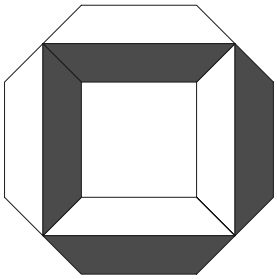


Universität Karlsruhe (TH)

Institut für Betriebs- und Dialogsysteme

Von Punktwolken zu Dreiecksnetzen

Diplomarbeit

von

Stefan Preuß

Januar 2002

Betreuer:
Prof. Dr. Hartmut Prautzsch
Dr. Lars Linsen

Hiermit erkläre ich, die vorliegende Arbeit selbstständig erstellt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Karlsruhe, den 31. Januar 2002

_____

(Stefan Preuß)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

In vielen Bereichen, wie z.B. in der Medizin, der Geografie aber auch bei der generellen Analyse von Prototypen ist es notwendig, reale Objekte oder Oberflächen in Daten umzusetzen, um sie weiterverarbeiten zu können.

Hierzu werden stichprobenartig einzelne Punkte der Oberfläche abgetastet, die dann in ein Koordinatensystem übernommen werden. Die Punktmenge, die das darzustellende Objekt repräsentiert, wird als Punktwolke bezeichnet. Dabei gehen weitere Informationen zur Beziehung zwischen diesen Punkten (z.B. Objektkrümmung, Oberflächenrichtung und Geschlossenheit der Oberfläche) verloren, so das man bei einer Rekonstruktion des Objektes nur auf die puren Punktpositionen zugreifen kann.

Viele Ansätze haben sich schon mit dieser Problematik beschäftigt und verschiedene Lösungsvorschläge erarbeitet.

Ziel dieser Diplomarbeit ist es nun, eine Rekonstruktion eines Objektes durch dessen Darstellung in einem Dreiecksnetz zu erzeugen. Hierzu werden um jeden Punkt der Punktwolke Dreiecksfächer durch Verbindung mit dessen umliegenden Nachbarpunkten gebildet, wie sie bei den Visualisierungsmethoden von Lars Linsen [Linsen '01] hergeleitet wurden. In der Weiterentwicklung seines Ansatzes sollen nun aber schon bei der Erstellung der Fächerdreiecke mögliche Fehler abgefangen werden. Hierzu wurden Fächerkonstellationen analysiert und daraus ein Algorithmus entwickelt, der fehlerhafte oder unvorteilhafte Punktverbindungen verhindert, indem er deren Ursachen ausschließt. Des weiteren sollte analysiert werden, ob dieser Ansatz im Vergleich zu anderen Ansätzen zu einer Zeitersparnis führt.

1.1 Überblick über die Arbeit

Zunächst sollen ein paar Definitionen einiger in dieser Arbeit oft verwendeten Begriffe gegeben werden.

Im zweiten Kapitel werden dann neben der Dissertation von Lars Linsen, die die Basis dieser Arbeit lieferte, weitere Arbeiten vorgestellt, die im Ansatz dieser Arbeit ähneln oder aber andere Ansätze verfolgen, und versucht so ein breites Spektrum über die Problematik der Objektrekonstruktion zu zeichnen. Dies geschieht ausführlich, um Nachfolgenden einen Überblick über bisher veröffentlichte Ansätze zu diesem Thema

zu bieten.

Das dritte Kapitel widmet sich dem entwickelten Algorithmus zur Rekonstruktion oder Erzeugung eines Dreiecksnetzes aus einer ungeordneten Punktemenge. Dazu wird im Vorfeld die Aufgabestellung und die auftretenden Probleme analysiert. Anschließend wird das Umkreiskriterium, welches eng mit der Delaunay-Triangulierung verknüpft ist, näher untersucht und Sätze hergeleitet mit denen man auf dieses Kriterium einfacher und schneller testen kann. Danach werden die nötigen Vorverarbeitungen dargestellt. Dazu gehört die Vorsortierung der Punktemenge und die Berechnung der Oberflächennormalen zu jedem Punkt. Schließlich werden die einzelnen Schritte des Algorithmus ausführlich beschrieben und erörtert, inwieweit der Algorithmus die Aufgabestellung bewältigt. Daraus entstehen Anforderungen an die Punktwolke, die anschließend definiert werden.

Im vierten Kapitel werden die praktischen Erfahrungen zusammengefasst und ein Ausblick gegeben, wie die in dieser Arbeit gesammelten Erkenntnisse weiter genutzt und/oder entwickelt werden können.

Nach dem Literaturverzeichnis wird im Anhang ein Überblick über die Implementation des Algorithmus gegeben.

1.2 Begriffsdefinition

Um Missverständnisse zu vermeiden, möchte ich zu Anfang die in dieser Arbeit oft verwendete Begriffe definieren bzw. präzisieren.

Raum: Bei dem Begriff Raum bezieht sich diese Arbeit auf das realitätsnahe mathematische Modell des euklidischen Raums in zwei oder drei Dimensionen, mit der definierten Euklidnorm zur Bewertung von Längen.

Punktwolke: Unter einer Punktwolke wird in diesem Zusammenhang eine Menge ungeordneter Koordinaten (3-stellige Wertetupel) beschrieben, die Positionen oder Punkte im euklidischen $\mathbb{R}^3$ repräsentieren. Diese Punkte bilden eine Teilmenge der Oberfläche eines dreidimensionalen Objektes.

Triangulation der Punktwolke: Triangulation bezeichnet eine Menge von Relationen von untereinander verschiedenen drei Elementen der Punktwolke, sprich Punkten im euklidischen $\mathbb{R}^3$. Anschaulich bedeutet dies eine Menge aus Dreiecken, die durch Verbindung von jeweils drei Punkten der Punktwolke gebildet werden.

Triangulierung: Dieser Begriff steht für den Erstellungvorgang einer Triangulation.

Korrekte Triangulation: Von einer korrekten Triangulation wird in diesem Zusammenhang gesprochen, wenn für je zwei beliebige Dreiecke der Triangulation gilt, dass sie sich nicht enthalten oder durchdringen. Es soll weiterhin für alle Punktepaare die eine Kante eines Dreiecks bilden gelten, dass diese Kante Bestandteil von nur maximal zwei Dreiecken ist.

Vollständige Triangulation: Eine vollständige Triangulation bedeutet hier, dass für jeden Punkt der Punktwolke gilt, dass er Element mindestens eines Dreiecks der Triangulation ist.

Geschlossene Triangulation: Eine geschlossene Triangulation bedeutet, dass es zu jedem Dreieck eine klar definierte Außen- und Innenseite gibt. Bildlich gesprochen wird die Objektoberfläche komplett abgedeckt.

Zufriedenstellende Triangulation: Dies ist ein etwas subjektiver Begriff. Es wird hier von einer zufriedenstellenden Triangulation gesprochen, wenn sie korrekt, vollständig und an den Stellen geschlossen ist, an denen das zugrunde liegende Objekt eine Oberfläche aufweist. Die Oberfläche des zugrunde liegenden Objektes sollte durch die Dreiecksmenge hinreichend optimal angenähert werden.

Nachbar eines Punktes p: Dies bezeichnet einen weiteren Punkt $p' \neq p$ der Punktwolke, dessen Abstand zum Punkt p geringer ist als ein definierter maximaler Abstand *d*. Dieser Abstand *d* wird manchmal auch als Suchradius bezeichnet.

Fächer: Ein Fächer ist eine Menge von Dreiecken, die durch einen gemeinsamen Punkt, dem Fächermittelpunkt, und weiteren Punkten, den Randpunkten, begrenzt werden und eine korrekte Triangulation dieser Punkte bilden.

Vollständiger Fächer: ein Fächer wird als vollständig bezeichnet, wenn jeder Randpunkt in genau zwei Dreiecken des Fächers enthalten ist. Ein vollständiger Fächer entspricht also einer Triangulation des in [Schmitt '96] (S.18) definierten Sternpolygons.

Fächerradius: Dies ist der minimale Radius eines Kreises um den Fächermittelpunkt, der alle Randpunkte enthält. Alternativ könnte man ihn auch als den Abstand des Fächermittelpunktes zum am weitest entfernten Fächerandpunkt definieren.

Fächerspeiche: Als Fächerspeiche wird die Strecke vom Fächermittelpunkt zu einem Randpunkt des Fächers bezeichnet.

1.3 Notationen

Im folgenden werden oft verwendete Variablenbezeichnungen aufgeführt.

d, r,...	Kleine kursiv gedruckte lateinische Buchstaben bezeichnen Längen oder Strecken.
α, β, γ	Kleine griechische Buchstaben bezeichnen Winkel.
a, b, c,...	Kleine lateinische Buchstaben bezeichnen Punkte.
n,...	Kleine fett gedruckte lateinische Buchstaben bezeichnen Vektoren.
A, L,...	Große lateinische Buchstaben bezeichnen Mengen, Matrizen und geometrische Objekte.

p	Mit einem p wird meist der gerade zu bearbeitende Punkt bezeichnet
c	Mit einem c wird meist der Mittelpunkt eines Fächers bezeichnet (engl: center)
n	n bezeichnet einen Nachbarpunkt
r	r kennzeichnet einen Randpunkt eines Fächers
n', r', p'	der Strich kennzeichnet eine Abbildung des ursprünglichen Punktes in eine Tangentialebene. n' ist also z.B. die Abbildung von n
$a << b$	a ist sehr viel kleiner als b

2 Zugrundeliegende und weiterführende Arbeiten

Das Problem, durch Punktwolken repräsentierte Objekte zu rekonstruieren, ist schon oft in vielfältigen Ansätzen behandelt worden. Diese Arbeit befasst sich mit dem Ansatz von Lars Linsen [Linsen '01], der hier weiterentwickelt wird. Trotzdem soll hier aus den im Überblick genannten Gründen eine Zusammenfassung der anderen aktuellen Arbeiten zu diesem Thema gegeben werden.

Die in den Zusammenfassungen oft erwähnte Delaunay-Triangulierung wird in Kapitel 3.3 kurz beschrieben und kann ausführlich bei [Schmitt '96] und [Herrmann '98] nachgeschlagen werden.

2.1 Grundlage

„Oberflächenrepräsentation durch Punktwolken“ – Lars Linsen

Wie schon angedeutet bilden die Basis dieser Arbeit die Erfahrungen zur Visualisierung einer Punktwolke in der Dissertation von Lars Linsen [Linsen '01]. In seiner Arbeit beschäftigt er sich hierbei nicht nur mit der Visualisierung der Punktwolken, sondern auch mit den Möglichkeiten, diese ohne weitere Zusatzinformation (z. B. Oberflächennormalen, Dreiecksnetz, u.a.) zu bearbeiten. Es werden Methoden betrachtet, die Punktwolke bzw. das durch sie repräsentierte Objekt zu glätten, ohne charakteristische Merkmale zu zerstören. Des weiteren werden Verfahren entwickelt, die Auflösung der Punktwolkendarstellung zu verringern oder sie bei Bedarf zu erhöhen und dabei auch wieder die Charakteristika des Objektes zu erhalten. Ein weiteres Kapitel beschäftigt sich mit der Modellierung von Objekten, die durch Punktwolken repräsentiert werden. Hierbei werden einerseits Methoden untersucht, die Punktwolke in geringer Auflösung zu bearbeiten und die Änderungen auf eine höhere Auflösung zu übertragen. Des Weiteren werden Schnitte, Vereinigungen und Differenzen von Punktwolken realisiert. Die Arbeit schließt mit Betrachtungen zu interagierenden Partikelsystemen.

Die in der Dissertation hergeleitete Visualisierung einer Punktwolke, die in einem weiteren Schritt zu einer nahezu korrekten Triangulation weiterverarbeitet wird, soll nun in dieser Arbeit weiterentwickelt werden. Bei der Visualisierung wurde jeder Punkt von einem Dreiecksfächer umgeben, der mit einer festen Anzahl seiner räumlichen Nachbarn erzeugt wird. Eine Ausgleichsebene durch diese Nachbarn wird

erzeugt, um eine Normalenrichtung, bzw. Tangentialebene für den gerade betrachteten Punkt zu bestimmen. Die Nachbarn werden auf diese Tangentialebene projiziert um zu prüfen, ob die Anzahl der Nachbarn ausreicht, einen vollständigen Fächer zu bilden. Hierfür wird in der Projektion des Fächers getestet, ob die Winkel zwischen den Speichen des Fächers einen festen Wert nicht überschreiten. Um aus den Fächern eine Triangulation der Punktwolke zu erzeugen, werden im folgenden Optimierungsschritt mehrfach erzeugte Dreiecke entfernt und von mehreren Dreiecken abgedeckte Flächen ermittelt und auf die zur vollständigen Objektüberdeckung nötigen Dreiecke beschränkt.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 35.000 Punkten in etwa 8 Sekunden (PC mit Athlon K7 800Mhz).

2.2 Ähnliche Ansätze

Es gibt weitere Ansätze die Punktwolken oder Teile davon auf einen zweidimensionalen Raum abzubilden, und die dort ermittelten oder erzeugten Relationen der Punkte in den dreidimensionalen zurückzutransferieren.

„Surface Reconstruction based on Lower Dimensional Localized Delaunay Triangulation“ – M. Gopi und andere

Der Ansatz von M. Gopi, S. Krishnan und C.T. Silva, beschrieben in [Gopi et al. '00], ist dem hier entwickelten sehr ähnlich. Zunächst wird die Oberflächennormale in jedem Punkt der Punktwolke aus einer festen Anzahl von Nachbarpunkten ermittelt. Hierzu werden die Vektorprodukte der zu ermittelnden Normalen mit den Verbindungsvektoren zu den Nachbarpunkten betrachtet. Diese würden im Falle einer idealen planaren Fläche stets null ergeben. Die Varianz dieser Vektorprodukte wird in ein Minimierungsproblem überführt und nach der zu bestimmenden Normale aufgelöst.

Um nun eine Triangulation zu erzeugen, werden die Nachbarn jedes Punktes in dessen Tangentialebene gedreht. Dort werden sie nach dem Winkel ihrer zweidimensionalen Polarkoordinate in dieser Ebene sortiert. Nun wird eine Delaunaytriangulation um den Punkt erzeugt, indem für jeweils drei in der Winkelsortierung nebeneinander liegenden Punkte ein Voronoidiagramm erzeugt wird. Damit wird der mittlere Punkt auf

Delaunaynachbarschaft getestet.

Um mit diesem Algorithmus eine zufriedenstellende Triangulation zu erhalten wird in [Gopi et al. '00] eine recht strenge Anforderungen an die Punktwolke entwickelt. Es wird bewiesen, dass der Algorithmus eine korrekte Triangulation erzeugt, wenn die Krümmung zwischen zwei Punkten der Punktwolke beschränkt ist. Es wird also eine Punktwolke verlangt, bei der die Änderung der Normale (bzw. deren Produkt) eines Punktes zu der seiner Nachbarn konstant ist. Des weiteren muss der Abstand zur Rückseite des Objektes größer sein als der maximale Abstand zu den Nachbarn.

Da aber grundsätzlich nicht von einer der Objektkrümmung angepassten Punktdichte ausgegangen werden kann, werden noch ein paar Schritte zum Ausbessern der bisherigen Triangulation vorgenommen. Bei der oben beschriebenen Betrachtung eines Vierecks aus einem Punkt und drei seiner Nachbarn werden mehrdeutige Delaunay-Triangulationen abgefangen. Schließlich wird, nachdem jeder Punkt bearbeitet wurde, noch nach Löchern in der Struktur gesucht und diese je nach Größe geschlossen.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 35.000 Punkten in etwa 18 Sekunden (SGI-Onyx mit R10000 194Mhz).

„Meshless parameterization and surface reconstruction“ – Michael S. Floater und Martin Reimers

Auch in diesem Ansatz (beschrieben in [Floater/Reimers '01]) wird das dreidimensionale Problem der Oberflächenrekonstruktion in einem zweidimensionalen Raum übertragen und dort gelöst. In ihrem Ansatz wird die Punktemenge in Randpunkte und innere Punkte aufgeteilt. Die Ermittlung der Randpunkte erfolgte manuell oder über eine lokale Delaunay-Triangulierung, ähnlich wie sie in dieser Arbeit oder in [Linsen '01] benutzt wird. Die Nachbarn eines Punktes werden auf dessen Tangentialebene projiziert und dort trianguliert. Ist der Punkt Randpunkt dieser lokalen Triangulation, so wird er in die Menge der Randpunkte aufgenommen. Dann werden die Randpunkte in einen zweidimensionalen Parameterraum projiziert, wo sie ein konvexes Gebilde formen. Die Projektion p_i' der inneren Punkte p_i ergibt sich aus einer parametrischen Darstellung aus den Projektionen n_j' der Nachbarpunkte n_j, die sich innerhalb eines festen Radius *d* befinden:

$$p_i' = \sum_{alle\ j} \lambda_{ij}\ n_j' \quad \text{mit} \quad \sum_{alle\ j} \lambda_{ij} = 1 \quad \text{für alle i und} \begin{matrix} 1 < i < \text{Anzahl innerer Punkte} \\ 1 < j < \text{Anzahl Nachbarpunkte} \end{matrix}$$

Diese Summen bilden ein Gleichungssystem aus dessen Lösung sich die Projektion ergibt. Zur Erzeugung des endgültigen Gitternetzes wird die Projektion trianguliert und die daraus entstandenen Verbindungen in den dreidimensionalen Raum übernommen. Michael S. Floater und Martin Reimers stellen dabei drei Möglichkeiten vor, die Parameter λ_{ij} zu erzeugen. In einer ersten Methode werden die λ_{ij} zu den Nachbarn auf ^{1}/(Anzahl der Randpunkte im Radius *d)* gesetzt. In einer zweiten Methode werden die λ_{ij} zusätzlich mit der Distanz zum gerade betrachteten Punkt p_i gewichtet. Die dritte Methode benutzt eine wie oben beschriebene lokale Triangulation des Punktes p_i und dessen Nachbarn n_j in der Tangentialebene des Punktes p_i. Den daraus verbleibenden Delaunay-Nachbarn werden dann Parameter λ_{ij} zugewiesen, die aus formerhaltenden Gewichten erzeugt werden, die Michael S. Floater in dem Artikel „Parametrization and smooth approximation of surface triangulations“ (Computer Aided Geometric Design 14, Seite 231-250) beschreibt.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 30.000 Punkten in etwa 40-50 Sekunden. (In [Floater/Reimers '01] werden leider keine Angaben zur Rechnerleistung gemacht.)

„Spiraling Edge: Fast Surface Reconstruction from Partially Organized Sample Points“ – Patricia Crossno und Edward Angel

Der Ansatz von Patricia Crossno und Edward Angel [Crossno/Angel '99] arbeitet zwar nicht mit Abbildungen der Punktwolke in einen zweidimensionalen Raum, ähnelt dem hier entwickelten Ansatz aber doch in einigen Punkten.

In ihrem Ansatz wird, anfangend mit einem Punkt und dessen nächsten Nachbarn, eine bereits triangulierte Oberfläche schrittweise an den Rändern erweitert. Hierzu werden zu einem Randpunkt seine Nachbarn bestimmt und mit einer Auswahl daraus ein Dreicksfächer gebildet. Der bearbeitete Punkt p wird markiert und steht für weitere Dreiecke nicht mehr zur Verfügung. Dafür werden die Randpunkte des Fächers in die Punktemenge, die den Rand repräsentiert, eingefügt. Ob ein Nachbar als Fächerrandpunkt angenommen wird, hängt von mehreren Faktoren ab. Die mit diesen

Nachbarn gebildete Fächerspeichen nehmen Winkel zu einem Referenzvektor ein (üblicherweise die Verbindung zum nächsten Punkt auf der bisherigen Flächenkante). Sie werden nach diesen Winkeln aufsteigend sortiert. Punkte, die in den Winkelbereich zwischen den beiden Randkanten des gerade zu bearbeitenden Punktes fallen, werden als Fächerrandpunkte abgelehnt. Ebenso werden Punkte nicht angenommen, die nicht in einer Umkugel, gebildet um den Fächermittelpunkt p und ihren Vorgänger und Nachfolger in der Winkelsortierung, liegen. Da zur Winkelsortierung von einer ungefähr planaren Umgebung von p ausgegangen wird, müssen noch weitere Tests erfolgen, um unerwünschte Kanten abzufangen. Zum einen werden Nachbarn, deren Normale um mehr als 60° von der Normale des Punktes p abweichen, abgelehnt. Wenn der potentielle Fächerrandpunkt Teil des Randes der bisherigen Triangulation ist, wird geprüft, ob dieser Punkt den aktuellen Punkt p in seinem Fächer aufnehmen würde. Löcher und Ränder, die zu dem Objekt gehören werden nochmals speziell abgefragt, um die Liste der Randpunkte konsistent zu halten.

Wenn der forschreitende Rand der gebildeten Fläche Löcher einschließt, werden diese erkannt wenn sie von nicht mehr als drei Punkten gebildet werden. Größere Löcher können nach Angaben von [Crossno/Angel '99] zu Überlappungen führen, wenn Punkte innerhalb dieser Löcher sich mit noch unbearbeiteten Punkten außerhalb der bisherigen Triangulation verbinden. Der Algorithmus braucht neben der Punktwolke zu jedem Punkt die Normale, seine Nachbarn und die Information, ob dieser Punkt ein Randpunkt eines Loches oder des Objektes ist.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 43.000 Punkten in 2,3 Sekunden.

