Approximation verallgemeinerter Polyeder

Rainer Michael Schmid

26. August 1999

Diplomarbeit

nach einem Thema von Prof. Dr. Günter Hotz
in der Technischen Fakultät, Fachbereich 14 – Informatik, der Universität des Saarlandes, Saarbrücken.

Hiermit versichere ich an Eides Statt, dass ich diese Arbeit nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Saarbrücken, im August 1999

Anmerkung
Diese Arbeit wurde nach den Regeln der neuen Rechtschreibung erstellt.

Inhaltsverzeichnis

Mathematische Symbole

$u, v, s, t, \ldots$	Skalare	
$\vec{a}, \vec{b}, \ldots, \vec{p}, \vec{q}$	Vektoren	
$\vec{P}, \vec{Q}, \ldots$	Ortsvektoren von Punkten	
$\vec{r}(t)$	Raumkurve	
$r^{-1}(\vec{P})$	Inverse zu $\vec{r}(t)$	
$\vec{F}(u, v)$	Parametrisierung einer Oberfläche	
$\vec{F}^{-1}(\vec{P})$	Inverse zu $\vec{F}(u, v)$	
$\\| \cdot \\|$	Euklidische Norm	
C^i-Kurve	Kurve mit i-fach differenzierbarer Parametrisierung	
C^i-Fläche	Fläche mit i-fach differenzierbarer Parametrisierung	
$\#M$	Mächtigkeit der Menge M	
$\delta^H(A, B)$	Hausdorff-Distanz zwischen Punktemengen A und B	S.42
$\vec{\delta}^H(A, B)$	Gerichtete Hausdorff-Distanz zwischen Punktemengen A und B	S.42
$\mathfrak{C}_{\vec{r}}$	Die durch $\vec{r}$ parametrisierte Kurve	S.21
$\mathcal{R}$	Menge aller parametrisierbarer Kurven	S.21
$\mathfrak{O}_{\vec{F}}$	Die durch $\vec{F}$ parametrisierte Oberfläche	S.21
$\mathcal{F}$	Menge aller parametrisierbaren Oberflächen	S.21
$\vec{g}_{v_1 v_2}$	Parametrisierung der Geraden durch die Punkte v_1 und v_2	S.35
$\vec{E}_{v_1 v_2 v_3}$	Parametrisierung der Ebene durch die Punkte v_1, v_2 und v_3	S.35
$\mathfrak{B}$	Körper (im geometrischen, *nicht* im algebraischen Sinn)	S.32
$\mathfrak{V}_{\mathfrak{B}}$	Menge aller Knoten des Körpers $\mathfrak{B}$	S.32
$\mathfrak{E}_{\mathfrak{B}}$	Menge aller Kanten des Körpers $\mathfrak{B}$	S.32
$\mathfrak{L}_{\mathfrak{B}}$	Menge aller Schleifen des Körpers $\mathfrak{B}$	S.32
$\mathfrak{F}_{\mathfrak{B}}$	Menge aller Flächen des Körpers $\mathfrak{B}$	S.32
$\mathfrak{A}_E(e, \rho)$	Approximation der Kante e mit Genauigkeit ρ	S.33
$\mathfrak{A}_F(f, \rho)$	Approximation der Fläche f mit Genauigkeit ρ	S.33
$\mathfrak{A}_L$	Menge der approximierten Kanten der Schleifenmenge L	S.34
$\mathfrak{E}_L$	Menge aller Kanten der Schleifenmenge L	S.34

p_L	Polygon in der Ebene $\mathbb{R}^2$	S.34
p_L^Δ	Triangulierung des Polygons p_L	S.34
ρ, ρ_e, ρ_F	Genauigkeitsparameter für die Approximation eines Körpers, einer Kante bzw. einer Fläche	

1. Einleitung

In vielen Bereichen der Computergrafik und in dazu verwandten Gebieten wie z.B. CAD/CAM, in denen mit dreidimensionalen Objekten gearbeitet wird, ist es oft wünschenswert, nicht nur eine Menge von Polygonen oder Polyeder repräsentieren zu können, sondern auch Objekte mit gekrümmten Oberflächen.

Es gibt Anwendungen, bei denen es notwendig ist, dass eine Objektrepräsentation einen festen Körper darstellt. In diesen Fällen ist es wichtig, dass ein Objekt keine „Lücken“ oder „Durchdringungen“ aufweist. „Lücken“ entstehen, wenn zu einer Kante nur eine Fläche adjazent ist. Flächen, die sich durchdringen, sind ebenso unerwünscht. Die Objektrepräsentaton sollte diese beiden Fälle also nicht nicht enthalten.

Objekte mit gekrümmten Oberflächen, die gleichzeitig auch noch feste Körper repräsentieren, wollen wir im nachfolgenden verallgemeinerte Polyeder nennen. Dabei können die gekrümmten Oberflächen durch recht beliebige Kantenzüge (gegebenenfalls mit gekrümmten Kanten) begrenzt sein. Die genaue Repräsentation wird in Kapitel 3 genauer erläutert. In Abbildung 1.1 sieht man ein einfaches Beispiel eines solchen verallgemeinerten Polyeders.

Nun gibt es aber eine Reihe Anwendungen bzw. Algorithmen, die nur mit Polyederdaten arbeiten; hierzu zählen:

- *Echtzeitfähige Visualisierung* von dreidimensionalen Szenen erfordert polygo-

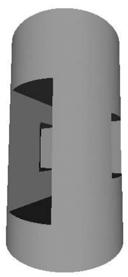


Abbildung 1.1.: Beispiel eines verallgemeinerten Polyeders.

nale Daten.

- Apparate zur *Stereolithographie* erfordern, dass die verwendeten Objektdaten aus Dreiecken bestehen. Insbesondere darf das Objekt keine Lücken und sich durchdringende Flächen haben. Die Entwurfsdaten vom CAD-System sind aber in der Regel verallgemeinerte Polyeder.

- *Finite Elemente Berechnungen* im Dreidimensionalen erfordern in der Regel eine Aufteilung des Objekts in Tetraeder. Ein Zwischenschritt bei diesem Prozess ist die Triangulierung der Objektoberfläche.

- Im Projekt SILVIA[1] zur *Bewegungssimulation* arbeiten momentan noch alle dreidimensionalen Algorithmen ausschließlich mit Polyederdaten.

Anhand dieser Beispiele sieht man, dass es wünschenswert ist, Algorithmen zu haben, die verallgemeinerte Polygone approximieren. Ein solcher Approximationsalgorithmus muss dabei sicherstellen, dass das entstehende Objekt einen festen Körper repräsentiert, sofern das auch für das Ausgangsobjekt schon galt. Die vorliegende Arbeit beschäftigt sich mit dieser Aufgabenstellung im Rahmen des Projektes SILVIA.

Es wird ein Algorithmus vorgestellt, der dies leistet. Dieser Algorithmus wurde für das Projekt SILVIA implementiert. Einige Beispielapproximationen dieses Algorithmus findet man im Anhang A.

Überblick und Gliederung

Die vorliegende Arbeit ist im Projekt SILVIA angesiedelt. Aus diesem Grund wird in Kapitel 2 die Bibliothek SILVIA kurz erläutert – zum einen wird die Aufgabenstellung der Bibliothek motiviert, und zum anderen wird ihre Architektur kurz beschrieben und gezeigt, in welchem Bereich diese Arbeit angesiedelt ist.

In Kapitel 3 wird beschrieben, wie dreidimensionale Objekte im ACIS-Datenformat repräsentiert werden, da dieses Format bei SILVIA verwendet wird. Für die verwendeten Kurven- und Oberflächentypen gilt, dass diese parametrisierbar sind. Diese Eigenschaften werden im Algorithmus an entscheidenden Stellen gebraucht. Deswegen wird in Kapitel 4 definiert, was eine parametrisierte Kurve bzw. eine parametrisierte Oberfläche ist, und anschließend werden für die verwendeten Kurven- und Oberflächentypen jeweils eine Parametrisierung und deren Inverse angegeben.

Kapitel 5 ist der Hauptteil, in dem der Algorithmus vorgestellt wird. Zuerst wird in Abschnitt 5.1 die Aufgabenstellung noch einmal genauer erläutert und von verwandten Fragestellungen abgegrenzt. In Abschnitt 5.2 wird versucht, verschiedene mögliche

[1] Eine Beschreibung der Bibliothek SILVIA findet man in Kapitel 2.

Lösungsansätze zu strukturieren. Es werden auch verschiedene Eigenschaften dieser Lösungsansätze diskutiert. Anhand dieser Diskussion wird dann begründet, für welches Verfahren ich mich entschieden habe. In Abschnitt 5.3 wird ein Algorithmus für dieses Verfahren vorgestellt – zunächst nur grob und dann im Folgenden immer detailreicher. Dadurch sollte es jedem möglich sein, gegebenenfalls den Algorithmus selber zu implementieren. Schließlich wird in Abschnitt 5.4 noch erläutert, was man tun muss, um den Algorithmus um neue Kurven- oder Oberflächentypen zu erweitern.

Im ersten Teil des Schlusskapitels 6 werden Algorithmen zur Approximation verallgemeinerter Polyeder aus verschiedenen Veröffentlichungen kurz dargestellt, nach der Klassifikation aus Abschnitt 5.2 charakterisiert und mit dem in dieser Arbeit vorgestellten Algorithmus verglichen. Im zweiten Teil dieses Kapitels wird ein Ausblick gegeben, in welche Richtungen der Algorithmus erweitert und verbessert werden könnte.

Der Anhang A besteht aus einigen Beispielbildern, die die Ergebnisse des Algorithmus illustrieren sollen.

Danksagung

Mein Dank gilt Herrn Prof. Dr. Dr. h.c. mult. Günter Hotz für die Vergabe der Diplomarbeit und die interessante Themenstellung. Besonders danken möchte ich auch Herrn Dr. Elmar Schömer, der immer mit Rat zur Stelle war und der sich auch für das Korrekturlesen zur Verfügung gestellt hat. Danken möchte ich auch den anderen Korrekturlesern, Andreas Kerzmann und Thomas Warken, die mich noch auf einige Fehler aufmerksam gemacht haben.
Ein ganz besonderer Dank gilt natürlich meinen Eltern, die mir das Studium erst ermöglicht haben.

2. Die Bibliothek SiLVIA

Technische Entwicklungen gehen immer schneller vonstatten. Die Geschwindigkeit, mit der neue Ideen umgesetzt werden, nimmt rapide zu. Will eine Firma bei der Produktentwicklung noch konkurrenzfähig sein, so ist notwendig, die Zeit von der Idee zum fertigen Produkt zu verkürzen.

Dies gilt z.B. auch für den Entwurfs- und Konstruktionsprozess in Automobilfirmen. Wird ein neuer Automobiltyp entworfen, so ist es wichtig, dass der Ein- und Ausbau von entscheidenden Teilen möglich bzw. einfach zu bewerkstelligen ist. Es ist z.B. nicht akzeptabel, dass man für den Wechsel einer Glühbirne den halben Motor auseinandernehmen muss.

Um einen Entwurf ausgiebig auf solche Designfehler zu testen, werden üblicherweise Prototypen gebaut. Da ein Entwurf in der Regel nicht sofort fehlerfrei ist, findet ein Redesign-Prozess statt, der wieder getestet werden muss, und so fort.

Es ist wünschenswert, diesen Design-Test-Redesign-Zyklus zu verkürzen, zu verbilligen und zu optimieren, da sich eine Verbesserung direkt in einen Marktvorteil auswirken kann. Firmen setzen nun darauf, mit Hilfe des Computers dies zu erreichen. Die realen Prototypen sollen mehr und mehr durch virtuelle Prototypen in einer VR-Umgebung ersetzt werden. Welche Schritte hierfür notwendig bzw. wünschenswert sind, kann man in [JCL97] nachlesen.

Ein Szenario für virtuelle Prototypen könnte z.B. aus einem Ingenieur bestehen, der in einer VR-Umgebung interaktiv eine Ein- oder Ausbausimulation an einem virtuellen Prototypen, der direkt aus den CAD-Entwurfsdaten gewonnen wurde, vornimmt. Die Ergebnisse sollten idealerweise auch an das CAD-Programm weitergeleitet werden können.

Dieses Szenario zeigt schon einige wichtige Komponenten, die in einem solchen System realisiert werden müssen:

- Übernahme von CAD-Daten an die VR-Umgebung
- Visualisierung des virtuellen Prototypen in Echtzeit
- Bereitstellung adäquater Eingabegeräte

- *echtzeitfähige* Bewegungssimulation
- Rücktransfer der Simulationsdaten an das CAD-System

Hierbei nimmt natürlich die Bewegungssimulation einen zentralen Platz ein. Das Projekt SILVIA am Lehrstuhl von Prof. Dr. Dr. h.c. mult. G. Hotz beschäftigt sich mit diesen Aufgabenstellungen. Der Schwerpunkt der Forschung am Lehrstuhl liegt dabei auf der echtzeitfähigen Bewegungssimulation. Die anderen Aspekte kann man aber nicht vernachlässigen. Das Projekt SILVIA hat sich zum Ziel gesetzt, eine Softwarebibliothek zu dem Aufgabenbereich der Bewegungssimulation in virtuellen Welten zu schaffen. SILVIA ist die Abkürzung für *Simulation Library for Virtual Reality and Interactive Applications*.

Für die physikalische Simulation gibt es momentan zwei Vorgehensweisen, die auch in der Bibliothek SILVIA realisiert wurden:

- Die *zwangsbasierte Simulation*, für die Baraff [Bar89] wesentliche Arbeit geleistet hat. In [War99] ist sie für das SILVIA Projekt beschrieben.
- Die *impulsbasierte Simulation*, die auf Mirtich [Mir96] zurückgeht. Im Rahmen von SILVIA ist sie in [Len99] beschrieben.

Die anderen Aspekte von SILVIA werden nachfolgend noch kurz erläutert. Hilfreich hierzu ist auch die Abbildung 2.1, in der die Architektur von SILVIA dargestellt ist (eine ähnliche Beschreibung von SILVIA findet man auch in [HKL+99]).

Auf der linken Seite der Abbildung 2.1 sieht man die Teile, die unterstützt werden, um ein Graphical User Interface (GUI) zu realisieren. Die Grafikausgabe erfolgt über OpenGL, und Menüs können mit Tcl/Tk realisiert werden. Daneben gibt es noch die Möglichkeit für verteilte virtuelle Welten, die auf Java und Vrml aufgebaut sind.

Die rechten Kreise beschreiben die eigentliche Bibliothek SILVIA. Im Kern findet man die Objektrepräsentation, die auf der ACIS-Datenstruktur beruht. Diese ist ausführlicher in Kapitel 3 beschrieben.

Direkt auf dieser Datenstruktur bauen dann Algorithmen für Hüllkörperhierarchien und zur Raumpartitionierung auf. Dies wird vor allem für eine effiziente Kollisionserkennung und für effiziente Abstandsberechnungen benötigt, die zusammen mit typischen CAD-Operationen die nächste Schicht bilden.

Zu diesen CAD-Operationen zählt insbesondere auch der in dieser Arbeit vorgestellte Algorithmus zur Approximation verallgemeinerter Polyeder. Aber auch die Berechnung der physikalischen Momente für Körper (Schwerpunkt, Masse, Trägheitsmomente) und einfache Objektmanipulationen wie verschieben, rotieren, etc. fallen in diesen Bereich.

