Rekonstruktion von Oberflächen aus Punktewolken

Studienarbeit im Fach Informatik

vorgelegt von

Marc Axel Johannes Wagner

geb. am 17. September 1975 in Nürnberg

angefertigt am

Institut für Informatik 9
Lehrstuhl für Graphische Datenverarbeitung
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Ulf Labsik

Beginn der Arbeit: 1. November 2000

Abgabe der Arbeit: 31. Juli 2001

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Aufbau dieser Studienarbeit

Im Folgenden wird ein kurzer Überblick über die vorliegende Ausarbeitung gegeben.

In diesem ersten Kapitel werden einige grundlegende Aspekte der Oberflächenrekonstruktion erläutert. Es wird die Problemstellung definiert, kurz auf bisherige Arbeiten in diesem Bereich eingegangen, und auf bei der Oberflächenrekonstruktion häufig auftretende Schwierigkeiten hingewiesen. Im Anschluss daran wird die Bedeutung verschiedener umgangssprachlicher Formulierungen und abkürzender Schreibweisen erläutert, die im Rahmen dieser Studienarbeit häufig verwendet werden.

Das primäre Ziel dieser Forschungsarbeit war es, einen neuen Algorithmus zur Rekonstruktion von Oberflächen aus Punktewolken zu entwickeln. In Kapitel 2 wird dieser neu entwickelte Algorithmus in einer Kurzfassung vorgestellt. Dabei werden lediglich grundlegende Ideen, keinesfalls jedoch Details beschrieben.

Der Rekonstruktionsalgorithmus besteht im Wesentlichen aus sechs voneinander unabhängigen Schritten, die nacheinander ausgeführt werden. Diese sechs Schritte werden in den Kapiteln 3 bis 8 ausführlich dargestellt.

Selbstverständlich wurde der im Rahmen dieser Arbeit entwickelte Algorithmus auch implementiert und intensiv getestet. In Kapitel 9 werden zahlreiche von der Implementierung gelieferte Ergebnisse präsentiert und diskutiert.

In Kapitel 10 wird ein Ausblick auf weitere Forschungsmöglichkeiten, diese Studienarbeit betreffend, gegeben.

Kapitel 11 schliesslich fasst die wesentlichen Punkte dieser Studienarbeit noch einmal zusammen.

Die im Rahmen dieser Forschungsarbeit erzielten Ergebnisse sind derart umfangreich, dass bei ihrer Darstellung in der vorliegenden Ausarbeitung der übliche Umfang einer Studienarbeit, in etwa vierzig Seiten, deutlich überschritten wurde. Teilalgorithmen, die aus anderen Quellen

übernommen wurden, werden daher, wenn überhaupt, nur kurz erläutert. Es wird jedoch stets auf die Originalarbeiten verwiesen. Verschiedene Beweise, die zum Verständnis dieser Arbeit nicht unbedingt erforderlich sind, wurden aus Platzgründen ebenfalls weggelassen.

Um die folgenden Ausführungen verstehen zu können, sollte der Leser gewisse Grundkenntnisse in geometrischer Modellierung, Algorithmik und Komplexitätstheorie besitzen.

1.2 Was ist Oberflächenrekonstruktion?

Aus einer ungeordneten Menge von dreidimensionalen Punkten, die einer geschlossenen Oberfläche entstammen, ohne Kenntnis dieser Oberfläche ein fehlerfreies und geschlossenes Dreiecksnetz zu konstruieren, dessen Form relativ gut der Form dieser Oberfläche entspricht, wird in dieser Studienarbeit als Oberflächenrekonstruktion bezeichnet.

Oberflächenrekonstruktion kommt hauptsächlich bei beziehungsweise vor der Darstellung real existierender, dreidimensionaler Objekte mit dem Computer zur Anwendung. Das betreffende Objekt wird dazu zuerst von einem Scanner abgetastet, der eine ungeordnete Menge von Punkten, im Folgenden häufig als Punktewolke bezeichnet, liefert. Da heute gängige Graphikkarten hauptsächlich darauf ausgelegt sind, viele Dreiecke in kurzer Zeit darzustellen, muss aus der Punktewolke eine geeignete Triangulierung erzeugt werden. Unter einer geeigneten Triangulierung wird in diesem Zusammenhang einerseits eine fehlerfreie und geschlossene Triangulierung verstanden, also ein topologisch korrektes Dreiecksnetz ohne Löcher oder sich schneidende Dreiecke, andererseits eine Triangulierung, deren Form relativ gut der Form des gescannten Objekts entspricht. Dieser Übergang von einer gegebenen Punktewolke zu einem Dreiecksnetz stellt die in dieser Arbeit intensiv betrachtete Oberflächenrekonstruktion dar.

1.3 Bisherige Arbeiten zur Oberflächenrekonstruktion

Es existieren zahlreiche Arbeiten, die sich mit dem Problem der Oberflächenrekonstruktion beschäftigen. Teilweise sind sich diese Arbeiten sehr ähnlich, teilweise verwenden sie jedoch auch völlig unterschiedliche Methoden, um aus einer Punktewolke ein Dreiecksnetz zu generieren. Eine eindeutige Klassifizierung der bisher entwickelten Algorithmen ist schwierig, da die Grenzen zwischen den verschiedenen Verfahren fliessend sind. Die im Folgenden stichpunktartig präsentierte Einteilung in Klassen entstammt [MeMu97b]. Durch sie soll dem Leser eine grobe Vorstellung vermittelt werden, auf welchen grundsätzlich verschiedenen Wegen an das Problem der Oberflächenrekonstruktion herangegangen werden kann.

1.3.1 Oberflächenrekonstruktion durch räumliche Unterteilung

Bei diesen Algorithmen wird zuerst der Teil des Raumes, in dem sich die Punktewolke befindet, in viele kleine Zellen unterteilt. Dann wird versucht, diejenigen Zellen zu selektieren, die entweder von der Oberfläche des zur Punktewolke gehörigen Objekts geschnitten

werden (oberflächenorientierte Verfahren), oder von dieser Oberfläche eingeschlossen werden (volumenorientierte Verfahren). Aus den selektierten Zellen wird schliesslich ein Dreiecksnetz erzeugt.

Beispiele für ein derartiges Vorgehen sind [AlSc96], [BaBe97] und [HoDe92].

1.3.2 Oberflächenrekonstruktion mit Hilfe einer Abstandsfunktion

Bei derartigen Verfahren wird versucht, eine Funktion zu finden, die zu jedem beliebigen Punkt im Raum eine gute Näherung für den kleinsten Abstand zu der Oberfläche des zur Punktewolke gehörigen Objekts berechnet. Alle Punkte, für die eine solche Abstandsfunktion den Wert null liefert, liegen auf der durch sie beschriebenen Oberfläche. Mit Hilfe dieser Abstandsfunktion wird dann ein Dreiecksnetz erzeugt.

Beispiele für ein derartiges Vorgehen sind [BiTs95] und [HoDe92].

1.3.3 Oberflächenrekonstruktion durch Formveränderung

Bei diesen Methoden wird jeweils mit einem geschlossenen Dreiecksnetz begonnen, das mindestens so viele Punkte besitzt wie die gegebene Punktewolke. Im Idealfall hat ein solches Dreiecksnetz bereits eine Form, die der Form des zur Punktewolke gehörigen Objekts ähnlich ist. Ist die Form des zur Punktewolke gehörigen Objekts jedoch unbekannt, kann auch mit einer beliebigen anderen Form, beispielsweise einer Kugel, begonnen werden. Um eine Formveränderung zu erzielen, werden nun verschiedene Punkte des Dreiecksnetzes verschoben, ohne jedoch die Topologie des Netzes zu verändern. Mit dem Verschieben der Punkte wird solange fortgefahren, bis das Dreiecksnetz gut genug die Oberfläche des zur Punktewolke gehörigen Objekts approximiert.

Ein Beispiel für ein derartiges Vorgehen ist [AlSc96].

1.3.4 Inkrementelle Oberflächenrekonstruktion

Bei diesen Algorithmen wird mit einer einzigen Kante oder einem einzigen Dreieck begonnen. Schrittweise werden dann weitere Kanten und Dreiecke in die jeweils aktuelle Dreiecksnetz-Konfiguration eingefügt. Dieses Hinzufügen von Kanten und Dreiecken wird abgebrochen, wenn keine weiteren Kanten oder Dreiecke mehr eingefügt werden können, die das Dreiecksnetz zu einer besseren Approximation des zur Punktewolke gehörigen Objekts machen.

Beispiele für ein derartiges Vorgehen sind [BeMi99], [GoKr00], [Menc95] und [MeMu97a].

1.4 Probleme bei der Oberflächenrekonstruktion

Das durch ein Rekonstruktionsverfahren erzeugte Dreiecksnetz soll einerseits fehlerfrei und geschlossen sein, andererseits eine Form aufweisen, die relativ gut der Form des zur gegebenen Punktewolke gehörigen Objekts entspricht. Dieses Ziel zu erreichen bereitet vor allem dann Schwierigkeiten, wenn die Punktdichte der Punktewolke in kritischen Bereichen zu gering ist. Solche kritischen Bereiche sind einerseits Bereiche, in denen das zur Punktewolke gehörige Objekt eine sehr starke Krümmung aufweist, andererseits Bereiche, in denen das zur Punktewolke gehörige Objekt sehr dünn ist.

1.4.1 Probleme in Bereichen starker Krümmung

Ist die Punktdichte der gegebenen Punktewolke in einem bestimmten Bereich klein im Vergleich zur maximalen Krümmung des zu dieser Punktewolke gehörigen Objekts in diesem Bereich (dieser Fall tritt vor allem an scharfen Kanten und Spitzen auf, da dort die Krümmung unendlich ist), liegen nicht benachbarte Punkte häufig näher zusammen als benachbarte Punkte. Als benachbarte beziehungsweise nicht benachbarte Punkte werden in diesem Zusammenhang Punkte bezeichnet, die auf der gescannten Oberfläche direkt beziehungsweise nicht direkt nebeneinander liegen. Da nahezu alle Rekonstruktionsalgorithmen bei der Suche nach einer geeigneten Triangulierung auf irgendeine Art und Weise den Abstand zwischen den Punkten der gegebenen Punktewolke verwenden, entsprechen die Formen der erzeugten Dreiecksnetze in solchen Bereichen starker Krümmung häufig nicht mehr der Form des zu dieser Punktewolke gehörigen Objekts (Abbildung 1.1; die Abbildung zeigt einen Schnitt durch ein dreidimensionales Objekt; das Innere ist dabei grau gefärbt). Bei vielen Algorithmen weisen die rekonstruierten Dreiecksnetze in derartigen Bereichen sogar Löcher und Fehler auf.

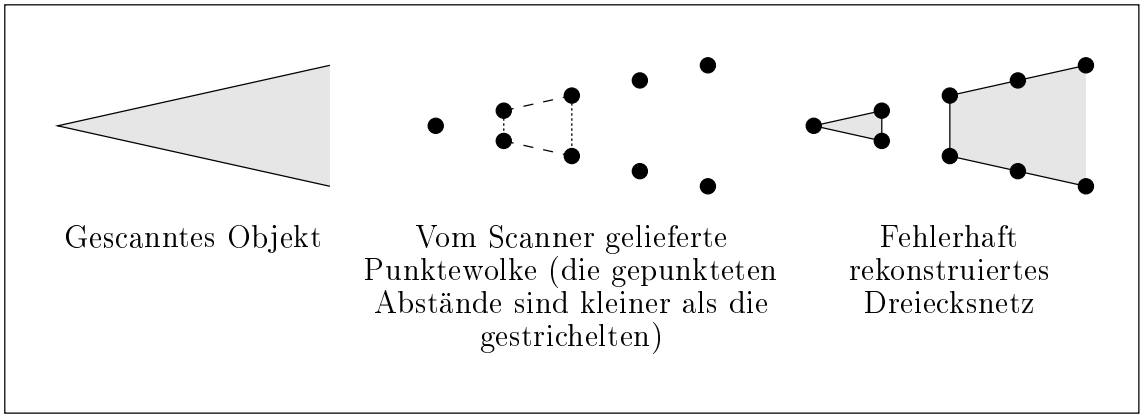


Abbildung 1.1: Fehlerhafte Rekonstruktion an einer scharfen Kante

1.4.2 Probleme bei sehr dünnen Objekten

Ist die Punktdichte der gegebenen Punktewolke in einem bestimmten Bereich klein im Vergleich zum Kehrwert des kleinsten Durchmessers des zu dieser Punktewolke gehörigen Objekts in diesem Bereich (dieser Fall tritt vor allem bei sehr dünnen Objekten auf), liegen nicht benachbarte Punkte häufig näher zusammen als benachbarte Punkte. Dies führt, wie bereits in Unterabschnitt 1.4.1 erläutert, häufig dazu, dass Rekonstruktionsalgorithmen Dreiecksnetze erzeugen, die unvollständig und fehlerhaft sind, oder deren Formen nicht mehr der Form des zur gegebenen Punktewolke gehörigen Objekts entsprechen (Abbildung 1.2; die Abbildung zeigt einen Schnitt durch ein dreidimensionales Objekt; das Innere ist dabei grau gefärbt).

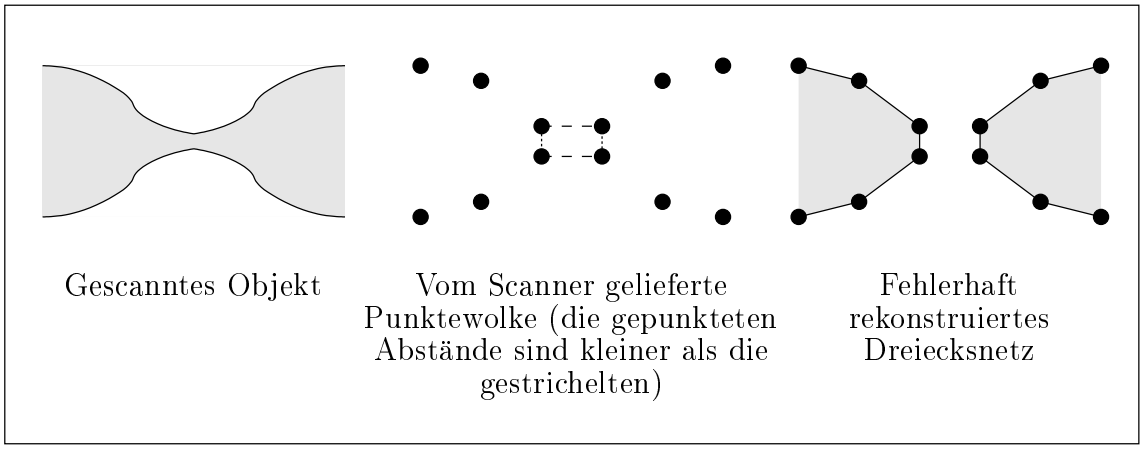


Abbildung 1.2: Fehlerhafte Rekonstruktion an einer sehr dünnen Stelle

1.4.3 Lösungsmöglichkeiten

Eine Möglichkeit, den eben angesprochenen Problemen zu begegnen, besteht darin, Abtasttheoreme einzuführen, die problematische Punktewolken von vornherein ausschliessen. Solche Abtasttheoreme geben meistens eine mindestens notwendige Abtastrate in Abhängigkeit von der maximalen Krümmung und dem minimalen Durchmesser des abzutastenden Objekts für jeden Punkt der Oberfläche dieses Objekts vor. Die meisten Arbeiten zur Oberflächenrekonstruktion verzichten auf Abtasttheoreme, da diese einerseits mathematisch schwierig zu formulieren und andererseits in der Praxis wenig hilfreich sind, letzteres deshalb, weil Scanner in der Regel mit konstanter Abtastrate arbeiten, und die Krümmungen und minimalen Durchmesser der zu scannenden Objekte meistens ohnehin nicht bekannt sind. Ein Beispiel für eine Arbeit, deren Rekonstruktionsverfahren auf einem derartigen Abtasttheorem basiert, ist [GoKr00].

Eine andere Lösungsmöglichkeit sind Algorithmen zur Nachbearbeitung von unvollständigen oder fehlerhaften, rekonstruierten Dreiecksnetzen. Solche Algorithmen können zwar eine fehlerfreie und geschlossene Triangulierung garantieren, werden aber sicher niemals für jede

Punktewolke eine Oberfläche liefern, deren Form der Form des zu dieser Punktewolke gehörigen Objekts entspricht. Anders ausgedrückt kann für jedes Rekonstruktionsverfahren mit Nachbearbeitungsalgorithmus und jedes Objekt eine zu diesem Objekt gehörige Punktewolke konstruiert werden, bei der unerwünschte Ergebnisse produziert werden.

Erstaunlicherweise existiert zur Nachbearbeitung von Dreiecksnetzen kaum Literatur, obwohl (oder vielleicht gerade weil?) es sich dabei ganz sicher nicht um ein einfaches Problem handelt. Die meisten Arbeiten zur Oberflächenrekonstruktion beschränken sich auf „gutartige“ Punktewolken, also solche, bei der oben genannte Schwierigkeiten nicht auftreten, ohne jedoch genau zu spezifizieren, welche Punktewolken „gutartig“ sind. Einige Arbeiten behandeln bei der Rekonstruktion auftretende Schwierigkeiten in wenigen Sätzen, was der Komplexität des Problems in keiner Weise gerecht wird. Leider wird durch solche Verfahren auch nur der einfachere Teil der in Dreiecksnetzen möglichen Fehler beseitigt.

Eines der Hauptanliegen dieser Arbeit ist es, einen neu entwickelten Algorithmus zur Nachbearbeitung von unvollständigen und fehlerhaften Triangulierungen zu präsentieren, der in jedem Fall ein fehlerfreies und geschlossenes Dreiecksnetz liefert, dessen Form in den meisten Fällen auch relativ gut der Form des zur gegebenen Punktewolke gehörigen Objekts entspricht.

1.5 Terminologie und Konventionen

1.5.1 Bedeutung häufig verwendeter Formulierungen

Um Missverständnissen in den folgenden Kapiteln vorzubeugen, soll hier kurz auf die Bedeutung verschiedener, häufig verwendeter, umgangssprachlicher Ausdrücke und Formulierungen eingegangen werden.

Ein **Dreiecksnetz** oder eine **Triangulierung** besteht aus Punkten, Kanten und Dreiecken. Die Punkte sind die Punkte der gegebenen Punktewolke. Kanten kommen nur in Form von Seiten eines oder mehrerer Dreiecke vor, das heisst eine Kante wird niemals isoliert ohne Dreiecke existieren.

Zu den **Kanten eines Punktes** gehören alle Kanten, für die dieser Punkt Endpunkt ist. Die **Dreiecke eines Punktes** sind diejenigen Dreiecke, für die dieser Punkt Eckpunkt ist. Die **Dreiecke einer Kante** sind diejenigen Dreiecke, die diese Kante als eine ihrer drei Seiten haben.

Ein **topologisch korrektes Dreiecksnetz** ist ein Dreiecksnetz, in dem jede Kante höchstens zwei Dreiecke hat, und in dem es keinen geschlossenen Dreiecksfächer gibt, an dessen Spitze weitere Dreiecke hängen. Ein **fehlerfreies Dreiecksnetz** ist ein topologisch korrektes Dreiecksnetz, in dem es keine sich schneidenden Dreiecke gibt.

Eine **offene Kante** ist eine Kante mit nur einem Dreieck. Ein **Loch** ist ein Zyklus von offenen Kanten, nicht zu verwechseln mit einem Henkel, wie er bei Objekten mit *Genus* ≥ 1

vorkommt, beispielsweise einem Torus. Ein **geschlossenes Dreiecksnetz** ist ein Dreiecksnetz ohne offene Kanten und damit auch ohne Löcher.

1.5.2 Bedeutung häufig verwendeter Zeichen

Verschiedene Grössen und Objekte werden vor allem in Formeln und Abbildungen durch bestimmte Zeichen dargestellt beziehungsweise abgekürzt. Bei deren Benennung wurde eine einheitliche Konvention befolgt. Die Bedeutung der einzelnen Zeichen kann folgender Tabelle entnommen werden. Bei Verwendung dieser Zeichen im weiteren Verlauf dieser Arbeit wird deren Bedeutung meist nicht mehr ausdrücklich erwähnt.

P	Die gegebene Punktewolke. $P = \{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$.
n	Die Anzahl der Punkte der gegebenen Punktewolke.
$\mathcal{T}$	Ein aus Punkten, Kanten und Dreiecken bestehendes Dreiecksnetz.
$\mathbf{v_j}$	Sowohl ein Punkt der gegebenen Punktewolke P als auch jedes zu P gehörigen Dreiecksnetzes $\mathcal{T}$.
$e(\mathbf{v_j}, \mathbf{v_k})$	Eine Kante eines zu P gehörigen Dreiecksnetzes $\mathcal{T}$ mit Endpunkten $\mathbf{v_j}$ und $\mathbf{v_k}$.
$\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$	Ein Dreieck eines zu P gehörigen Dreiecksnetzes $\mathcal{T}$ mit Eckpunkten $\mathbf{v_j}$, $\mathbf{v_k}$ und $\mathbf{v_l}$.
$N_{j,k}$	Die Menge der k nähesten Nachbarn des Punktes $\mathbf{v_j}$ der gegebenen Punktewolke P.
$T_{xxx}(\ldots)$	Laufzeitverhalten des (Teil-)Algorithmus *xxx*. Die Angabe erfolgt in der in der Komplexitätstheorie üblichen O-Notation.
$\overline{T}_{xxx}(\ldots)$	Mittleres Laufzeitverhalten des (Teil-)Algorithmus *xxx*. Die Angabe erfolgt in der in der Komplexitätstheorie üblichen O-Notation.

Tabelle 1.1: Bedeutung häufig verwendeter Zeichen

Kapitel 2

Der entwickelte Algorithmus im Überblick

In diesem Kapitel werden die wichtigsten Eigenschaften des entwickelten Rekonstruktionsalgorithmus, Eingabe, Ausgabe und Laufzeitverhalten, beschrieben. Ausserdem werden die Funktionsweisen der einzelnen Schritte des Algorithmus kurz erläutert. Dabei werden lediglich grundlegende Ideen und Prinzipien präsentiert. Detaillierte Ausführungen diesbezüglich sind in den Kapiteln 3 bis 8 zu finden.

2.1 Eingabe und Ausgabe des Algorithmus

Der im Rahmen dieser Studienarbeit entwickelte Algorithmus zur Oberflächenrekonstruktion erzeugt zu jeder Punktewolke $P = \{\mathbf{v_1}, \ldots \mathbf{v_n}\}$ mit den Eigenschaften

- Die Punkte der Punktewolke P sind dreidimensional, das heisst $\mathbf{v_j} \in P \ \rightarrow \ \mathbf{v_j} \in \mathbb{R}^3$.
- Die Punktewolke P enthält mindestens dreizehn Punkte[1], das heisst $n \geq 13$.

ein Dreiecksnetz $\mathcal{T}$ mit folgenden Eigenschaften.

- Die Eckpunkte aller Dreiecke des Dreiecksnetzes $\mathcal{T}$ entstammen der Punktewolke P, das heisst $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l}) \in \mathcal{T} \ \rightarrow \ (\mathbf{v_j} \in P \ \wedge \ \mathbf{v_k} \in P \ \wedge \ \mathbf{v_l} \in P)$.
- Das Dreiecksnetz $\mathcal{T}$ weist keine topologischen Fehler auf.
- Das Dreiecksnetz $\mathcal{T}$ besitzt keine sich schneidenden Dreiecke.
- Das Dreiecksnetz $\mathcal{T}$ ist geschlossen.

[1] Diese Untergrenze ist für die Praxis wohl relativ uninteressant und wurde nur der Vollständigkeit halber angegeben. Gängige Punktewolken bestehen aus etwa 1000 bis 100000 Punkten.

- Das Dreiecksnetz $\mathcal{T}$ besitzt kaum unnötige Zacken und Scharten.
- Die Form des Dreiecksnetzes $\mathcal{T}$ entspricht in den meisten Fällen relativ gut der Form des zur Punktewolke P gehörigen Objekts.

Die beiden letzten Aussagen über das erzeugte Dreiecksnetz sind unpräzise und subjektiv. Präzise und objektive Aussagen über diese Eigenschaften bei von Rekonstruktionsalgorithmen erzeugten Dreiecksnetzen zu machen ist jedoch nur dann möglich, wenn die zulässigen Eingabe-Punktewolken erheblichen Einschränkungen unterworfen werden. Im hier vorliegenden Fall von (fast) beliebigen Punktewolken sind derartige Aussagen also unmöglich.

Für nahezu alle Punktewolken P weist das erzeugte Dreiecksnetz $\mathcal{T}$ zusätzlich noch folgende Eigenschaft auf.

- Alle Punkte der Punktewolke P sind Eckpunkte von Dreiecken des Dreiecksnetzes $\mathcal{T}$, das heisst $\mathbf{v_j} \in P \ \rightarrow \ (\exists \triangle (\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l}) \ . \ \triangle (\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l}) \in \mathcal{T})$.

Fälle, bei denen die erzeugten Dreiecksnetze diese Eigenschaft nicht erfüllen, wurden während der vielen Testläufe im Rahmen dieser Studienarbeit nie beobachtet. Punktewolken, bei denen solche Fälle auftreten, sind extrem theoretische Konstrukte und daher für die Praxis ohnehin weniger interessant. Nähere Informationen dazu sind in Abschnitt 7.2 zu finden.