2.3 Der Blick über den Tellerrand – andere Ansätze zur Triangulierung von Punktmengen

„Entwicklung eines Algorithmus zur optimalen Triangulation von 3D-Punktwolken“ – Thomas Herrmann

In seiner Diplomarbeit [Herrmann '98] gibt Thomas Herrmann einen kleinen Einblick in den der Triangulationsproblematik zugrunde liegenden mathematischen Bereich der Topologie, eine ausführliche Einleitung zur Delaunay-Triangulation sowie einen aus-

führlichen Überblick über die grundlegenden Algorithmen zur Rekonstruktion von durch Punktwolken repräsentierte Objekte. Es wird der Ansatz von M.-E. Algorri und F. Schmitt vorgestellt, der über eine geeignete Voxelbildung aus der Puntkwolke eine Triangulation erzeugt. Anschließend wird das Verfahren von H. Hoppe und anderen beschrieben, bei dem zu jedem Punkt eine Tangentialebene berechnet und über eine Distanzfunktion das Objekt mit vordefinierten Voxel-Objekten angenähert wird. Diese wird in weiteren Schritten verfeinert. Auch die oft zitierten α-Shapes von H. Edelsbrunner und E. Mücke werden beschrieben. In ihrem Ansatz gehen sie der Frage nach, wie sich eine Oberfläche eigentlich definiert. Sie gehen von einer dreidimensionalen Delaunay-Triangulation aus und prüfen für jede Fläche, ob in einer Umkugel um diese Fläche mit einem Radius α weitere Punkte existieren. Ist dies nicht der Fall, so wird diese Fläche als Teil der Oberfläche des Objektes betrachtet. Der Erfolg dieses Ansatzes hängt dabei von einer gleichmäßigen Verteilung der Punkte, sowie von einem günstig gewählten Radius α ab. Dieses Verfahren wird in der ebenfalls beschriebenen Arbeit von C. Bajaj, F. Bernardini und anderen über α-Solids weiterentwickelt. Sie befasst sich mit einem Verfahren, algorithmisch einen möglichst guten α-Wert zu ermitteln. Des weiteren werden zwei Ansätze von Jean Daniel Boissonnat dargestellt. Im ersten Ansatz werden aus einer dreidimensionalen Delaunay-Triangulation schrittweise innere Tetraeder entfernt, bis nur noch äußere Flächen -die Oberfläche des Objektes- übrig bleiben. Dieses Verfahren wird in in einer weiteren dargestellten Arbeit von Isselhard, Brunett und Schreiber verfeinert. Das zweite beschriebene Verfahren von Jean Daniel Boissonnat erzeugt eine Dreiecksnetz, das die Oberfläche eines Objektes annähert, in dem von einem Startdreieck ausgehend die Randkanten der bisherigen Triangulation mit neuen Punkten zu weiteren Dreiecken erweitert werden. Dabei werden nur die Dreiecke erzeugt, deren Normale minimal von denen der angrenzenden Dreiecke abweicht. Beschrieben wird auch ein Ansatz von R. Mencel und H. Müller. Dabei wird zunächst ein minimal spannender Baum über die Punktwolke erzeugt, der dann zu einem Dreiecksnetz geschlossen wird. Die letzte dargestellte Arbeit von G. Tusk und M. Leroy beschäftigt sich mit der Triangulation und Zusammensetzung von Objekten, zu denen nur Höhendaten aus verschiedenen Blickrichtungen vorliegen.

Zu seiner eigenen Implemetierung eines Verfahrens entwickelt Thomas Herrmann eine

Synthese und Weiterentwicklung einer dreidimensionalen Delaunay-Triangulation und den von C. Bajaj, F. Bernardini und anderen entwickelten α-Shapes.

„The Ball Pivoting Algorithm for Surface Reconstruction" – Fausto Bernardini und andere

Der in [Bernardini et al. '99] vorgestellte Algorithmus erzeugt ein Dreiecksnetz aus einer Punktwolke, bei der die Oberflächennormalen zu jedem Punkt bekannt sind.

Hierfür wird eine von einem Startdreieck ausgehende Fläche schrittweise erweitert. Dabei wird zu jeder Randkante der bisher triangulierten Fläche eine gedachte Kugel gebildet, die einem festen Radius *r* besitzt und in der die Kante eine Sehne bildet. Wenn diese Kugel nun um die Kante rotiert und dabei einen Punkt der Punktwolke streift, so stellt dieser Punkt einen Kandidaten zur Bildung eines neuen Dreiecks mit der „Drehkante" dar. Ob sich dieser Punkt nun zur Erzeugung eines Dreieck eignet, wird bei einem Vergleich der Normale mit der Normale der Kantenendpunkte ermittelt. Um die möglichen Schnittpunkte mit dieser Kugel schnell finden zu können, wird die Punktwolke in einem Vorverarbeitungsschritt in ein dreidimensionales Zellraster der Kantenlänge 2*r* einsortiert. Dieser Algorithmus stellt natürlich einige Anforderungen an die Punktwolkendaten. Zum einen muss die Oberflächennormale zu jedem Punkt bekannt sein. Des weiteren muss die Punktdichte hoch genug sein, damit eine Kugel mit Radius *r* potentielle Nachbarpunkte streifen kann. Schließlich darf der Krümmumgsradius der Krümmungen des Objektes nicht kleiner sein als der Kugelradius r, um z.B. Querverbindungen innerhalb des Objektes zu vermeiden. Nach Angaben von [Bernardini et al. '99] hat sich der Algorithmus als stabil gegenüber Fehlern in der Punktwolke und schnell im Ablauf erwiesen und kann mit Hilfe mehrerer Durchläufe des Algorithmus mit verändertem *r* für Punktwolken mit variabler Punktdichte verwendet werden.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 35.000 Punkten in etwa 2,1 Sekunden (PC mit Pentium II 450Mhz).

„Delaunay Based Shape Reconstruction from Large Data" – Tamal K. Dey, Joachim Giesen und James Hudson

Tamal K. Dey, Joachim Giesen und James Hudson haben ein Verfahren entwickelt,

um eine dreidimensionale Delaunay-Triangulierung über große Datenmengen zu realisieren und in [Dey et al. '01] beschrieben. Grundidee dieses Ansatzes bildet eine Aufteilung der Punktwolke in einen „Octree“, bei dem der Raum in Quader unterteilt wird, die eine feste Anzahl Punkte enthalten. Diese Quader werden in einer Baumstruktur verwaltet (näheres in Kapitel 3.4). Um eine Verbindung mit den in benachbarten Quadern erzeugten Dreiecksnetzen zu ermöglichen, erhält jeder Quader noch die Randpunkte seines Nachbarn. Innerhalb dieser Quader wird dann eine dreidimensionale Delaunay-Triangulation vorgenommen. Bevor die einzelnen Teiltriangulationen zusammengesetzt werden, werden noch nach innen weisende Randdreiecke (sie bilden einen spitzen Winkel zur generierten Teiloberfläche) entfernt. Vor der Triangulation müssen allerdings in einer Vorverarbeitung zuerst die zum Objekt gehörenden Ränder ermittelt werden, damit diese nicht beim Zusammensetzen der einzelnen Teiltriangulationen oder bei der teilweisen Bearbeitung irrtümlicherweise geschlossen werden.

In einem Beispiel trianguliert dieser Ansatz ein Objekt von ca. 330.000 Punkten in etwa 15 Minuten (PC mit Pentium III 733Mhz).

„Reconstruction and representation of 3D Objects with radial Basis Functions“ – J.C. Carr und andere

Der Ansatz von J.C. Carr und anderen [Carr et al. '01] stellt ein Beispiel dafür dar, durch eine Punktwolke angenäherte Objekte durch eine stetige und differenzierbare Funktion zu beschreiben. Die Funktion ergibt für jeden Punkt im Raum dessen gerichteten Abstand (ob der Punkt innen oder außen liegt) zur Objektoberfläche und wird implizit aus den Punktwolkenpunkten erzeugt (dort beträgt der Abstand 0). Die Erzeugung der Funktion wird noch verbessert, indem Punkte entlang der Normalen verschoben werden und so mit bekannten Abstand zur Berechnung hinzugenommen werden. Da die Berechnung dieser Funktion sehr aufwendig ist, werden einige Verfahren vorgestellt den Aufwand zu minimieren. Zum einen wird auf ein Verfahren von Greengard und Rohklin verwiesen, die Berechnung dieser Funktion durch Annäherungen zu beschleunigen, da die Funktion für die meisten Anwendungen nicht mathematisch exakt sein muss. Zum anderen wird die Punktemenge ausgedünnt, so dass nur ein notwendiges Minimum an Punkten für die Berechnung einer hinreichend

genauen Funktion herangezogen werden muss. Mit Hilfe der erzeugten Funktion können dann Dreiecksnetze in beliebiger Auflösung generiert werden. Sie läßt sich aber auch dazu verwenden, stark rauschende Punktdaten zu glätten und unvollständige Gitternetze zu schließen.

In einem Beispiel reduziert dieser Ansatz ein Objekt von ca. 13.000 Punkten auf 4.300 Funktionstützstellen und bestimmt die Funktion in etwa 97 Sekunden (PC mit Pentium III 550Mhz).

„Triangulations in CGAL" – Jean Daniel Boissonnat und andere

Die Bibliothek CGAL (Computational Geometry Algorithms Library) ist eine Gemeinschaftsentwicklung von acht europäischen Forschungsgruppen, und beinhaltet eine Reihe von Datenstrukturen und Methoden, die auf diesem Gebiet immer wieder Verwendung finden. Unter anderem beinhaltet die Bibliothek derzeit Methoden für einfache, Delaunay- und reguläre (für Punktdaten mit Gewichtungen) Triangulationen für den zwei- und dreidimensionalen Fall, erzwungene Triangulationen (mit vorgegebenen Kanten) und erzwungene Delaunay-Triangulationen nur für den zweidimensionalen Fall. [Boissonnat '00] beschreibt die verwendete Organisation der Datenstrukturen. Die Triangulationsalgorithmen erzeugen ein Dreiecksnetz, indem die zu bearbeitenden Punkte schrittweise zur bisherigen Triangulation hinzugefügt werden und diese dementsprechend angepasst wird. Die Bibliothek beinhaltet auch Methoden, vorhandene Triangulation zu überprüfen.

Die Bibliothek wurde für diese Arbeit noch nicht verwendet, wurde aber z.B. bei der hier beschriebenen Arbeit „Delaunay Based Shape Reconstruction from Large Data" von [Dey et al. '01] benutzt.

In einem Beispiel trianguliert die Bibliothek ein Objekt von ca. 145.000 Punkten in etwa 51 Sekunden (PC mit Pentium III 500Mhz).

3 Der Algorithmus

Grundlage bildete, wie schon in Kapitel 2.1 beschrieben, der Ansatz von Lars Linsen [Linsen '01] (Kap.2.3 S. 15-24) zur Visualisierung von durch Punktwolken angenäherte Objekte. Er erzeugte aus von den Punkten und ihren jeweiligen Nachbarn gebildeten Fächern eine schnellen Visualisierung einer Punktwolke. In einem anschließenden Schritt wurde die Fächermenge optimiert, um eine fast zufriedenstellenden Triangulation ohne Überschneidungen und mehrfachen Dreiecken zu erhalten.

In diesem Kapitel wird nun der daraus abgeleitete Algorithmus vorgestellt, der keine anschließende Nachbearbeitung mehr benötigt. Zunächst werden die Punktkonstellationen die zu einem unkorrekten Dreiecksnetz führen dargestellt. Danach wird Umkreiskriterium vorgestellt, welches die Güte eines erzeugten Dreiecksnetzes kennzeichnet. Daran schließt sich ein kleiner Exkurs in die planare Geometrie an, um einige Beziehungen zwischen Dreiecken, deren Innenwinkel und ihren Umkreisen herzuleiten. Die anschließende Vorstellung des Algorithmus befasst sich mit den Vorverarbeitungschritten, gibt eine Übersicht über die einzelnen Ablaufschritte des Algorithmus, erläutert danach die Funktion der einzelnen Schritte und geht kurz auf Lösungsansätze ein, die in vorherigen Versionen des Algorithmus verfolgt wurden. Das Kapitel schließt mit den aus dem Algorithmus hervorgehenden Anforderungen an die Punktwolke,damit eine zufriedenstellende Triangulation erstellen wird.

3.1 Definition der Problemstellung

Aufgabe des Algorithmus ist nun, zu jedem Punkt einen Fächer aus seinen Nachbarn zu bilden. Dabei sollte er aber bei der Bildung schon vorhandene Fächer mitzuberücksichtigen, so dass eine vollständige, korrekte und je nach Objekt geschlossene und damit auch eine zufriedenstellende Triangulation der Punktwolke ohne eine anschließende Nachbearbeitung entsteht.

Folgende Problemfälle bei der Bildung der Fächer sind also abzufangen:

Fall a) Ein zu bildendes Fächersegment (Dreieck) ist Bestandteil eines schon vorhandenen Fächers (Abbildung 3.1.1).

Fall b) Die Fächer, bzw. deren Dreiecke nähern das Objekt nicht vollständig an; die von der Triangulation aufgespannte Fläche besitzt „Löcher“ (Abbildung 3.1.2).

Fall c) Der zu bildende Fächer schneidet bisher erstellte Fächer oder ist teilweise in ihnen enthalten.(Abbildung 3.1.3 und Abbildung 3.1.5).

Fall d) Dreieckskanten des Fächers würden mehr als zwei Dreiecke begrenzen. Oft wird hiebei von unerwünschter Tetraederbildung gesprochen und geht mit einer Änderung des Geschlechts der erzeugten Fläche einher (Abbildung 3.1.7).

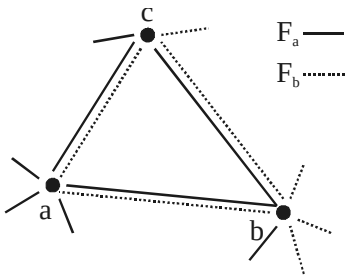


Abbildung 3.1.1: mehrfach definierte Dreiecke (△a b c) durch die Fächer F_a und F_b.

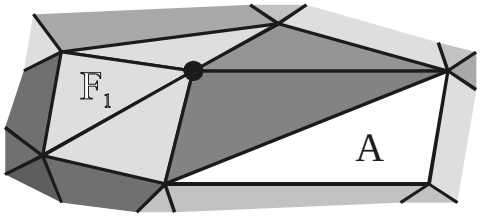


Abbildung 3.1.2: Unzureichende Abdeckung des Objektes durch das Dreiecksnetz. Es enthält Löcher: die Fläche A ist nicht abgedeckt.

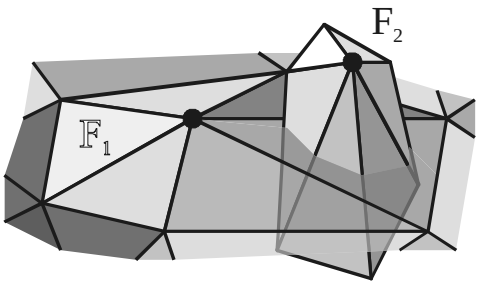


Abbildung 3.1.3: Der Fächer F_1 und der Fächer F_2 schneiden sich.

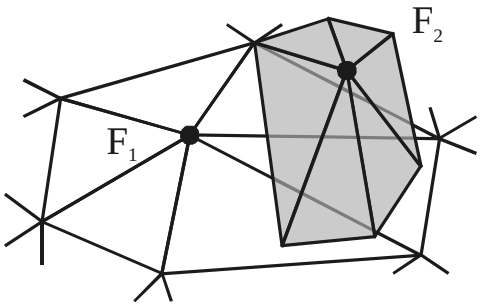


Abbildung 3.1.4: Schnitt zweier Fächer (F_1 und F_2) in der zweidimensionalen Abbildung

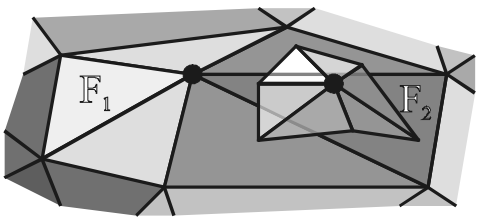


Abbildung 3.1.5: Ein Fächer (F_2) ist in einem anderen (F_1) enthalten.

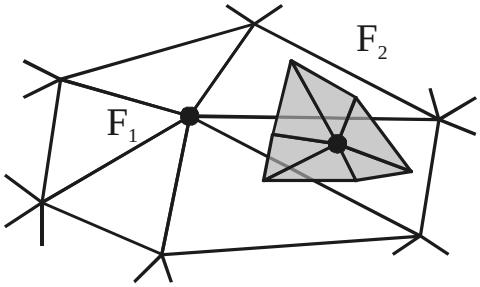


Abbildung 3.1.6: Fächerenthaltungen in der zweidimensionalen Abbildung.

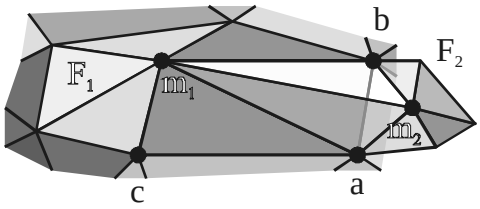


Abbildung 3.1.7: Tetraederbildung.

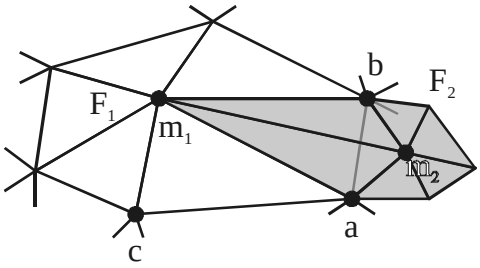


Abbildung 3.1.8: Tetraederbildung in der zweidimensionalen Abbildung.

Zu Abbildung 3.1.7 und Abbildung 3.1.8: Der Mittelpunkt m_1 des Fächers F_1 bildet mit a und b ein Dreieck des Fächers F_1. Der Mittelpunkt m_2 von Fächer F_2 bildet mit m_1 und a sowie mit b und m_1 zwei Dreiecke. Die Dreiecke umschließen ein (zwar nach m_2 b a offenes) dreidimensionales Gebilde. Das Geschlecht der vom Dreiecksnetz gebildeten Oberfläche hat sich erhöht. die Kante m_1 a begrenzt z. B. drei Dreiecke ($\triangle m_1$ a b, $\triangle m_1$ c a und $\triangle m_1$ a m_2), die sich dann in der zweidimensionalen Abbildung überschneiden.

Da der Algorithmus die Kanten der Triangulation in einer zweidimensionalen Abbildung eines Teils der Punktwolke ermittelt, müssen Konstellationen in diesem zweidimensionalen Raum erkannt werden, die zu den oben beschriebenen Problemen führen.

An Fall a) ändert sich auch in zweidimensionalen Abbildung nichts (Abbildung 3.1.1). Eine Mehrfachbelegung von schon bestehenden Verbindungen zwischen zwei Punkten, insbesondere eine mehrfache Erzeugung des gleichen Dreiecks, ist zu vermeiden.

Auch Fall b) zeigt sich analog in der zweidimensionalen Abbildung, wenn Punkte mit ihren näheren Nachbarn in der Oberfläche nicht verbunden werden (vgl. Abbildung 3.1.2).

Fall c) und d) sind in der zweidimensionalen Abbildung in Enthaltungen und Überschneidungen (Abbildung 3.1.4, Abbildung 3.1.6 und Abbildung 3.1.8) erkennbar. So ist also darauf zu achten, dass fertiggestellte Dreiecke in der Abbildung keine weiteren Elemente (Punkte, Kanten, Dreiecke oder sogar Fächer) enthalten und Kanten sich nicht schneiden.

3.2 Das Umkreiskriterium oder „schönere Dreiecke“

Eine Triangulation mag korrekt, vollständig und geschlossen sein, aber trotzdem den Ansprüchen des Anwenders nicht genügen. Nicht nur aus ästhetischen Gründen sind lange schmale Dreiecke, die sehr spitze Innenwinkel enthalten, nicht wünschenswert. Die kleinen Winkel führen oft zu numerischen Problemen und nach praktischen Erfahrungen erhöhen lang gezogene Dreiecke das Risiko der überlappenden Verbindungen.

Die hier schon oft genannte Delaunay-Triangulierung wird oft als Paradebeispiel einer „schönen Triangulierung“ aus möglichst vielen gleichschenkligen Dreiecken beschrieben. Sie wurde von Boris Delaunay*) als duale Zerlegung des Voronoi-Diagramms hergeleitet. Letzteres teilt den Raum anhand einer Menge M von Punkten in Gebiete nach minimalen Abständen zu diesen Punkten. D.h. ein Voronoi-Gebiet zu einem Punkt p enthält alle Punkte des Raums die näher an p liegen als zu anderen Punkten der Punktemenge M. Bei der Delaunay-Triangulierung werden die Punkte mit einer Kante verbunden, deren Voronoi-Gebiete benachbart sind. Ausführliche Beschreibungen zu diesem Thema sind zum Beispiel in [Schmitt '96] und [Herrmann '98] zu finden. Im Zusammenhang mit den Delaunay-Triangulierungen werden oft das Umkreis- oder Umkugelkriterium erwähnt, deren die Delaunay-Triangulation genügt und die Dreiecksnetze mit einer optimalen Aufteilung bezüglich der Gleichschenklichkeit kennzeichnen. Dabei gilt eine Triangulierung als optimal, wenn sich im Umkreis (Umkugel) jedes Dreiecks kein weiterer Punkt der Punktemenge befindet. Punkte, die sich in einem solchen Umkreis befinden, werden manchmal auch als Delaunay-Nachbarn bezeichnet.

*) Boris Delaunay: „sur la sphère vide“, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskii i Estes vennyka Nauk, 7: 793-800, 1934.