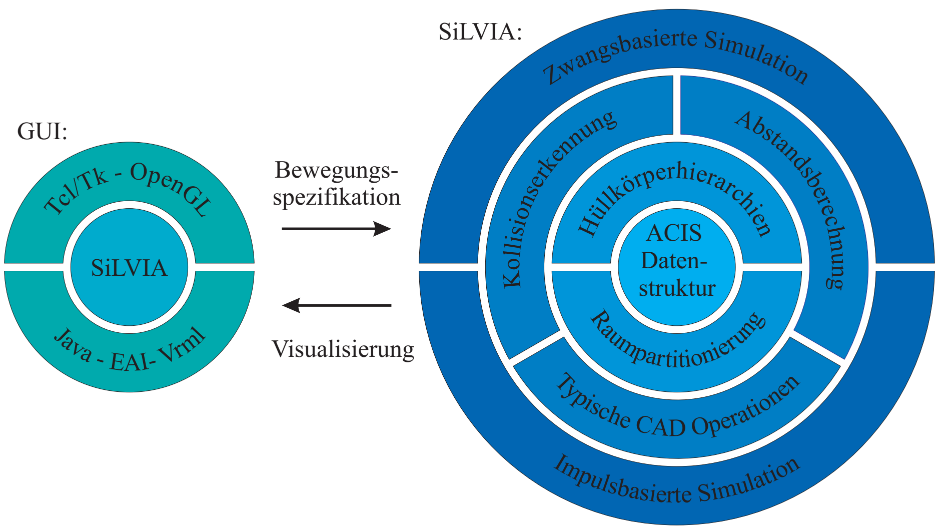


Abbildung 2.1.: Der Aufbau von SILVIA.

Die äußere Schicht wird von den eigentlichen Algorithmen zur Simulation gebildet. Wie schon gesagt, wurden die zwei Paradigmen zwangsbasierte Simulation und impulsbasierte Simulation implementiert. Diese Algorithmen benötigen teilweise unterschiedliche Teile aus darunterliegenden Schichten.

Insgesamt bildet dieser Entwurf eine gute Möglichkeit, die Bibliothek für vielfältige Einsätze zu gebrauchen. Die Bibliothek ist komplett in C/C++ geschrieben und läuft auf vielen gängigen Plattformen (Linux, Solaris, Windows NT). Die Grafikausgabe ist von der eigentlichen Bibliothek abgetrennt. Durch diese Trennung erhöht sich die Plattformunabhängigkeit, da die Grafikausgabe leicht auf andere Systeme erweitert werden kann. Dadurch erhöht sich der mögliche Einsatzrahmen.

Ferner erlaubt diese Architektur auch recht einfach das Hinzufügen neuer Algorithmen. Eine Erweiterung von SILVIA um neue oder bessere Funktionalitäten sollte deswegen problemlos möglich sein.

3. Objektrepräsentation

In diesem Kapitel werden die genauen Datenstrukturen zur Objektrepräsentation erläutert. Ein besonderes Augenmerk wird hierbei auf die Repräsentation von Flächen gelegt, da dies später von großer Bedeutung sein wird.

Im Projekt SILVIA werden Objekte durch eine Untermenge des ACIS-Formats aus [Spa96] benutzt. Die Repräsentationsform ist eine Randdarstellung (boundary representation oder b-rep)[1]; d.h. die Objekte werden durch ihre Begrenzungsflächen und -kanten beschrieben.

Das Dateiformat nimmt zunächst eine grobe Unterteilung in topologische und geometrische Daten vor.

Die topologischen Klassen beschreiben Beziehungen zwischen einem Körper (`Body`[2]), seinen Teilen (`Lump`), Schalen (`Shell`), Flächen (`Face`), Schleifen (`Loop`), Halbkanten (`Coedge`), Kanten (`Edge`) und Knoten (`Vertex`) zueinander. Die geometrischen Klassen beschreiben hingegen konkrete geometrische Ausprägungen von Oberflächen (`Surface`), Kurven (`Curve`) und Punkten (`Point`).

Als Oberflächentypen werden Ebenen (`Plane`), Kugeln (`Sphere`), Kegel bzw. Zylinder (`Cone`) und Tori (`Torus`) verwendet; als Kurventypen Geraden (`Straight`) und Ellipsen (`Ellipse`).

Die Beschränkung auf diese beiden einfachen Kurventypen (Gerade und Ellipse) wird vorgenommen, da dies neben NURBS- und Bézier-Kurven die einzigen Kurventypen im ACIS-Format sind. Außerdem reichen diese für die vorgestellten Oberflächentypen aus, um komplexere Obejkte zu erzeugen. Eine Erweiterung der Algorithmen auf andere Kurventypen ist nicht weiter schwierig. Was man dazu tun muss, ist in Abschnitt 5.4.1 beschrieben.

Zusätzlich gibt es noch die Klasse `Entity`, von der alle Klassen abgeleitet werden. Abbildung 3.1 zeigt die hierarchische Klassenableitungen (aufgeteilt nach topologischen und geometrischen Klassen).

[1] Eine ausführliche Beschreibung verschiedener Repräsentationsformen entnehme man [FvDFH96].

[2] In den Klammern wird jeweils der Name aus [Spa96] angegeben, der auch für die Implementierung (mit dem Präfix `Acis`) verwendet wurde.

Topologische Klassen: Geometrische Klassen:

```
Entity                 Entity
├─► Body               ├─► Point
├─► Lump               ├─► Curve
├─► Shell              │    ├─► Straight
├─► Face               │    └─► Ellipse
├─► Loop               └─► Surface
├─► Coedge                  ├─► Plane
├─► Edge                    ├─► Sphere
├─► Vertex                  ├─► Cone
├─► Wire                    └─► Torus
└─► Transform
```

Abbildung 3.1.: Ableitung der ACIS Klassen

3.1. Topologischer Aufbau

Den kompletten topologischen Aufbau eines Körpers entnehme man Abbildung 3.2. Die durchgezogenen Pfeile in der Abbildung bedeuten, dass ein Objekt Kind oder Geschwister eines anderen ist. Die gestrichelten Pfeile sind Verweise auf den Vater. Nachfolgend wird die Bedeutung der einzelnen Klassen näher erläutert.

Ein *Körper* besteht aus einem oder mehreren *Teilen*. Teile sind unzusammenhängende Stücke eines Körpers. Diese Teile wiederum bestehen aus *Schalen*, die ihrerseits aus *Flächen* zusammengesetzt sind. Zwei Schalen besitzen keine Flächen, die zueinander adjazent sind. Als Beispiel diene eine hohle Kugel: diese besteht aus zwei Schalen – einer äußeren und einer inneren, die den Hohlraum beschreibt.

Eine Fläche hat eine konkrete geometrische Form (realisiert durch einen Zeiger auf die geometrische Klasse für *Oberflächen*). Ferner ist eine Fläche in der Regel durch eine oder mehrere *Schleifen* begrenzt (es gibt auch Fälle, in denen Flächen durch keine Schleifen begrenzt sind; z.B. eine komplette Kugel).

Die begrenzenden Schleifen sind ihrerseits aus *Halbkanten* zusammengesetzt. Halbkanten entsprechen gerichteten Kanten in einer Schleife. Sie sind notwendig, da eine Kante in mehreren Loops vorkommen kann (evtl. entgegengesetzt orientiert). Eine Halbkante enthält einen Verweis auf die konkrete *Kante*, die sie repräsentiert. Zusätzlich ist noch ein Flag vorhanden, das angibt, ob die Halbkante entgegengesetzt zu der Kante orientiert ist. Alle Halbkanten, die zu der gleichen Kante gehören sind in einer zyklischen Liste verkettet.

Die Halbkanten einer Schleife sind orientiert: in der Ebene sind äußere Schleifen im Gegenuhrzeigersinn orientiert und innere Schleifen im Uhrzeigersinn. Siehe hierzu auch Beispiel 3.1. Für allgemeinere Oberflächentypen sind die Bezeichnungen „äußere und innere Schleifen“ nicht mehr möglich. Hier gilt aber entsprechend, dass sich das

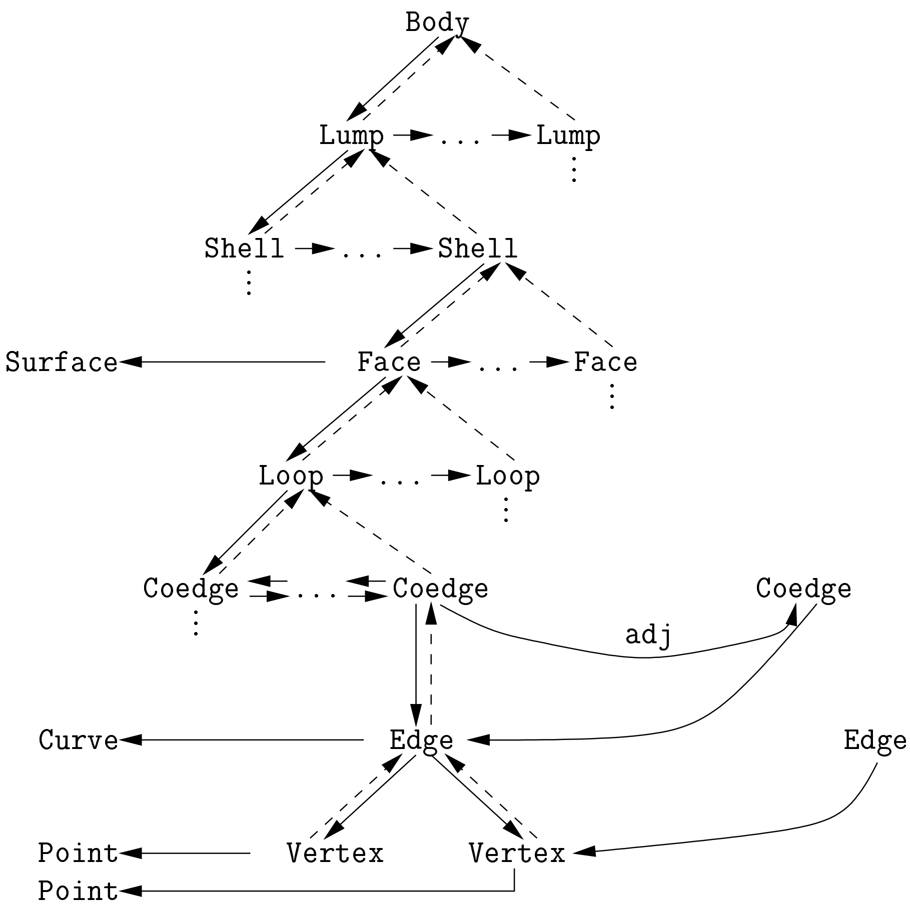


Abbildung 3.2.: Topologischer Aufbau eines ACIS-Body.

Innere der Fläche links[3] von der Halbkante befindet.

Mit Hilfe der Halbkanten sind wichtige Anfragen möglich; z.B. möchte man alle Flächen erreichen, die zu einer Kante adjazent sind. Dies wird erreicht, indem man eine Halbkante zu dieser Kante betrachtet. Die zu einer Halbkante gehörige Fläche ist über ihre Schleife erreichbar. Indem man alle Halbkanten durchläuft, die zu der Kante gehören, erreicht man alle Flächen, die zu dieser Kante adjazent sind.

Kanten können aus Strecken oder Ellipsenbögen bestehen. Repräsentiert werden sie durch eine konkrete geometrische Ausprägung (Klasse für *Kurven*) und einen *Start-* und *Endknoten*. Diese Knoten haben auch eine geometrische Ausprägung: einen *Punkt*. Hierbei sind einige Sonderfälle zu beachten:

- Ist die Kante eine geschlossene Kurve, so sind entweder beide Verweise auf den Knoten `NULL` oder identisch.
- Einzelne Punkte (z.B. eine Kegelspitze) werden als Kanten repräsentiert, indem der Verweis auf die Kurve `NULL` ist und die beiden Verweise auf den Knoten identisch sind.

3.1.1. Beispiel

Beispiel 3.1 In Abbildung 3.3 sieht man eine (teilweise krummlinig begrenzte) Fläche mit Löchern. Die Pfeile bei den Kanten geben die Richtung der zugehörigen Kanten und Halbkanten an (in diesem Beispiel wird angenommen, dass Kante und Halbkante gleich orientiert sind, was im Allgemeinen aber nicht gilt). Unter dem Bild ist ein Ausschnitt aus der Repräsentation der Fläche skizziert (c_i bezeichnet hierbei die zur Kante e_i gehörende Halbkante).

Zu beachten ist, dass die Reihenfolge der Schleifen nicht von Bedeutung ist. Die Reihenfolge der Halbkanten ist sehr wohl von Bedeutung, da dadurch entschieden wird, was das Äußere bzw. das Innere der Fläche ist.

Beispiel 3.2 Als nächstes soll ein kompletter Körper als Beispiel dienen: eine Kugel mit einer zylindrischen Durchbohrung (Abbildung 3.4). Wie dies in ACIS repräsentiert wird, kann man in Abbildung 3.5 sehen.

Der Körper B_1 besteht aus nur einem Teil L_1 und einer Schale S_1. Die Schale besteht aus zwei Flächen: der Kugelfläche f_1 und der Zylinderfläche f_2. Das Richtungsflag für die Fläche f_2 gibt an, dass die Normale gekippt werden muss (d.h. die Normale zeigt dann in das „Innere" des Zylinders).

[3] „Links" bedeutet hierbei, dass man in der Richtung der Halbkante bzgl. der Normalenrichtung der Oberfläche schaut.

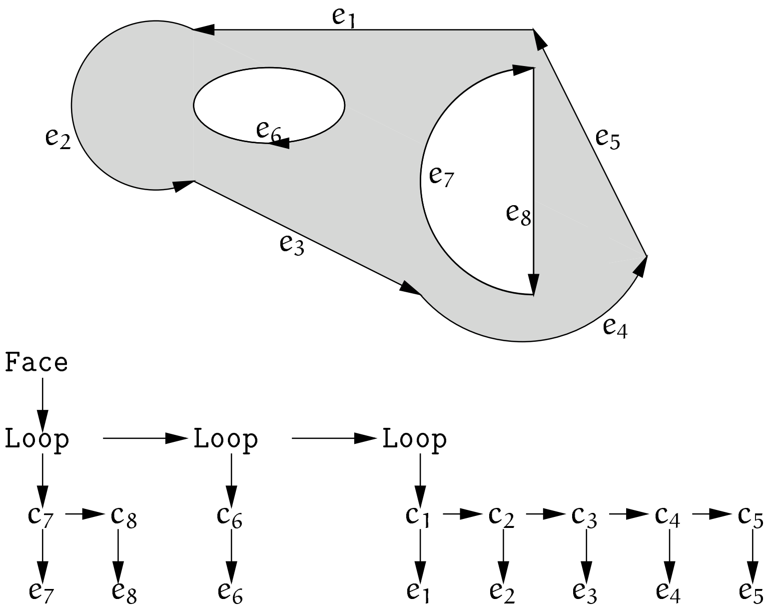


Abbildung 3.3.: Beispiel zur Definition einer Fläche.

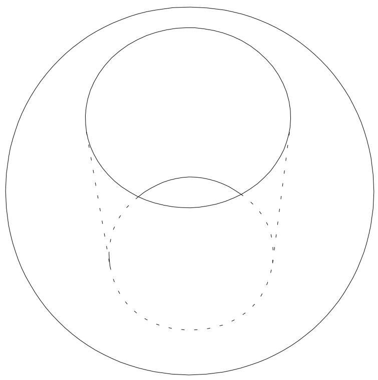


Abbildung 3.4.: Beispiel einer Kugel mit einem zylindrischen Loch.