2.2 Die sechs Schritte des Algorithmus

Der in dieser Studienarbeit entwickelte Rekonstruktionsalgorithmus besteht im Wesentlichen aus sechs voneinander unabhängigen Schritten, die nacheinander ausgeführt werden. Diese sechs Schritte werden im Folgenden kurz beschrieben.

2.2.1 Erzeugen einer initialen Triangulierung (*Schritt 1*)

Bei diesem Schritt geht es darum, zu einer gegebenen Punktewolke P eine erste Triangulierung zu erzeugen. Einerseits sollte diese Triangulierung einem fehlerfreien und geschlossenen Dreiecksnetz bereits relativ nahe kommen, andererseits sollte die Form der erzeugten Triangulierung möglichst gut der Form des zu P gehörigen Objekts entsprechen.

2.2.2 Entfernen von schlechten Dreiecken (*Schritt 2*)

Das in *Schritt 1* gewonnene Dreiecksnetz besitzt in vielen Fällen Dreiecke, die sich schneiden, die die Topologie des Netzes verletzen, oder die die Erweiterung der Triangulierung zu einem geschlossenen Dreiecksnetz verhindern. Bei diesem Schritt geht es darum, möglichst viele solcher Dreiecke zu entdecken und zu entfernen.

2.2.3 Triangulieren einfacher Löcher (*Schritt 3*)

Bei diesem Schritt wird versucht, lediglich durch Einfügen neuer Dreiecke möglichst viele der vorhandenen Löcher zu schliessen. Häufig existieren jedoch auch Löcher, die nur dann trianguliert werden können, wenn vorher Dreiecke aus dem bestehenden Netz entfernt werden. Solche Löcher können durch das in diesem Schritt verwendete Verfahren nicht geschlossen werden.

2.2.4 Triangulieren komplexer Löcher (*Schritt 4*)

Bei diesem Schritt geht es darum, alle noch vorhandenen Löcher, also diejenigen, die in *Schritt 3* nicht trianguliert werden konnten, zu schliessen. Dazu werden sowohl störende Dreiecke aus dem Netz entfernt, als auch neue Dreiecke in das Netz eingefügt. Dieses Einfügen und Entfernen geschieht mit Hilfe eines zufallsgesteuerten Optimierungsverfahrens, bekannt unter dem Namen *Simulated-Annealing*.

2.2.5 Einfügen isolierter Punkte (*Schritt 5*)

Bei diesem Schritt wird versucht, alle Punkte der gegebenen Punktewolke, die nicht Eckpunkte eines oder mehrerer Dreiecke sind, in das Dreiecksnetz zu integrieren. Um einen solchen isolierten Punkt in das bestehende Dreiecksnetz einzufügen, wird in diesem ein geeignetes Loch erzeugt. Dieses Loch wird dann so trianguliert, dass der isolierte Punkt im Netz enthalten ist.

2.2.6 Glätten des erzeugten Dreiecksnetzes (*Schritt 6*)

Bei diesem Schritt geht es darum, das erzeugte Dreiecksnetz zu glätten. Einerseits werden unnötige Zacken und Scharten aus dem Netz entfernt, andererseits werden Kanten deutlicher herausgearbeitet. Das Dreiecksnetz soll dadurch ein gefälligeres Aussehen erhalten. Die Anzahl und die Lage der Punkte der gegebenen Punktewolke bleiben dabei unverändert.

2.3 Laufzeitverhalten des Algorithmus

Eine präzise Aussage über das Laufzeitverhalten des vollständigen Rekonstruktionsalgorithmus in Abhängigkeit von n, der Anzahl der Punkte der gegebenen Punktewolke, ist wegen der zufallsgesteuerten Optimierung in *Schritt 4* unmöglich. Erschwerend kommt hinzu, dass die Laufzeitverhalten der meisten Schritte von Faktoren beeinflusst werden, die nicht ausschliesslich von n abhängen. Diese Laufzeitverhalten können teilweise nur so grob abgeschätzt werden, dass die entsprechenden Aussagen für die Praxis keinen Wert besitzen. Detaillierte Betrachtungen der einzelnen Laufzeitverhalten sind in den Abschnitten *Laufzeitverhalten* der Kapitel 3 bis 8 zu finden.

Zahlreiche Experimente mit verschiedensten Punktewolken im Rahmen dieser Studienarbeit deuten darauf hin, dass das Laufzeitverhalten des vollständigen Rekonstruktionsalgorithmus für die meisten Punktewolken besser als quadratisch in n ist. Das mittlere Laufzeitverhalten des Algorithmus scheint für sehr viele Punktewolken in etwa dem Laufzeitverhalten von *Schritt 1* zu entsprechen. Diese Beobachtungen geben Grund zu der Annahme, dass

$$\overline{T}_{Oberflächenrekonstruktion}(n) \quad \approx \quad O(n^{\frac{5}{3}}) \tag{2.1}$$

gilt.

Kapitel 3

Erzeugen einer initialen Triangulierung (*Schritt 1*)

Bei diesem Schritt geht es darum, zu einer gegebenen Punktewolke P eine erste Triangulierung zu erzeugen. Einerseits sollte diese Triangulierung einem fehlerfreien und geschlossenen Dreiecksnetz bereits relativ nahe kommen, andererseits sollte die Form der erzeugten Triangulierung möglichst gut der Form des zu P gehörigen Objekts entsprechen.

3.1 Das Verfahren von *Gopi*, *Krishnan* und *Silva*

Das in diesem Abschnitt präsentierte Verfahren entstammt [GoKr00].

In einem geschlossenen Dreiecksnetz ist jedem Punkt ein geschlossener Dreiecksfächer zugeordnet, bestehend aus allen Dreiecken, die diesen Punkt als Eckpunkt haben. In diesem Verfahren wird versucht, durch Konstruktion von jeweils einem Dreiecksfächer für jeden Punkt $\mathbf{v_j} \in P$ eine möglichst gute initiale Triangulierung zu erzeugen. Die zwölf zu $\mathbf{v_j}$ nächstgelegenen Punkte in P bilden die Menge der möglichen Eckpunkte für Dreiecke des Dreiecksfächers von $\mathbf{v_j}$. Diese zwölf Punkte werden durch Rotationen in die geschätzte Tangentialebene am Punkt $\mathbf{v_j}$ projiziert. Eine in dieser Ebene stattfindende *Delaunay*-Triangulierung dieser zwölf projizierten Punkte und des Punktes $\mathbf{v_j}$ liefert den gesuchten Dreiecksfächer von $\mathbf{v_j}$.

3.1.1 Ermitteln der zwölf nähesten Nachbarn

Definition 3.1 *Die Menge $N_{j,k}$ der k nähesten Nachbarn des Punktes $\mathbf{v_j}$ der gegebenen Punktewolke P wird von den k zu $\mathbf{v_j}$ nächstgelegenen Punkten der Menge $P - \{\mathbf{v_j}\}$ gebildet.*

Es existieren zahlreiche Algorithmen zur Nähesten-Nachbar-Suche, die sich anhand ihrer Funktionsweisen und Laufzeitverhalten stark unterscheiden. In dieser Studienarbeit wurde zum Ermitteln der zwölf nähesten Nachbarn eines jeden Punktes in P ein von *J. H. Fried-*

man, *F. Baskett* und *L. J. Shustek* entwickeltes Verfahren verwendet, das einerseits einfach zu implementieren ist, andererseits ein relativ gutes Laufzeitverhalten aufweist. Im Folgenden wird die Funktionsweise dieser Methode grob beschrieben. Details können in [FrBa75] nachgelesen werden.

Die Punkte der Punktewolke werden einmalig entlang jeder der drei Koordinatenachsen sortiert. Sei nun $\mathbf{v_j}$ der Punkt, zu dem die Menge der k nähesten Nachbarn ermittelt werden soll, und die x_i-Achse diejenige Koordinatenachse, die in der Umgebung des Punktes $\mathbf{v_j}$ die geringste Punktdichte aufweist. Als mögliche Kandidaten für die Menge der k nähesten Nachbarn des Punktes $\mathbf{v_j}$ werden nun nacheinander die Punkte mit den kleinsten Abständen zu $\mathbf{v_j}$ entlang der x_i-Achse betrachtet. Die ersten k Punkte werden direkt in die zu Beginn leere Menge $N_{j,k}$ aufgenommen. Sei danach $\mathbf{v_l}$ der jeweils betrachtete Punkt und $\mathbf{v_m}$ der Punkt mit dem grössten dreidimensionalen Abstand zu $\mathbf{v_j}$ in der Menge $N_{j,k}$. Ist der dreidimensionale Abstand von $\mathbf{v_l}$ zu $\mathbf{v_j}$ kleiner als der dreidimensionale Abstand von $\mathbf{v_m}$ zu $\mathbf{v_j}$, wird $\mathbf{v_l}$ in $N_{j,k}$ eingefügt und $\mathbf{v_m}$ aus $N_{j,k}$ entfernt. Ist der eindimensionale Abstand von $\mathbf{v_l}$ zu $\mathbf{v_j}$ entlang der x_i-Achse grösser als der dreidimensionale Abstand von $\mathbf{v_m}$ zu $\mathbf{v_j}$, wird das Verfahren abgebrochen, da $\mathbf{v_l}$ und alle noch zu betrachtenden Punkte mit Sicherheit grössere Abstände zu $\mathbf{v_j}$ besitzen als jeder der Punkte der Menge $N_{j,k}$. Die Menge $N_{j,k}$ enthält dann also genau die k nähesten Nachbarn von $\mathbf{v_j}$.

Wurden die Punkte der Punktewolke bereits entlang der drei Koordinatenachsen sortiert, beträgt das mittlere Laufzeitverhalten zum Ermitteln der zwölf nähesten Nachbarn eines Punktes

$$\overline{T}_{\textit{Zwölf nähesten Nachbarn, ein Punkt}}(n) \quad = \quad O(n^{\frac{2}{3}}) \quad . \tag{3.1}$$

Das mittlere Laufzeitverhalten zum Ermitteln der zwölf nähesten Nachbarn für alle Punkte einschliesslich des Sortierens der Punkte entlang der drei Koordinatenachsen beträgt demnach

$$\overline{T}_{\textit{Zwölf nähesten Nachbarn, alle Punkte}}(n) \quad = \quad O(n^{\frac{5}{3}}) \quad . \tag{3.2}$$

3.1.2 Schätzen der Tangentialebenen

Das Verfahren zum Schätzen der Tangentialebene des zur gegebenen Punktewolke P gehörigen Objekts an einem Punkt $\mathbf{v_j} \in P$ entstammt [HoDe92].

Als Tangentialebene an einem Punkt $\mathbf{v_j} \in P$ wird diejenige Ebene gewählt, die $\mathbf{v_j}$ enthält, und die den kleinsten Abstand zu den zwölf nähesten Nachbarn $N_{j,12}$ von $\mathbf{v_j}$ im Sinne kleinster Fehlerquadrate aufweist. In zahlreichen Experimenten mit verschiedenen Punktewolken hat sich gezeigt, dass die zwölf nähesten Nachbarn eines Punktes beim Ermitteln der Tangentialebene an diesem Punkt in den meisten Fällen eine recht gute Wahl sind. Sowohl mehr als auch weniger Punkte zu betrachten führt in der Regel zu immer stärkeren Abweichungen von der tatsächlichen Tangentialebene des zu P gehörigen Objekts an der entsprechenden Stelle.

Als Normale $\mathbf{n_j}$ der Tangentialebene am Punkt $\mathbf{v_j}$ wird der zum kleinsten Eigenwert der Matrix

$$\mathbf{M_j} = \sum_{\mathbf{v} \in N_{j,12}} (\mathbf{v} - \mathbf{v_j}) \otimes (\mathbf{v} - \mathbf{v_j}) \tag{3.3}$$

gehörige Eigenvektor gewählt. Zum Berechnen der Eigenwerte und Eigenvektoren der symmetrischen Matrix $\mathbf{M_j}$ wurde in dieser Studienarbeit das *Jacobi*-Verfahren verwendet, welches unter anderem in [PrTe97] ausführlich dargestellt wird. Die Tangentialebene am Punkt $\mathbf{v_j}$ wird durch $\mathbf{v_j}$ und $\mathbf{n_j}$ vollständig beschrieben.

Das Laufzeitverhalten zum Schätzen einer Tangentialebene ist unabhängig von der Grösse der Punktewolke. Es gilt daher

$$T_{\textit{Tangentialebene, ein Punkt}}(n) = O(1) \quad . \tag{3.4}$$

Das Laufzeitverhalten zum Ermitteln der Tangentialebenen an allen Punkten der gegebenen Punktewolke beträgt demnach

$$T_{\textit{Tangentialebene, alle Punkte}}(n) = O(n) \quad . \tag{3.5}$$

3.1.3 Projektion der zwölf nähesten Nachbarn in die Tangentialebenen

Um den Dreiecksfächer eines Punktes $\mathbf{v_j} \in P$ zu berechnen, werden zuerst die zwölf nähesten Nachbarn von $\mathbf{v_j}$ durch Rotationen in die Tangentialebene E_j von $\mathbf{v_j}$ projiziert. Die Rotation eines Punktes $\mathbf{v_k} \in N_{j,12}$ in die Tangentialebene E_j findet in der durch den Punkt $\mathbf{v_j}$ und die beiden Vektoren $\mathbf{n_j}$ und $\mathbf{v_k} - \mathbf{v_j}$ festgelegten Ebene statt. Dabei wird um den Punkt $\mathbf{v_j}$ rotiert. Die Rotationsrichtung (also im oder gegen den Uhrzeigersinn) ist so zu wählen, dass der Winkel, um den rotiert wird, der kleinere der beiden möglichen ist, dass also in Richtung $\mathbf{n_j}$ betrachtet die Projektion von $\mathbf{v_k} - \mathbf{v_j}$ in die gleiche Richtung zeigt wie $\mathbf{v_k} - \mathbf{v_j}$ (Abbildung 3.1).

Seien die beiden Vektoren $\mathbf{e_{j,x}}$ und $\mathbf{e_{j,y}}$ eine beliebige orthonormale Basis der Tangentialebene E_j. Die beiden Komponenten x_k und y_k des in die Ebene E_j rotierten Punktes $\mathbf{v_k} \in N_{j,12}$ in dieser neuen Basis können mit den Formeln

$$\tilde{x_k} = \langle \mathbf{e_{j,x}}, \mathbf{v_k} - \mathbf{v_j} \rangle \tag{3.6}$$

$$\tilde{y_k} = \langle \mathbf{e_{j,y}}, \mathbf{v_k} - \mathbf{v_j} \rangle \tag{3.7}$$

$$x_k = \frac{\|\mathbf{v_k} - \mathbf{v_j}\|}{\sqrt{\tilde{x_k}^2 + \tilde{y_k}^2}} \cdot \tilde{x_k} \tag{3.8}$$

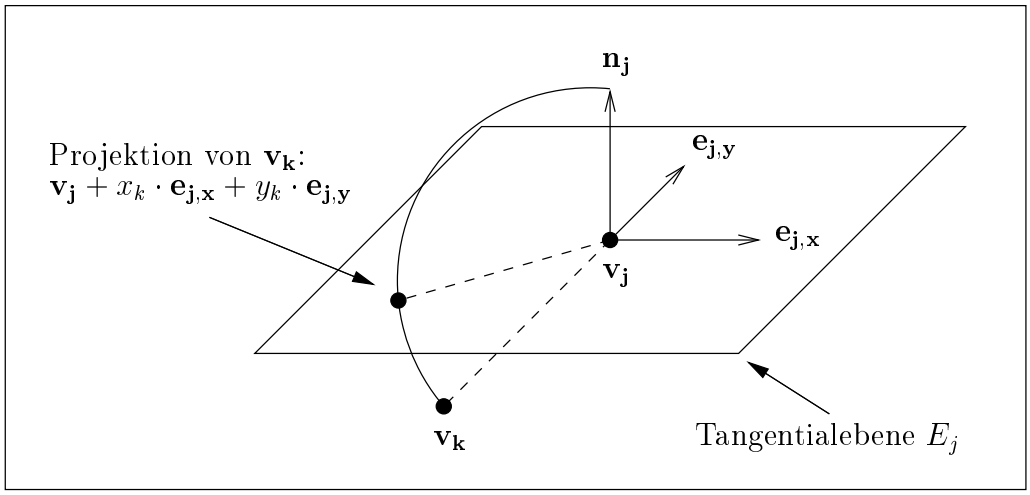


Abbildung 3.1: Rotation von $\mathbf{v_k}$ in die Tangentialebene E_j von $\mathbf{v_j}$

$$y_k = \frac{\|\mathbf{v_k} - \mathbf{v_j}\|}{\sqrt{\tilde{x_k}^2 + \tilde{y_k}^2}} \cdot \tilde{y_k} \tag{3.9}$$

berechnet werden. Ist sowohl $\tilde{x_k} = 0$ als auch $\tilde{y_k} = 0$, das heisst in Richtung $\mathbf{n_j}$ betrachtet liegt $\mathbf{v_k}$ genau vor oder hinter $\mathbf{v_j}$, können die Formeln 3.8 und 3.9 wegen der Division durch null nicht verwendet werden. In einem solchen Fall wird $x_k = 0$ und $y_k = 0$ gesetzt.

Durch die Rotationen der zwölf nähesten Nachbarn von $\mathbf{v_j}$ in die Tangentialebene E_j wird das nachfolgende Problem der Triangulierung des Punktes $\mathbf{v_j}$ und seiner zwölf nähesten Nachbarn von drei auf zwei Dimensionen reduziert.

Das Laufzeitverhalten der Rotationen der zwölf nähesten Nachbarn eines Punktes in dessen Tangentialebene ist unabhängig von der Grösse der Punktewolke. Es gilt daher

$$T_{\textit{Rotationen, ein Punkt}}(n) = O(1) \quad . \tag{3.10}$$

Das Laufzeitverhalten der Rotationen der zwölf nähesten Nachbarn in die entsprechenden Tangentialebenen für alle Punkte beträgt demnach

$$T_{\textit{Rotationen, alle Punkte}}(n) = O(n) \quad . \tag{3.11}$$

3.1.4 Berechnen der Dreiecksfächer

Der Dreiecksfächer eines Punktes $\mathbf{v_j} \in P$ ergibt sich aus einer *Delaunay*-Triangulierung des Punktes $\mathbf{v_j}$ und seiner in dessen Tangentialebene E_j projizierten zwölf nähesten Nachbarn. Den Dreiecksfächer von $\mathbf{v_j}$ bilden diejenigen Dreiecke, die die folgenden beiden Kriterien

erfüllen (Abbildung 3.2 zeigt hierfür ein Beispiel).

- Einer der Eckpunkte des Dreiecks ist $\mathbf{v_j}$.
  Da nur der Fächer des Punktes $\mathbf{v_j}$ von Interesse ist, können alle Dreiecke, für die $\mathbf{v_j}$ nicht Eckpunkt ist, vernachlässigt werden.

- Der Winkel des Dreiecks am Eckpunkt $\mathbf{v_j}$ ist kleiner oder gleich 120°.
  Dreiecke mit grösseren Winkeln als 120° im Zentrum des Fächers sind erfahrungsgemäss häufig schlecht (Definition 4.1; auf schlechte Dreiecke wird erst in Kapitel 4 genauer eingegangen) und werden deshalb erst gar nicht ins Dreiecksnetz eingefügt.

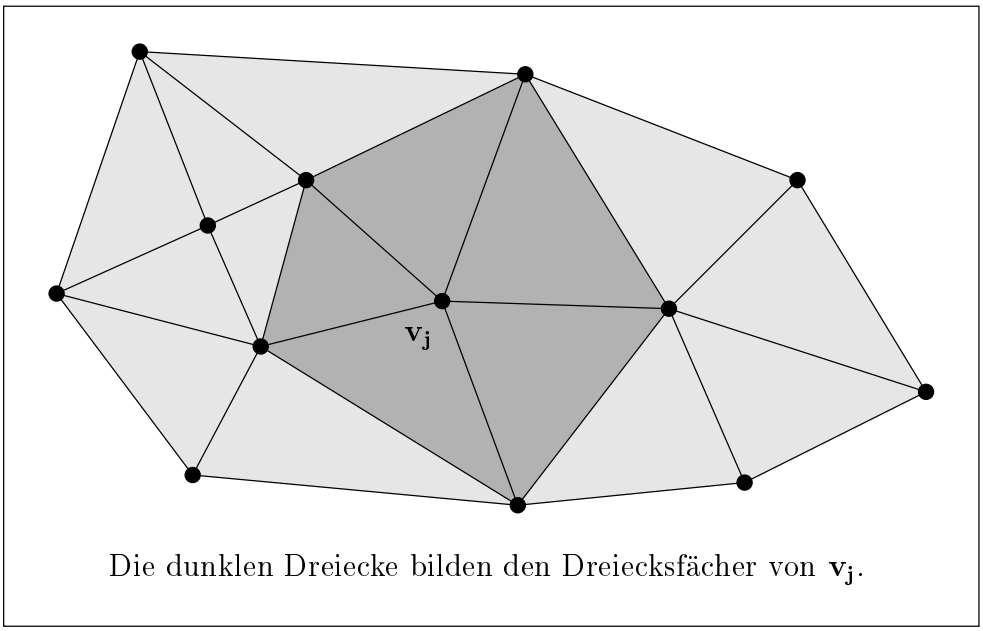


Abbildung 3.2: Berechnen des Dreiecksfächers von $\mathbf{v_j}$ mittels *Delaunay*-Triangulierung

Aufgrund dieser beiden Kriterien kommt es hin und wieder vor, dass der Dreiecksfächer eines Punktes nicht geschlossen ist, oder zu einem Punkt sogar mehrere nicht-geschlossene Dreiecksfächer existieren.

Mehr als zwölf näheste Nachbarn beim Berechnen eines Dreiecksfächers zu betrachten, stellt in den meisten Fällen einen unnötigen Aufwand dar. Die Ergebnisse werden dadurch zwar keinesfalls schlechter, in der Regel jedoch auch nicht besser. Dreiecke, die diese zusätzlich betrachteten Punkte als Eckpunkte haben, gehören nämlich meistens nicht dem gesuchten Dreiecksfächer an und werden daher ohnehin vernachlässigt.

Für den hier vorliegenden Spezialfall (nur die Dreiecke eines Fächers sind von Interesse) existiert ein sehr effizientes Verfahren zur *Delaunay*-Triangulierung. Aus Platzgründen kann

in dieser Studienarbeit nicht darauf eingegangen werden. Eine detaillierte Beschreibung findet sich jedoch in [GoKr00].

Das Laufzeitverhalten zum Berechnen des Dreiecksfächers eines Punktes ist unabhängig von der Grösse der Punktewolke. Es gilt daher

$$T_{Dreiecksfächer,\ ein\ Punkt}(n) \quad = \quad O(1) \quad . \tag{3.12}$$

Das Laufzeitverhalten zum Berechnen der Dreiecksfächer aller Punkte beträgt demnach

$$T_{Dreiecksfächer,\ alle\ Punkte}(n) \quad = \quad O(n) \quad . \tag{3.13}$$

3.2 Laufzeitverhalten

Das Laufzeitverhalten beim Erzeugen einer initialen Triangulierung wird von der Nähesten-Nachbar-Suche dominiert. Es liegt daher ein mittleres Laufzeitverhalten von

$$\overline{T}_{Erzeugen\ einer\ initialen\ Triangulierung}(n) \quad = \quad O(n^{\frac{5}{3}}) \tag{3.14}$$

vor.

Kapitel 4

Entfernen von schlechten Dreiecken (*Schritt 2*)

Definition 4.1 *Ein schlechtes Dreieck in einem bestehenden Dreiecksnetz $\mathcal{T}$ ist ein Dreieck, welches ausschliesst, dass $\mathcal{T}$ durch Hinzufügen neuer Dreiecke zu einem fehlerfreien und geschlossenen Dreiecksnetz erweitert wird.*

Die meisten schlechten Dreiecke sind Dreiecke, die Schnitte mit anderen Dreiecken aufweisen, oder Dreiecke, die topologische Fehler bewirken. Gelegentlich treten jedoch auch Dreiecke auf, die selbst keine Fehler verursachen, jedoch das Schliessen eines vorhandenen Loches verhindern. Solche Dreiecke gehören ebenfalls zur Menge der schlechten Dreiecke.

Bei diesem Schritt geht es darum, in einer bestehenden Triangulierung möglichst viele dieser schlechten Dreiecke zu entdecken und zu entfernen.

Folgende Dreiecke werden entfernt, da sie entweder stark degeneriert (*Fall 1*) oder mit hoher Wahrscheinlichkeit (*Fall 2*) oder mit Sicherheit (*Fall 3*, *Fall 4*, *Fall 5* und *Fall 6*) schlecht sind.

- Dreiecke mit mindestens einem sehr kleinen Winkel (*Fall 1*).
- Dreieckspaare (Dreiecke mit einer gemeinsamen Kante) mit zu kleinem Zwischenwinkel (*Fall 2*).
- Dreiecke an Kanten mit mehr als zwei Dreiecken (*Fall 3*).
- Sich schneidende Dreiecke mit einem gemeinsamen Eckpunkt (*Fall 4*).
- Sich schneidende Dreiecke mit keinem gemeinsamen Eckpunkt (*Fall 5*).
- An der Spitze eines geschlossenen Fächers hängende Dreiecke (*Fall 6*).