3.3 Sätze für Dreiecke und deren Umkreis

Da im Algorithmus die Speichen eines zu bildenden Fächers nach Winkel sortiert abgespeichert werden, werden im folgenden einige Beziehung zwischen Dreiecken, ihren Innenwinkeln und ihren Umkreisen hergeleitet, um diese berechneten Winkel weiter zu verwenden. Grundlage bilden hierfür hauptsächlich der Satz des Thales und der Sinussatz beliebiger Dreiecke.

Satz des Thales:

Die Winkel γ, die die Strecken vom einem beliebigen Punkt c des Bogens zu dessen Endpunkten a und b bilden, sind gleich groß und halb so groß wie der zugehörige Mittelpunktswinkel. Spezialfall bildet ein Mittelpunktswinkel von 180°: Der Mittelpunkt liegt in der Mitte der Strecke a b, wobei der Bogen einen Halbkreis über diese Strecke bildet. In diesem Fall bilden alle Punkte des Bogens einen Winkel von 90° mit den Endpunkten.

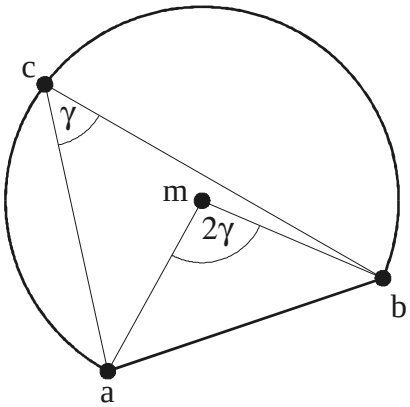


Abbildung 3.3.1: Satz des Thales

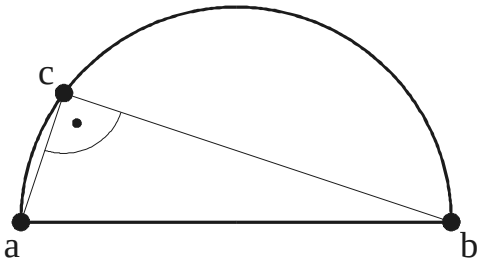


Abbildung 3.3.2: Spezialfall des Satz des Thales: Innenwinkel im Halbkreis immer 90°

Sinussatz:

Gegeben sei ein Dreieck mit den Kantenlängen *a*, *b*, und *c*, sowie deren gegenüberliegenden Winkeln α, β, und γ. Dann gilt:

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma} = 2 \cdot Umkreisradius$$

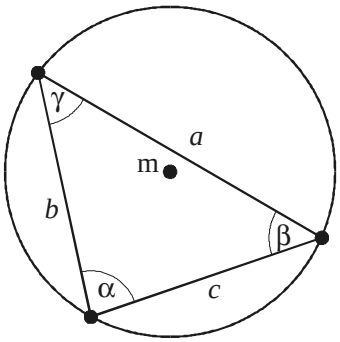


Abbildung 3.3.3: Sinussatz

Satz 3.3.1: gegenüberliegende Winkel

Gegeben seien im zweidimensionalen euklidischen Raum eine Sehne s und ein Kreis U. Die Schnittpunkte der Sehne mit U seien a und b mit a ≠ b. Sei c ein Punkt auf U im Kreisbogen von b nach a. Sei c' ein Punkt auf U im Kreisbogen von a nach b. γ sei der Winkel $\sphericalangle$acb, γ' sei der Winkel $\sphericalangle$bc'a. Dann gilt:

$\gamma = 180° - \gamma'$ mit $0° \le \gamma, \gamma' \le 180°$.

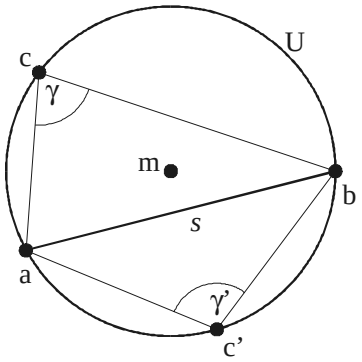


Abbildung 3.3.4: Satz 3.3.1: Gegenüberliegende Umkreiswinkel

Beweis:

$0° \le \gamma, \gamma' \le 180°$:

a,b,c und a,b,c' bilden Dreiecke, deren Innenwinkel positiv sind und zusammen 180° ergeben. Somit können diese einzeln nicht größer als 180° sein.

$\gamma = 180° - \gamma'$:

Nach dem allgemeinen Satz des Thales besitzt für feste a,b und beliebige c der Winkel γ den gleichen Wert. Dies gilt damit auch für beliebige c', bei denen der gleiche Winkel γ' auftritt. Seien nun d, d' Schnittpunkte des Umkreises U mit der Mittelsenkrechte t der Sehne s. Auch bei ihnen treten die Winkel γ, bzw. γ' auf. t enthält

alle Punkte, die von a und b den gleichen Abstand besitzen, somit auch den Mittelpunkt M von U. t teilt somit U in zwei Halbkreisbögen. Nach dem Spezialfall des Satz des Thales ist der Winkel über einen Halbkreis immer ein rechter. In den Dreiecken d a d' und d' b d betragen demnach die Innenwinkel bei a und b 90°. In einem Viereck d a d' b (dessen Innenwinkel zusammen 360° ergeben) ergeben demnach die Innenwinkel bei d und d' zusammen 180°. => $\gamma + \gamma' = 180°$.

□

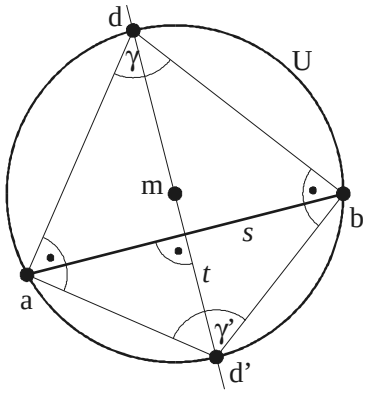


Abbildung 3.3.5: Beweis Satz 3.3.1 über Seitenhalbierende von ab

Satz 3.3.2: Punktlage bezüglich des Kreisbogens

Gegeben seien paarweise verschiedene Punkte a und b im zweidimensionalen euklidischen Raum. Dazu sei ein beliebiger Kreisbogen U über der Strecke a,b, welche zusammen eine Fläche A beinhalten. d sei ein beliebiger Punkt auf den Kreisbogen. Der Innenwinkel beim Punkt d eines Dreiecks $\triangle$a b d ist φ'. Die Gerade durch a und b teile den Raum in zwei Halbräume H_1 und H_2. Ohne Beschränkung der Allgemeinheit befinde sich der Kreisbogen im Halbraum H_1. Für einen beliebigen Punkt $p \in H_1$ gilt:

Der Innenwinkel φ bei Punkt p des Dreiecks a b p ist größer als der Winkel φ' genau dann, wenn $p \in A$.

Der Innenwinkel φ bei Punkt p des Dreiecks a b p ist kleiner als der Winkel φ' genau dann, wenn $p \notin \{A \cup U\}$.

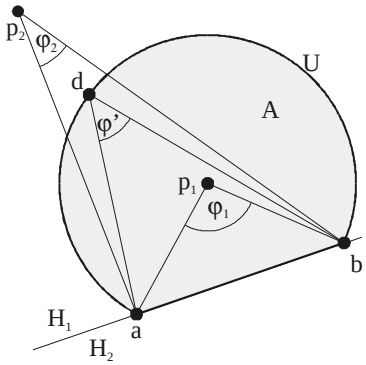


Abbildung 3.3.6: Satz 3.3.2

$\varphi_1 > \varphi' \Leftrightarrow p_1 \in A$

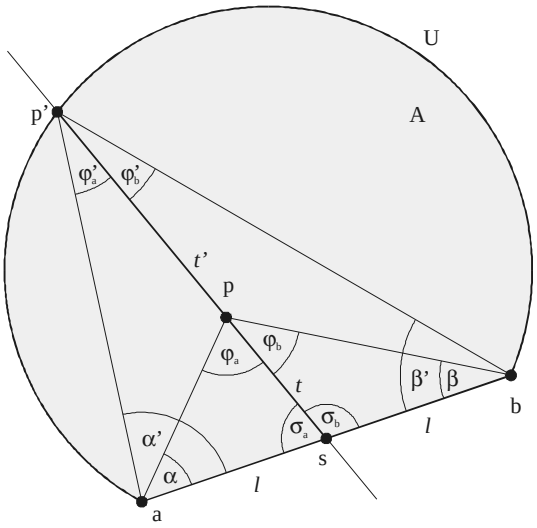
$\varphi_2 < \varphi' \Leftrightarrow p_2 \notin \{A \cup U\}$

Beweis:

Sei s der Mittelpunkt der Strecke a b. Dann existiert eine Gerade durch s und p,

welche U in einem Punkt p' schneidet. Der Abstand von s nach p sei *t*, der von s nach p' sei *t'* und der Abstand von a und b sei 2*l*. Ferner seien im Dreieck △a s p die Innenwinkel der Eckpunkte α, σ_a und φ_a; im Dreieck △a s p' α' σ_a und φ_a'. Analog seien im Dreieck △b p s und △b p' s die Innenwinkel β φ_b und σ_b, bzw. β' φ_b' und σ_b. (siehe Abbildung 3.3.7 und Abbildung 3.3.8)

Abbildung 3.3.7: Beweis Satz 3.3.2 - $p \in A$



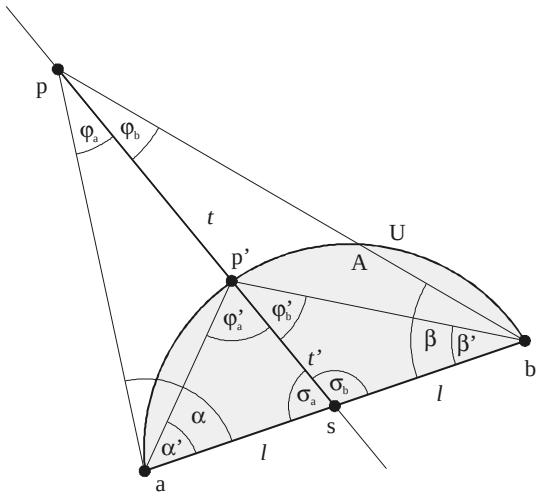


Abbildung 3.3.8: Beweis Satz 3.3.2 - $p \notin \{A \cup U\}$

Damit gilt: $0° < \alpha, \alpha', \beta, \beta', \sigma_a, \sigma_b, \varphi_a, \varphi_a', \varphi_b, \varphi_b', \varphi, \varphi' < 180°$,

$\varphi = \varphi_1 + \varphi_b$ und $\varphi' = \varphi_1' + \varphi_b'$.

Der Winkel σ_a ist in den Dreiecken △a s p und △a s p' identisch.

Der Winkel σ_b ist in den Dreiecken △b p s und △b p' s identisch.

Im Dreieck a s p gilt:

$$\frac{l}{\sin\varphi_a} = \frac{t}{\sin\alpha} \qquad \text{(Sinussatz)}$$

$$= \frac{t}{\sin(180° - (\sigma_a + \varphi_a))}$$

$$= \frac{t}{\sin(\sigma_a + \varphi_a)} \qquad (0° < \sigma_a + \varphi_a < 180° \text{ da Innenwinkel eines Dreiecks})$$

$$t = l\frac{\sin(\sigma_a+\varphi_a)}{\sin\varphi_a} = l\frac{\sin\sigma_a\cos\varphi_a+\cos\sigma_a\sin\varphi_a}{\sin\varphi_a}$$

$$= l\left(\frac{\sin\sigma_a\cos\varphi_a}{\sin\varphi_a}+\frac{\cos\sigma_a\sin\varphi_a}{\sin\varphi_a}\right)$$

$$= \frac{1}{\tan\varphi_a}\ (l\ \sin\sigma_a) + l\ \cos\sigma_a$$

$$= \frac{1}{\tan\varphi_a}\ f_a + s_a$$

Mit $f_a := l \sin\sigma_a$ und $s_a := l\cos\sigma_a$
und $0 < f_a < l$ und $-l < s_a < l$

Analog gilt im Dreieck a s p':

$$t' = \frac{1}{\tan\varphi'_a}\ f_a + s_a$$

Analog gilt im Dreieck b p s:

$$t = \frac{1}{\tan\varphi_b}\ f_b + s_b$$

Mit $f_b := l \sin\sigma_b$ und $s_b := l\cos\sigma_b$
und $0 < f_b < l$ und $-l < s_b < l$

Analog gilt im Dreieck b p' s:

$$t' = \frac{1}{\tan\varphi'_b}\ f_b + s_b$$

1. zu Beweisen: $p \notin \{A \cup U\} \Rightarrow \varphi < \varphi'$

   p liegt außerhalb des Kreisbogens:

   $\Rightarrow \quad t > t'$

   $$\Rightarrow \quad \frac{1}{\tan\varphi_a}\ f_a + s_a > \frac{1}{\tan\varphi'_a}\ f_a + s_a \text{ und}$$

   $$\frac{1}{\tan\varphi_b}\ f_b + s_b > \frac{1}{\tan\varphi'_b}\ f_b + s_b$$

   $$\Rightarrow \quad \frac{1}{\tan\varphi_a} > \frac{1}{\tan\varphi'_a} \text{ und } \frac{1}{\tan\varphi_b} > \frac{1}{\tan\varphi'_b}$$

$\Rightarrow$ $\tan \varphi_a < \tan \varphi_a'$ und $\tan \varphi_b < \tan \varphi_b'$

$\Rightarrow$ $\varphi_a < \varphi_a'$ und $\varphi_b < \varphi_b'$

$\Rightarrow$ $\varphi_a + \varphi_b < \varphi_a' + \varphi_b'$

$\Rightarrow$ $\varphi < \varphi'$

2. zu Beweisen: $p \in A \Rightarrow \varphi > \varphi'$

   p liegt innerhalb des Kreisbogens:

   $\Rightarrow$ $t < t'$

   $\Rightarrow$ $\frac{1}{\tan \varphi_a} f_a + s_a < \frac{1}{\tan \varphi'_a} f_a + s_a$ und

   $\frac{1}{\tan \varphi_b} f_b + s_b < \frac{1}{\tan \varphi'_b} f_b + s_b$

   $\Rightarrow$ $\frac{1}{\tan \varphi_a} < \frac{1}{\tan \varphi'_a}$ und $\frac{1}{\tan \varphi_b} < \frac{1}{\tan \varphi'_b}$

   $\Rightarrow$ $\tan \varphi_a > \tan \varphi_a'$ und $\tan \varphi_b > \tan \varphi_b'$

   $\Rightarrow$ $\varphi_a > \varphi_a'$ und $\varphi_b > \varphi_b'$

   $\Rightarrow$ $\varphi_a + \varphi_b > \varphi_a' + \varphi_b'$

   $\Rightarrow$ $\varphi > \varphi'$

3. zu Beweisen: $\varphi < \varphi' \Rightarrow p \notin \{A \cup U\}$

   Beweis über Widerspruch:

   Wäre $p \in A$ dann würde nach 2. gilt dann $\varphi > \varphi'$ - Widerspruch zu $\varphi < \varphi'$

   Wäre $p \in U$ dann würde nach dem Satz des Thales gilt dann $\varphi = \varphi'$ - Widerspruch zu $\varphi < \varphi'$

   $\Rightarrow \varphi < \varphi' \Rightarrow p \notin \{A \cup U\}$

4. zu Beweisen: $\varphi > \varphi' \Rightarrow p \in A$

   Beweis über Widerspruch:

   Wäre $p \in U$, würde nach dem Satz des Thales gelten $\varphi = \varphi'$

   - Widerspruch zu $\varphi > \varphi'$

Wäre $p \notin \{A \cup U\}$, dann würde nach 1. gelten $\varphi < \varphi'$

- Widerspruch zu $\varphi > \varphi'$

□

Satz 3.3.3: Umkreisenthaltung

Gegeben seien paarweise verschiedene Punkte a, b, c und p im zweidimensionalen euklidischen Raum. Die Punkte a, b, c bilden ein Dreieck mit einem Umkreis U, dessen Radius *r* sei. γ sei der Winkel $\sphericalangle bca$, φ der orientierte Winkel $\sphericalangle apb$. ($0° < \gamma < 180°$, $0° \leq \varphi \leq 360°$) Die Gerade durch a und b teile den Raum in zwei Halbräume H_1 und H_2. Dann gilt:

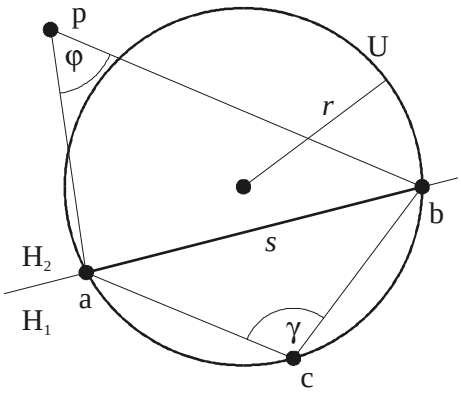


Abbildung 3.3.9: Satz 3.3.3: Der Innenwinkel bei p im Vergleich zum Innenwinkel bei c

- Für $\varphi = 360°$ oder $\varphi = 0°$ liegt p nicht im Umkreis U.
- Für $\varphi = 180°$ liegt p im Umkreis.
- Für $0° < \varphi < 180°$:

  Der Punkt p liegt im Umkreis U genau dann, wenn $\varphi > 180° - \gamma$

  Der Punkt p liegt außerhalb vom Umkreis U genau dann, wenn $\varphi < 180° - \gamma$

  Der Punkt p liegt auf dem Umkreis U genau dann wenn wenn $\varphi = 180° - \gamma$

- Für $180° < \varphi < 360°$:

  Der Punkt p liegt im Umkreis U genau dann, wenn $360° - \varphi > \gamma$

  Der Punkt p liegt außerhalb vom Umkreis U genau dann, wenn $360° - \varphi < \gamma$

  Der Punkt p liegt auf dem Umkreis U genau dann wenn wenn $360° - \varphi = \gamma$

Beweis:

1. Für φ = 360° oder φ = 0°:

   Die Punkt a,b und p liegen auf einer Geraden. Die Verbindungsvektoren a-p und b-p weisen in die selbe Richtung. p liegt nicht auf der Strecke von a nach b und damit nicht im Umkreis.

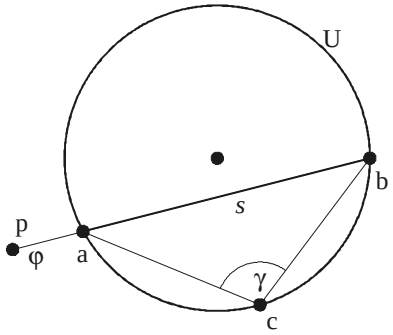


Abbildung 3.3.10: Beweis Satz 3.3.3 - $\varphi = 0°$

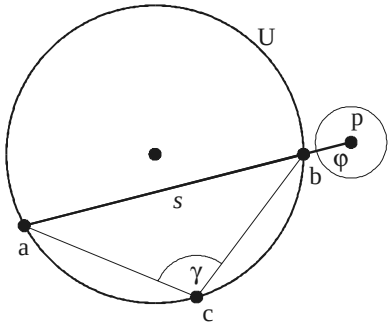


Abbildung 3.3.11: Beweis Satz 3.3.3 - $\varphi = 360°$

2. Für φ = 180°:

   Die Punkt a,b und p liegen auf einer Geraden. Die Verbindungsvektoren a-p und b-p weisen in die entgegengesetzte Richtung. p liegt auf der Strecke von a nach b und damit im Umkreis.

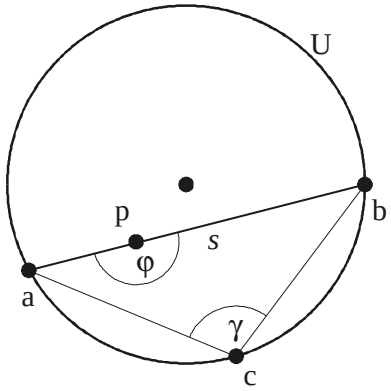


Abbildung 3.3.12: Beweis Satz 3.3.3 - $\varphi = 180°$

3. Für $0° < \varphi < 180°$:

   p und c sind nicht Elemente des gleichen Halbraumes.

   Ohne Beschränkung der Allgemeinheit liege p in H_2 und c in H_1.

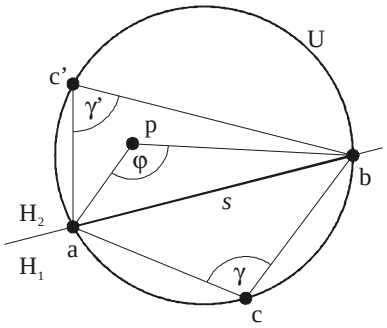


Abbildung 3.3.13: Beweis Satz 3.3.3 - $0° < \varphi < 180°$ - p im Umkreis

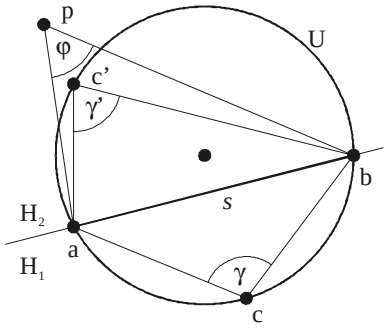


Abbildung 3.3.14: Beweis Satz 3.3.3 - $0° < \varphi < 180°$ - p nicht im Umkreis

   Laut Satz 3.3.1 bilden alle Punkte $c' \in \{U \cap H_2\}$ den Winkel $\gamma' = \sphericalangle bc'a$ mit $\gamma' = 180° - \gamma$.