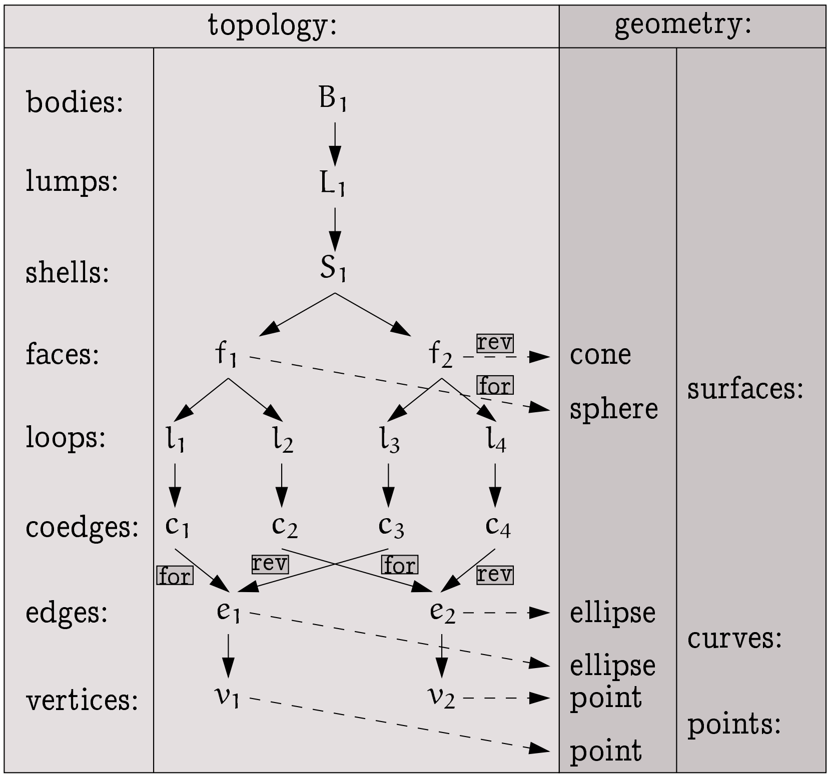


Abbildung 3.5.: Repräsentation einer Kugel mit einem zylindrischen Loch.

Beide Flächen sind jeweils durch zwei Schleifen (l_1 bis l_4) begrenzt. Jede dieser Schleifen besteht aus einer Halbkante (c_1 bis c_4), die ihrerseits auf eine Kante (e_1 oder e_2) verweisen. Man sieht, dass man nur zwei Kanten braucht (der obere und untere Kreis). Das Richtungsflag der Halbkanten sagt dann, ob man die Richtung der Kante invertieren muss oder nicht.

3.2. Geometrische Daten

Bisher wurde nur beschrieben, wie topologische Beziehungen ausgedrückt werden. In diesem Abschnitt soll beschrieben werden, wie die geometrischen Daten repräsentiert werden.

3.2.1. Punkt

Ein Punkt wird einfach durch seine drei Koordinaten dargestellt.

3.2.2. Kurven

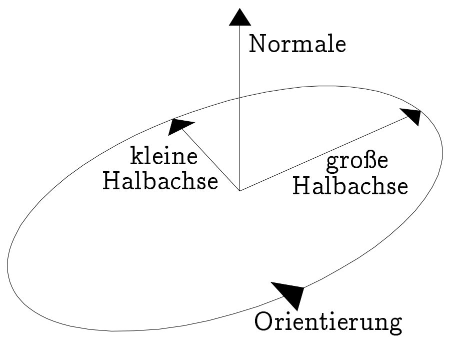
Gerade: Eine Gerade wird durch einen *Punkt*, der auf der Geraden liegt und einen *Richtungsvektor* angegeben. Die Orientierung der Kante ist in Richtung dieses Vektors.

Ellipse: Eine Ellipse wird durch den *Mittelpunkt* der Ellipse, einen *Einheits-Normalenvektor*, der die Ebene angibt, in der die Ellipse liegt, einen Vektor, der die *große Halbachse* angibt (sowohl die Richtung als auch die Länge d.h. den Radius der großen Halbachse) und dem *Verhältnis* des Radius der kleinen Halbachse zu dem Radius der großen Halbachse repräsentiert. Dabei steht die kleine Halbachse senkrecht auf der großen Halbachse und der Normalen. Die Orientierung der Ellipse ergibt sich mit der „rechten Hand“-Regel bezüglich der Richtung des Normalenvektors. Siehe hierzu auch Abbildung 3.6(a).

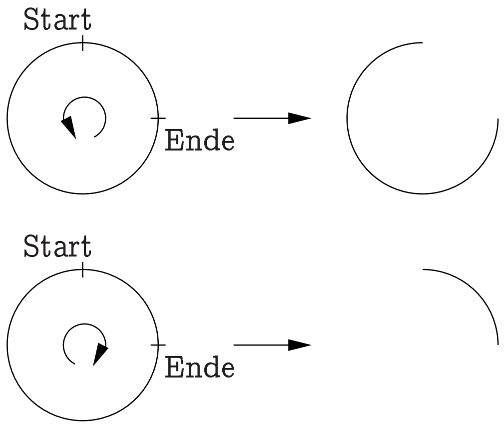
Die Orientierung der Ellipse ist wichtig, wenn eine Kante nur aus einem Teil der Ellipse besteht. Siehe hierzu auch Abbildung 3.6(b).

3.2.3. Oberflächen

Ebene: Eine Ebene wird durch einen *Punkt* in der Ebene und einen *Normalenvektor* spezifiziert. Zusätzlich wird noch ein *Richtungsvektor* gespeichert, der senkrecht zu der Normalen ist. Dies ist wichtig für die Parametrisierung (siehe



(a) Die Orientierung ergibt sich aus dem Normalenvektor.



(b) Die Orientierung ist wichtig für die Definition elliptischer Kanten.

Abbildung 3.6.: Definition einer `Ellipse` und elliptischer Kanten.

Abschnitt 4.3.1). Ferner enthält die Klasse noch ein *Flag*, ob das Äußere sich in Richtung der Normalen oder entgegengesetzt dazu befindet.

Kugel: Eine Kugel spezifiziert man durch den *Mittelpunkt* und den *Radius*. Ein *Flag* gibt an, ob das Äußere aus den Punkten besteht, die einen größeren Abstand als den Radius zum Mittelpunkt besitzen, oder aus denen, die einen kleineren Abstand zum Mittelpunkt besitzen. Zusätzlich sind noch zwei Vektoren gespeichert, die die *Richtung des Pols* und die *Richtung des* (u, v)*-Ursprungs*[4] angeben. Beide Vektoren sind normiert. Sie werden für die Parametrisierung (Abschnitt 4.3.2) benötigt.

Kegel und Zylinder: Die Repräsentation eines Kegels besteht aus einer *Ellipse* und dem *Sinus-* und *Kosinuswert* des halben Hauptwinkels. Weiter gibt ein *Flag* an, wo sich das Äußere befindet. Die Normale der Basisellipse definiert die Achse des Kegels.

Das Vorzeichen der Sinus- und Kosinuswerte liefert eine Fallunterscheidung für die genaue Gestalt des Kegels. Sind die Vorzeichen gleich, so nimmt der Querschnitt in Achsenrichtung zu, andernfalls ab. Ist der Kosinuswert positiv, so ist die Oberfläche von der Achse weg orientiert, ist er negativ, so ist sie zu der Achse hin orientiert. Für eine Übersicht dieser Definition siehe Tabelle 3.1.

Ist der Sinuswert Null, so spezifiziert man einen Zylinder. Falls der Kosinuswert Null ist, so hat man ein planares Gebilde.

Torus: Der Torus wird durch den *Mittelpunkt*, die *Normale* zu der Ebene, in der der Hauptkreis liegt, und durch den *Haupt-* und *Nebenradius* spezifiziert. Des Weiteren gibt ein *Flag* an, wo sich das Äußere befindet. Zusätzlich gibt ein normierter Vektor die *Richtung des* (u, v)*-Ursprungs*[5] an; dies wird für die Parametrisierung (Abschnitt 4.3.4) benötigt.

[4] Der (u, v)-Ursprung ist der Bildpunkt der Parametrisierung, der zu dem Punkt $(0, 0)$ im Parameterraum gehört. Siehe hierzu auch Kapitel 4. Die Richtung des (u, v)-Ursprungs bezeichnet einen Vektor vom Mittelpunkt der Kugel in Richtung dieses Punktes.

[5] Vergleiche hierzu die Anmerkung bei der Kugel.

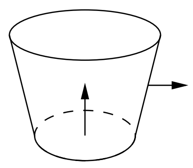
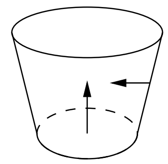
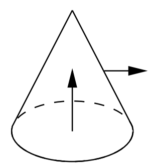
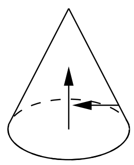
Sinus	Kosinus	Wirkung	Beispiel
positiv	positiv	Querschnittsfläche nimmt in Achsenrichtung zu. Oberfläche ist von der Achse weg orientiert (konvex).	
negativ	negativ	Querschnittsfläche nimmt in Achsenrichtung zu. Oberfläche ist zu der Achse hin orientiert (konkav).	
negativ	positiv	Querschnittsfläche nimmt in Achsenrichtung ab. Oberfläche ist von der Achse weg orientiert (konvex).	
positiv	negativ	Querschnittsfläche nimmt in Achsenrichtung ab. Oberfläche ist zu der Achse hin orientiert (konkav).	

Tabelle 3.1.: Zusammenhang zwischen Vorzeichen von Sinus- und Kosinuswerten und dem Aussehen des definierten Kegels

4. Parametrisierungen

Die Kurven- und Oberflächentypen, die wir betrachten sind parametrisierbar. Dies wird für die Approximation gebraucht, da eine Punktmenge im Parameterraum der Oberfläche trianguliert wird. Ebenso wird die Parametrisierung der Kurven für die Approximation von Kanten gebraucht. In diesem Kapitel werden entsprechende Parametrisierungen vorgestellt.

Für den in Kapitel 5 vorgestellten Algorithmus ist es auch notwendig, zu einem Punkt im Modellraum den entsprechenden Punkt im Parameterraum zu bestimmen. Deswegen wird zu jeder Parametrisierung noch die Berechnung ihrer Inversen hergeleitet.

4.1. Definitionen

Zuerst wird aber definiert, was man unter einer parametrisierten Kurve bzw. Oberfläche versteht.

Definition 4.1 Sei $I \subseteq \mathbb{R}$ und $\vec{r} : I \to \mathbb{R}^3$ mit $t \mapsto (r_1(t), r_2(t), r_3(t))^T$. Die *durch* $\vec{r}$ *parametrisierte Kurve* $\mathfrak{C}_{\vec{r}}$ ist definiert als

$$\mathfrak{C}_{\vec{r}} := \{\vec{r}(t) \mid t \in I\}.$$

I heißt auch der *Parameterraum von* $\mathfrak{C}_{\vec{r}}$ und $\mathbb{R}^3$ der *Modellraum.* Mit

$$\mathcal{R} := \{\vec{r} \mid \vec{r} : I \to \mathbb{R}^3 \text{ für ein } I \subseteq \mathbb{R}\}$$

wird die *Menge aller parametrisierbaren Kurven* bezeichnet.

Definition 4.2 Seien $I, J \subseteq \mathbb{R}$ und $\vec{F} : I \times J \to \mathbb{R}^3$ mit $(u, v) \mapsto (F_1(u, v), F_2(u, v), F_3(u, v))^T$. Die *durch* $\vec{F}$ *parametrisierte Oberfläche* $\mathfrak{O}_{\vec{F}}$ ist definiert als

$$\mathfrak{O}_{\vec{F}} := \{\vec{F}(u, v) \mid u \in I, v \in J\}.$$

$I \times J$ heißt auch der *Parameterraum von* $\mathfrak{O}_{\vec{F}}$ und $\mathbb{R}^3$ der *Modellraum.* Mit

$$\mathcal{F} := \{\vec{F} \mid \vec{F} : I \times J \to \mathbb{R}^3 \text{ für } I, J \subseteq \mathbb{R}\}$$

wird die *Menge aller parametrisierbaren Oberflächen* bezeichnet.

Bemerkung 4.1 Die Parametrisierung zu einer gegebenen Kurve bzw. Oberfläche ist nicht eindeutig. Nachfolgend wird lediglich eine mögliche Parametrisierung vorgestellt, die dann für die Algorithmen und deren Implementierung verwendet wird. Die vorgestellten Parametrisierungen orientieren sich an dem Vorschlag aus [Spa96], da bei der konkreten Implementierung auf den ACIS-Daten gearbeitet wird und die Parametrisierung dafür ausgelegt ist.

4.2. Parametrisierungen der Kurven

Die in ACIS verwendeten Kurventypen sind parametrisierbar. In diesem Abschnitt sollen die entsprechenden Parametrisierungen und ihre Inverse angegeben werden.

4.2.1. Gerade

Sei $\vec{P}$ ein Punkt auf der Geraden g und $\vec{a}$ ein normierter Richtungsvektor von g, dann wird g durch

$$\vec{r}_g(t) = \vec{P} + t\vec{a}, \quad t \in \mathbb{R} \tag{4.1}$$

parametrisiert.

Die Inverse $r_g^{-1}(\vec{Q})$ von $\vec{r}_g$ ist die Lösung t des linearen Gleichungssystems[1] $t \cdot \vec{a} = \vec{Q} - \vec{P}$.

4.2.2. Ellipse

Sei $\vec{C}$ der Mittelpunkt der Ellipse e und $\vec{a}$ bzw. $\vec{b}$ die große bzw. kleine Halbachse (sowohl Richtung als auch Länge), dann wird e durch

$$\vec{r}_e(t) = \vec{C} + \cos t \cdot \vec{a} + \sin t \cdot \vec{b}, \quad t \in (-\pi, \pi] \tag{4.2}$$

parametrisiert. Bei einem Kreis entspricht t anschaulich dem Winkel zwischen der großen Halbachse $\vec{a}$ und dem „Radius" durch den Punkt $\vec{r}_e(t)$. Bei einer Ellipse ist es nicht direkt der Winkel. Eine Ellipse erhält man z.B. aus einem Kreis mit Radius $\vec{b}$ durch Streckung in Richtung $\vec{a}$. Eine entsprechende Konstruktion sieht man in Abbildung 4.1.

Um die Inverse $r_e^{-1}(\vec{Q})$ von $\vec{r}_e$ zu berechnen, muss man die Lösung von

$$\cos t \cdot \vec{a} + \sin t \cdot \vec{b} = \vec{Q} - \vec{C} \tag{4.3}$$

[1] Liegt $\vec{Q}$ auf g, so existiert die Lösung und ist eindeutig bestimmt.

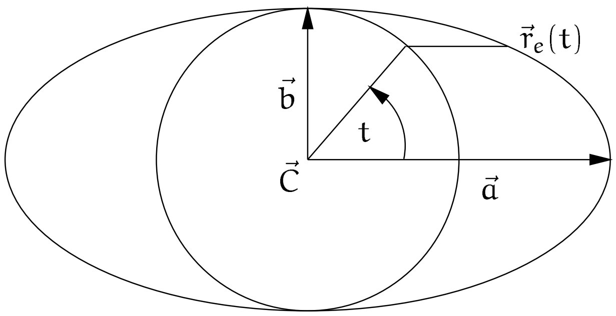


Abbildung 4.1.: Parametrisierung einer Ellipse.

bestimmen. Hierfür nutzt man aus, dass $\vec{a}$ und $\vec{b}$ senkrecht sind. Multipliziert man Gleichung 4.3 mit $\vec{a}$ bzw. $\vec{b}$, so erhält man

$$\cos t = \frac{\vec{a} \cdot (\vec{Q} - \vec{C})}{\vec{a} \cdot \vec{a}}$$
$$\sin t = \frac{\vec{b} \cdot (\vec{Q} - \vec{C})}{\vec{b} \cdot \vec{b}}.$$

Liegt $\vec{Q}$ auf e, so besitzt dieses Gleichungssystem eine Lösung, die für $t \in (-\pi, \pi]$ eindeutig ist.