Diese sechs Fälle[1] umfassen den Grossteil aller schlechten Dreiecke, leider jedoch nicht alle. Es ist nämlich durchaus möglich, dass ein Dreieck, obwohl es in keine der oben genannten sechs Klassen fällt, das Schliessen eines Loches verhindert. In einem solchen Fall ist es wegen diesem Dreieck nicht mehr möglich neue Dreiecke konfliktfrei einzufügen, um ein bestehendes Loch zu schliessen. Solche Dreiecke sind nur sehr schwer zu entdecken. Derartige Problemfälle werden beim Triangulieren komplexer Löcher (Kapitel 6) behandelt.

Dieser Teil des Algorithmus kommt das erste Mal nach dem Erzeugen einer initialen Triangulierung zum Einsatz. Die im Folgenden genauer beschriebenen sechs Fälle und die dazugehörigen Tests finden jedoch auch bei allen weiteren Schritten des Rekonstruktionsalgorithmus Verwendung, dem Triangulieren einfacher und komplexer Löcher (Kapitel 5 und 6), dem Einfügen isolierter Punkte (Kapitel 7) und dem Glätten des erzeugten Dreiecksnetzes (Kapitel 8).

4.1 Dreiecke mit mindestens einem sehr kleinen Winkel (*Fall 1*)

Dreiecke mit mindestens einem sehr kleinen Winkel sind nach obiger Definition nicht zwangsläufig schlecht. Sie sind jedoch äusserst unerwünscht, da sie nahezu zu Strichen degeneriert sind. Als minimal zulässiger Winkel in einem Dreieck wurde in der Implementierung $10^{-10} rad$ gewählt.

Der Winkel α in einem Dreieck $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ am Eckpunkt $\mathbf{v_j}$ beziehungsweise der Winkel α zwischen zwei Vektoren $\mathbf{v_k} - \mathbf{v_j}$ und $\mathbf{v_l} - \mathbf{v_j}$ lässt sich effizient mit der Formel

$$\alpha \quad = \quad \angle(\mathbf{v_k} - \mathbf{v_j}, \mathbf{v_l} - \mathbf{v_j}) \quad = \quad \arctan \frac{\|(\mathbf{v_k} - \mathbf{v_j}) \times (\mathbf{v_l} - \mathbf{v_j})\|}{\langle \mathbf{v_k} - \mathbf{v_j}, \mathbf{v_l} - \mathbf{v_j} \rangle} \tag{4.1}$$

berechnen[2] (Abbildung 4.1).

Das Laufzeitverhalten eines solchen Tests ist

$$T_{\textit{Test 1, ein Dreieck}}(n) \quad = \quad O(1) \quad . \tag{4.2}$$

Das Laufzeitverhalten zum Testen aller Dreiecke des bestehenden Netzes auf *Fall 1* beträgt demnach

[1] Im weiteren wird auf diese sechs Fälle und die dazugehörigen Tests und Dreiecke häufig nur noch mit der entsprechenden Nummer verwiesen (zum Beispiel *Fall 2*, *Test 5* oder *Fall-6-Dreieck*), ohne deren Bedeutung erneut zu erläutern.

[2] Bei der Implementierung dieses Verfahrens wurde die Funktion *atan2* der C-Standard-Bibliothek verwendet, der Zähler und Nenner separat übergeben werden. Steht eine solche Funktion nicht zur Verfügung ist obige Formel wegen der möglichen Division durch null mit Vorsicht zu gebrauchen.

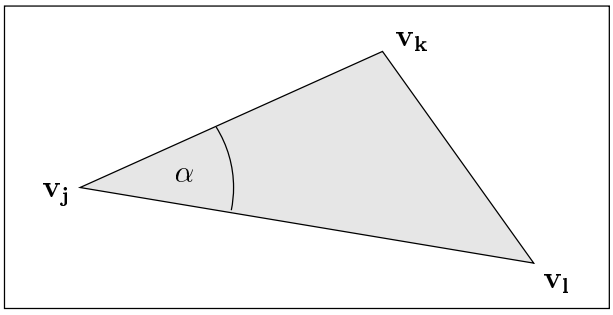


Abbildung 4.1: Berechnen eines Winkels in einem Dreieck

$$T_{\textit{Test 1, alle Dreiecke}}(n) \quad = \quad O(n) \quad . \tag{4.3}$$

4.2 Dreieckspaare mit zu kleinem Zwischenwinkel (*Fall 2*)

Definition 4.2 *Zwei Dreiecke bilden ein Dreieckspaar, wenn sie eine gemeinsame Kante besitzen. Der Zwischenwinkel zweier Dreiecke eines Dreieckspaares ist der Winkel zwischen den beiden Ebenen, in denen diese Dreiecke liegen.*

Dreiecke eines Dreieckspaares mit zu kleinem Zwischenwinkel sind nur dann mit Sicherheit schlecht, wenn der Zwischenwinkel $0°$ beträgt, da sie sich dann schneiden. Ist der Zwischenwinkel klein, ist zumindest eines der beiden Dreiecke mit grosser Wahrscheinlichkeit schlecht. Meistens beschreiben derart „aufeinander liegende“ Dreiecke denselben Teil einer Oberfläche, für den verschiedene Triangulierungen möglich sind. Wenn ein solches Paar entdeckt wird, wird das Dreieck mit dem kleineren Winkel zur gemeinsamen Kante entfernt. Dieses Kriterium ist einfach zu überprüfen und liefert in den meisten Fällen die schönere Triangulierung im Sinne maximaler kleinster Winkel (Abbildung 4.2; Dreieck 1 und Dreieck 2 beschreiben denselben Teil einer Oberfläche, eines von beiden ist also unerwünscht; Dreieck 2 hat den spitzeren Winkel zur gemeinsamen Kante und wird daher entfernt; das Ergebnis ist Triangulierung 1). Der minimal zulässige Zwischenwinkel ist vom Benutzer vorzugegeben. Experimente haben gezeigt, dass dieser etwa $90°$ betragen sollte.

Der Zwischenwinkel α zweier Dreiecke $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ und $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_m})$ kann mit Hilfe der Formeln

$$\mathbf{n_0} \quad = \quad (\mathbf{v_j} - \mathbf{v_l}) \times (\mathbf{v_k} - \mathbf{v_l}) \tag{4.4}$$

$$\mathbf{n_1} \quad = \quad (\mathbf{v_k} - \mathbf{v_m}) \times (\mathbf{v_j} - \mathbf{v_m}) \tag{4.5}$$

$$\alpha \quad = \quad 180° - \angle(\mathbf{n_0}, \mathbf{n_1}) \tag{4.6}$$

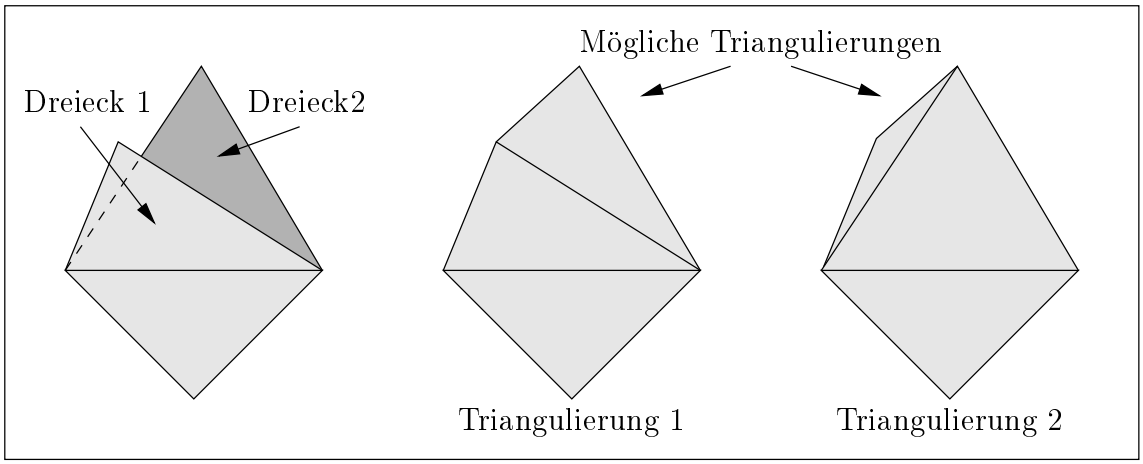


Abbildung 4.2: „Aufeinander liegende“ Dreiecke

berechnet werden, wobei $\mathbf{n_0}$ die Normale von $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ und $\mathbf{n_1}$ die Normale von $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_m})$ ist (Abbildung 4.3).

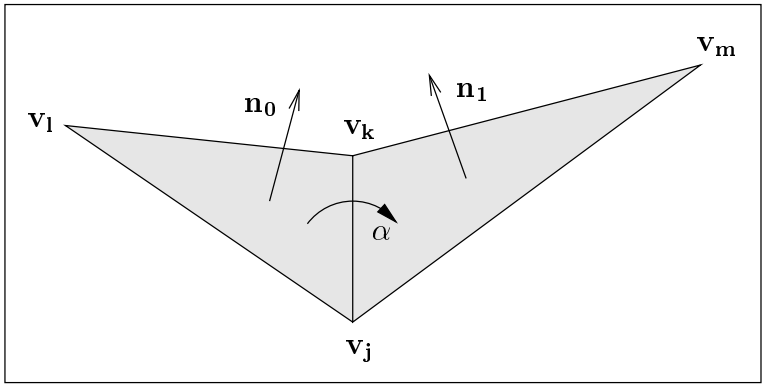


Abbildung 4.3: Berechnen des Zwischenwinkels zweier Dreiecke

Das Laufzeitverhalten eines solchen Tests ist

$$T_{\textit{Test 2, ein Dreieck}}(n) \quad = \quad O(1) \quad . \tag{4.7}$$

Das Laufzeitverhalten zum Testen aller Dreieckspaare des bestehenden Netzes auf *Fall 2* beträgt demnach

$$T_{\textit{Test 2, alle Dreiecke}}(n) \quad = \quad O(n) \quad . \tag{4.8}$$

4.3 Dreiecke an Kanten mit mehr als zwei Dreiecken (*Fall 3*)

Dreiecke an Kanten mit mehr als zwei Dreiecken sind mit Sicherheit schlecht, da die Topologie des Dreiecksnetzes in einem solchen Fall nicht mehr korrekt ist (Abbildung 4.4). Bis auf zwei Dreiecke werden alle entfernt, diejenigen mit den kleinsten Winkeln zur gemeinsamen Kante zuerst. Dieses Kriterium ist einfach zu überprüfen und liefert in den meisten Fällen die schönere Triangulierung im Sinne maximaler kleinster Winkel.

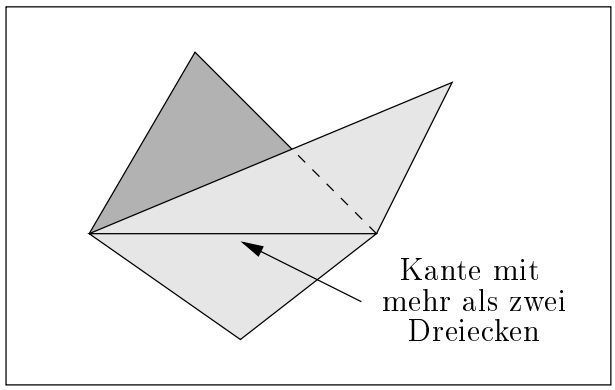


Abbildung 4.4: Dreiecke an einer Kante mit mehr als zwei Dreiecken

Das Laufzeitverhalten eines solchen Tests ist

$$T_{\textit{Test 3, ein Dreieck}}(n) \quad = \quad O(1) \quad . \tag{4.9}$$

Das Laufzeitverhalten zum Testen aller Kanten des bestehenden Netzes auf *Fall 3* beträgt demnach

$$T_{\textit{Test 3, alle Dreiecke}}(n) \quad = \quad O(n) \quad . \tag{4.10}$$

4.4 Sich schneidende Dreiecke mit einem gemeinsamen Eckpunkt (*Fall 4*)

Nach Definition 4.1 sind alle sich schneidenden Dreiecke schlecht, also auch solche mit einem gemeinsamen Eckpunkt. Wurden alle *Fall-1-Dreiecke*, *Fall-2-Dreiecke* und *Fall-3-Dreiecke* bereits entfernt, gibt es in aller Regel nur noch wenige Dreiecke mit einem gemeinsamen Eckpunkt, die sich schneiden. Das Dreieck mit dem kleinsten Winkel wird entfernt. Dieses Kriterium ist einfach zu überprüfen und liefert in den meisten Fällen zufriedenstellende Ergebnisse. Wegen der geringen Anzahl derartiger Dreiecke ist eine Verfeinerung dieses Kriteriums wenig sinnvoll.

Um herauszufinden, ob sich zwei Dreiecke mit einem gemeinsamen Eckpunkt schneiden, muss zuerst überprüft werden, ob die beiden Dreiecke in einer Ebene liegen oder nicht. Dazu werden deren Normalen berechnet und verglichen. Sind diese Normalen parallel oder antiparallel, so liegen die beiden Dreiecke, da sie einen gemeinsamen Eckpunkt besitzen, in derselben Ebene. Es ist bei diesem Test vorteilhaft, die Normale $\mathbf{n}$ eines Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ mit den Formeln

$$\mathbf{a} = \mathbf{v_j} - \mathbf{v_i} \tag{4.11}$$

$$\mathbf{b} = \mathbf{v_k} - \mathbf{v_j} \tag{4.12}$$

$$\mathbf{c} = \mathbf{v_i} - \mathbf{v_k} \tag{4.13}$$

$$\mathbf{n} = \frac{(\mathbf{a} \times \mathbf{b}) + (\mathbf{b} \times \mathbf{c}) + (\mathbf{c} \times \mathbf{a})}{\|(\mathbf{a} \times \mathbf{b}) + (\mathbf{b} \times \mathbf{c}) + (\mathbf{c} \times \mathbf{a})\|} \tag{4.14}$$

zu bestimmen, da diese auch für sehr spitze Dreiecke numerisch recht stabil sind.

4.4.1 Nicht in einer Ebene liegende Dreiecke

Liegen zwei Dreiecke $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ und $\triangle(\mathbf{v_i}, \mathbf{v_l}, \mathbf{v_m})$ nicht in einer Ebene, schneiden sie sich genau dann, wenn die dem gemeinsamen Punkt $\mathbf{v_i}$ gegenüberliegende Kante $e(\mathbf{v_l}, \mathbf{v_m})$ die durch das Dreieck $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ definierte „Viertelebene“ schneidet (Abbildung 4.5). Dafür muss das lineare Gleichungssystem

$$\mathbf{v_i} + \alpha \cdot (\mathbf{v_j} - \mathbf{v_i}) + \beta \cdot (\mathbf{v_k} - \mathbf{v_i}) = \mathbf{v_l} + \gamma \cdot (\mathbf{v_m} - \mathbf{v_l}) \tag{4.15}$$

gelöst werden. Als effizientes, numerisch stabiles Verfahren bietet sich dafür der *Gauss*-Algorithmus mit relativer Spaltenmaximum-Strategie an, der unter anderem in [Grab00] ausführlich dargestellt wird. Ein Schnitt liegt genau dann vor, wenn $\alpha \geq 0$, $\beta \geq 0$ und $0 \leq \gamma \leq 1$.

4.4.2 In einer Ebene liegende Dreiecke

Liegen zwei Dreiecke $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ und $\triangle(\mathbf{v_i}, \mathbf{v_l}, \mathbf{v_m})$ in einer Ebene, kann das Problem leicht auf zwei Dimensionen reduziert werden, indem die unwichtigste der drei Dimensionen vernachlässigt wird. Festgelegt wird diese durch die betragsmässig grösste Komponente der Normale der beiden Dreiecke. Im Zweidimensionalen werden die Winkel α (Winkel zwischen den Dreiecksseiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_k})$), β (Winkel zwischen den Dreiecksseiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_l})$) und γ (Winkel zwischen den Dreiecksseiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_m})$) berechnet (Abbildung 4.6). Es ist zu beachten, das alle Winkel in derselben Richtung, also entweder im oder gegen den Uhrzeigersinn gemessen werden. Die Richtung ist so zu wählen, dass der Win-

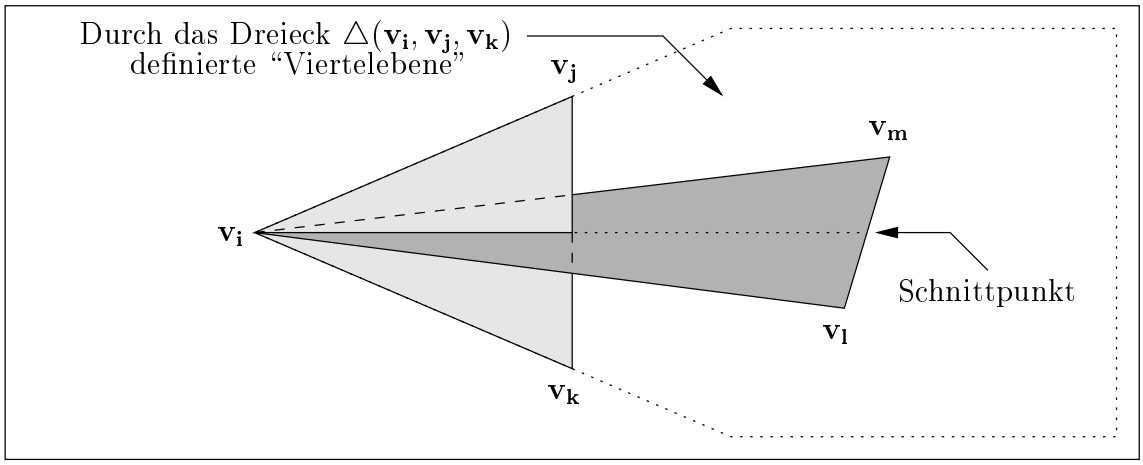


Abbildung 4.5: Nicht in einer Ebene liegende Dreiecke

kel α kleiner oder gleich 180° ist, also dem Winkel des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ am Eckpunkt $\mathbf{v_i}$ entspricht. Die beiden Dreiecke schneiden sich, falls mindestens eine der folgenden drei Bedingungen erfüllt ist.

1. $\beta \leq \alpha$, das heisst die Seite $e(\mathbf{v_i}, \mathbf{v_l})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_l}, \mathbf{v_m})$ liegt zwischen den beiden Seiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_k})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$.

2. $\gamma \leq \alpha$, das heisst die Seite $e(\mathbf{v_i}, \mathbf{v_m})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_l}, \mathbf{v_m})$ liegt zwischen den beiden Seiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_k})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$.

3. $\beta > \alpha$ und $\gamma > \alpha$ und $|\beta - \gamma| \geq 180°$, das heisst die beiden Seiten $e(\mathbf{v_i}, \mathbf{v_l})$ und $e(\mathbf{v_i}, \mathbf{v_m})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_l}, \mathbf{v_m})$ schliessen die beiden Seiten $e(\mathbf{v_i}, \mathbf{v_j})$ und $e(\mathbf{v_i}, \mathbf{v_k})$ des Dreiecks $\triangle(\mathbf{v_i}, \mathbf{v_j}, \mathbf{v_k})$ ein.

4.4.3 Laufzeitverhalten

Das Laufzeitverhalten eines solchen Tests ist unabhängig davon, ob die beiden Dreiecke in einer Ebene liegen oder nicht. Es gilt

$$T_{\textit{Test 4, ein Dreieck}}(n) \quad = \quad O(1) \quad . \tag{4.16}$$

Das Laufzeitverhalten zum Testen aller Dreiecke des bestehenden Netzes auf *Fall 4* beträgt demnach

$$T_{\textit{Test 4, alle Dreiecke}}(n) \quad = \quad O(n) \quad . \tag{4.17}$$

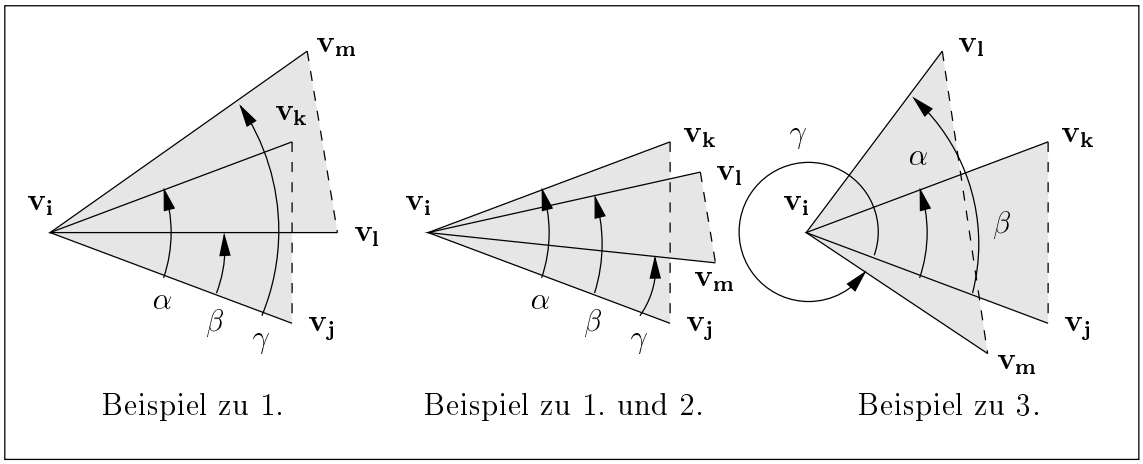


Abbildung 4.6: In einer Ebene liegende Dreiecke

4.5 Sich schneidende Dreiecke mit keinem gemeinsamen Eckpunkt (*Fall 5*)

Nach Definition 4.1 sind alle sich schneidenden Dreiecke schlecht, also auch solche mit keinem gemeinsamen Eckpunkt (Abbildung 4.7). Wurden bereits alle *Fall-1-Dreiecke*, *Fall-2-Dreiecke*, *Fall-3-Dreiecke* und *Fall-4-Dreiecke* entfernt, gibt es in den meisten Fällen gar keine Dreiecke mit keinem gemeinsamen Eckpunkt mehr, die sich schneiden. Tritt ein solches Paar dennoch auf, wird das Dreieck mit dem kleinsten Winkel entfernt. Dieses Kriterium ist einfach zu überprüfen und liefert in den meisten Fällen zufriedenstellende Ergebnisse. Wegen der äusserst geringen Anzahl derartiger Dreiecke ist eine Verfeinerung dieses Kriteriums wenig sinnvoll.

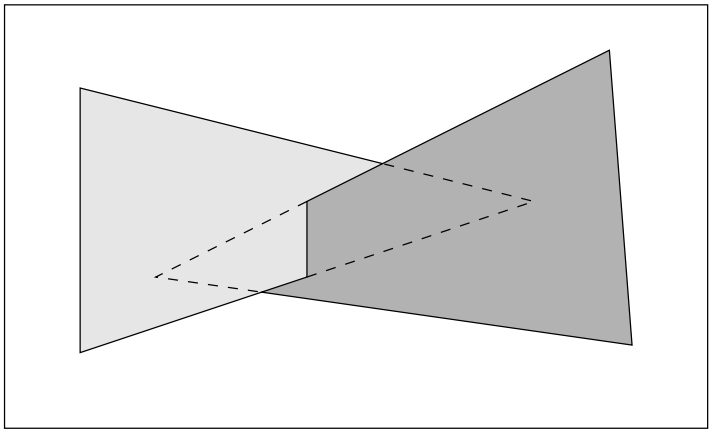


Abbildung 4.7: Sich schneidende Dreiecke mit keinem gemeinsamen Eckpunkt

Effizient zu überprüfen, ob sich zwei Dreiecke mit keinem gemeinsamen Eckpunkt schneiden

oder nicht, ist keine einfache Angelegenheit. Der in dieser Studienarbeit verwendete Algorithmus für diesen Test entstammt [Moel97]. Aus Platzgründen wird in dieser Ausarbeitung nicht weiter darauf eingegangen.

Im Gegensatz zu *Fall 1*, *Fall 2*, *Fall 3*, *Fall 4* und *Fall 6* kann dieser Test nicht lokal durchgeführt werden, das heisst es reicht nicht, nur in der topologischen Nachbarschaft eines Dreiecks nach Konflikten zu suchen. Das Laufzeitverhalten eines solchen Tests ist daher

$$T_{\textit{Test 5, ein Dreieck}}(n) \quad = \quad O(n) \quad . \tag{4.18}$$

Sollen alle Dreiecke eines bestehenden Netzes auf *Fall 5* getestet werden, werden die Dreiecke zuerst entsprechend ihrer minimalen Eckpunkt-Komponente entlang einer Achse sortiert. Dadurch kann die Menge der notwendigen Tests stark reduziert werden. Als Kandidaten für einen Konflikt kommen nämlich nur die Dreiecke in Frage, deren minimale Eckpunkt-Komponente sowohl grösser oder gleich der minimalen Eckpunkt-Komponente als auch kleiner oder gleich der maximalen Eckpunkt-Komponente des zu testenden Dreiecks ist. Als Achse sollte diejenige gewählt werden, entlang der die Punktewolke die grösste Ausdehnung besitzt. Abhängig vom Dreiecksnetz liegt demnach das Laufzeitverhalten zum Testen aller Dreiecke des bestehenden Netzes auf *Fall 5* irgendwo im Bereich

$$O(n \cdot \log n) \quad \leq \quad T_{\textit{Test 5, alle Dreiecke}}(n) \quad \leq \quad O(n^2) \quad . \tag{4.19}$$

4.6 An der Spitze eines geschlossenen Fächers hängende Dreiecke (*Fall 6*)

An der Spitze eines geschlossenen Fächers hängende Dreiecke sind mit Sicherheit schlecht, da in solchen Fällen die Topologie des Dreiecksnetzes nicht mehr korrekt ist (Abbildung 4.8). Der geschlossene Fächer wird behalten, alle anderen Dreiecke werden entfernt. Dieses Kriterium liefert in den meisten Fällen eine der Form des zur gegebenen Punktewolke gehörigen Objekts entsprechende Triangulierung. Im äusserst seltenen Fall von mehreren geschlossenen Fächern wird ein beliebiger behalten.