   $\Rightarrow$ p liegt auf dem Umkreis U $\Leftrightarrow \varphi = 180° - \gamma$.

   Weiterhin folgt aus Satz 3.3.2:

   p liegt im Umkreis U $\Leftrightarrow \varphi > \gamma'$.

   $\Rightarrow$ p liegt im Umkreis U $\Leftrightarrow \varphi > 180° - \gamma$.

   p liegt außerhalb vom Umkreis U $\Leftrightarrow \varphi < \gamma'$.

   $\Rightarrow$ p liegt außerhalb vom Umkreis U $\Leftrightarrow \varphi < 180° - \gamma$.

4. Für $180° < \varphi < 360°$:

   p und c sind Elemente des gleichen Halbraumes.

   Sei $\varphi' = 360° - \varphi$ der Winkel $\sphericalangle apb$.

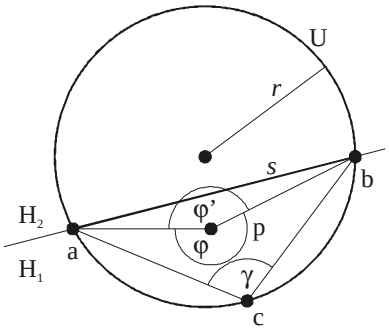


Abbildung 3.3.15: Beweis Satz 3.3.3 - 180°< φ < 360° - p im Umkreis

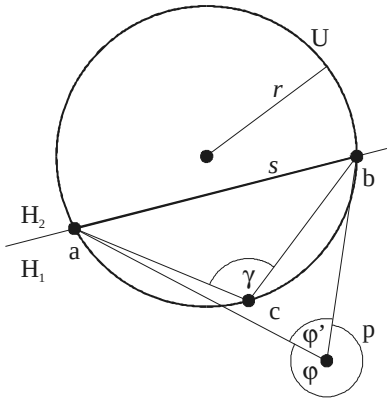


Abbildung 3.3.16: Beweis Satz 3.3.3 - 180°< φ < 360° - p nicht im Umkreis

Dann folgt aus dem Satz des Thales

p liegt auf dem Umkreis U $\Leftrightarrow \varphi' = \gamma$

$\Rightarrow$ p liegt auf dem Umkreis U $\Leftrightarrow 360° - \varphi = \gamma$

Weiterhin folgt aus Satz 3.3.2:

p liegt im Umkreis U $\Leftrightarrow \varphi' > \gamma$

$\Rightarrow$ p liegt im Umkreis U $\Leftrightarrow 360 - \varphi > \gamma$.

p liegt außerhalb vom Umkreis U $\Leftrightarrow \varphi' < \gamma$

$\Rightarrow$ p liegt außerhalb vom Umkreis U $\Leftrightarrow 360 - \varphi < \gamma$.

□

Satz 3.3.4: gegenseitige Umkreisenthaltung

Gegeben seien paarweise verschiedene Punkte a, b, c und p im zweidimensionalen euklidischen Raum. Die Punkte a, c, b bilden ein Dreieck mit einem Umkreis U_{bca}. Die Punkte b, p, a bilden ein Dreieck mit einem Umkreis U_{apb}. γ sei der Winkel $\sphericalangle$acb, φ der Winkel $\sphericalangle$bpa. ($0° < \gamma, \varphi < 180°$). Die Gerade durch a und b teile den Raum in zwei Halbräume H_1 und H_2. Dann gilt:

c liegt im Umkreis U_{apb} genau dann, wenn auch p im Umkreis U_{bca} liegt.

c liegt nicht im Umkreis U_{apb} genau dann, wenn auch p nicht im Umkreis U_{bca} liegt.

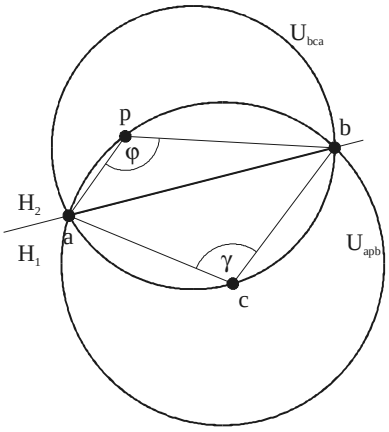


Abbildung 3.3.17: Satz 3.3.4 -
$c \in U_{apb} \Leftrightarrow p \in U_{bca}$

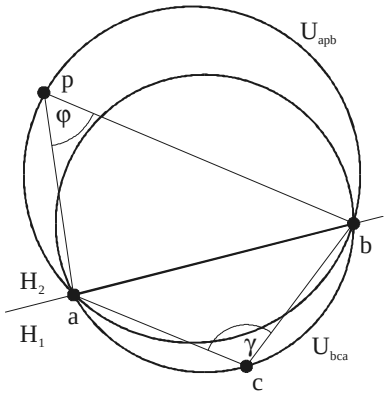


Abbildung 3.3.18: Satz 3.3.4 -
$c \notin U_{apb} \Leftrightarrow p \notin U_{bca}$

Beweis:

Da $0° < \gamma, \varphi < 180°$ liegen c und p in verschiedenen Halbräumen.

c liegt im Umkreis U_{apb}
(nach Satz 3.3.3) $\Leftrightarrow \gamma > 180° - \varphi$
$\Leftrightarrow \gamma - 180° > -\varphi$
$\Leftrightarrow 180° - \gamma < \varphi$
(nach Satz 3.3.3) $\Leftrightarrow$ p liegt im Umkreis U_{bca}

Negation:

c liegt nicht im Umkreis U_{apb}
(nach Satz 3.3.3) $\Leftrightarrow \gamma < 180° - \varphi$
$\Leftrightarrow \gamma - 180° < -\varphi$
$\Leftrightarrow 180° - \gamma > \varphi$
(nach Satz 3.3.3) $\Leftrightarrow$ p liegt nicht im Umkreis U_{bca}

□

Satz 3.3.5: Enthaltungen im Umkugeläquator

Gegeben seien paarweise verschiedene Punkte a, b, c und p im dreidimensionalen euklidischen Raum. Die Punkte a, b, c bilden ein Dreieck mit einer Umkugel U, dessen Radius r sei. U' sei der Umkreis in der Dreiecksebene E_{abc} von a, b und c. p' sei die Drehung des Punktes p in die Dreiecksebene E_{abc} mit Drehzentrum a,b oder c in einer senkrecht auf E_{abc} stehenden Rotationsebene E_{rot}.

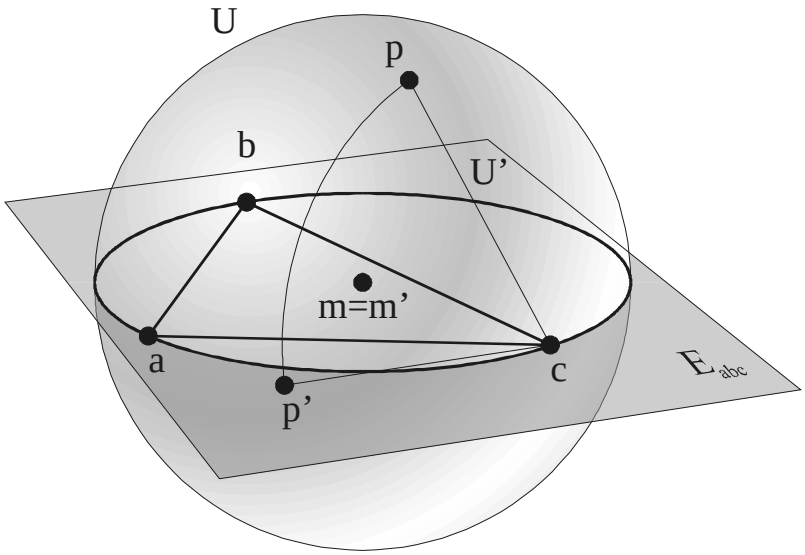


Abbildung 3.3.19: Satz 3.3.5: $p' \notin U' \Rightarrow p \notin U$

Ist p' nicht innerhalb oder auf dem Umkreis U', dann ist auch p nicht in der Umkugel U.

Beweis:

Eigenschaften dieses Aufbaus:

Der Mittelpunkt m von U liegt in E_{abc}, und ist somit mit dem Mittelpunkt m' von U' identisch.

Sei ohne Beschränkung der Allgemeinheit c das Drehzentrum für die Drehung von p nach p'.

Es existiert eine Gerade g durch c und p'. Sie ist auch das Ergebnis des Schnittes der Dreiecksebene E_{abc} mit der darauf senkrecht stehenden Rotationsebene E_{rot} des Punktes p in die Dreiecksebene. Wenn g keine Tangente an U' ist, dann existiert ein Schnittpunkt $c' \neq c$ mit dem Umkreis U', sonst sei $c' = c$. Der Schnitt von E_{rot} mit der Umkugel U ergibt einen Kreis U_s', dessen Mittelpunkt m_s gleich dem Mittelpunkt der Strecke von c nach c' ist. Die Länge d_s dieser Strecke ist gleich dem Durchmesser von U_s.

Dann gilt:

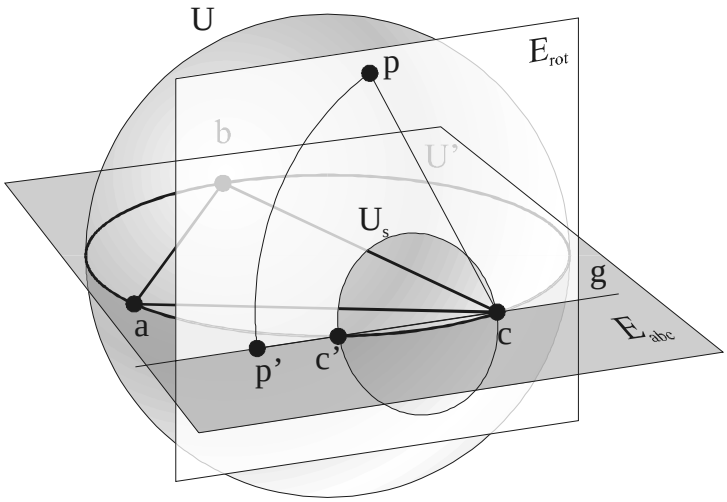


Abbildung 3.3.20: Beweis Satz 3.3.5 Punkt 1: dreidimensionale Ansicht.

1. Wenn c' auf der Strecke von c nach p' liegt (Abbildung 3.3.20), dann ist die Länge der Strecke d_s von c nach c' kleiner als die Länge der Strecke r_{rot} von c nach p', da p' nicht innerhalb oder auf dem Umkreis liegt. p liegt auf einem Kreis U_p auf der Rotationebene mit dem Radius r_{rot} und dem Mittelpunkt c. Der Kreis U_p schließt den Schnittkreis U_s ein, ohne diesen zu schneiden oder zu berühren, da $c \in U_s$ und $d_s < r_{rot}$. $\Rightarrow p \notin U$.

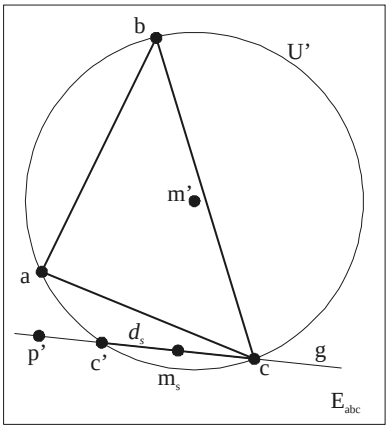


Abbildung 3.3.21: Beweis Satz 3.3.5 Punkt 1: Ansicht der Dreiecksebene E_{abc}.

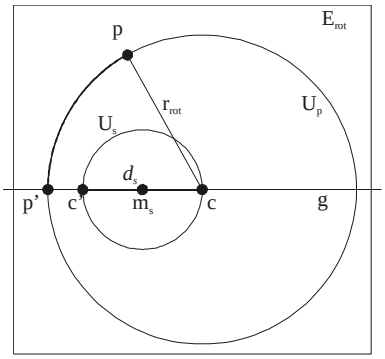


Abbildung 3.3.22: Beweis Satz 3.3.5 Punkt 1: Ansicht der Rotationsebene E_{rot}.

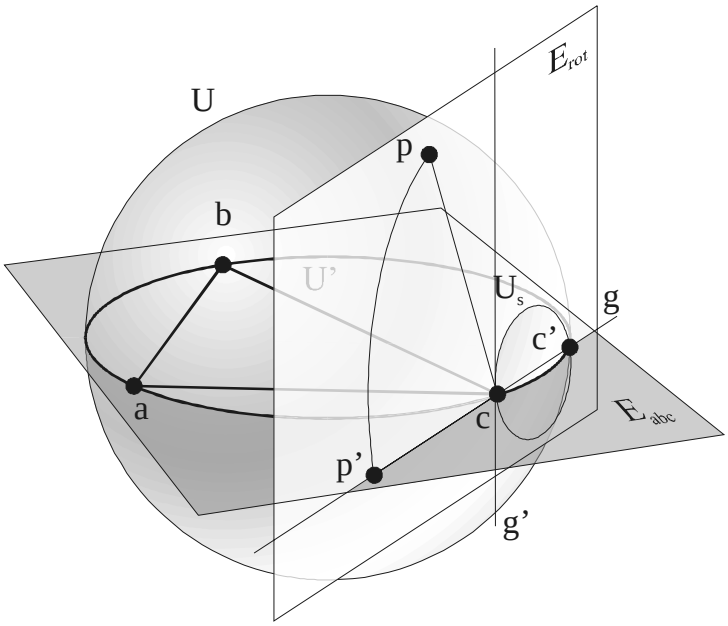


Abbildung 3.3.23: Beweis Satz 3.3.5 Punkt 2: dreidimensionale Ansicht.

2. Wenn c' nicht auf der Strecke von c nach p' liegt (Abbildung 3.3.23), dann teilt eine Gerade $g' \perp g$ mit $g' \in E_{rot}$ und $c \in g'$ die Ebene E_{rot} in zwei Halbebenen $H_{c'}$ und H_p. Die Punkte c' und p sind dann nicht Element der gleichen Halbebene. Sei $c' \in H_{c'}$ und $p \in H_p$. Somit gilt:

   Der Schnitt von E_{rot} mit der Umkugel U (U_s) ist in nur in der Halbebene $H_{c'}$ enthalten und hat mit der Gerade g' nur den Punkt c gemeinsam:

   $c' \in H_{c'} \Rightarrow U_s \in H_{c'} \setminus (g' \setminus \{c\})$

   Der Punkt p kann wegen der Rotation nur in der Halbebene H_p oder auf $g' \setminus \{c\}$ (da $p \neq c$) liegen. Somit ist $p \notin U_s$ und damit $p \notin U$.

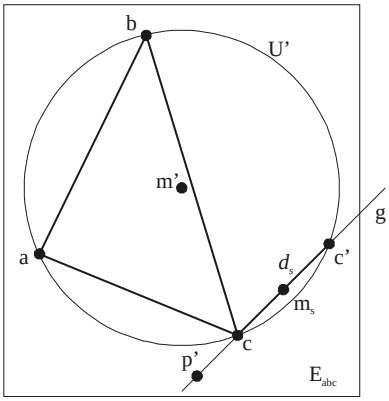


Abbildung 3.3.24: Beweis Satz 3.2.5 Punkt 2: Ansicht der Dreiecksebene E_{abc}.

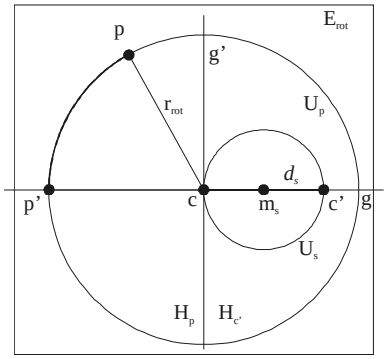


Abbildung 3.3.25: Beweis Satz 3.2.5 Punkt 2: Ansicht der Rotationsebene E_{rot}.

□

Satz 3.3.6: Viereckteilung

Gegeben seien paarweise verschiedene Punkte a, b, c und p im zweidimensionalen euklidischen Raum, die ein Viereck bilden. Seien α, β, γ und φ die Innenwinkel des Vierecks an den jeweiligen Punkten a, b, c und p. {a, b} und {c, p} seien ohne Beschränkung der Allgemeinheit jeweils unverbundene, also gegenüberliegende Punkte des Vierecks und die Summe der Innenwinkel α und β größer als die Summe von γ und φ. Dann gilt:

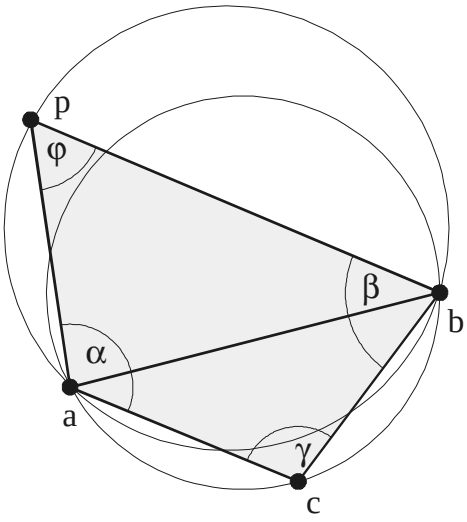


Abbildung 3.3.26: Satz 3.3.6: Teilung des Vierecks mit einer Kante durch die in der Summe größeren Innenwinkel.

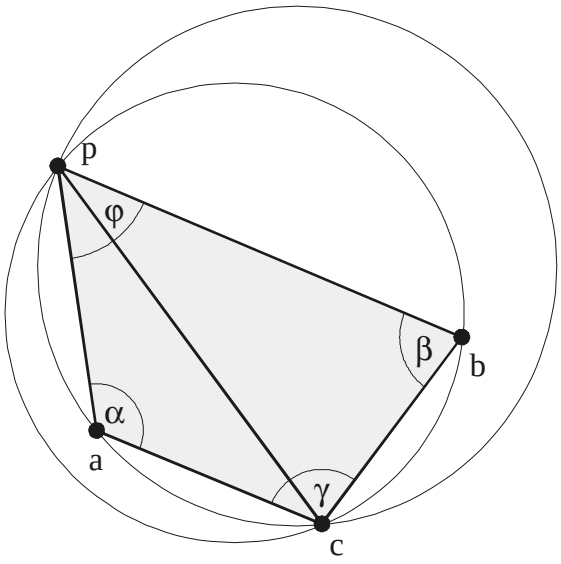


Abbildung 3.3.27: Satz 3.3.6: Teilung des Vierecks mit einer Kante durch die in der Summe kleineren Innenwinkel.

Wird das Viereck a c b p mit einer im Viereck liegenden Kante (Diagonale) durch zwei unverbundene (also gegenüberliegende) Punkte geteilt (a und b), deren Summe der Innenwinkel größer ist als die Summe der Innewinkel der anderen Punkte (c und p), so liegt p nicht innerhalb des Umkreises des Dreiecks $\triangle$a c b und c liegt nicht innerhalb des Umkreis des Dreiecks $\triangle$b p a (s. Abbildung 3.3.26). Spezialfall: Wenn $\alpha+\beta = \gamma+\varphi$, liegen a,b,c und p auf einem Umkreis.

Beweis:

Da a, b, c und d ein Viereck bilden ist $\alpha+\beta+\gamma+\varphi = 360°$

Spezialfall:

$\alpha+\beta = \gamma+\varphi$

$\Rightarrow \alpha+\beta + \alpha+\beta = 360°$

$\Rightarrow \alpha+\beta = 180°$

$\Rightarrow \alpha = 180° - \beta.$

Somit liegt nach Satz 3.3.3 der Punkt a auf dem Umkreis des Dreiecks c b p.

Allgemein:

Eine Gerade entlang einer inneliegenden Diagonalen in einem Viereck teilt den Raum in zwei Halbräume. Die Punkte, die nicht auf der Diagonalen liegen, liegen in verschiedenen Halbräumen (sonst würde die Diagonale außerhalb des Vierecks liegen).

$\alpha+\beta + \gamma+\varphi = 360°$

$\Rightarrow \alpha+\beta = 360° - (\gamma+\varphi)$

Da $\gamma+\varphi < \alpha+\beta$

$\Rightarrow \gamma+\varphi < 360° - (\gamma+\varphi)$

$\Rightarrow \gamma+\varphi < 180°$

$\Rightarrow \varphi < 180° - \gamma$

Somit liegt nach Satz 3.3.3 der Punkt nicht p im Umkreis des Dreiecks a b c und nach Satz 3.3.4 ist dann auch der Punkt c nicht im Umkreis des Dreiecks b p a.

Dieser Satz gilt auch für nicht konvexe Vierecke. In diesem Fall existiert ein Punkt (in Abbildung 3.3.28 der Punkt b) mit einem Innenwinkel $\beta \geq 180°$ (nur einer wegen der Innenwinkelsumme von 360°). Dessen Summe mit seinem gegenüberliegenden Innenwinkel ist so auf jeden Fall größer als die Summe der beiden anderen. Mit einer Kante durch c zu dessen gegenüberliegenden Punkt wird die einzig mögliche innenliegende Diagonale erzeugt und die obige Beweisführung ist auch auf diesen Fall anwendbar.