4.3. Parametrisierungen der Oberflächen

Ähnlich wie die Kurven sind auch die betrachteten Oberflächen parametrisierbar. Sie sollen in diesem Abschnitt näher betrachtet werden.

4.3.1. Ebene

Sei $\vec{P}$ ein Punkt in der Ebene E. Sei $\vec{n}$ der Normalenvektor von E und $\vec{u}_d$ ein Richtungsvektor. Sagt das Richtungsflag der Klasse, dass die Normalenrichtung gekippt ist, dann sei $\vec{v}_d := \vec{u}_d \times \vec{n}$, sonst $\vec{v}_d := \vec{n} \times \vec{u}_d$. Die Ebene E wird durch

$$\vec{F}_E(u, v) = \vec{P} + u \cdot \vec{u}_d + v \cdot \vec{v}_d, \quad u, v \in \mathbb{R} \tag{4.4}$$

parametrisiert.

Ähnlich wie bei der Geraden ist die Inverse $\vec{F}_E^{-1}(\vec{Q})$ von $\vec{F}_E$ die Lösung des linearen Gleichungssystems[2] $u \cdot \vec{u}_d + v \cdot \vec{v}_d = \vec{Q} - \vec{P}$.

[2] Liegt $\vec{Q}$ in E, so existiert die Lösung und ist eindeutig bestimmt.

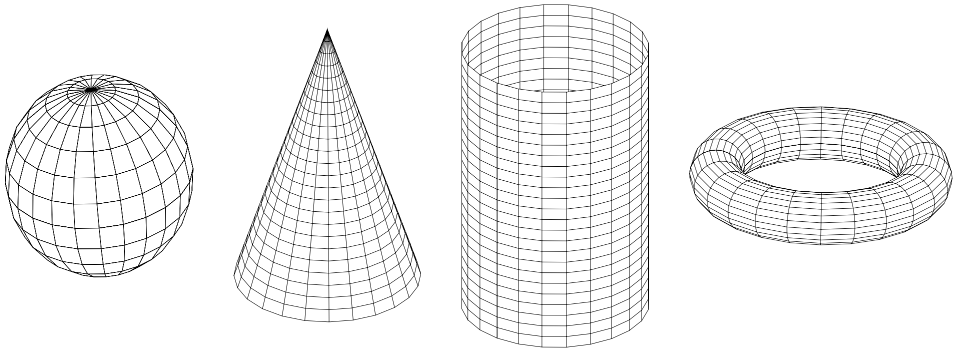


Abbildung 4.2.: Koordinatenlinien der Parametrisierungen

4.3.2. Kugel

Sei $\vec{C}$ der Mittelpunkt der Kugel K und r ihr Radius. Sei $\vec{p}$ die Richtung des Pols und $\vec{q}$ die Richtung des (u, v)-Ursprungs. Sagt das Richtungsflag der Klasse, dass die Normalenrichtung gekippt ist, dann sei $\vec{s} := \vec{q} \times \vec{p}$, sonst $\vec{s} := \vec{p} \times \vec{q}$. Die Kugel K wird durch

$$\vec{F}_K(u, v) = \vec{C} + r \sin u \cdot \vec{p} + r \cos u \cdot (\cos v \cdot \vec{q} + \sin v \cdot \vec{s}), \quad u \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], v \in (-\pi, \pi] \tag{4.5}$$

parametrisiert. Anschaulich entspricht diese Parametrisierung Längen- und Breitengraden auf der Kugel (siehe auch Abbildung 4.2). Die Breitengrade entsprechen dabei u und die Längengrade v.

Um die Inverse $\vec{F}_K^{-1}(\vec{Q})$ von $\vec{F}_K$ zu berechnen, muss man die Lösung von

$$r \sin u \cdot \vec{p} + r \cos u \cdot \cos v \cdot \vec{q} + r \cos u \cdot \sin v \cdot \vec{s} = \vec{Q} - \vec{C} \tag{4.6}$$

bestimmen. Man beachte, dass $\vec{p}$, $\vec{q}$ und $\vec{s}$ jeweils senkrecht zueinander stehen und normiert sind. Multipliziert man also 4.6 mit $\vec{p}$, $\vec{q}$ bzw. $\vec{s}$, so erhält man die Gleichungen

$$\sin u = \frac{1}{r} \cdot \vec{p} \cdot (\vec{Q} - \vec{C}) \tag{4.7}$$

$$\cos u \cdot \cos v = \frac{1}{r} \cdot \vec{q} \cdot (\vec{Q} - \vec{C}) \tag{4.8}$$

$$\cos u \cdot \sin v = \frac{1}{r} \cdot \vec{s} \cdot (\vec{Q} - \vec{C}). \tag{4.9}$$

Ist $\vec{Q}$ ein Punkt auf der Kugeloberfläche, dann kann man aus 4.7 u bestimmen mit $u \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Gilt $u \neq \pm\frac{\pi}{2}$, so kann man mit Hilfe von 4.8 und 4.9 v für $v \in (-\pi, \pi]$

berechnen; andernfalls ist der Punkt $\vec{Q}$ der Nord- oder Südpol der Kugel, und die Wahl von v ist beliebig.

4.3.3. Kegel und Zylinder

Sei $\vec{C}$ der Mittelpunkt der Basisellipse des Kegels bzw. des Zylinders K. Seien $\vec{a}$ und $\vec{b}$ die große und kleine Halbachse der Basisellipse. Sei $\vec{n}$ der Normalenvektor der Ebene, in der die Basisellipse liegt (um 180°gekippt, falls das Richtungsflag der Klasse sagt, dass die Normalenrichtung gekippt ist). Seien s bzw. c der Sinus- bzw. Kosinuswert des halben Hauptwinkels. Sei r die Länge der großen Halbachse. Dann wird K durch

$$\vec{F}_K(u,v) = \vec{C} + (1 + s \cdot u) \cdot \underbrace{(\cos v \cdot \vec{a} + \sin v \cdot \vec{b})}_{\substack{\text{Basisellipse bei } v \\ \text{ausgewertet und} \\ \text{Mittelpunkt abgezogen}}} + c \cdot r \cdot u \cdot \vec{n}, \quad u \in \mathbb{R}, v \in (-\pi, \pi] \tag{4.10}$$

parametrisiert. Die anschauliche Interpretation des Parameters v ist identisch mit der Anschauung, die schon bei der Beschreibung der Ellipse angegeben wurde (als Ellipse wird die Basisellipse verwendet). Der Parameter u gibt die Länge an, die man dann noch entlang der Höhe zu gehen hat (siehe auch Abbildung 4.2).

Um die Inverse $\vec{F}_K^{-1}(\vec{Q})$ von $\vec{F}_K$ zu berechnen muss man die Lösung von

$$(1 + s \cdot u) \cdot (\cos v \cdot \vec{a} + \sin v \cdot \vec{b}) + c \cdot r \cdot u \cdot \vec{n} = \vec{Q} - \vec{C} \tag{4.11}$$

bestimmen. Hierfür nutzt man aus, dass $\vec{n}$, $\vec{a}$ und $\vec{b}$ jeweils senkrecht sind und dass $\vec{n}$ normiert ist. Multipliziert man Gleichung 4.11 mit $\vec{n}$, $\vec{a}$ bzw. $\vec{b}$, so erhält man

$$\begin{aligned} u &= \frac{\vec{n} \cdot (\vec{Q} - \vec{C})}{c \cdot r} \\ \cos v &= \frac{\vec{a} \cdot (\vec{Q} - \vec{C})}{(1 + s \cdot u)(\vec{a} \cdot \vec{a})} \\ \sin v &= \frac{\vec{b} \cdot (\vec{Q} - \vec{C})}{(1 + s \cdot u)(\vec{b} \cdot \vec{b})}. \end{aligned}$$

Liegt $\vec{Q}$ auf K, so besitzt dieses Gleichungssystem eine Lösung, die für $v \in (-\pi, \pi]$ eindeutig ist, falls $\vec{Q}$ nicht die Kegelspitze ist.

4.3.4. Torus

Sei $\vec{C}$ der Mittelpunkt des Torus T. Sei $\vec{n}$ die Normale der Ebene, in der der Hauptkreis liegt. Sei $\vec{q}$ die Richtung des (u,v)-Ursprungs. Sagt das Richtungsflag der Klasse, dass die Normalenrichtung gekippt ist, dann sei $\vec{p} := \vec{q} \times \vec{n}$, sonst $\vec{p} := \vec{n} \times \vec{q}$. Sei r der Wert des Nebenradius und R der Wert des Hauptradius. Der Torus T wird durch

$$\vec{F}_T(u,v) = \vec{C} + r \sin u \cdot \vec{n} + (R + r\cos u) \cdot (\cos v \cdot \vec{q} + \sin v \cdot \vec{p}), \quad u, v \in (-\pi, \pi] \tag{4.12}$$

parametrisiert. Anschaulich parametrisiert u die „Nebenkreise“. Durch v werden die Kreise auf dem Torusmantel parametrisiert, die in zur Ebene des Hauptkreises parallelen Ebenen liegen (siehe Abbildung 4.2).

Um die Inverse $\vec{F}_T^{-1}(\vec{Q})$ von $\vec{F}_T$ zu berechnen muss man die Lösung von

$$r \sin u \cdot \vec{n} + (R + r\cos u) \cdot \cos v \cdot \vec{q} + (R + r\cos u) \cdot \sin v \cdot \vec{p} = \vec{Q} - \vec{C} \tag{4.13}$$

bestimmen. Hierzu beachte man, dass $\vec{n}$, $\vec{p}$ und $\vec{q}$ ein Orthonormalsystem bilden. Multipliziert man Gleichung 4.13 mit $\vec{n}$, $\vec{p}$ bzw. $\vec{q}$, so erhält man

$$r \cdot \sin u = \vec{n} \cdot (\vec{Q} - \vec{C}) \tag{4.14}$$

$$(R + r\cos u) \cdot \sin v = \vec{p} \cdot (\vec{Q} - \vec{C}) \tag{4.15}$$

$$(R + r\cos u) \cdot \cos v = \vec{q} \cdot (\vec{Q} - \vec{C}). \tag{4.16}$$

Es gilt $R > r$ und somit ist $(R + r\cos u) > 0$. Deswegen kann man aus 4.15 und 4.16 mit der arctan2-Funktion[3] v berechnen: $v = \arctan2(\vec{p} \cdot (\vec{Q} - \vec{C}), \vec{q} \cdot (\vec{Q} - \vec{C}))$ mit $v \in (-\pi, \pi]$.

Anschließend muss man noch u mit $u \in (-\pi, \pi]$ bestimmen. Hierzu kann man die Gleichungen 4.14 und eine der beiden Gleichungen 4.15 bzw. 4.16 verwenden. Die Wahl, ob man nun Gleichung 4.15 oder 4.16 verwendet, hängt davon ab, ob $|\sin v| > |\cos v|$ ist, da man durch diesen Wert dividieren muss und er nahe bei Null liegen kann. Aus Gründen der numerischen Stabilität ist es deswegen günstiger den betragsmäßig größeren Wert zu nehmen.

[3] Die Funktion $\arctan2(a, b)$ berechnet den Arkustangens von $\frac{a}{b}$ unter Berücksichtigung des korrekten Vorzeichens (dieses wird aus den Vorzeichen von a und b bestimmt). Ist $\lambda > 0$, so gilt

$$\arctan2(\lambda a, \lambda b) = \arctan2(a, b).$$

Besitzt das Gleichungssystem

$$\sin x = a \quad \cos x = b$$

im Bereich $x \in (-\pi, \pi]$ eine Lösung, so berechnet $\arctan2(a, b)$ diese.

5. Approximationsalgorithmus

In diesem Kapitel wird beschrieben, wie man Körper mit gekrümmten Oberflächen approximiert. Es wird davon ausgegangen, dass die Objektdaten in einer Randdarstellung vorliegen, die grob dem Aufbau des ACIS-Formats entsprechen (vergleiche auch Abschnitt 3).

Zunächst werden die an die Approximation gestellten Eigenschaften dargestellt und erläutert. Aus diesen Eigenschaften ergeben sich mehrere mögliche Vorgehensweisen zur Approximation, die kurz vorgestellt und diskutiert werden, um die Entscheidung für die von mir gewählte Vorgehensweise zu begründen. Diese wird dann genauer ausgeführt – zunächst der Gesamtalgorithmus nur grob und anschließend werden wichtige Details noch genauer beschrieben.

5.1. Aufgabenstellung und Abgrenzung

Für physikalisch basierte Simulationen ist es wichtig, dass die Modelldaten feste Körper repräsentieren. Insbesondere bedeutet dies, dass es keine einzelnen Flächen geben darf und dass sich Flächen nicht durchdringen.

Werden Körper mit gekrümmten Flächen durch Polygone approximiert, so müssen diese Eigenschaften erhalten bleiben. Betrachtet man nur das Problem, dass man einzelne Flächen approximieren will, so kann man leicht erreichen[1], dass keine Lücken im Inneren der Fläche entstehen und dass keine Fläche eine andere durchdringt (sofern dies auch schon bei der Originalfläche der Fall war).

Die eigentliche Schwierigkeit kommt somit dadurch zustande, dass man diese Eigenschaften nicht nur für einzelne Flächen, sondern für den gesamten Körper fordern muss. D.h. die Flächen müssen unter Berücksichtigung topologischer Beziehungen zu anderen Flächen und Kanten approximiert werden, um Lücken und Durchdringungen zu vermeiden.

[1] Führt man die Approximation z.B. im Parameterraum der Fläche durch, so muss man nur darauf achten, dass dort keine Lücken entstehen. Dies leisten z.B. wohlbekannte Triangulierungsalgorithmen.

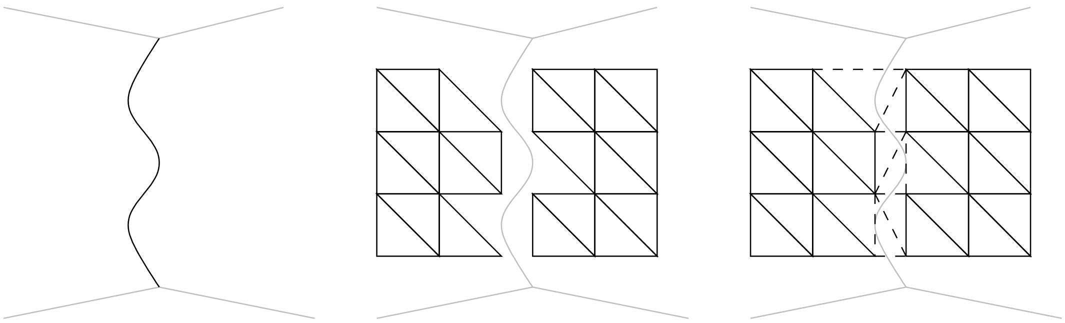


Abbildung 5.1.: Die einzelnen Schritte zur Approximation bei der flächenorientierten Vorgehensweise.

5.2. Mögliche Lösungsansätze

Aus den bisherigen Überlegungen kristallisieren sich zwei mögliche Vorgehensweisen heraus: zum einen kann man zunächst alle Flächen unabhängig voneinander approximieren. Dabei können (und werden) Inkonsistenzen entstehen, die in einem zweiten Schritt beseitigt werden müssen. Stellt der Algorithmus zur Flächenapproximation sicher, dass das Ergebnis „innerhalb“ der Begrenzungskanten liegt, so können keine Durchdringungen in der Nähe der Kanten auftreten. D.h. man kann sich darauf einschränken, nur eventuell entstandene Lücken zu beseitigen. Diese Vorgehensweise wurde z.B. in [SH92] benutzt. Im Nachfolgenden wird sie immer als *flächenorientierte* Vorgehensweise bezeichnet, da der Körper ausgehend von der Fläche approximiert wird. Die einzelnen Schritte dieser Vorgehensweise sind auch in Abbildung 5.1 skizziert.

