulations, in [24], 193–233.

[28] Frey P., Borouchaki H., George P. L. (1996). Delaunay Tetrahedralization using an Advancing-Front Approach, *Proc. 5th Int. Meshing Roundtable, Pittsburgh, PA, Techn. Rep. SAND96-2301, Sandia National Lab.*, 31-46

[29] George P. L. (1991). *Automatic Mesh Generation. Applications to Finite Element Methods*, Wiley, New York, TLB 2246.

[30] George P. L. and Seveno E. (1994). The Advancing-Front Mesh Generation Method Revisited, *International Journal For Numerical Methods in Engineering* **37**, 3605–3619.

[31] Gilding B. (1988). A Numerical Grid Generation Technique, *Comp.& Fluids* **16**, 47–58.

[32] Globisch G. (1995). PARMESH - A Parallel Mesh Generator, *Parallel Computing* **21**, 509–524, 64 p 7.

[33] Hackbusch W. and Trottenberg U. (eds.) (1982). Multigrid Methods, *Lecture Notes in Mathematics* 960, Berlin, TAY 810402.

[34] Hackbusch W. (1986). *Theorie und Numerik elliptischer DGL*, Teubner Studienbücher, Stuttgart, TLB 1876.

[35] Hackbusch W. (1985). *Multi-Grid Methods and Applications*, Springer, Berlin, TLB 1729.

[36] Henshaw W. D. (1996). Automatic Grid Generation, *Acta Numerica*, 121–148.

[37] Ho-Le K. (1988). Finite Element Mesh Generation Methods: a Review and Classification, *Computer Aided Design* **20**, 27–38, 80 c 3.

[38] Holmes D. and Synder D. (1988). The Generation of Unstructured Triangular Meshes Using Delaunay Triangulation, in *Proceedings, Second International Conference on Numerical Grid Generation for Computational Fluid Mechanics*, (Sengupta S, Hauser J., Eiseman P. and Thompson J., eds), Pineridge Press, Swansea, UK.

Adaptive Refinement of Unstructured Finite-Element-Meshes, *J. of Finite Elements in Analysis and Design, to appea*

[39] Jones M. T. and Plassmann P. E. (1994). Adaptive Refinement of Unstructured Finite-Element-Meshes, *J. of Finite Elements in Analysis and Design, to appea*

[40] Jung Y. H. and Lee K. (1993). Tetrahedron-Based Octree Encoding for Automatic Mesh Generation, *Computer Aided Design* **25**, 141–153, 80 c 3.

[41] Knupp P. M.. Intrinsic Algebraic Grid Generation, Chapter 6 in [18].

[42] Knupp P. M.; Steinberg S. (1993). *The Fundamentals of Grid Generation.* CRC Press, Boca Raton.

[43] Köckler N. and Schäfer Ch. (1991). Parallele Vorkonditionierung durch Gebietszerlegung, *Proc. 2. PASA Workshop 'Parallele Systeme und Algorithmen'*, Paderborn.

[44] **Lingas A.** (1986). The Greedy and Delauney Triangulations are not Bad in the Average Case, *Information Processing Letters*, **22**, 25–31.

Quality local refinement of tetrahedra meshes based on bisection, *SIAM Journal on Scientific Computing,* **Vol. 16**, 1269-1291

[45] **Liu A. and Joe B.** (1995). Quality local refinement of tetrahedra meshes based on bisection, *SIAM Journal on Scientific Computing,* **Vol. 16**, 1269-1291

[46] **Lo S. H.** (1995). Automatic Mesh Generation over Intersecting Surfaces, *International Journal For Numerical Methods in Engineering* **38**, 943–954.

[47] **Mastin C. W. and Thompson J. F.** (1978). Elliptic Systems and Numerical Transformations, *J. Math. Anal. Appl.* **62**, 52–62.

[48] **Mavriplis D. J.** (1995). An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness, *J. Comp. Phys.* **117**, 90–101.

[49] **McCormick St. F.** (1989). *Multilevel Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia, TLB 2181.

[50] **McCormick St. F. (ed.)** (1987). *Multigrid Methods*, SIAM, Philadelphia.

[51] **Miller G. L., Teng S.-H., Thurston W., Vavasis S. A.** (1993). *Automatic Mesh Partitioning.* In: George, Gilbert, Liu (ed.): *Graph Theory and Sparse Matrix Computation.* IMA Volumes in Mathematics and its Applications, Springer Verlag, Vol. **56**, 57–84.

A comparsion of adaptive refinement techniques for elliptic problems, *ACM Transactions on Mathematical Software,* **15**, S. 326-347

[52] **Mitchell W. F.** (1989). A comparsion of adaptive refinement techniques for elliptic problems, *ACM Transactions on Mathematical Software,* **15**, S. 326-347

Quality Mesh Generation in Three Dimension, *Proc. 8th ACM Conf. on Comp. Geometry*, S. 212-221

[53] **Mitchell S. A., Vavasis S. A.** (1992). Quality Mesh Generation in Three Dimension, *Proc. 8th ACM Conf. on Comp. Geometry*, S. 212-221

[54] **Müller J. D., Roe P. L. and Deconinck H.** (1993). A Frontal Approach for Internal Node Generation in Delaunay Triangulations, *International Journal For Numerical Methods in Fluids* **17** (3), 241–256.

[55] **Neugebauer F.** (1996). *Das Projekt ParNet*, Vortrag und persönliche Mitteilungen.

[56] **Neugebauer F. and Diekmann R.** (1996). Improved Mesh Generation: Not Simple but Good, to appear.