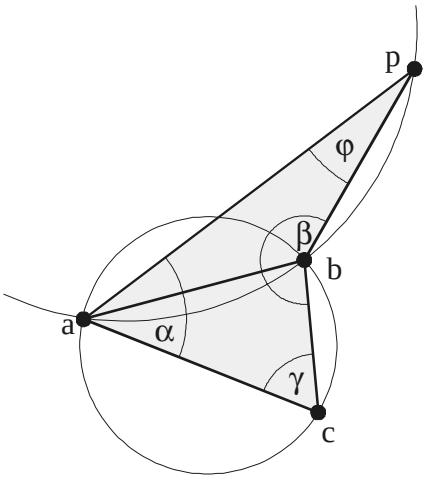


Abbildung 3.3.28: Beweis Satz 3.3.6: Der Satz gilt auch für nicht konvexe Vierecke

□

Bemerkung: Bei einer Teilung eines konvexen Vierecks mit einer Strecke durch die Punkte mit der kleineren Innenwinkelsumme {c p} würde a im Umkreis des Dreiecks △c b p und b im Umkreis des Dreiecks △p a c liegen. Der Beweis würde analog zu dem oben geführt werden. Ein nicht konvexes Viereck ließe sich nicht mit einer Kante durch die Punkte mit der kleineren Innenwinkelsumme teilen, da die Kante außerhalb des Vierecks läge.

3.4 Vorverarbeitung

Zur Ermittlung der potentiellen Nachbarn für eine Fächerbildung eines Punktes werden die räumlich nächsten Punkte gesucht und diese in die Tangentialebene des Punktes projiziert oder gedreht. Dies führt zu folgenden notwendigen Vorverarbeitungsschritten:

Sortierung in Zellraster

Eine oft gebrauchte Standardfunktion dieses Algorithmus ist die Bestimmung der nächsten Punkte, die Nachbarn n, zu einem gegebenen Punkt c. Bei der naiven Vorgehensweise müssten für jeden Punkt alle Punkte betrachtet werden, was zu einem quadratischen Aufwand (**O**(n^2), wenn n die Gesamtanzahl der Punkte darstellt) führen würde. Um also möglichst effizient die Nachbarpunkte zu ermitteln, haben sich bei der in Kapitel 2 vorgestellten Verfahren und in der Literatur (z.B.[Ottman/Widmayer '93] und [Schmitt '96]) zwei Arten der Vorsortierung als praktikabel erwiesen. Bei der ersten Sortierung werden die Punkte in einem sogenannten „Octtree“ angeordnet. Hierbei wird der Raum, der die Punktwolke enthält, rekursiv entlang der Mittelebenen in jeweils acht gleich große Unterräume geteilt, bis jeder Quader nicht mehr als eine fest angegebene Menge an Punkten enthält. Somit enthält ein Quader oder Knoten dieser Baumstruktur acht Verweise auf weitere Knoten oder Blätter, die eine feste Anzahl Punkte enthalten. Diese Sortierung wird hauptsächlich bei Echtzeitdarstellungen verwendet, da die Datenstruktur sich anbietet, zur Darstellung die Komplexität und Feinheiten der Objekte je nach Abstand zum Betrachter zu reduzieren.

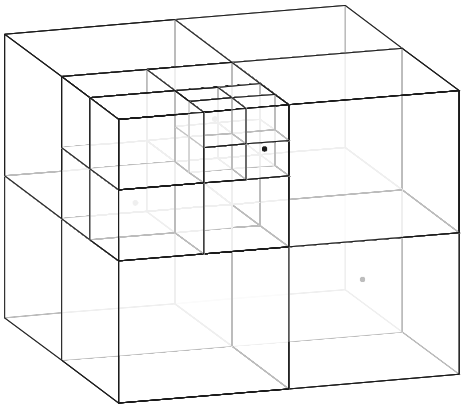


Abbildung 3.4.1: Octree: Raumaufteilung und Sortierung der Punkte in Quader mit begrenzter Anzahl von Punkten pro Quader..

Da in dem hier betrachteten Algorithmus die Anzahl der zur Fächerung eines Punktes benötigten Nachbarn nicht feststeht und nur bis zu einem maximalen Abstand gesucht werden sollte, wurde die Punktwolke in einem dreidimensionalen Zellraster angeordnet – dem zweiten Sortierverfahren. Hierbei wird der die Punktwolke enthaltene Raum in gleich große Quader oder Würfel geteilt, in denen die Punkte, ähnlich einem „Hash“-verfahren (näheres s. [Ottman/Widmayer '93]), einsortiert werden. Im ungünstigsten Fall

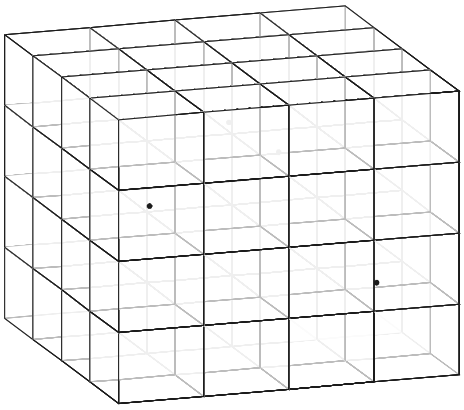


Abbildung 3.4.2: Zellteilung: Raumaufteilung und Sortierung der Punkte in gleich große Quader.

könnten hierbei der Großteil der Punkte in einen einzigen Quader eingeordnet werden. In der Praxis sind die Punkte im Raum einigermaßen gleich verteilt, so dass man bei einer von der Punktanzahl abhängigen Zellgröße eine obere Schranke der Punktanzahl *k* pro Quader erwarten kann. In [John '99] wird dazu eine Kantenlänge *l* der Quader, bzw. Würfel hergeleitet, die sich aus der Oberflächengröße des Umquaders (bounding-box) der Punktwolke ergibt und bei einer Gleichverteilung der Punkte ungefähr k Punkte pro Quader einsortiert werden:

$$l = \sqrt{2(d_x d_y + d_y d_z + d_x d_z)\frac{k}{n}}$$

Dabei sind d_x, d_y und d_z die Ausmaße der bounding-box und n die Anzahl der Punkte der Punktwolke.

Diese Schätzung für *l* wurde auch in der Implemetierung des hier vorgestellten Algorithmus verwendet.
Somit beträgt der Aufwand zur Vorsortierung einer Anzahl von n Punkten der Punktwolke **O**(n): Die Ausmaße der Punktwolke bestimmen **O**(n) und jeden Punkt einsortieren **O**(n). Die Suche aller Nachbarn zu einem Punkt mit maximalen Abstand *d* und einer Kantenlänge *l* der Zellwürfel kann nun mit einem Aufwand **O**($\lceil d/l \rceil^3 \cdot k$) erfolgen, welches bei günstiger Punktverteilung ($k << n$) und kleinem Suchradius *d* einen erheblichen Geschwindigkeitsgewinn bedeutet.

Normalenbestimmung

Um mit einer Tangentialebene in einem Punkt arbeiten zu können, muss die Normale an diesem Punkt bestimmt werden. Das Problem, die Normale eines Punktes c der Punktwolke exakt zu ermitteln, ist vielschichtig, aufwendig und könnte alleine eine ganze Arbeit füllen.

Die meisten der in Kapitel 2 vorgestellten Arbeiten beschränken sich darauf, eine Ausgleichsebene durch eine feste Anzahl Nachbarn von c zu legen (Abbildung 3.4.3). Diese liefert in den meisten Fällen eine genügend exakte Normale für den Punkt c. An Kanten des Objektes kann die Ausgleichebene jedoch senkrecht zur gewünschten Tangentialebene stehen (Abbildung 3.4.4). Um dem zu begegnen wird bei [Linsen '01]

die Normale am Verbindungsvektor der nächsten zwei Nachbarn um 90° gedreht, sollte sich kein vernünftiger Fächer mit der Normale aus der Ausgleichsebene ermitteln lassen.

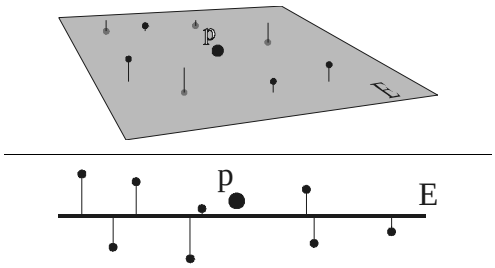


Abbildung 3.4.3: Ausgleichsebene E durch die Nachbarn des Punktes p. E liegt so, dass die Abstände der Nachbarn zu E minimiert sind.

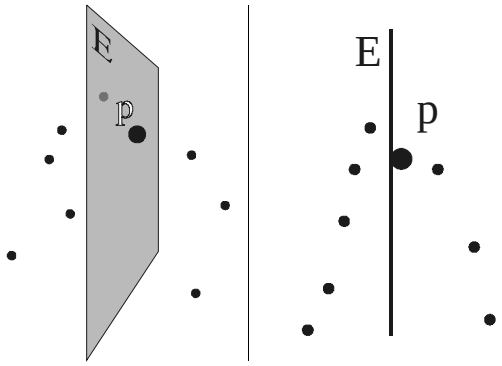


Abbildung 3.4.4: An Kanten des Objektes liefert die Ausgleichsebene nicht immer die gewünschte Normale

In der Praxis hat sich gezeigt, dass für den folgenden Algorithmus eine exakte Normale nicht unbedingt vonnöten ist, solange die Projektionen auf die Tangentialebenen konsistent bleiben (Näheres dazu in Kapitel 3.10). Daher konnte man sich auf eine Normalenbestimmung über die Ebene durch die drei nächsten Nachbarn beschränken und bei Nichtzustandekommen eines Fächers auf komplexere Verfahren (Ausgleichsebene durch mehrere Nachbarn und Drehung wie bei [Linsen '01] und [John '99], Kapitel 2.5) zurückgreifen. Der daraus resultierende Geschwindigkeitsgewinn bei der Vorverarbeitung ist linear und daher im **O**-Kalkül nicht ersichtlich, aber bei größeren Punktmengen deutlich spürbar (z.B. beim Hasen Seite 59 Abbildung 4.1.1: ca. 2,0 sec Berechnung der Normalen über eine Ausgleichsebene durch 6 Nachbarn, gegenüber ca. 0,5 sec bei einer Berechnung der Ebene durch drei Nachbarn).

Werden nach der Triangulierung exakte und orientierte Normalen (z.B. für eine Visualisierung über Schattierungen nach Phong oder Gouraud) gebraucht, können diese dann aus den zu jedem Punkt ermittelten Kanten berechnet und wie in [John '99], Kapitel 2.6 über einen minimal spannenden Baum durch diese Kanten orientiert werden.

3.5 Grundsätzliche Arbeitsweise des Algorithmus

Der Algorithmus erzeugt ein Dreiecksnetz, in dem zu jedem Punkt der Punktwolke

ein Fächer mit dessen Nachbarn gebildet wird. Hierbei wird der Fächer durch schrittweise Hinzunahme von Punkten zur Menge der Randpunkte erzeugt. Dabei werden vorerst schon bei jedem Kandidaten einige Fälle getestet, bei denen eine Hinzunahme zu einem der Problemfälle a-d aus Kapitel 3.1 führen würde. Sind genügend Kandidaten gefunden um einen Fächer zu erzeugen, bei dem die Fächerspeichen keinen Winkel größer als 90° bilden würden, wird der Fächer geschlossen (die im Fächer benachbarten Kandidaten untereinander und mit dem Fächermittelpunkt verbunden). Dabei wird nochmals getestet, ob diese Verbindungen problematische Konstellationen gemäß Kapitel 3.1 erzeugen würden und ob dadurch „schöne Dreiecke“ gemäß Kapitel 3.2 und 3.3 entstehen.

3.6 Überblick über die Schritte des Algorithmus

Folgende Schritte wurden also zur Erzeugung eines möglichst vollständigen Fächerbildung um einen Punkt c durchgeführt:

1. Es werden alle Nachbarn n in einer Umkugel mit Radius *d* bestimmt und nach Abstand zum Punkt c in einer Liste L_{dist} sortiert. In Abbildung 3.6.1 wäre $L_{dist} = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}$.

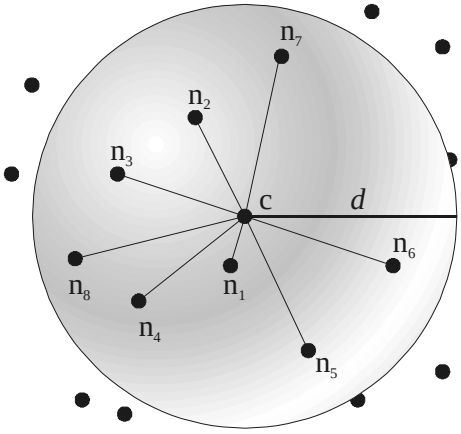


Abbildung 3.6.1: Algorithmusschritt 1

2. Der nächste Nachbar n_i aus der Liste L_{dist}, wird in die Tangentialebene von c projiziert (als n_i'') oder in die Tangentialebene gedreht (als n_i').

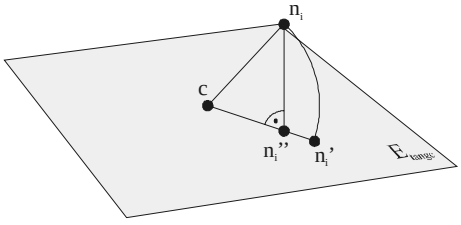


Abbildung 3.6.2: Algorithmusschritt 2

3. Es wird die Position , d.h der Index j [*)], bestimmt, den die Abbildung des Punktes n_i' in der nach Winkel (Polarkoordinate in der Tangentialebene) sortierten Menge L_{winkel} der bisherigen Fächerrandpunkte r von c einnehmen würde. Anmerkung: L_{winkel} ist eine zirkuläre Liste, was bedeutet, dass der Nachfolger des letzten Elementes das Erste und der Vorgänger des Ersten das letzte Element ist.

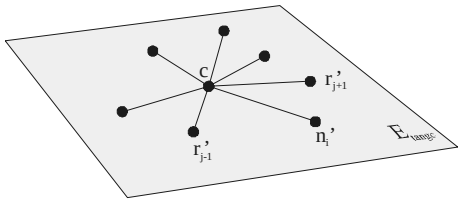


Abbildung 3.6.3: Algorithmusschritt 3

4. Bevor der Nachbarpunkt n_i bzw. dessen Projektion n_i' endgültig in die Liste der Fächerrandpunkte L_{winkel} an Position j eingefügt wird, wird geprüft ob er folgenden Kriterien genügt:

   a) n_i ist noch nicht in der Menge der bisherigen Fächerrandpunkte enthalten. (in z.B. Abbildung 3.6.4 war der Punkt n_i schon als Fächerrandpunkt r_i in der Fächerrandliste L_{winkel} enthalten)

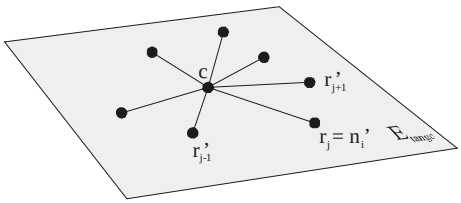


Abbildung 3.6.4: Algorithmusschritt 4a)

*) Damit die folgenden Bezeichnungen formal korrekt bleiben, sei ohne Beschränkung der Allgemeinheit 1 < j < Anzahl der Fächerandpunkte max. Bei j=1 wäre das Vorgängerelement eben r_{rmax} und bei $j=r_{max}$ das Nachfolgeelement r_1.

b) n_i' fällt nicht auf c (Abbildung 3.6.5) und der Winkelabstand zum vorherigen und nächsten Element der bisherigen Randpunkte (r_{j-1} und r_{j+1}) ist hinreichend groß (Abbildung 3.6.6).

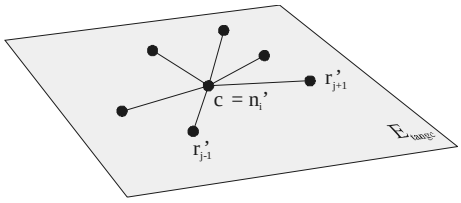


Abbildung 3.6.5: Algorithmusschritt 4b)

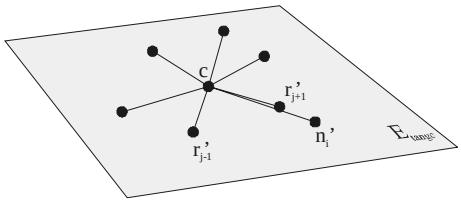


Abbildung 3.6.6: Algorithmusschritt 4b)

c) Der Punkt n_i' liegt im Umkreis des Dreiecks der projizierten/gedrehten Punkte c, r'_{j-1} und r'_{j+1}.

Sind alle Kriterien erfüllt, wird n_i' an der Position j in die Liste L_{winkel} einsortiert.

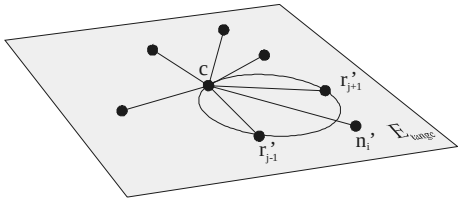


Abbildung 3.6.7: Algorithmusschritt 4c)

5. Es wird ein weiterer Nachbar n_{i+1} genommen und versucht, diesen ab Schritt 2 zur Randmenge hinzuzufügen.

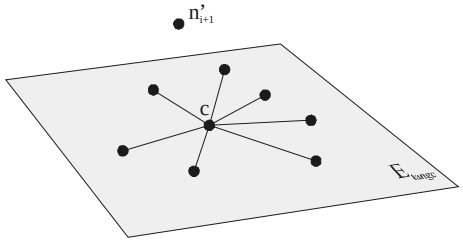


Abbildung 3.6.8: Algorithmusschritt 5

6. Sind keine weiteren Nachbarn in der Umkugel mit Radius *d* vorhanden, wird geprüft, ob im Fächer beim Punkt c keine Winkel größer als ein festes α auftreten. (z.B. Abbildung 3.6.9 $\gamma_i < \alpha = 90°$ für alle i) Ist dies der Fall, so wird der Fächer bei Schritt 7 geschlossen.

   Sonst wird der Suchradius *d* erweitert und neue Nachbarn ermittelt (Abbildung 3.6.10) und diese ab Schritt 1 bearbeitet oder bei zu großem $d > d_{max}$ abgebrochen und trotzdem versucht den Fächer ab Schritt 7 einigermaßen vollständig zu schließen.

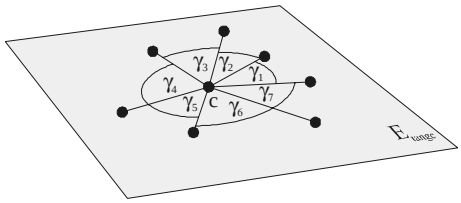


Abbildung 3.6.9: Algorithmusschritt 6

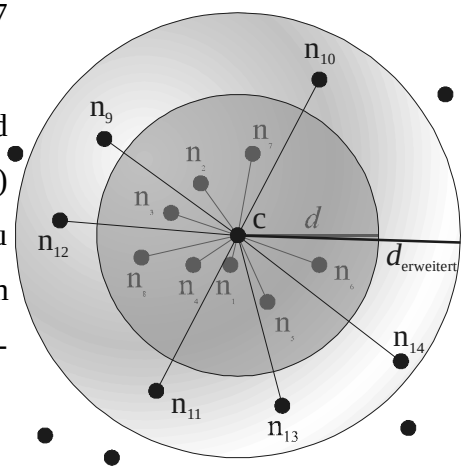


Abbildung 3.6.10: Algorithmusschritt 6

7. Der Fächer um diesen Punkt c ist fertig und kann geschlossen werden. Das bedeutet, das jedem Fächerrandpunkt sein Vorgänger und Nachfolger in der Liste L_{winkel}, sowie der Fächermittelpunkt c in seine eigene Fächerkandidatenliste einsortiert wird. Doch vorher müssen noch einige Bedingungen erfüllt sein:

   a) Zuerst wird für jeden Kandidaten der Liste L_{winkel} geprüft, ob eine Verbindung mit seinem Vorgänger in der Liste zu einem Problemfall aus Kapitel 3.1 führen könnte. Ist dies der Fall, wird der Punkt aus der Liste entfernt. In Abbildung 3.6.12 würde z.B.eine Verbindung von c mit n_i' Kanten des schon erzeugten Fächers um c' schneiden.

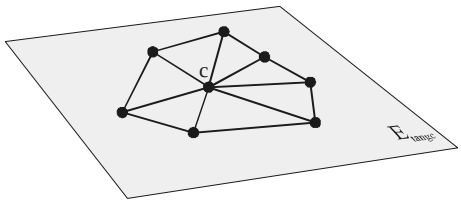


Abbildung 3.6.11: Algorithmusschritt 7

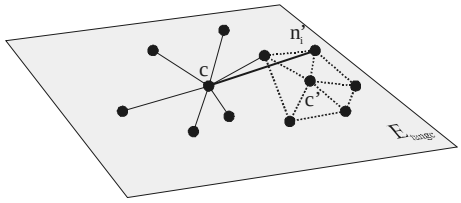


Abbildung 3.6.12: Algorithmusschritt 7a)

b) Der Rand wird nun geschlossen, wobei jedem Randpunkt r_j sein Vorgänger und Nachfolger als Randpunkt seines eigenen Fächers zugewiesen wird.

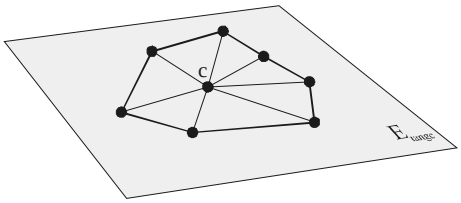


Abbildung 3.6.13: Algorithmusschritt 7b)

c) Die Randpunkte werden mit dem Fächermittelpunkt c verbunden. Dabei wird getestet, ob die Verbindung c r_j „schönere“ Dreiecke erzeugt, als eine Verbindung r_{j-1} mit r_{j+1} ergeben würde (siehe Fußnote zu Punkt 3).