Die andere Vorgehensweise vermeidet dieses Verbinden von Lücken, indem man zuerst alle Begrenzungskanten approximiert und dann erst in einem zweiten Schritt die Flächen. Dabei muss darauf geachtet werden, dass die approximierten Begrenzungskanten die Approximation der Fläche auch wirklich begrenzen. Diese Vorgehensweise wurde z.B. in [Cui98] benutzt. Im Nachfolgenden wird sie immer als *kantenorientierte* Vorgehensweise bezeichnet, da zunächst die Kanten approximiert werden und davon ausgehend die Flächen (und somit der ganze Körper).

Es gibt viele Möglichkeiten, wie man den letzten Schritt (d.h. die Approximation der Fläche) hierbei realisieren kann. In Abbildung 5.2 sind zwei Möglichkeiten skizziert:

- Ähnlich wie bei der flächenorientierten Vorgehensweise approximiert man die

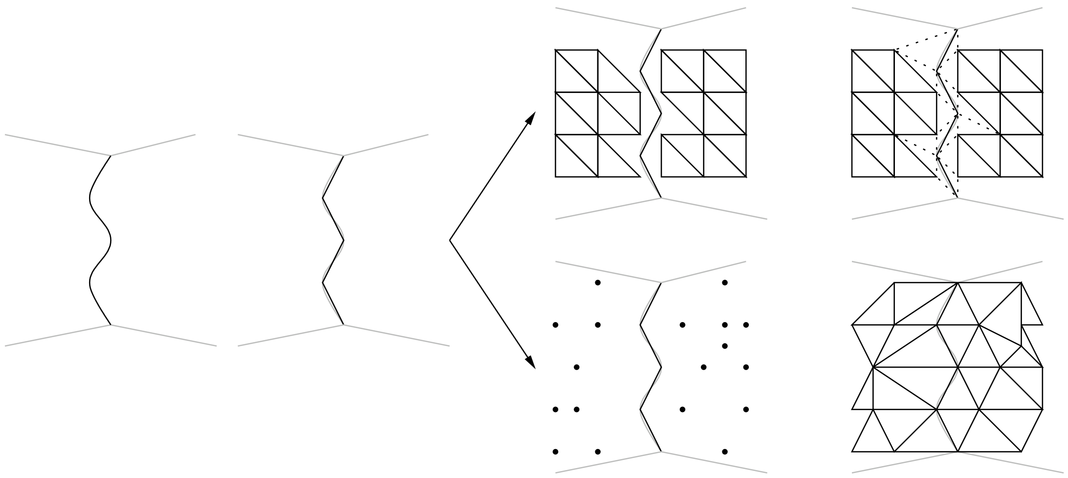


Abbildung 5.2.: Die einzelnen Schritte zur Approximation bei der kantenorientierten Vorgehensweise; die beiden Möglichkeiten zur Approximation der Fläche sind skizziert.

Flächen wieder unabhängig voneinander und muss dann die entstanden Lücken schließen. Der Unterschied zur flächenorientierten Vorgehensweise besteht darin, dass man die Fläche mit der approximierten Kante verbinden muss, wozu man die topologischen Informationen der Originaldaten sehr gut ausnutzen kann.

- Die zweite Möglichkeit lässt Lücken gar nicht erst entstehen. Die eigentliche Approximation findet im Parameterraum der Oberfläche statt. Die approximierten Begrenzungskanten bilden darin ein Polygon (gegebenenfalls mit Löchern). Trianguliert man das Polygon unter Einfügung innerer Punkte[2], so ist die Hoffnung, dass daraus eine Triangulierung im Objektraum ohne Durchdringung entsteht.

Die flächenorientierte Vorgehensweise besitzt gegenüber der kantenorientierten Vorgehensweise einen gravierenden Nachteil: man kann nämlich die topologischen Beziehungen adjazenter Flächen, die im ACIS-Format vorhanden sind, nicht ausnutzen. Vielmehr muss man über räumliche Abstandsmaße versuchen, Punkte zu finden, die nahe beieinander liegen und deswegen mit einer Kante verbunden werden müssen.

Da entsprechende topologische Informationen schon vorhanden waren, aber nicht ausgenutzt werden können, ist dies ein klarer Rückschritt. Die erste Möglichkeit der

[2] Es gibt mehrere Möglichkeiten, wie man diese Punkte bestimmen kann. Im Abschnitt 5.2.1 wird dies näher beschrieben.

kantenorientierten Vorgehensweise beseitigt gerade dieses Manko, da hier mit der approximierten Kante die topologische Information wieder hereinkommt.

Für dieses Verbinden der approximierten Fläche mit der approximierten Kante ist ein Triangulierungsalgorithmus notwendig. Deswegen ist die vorherige Approximation der Fläche nicht wirklich eine Arbeitserleichterung gegenüber der zweiten Möglichkeit; hier benötigt man auch einen Triangulierungsalgorithmus. Bei der Triangulation sind dann zwar mehr Punkte beteiligt, dafür fällt dann die Vorbereitung der Flächenapproximation weg. Aus diesen Gründen habe ich mich in meiner Arbeit für diese Methode entschieden, die nachfolgend auch nur noch erläutert wird.

5.2.1. Auswahl innerer Punkte

Es wurde schon erwähnt, dass bei der kantenorientierten Vorgehensweise (bei der zweiten Möglichkeit) innere Punkte für die Approximation ausgewählt werden müssen. In diesem Abschnitt werden kurz einige Möglichkeiten hierzu erläutert.

Es ist klar, dass die Punktauswahl so erfolgen muss, dass die daraus resultierende Approximation gewissen Genauigkeitskriterien genügt. Cuillière unterscheidet in [Cui98] grob zwei Möglichkeiten für Approximations-Verfahren:

- *A posteriori* Verfahren, die iterativ arbeiten und zunächst eine grobe Approximation erzeugen. Diese Approximation wird dann durch eine (lokale) Fehlerabschätzung bewertet und die Stellen, die noch zu ungenau sind, werden verfeinert.

- *A priori* Verfahren, die als erstes eine Fehlerabschätzung machen, aus der dann eine Punktverteilung abgeleitet wird. Die Fehlerabschätzung muss so sein, dass gewährleistet ist, dass die Genauigkeitskriterien eingehalten werden. Dabei muss die Punktverteilung nicht unbedingt gleichförmig sein, sondern kann durchaus lokal variieren.

Die a posteriori Verfahren haben den Vorteil, dass sie im Allgemeinen mit weniger Punkten die gewünschten Genauigkeitskriterien erreichen, da bei a priori Verfahren die Fehlerabschätzung nicht so genau ist.

Der Vorteil von a priori Verfahren ist allerdings der, dass man nur einmal die Approximation durchführen muss; man erspart sich die Iteration und erhält so einen Laufzeitvorteil. Beschränkt man sich ferner auf Oberflächentypen, die keine starken Krümmungsänderungen aufweisen, so sind gute a priori Verfahren nur unwesentlich schlechter als a posteriori Verfahren, da hierbei ein lokales Verfeinern kaum Vorteile gegenüber einer globalen Approximation besitzt.

Da die in dieser Arbeit betrachteten Oberflächentypen diese günstigen Krümmungseigenschaften besitzen, habe ich mich für ein a priori Verfahren entschieden.

Bisher wurde nur sehr grob über die Möglichkeiten der Punktauswahl diskutiert. Wie eine konkrete Auswahl aussieht, ist aber sehr entscheidend für das Ergebnis. So ist es durchaus möglich, dass auch eine a priori Approximation sich um unterschiedliche Krümmungen kümmert und an Stellen einer großen Krümmung eine größere Punktdichte erzeugt, um eine entsprechende Detailtreue zu gewährleisten.

Die Arbeit von Cuillière [Cui98] beschäftigt sich zu einem großen Teil mit der Frage, wie man eine a priori Punktverteilung für Oberflächen mit großen Krümmungsänderungen möglichst optimal erstellt. Im SILVIA-Projekt benutzen wir aber nur Oberflächentypen, bei denen dies kein Problem ist[3]. Deswegen ist eine „regelmäßige" Punktverteilung für unsere Zwecke ausreichend.

Diese „regelmäßige" Punktverteilung soll dabei für die Punkte im Parameterraum gelten. „Regelmäßig" soll hierbei zu „wohlgeformten" Dreiecken (d.h. möglichst gleichseitige) führen. Wie man später sehen wird, resultieren daraus gute Ergebnisse für die Approximation im Modellraum.

Natürlich möchte man auch die Genauigkeit der Approximation steuern. Dazu muss eine größere Punktdichte erzeugt werden. Dies kann man tun, indem man eine Beschränkung der Größe (d.h. Flächeninhalt) der Dreiecke vorgibt. Dadurch müssen kleinere Dreiecke erzeugt werden und die Punktdichte nimmt zu.

Diese beiden Aspekte wurden realisiert, indem die Punkte mit einem Triangulierungsalgorithmus erzeugt werden, der es erlaubt gewisse Qualitätskriterien anzugeben, die in der Triangulierung (sofern dies möglich ist) eingehalten werden. Hierzu fügt der Triangulierungsalgorithmus Punkte im Inneren des zu triangulierenden Polygons ein[4]. Die konkrete Implementierung benutzt zum einen als Qualitätskriterium eine Beschränkung des minimalen Winkels auf mindestens $20°$. Hierdurch soll die Wohlgeformtheit der Dreiecke sichergestellt werden. Das andere Qualitätskriterium ist eine Beschränkung des Flächeninhalts der Dreiecke. Dies wird aus der gewünschten Genauigkeit der Approximation bestimmt und legt die Feinheit der Approximation fest.

[3] Der Kegel bildet hier die Ausnahme. Er weist ja größere Änderungen im Krümmungsverhalten auf. An späterer Stelle wird darauf eingegangen, warum die Ergebnisse, die man hier erzielt, nicht allzu schlecht werden.

[4] Man kann die Definition einer Delaunay Triangulierung auf Polygone erweitern. Diese Erweiterung wird „Constrained Delaunay Triangulation" (CDT) genannt (siehe z.B. [Che87] oder [FP94]). Es gibt einige Arbeiten, die CDT-Algorithmen entwickeln, die eine Qualitätsverbesserung der Triangulierung nach verschiedenen Kriterien vornehmen (z.B. eine untere Schranke für den minimalen Winkel der Dreiecke einhalten). Siehe hierzu vor allem [Rup95, She96].

5.3. Der Algorithmus

In diesem Abschnitt wird zunächst der Algorithmus grob skizziert und anschließend werden wichtige Details näher betrachtet.

Um die Beschreibung des Algorithmus kompakt und verständlich zu halten, ist es notwendig einige Notationen einzuführen, die einen Teil der ACIS-Datenstruktur formalisieren. Es wurde darauf verzichtet, dies für die ganze Datenstruktur zu machen, da es für die Beschreibung des Algorithmus nicht notwendig ist und diese ansonsten unnötig kompliziert gemacht hätte.

5.3.1. Notationen

Körper

Ein Körper $\mathfrak{B}$ ist ein Tupel $\mathfrak{B} = (\mathfrak{V}_\mathfrak{B}, \mathfrak{E}_\mathfrak{B}, \mathfrak{L}_\mathfrak{B}, \mathfrak{F}_\mathfrak{B})$, bestehend aus einer Menge von Knoten $\mathfrak{V}_\mathfrak{B}$, Kanten $\mathfrak{E}_\mathfrak{B}$, Schleifen $\mathfrak{L}_\mathfrak{B}$ und Flächen $\mathfrak{F}_\mathfrak{B}$.

$\mathfrak{V}_\mathfrak{B} \subset \mathbb{R}^3$ bezeichnet die Menge aller Knoten des Körpers $\mathfrak{B}$. Die Knoten werden hierbei mit den zugehörigen Punkten einfach identifiziert[5].

Die Menge aller Kanten des Körpers $\mathfrak{B}$ wird mit $\mathfrak{E}_\mathfrak{B}$ bezeichnet. Ein Element aus $\mathfrak{E}_\mathfrak{B}$ ist ein Tripel $(\vec{r}, \nu_s, \nu_e)$, wobei $\vec{r} \in \mathcal{R}$ eine Parametrisierung der zur Kante gehörigen Kurve ist und $\nu_s, \nu_e \in \mathfrak{V}_\mathfrak{B}$ Start- und Endpunkte der Kante sind (die Kante ist somit orientiert, vergleiche hierzu auch Abschnitt 3.1).

Auf eine Formalisierung der Halbkanten wird verzichtet, da sie sich von einer Kante nur durch die Orientierung unterscheiden und dies im Algorithmus nur an einer Stelle gebraucht wird. An entsprechender Stelle wird dann kurz darauf eingegangen.

Aus diesem Grund wird eine Schleife als eine Folge von Kanten beschrieben – und nicht Halbkanten, wie es das ACIS-Format vorsieht. Die Menge aller Schleifen $\mathfrak{L}_\mathfrak{B}$ von $\mathfrak{B}$ ist somit $\mathfrak{L}_\mathfrak{B} \subset \bigcup\limits_{i=0}^{\infty} \mathfrak{E}_\mathfrak{B}^i$.

Die Menge der Flächen des Körpers $\mathfrak{B}$ wird mit $\mathfrak{F}_\mathfrak{B}$ bezeichnet. Ein Element aus $\mathfrak{F}_\mathfrak{B}$ ist ein Tupel $(\vec{F}, L)$, wobei $\vec{F} \in \mathcal{F}$ eine Parametrisierung der zur Fläche gehörigen Oberfläche und $L \subseteq \mathfrak{L}_\mathfrak{B}$ die Menge der Schleifen ist, die die Fläche begrenzen.

Approximation

Eine wichtige Operation des Algorithmus ist das Approximieren einer Kante durch einen Kantenzug. Dabei werden neue Knoten und Kanten erzeugt. Die Approximation

[5] Im ACIS-Format ist es möglich, dass zwei verschiedene Knoten den gleichen Punkt repräsentieren. Dies ist mit diesem Formalismus nicht möglich; da dies aber an keiner Stelle im Algorithmus relevant ist, wird der einfachere Formalismus verwendet.

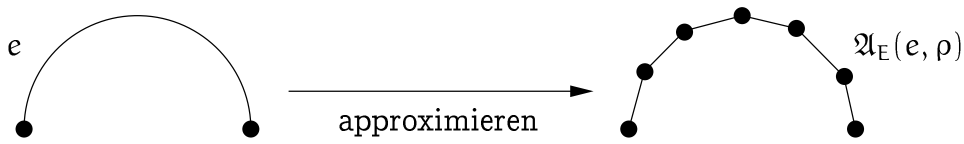


Abbildung 5.3.: Beispiel, wie eine Kante approximiert wird.

benötigt neben der Kante, die approximiert werden soll, noch eine Genauigkeitsangabe als Parameter. Insgesamt ist die Kantenapproximation $\mathfrak{A}_E$ also eine Funktion aus der Menge aller Kanten und $\mathbb{R}$ in die Menge aller möglichen Knoten und Kanten. Einer Kante $e = (\vec{r}, \nu_s, \nu_e) \in \mathfrak{E}_\mathfrak{B}$ und $\rho \in \mathbb{R}$ wird also ein Tupel $\mathfrak{A}_E(e, \rho) = (\mathfrak{V}_e, \mathfrak{E}_e)$ zugeordnet, mit $\nu_s, \nu_e \in \mathfrak{V}_e$ und zu jedem $\hat{\nu} \in \mathfrak{V}_e$ gibt es ein $\hat{e} = (\vec{r}_{\hat{e}}, \hat{\nu}_s, \hat{\nu}_e)$ mit $\hat{\nu}_s = \hat{\nu}$ oder $\hat{\nu}_e = \hat{\nu}$.