Das Laufzeitverhalten eines solchen Tests ist

$$T_{\textit{Test 6, ein Dreieck}}(n) \quad = \quad O(1) \quad . \tag{4.20}$$

Das Laufzeitverhalten zum Testen aller Dreiecke des bestehenden Netzes auf *Fall 6* beträgt demnach

$$T_{\textit{Test 6, alle Dreiecke}}(n) \quad = \quad O(n) \quad . \tag{4.21}$$

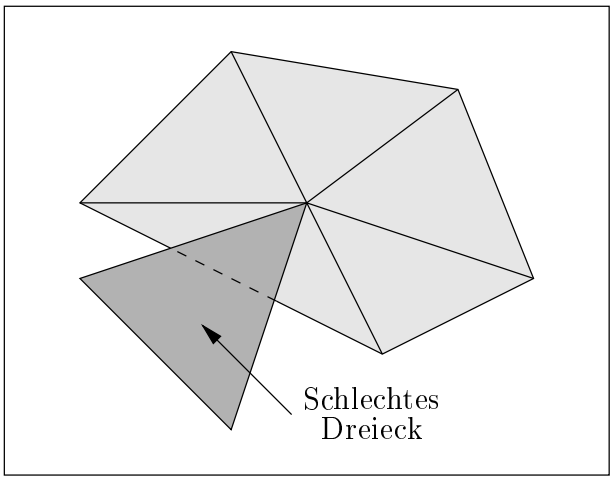


Abbildung 4.8: An der Spitze eines geschlossenen Fächers hängendes Dreieck

4.7 Reihenfolge der sechs Fälle

Die Reihenfolge, in der die oben beschriebenen sechs Klassen von schlechten Dreiecken aus einem bestehenden Dreiecksnetz entfernt werden, ist im Prinzip frei wählbar. Folgende Punkte sollten jedoch beachtet werden.

- *Fall 2* sollte vor *Fall 3* behandelt werden. Es kann sonst vorkommen, dass durch *Test 3* an einer Kante mit mehr als zwei Dreiecken alle bis auf zwei entfernt werden, und der Zwischenwinkel dieser beiden Dreiecke sehr klein ist. *Test 2* entfernt dann ein weiteres dieser beiden Dreiecke, wodurch eine offene Kante entsteht. Bei umgekehrter, besserer Reihenfolge bleiben dagegen fast immer zwei Dreiecke erhalten.
- *Fall 5* sollte nicht zuerst behandelt werden, da dies der bei weitem zeitaufwendigste Test ist. Je mehr Dreiecke vor *Test 5* entfernt werden, desto besser.

Eine mögliche sinnvolle Reihenfolge wird durch die Nummern der sechs Fälle vorgegeben, also zuerst *Fall 1*, dann *Fall 2*, *Fall 3*, *Fall 4*, *Fall 5* und zuletzt *Fall 6*.

4.8 Laufzeitverhalten

Das Laufzeitverhalten zum Entdecken schlechter Dreiecke wird eindeutig von *Fall 5* dominiert, da dieser Test der einzige ist, der nicht lokal durchgeführt werden kann.

Das Laufzeitverhalten zum Testen eines Dreiecks beträgt demnach

$$T_{\textit{Testen eines Dreiecks}}(n) \quad = \quad O(n) \quad . \tag{4.22}$$

Abhängig vom Dreiecksnetz liegt das Laufzeitverhalten zum Testen aller Dreiecke des bestehenden Netzes irgendwo im Bereich

$$O(n \cdot \log n) \quad \leq \quad T_{\textit{Entfernen von schlechten Dreiecken}}(n) \quad \leq \quad O(n^2) \quad . \tag{4.23}$$

Kapitel 5

Triangulieren einfacher Löcher (*Schritt 3*)

Definition 5.1 *Ein einfaches Loch ist ein Loch, das lediglich durch Einfügen neuer Dreiecke geschlossen werden kann. Um ein solches Loch zu schliessen, ist es nicht notwendig, irgendwelche störenden Dreiecke aus dem Netz zu entfernen.*

Bei diesem Schritt wird versucht, durch sukzessives Einfügen von neuen Dreiecken so viele einfache Löcher wie möglich zu schliessen. Dazu wird eine sortierte Liste von möglichen Einfügepositionen aufgebaut, deren erstes Element stets die momentan günstigste Einfügemöglichkeit ist. Danach wird an der durch das erste Listenelement vorgegebenen Stelle ein neues Dreieck eingefügt, dieses Listenelement entfernt, und die Liste der Einfügemöglichkeiten aktualisiert. Dieser Schritt wird solange wiederholt, bis keine weiteren sinnvollen Einfügemöglichkeiten mehr erkannt werden. Dies ist genau dann der Fall, wenn die Liste der Einfügepositionen leer ist.

Dieser Schritt setzt ein fehlerfreies Dreiecksnetz voraus. Ein durch *Schritt 1* und *Schritt 2* erzeugtes Dreiecksnetz erfüllt diese Voraussetzung.

5.1 Aufbauen einer Kantenpaar-Liste

Definition 5.2 *Ein Kantenpaar ist ein Paar von zwei offenen Kanten, die verschiedenen Dreiecken angehören, jedoch einen gemeinsamen Endpunkt besitzen.*

Kantenpaare stellen Einfügemöglichkeiten für Dreiecke dar. Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ repräsentiert die Einfügemöglichkeit für das Dreieck $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$.

Bevor mit dem Einfügen neuer Dreiecke begonnen wird, wird zuerst eine Kantenpaar-Liste aufgebaut, die alle als sinnvoll erkannten Einfügemöglichkeiten enthält. In diese Liste werden alle Kantenpaare $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ aufgenommen, bei denen der gemeinsame Endpunkt

$\mathbf{v_j}$ keine weiteren offenen Kanten besitzt. Besitzt ein Punkt nämlich mehr als zwei offene Kanten, ist es unter Umständen äusserst schwierig zu entscheiden, welche davon sinnvolle Kantenpaare, also Einfügemöglichkeiten für neue, nicht-schlechte Dreiecke bilden. Vor allem wenn die Kanten des Punktes nicht annähernd in einer Ebene liegen, also die Oberfläche des zur gegebenen Punktewolke gehörigen Objekts an dieser Stelle stark gekrümmt ist, existieren lokal betrachtet häufig mehrere sinnvolle Triangulierungen (Abbildung 5.1; die Kantenpaare an denen Dreiecke eingefügt wurden, sind durch Doppelpfeile gekennzeichnet; die Formen der beiden Triangulierungen unterscheiden sich gravierend, eine von beiden entspricht also mit Sicherheit nicht der Form des zur gegebenen Punktewolke gehörigen Objekts). Um diese Schwierigkeiten zu vermeiden wird die Kantenpaar-Liste auf eindeutige Fälle beschränkt.

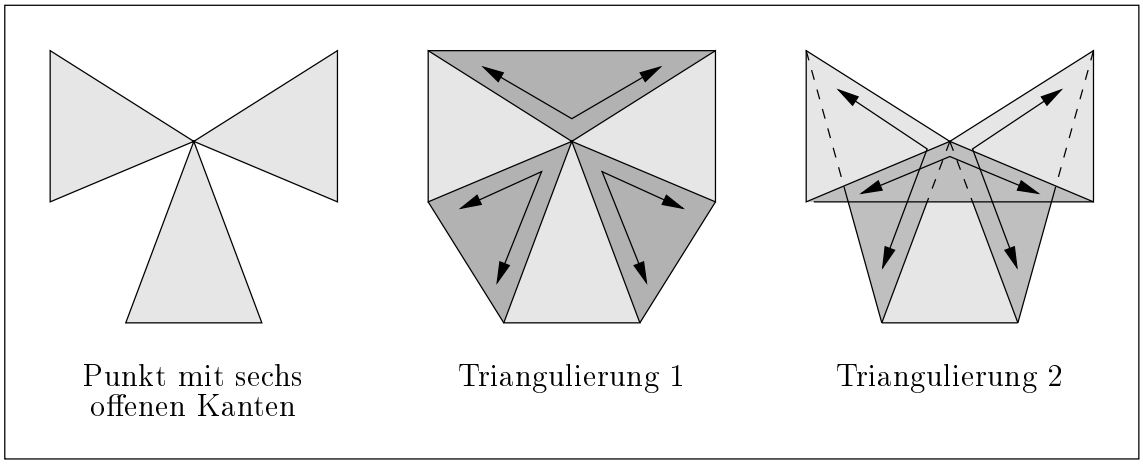


Abbildung 5.1: Verschiedene sinnvolle Triangulierungen

5.2 Sortieren der Kantenpaar-Liste

Für jedes Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ der Kantenpaar-Liste wird nun der Mittelwert der beiden Zwischenwinkel an den Kanten $e(\mathbf{v_j}, \mathbf{v_k})$ und $e(\mathbf{v_j}, \mathbf{v_l})$ berechnet (Abbildung 5.2). Für diese Berechnung wird das durch das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ vorgegebene Dreieck $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ als Teil des Dreiecksnetzes betrachtet. Die Zwischenwinkel können wie in Abschnitt 4.2 beschrieben ermittelt werden.

Die Kantenpaar-Liste wird dann nach den Mittelwerten der Zwischenwinkel sortiert, wobei grosse Winkel am Anfang der Liste, kleine Winkel am Ende der Liste eingeordnet werden. Da immer das jeweils erste Element der Kantenpaar-Liste abgearbeitet wird, gewährleistet dies, dass Dreiecke zuerst an Stellen geringer Krümmung eingefügt werden (grosse Zwischenwinkel bedeuten geringe, kleine Zwischenwinkel starke Krümmung). Eingefügte Dreiecke an Stellen starker Krümmung sind erfahrungsgemäss häufiger schlecht als Dreiecke an Stellen geringer Krümmung. Desweiteren werden dadurch Kanten, an denen relativ häufig Löcher auftreten,

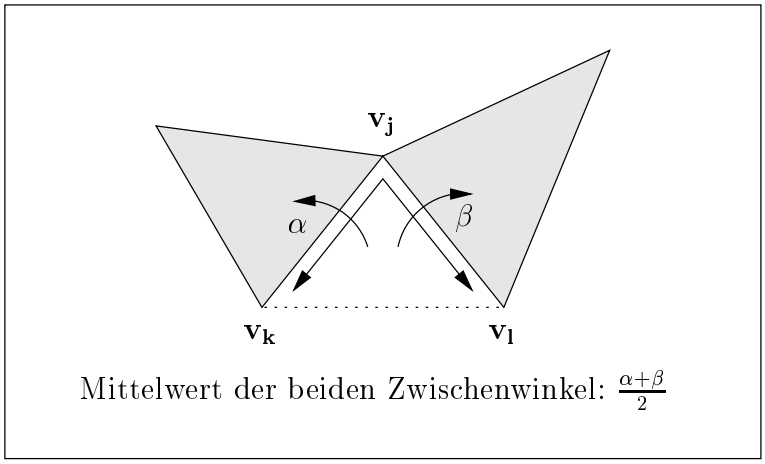


Abbildung 5.2: Die beiden Zwischenwinkel eines Kantenpaares

glatt statt schartig trianguliert (Abbildung 5.3).

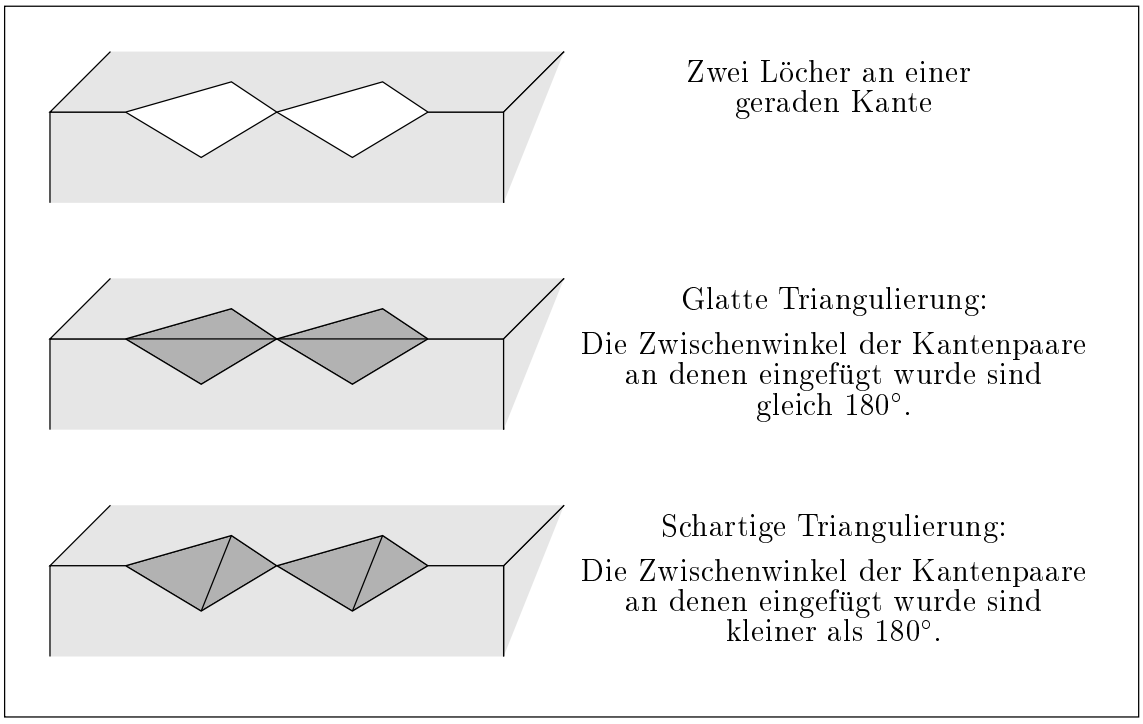


Abbildung 5.3: Glatte und schartige Triangulierung einer Kante

5.3 Einfügen neuer Dreiecke

Solange die Kantenpaar-Liste nicht leer ist, wird immer das jeweils erste Element aus ihr entfernt. An der durch dieses Kantenpaar vorgegebenen Stelle wird dann ein neues Dreieck eingefügt, falls dieses Dreieck nicht zu einer der sechs in Kapitel 4 beschriebenen Klassen von unerwünschten beziehungsweise schlechten Dreiecken gehört.

Die Kantenpaar-Liste soll einerseits immer genau diejenigen Kantenpaare enthalten, deren gemeinsamer Endpunkt keine weiteren offenen Kanten besitzt, andererseits stets nach den Mittelwerten der Zwischenwinkel ihrer Kantenpaare sortiert sein. Daher muss die Liste beim Einfügen eines neuen Dreiecks in den meisten Fällen aktualisiert werden. Wird das neue Dreieck $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$, vorgegeben durch das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$, eingefügt, können einer oder mehrere der folgenden Fälle auftreten.

a) Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_k}, \mathbf{v_l}))$ befindet sich in der Kantenpaar-Liste, ein Loch wurde geschlossen (Abbildung 5.4):
Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_k}, \mathbf{v_l}))$ muss aus der Liste entfernt werden.

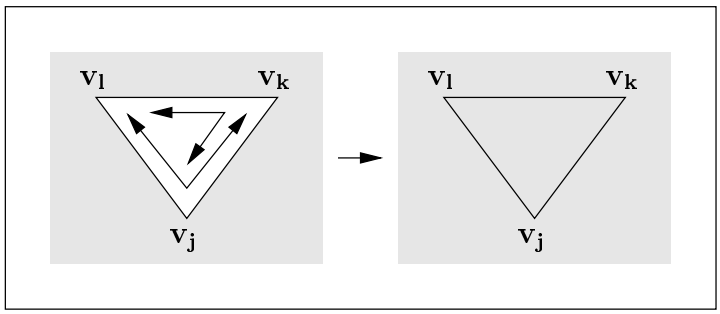


Abbildung 5.4: Beispiel zu Fall a)

b) Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_k}, \mathbf{v_m}))$ befindet sich in der Kantenpaar-Liste, kein Loch wurde geschlossen (Abbildung 5.5):
Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_k}, \mathbf{v_m}))$ muss durch das Kantenpaar $(e(\mathbf{v_k}, \mathbf{v_l}), e(\mathbf{v_k}, \mathbf{v_m}))$ ersetzt, und entsprechend dem neuen mittleren Zwischenwinkel in der Kantenpaar-Liste verschoben werden.

c) Das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_k}, \mathbf{v_l}))$ befindet sich nicht in der Kantenpaar-Liste (Abbildung 5.6):
Falls der Punkt $\mathbf{v_k}$ genau zwei offene Kanten hat muss ein neues, aus diesen Kanten bestehendes Kantenpaar sortiert in die Kantenpaar-Liste eingefügt werden.

d) Spiegelsymmetrischer Fall zu a).

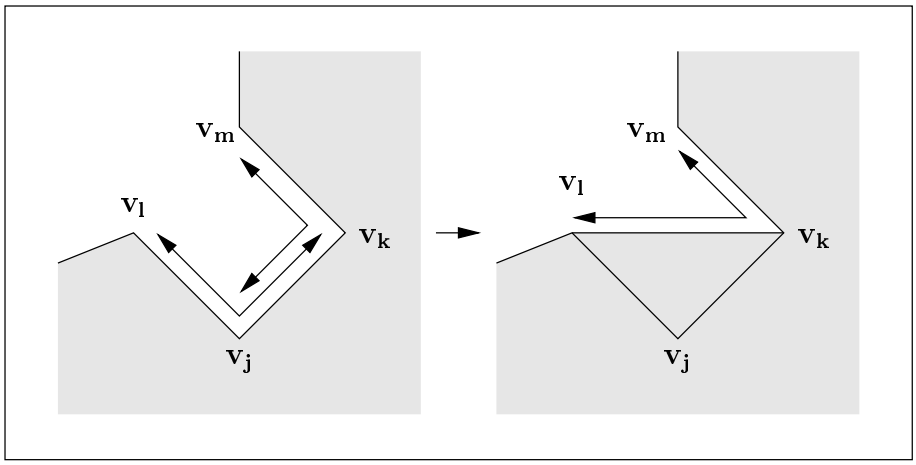


Abbildung 5.5: Beispiel zu Fall b)

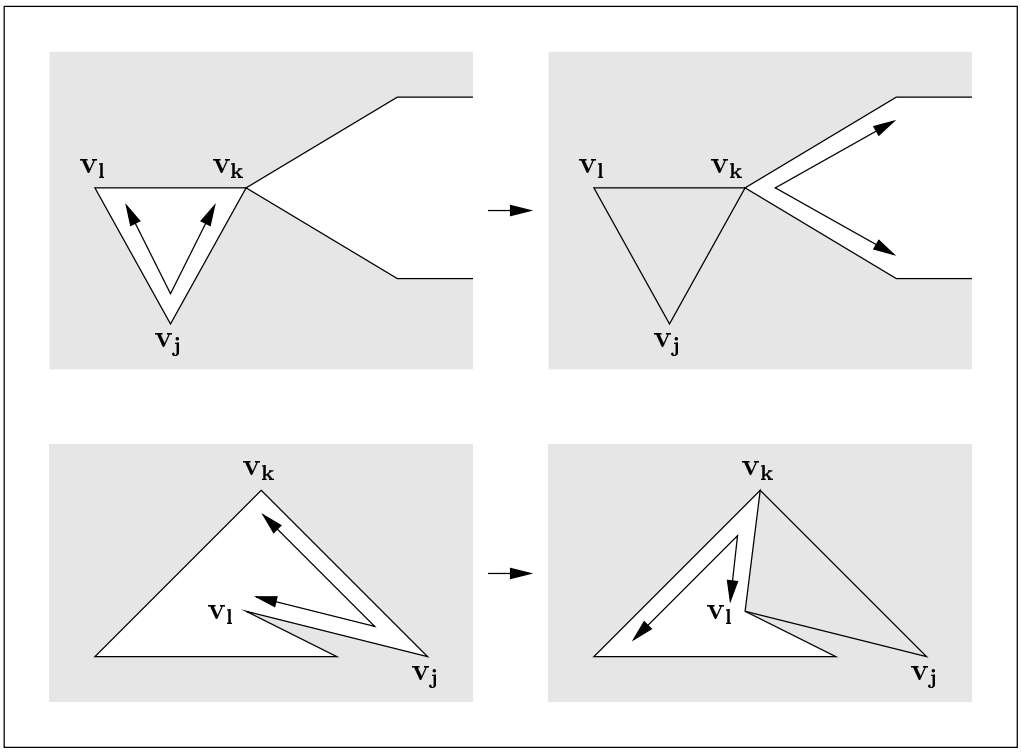


Abbildung 5.6: Beispiele zu Fall c)

e) Spiegelsymmetrischer Fall zu b).

f) Spiegelsymmetrischer Fall zu c).

Die Beschreibungen der spiegelsymmetrischen Fälle ergeben sich, wenn in den Fällen a), b)

und c) die Punkte $\mathbf{v_k}$ und $\mathbf{v_l}$ vertauscht werden.

Pseudocode für das Triangulieren einfacher Löcher: In den Zeilen 1 bis 5 wird eine Kantenpaar-Liste aufgebaut, die alle eindeutigen Einfügemöglichkeiten für Dreiecke enthält. Danach wird diese Liste absteigend nach den mittleren Zwischenwinkeln ihrer Kantenpaare sortiert (Zeilen 6 und 7). In den Zeilen 8 bis 13 wird der Kantenpaarliste die jeweils beste Einfügemöglichkeit entnommen und überprüft, ob ein Einfügen eines neuen Dreiecks an dieser Stelle Konflikte hervorrufen würde. Ist dies nicht der Fall, wird das neue Dreieck in das bestehende Netz aufgenommen, und die Kantenpaarliste entsprechend aktualisiert.

1	*liste* = *Leere Liste von Kantenpaaren.*
2	FOR (Alle Punkte $\mathbf{v_j} \in P$):
3	IF ($\mathbf{v_j}$ hat genau zwei offene Kanten $e(\mathbf{v_j}, \mathbf{v_k})$ und $e(\mathbf{v_j}, \mathbf{v_l})$):
4	IF ($e(\mathbf{v_j}, \mathbf{v_k})$ und $e(\mathbf{v_j}, \mathbf{v_l})$ gehören zu verschiedenen Dreiecken):
5	Füge das Kantenpaar $(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ in *liste* ein.
6	Berechne für alle Kantenpaare in *liste* den Mittelwert der beiden Zwischenwinkel.
7	Sortiere *liste* absteigend nach den Mittelwerten der Zwischenwinkel.
8	WHILE (*liste* ist nicht leer):
9	$(e(\mathbf{v_j}, \mathbf{v_k}), e(\mathbf{v_j}, \mathbf{v_l}))$ = *Erstes Element von liste.*
10	Entferne das erste Element aus *liste*.
11	IF ($\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ ist kein *Fall-1-Dreieck*, kein *Fall-2-Dreieck*, kein *Fall-3-Dreieck*, kein *Fall-4-Dreieck*, kein *Fall-5-Dreieck* und kein *Fall-6-Dreieck*):
12	Füge $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ in das Dreiecksnetz ein.
13	Aktualisiere *liste*.

Pseudocode für das Triangulieren einfacher Löcher

5.4 Laufzeitverhalten

Das Laufzeitverhalten für das Triangulieren einfacher Löcher hängt stark von der Anzahl m der offenen Kanten zu Beginn dieses Schrittes ab. Jede offene Kante gehört höchstens zu zwei Kantenpaaren in der Kantenpaar-Liste. Die Kantenpaar-Liste kann zu Beginn also maximal $2m$ Elemente enthalten.

Der Aufwand für das Abarbeiten eines Kantenpaares wird von den Tests auf schlechte Dreiecke dominiert (Kapitel 4). Es gilt daher

$$T_{Ein\ Kantenpaar}(n) \quad = \quad O(n) \quad . \tag{5.1}$$

Leider wird die Anzahl m der offenen Kanten nur dann reduziert, wenn ein Dreieck an der von dem Kantenpaar vorgegebenen Stelle eingefügt wird, nicht jedoch, wenn das Einfügen wegen Konflikten unmöglich ist. Die Anzahl der Kantenpaare in der Kantenpaar-Liste wird durch das Abarbeiten eines Kantenpaares unter Umständen sogar erhöht (Fälle c) und f), Abschnitt 5.3).

Diese Überlegungen führen zu einer sehr groben oberen Schranke

$$T_{Triangulieren\ einfacher\ Löcher}(n, m) \quad = \quad O(n \cdot m^2) \tag{5.2}$$

für das Laufzeitverhalten zum Triangulieren einfacher Löcher. Die Herleitung dieser oberen Schranke ist nicht ganz einfach. Aus Platzgründen wird in dieser Ausarbeitung nicht weiter darauf eingegangen. Da jedoch die Anzahl m der offenen Kanten nicht nur von n, der Grösse der Punktewolke, abhängt sondern auch von der Lage der Punkte, ist die Angabe einer oberen Schranke $T_{Triangulieren\ einfacher\ Löcher}(n, m)$ ohnehin nur von geringem Nutzen.