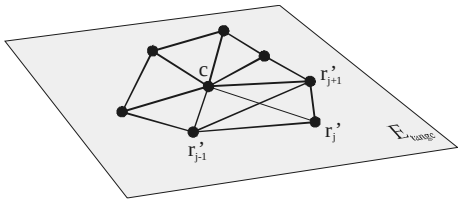


Abbildung 3.6.14: Algorithmusschritt 7c)

8. Nun kann bei Schritt 1 mit einem neuen noch nicht mit einem Fächer umgebenen Punkt der Punktwolke fortgefahren werden. Wie z.B. mit Punkt c_3 in Abbildung 3.6.15.

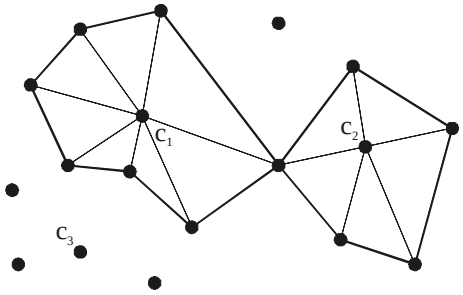


Abbildung 3.6.15: Algorithmusschritt 8

9. Haben alle Punkte ihren Fächer erhalten, werden von jedem Punkt die Fächerdreiecke in eine Dreiecksliste L_{Triang}, die Triangulation, eingetragen.

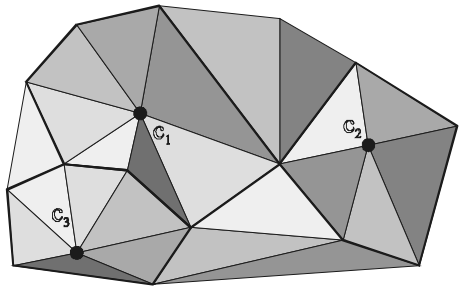


Abbildung 3.6.16: Algorithmusschritt 9

3.7 Details der einzelnen Schritte

Zu Schritt 1 und 2

Die schrittweise Hinzunahme des jeweils nächsten Nachbarn verhindert, dass Punkte im Innern eines schon bestehenden Fächersegmentes zu liegen kommen und vor allem

nähere Punkte bevorzugt hinzugenommen werden. In Abbildung 3.7.1 wurde der nächste Punkt n_1 nicht zuerst zum Fächer hinzugenommen.

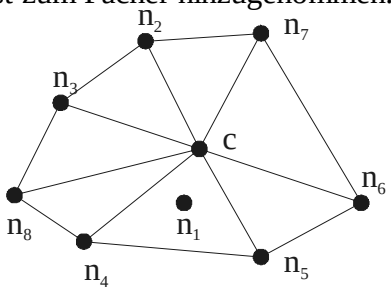


Abbildung 3.7.1 zu Schritt1: Hinzunahme des jeweils nächsten Nachbarpunktes verhindert hier gezeigte Einschlüsse.

Der anfängliche maximale Abstand *d* der Nachbarn sollte so gewählt werden, dass schon vorhandene Randpunkte (aus Schritt 4e und Schritt 6) sich innerhalb dieses Radius befinden, da sonst auch Punkte innerhalb der Fächerdreiecke übersehen werden könnten. Die Abbildung 3.7.2 zeigt, wie bei zu kleinem anfänglichen Suchradius d_{Start} kein weiterer Punkt für einen Fächer um c_3 gefunden werden würde. Die Kanten, die vom Fächer um c_1 und c_2 her schon bestehen, würden für einen kompletten Fächer um c_3 ausreichen. Bei dessen Schließung würde der Punkt p dann übersehen werden.

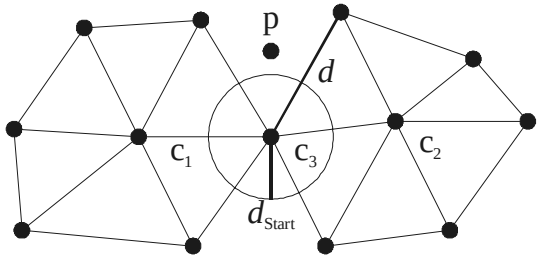


Abbildung 3.7.2 zu Schritt 1: Bei zu kleinem Suchradius könnten nötige Punkte übersehen werden.

Zu Schritt 2

Die orthogonale Projektion, bzw. Drehung ist eine Abbildung in ein zweidimensionales euklidisches Koordinatensystem in der Tangentialebene (Abbildung 3.7.3). Nullpunkt ist der Punkt c und als Basis dienen zwei orthogonale normierte Vektoren b_1 und b_2, die senkrecht zur Normale stehen. So bleiben Winkel- und Längengrößen im zwei- und dreidimensionalen Raum vergleichbar. Die Koordinaten **p**" der orthogonalen Projektion p" erhält man, indem der Verbindungsvektor von c nach p (**p)** mit

einer 2x3 Matrix multipliziert wird, deren Spaltenvektoren aus b_1 und b_2 bestehen:

$$\mathbf{p}'' = \begin{pmatrix} b_{1x} & b_{1y} & b_{1z} \\ b_{2x} & b_{2y} & b_{2z} \end{pmatrix} \mathbf{p}$$

Die Koordinaten **p**' der orthogonale Drehung p' erhält man, indem **p**" auf die ursprüngliche Länge von **p** skaliert wird.

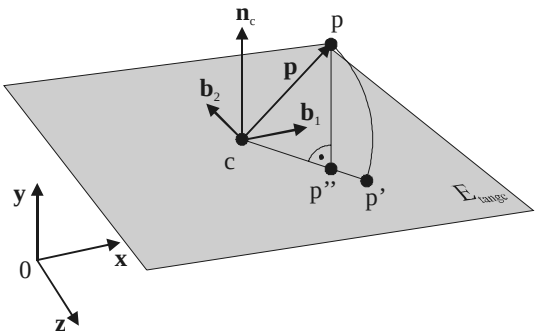


Abbildung 3.7.3 zu Schritt 2: Drehung eines Punktes P in die Tangentialebene des Punktes c als p'. Die orthogonale Projektion des Punktes als p''.

Ob zur weiteren Verarbeitung die Nachbarn in die Tangentialebene projiziert oder gedreht werden, gilt es abzuwägen. Bei den Arbeiten von Lars Linsen [Linsen '01] werden die Nachbarn in die Tangentialebene vom Punkt c orthogonal projiziert, bei [Gopi et al. '00] werden sie in die Tangentialebene gedreht. Während eine orthogonale Projektion etwas schneller ist (die Skalierung fällt weg), verhindert aber eine distanztreue Drehung der Punkte die Verbindung mit Punkten von der Rückseite des Objektes und ermöglicht eine korrekte Anwendung des Umkreiskriteriums (Näheres bei Schritt 4c). In Abbildung 3.7.4 würde die Projektion p" des Punktes p von der Rückseite einer starken Krümmung in den Umkreis projiziert werden. Die Drehung p' würde dagegen nicht zu einem Fächer um c hinzugenommen werden.

Zu Schritt 3

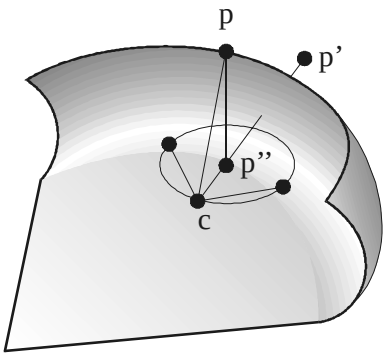


Abbildung 3.7.4 zu Schritt 2: Bei Projektion der Punkte kann das Umkreiskriterium nicht erwünschte Ergebnisse liefern.

Das Koordinatensystem der Tangentialebene ist auch die Referenz, um den Verbindungsvektoren vom Punkt c zu den projizierten Nachbarn n_i' einen eindeutigen Winkel zuzuordnen und so eine Sortierung zu ermöglichen (Abbildung 3.7.5).

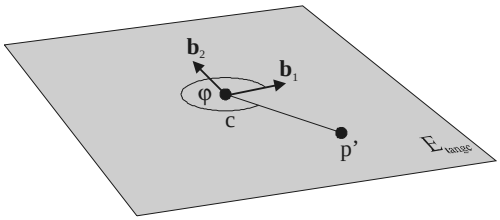


Abbildung 3.7.5 zu Schritt 3: der Winkel j entspricht den zweidimensionalen Polarkoordinaten von p'.

Zu Schritt 4a)

Dieser triviale Schritt verhindert die Mehrfacherzeugung der gleichen Kanten und Dreiecke (Abbildung 3.7.6).

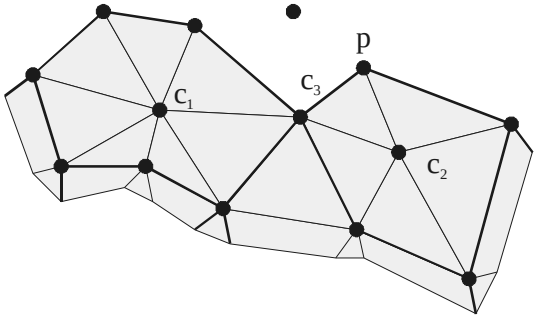


Abbildung 3.7.6 zu Schritt 4a): Der Punkt p wurde bei der Erzeugung des Fächers um c_2 schon in die Randpunktliste von c_3 eingetragen, so dass er nicht nochmals hinzugenommen werden darf.

Zu Schritt 4b)

Diese Abfrage verhindert eine falsche Winkelsortierung bei sehr kleinen numerischen Ungenauigkeiten, wie sie meist bei generisch erzeugten Punktwolken auftreten. In Abbildung 3.7.7 wäre nicht gesichert, ob der Punkt p' vor oder nach dem Punkt r_j' einsortiert wird. In diesem Fall sollte er nicht zum Fächer hinzugenommen werden. Liegt der Punkt allerdings im vom Fächermittelpunkt c und den Vorgänger und Nachfolger aufgespannten Dreieck, muss er trotzdem unbedingt zum Fächer hinzugenommen werden, damit später ermittelte Verbindungen von diesem Punkt aus Verbindungen des aktuellen Fächers nicht überlappen.

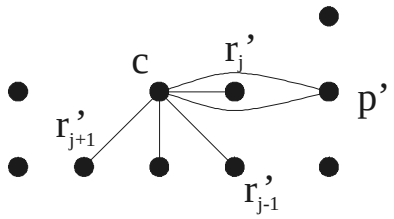


Abbildung 3.7.7 zu Schritt 4b): Ob p' nun vor oder hinter r_j' in die Winkelliste einsortiert wird ist hier von der Rundungsgenauigkeit abhängig.

Sollte die Projektion zufälligerweise in den Nullpunkt der Tangentialebene fallen, ist eine eindeutige Winkelbestimmung und daher Einsortierung nicht möglich (Abbildung 3.7.8). Dies tritt auf, wenn die Nachbarn von der Rückseite des Objektes in die Tangentialebene projiziert werden.

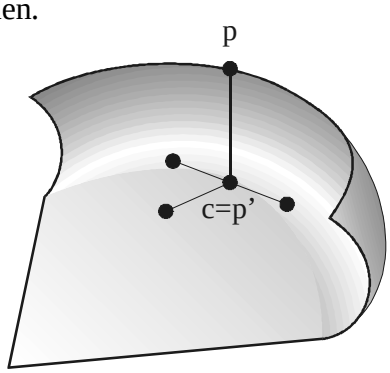


Abbildung 3.7.8 zu Schritt4b): Projektion des Punktes p auf c.

Zu Schritt 4c)

Dieser Schritt bewirkt, dass möglichst gleichschenklige Dreiecke zu diesem Fächer erzeugt werden. Zusammen mit Schritt 7c), der die Dreiecke mit benachbarten

Fächern vergleicht, wird damit eine Triangulation erzeugt, die den Delaunay-Kriterien genügt. Er entspricht etwa dem in [Gopi et al. '00] verwendeten Verfahren zur Bestimmung der Delaunay-Nachbarn. Wohingegen M. Gopi ein Voronoidiagramm der vier Punkte c (der Fächermittelpunkt), r_{j-1}', r_{j+1}' und dem zu prüfenden Punkt n_i' erzeugt, werden hier nur die Winkel $\sphericalangle r_{j-1}$' c r_{j+1}' und $\sphericalangle r_{j+1}$' n_i' r_{j-1}' verglichen und gemäß Satz 3.3.3 aus Kapitel 3.3 ausgewertet (Abbildung 3.7.9).

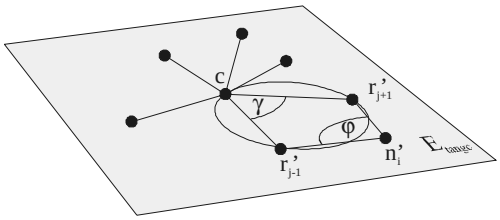


Abbildung 3.7.9 zu Schritt4c): Test des Unkreiskriteriums über Winkelvergleich nach Satz 3.3.3 aus Kapitel 3.3.

Da meistens nur Winkel untereinander, mit 180° oder der Differenz zu 180° verglichen werden, genügt es meistens, nur den Kosinus des Winkels zu kennen, der sich leicht aus dem Produkt der Vektoren ermitteln läßt, die den Winkel einschließen. Diese Vorauswahl in diesem Schritt verhindert die Erzeugung unnötig langer Dreiecke wenn in anderen Sektoren des Fächers noch Punkte fehlen, und verringert die im aufwendigen Schritt 7 zu betrachtenden Punkte.

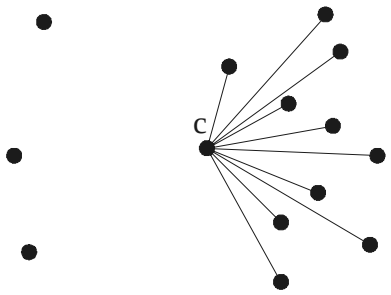


Abbildung 3.7.10 zu Schritt4c): Da der Fächer links von c noch unvollständig ist, werden weiterhin Nachbarn gesucht und hinzugenommen. Schritt 4c) verhindert eine hier dargestellte Ansammlung von Verbindungen auf der rechten Seite, die zu vielen langen spitzen Dreiecken führen würde.

Zu Schritt 6

In diesem Schritt wird geprüft, ob der bisher erzeugte Fächer den Punkt vollständig umschließt. Wie schon in [Linsen '01] beschrieben, wird damit sichergestellt, dass die

Objektfläche in der Umgebung des aktuellen Punktes herum komplett durch Dreiecke dargestellt wird. Grundsätzlich würde es hierfür reichen, zu testen, ob die Fächerspeichen Winkel < 180° einschließen (Abbildung 3.7.11).

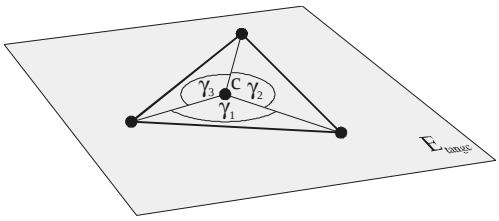


Abbildung 3.7.11 zu Schritt 6: vollständige Umschliessung durch einen Fächer, wenn alle Winkel bei c < 180° sind.

Um einer Delaunay-Triangulierung näher zu kommen und um eine korrekte Triangulation gewährleisten zu können (siehe Kapitel 3.9 Seite 56), hat sich ein Grenzwert von 90° bewährt. Des weiteren sollte der Suchradius d_{max} beschränkt werden. Einerseits, damit nicht Verbindungen zu konkaven Elementen des Objektes hergestellt werden (z.B. zwischen den Ohren des Hasen; s. Abbildung 4.1.3 Seite 59). Andererseits haben praktische Erfahrungen ergeben, dass sich unvollständige Fächer kaum noch mit sehr weit entfernten Punkten korrekt schließen lassen, und so auch aus Geschwindigkeitsgründen die Suchtiefe beschränkt sein sollte. Um nun ungewollte Löcher zu vermeiden, aber doch die Suchtiefe gering zu halten, hat sich in der Praxis ein maximaler Suchradius d_{max} vom 4 bis 5-fachen der Kantenlänge einer Zelle des Zellraster bewährt.

Zu Schritt 7a)

In diesen aufwendigen Schritt werden die Verbindungen, die den Rand des Fächers beschreiben, auf Schnitte mit bisherigen Kanten anderer Fächer getestet. Das Testverfahren ähnelt dem in [Schmitt '96] Kapitel 2.2.1 beschriebenen Bentley-Ottmann Algorithmus und in Kapitel 4.2.2 beschriebenen Algorithmus des Sweep-Verfahrens in Radarstrahltechnik.

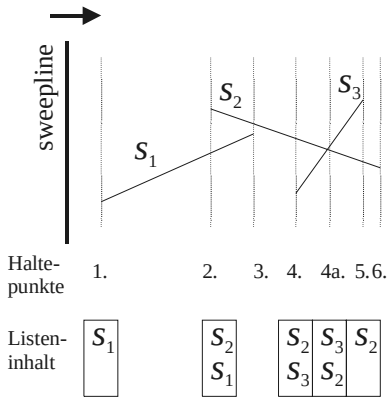


Abbildung 3.7.12 zu Schritt 7a): Bentley-Ottmann Algorithmus.

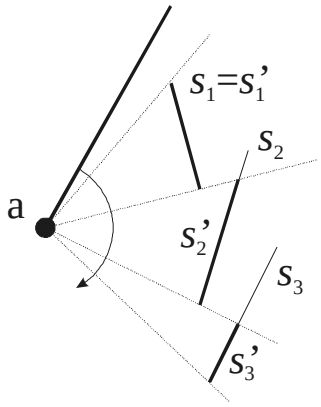


Abbildung 3.7.13 zu Schritt 7a): Radar-strahlverfahren.

Ersterer verwendet eine Sweep-Line zur Bestimmung von Schnittpunkten beliebiger Strecken im zweidimensionalen Raum. Hierzu wird zu jedem Haltepunkt eine nach Höhe sortierte Liste der zu untersuchenden Strecken erzeugt. Eine in der sweep-Richtung neu hinzukommende Liste wird nur mit Schnittpunkten mit ihrem Vorgänger und Nachfolger in dieser Liste geprüft und die Schnittpunkte als neue Haltepunkte der Sweepline eingefügt. In der Abbildung 3.7.12 werden z.B. am Haltepunkt 2. die Strecken s_1 und s_2 auf Schnittpunkte geprüft, am Haltepunkt 4. die Strecken s_2 und s_3. Bei deren Schnittpunkt wird ein neuer Haltepunkt 4a eingefügt, bei dem sich wie bei den restlichen Haltepunkten die Höhensortierung der Strecken ändert.

Das Radarstrahlverfahren ist eine Verwendung des Bentley-Ottmann Algorithmus mit radial laufender Sweep-Line und dient zur Sichtbarkeitsbestimmung von einem festen Augenpunkt aus. Bei dem in Abbildung 3.7.13 gezeigten Beispiel werden so die von dem Punkt a aus sichtbaren Teilstrecken s_1', s_2' und s_3' bestimmt.

Das nun verwendete Testverfahren prüft, ob jeder bisher ermittelte Fächerrandpunkt näher an dem Fächermittelpunkt c liegt, als bisherige gesetzte Kanten anderer Fächer.

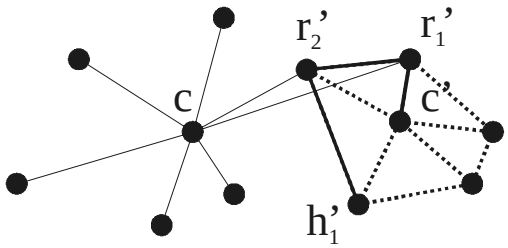


Abbildung 3.7.14 zu Schritt 7a): potentielle Fächerrandpunkte r_1 und r_2 mit schon bestehenden Kanten.

Die einzelnen Schritte des Testverfahrens:

Schritt I : Zu jedem bisherigen Fächerrandpunkt r_j werden die Vorgänger und Nachfolger des Punktes c in dessen Fächerrandpunktliste ermittelt. Dieser Vorgänger und Nachfolger werden als Hilfspunkte in die Fächerrandpunktliste L_{winkel} des Punktes c eingetragen. Dem Hilfspunkt *Vorgänger* wird noch eingetragen, dass bei ihm eine Kante beginnt, dem Hilfspunkt *Nachfolger*, dass bei ihm eine Kante endet und bei r_j beginnt und endet eine Kante. In der Abbildung 3.7.14 würde c im Fächer von $r_{1'}$ zwischen r_2' und c' eingeordnet werden. Da r_2 in der Fächerrandliste L_{winkel} schon enthalten ist, wird nur der Punkt c' als Hilfspunkt eingetragen. Ebenso wird vom Punkt r_2' nur h_1' als Hilfspunkt übernommen.

Schritt II: Es wird eine Liste L_{Kanten} erzeugt, die an der aktuellen Sweep-Strahlposition die Kanten sortiert nach Abstand zum Mittelpunkt c enthält. Da die bisher erzeugten Kanten sich nicht schneiden, ändert sich diese Sortierung zwischen den Elementen der Liste aus Fächerrand- und Hilfspunkten nicht.