Analog definiert man die Approximation für Flächen $\mathfrak{A}_F$. Sie ist eine Funktion aus der Menge aller Flächen und $\mathbb{R}$ in die Menge aller möglichen Flächen (hierzu benötigt man natürlich auch noch die entsprechenden Schleifen, Kanten und Knoten, die die Flächen begrenzen); d.h. es gilt $\mathfrak{A}_F(f, \rho) = (\mathfrak{V}_f, \mathfrak{E}_f, \mathfrak{L}_f, \mathfrak{F}_f)$, wobei $\mathfrak{V}_f$, $\mathfrak{E}_f$, $\mathfrak{L}_f$ und $\mathfrak{F}_f$ natürlich nicht beliebig sein werden, sondern vom Algorithmus sinnvoll berechnet werden[6].

Beide Definitionen sind im Moment noch sehr allgemein und viele Abbildungen erfüllen die gestellten Anforderungen. In den nächsten Abschnitten werden sie aber nach und nach mit Leben gefüllt.

5.3.2. Der Algorithmus grob

Mit diesen Notationen kann man jetzt den Algorithmus näher erläutern. Zunächst werden die groben Komponenten und ihr Zusammenspiel dargestellt.

Als erstes approximiert der Algorithmus alle Kanten; d.h. für jedes $e \in \mathfrak{E}_\mathfrak{B}$ wird $\mathfrak{A}_E(e, \rho_e)$ berechnet[7].

Im nächsten Schritt werden alle Flächen nacheinander approximiert. Dies geschieht im Parameterraum der jeweiligen Oberfläche mittels einer constraint-basierten Delaunay-Triangulierung (CDT) mit Qualitätsverbesserung. In [She96] ist ein solcher Algorithmus beschrieben. Insbesondere existiert davon auch eine Implementierung,

[6] Dies wird nicht formal durchgeführt. Bei der Beschreibung des Algorithmus wird man dann sehen, wie dies berechnet wird und was „sinnvoll“ ist; z.B. sollten die Approximationen der Schleifen als Kanten vorkommen, etc.

[7] Wie man später sehen wird, kann die Genauigkeit ρ_e von verschiedenen Faktoren abhängen. Sie ist also nicht zwangsläufig für jede Kante gleich.

die unter dem Namen *Triangle* als freie C-Bibliothek erhätlich ist. Diese Bibliothek wurde für die Implementierung des in dieser Arbeit vorgestellten Algorithmus verwendet. Auf einige Besonderheiten diesbezüglich wird später noch genauer eingegangen.

Will man die Fläche $f = (\vec{F}, L)$ approximieren, so bedeutet dies konkret, dass man die Approximationen $\mathfrak{A}_L$ aller Kanten der begrenzenden Schleifen aus L betrachten muss. Es gilt: $\mathfrak{A}_L := \bigcup_{e \in \mathfrak{E}_L} \mathfrak{A}_E(e, \rho_e)$. Hierbei bedeutet die Vereinigung von Tupeln, dass die Vereinigung jeweils komponentenweise durchgeführt wird; d.h. $(\mathfrak{V}_{e_1}, \mathfrak{E}_{e_1}) \cup (\mathfrak{V}_{e_2}, \mathfrak{E}_{e_2}) := (\mathfrak{V}_{e_1} \cup \mathfrak{V}_{e_2}, \mathfrak{E}_{e_1} \cup \mathfrak{E}_{e_2})$. $\mathfrak{E}_L$ ist dabei die Menge aller Kanten der Schleifen aus L[8].

Falls $L \neq \emptyset$, bilden diese Approximationen $\mathfrak{A}_L$ im Paramterraum ein Polygon $p_L \subset \mathbb{R}^2 \times \mathbb{R}^2$. Ein Polygon wird also einfach als Menge seiner Begrenzungskanten aufgefasst. Auf diese Weise kann man auch Polygone mit Löchern spezifizieren. Ist nämlich $\#L > 1$, so besitzt es Löcher. Dabei ist $p_L := \{(\vec{F}^{-1}(\nu_s), \vec{F}^{-1}(\nu_e)) \mid (\vec{r}, \nu_s, \nu_e) \in \mathfrak{E}_{\mathfrak{A}_L}\}$, wobei $(\mathfrak{V}_{\mathfrak{A}_L}, \mathfrak{E}_{\mathfrak{A}_L}) = \mathfrak{A}_L$. Das Polygon p_L dient dann gerade als Eingabe für den CDT-Algorithmus.

Der CDT-Algorithmus bietet die Möglichkeit, zwei Qualitätsmerkmale anzugeben, die die Ausgabe erfüllen muss, sofern die Eingabe dies überhaupt erlaubt. Dies erreicht der Algorithmus durch geschicktes Einfügen innerer Punkte, sogenannter Steiner Punkte.

Das eine Qualitätsmerkmal ist eine Schranke für den minimalen Winkel aller Dreiecke; man kann z.B. angeben, dass kein Winkel kleiner als 20° sein soll. Dieses Merkmal stellt sicher, dass die Dreiecke „möglichst gleichseitig“ sein sollen.

Das andere Qualitätsmerkmal ist der maximale Flächeninhalt, den ein Dreieck annehmen darf. über dieses Merkmal kann man die Genauigkeit der Approximation steuern, denn je kleiner das Dreieck ist, desto kleiner ist die maximale Abweichung von der gekrümmten Oberfläche; d.h. es „passt“ sich besser der gekrümmten Oberfläche an.

Die Triangulierung des Polygons p_L wird nachfolgend mit $p_L^\Delta \subset \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2$ bezeichnet. Dabei entspricht ein Tripel den drei Koordinaten eines Dreiecks.

Mit diesen Notationen ist dann die Approximation $\mathfrak{A}_F(f, \rho)$ der Fläche f

$$\mathfrak{A}_F(f, \rho) := (\mathfrak{V}_f, \mathfrak{E}_f, \mathfrak{L}_f, \mathfrak{F}_f)$$

[8]Genau:

$$\mathfrak{E}_L := \bigcup_{l \in L} \{e \in \mathcal{R} \times \mathfrak{V}_{\mathfrak{B}} \times \mathfrak{V}_{\mathfrak{B}} \mid \text{es gibt } e_1, \ldots, e_n \in \mathcal{R} \times \mathfrak{V}_{\mathfrak{B}} \times \mathfrak{V}_{\mathfrak{B}},\ n \geq 0,$$
$$\text{mit } l = (e_1, \ldots, e_i, e, e_{i+1}, \ldots, e_n)\}.$$

mit

$$\begin{aligned}
\mathfrak{V}_f &:= \{\vec{F}(v) \mid \text{es gibt } v', v'' \in \mathbb{R}^2 \text{ mit } (v, v', v'') \in p_L^\Delta \text{ oder } (v', v, v'') \in p_L^\Delta \\
&\qquad \text{oder } (v', v'', v) \in p_L^\Delta\} \\
\mathfrak{E}_f &:= \{(\vec{g}_{v_s v_e}, \vec{F}(v_s), \vec{F}(v_e)) \mid \text{es gibt } \hat{v} \in \mathbb{R}^2 \text{ mit } (v_s, v_e, \hat{v}) \in p_L^\Delta \text{ oder } (\hat{v}, v_s, v_e) \in p_L^\Delta \\
&\qquad \text{oder } (v_e, \hat{v}, v_s) \in p_L^\Delta\} \\
\mathfrak{L}_f &:= \{ \underbrace{(e_{v_1 v_2}, e_{v_2 v_3}, e_{v_3 v_1})}_{\substack{\text{Schleife, die aus den Kanten} \\ e_{v_i v_j} := (\vec{g}_{v_i v_j}, \vec{F}(v_i), \vec{F}(v_j)) \\ \text{besteht}}} \mid (v_1, v_2, v_3) \in p_L^\Delta\} \\
\mathfrak{F}_f &:= \{(\vec{E}_{v_1 v_2 v_3}, \underbrace{\{(e_{v_1 v_2}, e_{v_2 v_3}, e_{v_3 v_1})\}}_{\substack{\text{Die Fläche ist durch ein} \\ \text{Dreieck begrenzt; d.h. die} \\ \text{Schleife besteht aus den} \\ \text{drei Kanten.}}}) \mid (v_1, v_2, v_3) \in p_L^\Delta\},
\end{aligned}$$

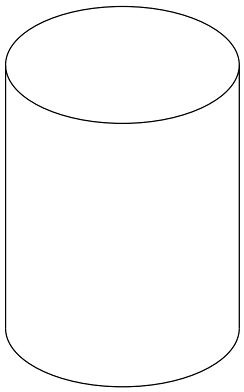
wobei $\vec{g}_{v_1 v_2}$ die Gerade durch die Punkte $\vec{F}(v_1)$ und $\vec{F}(v_2)$ parametrisiert und $\vec{E}_{v_1 v_2 v_3}$ entsprechend die Ebene durch die Punkte $\vec{F}(v_1)$, $\vec{F}(v_2)$ und $\vec{F}(v_3)$ parametrisiert. Im wesentlichen werden also die Punkte der Triangulierung vom Parameterraum in den Objektraum zurück gerechnet.

An dieser Stelle bemerkt man vielleicht, dass der CDT-Algorithmus nur Steiner-Punkte im Inneren des Polygons erzeugen darf und nicht auf dem Rand. Würde nämlich ein zusätzlicher Punkt auf dem Rand eingefügt, so läge dieser bei der Rückrechnung in den Modellraum nicht mehr auf dieser Geraden. Dieser Punkt wird aber bei der Approximation der anderen Fläche, die zu dieser Kante adjazent ist, wahrscheinlich nicht genauso eingefügt bzw. würde in einen anderen Punkt im Modellraum zurückgerechnet und es würde auf diese Weise eine Lücke entstehen (vergleiche auch Abbildung 5.4).

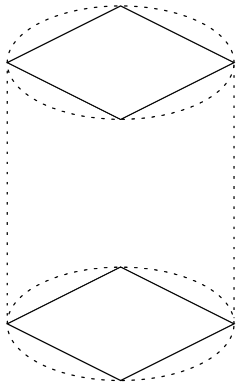
Ist andererseits aber die Kantenapproximation zu ungenau, so können die Qualitätsmerkmale am Rand nicht eingehalten werden. Deswegen ist schon bei der Kantenapproximation darauf zu achten, dass bei der Genauigkeit, mit der man eine Kante approximiert, die adjazenten Flächen berücksichtigt werden. Eine genaue Diskussion findet man in Abschnitt 5.3.4.

Ein weiterer Punkt, der einer genaueren Diskussion bedarf, ist das Problem, dass die Parametrisierungen der jeweiligen Oberflächentypen periodisch in einer (bei Kegel/Zylinder und Kugel) oder in beiden (bei Torus) Variablen sind. Normalerweise werden in der Literatur CDT-Algorithmen nur für die euklidische Ebene dargestellt (und funktionieren nicht für periodische Oberflächen).

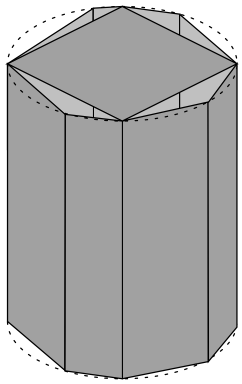
Eine Lösung dieses Problems wäre, die CDT-Algorithmen so abzuändern, dass sie auch auf periodischen Parameterräumen funktionieren (dies ist die Vorgehensweise



(a) Das Objekt, das approximiert werden soll.



(b) Die Approximationen der Kanten.



(c) Das falsch approximierte Objekt.

Abbildung 5.4.: Bei einem Zylinder wurden die Kreiskanten durch Vierecke approximiert. Bei der Approximation der Zylinderfläche wurden allerdings zusätzliche Punkte eingefügt und so entstanden Lücken.

Abbildung 5.5.: Einfügen von Schnittkanten beim Zylinder. Für eine genaue Diskussion aller Oberflächentypen siehe Abschnitt 5.3.3.

von Cuillière in [Cui98]). Die von mir gewählte Vorgehensweise benutzt hingegen einen „standard" CDT-Algorithmus und löst das Problem der Periodizität, indem die Oberfläche gerade an der Stelle bzw. den Stellen, an denen die Parametrisierung von π nach $-\pi$ umschlägt, durch Kanten im Inneren der Fläche zerschnitten wird (siehe hierzu auch Abbildung 5.5; in diesem Beispiel müssen vier neue Knoten und zwei neue Kanten hinzugefügt werden). Eine ausführliche Diskussion findet sich in Abschnitt 5.3.3.

Natürlich muss das Hinzufügen der Schnittkanten *vor* der Kantenapproximation erfolgen, da diese auch approximiert werden müssen.

Algorithmus 1 Objektapproximation

Eingabe: Körper $\mathfrak{B} = (\mathfrak{V}_\mathfrak{B}, \mathfrak{E}_\mathfrak{B}, \mathfrak{L}_\mathfrak{B}, \mathfrak{F}_\mathfrak{B})$ und Genauigkeit ρ
Ausgabe: Approximation $\mathfrak{A}_\mathrm{B}(\mathfrak{B}, \rho)$ des Körpers $\mathfrak{B}$

  // bestimme die Schnittkanten und füge sie hinzu
  $\mathfrak{E}_\mathrm{S} := \emptyset$ *// Initialisierung der Schnittkantenmenge*
  für alle Flächen $\mathrm{f} \in \mathfrak{F}_\mathfrak{B}$ **tue**
    bestimme die Menge der Schnittkanten $\mathfrak{E}_\mathrm{f}$ zur Fläche f
    $\mathfrak{E}_\mathrm{S} := \mathfrak{E}_\mathrm{S} \cup \mathfrak{E}_\mathrm{f}$
    für alle Kanten $e \in \mathfrak{E}_\mathrm{f}$ **tue**
      erweitere $\mathfrak{V}_\mathfrak{B}$ um Start- und Endpunkt von e
      ändere $\mathfrak{E}_\mathfrak{B}$ und $\mathfrak{L}_\mathfrak{B}$, so dass Start- und Endpunkt von e „sinnvoll" die Kanten aus $\mathfrak{E}_\mathfrak{B}$ zerschneiden
    Ende für
  Ende für

  // approximiere alle Kanten (auch die Schnittkanten)
  für alle Kanten $e \in \mathfrak{E}_\mathfrak{B} \cup \mathfrak{E}_\mathrm{S}$ **tue**
    bestimme Genauigkeit ρ_e aus Genauigkeit ρ und der zu e adjazenten Flächen
    berechne $\mathfrak{A}_\mathrm{E}(e, \rho_e)$
  Ende für

  // approximiere die Flächen
  $\mathfrak{A}_\mathrm{B} := \emptyset$
  für alle Flächen $\mathrm{f} \in \mathfrak{F}_\mathfrak{B}$ **tue**
    $\mathfrak{A}_\mathrm{B} := \mathfrak{A}_\mathrm{B} \cup \mathfrak{A}_\mathrm{F}(\mathrm{f}, \rho)$
  Ende für

Insgesamt ergibt sich dann der Algorithmus 1 für die Approximation eines Körpers

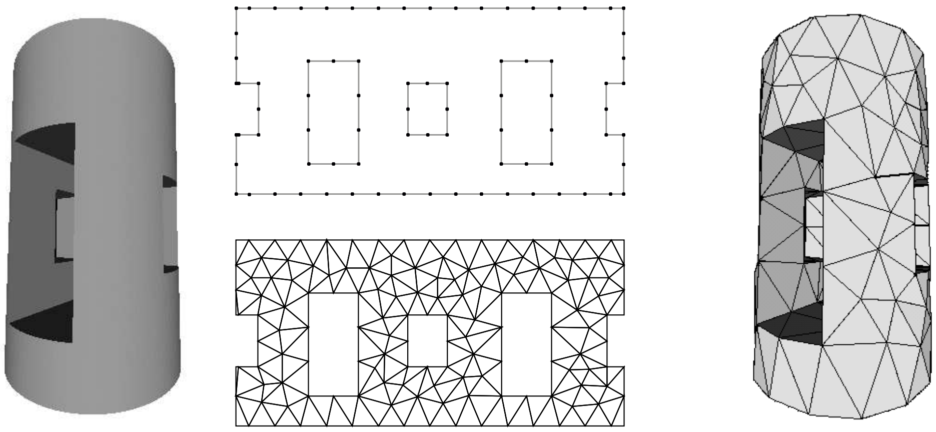


Abbildung 5.6.: Die wichtigen Schritte des Approximationsalgorithmus am Beispiel eines Zylinders mit zwei Durchbohrungen: das Original, das Eingabepolygon im Parameterraum, die Ausgabe des CDT-Algorithmus und die daraus resultierende Approximation.