In der Praxis hat sich jedoch unabhängig von den Grössen der Punktewolken und den Formen der dazugehörigen Objekte gezeigt, dass das Triangulieren einfacher Löcher nur einen sehr geringen Teil zur Gesamtlaufzeit des Rekonstruktionsalgorithmus beiträgt.

Kapitel 6

Triangulieren komplexer Löcher (*Schritt 4*)

Definition 6.1 *Ein komplexes Loch ist ein Loch, das nicht ausschliesslich durch Einfügen neuer Dreiecke trianguliert werden kann. Um ein komplexes Loch zu schliessen, ist es notwendig, mindestens ein störendes Dreieck, unter Umständen sogar mehrere, aus dem Netz zu entfernen.*

Bei diesem Schritt geht es darum, alle noch verbliebenen Löcher zu schliessen. Wenn dieser Schritt direkt auf das Triangulieren einfacher Löcher folgt (*Schritt 3*), handelt es sich bei diesen Löchern ausschliesslich um komplexe Löcher. Das Schliessen der Löcher geschieht mit Hilfe eines zufallsgesteuerten Optimierungsverfahrens, bekannt unter dem Namen *Simulated-Annealing*. Der Hauptunterschied zu allen bisherigen Schritten besteht darin, dass das Dreiecksnetz zeitweise in schlechtere Zustände gebracht wird, um letztendlich zu sehr viel besseren Triangulierungen im Sinne von weniger offenen Kanten zu gelangen.

Dieser Schritt setzt ein fehlerfreies Dreiecksnetz voraus. Ein durch *Schritt 1*, *Schritt 2* und *Schritt 3* erzeugtes Dreiecksnetz erfüllt diese Voraussetzung.

6.1 *Simulated-Annealing*

Simulated-Annealing ist ein stochastischer Algorithmus zur Lösung diskreter Optimierungsprobleme.

Es geht darum, eine globale Minimalstelle einer Bewertungsfunktion $f : Z \to \mathbb{R}$ zu finden. Diese Funktion bildet eine endliche, unter Umständen jedoch sehr grosse Menge von Zuständen oder Konfigurationen $Z = \{z_1, z_2, \ldots, z_q\}$ in die Menge der reellen Zahlen ab.

Auswerten der Funktion f für alle Elemente von Z löst das Optimierungsproblem zwar theoretisch, scheidet jedoch bei sehr grossen Zustandsmengen aus Zeitgründen aus.

Beim *Simulated-Annealing* wird zunächst für jeden Zustand $z_j \in Z$ eine kleine Menge von r_j Nachbarzuständen $Z_j = \{z_{i_j(1)}, z_{i_j(2)}, \ldots, z_{i_j(r_j)}\}$ definiert, in der Regel bestehend aus ähnlichen Konfigurationen. Ein beliebiger Zustand $z_j \in Z$ wird als Startzustand gewählt. Nun wird versucht schrittweise von der jeweils aktuellen Konfiguration z_j zu einer zufällig gewählten Nachbarkonfiguration $z_k \in Z_j$ überzugehen. Wird dieser Zustand von der Bewertungsfunktion als besser eingestuft, das heisst $f(z_k) < f(z_j)$, findet der Übergang augenblicklich statt, andernfalls entscheidet ein Zufallsexperiment, ob der Übergang durchgeführt wird oder nicht. Die Wahrscheinlichkeit p im aktuellen Zustand z_j einen schlechteren Zustand z_k zu akzeptieren beträgt

$$p(z_j, z_k) \quad = \quad e^{-\frac{c \cdot (f(z_k) - f(z_j))}{T}} \quad , \tag{6.1}$$

hängt also von der Differenz $f(z_k) - f(z_j)$, dem sogenannten Temperaturparameter T und einer fest gewählten, positiven Konstante c ab. Die Temperatur T wird mit steigender Anzahl von Schritten immer weiter reduziert, bleibt aber stets positiv. Es wird im Laufe der Zeit also immer unwahrscheinlicher von einer besseren zu einer schlechteren Konfiguration zu kommen. Läuft der Algorithmus lange Zeit mit günstig gewählten Parametern, ist die Wahrscheinlichkeit sehr gross, ein globales Minimum der Funktion f zu erreichen.

Im Rahmen dieser Ausarbeitung kann nicht näher auf die zugrunde liegende Theorie und die praktische Anwendung von *Simulated-Annealing* im Allgemeinen eingegangen werden. Eine detailliertere Darstellung findet sich in [Grae98].

6.2 Triangulieren komplexer Löcher durch *Simulated-Annealing*

Der im Folgenden präsentierte Algorithmus verwendet zwar im Wesentlichen, nicht jedoch in allen Details das originale *Simulated-Annealing*-Verfahren. Dies liegt vor allem daran, dass *Simulated-Annealing* eine ganze Reihe von Forderungen an die Mengen Z_j der Nachbarzustände stellt, die im konkreten Fall der Oberflächenrekonstruktion nur mit erheblichem Aufwand zu überprüfen und zu erfüllen sind. Trotzdem liefert der leicht veränderte *Simulated-Annealing*-Algorithmus ausgesprochen gute Ergebnisse. Sämtliche im Dreiecksnetz vorhandenen Löcher werden durch ihn geschlossen, wobei die Form des Dreiecksnetzes in den meisten Fällen recht gut der Form des zur Punktewolke gehörigen Objekts entspricht.

6.2.1 Zustände und Nachbarzustände

Die Menge Z der Zustände wird von allen Dreiecksnetzen $\mathcal{T}_j$ gebildet für die Folgendes gilt.

- Die Eckpunkte aller Dreiecke von $\mathcal{T}_j$ entstammen der gegebenen Punktewolke, das heisst $\triangle(\mathbf{v_k}, \mathbf{v_l}, \mathbf{v_m}) \in \mathcal{T}_j \;\rightarrow\; (\mathbf{v_k} \in P \;\wedge\; \mathbf{v_l} \in P \;\wedge\; \mathbf{v_m} \in P)$.

- $\mathcal{T}_j$ ist fehlerfrei, es gibt also weder topologische Fehler noch sich schneidende Dreiecke.

Die Menge Z_j der Nachbarzustände eines Dreiecksnetzes $\mathcal{T}_j$ bilden diejenigen Dreiecksnetze, die entweder durch Einfügen eines Dreiecks und Entfernen aller dabei störenden Dreiecke oder durch Entfernen eines isolierten Dreiecks aus $\mathcal{T}_j$ erzeugt werden können. Präziser formuliert enthält die Z_j die folgenden Elemente.

- Alle Dreiecksnetze $\mathcal{T}_k$ die aus $\mathcal{T}_j$ entstehen, indem an einem Kantenpaar (Definition 5.2) ein neues Dreieck eingefügt wird und alle dabei störenden Dreiecke (*Fall 2*, *Fall 3*, *Fall 4*, *Fall 5* und *Fall 6*, Kapitel 4) entfernt werden. Das neue Dreieck darf nicht zu spitz sein (*Fall 1*, Kapitel 4).
- Alle Dreiecksnetze $\mathcal{T}_k$ die aus $\mathcal{T}_j$ entstehen, indem ein isoliertes Dreieck, ein Dreieck mit drei offenen Kanten, entfernt wird.

6.2.2 Bewertungsfunktionen

Nach Beendigung des *Simulated-Annealing* soll das dann aktuelle Dreiecksnetz die folgenden beiden Kriterien erfüllen, die durch zwei verschiedene Bewertungsfunktionen beschrieben werden. Dies ist ein weiterer Unterschied zum originalen *Simulated-Annealing*-Verfahren, bei dem es nur eine einzige Bewertungsfunktion gibt.

a) Das Dreiecksnetz ist geschlossen, besitzt also keine Löcher mehr.
   Dieses ist das bei weitem wichtigere der beiden Kriterien. Solange dieses Kriterium nicht erfüllt ist, muss mit dem *Simulated-Annealing* fortgefahren werden. Ist es jedoch erfüllt wird sofort abgebrochen.

b) Das Dreiecksnetz ist nach Möglichkeit nicht in mehrere Teile zerfallen und weist keine unnatürlich schartige oder zackige Triangulierung auf.
   Durch dieses Kriterium wird versucht, extreme Formveränderungen des Dreiecksnetzes durch das *Simulated-Annealing* auszuschliessen.

Kriterium a): Dieses Kriterium ist genau dann erfüllt, wenn das Dreiecksnetz keine offenen Kanten mehr besitzt. Je weniger offene Kanten existieren, desto weniger beziehungsweise kleinere Löcher sind im Dreiecksnetz. Kriterium a) wird durch die Bewertungsfunktion

$$f_a(\mathcal{T}) \quad = \quad \textit{Anzahl der offenen Kanten im Dreiecksnetz } \mathcal{T} \tag{6.2}$$

beschrieben. Der minimale Funktionswert von f_a ist offensichtlich null. Es ist von grossem Vorteil, dass dieser minimale Funktionswert bekannt ist. Der *Simulated-Annealing*-Algorithmus erkennt aufgrund dieser Information sofort, wenn eine globale Minimalstelle von f_a, ein Dreiecksnetz ohne Löcher, erreicht worden ist. Das *Simulated-Annealing* kann dann augenblicklich abgebrochen werden.

Kriterium b): Um zu verhindern, dass das Dreiecksnetz in Bereichen, in denen viele Löcher existieren, eine unnatürlich schartige Form erhält oder gar in mehrere Teile zerfällt, wird versucht, die mittlere Krümmung der Oberfläche so gering wie möglich zu halten.

Krümmung darf bei Dreiecksnetzen nicht im streng mathematischen Sinn verstanden werden, da die Krümmung an Kanten unendlich beträgt, während Dreiecksflächen eine Krümmung von null aufweisen. Eine Methode, die mittlere Krümmung eines Dreiecksnetzes zu bewerten, ist die Winkel zwischen den Normalen benachbarter Dreiecke zu berechnen und zu summieren, wobei die einzelnen Winkel mit den Längen der gemeinsamen Kanten gewichtet werden. Die Formel zur Berechnung der mittleren Krümmung $\overline{C}$ eines Dreiecksnetzes $\mathcal{T}$ lautet dann

$$\overline{C}(\mathcal{T}) \quad = \quad \frac{\sum_{e \in E} \alpha_e \cdot \|e\|}{\sum_{e \in E} \|e\|} \quad , \tag{6.3}$$

wobei die E die Menge aller Kanten mit zwei Dreiecken, α_e der Winkel zwischen den Normalen der beiden Dreiecke der Kante e und $\|e\|$ die Länge der Kante e ist (Abbildung 6.1). Weitere Informationen über Krümmung von Dreiecksnetzen finden sich in [DyHo00].

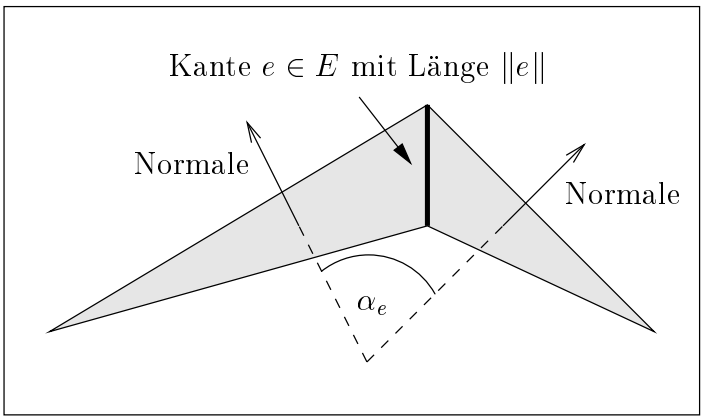


Abbildung 6.1: Berechnung der mittleren Krümmung eines Dreiecksnetzes

Als Bewertungsfunktion für Kriterium b) würde sich also die Formel für die mittlere Krümmung des Dreiecksnetzes (Gleichung 6.3) anbieten. Vergleichsweise gute Ergebnisse bei deutlich geringerem Rechenaufwand werden jedoch erzielt, wenn nur die Kanten betrachtet werden, an denen sich die Krümmung tatsächlich ändert. Zwar ist eine solche Funktion nicht mehr in der Lage, ein einzelnes Dreiecksnetz zu bewerten, sie kann jedoch dazu verwendet werden, einen Zustandsübergang zwischen zwei Netzen zu bewerten. Da beim *Simulated-Annealing* lediglich Differenzen von Bewertungen benachbarter Zustände benötigt werden, kann Kriterium b) durch die Bewertungsfunktion

$$f_b(\mathcal{T}_{nachher}) - f_b(\mathcal{T}_{vorher}) \quad = \quad \frac{\sum_{e \in E_{nachher}} \alpha_e \cdot \|e\|}{\sum_{e \in E_{nachher}} \|e\|} - \frac{\sum_{e \in E_{vorher}} \alpha_e \cdot \|e\|}{\sum_{e \in E_{vorher}} \|e\|} \tag{6.4}$$

beschrieben werden. E_{vorher} ist die Menge aller Kanten des Dreiecksnetzes $\mathcal{T}_{vorher}$ vor der Zustandsänderung mit zwei Dreiecken, die eine der folgenden drei Bedingungen erfüllen.

- Die Kante ist nach der Zustandsänderung nicht mehr vorhanden.
- Die Kante hat nach der Zustandsänderung nur noch ein Dreieck.
- Die Kante hat nach der Zustandsänderung noch immer zwei Dreiecke, ihre Krümmung hat sich jedoch verändert.

Analog dazu ist $E_{nachher}$ die Menge aller Kanten des Dreiecksnetzes $\mathcal{T}_{nachher}$ nach der Zustandsänderung mit zwei Dreiecken, die eine der folgenden drei Bedingungen erfüllen.

- Die Kante war vor der Zustandsänderung noch nicht vorhanden.
- Die Kante hatte vor der Zustandsänderung nur ein Dreieck.
- Die Kante hatte vor der Zustandsänderung ebenfalls zwei Dreiecke, ihre Krümmung hat sich jedoch verändert.

Ist eine der beiden Mengen E_{vorher} oder $E_{nachher}$ leer, tritt in der rechten Seite von Gleichung 6.4 ein unbestimmter Ausdruck auf. In solchen Fällen wird die Differenz $f_b(\mathcal{T}_{nachher}) - f_b(\mathcal{T}_{vorher}) = 0$ gesetzt.

6.2.3 Ein *Simulated-Annealing*-Schritt

Ein *Simulated-Annealing*-Schritt besteht darin, zufällig einen möglichen Zustandsübergang auszuwählen, und diesen abhängig von einem Zufallsexperiment auszuführen oder abzulehnen.

Sei $\mathcal{T}_j$ das aktuelle Dreiecksnetz. Aus der Menge Z_j der Nachbarzustände des Dreiecknetzes $\mathcal{T}_j$ wird zufällig ein Dreiecksnetz $\mathcal{T}_k$ ausgewählt. Dieses Dreiecksnetz $\mathcal{T}_k$ wird als neuer aktueller Zustand übernommen, wenn es von den beiden Bewertungsfunktionen f_a und f_b akzeptiert wird. f_a akzeptiert das Dreiecksnetz $\mathcal{T}_k$ mit einer Wahrscheinlichkeit von

$$p_a(\mathcal{T}_j, \mathcal{T}_k) \quad = \quad e^{-\frac{c_a \cdot (f_a(\mathcal{T}_k) - f_a(\mathcal{T}_j))}{T}} \quad , \tag{6.5}$$

f_b akzeptiert das Dreiecksnetz $\mathcal{T}_k$ mit einer Wahrscheinlichkeit von

$$p_b(\mathcal{T}_j, \mathcal{T}_k) \quad = \quad e^{-\frac{c_b \cdot (f_b(\mathcal{T}_k) - f_b(\mathcal{T}_j))}{T}} \quad . \tag{6.6}$$

Wegen $p_a(\mathcal{T}_j, \mathcal{T}_k) \geq 1$ beziehungsweise $p_b(\mathcal{T}_j, \mathcal{T}_k) \geq 1$ genau dann, wenn $f_a(\mathcal{T}_k) - f_a(\mathcal{T}_j) \leq 0$ beziehungsweise $f_b(\mathcal{T}_k) - f_b(\mathcal{T}_j) \leq 0$, werden Übergänge zu besseren Zuständen immer akzeptiert, während Übergänge zu schlechteren Zuständen stets mit einer gewissen Wahrscheinlichkeit

abgelehnt werden. Die beiden Konstanten c_a und c_b können im Bereich der positiven Zahlen frei gewählt werden. Durch sie kann der Benutzer steuern, mit welcher Wahrscheinlichkeit ein schlechterer Zustand akzeptiert wird, und welches Gewicht die beiden Bewertungsfunktionen f_a (bewertet die Anzahl der offenen Kanten) und f_b (bewertet die Krümmung der Oberfläche) erhalten. Sinnvolle, durch Experimente ermittelte Werte sind $c_a = 1.0$ und $c_b = 3.0$.

Pseudocode eines *Simulated-Annealing*-Schrittes: In den Zeilen 2 bis 13 wird zufällig ein Element $\mathcal{T}_k$ aus der Menge der Nachbarzustände Z_j des Dreiecksnetzes $\mathcal{T}_j$ ausgewählt. Die Abfrage in Zeile 6 dient dazu, Endlosschleifen bei leerem Z_j zu vermeiden. Eigentlich müsste vor dem Abbruch in den Zeilen 7 und 8 überprüft werden, ob die Menge Z_j tatsächlich leer ist, und falls nicht, ein Rücksprung nach 2 erfolgen. Der für einen solchen Test benötigte Zeitaufwand ist jedoch relativ hoch. Andererseits ist es durchaus vertretbar, hin und wieder einen *Simulated-Annealing*-Schritt auszulassen. Die Zeilen 14 bis 18 beziehungsweise 19 bis 23 beenden den *Simulated-Annealing* Schritt, falls der Übergang zum Dreiecksnetz $\mathcal{T}_k$ von der Bewertungsfunktion f_a beziehungsweise f_b abgelehnt wird.

6.2.4 Das vollständige *Simulated-Annealing*-Verfahren

Das vollständige *Simulated-Annealing*-Verfahren besteht aus einer festen, vom Benutzer vorzugebenden Anzahl s von Phasen, wobei jede Phase aus einer festen, ebenfalls vom Benutzer vorzugebenden Anzahl t von Schritten besteht.

Die verschiedenen Phasen unterscheiden sich lediglich durch den Wert der Temperatur T. Werden die Phasen absteigend von $s-1$ bis 0 durchnumeriert, lautet die Formel für die Temperatur T in Phase k

$$T(k) \quad = \quad (T_1)^k \qquad k = s-1, \ldots, 0 \quad , \tag{6.7}$$

wobei T_1, die Temperatur in Phase 1, ein vom Benutzer vorzugebender Wert grösser eins ist. Durch die Temperatur T_1 kann gesteuert werden, wie stark beziehungsweise schwach die Wahrscheinlichkeit, schlechtere Zustände zu akzeptieren, im Lauf der Phasen abnimmt.

Das *Simulated-Annealing*-Verfahren wird abgebrochen, wenn keine offenen Kanten mehr vorhanden sind oder aber wenn alle Phasen vollständig durchlaufen wurden. In letzterem Fall sollte das *Simulated-Annealing* natürlich erneut gestartet werden, eventuell mit einer höheren Temperatur T_1 oder einer grösseren Anzahl s von Phasen oder t von Schritten.

In äusserst seltenen Fällen treten Konfigurationen auf, von denen aus es nicht mehr möglich ist, über einen Pfad von Nachbarzuständen ein geschlossenes Dreiecksnetz zu erreichen. Es handelt sich dabei meist um kleine Löcher, gebildet von drei oder vier offenen Kanten, bei denen immer wieder dieselben Dreiecke eingefügt und entfernt werden. Die Ursachen dafür können zum Beispiel numerische Probleme bei *Test 4* (Abschnitt 4.4) oder *Test 5* (Abschnitt 4.5), oder eine ungünstige Lage der Punkte bei einem zu grossen, minimalen Zwi-

1	$\mathcal{T}_j$ = *Aktuelles Dreiecksnetz*.
2	Wähle gleichverteilt zufällig eine offene Kante.
3	Wähle gleichverteilt zufällig einen der beiden Endpunkte dieser offenen Kante.
4	Wähle gleichverteilt zufällig eine andere, an diesem Endpunkt hängende, offene Kante.
5	IF (Die beiden gewählten Kanten gehören zu zwei verschiedenen Dreiecken, sie bilden also ein Kantenpaar):
6	IF (Das durch das Kantenpaar vorgegebene Dreieck ist zu spitz (*Fall 1*)):
7	Behalte $\mathcal{T}_j$ als aktuelles Dreiecksnetz.
8	Beende den *Simulated-Annealing*-Schritt.
9	Erzeuge $\mathcal{T}_k$ aus $\mathcal{T}_j$ durch Einfügen eines Dreiecks an der durch das Kantenpaar vorgegeben Stelle und Entfernen aller störenden Dreiecke (*Fall 2*, *Fall 3*, *Fall 4*, *Fall 5* und *Fall 6*).
10	ELSE IF (Die beiden gewählten Kanten gehören zu einem isolierten Dreieck):
11	Erzeuge $\mathcal{T}_k$ aus $\mathcal{T}_j$ durch Löschen dieses isolierten Dreiecks.
12	ELSE IF (Die beiden gewählten Kanten gehören zu einem nicht-isolierten Dreieck):
13	GOTO 2.
14	IF ($f_a(\mathcal{T}_k) > f_a(\mathcal{T}_j)$):
15	Ermittle gleichverteilt eine Zufallszahl $p \in [0,1]$.
16	IF ($p > p_a(\mathcal{T}_j, \mathcal{T}_k) = e^{-\frac{c_a \cdot (f_a(\mathcal{T}_k) - f_a(\mathcal{T}_j))}{T}}$):
17	Behalte $\mathcal{T}_j$ als aktuelles Dreiecksnetz.
18	Beende den *Simulated-Annealing*-Schritt.
19	IF ($f_b(\mathcal{T}_k) > f_b(\mathcal{T}_j)$):
20	Ermittle gleichverteilt eine Zufallszahl $p \in [0,1]$.
21	IF ($p > p_b(\mathcal{T}_j, \mathcal{T}_k) = e^{-\frac{c_a \cdot (f_b(\mathcal{T}_k) - f_b(\mathcal{T}_j))}{T}}$):
22	Behalte $\mathcal{T}_j$ als aktuelles Dreiecksnetz.
23	Beende den *Simulated-Annealing*-Schritt.
24	Übernehme $\mathcal{T}_k$ als aktuelles Dreiecksnetz.
25	Beende den *Simulated-Annealing*-Schritt.

Pseudocode eines *Simulated-Annealing*-Schrittes

schenwinkel bei *Test 2* (Abschnitt 4.2) sein. Als einfacher, jedoch äusserst wirkungsvoller Trick hat sich in solchen Fällen das Vergrössern von vorhandenen Löchern erwiesen. Dabei werden einfach diejenigen Dreiecke entfernt, die im aktuellen Netz mindestens eine offene Kante besitzen (Abbildung 6.2; die dunklen Dreiecke werden entfernt). Da es für grössere Löcher

meist eine ganze Reihe von möglichen Triangulierungen gibt, findet der zufallsgesteuerte Prozess des *Simulated-Annealing* auf diese Weise irgendwann eine Möglichkeit das Dreiecksnetz zu schliessen.

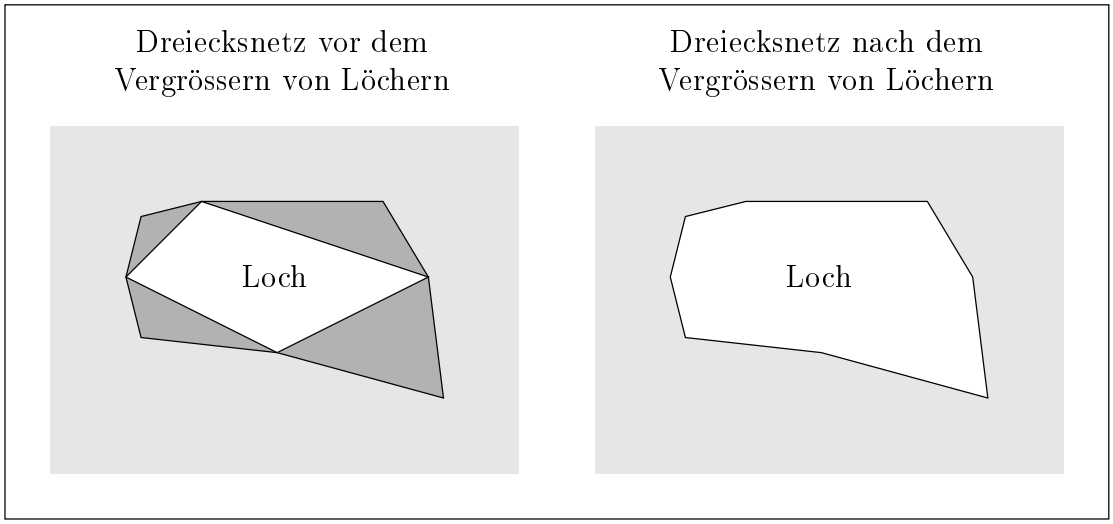


Abbildung 6.2: Vergrössern von Löchern

Gute Ergebnisse wurden mit folgender Strategie erzielt. Als Parameter werden $T_1 = 1.25$, $s = 1$ und $t = 10 \cdot$ *Anzahl der offenen Kanten* gewählt. Werden die komplexen Löcher im ersten Durchlauf nicht vollständig geschlossen, wird die Anzahl der Phasen solange um 1 erhöht, bis keine offenen Kanten mehr vorhanden sind. Konnte das Dreiecksnetz, obwohl die Anzahl s der Phasen bereits auf 10 erhöht wurde, noch immer nicht geschlossen werden, werden die vorhandenen Löcher, wie weiter oben beschrieben, vergrössert und s wird auf 6 zurückgesetzt. Erreicht s erneut den Wert 10 (dieser Fall ist im Laufe der vielen Tests im Rahmen dieser Arbeit niemals aufgetreten), werden die Löcher zweimal vergrössert, und s wird wieder auf 6 zurückgesetzt. Dieses Schema wird solange fortgesetzt, bis das Dreiecksnetz keine Löcher mehr aufweist.