Schritt III: Der Sweep-Strahl läuft nun radial die Punkte in der Liste L_{winkel} ab. An jedem Element der Liste wird geprüft:

- Startet hier eine Kante, so wird sie in die Kantenliste L_{Kanten} einsortiert.
- Endet hier eine Kante, so wird sie aus der Kantenliste L_{Kanten} gelöscht.
- Ist das Element ein Fächerrandpunkt (kein Hilfspunkt), so wird geprüft, ob dieser Punkt und der Fächermittelpunkt c auf der selben Seite der vordersten Kante der Kantenliste L_{kanten} liegen. Ist dies nicht der Fall, dann würde eine Verbindung von c zu diesem Punkt mindestens diese Kante schneiden und so wird der Punkt aus der Fächerrandpunktliste L_{Winkel} gelöscht.

In der Abbildung 3.7.14 würde L_{Kanten} die Kante von h_1' nach r_2' enthalten, wenn der radiale Sweepstrahl gerade h_1' passiert hätte. Bei c' würde dann die Kante von c' nach r_1' hinzukommen, die nach der von h_1' nach r_2' ein-

sortiert werden würde. Der Punkt c' würde dann auch aus der Liste der Fächerrandpunkte gelöscht werden, da er von c aus sich hinter ersten Kante in der Kantenliste (h_1' nach r_2') befindet.

Der Sweep-Strahl braucht allerdings einen Vorlauf von 180°, um alle Kanten zu enthalten, die von einer Kante von c zum ersten getesteten Fächerrandpunkt geschnitten werden könnten.

Schritt IV: Nachdem der Sweepstrahl ein Gebiet von 540° überstrichen hat, werden alle Hilfspunkte gelöscht und die verbliebenen Punkte können nun in Schritt 7b) verbunden werden.

Zu Schritt 7c)

Um zu einer Delaunay-Triangulierung um den Punkt c zu kommen, werden nacheinander alle benachbarten Dreiecke des Fächers getestet. Das von zwei benachbarten Dreiecken aufgespannte Viereck wird nach Satz 3.3.6 aus Kapitel 3.3 entsprechend neu in Dreiecke aufgeteilt. Die daraus entstandenen Dreiecke werden weiter mit dem nächsten benachbarten Dreieck getestet, bis alle Speichen des Fächers getestet wurden.

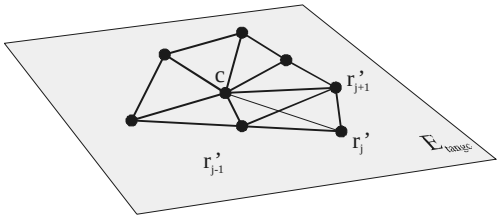


Abbildung 3.7.15 zu Schritt 7c): Die Dreiecke $\triangle c\, r_{j-1}'\, r_{j+1}'$ und $\triangle r_j'\, r_{j+1}'\, r_{j-1}'$ erfüllen im Gegensatz zu den Dreiecken $\triangle c\, r_{j-1}'\, r_j'$ und $\triangle c\, r_j'\, r_{j+1}'$das Umkreiskriterium.

Anfänglich wurden in einem einfachen und sehr schnellen Test die Längen der zwei möglichen Diagonalen verglichen und die kürzere beibehalten. Dies führte jedoch bei manchen Konstellationen zu unbefriedigenden Erebnissen.

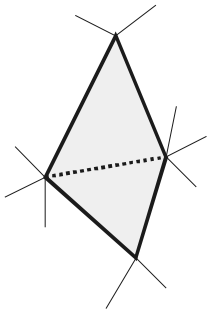


Abbildung 3.7.16: zu Schritt 7c): Das Viereck entlang der kürzeren Diagonale teilen...

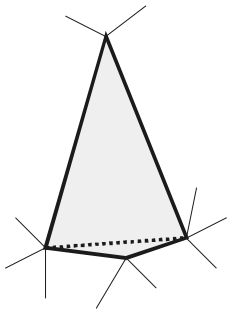


Abbildung 3.7.17: ... führt nicht immer zu befriedigenden Ergebnissen.

Daraufhin wurden nach Satz 3.3.6 aus Kapitel 3.3 die Innenwinkel im Viereck verglichen. Nach Satz 3.3.5 aus Kapitel 3.3 müssten für diesen Test der Eckpunkt r_{j-1} des einen Dreiecks ($\triangle c\ r_{j-1}'\ r_j'$) in die Ebene des anderen Dreiecks ($\triangle c\ r_j'\ r_{j+1}'$) gedreht werden, um eine korrekte Aussage darüber zu erhalten, ob der Punkt p in der Umkugel des Dreiecks $\triangle c\ r_j'\ r_{j+1}'$ enthalten ist. Da aber dieser Algorithmus von einer ungefähren planaren Umgebung um die Punkte ausgeht, genügt es in der Praxis, mit den in die Tangentialebene des Fächermittelpunktes gedrehten Punkten zu arbeiten und das Problem planar nur nach Satz 3.3.6 zu lösen.

Dieser Schritt verändert auch Dreiecke schon vorhandener Fächer, jedoch wird dabei nur die Diagonale eines Vierecks umgeklappt, was nichts an der bisher abgedeckten Fläche des Objektes ändert.

3.8 Anmerkungen zu vorherigen Versionen des Algorithmus

Bei einem Vorläufer des oben skizzierten Algorithmus wurde schon bei der Hinzunahme der Kandidaten bei Schritt 4 versucht, sämtliche Problemfälle herauszufiltern. Er wurde um folgende Kriterien erweitert:

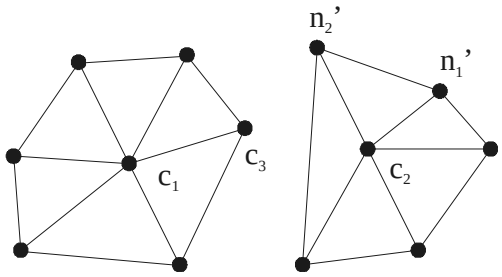


Abbildung 3.8.1: Vom Vorgänger abgefangene Konstellationen:

- Der Punkt n_i wurde noch nicht bearbeitet, d.h. von einem Fächer umgeben. Dies vermeidet Verbindungen in schon bestehende Fächer hinein. In der AbbildungAbbildung 3.8.1 würde dadurch eine Verbindung von c_3 mit c_2 verhindert.
- Der vorherige oder nächste Randpunkt (r_{j-1} und r_{j+1}) wurde noch nicht von einem Fächer umgeben. Auch dieses Kriterium verhindert Verbindungen innerhalb schon bestehender Fächer. In der AbbildungAbbildung 3.8.1 wäre eine Verbindung von n_1' zu c_3 nicht gestattet, da c_3 zwischen c_2 und n_2' in der Fächerrandliste von n_1' eingeordnet werden würde und c_2 schon umgeben ist.
- Der Punkt n_i nimmt den Punkt c nach allen Kriterien des Schrittes 4 in die Menge seiner Randpunkte auf. Dieser Test verhindert ungewollte Verbindungen bei startken Krümmungen, wenn sich die Anordnungen in benachbarten Projektionen unterscheiden.

Weiterhin wurden bei Schritt 7 ohne Prüfung die Fächerrandpunkte mit ihren Vorgänger, Nachfolger und dem Punkt c verbunden und der Fächer danach in die Dreiecksliste L_{triang} übertragen. Dies hatte den Vorteil, das die Fächerrandlisten fertig „gefächerter“ Punkte gelöscht und somit Arbeitsspeicher gespart werden konnte.

Obwohl diese Variante etwas schneller (z.B. Faktor 0.75 bis 0.85 bei der Punktwolke des Hasen) und bei den meisten gutartigen Punktverteilungen ansprechende Ergebnisse lieferte, wurden folgende seltene, aber vorkommende Konstellationen nicht abgefangen:

- Querverbindungen über schon bestehende Fächer hinweg wurden nicht erkannt (Abbildung 3.8.2).

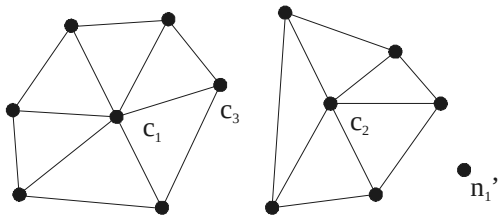


Abbildung 3.8.2: Die Verbindung von c_1 nach n_1' würde zustandekommen

- Spitze und lange Dreiecke an Rändern schon berechneter Dreiecke wurden nicht umgeformt (Abbildung 3.8.3).

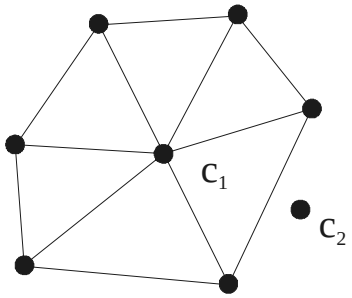


Abbildung 3.8.3: c_2 würde am Rand des schon bestehende Fächers um c_1 ein spitzes Dreieck bilden.

Weiterhin wirkte sich ein Fehler in der Verbindungsbestimmung auf die Fächerbildung der angrenzenden Punkte aus, da diese mit fehlerhaften Beziehungen weiterfächerten, ohne diese mit dem „Wissen“ aus ihren Nachbarschaften korrigieren zu können (wie bei Schritt 7 c))

3.9 Beurteilung des Algorithmus

Dieser Algorithmus ist nun in der Lage, die in Kapitel 3.1 beschriebenen vier Problemfälle, bzw. deren planare Entsprechungen, zu erkennen und zu vermeiden.

Da zu jedem Punkt versucht wird einen vollständigen Fächer zu bilden, wird kein Punkt am Rande oder außerhalb des Dreiecksnetzes zu liegen kommen. Ungewollte Löcher entstehen so nur, wenn die Punktdichte derart gering oder unregelmäßig ist, dass in dem maximalen Suchradius d_{max} nicht genügend Nachbarn gefunden werden können oder mit der Rückseite des Objektes verbunden werden. Damit ist sichergestellt, dass eine vollständige Triangulation erzeugt wird. Es kann jedoch passieren das kleine Löcher, die zum Objekt gehören, irrtümlicherweise geschlossen werden.

Wenn die Umgebung jedes Punktes p der Punktwolke ungefähr planar ist, läßt sich, wie in Kapitel 3.1 beschrieben, eine korrekte Triangulation sicherstellen, wenn bei der Triangulierung der Abbildung der Nachbarn mit maximalen Abstand *d* keine Schnitte der Kanten auftreten. Da die lokale Planarität einer Umgebung mathematisch schwer zu erfassen ist ([Gopi et al. '00] stellt einen guten Ansatz in dieser Richtung dar), wird die Behauptung nur für eine korrekte Triangulation einer ebenen Punktwolke bewiesen. Für allgemeine Betrachtungen im dreidimensionalen Fall wäre Satz 3.3.5 aus Kapitel 3.3 ein Anfang, der aber aus Zeitgründen in dieser Arbeit nicht weiter

verfolgt und ausgeführt werden konnte.

Zumindest im zweidimensionalen Fall ist dies bei dem hier beschriebenen Algorithmus gesichert, da für jeden Fächer sichergestellt wird, dass bei seiner Bildung keine abgebildeten Kanten in dessen Fächerradius geschnitten werden oder in ihm enthalten sind.

Eine Art konstruktiver Beweis ergibt sich aus den einzelnen Arbeitsschritten des Algorithmus, der die Konstellationen, die zu Schnitt oder Enthaltungen führen, abfängt.

Schritt 1 und 4 sorgen dafür, dass sich keine Punkte innerhalb schon gebildeter Fächer befinden, die bei späterer „Umfächerung“ zu Enthaltungen oder Schnitten führen würde (s. Abbildung 3.7.1 Seite 44).

Im Schritt 7 des Algorithmus wird sichergestellt, dass Verbindungen, deren Endpunkte sich im Fächerradius befinden, nicht geschnitten werden.

Kanten, deren Endpunkte nicht im Fächerradius enthalten sind, aber diesen schneiden (die Kante von n_1' n_2'), treten nicht auf, da diese nur bei Fächerspeichenwinkel bei p > 90° auftreten*), welches im Schritt 6 bis zu einem Suchradius d_{max} verhindert wird.

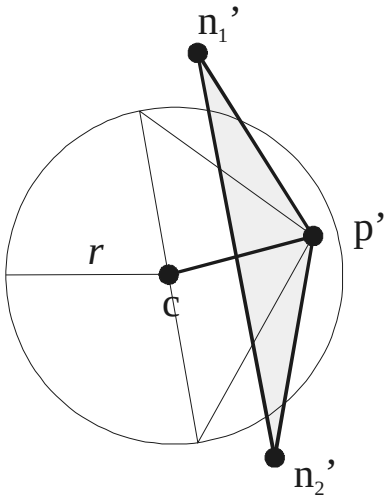


Abbildung 3.9.1: Kanten innerhalb des Fächerradius würden durch eine Kante c p geschnitten.

Im Vergleich mit den in Kapitel 2 vorgestellten ähnlichen Arbeiten zeigt sich, dass der Algorithmus als eine Weiterentwicklung des Ansatzes von [Gopi et al. '00] betrachtet werden kann. Hierbei sei die Nutzung der schon errechneten Winkel zur Evaluierung des Umkreiskriteriums erwähnt. Des weiteren steht hier die Schließung der Fächer im Vordergrund, die eine Delaunay-Triangulierung als „kosmetischen“ Arbeitsschritt verwendet, während sie bei M. Gopis Ansatz die Grundidee darstellt. Dies führt bei ihm zu Löchern im Dreiecksnetz, die nachträglich geschlossen werden müssen.

Der Ansatz von [Crossno/Angel '99] besitzt Ähnlichkeit mit dem in Kapitel 3.5 kurz

*) der Fächerwinkel muß größer als 90° sein, da es eine zu der Kante n_1' n_2' parallele Sehne durch c gibt. Die Verbindungen der Endpunkte dieser Sehne mit p ergibt ein Dreieck mit Innenwinkel größer als 90° (laut Satz 3.3.2), der aber kleiner ist, als der Innenwinkel bei p im Dreieck n_1' n_2 p. (Sonst würde c im diesem dreieck liegen, was von Schritt 1 und 4 vorgebeugt wird.)

erwähnten Vorgänger zu dem erarbeiteten Algorithmus. Dabei hat er aber auch dessen Schwächen, was sich z.B. darin zeigt, dass nicht sämtliche Überlappungen abgefangen werden.

Im Gegensatz zu diesen beiden Algorithmen verzichtet dieser hier vorgestellte Ansatz auf eine anschließende Nachbearbeitung und versucht Löcher und Überschneidungen schon bei der Erzeugung zu vermeiden.

3.10 Anforderung an die Punktwolke

Für den in Kapitel 3.5 und 3.9 vorgestellten Algorithmus ergeben sich Anforderungen an die Punktwolke, um zu einer zufriendestellenden Triangulation zu kommen. Damit sich das Triangulationsproblem lokal in einem zweidimensionalen Raum übertragen und dort lösen läßt, muss sich die Umgebung eines Punktes p eindeutig in diesen übertragen lassen. Das in [Gopi et al. '00] entwickelte Kriterium stellt eine Möglichkeit dar, wird aber selten in der Praxis erfüllt und ist für diesen Algorithmus auch nicht unbedingt nötig. Um eine korrekte Triangulation zu garantieren, sollte die Punktwolke also folgendem Kriterium genügend, dass sich aus dem Algorithmus ableitet:

Kriterium: Für jeden Punkt p der Punktwolke und dessen Nachbarn mit maximalen Abstand *d* zu p sollte gelten:

Zu einem beliebigen Punkt p werden die Nachbarn in dessen Tangentialebene abgebildet und miteinander verbunden. Bei einer Abbildung der Punkte und deren Verbindungen in die Tangentialebene eines anderen Punktes p' aus der Menge der Nachbarn sollten alle betrachteten Punkte (p,p' und die restlichen Nachbarn) auf der gleichen Seite der Verbindungen abgebildet werden.

Dies ist einerseits ein Anspruch an die Güte der berechneten oder vorgegebenen Normalen, sowie an eine der Objektkrümmung angepasste Punktdichte. Einfach gesagt kann hierbei die Normale ungenauer sein, je dichter die Objektfläche abgetastet wurde oder muss genauer berechnet werden, wenn die Punktdichte gröber ist.

4 praktische Erfahrungen und Ausblick

4.1 Praktische Ergebnisse

Zur praktischen Analyse des Algorithmus wurden generische Punktwolken, sowie Daten von realen Objekten verwendet.

reale Objekte:

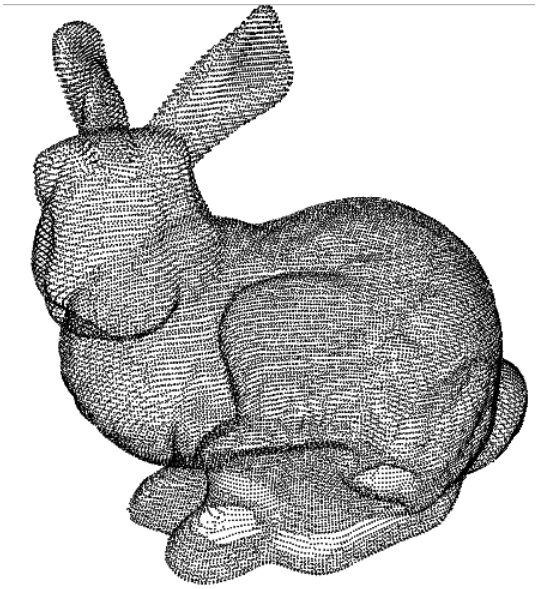


Abbildung 4.1.1: Die Punktwolke des Objekts „Hase"

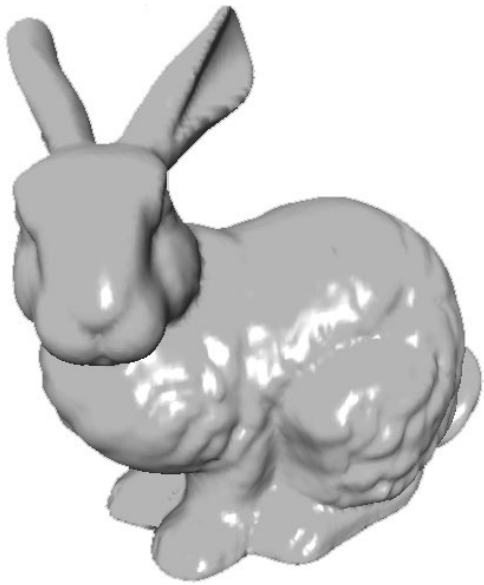


Abbildung 4.1.2: die Oberfläche des „Hasen" nach der Triangulation

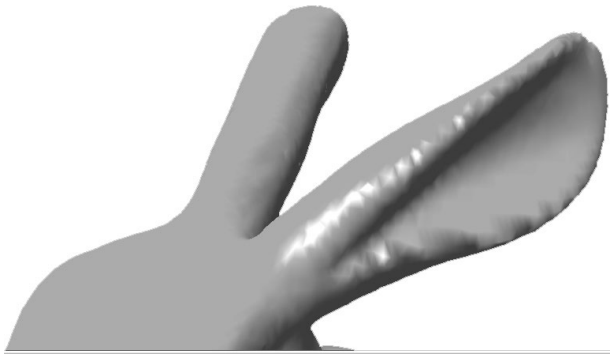


Abbildung 4.1.3: Detail der Triangulation

Als problematisch beim Hasen (Abbildung 4.1.1 – Abbildung 4.1.4) erwiesen sich vor allem der untere Teil der Figur, aufgrund der starken Krümmung und den zwei Löchern im Boden. Die Ohren, bei denen die Vorder und Rückseite sehr nahe zusammenkommen, stellten kein Problem für den Algorithmus dar, nachdem die Pro-

jektion der Nachbarn durch eine Drehung ersetzt wurde. Mit dieser Änderung lieferte der Algorithmus auch bei starken Krümmungen gute Ergebnisse.