𝔅. In Abbildung 5.6 sieht man die entscheidenden Schritte des Algorithmus am Beispiel eines Zylinders mit zwei Durchbohrungen: links ist das Original; in der Mitte sieht man das Eingabepolygon für den CDT-Algorithmus und darunter dessen Ausgabe; rechts sieht man dann die daraus resultierende Approximation.

5.3.3. Einfügen von Schnittkanten

Im Algorithmus 1 ist nicht näher erläutert, wie man zu einer Fläche ihre Schnittkantenmenge bestimmt. Dies soll in diesem Abschnitt erfolgen.

Die Schnittkanten werden über Schnittgeraden bestimmt. Die Schnittgeraden sind Geradengleichungen im Parameterraum der Oberfläche. Diese Schnittgeraden sind für jeden Oberflächentyp fest.

Um nun die Schnittkanten zu bestimmen, werden alle Schnittpunkte der Kanten einer Fläche mit den Schnittgeraden der Oberfläche bestimmt. Diese Schnittpunkte zerlegen die Schnittgerade in Strecken- bzw. Halbstreckenstücke. Von diesen Strecken sind diejenigen Schnittkanten, die auf der Fläche liegen.

Dies müssen aber nicht alle Schnittkanten sein. Es kann nämlich der Fall auftreten, dass eine Schnittgerade komplett auf der Fläche liegt, ohne eine Kante zu schneiden[9].

[9] Die Schnittgerade ist eine Gerade im Parameterraum. Bei zyklischen Oberflächen kann daraus im Modellraum ein beschränktes Objekt (z.B. ein Kreis) resultieren.

In diesem Fall muss man die Schnittgerade komplett als Schnittkante hinzufügen (da entsprechende Schnittgeraden im Modellraum beschränkt sind, kann man dies tun).

Als nächstes soll hergeleitet werden, welche Geraden man am günstigsten als Schnittgeraden verwendet. Im vorigen Abschnitt wurde gesagt, dass das Einfügen von Schnittkanten notwendig sei, um die Periodizität in der Parametrisierung aufzuheben. Es gibt aber noch einen zweiten Fall, in dem derartige Schnittkanten hilfreich sind. Betrachtet man nämlich die Parametrisierung der Kugel genauer, so stellt man fest, dass diese Parametrisierung zwei Singularitäten besitzt – den Nord- und Südpol.

Diese Singularitäten führen zu Problemen, weil zum einen die Inverse $\vec{F}^{-1}$ an diesen Stellen nicht eindeutig ist und zum anderen in ihrer Umgebung das Krümmungsverhalten der Koordinatenlinien ungünstig ist. Da die Approximation im Parameterraum durchgeführt wird, ist dieses Krümmungsverhalten aber entscheidend für die Güte der Approximation. Die Kugel würde nämlich in der Nähe der Pole durch kleinere Dreiecke als am Äquator approximiert werden, falls die Dreiecke im Parameterraum gleiche Größe besäßen. Natürlich ist dies unerwünscht (bei der Kegelapproximation wird dieser Effekt hingegen ausgenutzt; Details hierzu werden später in diesem Abschnitt erläutert).

Die Singularitäten bei der Kugel sind aber nur durch die Parametrisierung entstanden[10] und es gibt andere Parametrisierungen, bei denen diese beiden Punkte keine Singularitäten sind. Deswegen ist die Idee, dass man mit Schnittgeraden einfach die Polkappen abschneidet und den Rest, der unproblematisch ist, approximiert. In einem zweiten Schritt werden dann die fehlenden Gebiete mit einer anderen, „um 90° gedrehten" Parametrisierung approximiert (d.h. die Pole der neuen liegen auf dem Äquator der alten Parametrisierung).

Es gibt somit zwei Sorten von Schnittgeraden:

- Schnittgeraden, die die Periodizität der Parametrisierung aufheben.
- Schnittgeraden, die Probleme mit Singularitäten der Parametrisierung umgehen.

Wie diese Schnittgeraden für die einzelnen Oberflächentypen eingesetzt werden, wird jetzt erläutert. Es wird auch jeweils die Gleichung der Schnittgeraden im Parameterraum angegeben.

Für den Fall des Kegels/Zylinders ist die Schnittgerade denkbar einfach[11]. Da die Parametrisierung im Parameter v periodisch ist, kommt man mit einer Schnittgerade

[10] Für die Singularität beim Kegel (d.h. die Kegelspitze) gilt dies nicht. Deswegen muss man sich etwas anderes überlegen, wie man mit dem Problem der Singularität (verschiedene Punkte im Parameterraum werden auf den gleichen Punkt im Modellraum abgebildet) fertig wird. Dies wird in Abschnitt 5.3.6 erläutert.

[11] Die Singularität beim Kegel bleibt zunächst unberücksichtigt.

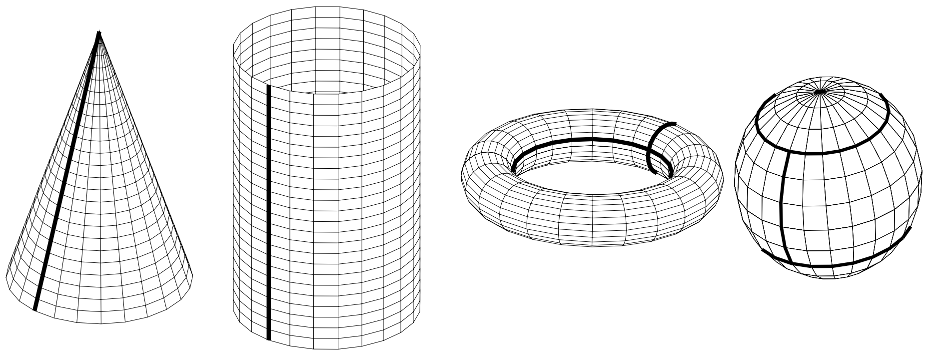


Abbildung 5.7.: Schnittgeraden für die Oberflächentypen

aus, nämlich gerade

$$v = \pi. \tag{5.1}$$

Siehe hierzu auch Abbildung 5.7.

Der Torus ist nicht nur in einem Parameter periodisch, sondern sowohl im Parameter u und v. Deswegen benötigt man zwei Schnittgeraden:

$$u = \pi \tag{5.2}$$
$$v = \pi \tag{5.3}$$

Das Aussehen der Schnittgeraden im Modellraum ist in Abbildung 5.7 eingezeichnet.

Am kompliziertesten sind die Schnittgeraden bei der Kugel. Zum einen ist sie ja im Parameter v periodisch. Zum anderen braucht man noch zusätzlich Schnittgeraden, um die Polkappen abzuschneiden. Insgesamt werden drei Schnittgeraden benötigt:

$$v = \pi \tag{5.4}$$
$$u = \frac{\pi}{4} + 0.1 \tag{5.5}$$
$$u = -\left(\frac{\pi}{4} + 0.1\right) \tag{5.6}$$

Das Aussehen der Schnittgeraden im Modellraum kann man in Abbildung 5.7 näher betrachten. Die beiden Schnittgeraden, um die Polkapppen abzuschneiden, (Gleichung 5.5 und 5.6) sind gerade so gewählt, dass der Winkel zwischen der „Äquatorebene“ und der Strecke durch den Kugelmittelpunkt und einem Punkt auf der Schnittgerade größer als 45° ist (siehe auch Abbildung 5.8). Dadurch ist sichergestellt, dass bei der „um 90° gedrehten“ Parametrisierung keine weiteren Schnittkanten hinzugefügt werden müssen.

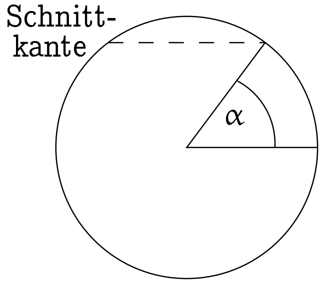


Abbildung 5.8.: Die Kugel von der Seite: der Winkel α für die Schnittgeraden der Polkappen muss größer als 45° sein.

Spezialfälle

Es kann der Fall auftreten, dass ein Schnittpunkt zwischen einer Schnittgeraden und einer Kante schon in der Knotenmenge vorhanden ist. In diesem Fall sollte man den Knoten nicht noch einmal hinzufügen. Genauso kann es passieren, dass eine Schnittkante schon als normale Kante auftritt. In diesem Fall sollte man die Schnittkante und die normale Kante identifizieren. Diese beiden Fälle treten in konkreten Anwendungen sogar sehr häufig auf, da die Objektdaten mit CAD Programmen erstellt wurden und die Schnittgeraden besondere Geraden sind, die auch von dem CAD Programm gerne genutzt werden.

Das große Problem bei diesen zwei Spezialfällen ist, dass es aufgrund numerischer Ungenauigkeiten nicht ganz einfach ist, zu bestimmen, wann zwei Punkte oder zwei Kanten gleich sind.

Bei dem Test, ob zwei Kanten gleich sind, genügt in unserem Fall der Test, ob Anfangs- und Endpunkt gleich sind, ob die Kurven in der gleichen Ebene liegen und ob sie gleich orientiert sind (bei Ellipsen).

Der Test, ob zwei Punkte gleich sind, arbeitet mit Toleranzen; d.h. ist der Abstand zweier Punkte kleiner als eine Toleranzgrenze, so werden diese Punkte als gleich angesehen. Der Wert für diese Toleranzgrenze ist genügend klein anzusetzen, so dass keine verschiedenen Punkte identifiziert werden. Andererseits muss er groß genug sein, damit gleiche Punkte wirklich identifiziert werden. Ein experimentell ermittelter Wert, der für alle gerechneten Beispielobjekte funktioniert hat, ist 10^{-6}.

5.3.4. Bestimmung der Genauigkeit für Kantenapproximation

Bisher wurde nur ganz allgemein von der Genauigkeit geredet, mit der eine Fläche approximiert werden soll. Insbesondere wurde auch dem Parameter ρ noch keine spezielle Bedeutung gegeben. Nun ist es an der Zeit, dies etwas genauer zu untersuchen.

Der Genauigkeitsparameter ρ wird als obere Schranke für die Hausdorff-Distanz[12] zwischen der approximierten Kante und der Originalkante interpretiert. In Abschnitt 5.3.5 wird erläutert, wie man für die einzelnen Oberflächentypen und deren Approximation den entsprechenden Genauigkeitsparameter berechnen kann.

In diesem Abschnitt wird erläutert, wie elliptische Kanten approximiert werden und wie man den dafür notwendigen Genauigkeitsparameter aus dem Genauigkeitsparameter für das gesamte Objekt bestimmt, so dass obige Interpretation eingehalten wird.

Ferner kann es notwendig sein, gerade Kanten zu approximieren (d.h. in diskrete Stücke zu unterteilen). Dies ist für Kanten notwendig, die zu einer kegelförmigen/zylindrischen Fläche adjazent sind. Dabei muss die Unterteilung so gewählt werden, dass bei der Flächenapproximation keine weitere Unterteilung der Kante notwendig ist.

Elliptische Kanten

Die einfachste Methode, die Kanten zu unterteilen, ist vermutlich eine äquidistante Unterteilung im Parameterraum. Dies hat sich in der Praxis auch als sehr praktikabel herausgestellt.

Der Algorithmus zur Approximation elliptischer Kanten geht also so vor, dass vom Startpunkt der Kante aus solange um eine konstante Schrittweite im Parameterraum weitergegangen wird, bis der Endpunkt überschritten wird. Die Punkte, die man so erreicht, werden dann als Punkte der Approximation verwendet (mit Ausnahme des letzten Punktes; für diesen verwendet man den Endpunkt).

Das Entscheidende ist somit, die Schrittweite so zu bestimmen, dass die Interpretation des Genauigkeitsparameters mittels Hausdorff-Distanzen eingehalten wird. Da der Parameter ρ eine obere Schranke angeben soll, genügt es den worst-case Fall zu betrachten. Dieser ist in Abbildung 5.9 dargestellt.

O.B.d.A. wird die ganze Rechnung in der Ebene an einer achsenorientierten Ellipse mit Mittelpunkt im Ursprung durchgeführt. Sei also ϵ die Hausdorff-Distanz, Δt die

[12] Die Hausdorff-Distanz zwischen zwei beschränkten Punktmengen $A, B \subset \mathbb{R}^d$ ist definiert als

$$\delta^H(A,B) := \max\left\{\vec{\delta}^H(A,B), \vec{\delta}^H(B,A)\right\},$$

mit der gerichteten Hausdorff-Distanz

$$\vec{\delta}^H(A,B) := \sup_{x\in A} \inf_{y\in B} \|x-y\|.$$

Siehe hierzu auch [Fuh98] und [ABGW90].

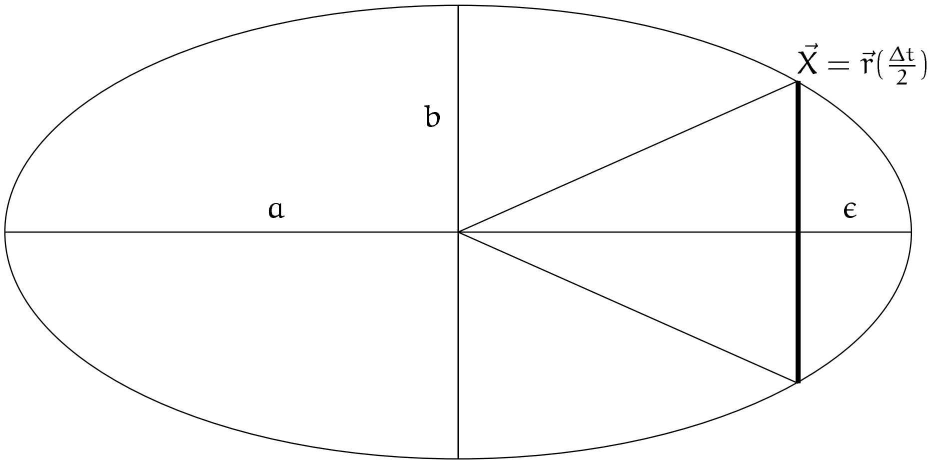


Abbildung 5.9.: Der Fall für den die Hausdorff-Distanz bei der Kantenapproximation am größten ist. Die fette Kante ist die Approximationskante, ϵ ist die Hausdorff-Distanz und Δt ist die Schrittweite.