Pseudocode des vollständigen *Simulated-Annealing*-Verfahrens: Die WHILE-Schleife in Zeile 3 startet das *Simulated-Annealing*-Verfahren solange immer wieder von vorne, bis das Dreiecksnetz vollständig geschlossen wurde. Die äussere FOR-Schleife (Zeile 5) bewirkt das Durchlaufen der einzelnen Phasen, die innere FOR-Schleife (Zeile 7) das Ausführen einer entsprechenden Anzahl von *Simulated-Annealing*-Schritten. Die Zeilen 14 bis 18 sorgen bei Problemfällen für das Vergrössern der vorhandenen Löcher.

```
1   c_a = 1.0, c_b = 3.0, T_1 = 1.25, s = 1.
2   h = 0.
3   WHILE (Dreiecksnetz besitzt offene Kanten):
4       t = 10 · Anzahl der offenen Kanten.
5       FOR (i = s − 1, s − 2, ..., 0):
6           T = (T_1)^i.
7           FOR (j = 1, 2, ..., t):
8               IF (Dreiecksnetz besitzt keine offenen Kanten mehr)
9                   BREAK.
10              Führe einen Simulated-Annealing-Schritt aus.
11          IF (Dreiecksnetz besitzt keine offenen Kanten mehr)
12              BREAK.
13      s = s + 1.
14      IF (s == 11):
15          s = 6.
16          h = h + 1.
17          FOR (i = 1, 2, ..., h):
18              Vergrössere die vorhandenen Löcher.
```

Pseudocode des vollständigen *Simulated-Annealing*-Verfahrens

6.3 Laufzeitverhalten

Das Laufzeitverhalten zum Triangulieren komplexer Löcher wird von sehr vielen, teilweise unvorhersehbaren oder schwer zu bewertenden Faktoren beeinflusst. Zu diesen Faktoren zählen die Anzahl der offenen Kanten und die Form des Dreiecksnetzes zu Beginn des *Simulated-Annealing*, die vom Benutzer gewählten Parameter des *Simulated-Annealing*, und die zufällig günstigen oder ungünstigen Ausgänge der Zufallsexperimente vor möglichen Übergangen zu schlechteren Zuständen. Eine Aussage über das Laufzeitverhalten dieses Schrittes vom theoretischen Standpunkt aus ist daher nahezu unmöglich.

In zahlreichen Experimenten hat sich jedoch gezeigt, dass der Zeitverbrauch des *Simulated-Annealing* bei den meisten Punktewolken den Zeitverbrauch der anderen Schritte des vollständigen Rekonstruktionsalgorithmus zumindest nicht wesentlich übersteigt. Es soll jedoch nicht unerwähnt bleiben, dass Punktewolken von realen dreidimensionalen Objekten existieren, bei denen die Lage der Punkte derart ungünstig ist, dass das *Simulated-Annealing*

ein Vielfaches der Zeit verbraucht, die die anderen Schritte benötigen. Experimentelle Ergebnisse sind in Kapitel 9 zu finden.

Kapitel 7

Einfügen isolierter Punkte (*Schritt 5*)

Definition 7.1 *Ein isolierter Punkt ist ein Punkt der gegebenen Punktewolke, der weder Eckpunkt eines Dreiecks, noch Endpunkt einer Kante ist.*

Beim Triangulieren komplexer Löcher (Kapitel 6) kommt es hin und wieder vor, dass ein *Simulated-Annealing*-Schritt alle Dreiecke eines Punktes und damit auch alle seine Kanten entfernt. Solche Punkte können durch das *Simulated-Annealing*-Verfahren nicht mehr in das Dreiecksnetz integriert werden. Bei diesem Schritt geht es darum, diese isolierten Punkte wieder in das Netz einzufügen.

Dieser Schritt setzt ein fehlerfreies und geschlossenes Dreiecksnetz voraus. Ein durch *Schritt 1*, *Schritt 2*, *Schritt 3* und *Schritt 4* erzeugtes Dreiecksnetz erfüllt diese Voraussetzungen.

7.1 Prinzip des Einfügens isolierter Punkte

Die isolierten Punkte werden in beliebiger Reihenfolge in das Dreiecksnetz eingefügt.

Sei $\mathbf{v_j}$ derjenige isolierte Punkt, der als nächstes eingefügt werden soll.

In der Nähe des isolierten Punktes $\mathbf{v_j}$ wird durch Entfernen eines oder mehrerer Dreiecke ein Loch im bestehenden Dreiecksnetz erzeugt. Dabei muss darauf geachtet werden, dass kein weiterer isolierter Punkt entsteht und das Loch nur einen einzigen Rand besitzt (Abbildung 7.1).

Nun wird für jede durch dieses Loch entstandene offene Kante ein neues Dreieck in das Netz eingefügt. Die Eckpunkte dieses Dreiecks sind die beiden Endpunkte der offenen Kante und der isolierte Punkt $\mathbf{v_j}$. Das erzeugte Loch wird dadurch wieder geschlossen, und der isolierte Punkt $\mathbf{v_j}$ in das Dreiecksnetz integriert (Abbildung 7.2).

Unter den vielen möglichen Löchern, die erzeugt werden können, um den isolierten Punkt $\mathbf{v_j}$ einzufügen, ist dasjenige zu wählen, das die folgenden beiden Kriterien erfüllt.

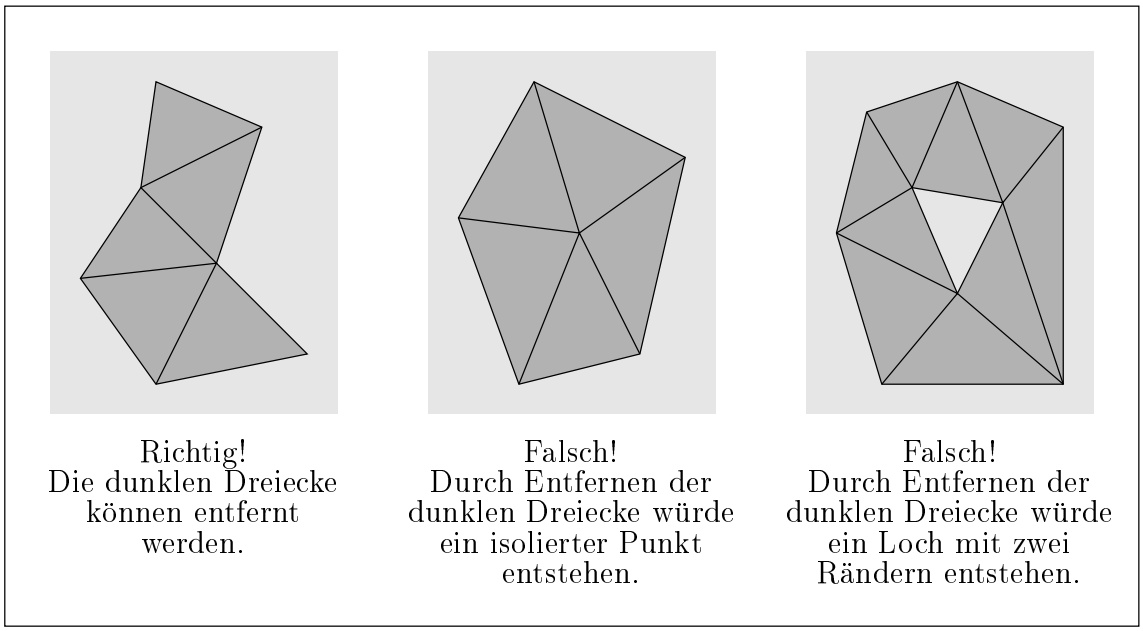


Abbildung 7.1: Erzeugen eines Loches

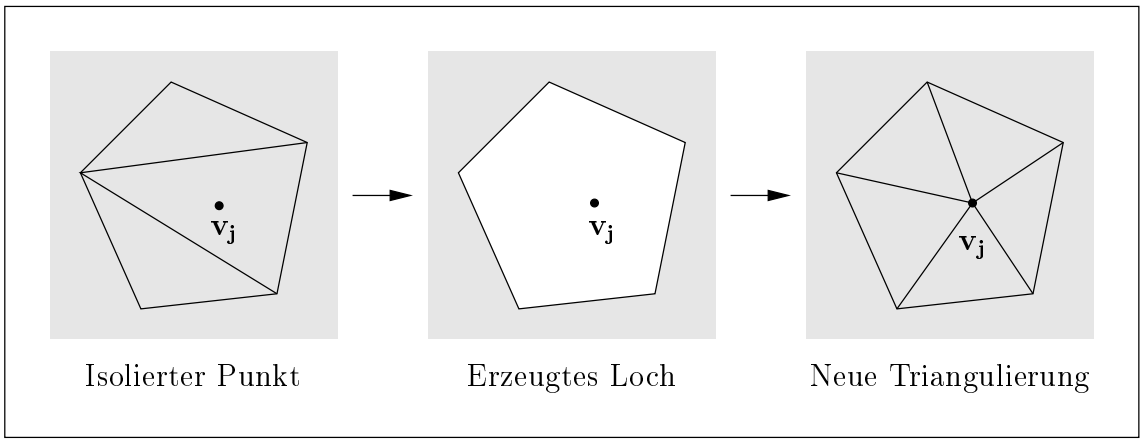


Abbildung 7.2: Einfügen eines isolierten Punktes

- Beim Einfügen der neuen Dreiecke kommt es nicht zu Konflikten mit dem bestehenden Netz (*Fall 1*, *Fall 2*, *Fall 4* und *Fall 5*, Kapitel 4; die Topologie des Dreiecksnetzes kann durch das Einfügen eines isolierten Punktes nicht verletzt werden, *Fall 3* und *Fall 6* müssen daher nicht überprüft werden).

- Der isolierte Punkt $\mathbf{v_j}$ wird möglichst glatt in das Dreiecksnetz eingefügt. Mathematisch ausgedrückt bedeutet das, dass die mittlere Krümmung $\overline{C}$ des Dreiecksnetzes (Gleichung 6.3, Kapitel 6) durch diese Wahl des Loches minimiert wird.

Das Loch zu finden, das diese beiden Kriterien erfüllt, ist wegen der enorm grossen Anzahl möglicher Löcher extrem zeitaufwendig. Um das eben beschriebene Verfahren praktisch verwenden zu können, muss es erst derart modifiziert werden, dass sein Laufzeitverhalten die Laufzeitverhalten der anderen Schritte des vollständigen Rekonstruktionsalgorithmus zumindest nicht wesentlich übersteigt, und es dabei immer noch gute Ergebnisse liefert.

7.2 Verbesserung des Laufzeitverhaltens

Um den Zeitverbrauch des eben beschriebenen Verfahrens drastisch zu reduzieren, werden die im Folgenden beschriebenen Massnahmen ergriffen.

7.2.1 Einschränkung der Anzahl der zu betrachtenden Löcher

Es werden nur noch Löcher betrachtet, die durch Entfernen von einem oder von zwei Dreiecken entstehen (Abbildung 7.3).

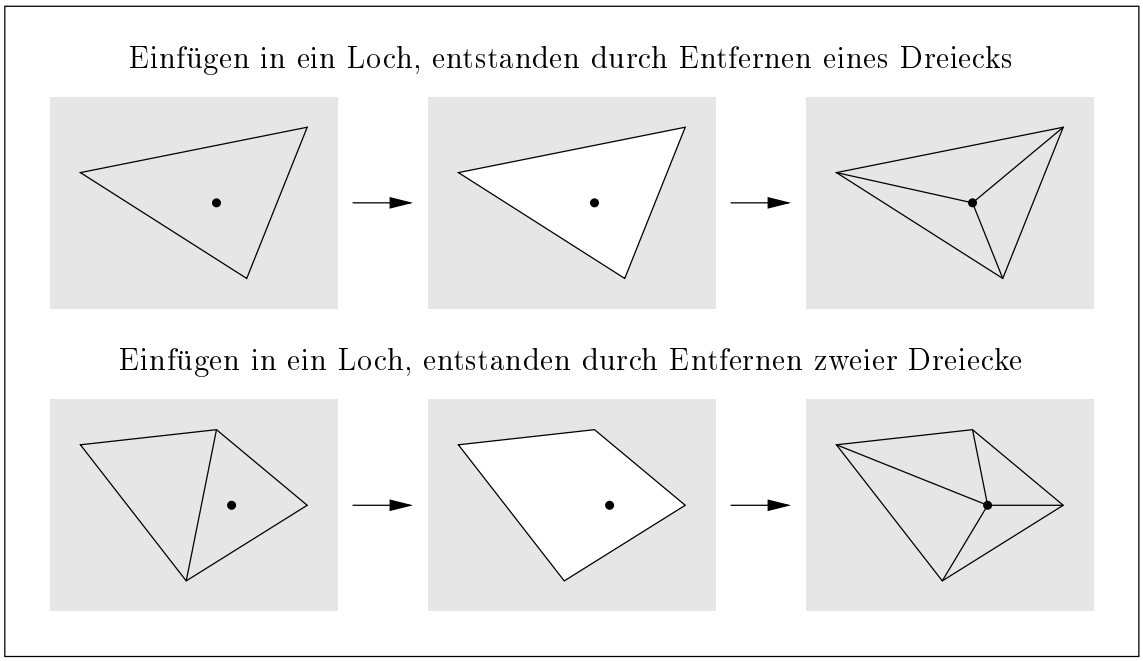


Abbildung 7.3: Einfügen in ein oder zwei Dreiecke grosse Löcher

Die durch diese Einschränkung erzielten Ergebnisse sind noch immer relativ gut, da die meisten isolierten Punkte bei Verwendung des in Abschnitt 7.1 beschriebenen Verfahrens ohnehin in kleine Löcher, entstanden durch Entfernen weniger Dreiecke, eingefügt werden würden. Darüber hinaus können beim Glätten des Dreiecksnetzes (Kapitel 8) die gleichen Konfigurationen entstehen, die durch Einfügen eines isolierten Punktes in ein grosses Loch erzeugt

werden würden.

Sich nur auf Löcher zu beschränken, die durch Entfernen eines Dreiecks entstehen, ist nicht ausreichend. Es kommt dabei relativ häufig zu Problemen mit isolierten Punkten, die genau auf oder in unmittelbarer Nähe einer Kante liegen. Diese Punkte können meistens nicht in ein an diese Kante angrenzendes Loch eingefügt werden, da mindestens eines der neuen Dreiecke einen zu spitzen Winkel hat (*Fall 1*, Kapitel 4) und deshalb nicht in das Dreiecksnetz eingefügt werden darf (Abbildung 7.4). Nicht an diese Kante angrenzende Löcher scheiden in der Regel ebenfalls aus, da es wegen der Nähe des isolierten Punktes zur Oberfläche des Dreiecksnetzes fast immer zu zu kleinen Zwischenwinkeln (*Fall 2*) oder zu Schnitten zwischen bestehenden und neuen Dreiecken (*Fall 4* und *Fall 5*) kommt.

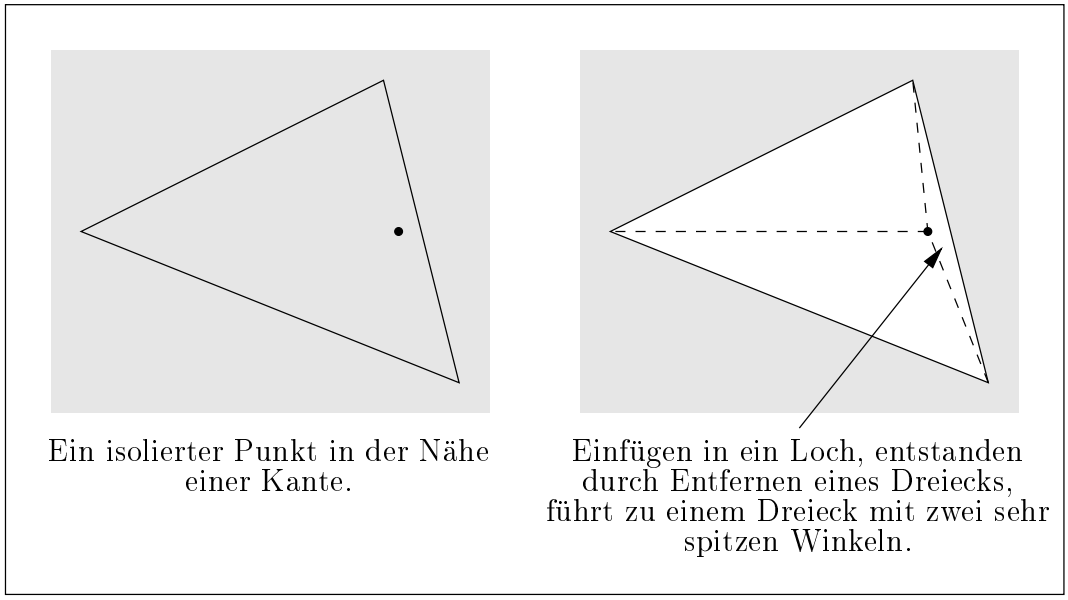


Abbildung 7.4: Probleme bei einem isolierten Punkt in der Nähe einer Kante

Es ist möglich, Dreiecksnetze zu konstruieren, bei denen das Einfügen von isolierten Punkten in Löcher, entstanden durch Entfernen von einem oder von zwei Dreiecken, stets zu Konflikten mit dem bestehenden Netz führt. Solche Konstrukte sind extrem theoretischer Natur und werden in der Praxis wohl so gut wie nie auftreten. Im Lauf der vielen Tests im Rahmen dieser Studienarbeit wurde ein solcher Fall zumindest nie beobachtet. Dennoch soll nicht unerwähnt bleiben, dass in einem derartigen Ausnahmefall das Einfügen eines isolierten Punktes oder mehrerer mit dem hier beschriebenen Verfahren nicht möglich ist.

7.2.2 Verwendung eines einfacheren Krümmungsmasses

Anstatt die mittlere Krümmung $\overline{C}$ des gesamten Dreiecksnetzes zu betrachten, wird ein sehr viel einfacher zu berechnendes Mass $\tilde{C}$ herangezogen, die Summe der Winkel zwischen den

Normalen benachbarter Dreiecke im neuen Dreiecksfächer. $\tilde{C}$ bewertet wie spitz beziehungsweise glatt der um den isolierten Punkt entstandene Dreiecksfächer ist. Die Formel zur Berechnung von $\tilde{C}$ lautet

$$\tilde{C}(Dreiecksfächer) = \sum_{e \in E_{Dreiecksfächer}} \alpha_e \quad . \tag{7.1}$$

Dabei ist $E_{Dreiecksfächer}$ die Menge der inneren Kanten des neuen Dreiecksfächers und α_e der Winkel zwischen den Normalen der beiden Dreiecke der Kante e.

Von allen Löchern, die durch Entfernen von einem oder von zwei Dreiecken erzeugt werden können, und in die der isolierte Punkt konfliktfrei eingefügt werden kann, wird dasjenige gewählt, bei dem $\tilde{C}$ minimal wird. Da es sehr viel schneller geht, $\tilde{C}$ für einen Dreiecksfächer zu berechnen als zu überprüfen, ob es beim Einfügen der Dreiecke dieses Fächers zu Konflikten mit dem bestehenden Netz kommt, wird zuerst für alle zu betrachtenden Löcher $\tilde{C}$ ermittelt. Anschliessend werden die Löcher in eine Liste eingeordnet und gemäss ihren $\tilde{C}$-Werten aufsteigend sortiert. Das erste Loch in der Liste, bei dem es beim Einfügen der neuen Dreiecke nicht zu Konflikten kommt, ist das, in das der isolierte Punkt eingefügt wird.

In zahlreichen Experimenten hat sich gezeigt, dass sich die Qualität der Ergebnisse bei Verwendung von $\tilde{C}$ anstatt von $\overline{C}$ kaum verändert.

Pseudocode für das Einfügen isolierter Punkte: Die Schleife in Zeile 1 fügt die isolierten Punkte nacheinander in das Dreiecksnetz ein. In den Zeilen 2 bis 8 wird eine Liste aller möglichen ein und zwei Dreiecke grossen Löcher erzeugt und nach den jeweiligen $\tilde{C}$-Werten aufsteigend sortiert. In den Zeilen 9 bis 12 wird dasjenige Loch ausgewählt, bei dem keine Konflikte beim Einfügen der neuen Dreiecke auftreten, und in das der isolierte Punkt am „glattesten“ eingefügt werden kann. Treten bei allen Löchern Konflikte auf, wird eine Warnung ausgegeben, und mit dem nächsten isolierten Punkt fortgefahren (Zeilen 13 und 14). In den Zeilen 15 bis 20 wird das ausgewählte Loch erzeugt, und der isolierte Punkt in dieses eingefügt.

7.3 Laufzeitverhalten

Das Laufzeitverhalten hängt linear von der Anzahl m der isolierten Punkte ab.

Ein geschlossenes Dreiecksnetz mit n Punkten besitzt etwa $2n$ Dreiecke und $3n$ Kanten (Eulerscher Polyedersatz). Die Anzahl der möglichen Löcher, bestehend aus einem Dreieck, entspricht der Anzahl der Dreiecke des Netzes, beträgt also ungefähr $2n$. Die Anzahl der möglichen Löcher, bestehend aus zwei Dreiecken, entspricht der Anzahl der Kanten des Netzes, beträgt also ungefähr $3n$. Für jeden isolierten Punkt wird daher eine Liste von Löchern der Grössenordnung n aufgebaut.

```
1   FOR (Alle isolierten Punkte v_iso):
2       liste = Leere Liste von Löchern.
3       FOR (Alle Dreiecke △(v_j, v_k, v_l)):
4           Füge das Loch, das durch Entfernen von △(v_j, v_k, v_l) entstehen würde, in liste
                ein.
5       FOR (Alle Dreieckspaare (△(v_j, v_k, v_l), △(v_j, v_k, v_m))):
6           Füge das Loch, das durch Entfernen der Dreiecke △(v_j, v_k, v_l) und
                △(v_j, v_k, v_m) entstehen würde, in liste ein.
7       Berechne C̃ für alle Löcher in liste.
8       Sortiere liste aufsteigend nach C̃.
9       n = Anzahl der Elemente von liste.
10      FOR (loch = liste[1], liste[2], ..., liste[n]):
11          IF (Einfügen von v_iso in loch führt nicht zu Konflikten (Fall 1, Fall 2, Fall 4
                und Fall 5)):
12              GOTO 15.
13      PRINT ("Warnung! v_iso konnte nicht eingefügt werden!").
14      CONTINUE.
15      IF (loch besteht aus einem Dreieck △(v_j, v_k, v_l)):
16          Entferne das Dreieck △(v_j, v_k, v_l) aus dem Netz.
17          Füge die Dreiecke △(v_iso, v_j, v_k), △(v_iso, v_k, v_l) und △(v_iso, v_l, v_j) in das
                Netz ein.
18      ELSE IF (loch besteht aus den Dreiecken △(v_j, v_k, v_l) und △(v_j, v_k, v_m)):
19          Entferne die Dreiecke △(v_j, v_k, v_l) und △(v_j, v_k, v_m) aus dem Netz.
20          Füge die Dreiecke △(v_iso, v_j, v_l), △(v_iso, v_l, v_k), △(v_iso, v_k, v_m) und
                △(v_iso, v_m, v_j) in das Netz ein.
```

Pseudocode für das Einfügen isolierter Punkte

Sortieren einer Liste der Grössenordnung n weist ein Laufzeitverhalten von

$$T_{Sortieren}(n) \quad = \quad O(n \cdot \log n) \tag{7.2}$$

auf.

Der Zeitverbrauch beim Überprüfen, ob bei einem bestimmten Loch das Einfügen der neuen

Dreiecke zu Konflikten mit dem bestehenden Netz führt, wird von *Test 5* dominiert. Das Laufzeitverhalten für diese Tests beträgt demnach

$$T_{\textit{Test 1, Test 2, Test 4, Test 5}}(n) \quad = \quad O(n) \tag{7.3}$$

(Unterkapitel 4.8).