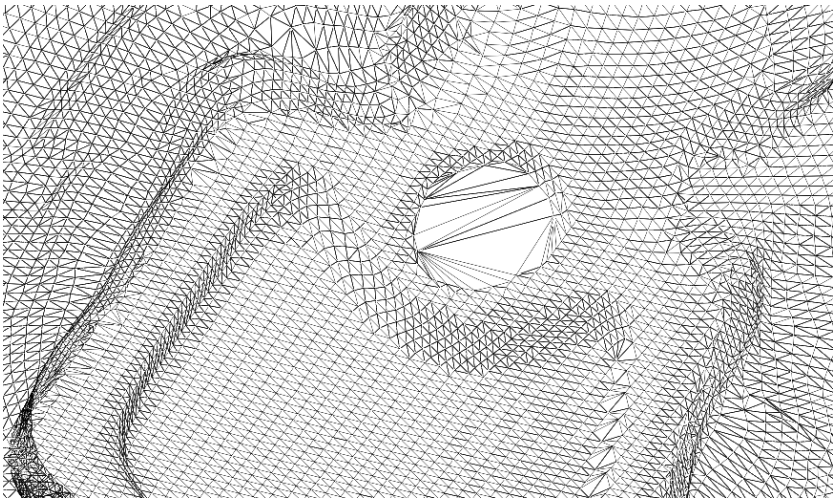


Abbildung 4.1.4: Das Loch im Boden des Objektes „Hase" wurde durch den Algorithmus geschlossen

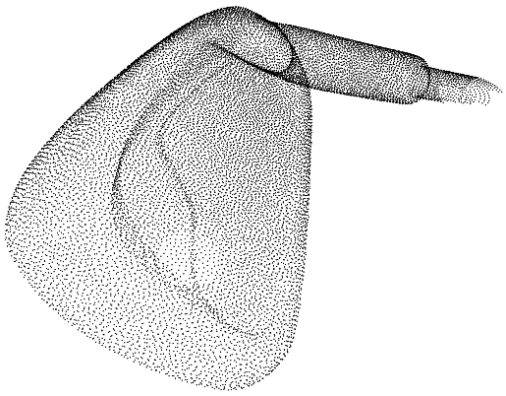


Abbildung 4.1.5: Punktwolke eines Golfschlägers

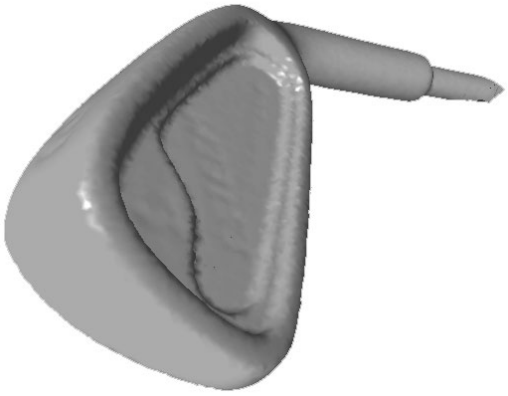


Abbildung 4.1.6: Schattierte Triangulation des Golfschlägers

Abbildung 4.1.7: Das Objekt „Fuß"

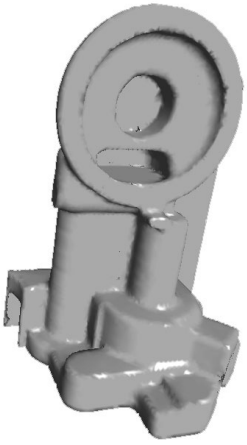


Abbildung 4.1.8: Das Objekt „Ölpumpe"

Objekt	**Punktanzahl**	**Normalenberechnung**	**Gitternetzberechnung**
Hase	35948	0,7	6,9
Golfschläger	16864	0,8	6,2
Fuß	20021	0,6	5,9
Ölpumpe	30937	0,5	7,5

Tabelle 4.1.1: Berechnungsdauer der Triangulation von realen Objekten (PC - Pentium III mit 1000Mhz und 256 MB)

generische Objekte:

Als ein Test, der fast alle problematischen Konstellationen erzeugt, hat sich eine generische Punktwolke aus zufällig gesetzen Punkten einer Kugeloberfläche erwiesen.

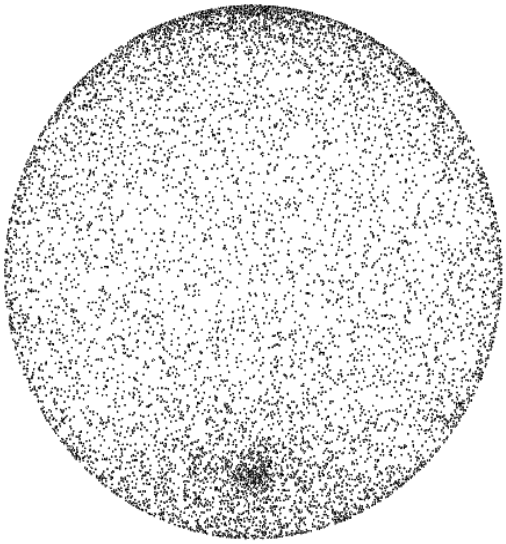


Abbildung 4.1.9: Eine Kugeloberfläche aus 10.000 zufällig gesetzten Punkten

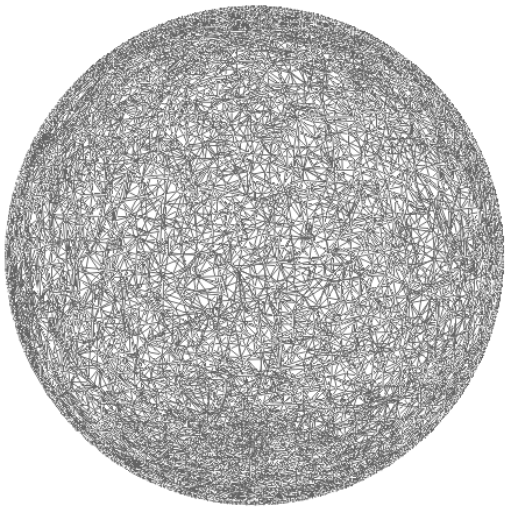


Abbildung 4.1.10: Triangulation der zufälligen Kugel

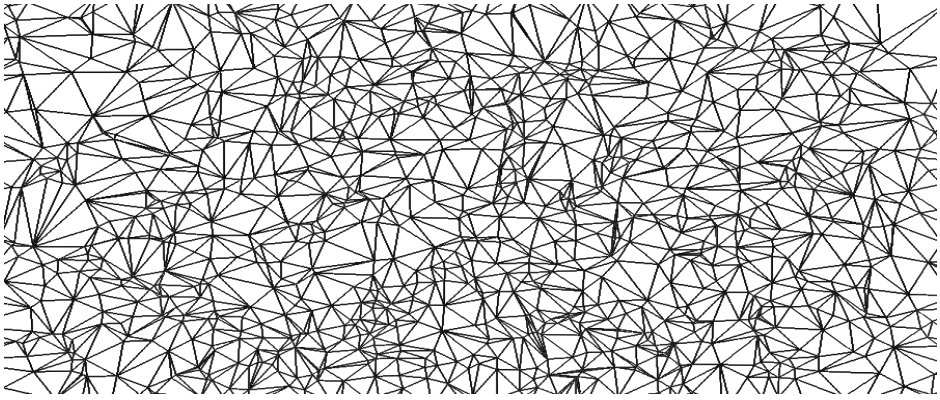


Abbildung 4.1.11: Detail der Triangulation der zufällig generierten Kugel

Um den Algorithmus auf kleine Winkel und mit einer optimalen Punktwolke zu testen wurden ebene Flächen mit einer quadratischen und einer hexagonalen Anordnung der Punkte generiert.

Objekt	**Punktanzahl**	**Normalenbe-rechnung**	**Gitternetzbe-rechnung**
Zufallskugel	5000	0,2	2,1
	10000	0,4	4,8
	40000	2,2	23,8
Quadratebene	40000	0,5	3,9
Hexagonalebene	40300	0,7	5,7

Tabelle 4.1.1: Berechnungsdauer der Triangulation von generisch erzeugten Objekten (PC - Pentium III mit 1000Mhz und 256 MB)

Wie schon im Kapitel 3.4 angedeutet, hat sich gezeigt, dass eine exakte Bestimmung der Normale für eine zufriedenstellende Triangulation nach diesem Algorithmus nicht unbedingt nötig ist. Die hier gezeigten Objekte wurden (bis auf die Löcher im Boden des Hasens, die sowieso problematisch sind) mit Normalen aus drei benachbarten Punkten zufriedenstellend trianguliert.

4.2 Fazit und Ausblick:

Der Algorithmus und dessen Programmierung war vor allem zur Erzeugung einer korrekten und zufriedenstellenden Triangulation auf der Basis der Arbeiten von Lars Linsen [Linsen '01] ausgerichtet.

Die Aufgabe, Triangulationen ohne Nachbearbeitungsschritte zu generieren, wird für Punktwolken, die dem in Kapitel 3.10 aufgestellten Kriterium genügen, erfüllt. Zu jedem Punkt werden Kandidaten gesucht, um einen Dreiecksfächer zu bilden, der die Umgebung der Oberfläche um diesen Punkt abdeckt. Diese Kandidaten werden auf Überschneidungen und der Einhaltung des Umkreiskriteriums geprüft, bevor der endgültige Fächer gebildet wird.

Löcher, die zu den Objekten gehören, werden hingegen nicht erkannt und der Algorithmus versucht diese zu schließen. Ein Vorverarbeitungsschritt, der solche Objektkanten erkennt und markiert wäre eine denkbare Erweiterung.

Es wurden auch in den einzelnen Schritten Überlegungen angestellt, das Verfahren zu beschleunigen. So wurde z.B. In Kapitel 3.3 das Umkreiskriterium untersucht und mit den Herleitungen auf einen einfachen Winkelvergleich beschränkt.Weiterhin wurden geometrische Beziehungen zwischen der Projektion und der Dreidimensionalen Punktkonstellation betrachtet, die noch weiter ausgeführt werden könnten. Doch es

mag weiteren Raum zur Optimierung geben. Schritt 7 des Algorithmus, der die meiste Zeit im Programmablauf beansprucht, könnte vielleicht noch anders bzw. geschickter gelöst werden. Des weiteren ließen sich die geometrischen Zusammenhänge zwischen einem Umkugelkriterium und dessen Abbildung in einen zweidimensionalen Raum weiter untersuchen. Der entwickelte Satz 3.3.5 stellt einen Anfang in diese Richtung dar.

Der Aufwand, die Konstellationen zu erkennen, die zu einer unbefriedigenden Triangulation führen können, war aus der Einfachheit des Ansatzes von Lars Linsen nicht vorauszusehen. Es erwies sich in zweierlei Hinsicht als zeitaufwendig, für jeden Punkt alle möglichen Problemfälle zu erkennen und zu verhindern. Einerseits verringert sich die Arbeitsgeschwindigkeit des Algorithmus pro weiteren Kriteriumtest für jeden Punkt, zum andern war aber auch die Analyse der Ursachen, die zu einem subjektiv erkennbar falschen Gitternetz führen, aufwendig. Eine Mischform könnte vielleicht einen schnelleres Verfahren ergeben. Zum Beispiel wäre eine Optimierung des Vorgängers dieses Algorithmus denkbar, mit einem anschließenden Schritt, der die selten auftretenden übrigen Fehlverbindungen und Löcher erkennt und behebt. Damit würde man sich dem in Kapitel 2 vorgestellten Ansatz von Patrica Crossno und Edward Angel annähern.

5 Literaturverzeichnis

[Bernardini et al. '99] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva und Gab, *The Ball-Pivoting Algorithm for Surface Reconstruction*, IEEE Transactions on Visualization and Computer Graphics , Seite 349-359, IEEE Press , 1999

[Boissonnat '00] Jean Daniel Boissonnat, Olivier Devillers, Monique Teillaud und Mariette Yvinec, *Triangulations in CGAL*, Proceedings of the sixteenth annual symposium on Computational geometry , Seite 11 ff, ACM Press New York, 2000

[Carr et al. '01] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum und T.R. Evans, *Reconstruction and representation of 3D Objetcs with Radial Basis Functions*, Proceedings of the 2001 conference on Computer Graphics , Seite 67-76, ACM Press New York, 2001

[Crossno/Angel '99] Patricia Crossno Edward Angel, *Spiraling Edge: Fast Surface reconstuction from Partially Organized Sample Points*, Proceedings of the conference on Visualization '99 , Seite 78-94, IEEE Computer Society Press Los Alamitos, 1999

[Dey et al. '01] Tamal K. Dey, Joachim Giesen und James Hudson, *Delaunay Based Shape Reconstruction from Large Data*, Proceedings IEEE 2001 symposium on parallel and large-data visualization and graphic , Seite 19-27, IEEE Press Piscataway, New York, 2001

[Floater/Reimers '01] Michael S. Floater und Martin Reimers, *Meshless parameterization and surface reconstruction*, Computer Aided Geometric Design , Seite 77-92, N H Elsevier , 2001

[Gopi et al. '00] M. Gopi, S. Krishnan, C.T. Silva, *Surface Reconstruction based on Lower Dimensional Localized Delaunay Triangulation*, Eurographics 2000 , 2000

[Herrmann '98] Thomas Herrmann: "*Entwicklung eines Algorithmus zur optimalen Triangulation von 3D-Punktwolken*", Diplomarbeit 1998, Universität Karlsruhe Forschungszentrum Informatik (FZI)

[John '99] Matthias John: "*Methoden zur Visualisierung und Verarbeitung von Punktwolken*", Diplomarbeit 1999,Universität Karlsruhe Institut für Betriebs- und Dialogsysteme

[Linsen '01] Lars Linsen: " *Oberflächenrepräsentation durch Punktwolken*", Shaker Verlag Aachen, 2001

[Ottman/Widmayer '93] Thomas Ottmann und Peter Widmayer: *Algorithmen und Datenstrukturen*, BI-Wissenschaftsverlag Mannheim, Leipzig, u.a., 1993

[Schmitt '96] Alfred Schmitt, Olivier Deussen, Marion Kreeb: *Einführung in graphisch-geometrische Algorithmen*, B.G. Teubner Stuttgart, 1996

[Stroustrup '94] Bjarne Stroustrup: *The C++ programming language (second edition)*, Addison-Wesley Publishing Company Reading, Massachusetts, 1994

[Wright et al. '00] Richard S. Wright Jr., Michael Sweet: *OpenGL Super Bible*, Waite Group Press Indianapolis, 2000

6 Anhang

Damit das im Laufe dieser Arbeit entstandene Programm weiterbearbeitet, oder Teile daraus wiederverwendet werden können, werden hier kurz die erstellten Klassen vorgestellt. Deren nähere Arbeitsweise kann dieser Diplomarbeit und dem kommentierten Quelltexten entnommen werden. Soweit von der Universität Karlsruhe nicht anders gehandhabt, kann (vom Autor aus) das Programm und dessen Quelltext nach der GNU public licence frei verbreitet und verändert werden.

6.1 Angaben zur Implementierung

Zur Analyse des Algorithmus wurde er unter Linux als Applikation des K Desktop Enviroments (KDE) mit der Entwicklungsumgebung KDevelop Version 2.0 implementiert. Zur Darstellung der Ergebnisse wurde die OpenGL – Schnittstelle der im KDE zugrunde liegenden QT-Bibliothek verwendet. Trotzdem wurde darauf geachtet, die für den Algorithmus benötigten Klassen von der Ein- und Ausgabefunktionalität, also von den systemspezifischen Bibliotheken zu trennen und unabhängig zu gestalten. Meistens werden nur Stringausgabe mittels QStrings aus der QT-Bibliothek, sowie OpenGL-Zeichenaufrufe verwendet. Die entsprechenden Klassenmethoden (in String, inGL...,) lassen sich einfach aus den Klassen herauslösen, sollten Sie ohne diese Bibliotheken verwendet werden.

Als Nachschlagewerke zur Programmierung wurden [Stroustrup '94], [Ottman/Widmayer '93] und [Wright et al. '00], sowie die Programmhilfe zur Kdevelop 2.0 Entwicklungsumgebung verwendet.

6.2 Überblick über die Struktur der implementierten C++ -Objekte

Es folgt eine nach Funktionalität und Klassenhierarchie sortierte Aufstellung der implementierten Klassen.

Grundsätzliche Mathematik:

vector2

vector3

verwendet: QT-Bibliothek, OpenGL-Bibliothek

Diese Klassen stellen zweidimensionale, bzw. dreidimensionale Vektoren dar und enthalten die grundlegenden Operatoren auf Vektoren (Addition, Subtraktion, Skalarmultiplikation, Vektormultiplikation, usw.).

QT wird nur für eine Stringausgabe verwendet und OpenGL-Aufrufe wenn, diese als Punkt oder Normale verwendet werden. Diese Abhängigkeiten lassen sich leicht herauslösen, wenn die Klassen ohne diese Bibliotheken genutzt werden sollen.

matrix2x3

matrix3x2

matrix3x3

Diese Klassen stellen verschieden Matrizenarten dar. Sie lassen sich verwenden um zweidimensionale Vektoren in dreidimensionale und umgekehrt zu überführen, oder im Fall der matrix3x3 Funktionen, um dreidimensionale Vektoren zu transformieren.

vector3mathe

verwendet: math.h

in dieser Klasse sind aufwendigere Vektoren und Matrizenoperationen gekapselt. Oft verwendete Funktionen, wie die Ermittlung der Länge eines Vektors, dessen Normalisierung, das Kreuzprodukt zweier Vektoren, aber auch die Berechnung von Polarkoordinaten und die Lösungmethoden für ein aus Matrix und Ergebnisvektor definiertes LGS sind hier enthalten.

Datenstrukturen und Organisation

color

verwendet: QT-Bibliothek, OpenGL-Bibliothek

Diese Klasse kapselt die Farbwerte Rot, Grün, Blau, und Alpha (Durchsichtigkeit) in einer Klasse. Die verwendeten Bibliotheken werden nur zur String-, bzw. OpenGL-Ausgabe verwendet und die entsprechenden Methoden lassen sich leicht herauslösen.

triangParams

Diese Klasse stellt einen Container für die Übergabe der Triangulationsparameter dar.

Punkt

verwendet: QT-Bibliothek, OpenGL-Bibliothek

Diese Klasse enthält die für einen Punkt der Punktwolke nötigen Daten und Methoden. Sie enthält neben den Koordinaten(vector3), Normalen(vector3 und Farbwerten (color) eine Liste seiner Nachbarpunkte

Punktarray

wird vererbt in: Punktarray_hold, Punktarray_pointer

Dies ist eine hauptsächlich virtuelle Klasse, die von ihren Ableitungen erst mit Funktion gefüllt wird. Grundsätzlich werden hier die Punkte in einem semidynamischen Feld angeordnet, dessen Größe verdoppelt und Punkte umkopiert werden, sollte die Feldgröße überschritten werden.

Punktarray_hold

abgeleitet von: Punktarray

Diese Klasse verwaltet ein Feld von Punkten.

Punktarray_pointer

abgeleitet von: Punktarray

Diese Klasse verwaltet eine Feld von Zeigern auf Punkte.

Punktlistelement

Diese Klasse enthält weitere Daten zu einem Punkt, wenn dieser in einer der folgenden Listenklassen organisiert wird. Sie kann den enthaltenen Punkt mitverwalten (d.h. er wird mit gelöscht, wenn das Element zerstört wird), oder nur einen Zeiger auf einen Punkt enthalten. Des weiteren werden hier Winkeldaten und Entfernungen gespeichert, die je nach Nachbarschaft unterschiedlich definiert sind. Ein Punkt kann so von vielen Punktlistelementen referenziert werden.

Punktliste

wird vererbt in: sortedByDistance, sortedByAngle

Diese Klasse organisiert die Punkte (Klasse Punkt) anhand der Klasse Punktlistelement in einer doppelt verketten Liste. Bei der Erzeugung wird angegeben, ob die Punkte mit den Punktlistelementen gelöscht werden sollen oder nicht.

sortedByDistance:

abgeleitet von: Punktliste

Diese Klasse sortiert ihre Elemente schon beim Einfügen nach Abstand zu einem bei der Erzeugung festgelegten Punkt. Die Punkte werden nicht mit den Elementen gelöscht.

sortedByAngle:

abgeleitet von: Punktliste

Diese Klasse ist ein Kernstück des Triangulationalgorithmus. Sie stellt die Nachbarschaftsliste und somit den zu bildenden Fächer um jeden Punkt dar. Bei ihrer Erzeugung wird ein Punkt mitgegeben, dessen Normale bestimmt sein sollte, da die Elemente schon beim Einfügen nach ihrem Polarkoordinatenwinkel, den ihre Abbildung auf der Tangentialebene einnimmt, sortiert werden. Die Punkte werden nicht mit den Elementen gelöscht.

cluster

abgeleitet von: Punktarray_Pointer

Diese Klasse repräsentiert einen Quader der Zellrasteraufteilung.

clusterspace

Diese Klasse ist das andere Kernstück des Triangulationalgorithmus. Sie stellt die Aufteilung des Raums in ein Zellraster dar, und verwaltet somit die Zellen (s. cluster) und ihre Inhalte. Sie enthält unter anderem die Methoden, um die Punktwolke zu triangulieren, einen Fächer zu bilden und die Normalen zu berechnen.

SweepStackElement

Diese Klasse enthält die Kanten (d.h Zeiger auf zwei Punkte) für den Radarstrahltest in Schritt 7a).

SweepStack

Diese Klasse verwaltet und sortiert die Kanten für den Radarstrahltest in Schritt 7a).

Dreieck

Diese Klasse repräsentiert ein Dreieck des zu erstellenden Gitternetzes, enthält also Pointer auf drei Punkte.

Dreiecksliste

Die Dreiecksliste verwaltet die Dreiecke wieder in einem semidynamischem Array und nimmt die fertige Triangulation auf.

Applikation

Primitiven

abgeleitet von: Punktliste

Dies ist eine Punktlistenklasse, die Methoden besitzt, um sich mit einer Punktwolke zu füllen, die diverse geometrische Primitiven darstellen.

idlg_stdfill

abgeleitet von: DLG_primitiven (QDialog)

Diese Basisklasse wurde mit dem QT-Dialogeditor erstellt und als idlg_stdfill mit Funktionalität gefüllt. Sie erzeugt den Dialog zur Primitivenauswahl.

timer

verwendet: QT-Bibliothek

Diese Klasse dient zur Zeitmessung und verwendet dafür einige QT-Klassen.

TriangulationApp

abgeleitet von KMainWindow

verwendet: QT-Bibliothek, KDE-Bibliotheken

Diese Klasse ist die aus dem KDE aufzurufende Applikation, und wurde ursprünglich von der Entwicklungsumgebung erzeugt.

TriangulationDoc

abgeleitet von QObject

verwendet: QT-Bibliothek, KDE-Bibliotheken

Diese Klasse kapselt die von der Applikation zu bearbeitenden Dokumente (Daten), und wurde ursprünglich von der Entwicklungsumgebung erzeugt.

TriangulationView

abgeleitet von QGLWidget

verwendet: QT-Bibliothek, KDE-Bibliotheken, OpenGL-Bibliothek

Diese Klasse ist verwaltet das OpenGL-Ausgabefenster.

TriangulationLog

abgeleitet von QTextView

verwendet: QT-Bibliothek, KDE-Bibliotheken

Diese Klasse verwaltet das Textfenster für die Status,meldungen.