[57] **Bank R. E.** (1994). *PLTMG Users' Guide – Edition 7.0*, SIAM, Philadelphia.

[58] **Rebay S.** (1993). Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and the Bowyer-Watson Algorithm, *J. Comp. Phys.* **106**, 125–138.

Mesh refinement processes based on the generalized bisection of simplices, *SIAM Journal of Numerical Analysis, Vol 21,* **No. 3**, 604-613

[**59**] **Rivara M. C.**(1984). Mesh refinement processes based on the generalized bisection of simplices, *SIAM Journal of Numerical Analysis, Vol 21,* **No. 3**, 604-613

[**60**] **Rivara M. C.** (1984). Mesh refinement processes based on the generalized bisection of simplices, *SIAM Journal of Numerical Analysis, Vol 21,* **No. 3**, 604-613

New Mathematical Tools and Techniques for the Refinement and/or Improvement of Unstructured Triangulations, *Proc. 5th Int. Meshing Roundtable, Pittsburgh, PA, Techn. Rep. SAND96-2301, Sandia Nat'l Lab.*, 77-86

[**61**] **Rivara M. C.** (1996). New Mathematical Tools and Techniques for the Refinement and/or Improvement of Unstructured Triangulations, *Proc. 5th Int. Meshing Roundtable, Pittsburgh, PA, Techn. Rep. SAND96-2301, Sandia Nat'l Lab.*, 77-86

A 3-d refinement algorithm suitable for adaptive and multigrid techniques, *Communications in Applied Numerical Methods, Vol. 8*, 281-290

[**62**] **Rivara M. C. and Levin C. (1992)**. A 3-d refinement algorithm suitable for adaptive and multigrid techniques, *Communications in Applied Numerical Methods, Vol. 8*, 281-290

[**63**] **Rüde U.** (1993). *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, SIAM, Philadelphia, TKY 1869.

[**64**] **Ruppert J.** (1992). A New and Simple Algorithm for Quality 2-D Mesh Generation, *TR CSD–92–694 University of California at Berkeley.*

[**65**] **Ruppert, J.**(1995). A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *J. of Algorithms* **18**, 548–585.

[**66**] **Saxena M. et. al.** (1995). Octree-Based Automatic Mesh Generation for Non-Mainfold Domains, *Engineering with Computers* **11**, 1–14, 64 e 5.

[**67**] **Schumaker L.** (1993). Computing Optimal Triangulations Using Simulated Annealing, *Computer Aided Geometric Design* **10**, 329–345, 64 c 36.

[**68**] **Shephard M. and George M.** (1991). Automatic Three-Dimensional Mesh Generation by the Finite-Octree Technique, *International Journal for Numerical Methods in Engineering* **32**, 709–747.

[**69**] **Shewchuk J. R.**(1996). Robust Adaptive Floating-Point Geometric Predicates. *Proc. 12th Annual Symp. Comp. Geometry*, ACM.

[**70**] **Shewchuk J. R.**(1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, *Proc. 1st Workshop Appl. Comp. Geometry*, ACM.

[**71**] **Shostko A. and Löhner R.** (1995). Three-Dimensional Parallel Unstructured Grid Generation, *International Journal For Numerical Methods in Engineering* **38**, 905–925.

[**72**] **Spekreijse S. P.** (1995). Elliptic Grid Generation based on Laplace Equations and Algebraic Transformations, *J. Comp. Phys.* **118**, 38–61.

[73] Spekreijse S. P., Boerstoel J. W., Vitagliano P. L. and Kuyvenhoven J. L. (1992). Domain Modeling and Grid Generation for Multi-Block Structured Grids with Application to Aerodynamic and Hydrodynamic Configurations, in *Proceedings, Software Systems for Surface Modeling and Grid Generation* (R. Smith, ed.), NASA Conference Publication 3143, 207–229.

[74] Starius G. (1977). Constructing Orthogonal Curvilinear Meshes by Solving Initial Value Problems, *Numer. Math* **28**, 25–48.

[75] Steger J. L. and Benek J. A. (1987). On the Use of Composite Grid Schemes in Computational Aerodynamics, *Computer Methods in Applied Mechanics and Engineering* **64**, 301–320.

[76] Steinberg S.; Roache P. J. (1986). Variational grid generation. *Numerical Methods for Partial Differential Equations* **2** , 71–96.

[77] Thompson J. F., Thames F. C. and Mastin C. W. (1974). Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate Systems for Fields Containing any Number of Arbitrary Two-Dimensional Bodies, *J. Comp. Physics* **15**, 299.

[78] Thompson J. F., Warsi Z. U. A. and Mastin C. W. (1985). *Numerical Grid Generation – Foundations and Applications*, North Holland, Amsterdam, TLB 1779.

[79] Tiow-Seng Tan (1994). An Optimal Bound for Conforming Quality Triangulations, *Proceedings 10th Computational Geometry*, 240–249, TTQ 94-390.

[80] Weatherhill N. P. et al. (1994). *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Pineridge Press, Swansea, UK.

[81] Wesseling P. (1992). *An Introduction to Multigrid Methods*, Wiley, Chichester.

[82] Winslow A. M. (1967). Numerical Solution of the Quasilinear Poisson Equation in a Nonuniform Triangle Mesh, *J. Comp. Physics* **2**, 149–172.

[83] Liseikin V. D. (1999). *Grid Generation Methods*, Springer, Berlin.

[84] Hoff K. E., Culver T., Keyser J., Lin M. and Manocha D. (1999). Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware, http://www.cs.unc.edu/ geom/voronoi/.