Es hat sich gezeigt, dass in den allermeisten Fällen in eines der Löcher am Anfang der Liste konfliktfrei eingefügt werden kann. Das liefert ein experimentell ermitteltes mittleres Laufzeitverhalten von

$$\overline{T}_{\textit{Einfügen isolierter Punkte}}(n, m) \quad = \quad O(n \cdot \log n \cdot m) \quad . \tag{7.4}$$

Im schlechtesten Fall muss beim Testen auf Konflikte immer die ganze Liste durchlaufen werden. Dies liefert eine, wohl in den meisten Fällen weit vom tatsächlichen Laufzeitverhalten entfernte Worst-Case-Abschätzung von

$$T_{\textit{Einfügen isolierter Punkte, Worst Case}}(n, m) \quad = \quad O(n^2 \cdot m) \quad . \tag{7.5}$$

Kapitel 8

Glätten des erzeugten Dreiecksnetzes (*Schritt 6*)

Bei diesem Schritt geht es darum, das erzeugte Dreiecksnetz zu glätten. Einerseits werden unnötige Zacken und Scharten aus dem Netz entfernt, andererseits werden Kanten deutlicher herausgearbeitet. Das Dreiecksnetz soll dadurch ein gefälligeres Aussehen erhalten. Die Anzahl und die Lage der Punkte der gegebenen Punktewolke bleiben dabei unverändert.

Dieser Schritt setzt ein fehlerfreies und geschlossenes Dreiecksnetz voraus. Ein durch *Schritt 1*, *Schritt 2*, *Schritt 3*, *Schritt 4* und *Schritt 5* erzeugtes Dreiecksnetz erfüllt diese Voraussetzungen.

8.1 Das Verfahren von *Dyn*, *Hormann*, *Kim* und *Levin*

Das in diesem Abschnitt präsentierte Verfahren entstammt [DyHo00].

Durch eine Reihe von Kanten-Vertauschoperationen, sogenannten Edge-Swaps, wird versucht, die Gesamtkrümmung des gegebenen Dreiecksnetzes zu minimieren. Dies führt in den meisten Fällen zu deutlich glatteren Dreiecksnetzen.

8.1.1 Krümmung bei Dreiecksnetzen

Wie bereits in Unterabschnitt 6.2.2 erläutert, darf Krümmung bei Dreiecksnetzen nicht im streng mathematischen Sinn verstanden werden, da die Krümmung an Kanten unendlich beträgt, während Dreiecksflächen eine Krümmung von null aufweisen. Eine Methode, die Gesamtkrümmung C eines Dreiecksnetzes zu berechnen, ist die Winkel zwischen den Normalen benachbarter Dreiecke zu bestimmen, mit den Längen der gemeinsamen Kanten zu gewichten und zu summieren. Die Formel zur Berechnung der Gesamtkrümmung C eines Dreiecksnetzes $\mathcal{T}$ ist der Formel zur Berechnung der mittleren Krümmung sehr ähnlich (Gleichung 6.3). Sie lautet

$$C(\mathcal{T}) \quad = \quad \sum_{e \in E} \alpha_e \cdot \|e\| \quad , \tag{8.1}$$

wobei die E die Menge aller Kanten, α_e der Winkel zwischen den Normalen der beiden Dreiecke der Kante e und $\|e\|$ die Länge der Kante e ist (Abbildung 8.1). Weitere Informationen zu Krümmung bei Dreiecksnetzen sind in [DyHo00] zu finden.

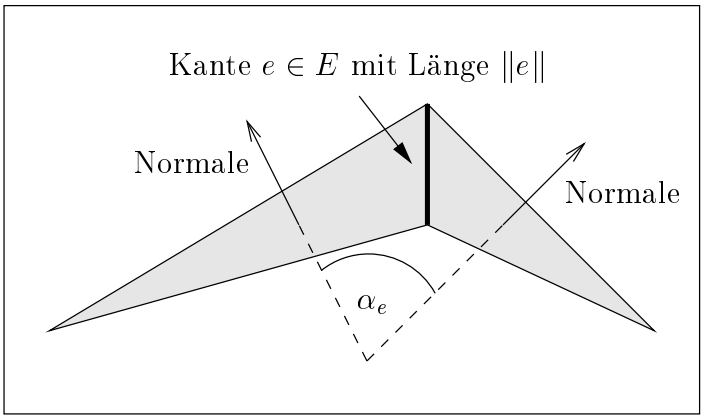


Abbildung 8.1: Berechnung der Gesamtkrümmung eines Dreiecksnetzes

8.1.2 Edge-Swaps

Ein Edge-Swap ist eine Kanten-Vertauschoperation. Die gemeinsame Kante $e(\mathbf{v_j}, \mathbf{v_k})$ zweier aneinandergrenzender Dreiecke $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ und $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_m})$ wird entfernt und durch die neue Kante $e(\mathbf{v_l}, \mathbf{v_m})$ ersetzt. Dadurch ändert sich natürlich die bestehende Triangulierung. Die Dreiecke $\triangle(\mathbf{v_j}, \mathbf{v_l}, \mathbf{v_m})$ und $\triangle(\mathbf{v_k}, \mathbf{v_l}, \mathbf{v_m})$ ersetzen die Dreiecke $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_l})$ und $\triangle(\mathbf{v_j}, \mathbf{v_k}, \mathbf{v_m})$ (Abbildung 8.2).

Um topologische Fehler zu vermeiden darf ein Edge-Swap nicht mit einer der inneren Kanten eines aus drei Dreiecken bestehenden Fächers ausgeführt werden. Dies hätte nämlich zur Folge, dass danach sowohl ein Dreieck als auch eine Kante doppelt im Netz vorhanden ist (Abbildung 8.3).

Edge-Swaps können ausserdem zu unerwünscht spitzen oder sich schneidenden Dreiecken führen. Ein Edge-Swap darf nur ausgeführt werden, wenn durch ihn keine *Fall-1-Dreiecke*, *Fall-2-Dreiecke*, *Fall-4-Dreiecke* und *Fall-5-Dreiecke* entstehen.

8.1.3 Minimieren der Krümmung

Der Algorithmus von *Dyn*, *Hormann*, *Kim* und *Levin* versucht durch eine Reihe von Edge-Swaps die Gesamtkrümmung C des gegebenen Dreiecksnetzes $\mathcal{T}$ zu minimieren.

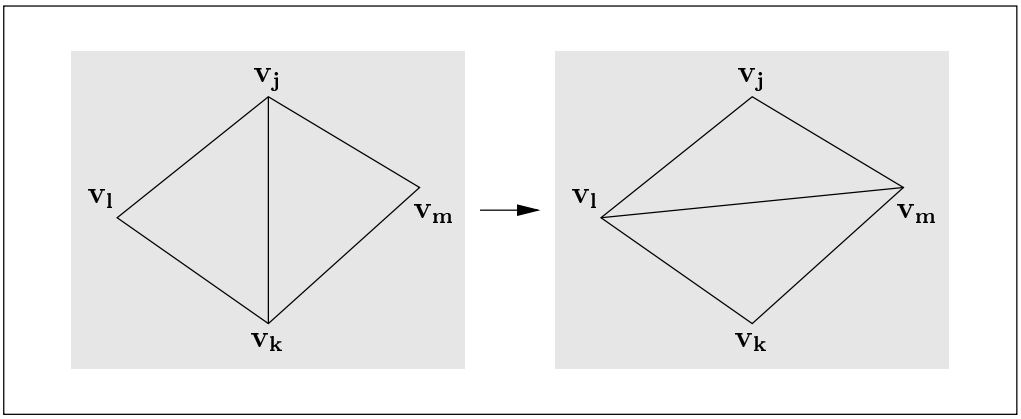


Abbildung 8.2: Ein Edge-Swap

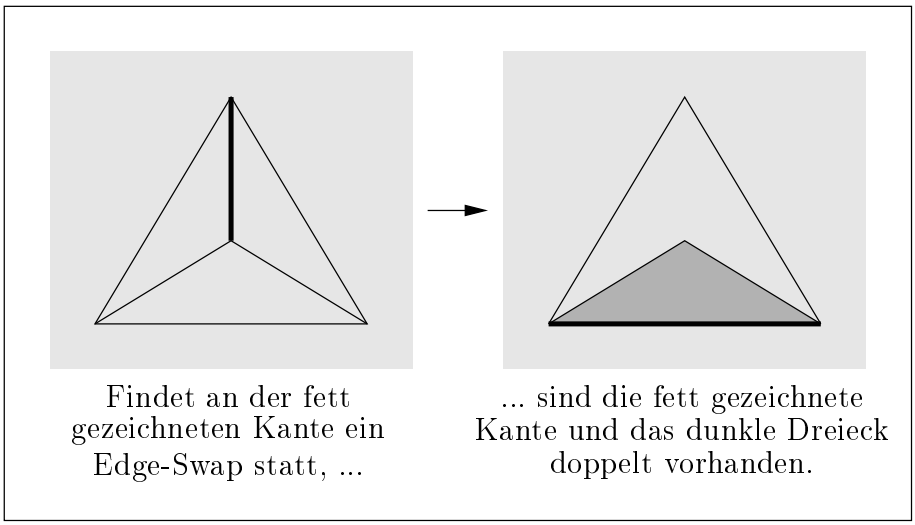
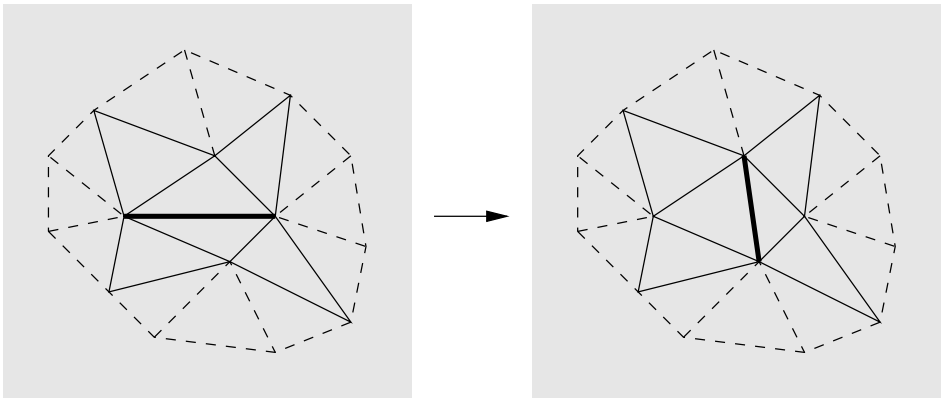


Abbildung 8.3: Ein nicht erlaubter Edge-Swap

Zuerst werden für alle möglichen Edge-Swaps s die jeweiligen Änderungen der Gesamtkrümmung $\Delta C(s) = C(\mathcal{T}_{nachher}) - C(\mathcal{T}_{vorher})$ berechnet. Die Edge-Swaps werden dann in eine Liste eingeordnet und nach ihren ΔC-Werten aufsteigend sortiert. Der erste Edge-Swap in der Liste ist also derjenige, der die Gesamtkrümmung C des Dreiecksnetzes am stärksten reduziert.

Es wird nun solange der jeweils erste Edge-Swap in der Liste ausgeführt, bis keine Edge-Swaps mehr existieren, die die Gesamtkrümmung C des Dreiecksnetzes verkleinern. Da die Liste stets sortiert gehalten wird, ist dies genau dann der Fall, wenn der ΔC-Wert des ersten Listenelements grösser oder gleich null ist. Nach jedem durchgeführten Edge-Swap muss die Liste der Edge-Swaps aktualisiert werden. Dies beinhaltet Hinzufügen neuer und Entfernen nicht mehr möglicher Edge-Swaps, Aktualisieren verschiedener ΔC-Werte und Wiederherstel-

len der Sortierung. Beim Aktualisieren der ΔC-Werte reicht es aus, lediglich die Kanten der beiden neuen und der daran angrenzenden Dreiecke zu betrachten. Nur bei den diesen Kanten zugeordneten Edge-Swaps können sich die Krümmungsdifferenzen ΔC verändert haben (Abbildung 8.4).



Nach dem Edge-Swap der fett gezeichneten Kante können sich nur die Krümmungsdifferenzen ΔC der Edge-Swaps verändert haben, die den nicht-gestrichelten Kanten zugeordnet sind.

Abbildung 8.4: Veränderung von Krümmungsdifferenzen ΔC nach einem Edge-Swap

Durch diesen Algorithmus wird mit Sicherheit eine lokale Minimalstelle, nicht notwendigerweise jedoch eine globale Minimalstelle der Gesamtkrümmung C des Dreiecksnetzes $\mathcal{T}$ erreicht. Dennoch sind die Ergebnisse dieses Glättungsverfahrens ausgesprochen gut. Die meisten unnötigen Spitzen und Scharten werden dadurch aus dem Netz entfernt.

Pseudocode des Algorithmus von *Dyn*, *Hormann*, *Kim* und *Levin*: In den Zeilen 1 bis 5 wird eine Liste aller möglichen Edge-Swaps aufgebaut und aufsteigend nach den jeweiligen ΔC-Werten sortiert. Die WHILE-Schleife in den Zeilen 6 bis 11 führt solange den jeweils besten Edge-Swap aus, bis keine Edge-Swaps mehr existieren, die die Gesamtkrümmung C des Dreiecksnetzes $\mathcal{T}$ reduzieren.

8.2 Verbesserung des Laufzeitverhaltens

Die Tatsache, dass die Liste der Edge-Swaps beim Algorithmus von *Dyn*, *Hormann*, *Kim* und *Levin* nach jedem Edge-Swap erneut sortiert werden muss, wirkt sich äusserst negativ auf das Laufzeitverhalten aus. Um das Verfahren deutlich zu beschleunigen wurde es im Rahmen dieser Studienarbeit derart verändert, dass nicht jeweils der beste Edge-Swap ausgeführt wird

1	*liste* = *Leere Liste von Edge-Swaps*.
2	FOR (Alle möglichen Edge-Swaps s):
3	Füge s in *liste* ein.
4	Berechne $\Delta C(s) = C(\mathcal{T}_{nachher}) - C(\mathcal{T}_{vorher})$ für alle Edge-Swaps s in *liste*.
5	Sortiere *liste* aufsteigend nach den ΔC-Werten der Edge-Swaps.
6	WHILE ($\Delta C($*liste*$[1]) < 0$):
7	Führe den Edge-Swap *liste*$[1]$ aus.
8	Entferne nicht mehr mögliche Edge-Swaps aus *liste*.
9	Füge neue mögliche Edge-Swaps in *liste* ein.
10	Aktualisiere die ΔC-Werte verschiedener Edge-Swaps. Dabei reicht es aus, nur die Kanten der beiden neuen und der daran angrenzenden Dreiecke zu betrachten.
11	Stelle die Sortierung von *liste* wieder her.

Pseudocode des Algorithmus von *Dyn*, *Hormann*, *Kim* und *Levin*

sondern ein beliebiger, der die Gesamtkrümmung C des Dreiecksnetzes $\mathcal{T}$ reduziert.

Bei diesem modifizierten Algorithmus werden die Kanten des Dreiecksnetzes in beliebiger Reihenfolge einmal durchlaufen. Dabei wird für jede Kante die rekursive Funktion *SWAP* aufgerufen.

Die Funktion *SWAP* überprüft, ob ein Edge-Swap dieser Kante möglich ist, und ob dieser die Gesamtkrümmung $C(\mathcal{T})$ reduziert. Ist das der Fall, wird dieser Edge-Swap ausgeführt. Um sicherzustellen, dass ein lokales Minimum der Gesamtkrümmung $C(\mathcal{T})$ erreicht wird, muss die Funktion *SWAP* nun rekursiv für diejenigen Kanten aufgerufen werden, bei denen sich die Krümmungsdifferenz der ihnen zugeordneten Edge-Swaps verändert haben kann. Diese Kanten sind, wie in Unterabschnitt 8.1.3 beschrieben, die Kanten der beiden neuen und der daran angrenzenden Dreiecke (Abbildung 8.4).

Die durch diesen modifizierten Algorithmus erzielten Ergebnisse sind noch immer relativ gut. Die Zeitersparnis im Vergleich zum ursprünglichen Verfahren ist beträchtlich.

Was sich noch immer äusserst ungünstig auf das Laufzeitverhalten auswirkt, sind die vor jedem Edge-Swap notwendigen Tests auf schlechte Dreiecke. *Test 1*, *Test 2* und *Test 4* sind dabei unproblematisch, da diese in konstanter Zeit durchgeführt werden können. *Test 5* hat jedoch ein Laufzeitverhalten von

$$T_{\textit{Test 5, ein Dreieck}}(n) \quad = \quad O(n) \tag{8.2}$$

(Abschnitt 4.5).

Eine nicht unbedingt elegante aber zumindest relativ effiziente Lösung ist die Folgende. Statt *Test 5* beim Glätten des Dreiecksnetzes immer wieder durchzuführen, werden alle schlechten *Fall-5-Dreiecke* (das sind erfahrungsgemäss wenig bis gar keine) nach Beendigung des Glättungsalgorithmus aus dem Netz entfernt. Dafür wird sehr viel weniger Zeit benötigt, als die vielen einzelnen Tests im Laufe des Algorithmus verbrauchen würden. Wurden Dreiecke entfernt, müssen wegen der dadurch entstandenen Löcher *Schritt 3* (Triangulieren einfacher Löcher), *Schritt 4* (Triangulieren komplexer Löcher) und *Schritt 5* (Einfügen isolierter Punkte) erneut ausgeführt werden. Aufgrund der in aller Regel geringen Anzahl von entstandenen Löchern wird das Dreiecksnetz durch diese Schritte kaum mehr verändert. Die vorher erzeugte „Glätte“ des Netzes bleibt also weitestgehend erhalten.

Pseudocode der rekursiven Funktion *SWAP*: Die Funktion *SWAP* wird mit einem Parameter, einer Kante, aufgerufen. Zuerst wird sichergestellt, dass der dieser Kante zugeordnete Edge-Swap die Gesamtkrümmung C des Dreiecksnetzes reduziert (Zeilen 3 bis 5), und dass durch ihn keine topologischen Fehler (Zeilen 6 und 7) und keine zu spitzen oder sich schneidenden Dreiecke (Zeilen 8 und 9; *Fall 5* wird dabei vernachlässigt) erzeugt werden. In Zeile 10 wird der Edge-Swap dann ausgeführt. Zuletzt wird für alle Kanten, an denen sich die Krümmungsdifferenzen der ihnen zugeordneten Edge-Swaps verändert haben können, die Funktion *SWAP* rekursiv aufgerufen (Zeilen 11 bis 13).

Pseudocode des modifizierten Glättungsalgorithmus: In den Zeilen 1 und 2 wird die rekursive Funktion *SWAP* für jede Kante einmal aufgerufen. Befinden sich danach *Fall-5-Dreiecke* im Netz, werden diese entfernt, und *Schritt 3* (Triangulieren einfacher Löcher), *Schritt 4* (Triangulieren komplexer Löcher) und *Schritt 5* (Einfügen isolierter Punkte) werden erneut ausgeführt (Zeilen 3 bis 7).

8.3 Laufzeitverhalten

Eine präzise Aussage über das Laufzeitverhalten des modifizierten Glättungsalgorithmus ist wegen der unvorhersehbaren Rekursionstiefe und Rekursionsbreite der Funktion *SWAP* nicht möglich.

Experimente haben gezeigt, dass bei den ersten Aufrufen von *SWAP* in der Hauptschleife (Zeile 1 im Pseudocode des modifizierten Glättungsalgorithmus) die Rekursionstiefen und die Rekursionsbreiten sehr hoch sind, bei späteren Aufrufen die Rekursionen jedoch meistens relativ bald abbrechen. Das ist dadurch zu erklären, dass die Edge-Swap-Operationen bei einem ungeglätteten Dreiecksnetz (viele Edge-Swaps sind möglich) wegen der kaskadenartigen Rekursion über grosse Bereiche des Netzes „hinweglaufen“ können. Bei späteren Aufrufen von *SWAP* ist das Dreiecksnetz bereits weitestgehend geglättet (nur noch wenige Edge-Swaps sind möglich). Die Edge-Swap-Operationen können sich dann nicht mehr „ausbreiten“, da sie von

```
1   FUNCTION SWAP (Kante e):
2       s = Der e zugeordnete Edge-Swap.
3       ΔC(s) = C(T_nachher) − C(T_vorher).
4       IF (ΔC(s) ≥ 0):
5           RETURN.
6       IF (e ist innere Kante eines aus drei Dreiecken bestehenden Fächers):
7           RETURN.
8       IF (s erzeugt Fall-1-Dreiecke, Fall-2-Dreiecke oder Fall-4-Dreiecke):
9           RETURN.
10      Führe s aus.
11      E = Menge der Kanten der beiden neuen und der daran angrenzenden Dreiecke
                abzüglich der gerade „geswappten“ Kante.
12      FOR (Alle ẽ ∈ E):
13          SWAP(ẽ).
```

Pseudocode der rekursiven Funktion $SWAP$

```
1   FOR (Alle Kanten e):
2       SWAP(e).
3   IF (Fall-5-Dreiecke sind im Dreiecksnetz vorhanden):
4       Entferne diese Fall-5-Dreiecke.
5       Führe Schritt 3 (Triangulieren einfacher Löcher) aus.
6       Führe Schritt 4 (Triangulieren komplexer Löcher) aus.
7       Führe Schritt 5 (Einfügen isolierte Punkte) aus.
```

Pseudocode des modifizierten Glättungsalgorithmus

Kanten umgeben sind, die nicht mehr „geswappt“ werden dürfen.

Während der vielen Testläufe im Rahmen dieser Arbeit hat sich gezeigt, dass die Edge-Swap-Operationen deutlich weniger Zeit verbrauchen, als der darauf folgende *Test 5*. Wegen der meistens relativ geringen Anzahl von *Test-5-Dreiecken*, die dabei entfernt werden, beeinflussen die danach stattfindenden Schritte, Triangulieren einfacher Löcher (*Schritt 3*), Triangulieren komplexer Löcher (*Schritt 4*) und Einfügen isolierter Punkte (*Schritt 5*), den Zeitverbrauch ebenfalls nur unwesentlich. Es kann also davon ausgegangen werden, dass für die meisten

Dreiecksnetze das Laufzeitverhalten dieses Schrittes dem Laufzeitverhalten von *Test 5* für das gesamte Dreiecksnetz entspricht. Dies führt zu einem experimentell ermittelten mittleren Laufzeitverhalten von

$$O(n \cdot \log n) \quad \leq \quad \overline{T}_{\textit{Glätten des erzeugten Dreiecksnetzes}}(n) \quad \leq \quad O(n^2) \quad . \tag{8.3}$$

Kapitel 9

Ergebnisse

Der in dieser Studienarbeit entwickelte Rekonstruktionsalgorithmus wurde anhand zahlreicher Punktewolken verschiedener Grösse getestet. Die Formen der zu diesen Test-Punktewolken gehörigen Objekte decken ein breites Spektrum ab. In diesem Kapitel wird einerseits eine Reihe von Messergebnissen präsentiert, entstanden im Rahmen einer Vielzahl von Testläufen, andererseits wird die Qualität der zu verschiedenen Test-Punktewolken erzeugten Dreiecksnetze genauer betrachtet.

Abbildung 9.1 zeigt aus verschiedenen Test-Punktewolken rekonstruierte Dreiecksnetze. Dem Leser soll dadurch eine Vorstellung von den Formen der zu diesen Punktewolken gehörigen Objekte gegeben werden. Die Grössen der entsprechenden Test-Punktewolken sind in Abbildung 9.1 in runden Klammern angegeben.

Da der Rekonstruktionsalgorithmus nicht-deterministisch arbeitet, wurden zu jeder der Test-Punktewolken zehn Dreiecksnetze generiert. Dadurch konnten generelle Tendenzen und Probleme leichter erkannt, und allgemein gültigere Aussagen über die Qualität der erzeugten Netze getroffen werden. Beim Erzeugen der Dreiecksnetze wurden die in den vorausgehenden Kapiteln vorgeschlagenen Parameter verwendet.

9.1 Messergebnisse

Bei jedem der jeweils zehn Rekonstruktionsläufe wurden die folgenden Grössen ermittelt.

- Die von den sechs Schritten des Rekonstruktionsalgorithmus benötigten Zeiten, $t_{Schritt\ 1}$, $t_{Schritt\ 2}$, $t_{Schritt\ 3}$, $t_{Schritt\ 4}$, $t_{Schritt\ 5}$ und $t_{Schritt\ 6}$ (*AMD Duron 600 Mhz*, 128 *MB* Hauptspeicher).
- Die Anzahl der offenen Kanten nach *Schritt 2* (deterministische Grösse, dass heisst bei gleicher Punktewolke bei jeden Rekonstruktionslauf gleich).
- Die Anzahl der offenen Kanten nach *Schritt 3* (deterministische Grösse, dass heisst bei

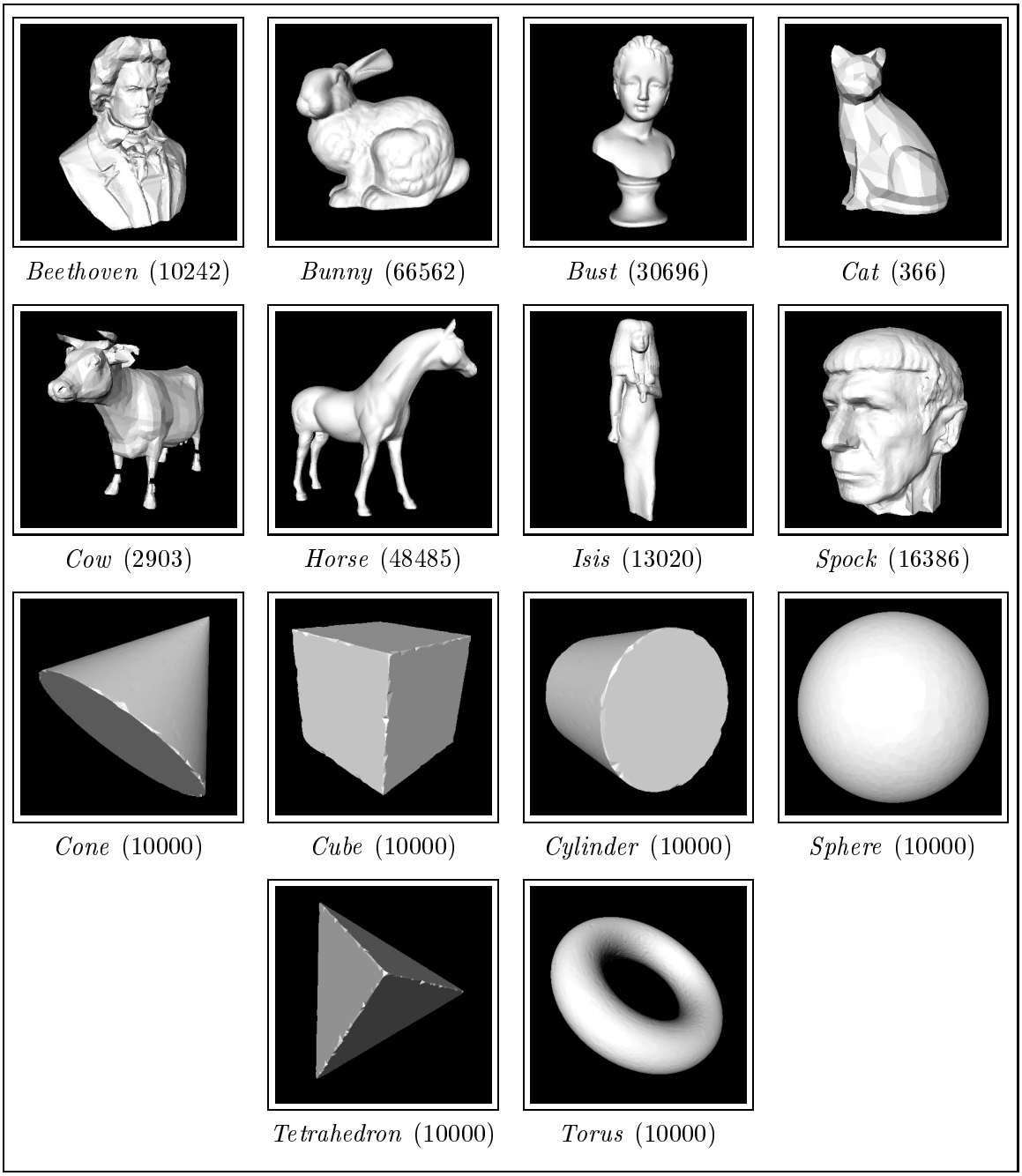


Abbildung 9.1: Aus verschiedenen Test-Punktewolken rekonstruierte Dreiecksnetze

gleicher Punktewolke bei jeden Rekonstruktionslauf gleich).

- Die Anzahl der isolierten Punkte nach *Schritt 4*.

Zu allen Messreihen $\{x_1, x_2, \ldots, x_{10}\}$ von nicht-konstanten Grössen wurde der Mittelwert

$$\overline{x} = \frac{1}{10}\sum_{i=1}^{10} x_i \tag{9.1}$$

und die Standardabweichung

$$\Delta x = \sqrt{\frac{1}{9}\sum_{i=1}^{10}(x_i - \overline{x})^2} \tag{9.2}$$

berechnet[1]. Diese Messergebnisse können Tabelle 9.1 und Tabelle 9.2 entnommen werden. Sie sind bei konstanten Grössen in der Form *Wert*, bei nicht-konstanten Grössen in der Form *Mittelwert* $\pm$ *Standardabweichung* angegeben.

9.2 Bewertung der Ergebnisse

Es fällt auf, dass bei kleinen Punktewolken in der Regel verhältnismässig mehr Zeit zum Schliessen der Löcher benötigt wird (*Schritt 3* und *Schritt 4*), als bei grossen Punktewolken. Dies liegt daran, dass die in *Schritt 1* erzeugte initiale Triangulierung an dünn abgetasteten kritischen Stellen (starke Krümmung, kleiner Durchmesser; Abschnitt 1.4) von deutlich schlechterer Qualität ist (verhältnismässig mehr schlechte Dreiecke und Löcher), als an dicht abgetasteten kritischen Stellen. Ein gutes Beispiel hierfür stellt das Modell *Bunny* dar. Bei 487 Punkten wurde 80% der Zeit zum Schliessen der Löcher benötigt, bei 7958 Punkten 31% und bei 66562 Punkten nur noch 4%.

Dass die initiale Triangulierung an scharfen Kanten viele Fehler aufweist, in Bereichen schwacher Krümmung jedoch fast fehlerfrei ist, zeigt auch der direkte Vergleich der beiden Modelle *Sphere* (keine Kanten) und *Tetrahedron* (sehr scharfe Kanten). Bei jeweils 10000 Punkten betrug die Zeit zum Schliessen der vorhandenen Löcher bei der Kugel nur 46 Sekunden, beim Tetraeder jedoch 419 Sekunden.

Da das in *Schritt 4* verwendete *Simulated-Annealing* ein zufallsgesteuertes Verfahren ist, liefern mehrere Rekonstruktionsläufe in der Regel verschiedene Dreiecksnetze (Abbildung 9.2). Erfahrungsgemäss unterscheiden sich diese Ergebnisse um so stärker, je dünner in kritischen Bereichen abgetastet wurde. Entspricht die Form des erzeugten Dreiecksnetzes nicht gut genug der Form des zur Punktewolke gehörigen Objekts, sollte der Algorithmus erneut gestartet werden.

Es hat sich gezeigt, dass der entwickelte Rekonstruktionsalgorithmus zu fast jeder Punktewolke, die keine aussergewöhnlich niedrige Punktdichte aufweist, in der Regel bereits im ersten Durchlauf ein Dreiecksnetz erzeugt, das die Form des zu dieser Punktewolke gehörigen Objekts

[1] Unter der Annahme, dass die x_i normalverteilt sind, liegen etwa 68% der x_i im Intervall $\overline{x} \pm \Delta x$. Nähere Informationen dazu sind in [Stoe98] zu finden.

Modell (# Punkte)	$t_{Schritt\ 1}$ in sec	$t_{Schritt\ 2}$ in sec	# offene Kanten nach *Schritt 2*	$t_{Schritt\ 3}$ in sec	# offene Kanten nach *Schritt 3*
Beethoven (2515)	2 ± 0	2 ± 0	1009	5 ± 0	196
Beethoven (10242)	12 ± 0	19 ± 1	917	17 ± 0	99
Bunny (487)	0 ± 0	0 ± 0	331	0 ± 1	81
Bunny (7958)	10 ± 0	14 ± 0	782	11 ± 0	26
Bunny (66562)	233 ± 16	376 ± 19	136	26 ± 8	9
Bust (8348)	8 ± 1	10 ± 0	1535	27 ± 1	10
Bust (30696)	54 ± 0	72 ± 0	760	28 ± 1	0
Cat (366)	0 ± 0	0 ± 0	88	0 ± 0	10
Cow (2903)	2 ± 0	2 ± 0	892	4 ± 1	255
Horse (9264)	9 ± 0	13 ± 0	1405	28 ± 0	76
Horse (48485)	103 ± 1	165 ± 1	728	82 ± 1	47
Isis (13020)	16 ± 0	21 ± 0	1862	39 ± 0	49
Spock (1779)	1 ± 1	2 ± 0	724	3 ± 0	139
Spock (16386)	26 ± 0	55 ± 1	408	8 ± 0	24
Cone (1000)	1 ± 1	1 ± 0	311	1 ± 0	33
Cone (10000)	14 ± 0	23 ± 0	2030	47 ± 0	165
Cube (1000)	0 ± 1	1 ± 0	318	1 ± 0	7
Cube (10000)	17 ± 0	93 ± 1	1908	45 ± 0	22
Cylinder (1000)	0 ± 1	1 ± 0	220	0 ± 0	5
Cylinder (10000)	17 ± 0	24 ± 0	1778	38 ± 1	31
Sphere (1000)	1 ± 0	1 ± 0	131	0 ± 0	0
Sphere (10000)	16 ± 0	19 ± 0	1314	28 ± 1	13
Tetrahedron (1000)	1 ± 1	1 ± 0	427	1 ± 0	78
Tetrahedron (10000)	13 ± 0	22 ± 0	2478	55 ± 1	231
Torus (1000)	0 ± 1	1 ± 0	159	0 ± 1	0
Torus (10000)	14 ± 0	17 ± 0	1499	31 ± 0	26

Tabelle 9.1: Messergebnisse (1. Teil)

Modell (# Punkte)	$t_{Schritt\ 4}$ in sec	# isolierte Punkte nach *Schritt 4*	$t_{Schritt\ 5}$ in sec	$t_{Schritt\ 6}$ in sec	t_{Gesamt} in sec
Beethoven (2515)	47 ± 13	13 ± 5	3 ± 1	5 ± 1	64 ± 13
Beethoven (10242)	136 ± 22	9 ± 2	11 ± 2	28 ± 1	222 ± 23
Bunny (487)	8 ± 2	4 ± 2	0 ± 0	1 ± 0	10 ± 2
Bunny (7958)	9 ± 4	0 ± 0	0 ± 0	20 ± 1	64 ± 4
Bunny (66562)	19 ± 16	0 ± 0	0 ± 0	459 ± 26	1113 ± 47
Bust (8348)	30 ± 9	0 ± 0	0 ± 0	16 ± 0	92 ± 9
Bust (30696)	0 ± 0	0 ± 0	0 ± 0	94 ± 1	249 ± 1
Cat (366)	1 ± 1	0 ± 0	0 ± 0	0 ± 1	1 ± 1
Cow (2903)	94 ± 18	31 ± 6	10 ± 2	5 ± 1	117 ± 19
Horse (9264)	88 ± 22	7 ± 3	8 ± 3	20 ± 1	166 ± 23
Horse (48485)	316 ± 219	8 ± 1	51 ± 7	204 ± 2	922 ± 217
Isis (13020)	81 ± 49	22 ± 36	30 ± 48	34 ± 3	220 ± 95
Spock (1779)	29 ± 8	10 ± 2	2 ± 0	3 ± 0	40 ± 8
Spock (16386)	3 ± 5	7 ± 3	12 ± 6	74 ± 1	178 ± 9
Cone (1000)	3 ± 1	3 ± 2	0 ± 1	2 ± 0	8 ± 2
Cone (10000)	257 ± 55	12 ± 3	13 ± 3	36 ± 1	390 ± 54
Cube (1000)	2 ± 1	0 ± 0	0 ± 0	2 ± 0	6 ± 1
Cube (10000)	34 ± 12	1 ± 0	1 ± 1	102 ± 1	292 ± 11
Cylinder (1000)	0 ± 0	0 ± 0	0 ± 0	3 ± 0	5 ± 1
Cylinder (10000)	13 ± 19	0 ± 0	0 ± 0	32 ± 0	124 ± 18
Sphere (1000)	0 ± 0	0 ± 0	0 ± 0	1 ± 0	3 ± 0
Sphere (10000)	18 ± 5	1 ± 0	1 ± 0	20 ± 0	102 ± 4
Tetrahedron (1000)	38 ± 15	6 ± 2	1 ± 0	2 ± 1	43 ± 15
Tetrahedron (10000)	364 ± 43	14 ± 3	16 ± 4	28 ± 0	498 ± 43
Torus (1000)	0 ± 0	0 ± 0	0 ± 0	2 ± 0	3 ± 1
Torus (10000)	16 ± 26	0 ± 0	0 ± 0	30 ± 0	109 ± 26

Tabelle 9.2: Messergebnisse (2. Teil)

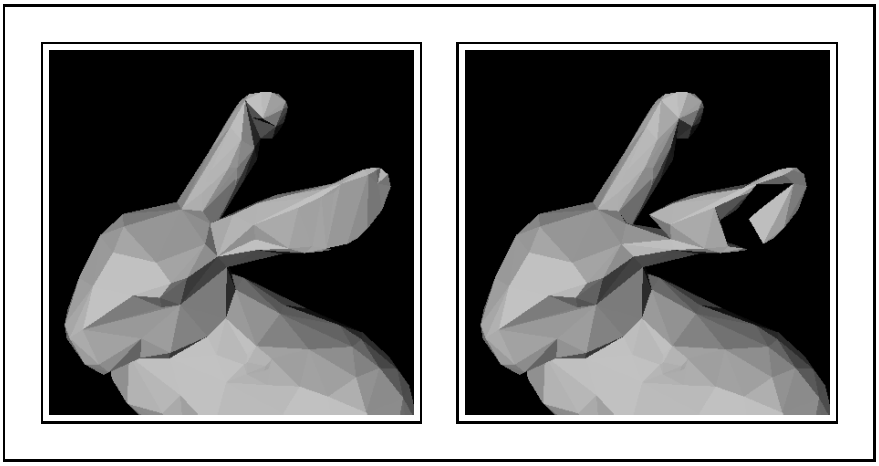


Abbildung 9.2: Zwei verschiedene Rekonstruktionen des Modells *Bunny* (487 Punkte)

gut approximiert. Darüber hinaus werden bei geschickter Wahl der vorzugebenden Parameter auch durch sehr dünne Abtastung entstandene Punktewolken meist nach wenigen Durchläufen zufriedenstellend trianguliert. Dies ist ein grosser Vorteil gegenüber vielen bisherigen Rekonstruktionsalgorithmen, da diese deterministisch arbeiten, und daher Punktewolken immer auf dieselbe, bei dünner Abtastung meist falsche Weise triangulieren.

Kapitel 10

Weitere Forschungsmöglichkeiten

Der in dieser Studienarbeit entwickelte Algorithmus liefert ausgesprochen gute Ergebnisse. Es gibt jedoch noch immer eine Reihe von Verbesserungsmöglichkeiten und bestehenden Problemen, die aufgrund der auf drei Monate begrenzten Bearbeitungszeit nicht näher betrachtet werden konnten. Im Folgenden wird zu jedem der sechs Schritte des Algorithmus ein kurzer Ausblick auf weitere Forschungsmöglichkeiten gegeben.

10.1 Erzeugen einer initialen Triangulierung (*Schritt 1*)

Je besser die Ergebnisse der initialen Triangulierung sind, desto weniger Zeit wird zur Nachbearbeitung benötigt (wenig offene Kanten bedeuten einen geringen Zeitverbrauch von *Schritt 3*, *Schritt 4* und *Schritt 5*), und desto besser ist die Qualität der vom vollständigen Rekonstruktionsalgorithmus erzeugten Dreiecksnetze. Es existieren zahlreiche Arbeiten, die sich mit dem Problem der initialen Triangulierung beschäftigen. Das hier verwendete Verfahren zum Erzeugen einer initialen Triangulierung kann durch ein beliebiges anderes ersetzt werden, da an das resultierende Dreiecksnetz keinerlei Forderungen gestellt werden (topologische Fehler, sich schneidende Dreiecke und Löcher sind zulässig). Ob einer dieser bereits bestehenden oder eventuell ein ganz neuer Algorithmus zum Erzeugen einer initialen Triangulierung zu besseren Endergebnissen führt, bleibt zu untersuchen.

10.2 Entfernen von schlechten Dreiecken (*Schritt 2*)

Ein grosses Problem beim Entfernen von schlechten Dreiecken ist das ungünstige Laufzeitverhalten von *Test 5*. Alle anderen Tests sind für ein bestimmtes Dreieck lokal und damit in konstanter Zeit durchführbar, während *Fall 5* ein lineares Laufzeitverhalten aufweist (Abschnitt 4.5). Ein Verfahren zu entwickeln, das den Zeitverbrauch von *Test 5* deutlich verringert, stellt sicher ein lohnendes Ziel dar.

Ein weiterer Punkt sind numerische Instabilitäten, die vor allem bei *Fall 4* und *Fall 5* hin

und wieder auftreten und zu falschen Ergebnissen führen können. Obwohl bei der Implementierung des entwickelten Rekonstruktionsalgorithmus bereits grosse Anstrengungen unternommen wurden, numerische Fehler zu vermeiden, können sie dennoch nicht ganz ausgeschlossen werden. Eine vom numerischen Standpunkt aus tiefergehende Betrachtung und ein eventuelles Überarbeiten der entsprechenden Teilalgorithmen wäre sicher hilfreich.

10.3 Triangulieren einfacher Löcher (*Schritt 3*)

Der Algorithmus zum Triangulieren einfacher Löcher ist ein deterministischer Algorithmus, der in den meisten Fällen sehr gute Ergebnisse liefert. Das im darauf folgenden Schritt stattfindende *Simulated-Annealing* ist dagegen ein zufallsgesteuertes Verfahren. Gerade wegen dieser Zufälligkeit entstehen unter Umständen auch bei „einfachen komplexen Löchern“ unschöne Ergebnisse. Den Algorithmus zum Triangulieren einfacher Löcher so zu erweitern, dass auch „einfache komplexe Löcher“ deterministisch geschlossen werden, ohne dabei jedoch das Laufzeitverhalten dramatisch zu verschlechtern, ist sicher ein lohnendes Forschungsziel.

10.4 Triangulieren komplexer Löcher (*Schritt 4*)

Um unnötig zackige Triangulierungen oder das Abreissen von einzelnen Punkten oder ganzen Netzteilen mit noch grösserer Wahrscheinlichkeit zu vermeiden, könnte das *Simulated-Annealing* durch weitere Bewertungsfunktionen ergänzt werden.

Darüber hinaus sind die Ergebnisse des *Simulated-Annealing* stark abhängig von der Wahl der Parameter. Wie die optimalen Parameter zu finden sind, ist ein sehr schwieriges Thema, zu dem es kaum Literatur gibt. Tiefergehende theoretische Betrachtungen und Experimente mit vielen verschiedenen Punktewolken könnten für den hier vorliegenden Fall der Oberflächenrekonstruktion Aufschluss bringen.

10.5 Einfügen isolierter Punkte (*Schritt 5*)

Die Ergebnisse dieses Schrittes sind erfahrungsgemäss relativ zufriedenstellend, sein Zeitverbrauch ist eher gering. Eine Verbesserung dieses Schrittes scheint also weder notwendig noch lohnend zu sein.

10.6 Glätten des erzeugten Dreiecksnetzes (*Schritt 6*)

Ein Problem ist, dass durch diesen Schritt nur ein lokales Minimum der Gesamtkrümmung des rekonstruierten Dreiecksnetzes erreicht wird. Dieses lokale Minimum entspricht im Allgemeinen nicht dem globalen und eigentlich gewünschten Minimum. Durch einen erneuten

Einsatz von *Simulated-Annealing* könnte die Wahrscheinlichkeit, ein globales Minimum oder zumindest ein sehr viel besseres lokales Minimum zu erreichen, deutlich gesteigert werden.

Kapitel 11

Zusammenfassung

In dieser Studienarbeit wurde ein neuer Algorithmus zur Rekonstruktion von Oberflächen aus Punktewolken entwickelt.

Dieser neue Algorithmus trianguliert nahezu beliebige dreidimensionale Punktewolken. Dabei wird garantiert, dass die erzeugten Dreiecksnetze sowohl fehlerfrei als auch geschlossen sind. Ausserdem entsprechen die Formen der generierten Dreiecksnetze in den meisten Fällen den Formen der zu den entsprechenden Punktewolken gehörigen Objekte.

Der Algorithmus gliedert sich in sechs verschiedene, voneinander unabhängige Teilalgorithmen. Jeder dieser Teilalgorithmen ist in sich abgeschlossen und kann daher auch isoliert verwendet werden, entweder zur Oberflächenrekonstruktion oder bei verwandten Problemen.

In Kapitel 4 (Entfernen schlechter Dreiecke (*Schritt 2*)), Kapitel 5 (Triangulieren einfacher Löcher (*Schritt 3*)), Kapitel 6 (Triangulieren komplexer Löcher (*Schritt 4*)) und Kapitel 7 (Einfügen isolierter Punkte (*Schritt 5*)) werden neu entwickelte Konzepte und Verfahren zur Nachbearbeitung und Reparatur von fehlerhaften und unvollständigen Dreiecksnetzen präsentiert. Auf dieses schwierige, für die Oberflächenrekonstruktion ungemein wichtige Thema wurde in bisherigen Arbeiten so gut wie gar nicht eingegangen. Es bleibt zu hoffen, dass diese Arbeit Anstoss zu weiteren Anstrengungen auf diesem Gebiet gibt.

Literaturverzeichnis

[AlSc96] M.-E. Algorri, F. Schmitt. *Surface Reconstruction from Unstructured 3D Data.* Computer Graphics Forum, Volume 15, Number 1. 1996.

[AmBe98] N. Amenta, M. Bern, M. Kamvysselis. *A New Voronoi-Based Surface Reconstruction Algorithm.* SIGGRAPH '98 Proceedings. 1998.

[BaBe97] C. L. Bajaj, F. Bernardini, G. Xu. *Reconstructing Surfaces and Functions on Surfaces from Unorganized Three-Dimensional Data.* Algorithmica, Volume 19. 1997.

[BeMi99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin. *The Ball-Pivoting Algorithm for Surface Reconstruction.* IEEE Transactions on Visualization and Computer Graphics, Volume 5, Number 4. 1999.

[Bent75] J. L. Bentley. *Multidimensional Binary Search Trees Used for Associative Searching.* Communications of the ACM, Volume 18, Number 9. 1975.

[BiTs95] E. Bittar, N. Tsingos, M.-P. Gascuel. *Automatic Reconstruction of Unstructured 3D Data: Combining a Medial Axis and Implicit Surfaces.* Computer Graphics Forum, Volume 14, Number 3. 1995.

[BrSe97] I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig. *Taschenbuch der Mathematik.* Verlag Harri Deutsch. 1997.

[DeMe99] M. Desbrun, M. Meyer, P. Schröder, A. H. Barr. *Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow.* SIGGRAPH '99 Proceedings. 1999.

[DyHo00] N. Dyn, K. Hormann, S.-J. Kim, D. Levin. *Optimizing 3D Triangulations Using Discrete Curvature Analysis.* 2000.

[Fova96] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. *Computer graphics: principles and practice.* Addison-Wesley. 1996.

[FrBa75] J. H. Friedman, F. Baskett, L. J. Shustek. *An Algorithm for Finding Nearest Neighbors.* IEEE Transactions on Computers. 1975.

[FrBe77] J. H. Friedman, J. L. Bentley, R. A. Finkel. *An Algorithm for Finding Best Matches in Logarithmic Expected Time.* ACM Transactions on Mathematical Software, Volume 3, Number 3. 1977.

[GoKr00] M. Gopi, S. Krishnan, C. T. Silva. *Surface Reconstruction based on Lower Dimensional Localized Delaunay Triangulation.* Computer Graphics Forum, Volume 19, Number 3. 2000.

[GoMe97] B. Gou, J. Menon, B. Willette. *Surface Reconstruction Using Alpha Shapes.* Computer Graphics Forum, Volume 16, Number 4. 1997.

[Gou97] B. Gou. *Surface reconstruction: from points to splines.* Computer-Aided Design, Volume 29, Number 4. 1997.

[Grab95] H. Grabmüller. *Mathematik für Ingenieure I (für Elektrotechniker).* 1995.

[Grab98a] H. Grabmüller. *Mathematik für Ingenieure II (für Informatiker).* 1998.

[Grab98b] H. Grabmüller. *Mathematik für Ingenieure III (für Informatiker).* 1998.

[Grab00] H. Grabmüller. *Numerik I (für Ingenieure).* 2000.

[Grae98] F. Graef. *Wahrscheinlichkeitsrechnung 2 – Dynamische stochastische Systeme mit diskretem Zustandsraum.* 1998.

[HoDe92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Surface Reconstruction from Unorganized Points.* Computer Graphics, Volume 26, Number 2. 1992.

[HoDe93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Mesh Optimization.* SIGGRAPH '93 Proceedings. 1993.

[HoDe94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, W. Stuetzle. *Piecewise Smooth Surface Reconstruction.* SIGGRAPH '94 Proceedings. 1994.

[LiLa98] S. B. Lippman, J. Lajoie. *C++ Primer.* Addison-Wesley. 1998.

[MeMu97a] R. Mencl, H. Müller. *Graph-Based Surface Reconstruction Using Structures in Scattered Point Sets.* 1997.

[MeMu97b] R. Mencl, H. Müller. *Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points.* 1997.

[Menc95] R. Mencl. *A Graph-Theoretic Approach to Surface Reconstruction.* 1995.

[Moel97] T. Möller. *A Fast Triangle-Triangle Intersection Test.* Journal of Graphics Tools, Volume 2, Number 2. 1997.

[PrTe97] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press. 1997.

[RuMc86] D. E. Rumelhart, J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations.* MIT Press. 1986.

[Schu93] L. L. Schumaker. *Computing optimal triangulations using simulated annealing.* Computer Aided Geometric Design, Volume 10. 1993.

[Stoe98] H. Stöcker. *Taschenbuch der Physik.* Verlag Harri Deutsch. 1998.

[Wern94] J. Wernecke. *The inventor mentor: programming Object-oriented 3D graphics with Open Inventor, release 2.* Addison-Wesley. 1994.

Erklärung

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ich bin damit einverstanden, daß die Arbeit veröffentlicht wird und daß in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Graphische Datenverarbeitung, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Erlangen, den 31. Juli 2001

(Marc Axel Johannes Wagner)