Schrittweite im Parameterraum (für eine anschauliche Interpretation des Parameters Δt siehe Abschnitt 4.2.2). Mit a bzw. b wird die Länge der großen bzw. kleinen Halbachse bezeichnet. Sei $\vec{X}$ der Punkt wie in Abbildung 5.9.

Es gilt dann

$$\vec{X} = \begin{pmatrix} a \cdot \cos \frac{\Delta t}{2} \\ b \cdot \sin \frac{\Delta t}{2} \end{pmatrix}$$

und somit

$$\epsilon = a - a \cdot \cos \frac{\Delta t}{2}.$$

Will man eine vollständige Ellipse approximieren, so sind hierfür mindestens drei Punkte notwendig. Daraus ergibt sich, dass $\Delta t < \pi$ sein muss. Diese Einschränkung an Δt wollen wir auch immer machen. Deswegen gilt

$$0 < \cos \frac{\Delta t}{2} < 1. \tag{5.7}$$

Da der Genauigkeitsparameter ρ eine obere Schranke für die Hausdorff-Distanz sein soll, gilt insbesondere

$$\begin{aligned} \epsilon &\leq \rho \\ a\left(1 - \cos\frac{\Delta t}{2}\right) &\leq \rho \\ \cos\frac{\Delta t}{2} &\geq 1 - \frac{\rho}{a} \end{aligned}$$

und wegen Gleichung 5.7 gilt sogar

$$\begin{aligned} \cos\frac{\Delta t}{2} &\geq \max\left\{0\,,\, 1 - \frac{\rho}{a}\right\} \\ \Delta t &\leq 2 \cdot \arccos\left(\max\left\{0\,,\, 1 - \frac{\rho}{a}\right\}\right). \end{aligned}$$

Schaut man sich die Bedingung, dass eine vollständige Ellipse durch mindestens drei Segmente approximiert werden soll, genauer an, so erhält man die etwas verschärfte Bedingung, dass $\Delta t \leq 2\pi/3$ sein soll. Insgesamt wird die Schrittweite zur Approximation elliptischer Kanten vom Algorithmus als

$$\Delta t := \min\left\{\frac{2\pi}{3}\,,\, \Delta t'\right\}, \quad \text{mit } \Delta t' := 2 \cdot \arccos\left(\max\left\{0\,,\, 1 - \frac{\rho}{a}\right\}\right) \tag{5.8}$$

berechnet. Damit ist dann die Interpretation der Genauigkeit als obere Schranke für die Hausdorff-Distanz erfüllt, und es ist sichergestellt, dass eine Fläche, die die Gestalt einer ganzen Ellipse hat, als korrekte Fläche approximiert wird.

Die Gleichung 5.8 kann noch vereinfacht werden:

Lemma 5.1 *Sei Δt wie in Gleichung 5.8. Es gilt* $\Delta t = 2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{a}\right\}\right)$.

Beweis Sei Δt wie in Gleichung 5.8. Wir unterscheiden zwei Fälle:

- Fall 1: $\max\left\{0, 1 - \frac{\rho}{a}\right\} = 0$
  D.h. $1 - \frac{\rho}{a} \leq 0$. Daraus folgt $\Delta t' = \pi$ und somit $\Delta t = \frac{2\pi}{3}$.
- Fall 2: $\max\left\{0, 1 - \frac{\rho}{a}\right\} \neq 0$
  D.h. es gilt $\max\left\{0, 1 - \frac{\rho}{a}\right\} = 1 - \frac{\rho}{a}$ und $1 - \frac{\rho}{a} > 0$. Es gilt dann

  $$\begin{aligned} \Delta t = \frac{2\pi}{3} &\iff \Delta t' \geq \frac{2\pi}{3} \\ &\iff 2 \cdot \arccos\left(1 - \frac{\rho}{a}\right) \geq \frac{2\pi}{3} \\ &\iff 1 - \frac{\rho}{a} \leq \cos\frac{\pi}{3} = \frac{1}{2}, \end{aligned}$$

  da $0 \leq 1 - \frac{\rho}{a} \leq 1$ ist.

Aus diesen beiden Fällen erhält man insgesamt, dass $\Delta t = \frac{2\pi}{3}$ genau dann gilt, wenn $1 - \frac{\rho}{a} \le \frac{1}{2}$ ist. Daraus folgt direkt $\Delta t = 2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{a}\right\}\right)$. $\square$

Gerade Kanten

Ist eine nicht gekrümmte Kante zu einer Kegel- oder Zylinderfläche adjazent, so muss diese in Segmente unterteilt werden, da der Triangulierungsalgorithmus keine Punkte auf den Kanten einfügen darf (vgl. Seite 35), diese Punkte aber für eine adäquate Approximation der Fläche notwendig sind.

Diese Unterteilung muss so fein sein, dass die Flächen mit der gewünschten Genauigkeit approximiert werden können. Um gute Ergebnisse zu erzielen, hat es sich bewährt, wenn die Längen der approximierten Kanten und die Längen der Segmente der approxmierten Basisellipse im Parameterraum gleich sind. Siehe hierzu auch Abschnitt 5.3.5.

Man muss also auch zu geraden Kanten aus dem Genauigkeitsparameter ρ einen Genauigkeitsparameter für die Kantenapproximation herleiten.

Die Länge der Basisellipse im Parameterraum ist gerade das Δt aus dem vorigen Abschnitt für diese Ellipse.

Seien $\vec{P}_1 = \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \vec{P}_2 = \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} \in \mathbb{R}^2$ zwei Punkte im Parameterraum, so dass die dazu korrespondierenden Punkte $\vec{F}(\vec{P}_1)$ und $\vec{F}(\vec{P}_2)$ im Modellraum auf der zu approximierenden Kante liegen. Dann gilt $v_1 = v_2$. Im Nachfolgenden wird o.B.d.A. $v_1 = v_2 = 0$ angenommen[13]. D.h. es wird eine Beschränkung der Kantenlänge im Modellraum gesucht, für den Fall, dass die Kantenlänge im Parameterraum durch den Wert Δt beschränkt sein soll. Für das Segment von $\vec{F}(\vec{P}_1)$ nach $\vec{F}(\vec{P}_2)$ bedeutet dies $|u_2 - u_1| \le \Delta t$.

Die Länge l dieses Segments im Modellraum ist

$$\begin{aligned} l &= \left\| \vec{F}(\vec{P}_2) - \vec{F}(\vec{P}_1) \right\| \\ &= \|((1 + s \cdot u_2) \cdot \vec{a} + c \cdot r \cdot u_2 \cdot \vec{n}) - ((1 + s \cdot u_1) \cdot \vec{a} + c \cdot r \cdot u_1 \cdot \vec{n})\| \\ &= \|s \cdot (u_2 - u_1) \cdot \vec{a} + c \cdot r \cdot (u_2 - u_1) \cdot \vec{n}\|, \end{aligned}$$

[13] Für den Fall, dass die Basisellipse ein Kreis ist, ist die Wahl irrelevant. Ansonsten ist dieser Fall der Worst-Case, da die entsprechende Kante an dieser Stelle die größte Steigung hat und somit die Abweichung der Länge im Parameterraum im Vergleich zu der Länge im Modellraum am größten ist.

und da $\vec{a}$ ein Vektor der Länge r ist, der senkrecht auf $\vec{n}$ steht (der normiert ist), gilt

$$\begin{aligned} l &= \sqrt{(s \cdot r \cdot (u_2 - u_1))^2 + (c \cdot r \cdot (u_2 - u_1))^2} \\ &= \sqrt{s^2 + c^2} \cdot r \cdot |u_2 - u_1| . \end{aligned}$$

Da s und c Sinus und Kosinus des gleichen Winkels sind, gilt sogar

$$l = r \cdot |u_2 - u_1| ,$$

und somit ergibt sich die Schranke für die Länge im Modellraum als

$$l \leq r \cdot \Delta t. \tag{5.9}$$

Es kann auch passieren, dass eine gerade Kante zu mehr als nur einem Kegel/Zylinder adjazent ist. In diesem Fall muss die Kante mit einer Längenbeschränkung approximiert werden, die das Minimum der Längenbeschränkungen aller adjazenten Flächen ist. Damit ist dann sichergestellt, dass alle Flächen mit der gewünschten Genauigkeit approximiert werden können. Evtl. wird eine Fläche „zu genau“ approximiert, was aber nicht weiter schlimm ist.

5.3.5. Bestimmung der Genauigkeit für Flächenapproximation

In diesem Abschnitt wird darauf eingegangen, wie man die Genauigkeit für die Flächenapproximation bestimmt. Auf Seite 34 wurde erwähnt, wie die Genauigkeit mittels Qualitätskriterien des CDT-Algorithmus gesteuert werden kann. Die Beschränkung auf Winkel kleiner als 20° ist für diesen Abschnitt unerheblich, da damit nur die Wohlgeformtheit der Dreiecke sichergestellt werden soll. Interessant für die Genauigkeit der Flächenapproximation ist die Beschränkung des Flächeninhalts.

Diese Beschränkung des Flächeninhalts ist somit der Genauigkeitsparameter ρ_F für die Flächenapproximation. Sie muss aus dem Genauigkeitsparameter ρ berechnet werden. Dies muss für jeden Oberflächentyp einzeln geschehen, da der CDT-Algorithmus im Parameterraum der entsprechenden Parametrisierung der Oberfläche arbeitet.

Das allgemeine Vorgehen ist hierbei das folgende: der Genauigkeitsparameter ρ gibt ja eine Beschränkung der Hausdorff-Distanz im Modellraum vor. Daraus resultiert auch eine Beschränkung der Kantenlänge im Modellraum und deswegen auch im Parameterraum. Mit Hilfe dieser Beschränkung μ der Kantenlänge im Parameterraum wird dann der Genauigkeitsparameter ρ_F für diese Fläche bestimmt, indem man sagt, dass „möglichst“ gleichseitige Dreiecke die gewünschte Form sind; also setzt man als

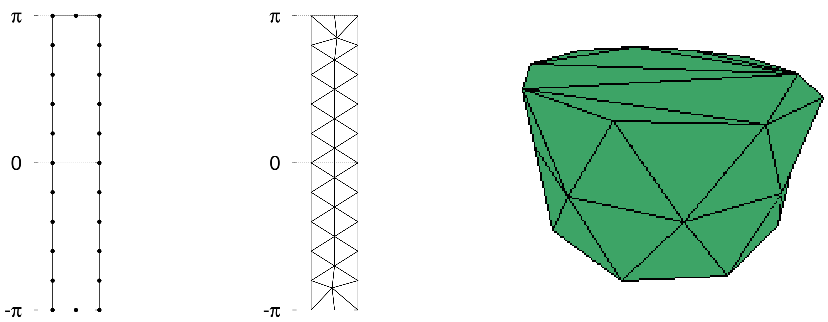


Abbildung 5.10.: Beispielapproximation eines Zylinders.

Beschränkung des Flächeninhalts gerade den Flächeninhalt eines gleichseitigen Dreiecks mit Kantenlänge μ an. Für bekanntes μ gilt somit

$$\rho_F := \frac{\pi}{6}\mu^2. \tag{5.10}$$

In den nächsten Abschnitten wird für jeden Oberflächentyp ein solcher Genauigkeitsparameter hergeleitet. Zu beachten ist, dass keine wirklichen Schranken angegeben werden. Dies ist schon aus dem Grund nicht möglich, da Triangle nicht garantiert, dass die Qualitätsmerkmale eingehalten werden (vergleiche hierzu auch Beispiel 5.3). Deswegen wird heuristisch ein solcher Wert hergeleitet, der keine zu schlechten Ergebnisse liefert. Wie die praktischen Tests und Beispiele zeigen, ist dies auch gelungen.

Kegel/Zylinder

Wie schon im Kapitel über die Berechnung der Genauigkeit für gerade Kanten (Seite 45) erwähnt, ist das Δt aus der Approximation der Basisellipse eine Beschränkung der Kantenlänge im Parameterraum des entsprechenden Kegels/Zylinders. Deswegen ist hier die Berechnung der Genauigkeit besonders einfach:

Sei Δt der Genauigkeitsparameter, mit dem die Basisellipse des Kegels/Zylinders approximiert werden würde. Dann berechnet sich der Genauigkeitsparameter ρ_F für die Flächenapproximation aus

$$\rho_F := \frac{\pi}{6}\Delta t^2. \tag{5.11}$$

Beispiel 5.1 In Abbildung 5.10 ist ein Beispiel für eine Approximation eines Zylinders abgebildet. Man sieht die Eingabe für den CDT-Algorithmus, dessen Ausgabe und die resultierende Approximation im Parameterraum. Bei den Eingabe- und Ausgabebildern ist die u-Achse nach rechts und die v-Achse nach oben aufgetragen. Man

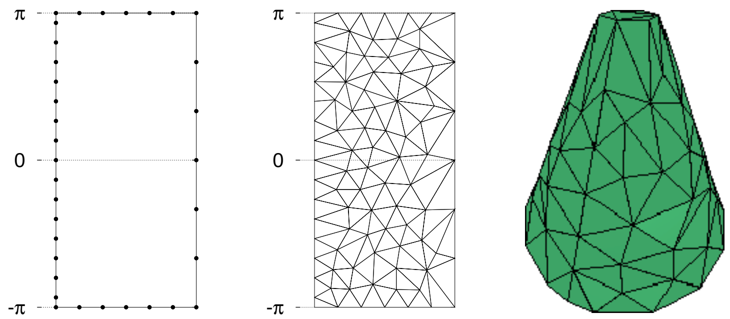


Abbildung 5.11.: Beispielapproximation eines Kegels.

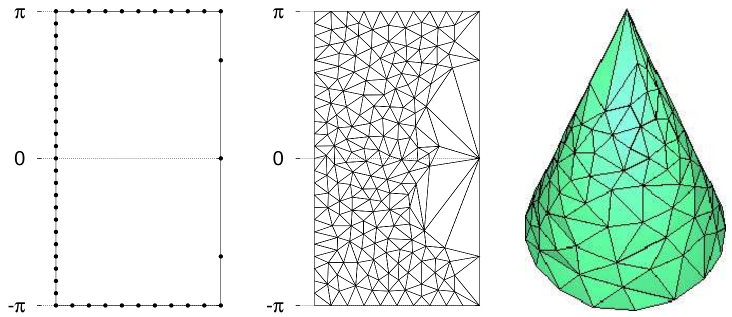


Abbildung 5.12.: Approximation eines Kegels mit Spitze als Beispiel dafür, dass man nicht mit harten Schranken für die Kantenlängen im Parameterraum arbeitet.

sieht ganz gut, dass die Längen der approximierten Kanten in etwa gleich sind. Die Dreiecke, die der CDT-Algorithmus erzeugt, sind – wie schon mehrfach als Wunsch geäußert – „relativ gleichseitig".

Beispiel 5.2 In Abbildung 5.11 sieht man ein entsprechendes Beispiel für die Approximation eine Kegelstumpfs. Auffallend ist hierbei, dass die obere Ellipse mit weniger Segmenten approximiert wurde, da der Genauigkeitsparameter eine Schranke für die Hausdorff-Distanz angibt, und dies ist bei der kleineren Ellipse schon mit weniger Segmenten erreicht. Im Parameterraum allerdings, haben die beiden Ellipsen die gleiche Länge (die Ellipsen entsprechen dem linken bzw. rechten Rand). Daraus folgt, dass auch die Längen der Kanten unterschiedlich sind.

Beispiel 5.3 In Abbildung 5.12 sieht man nun ein deutliches Beispiel dafür, dass nicht mit Schranken für die Längen im Parameterraum gearbeitet wird. Das Ergebnis ist aber trotzdem nicht schlecht, da es sich hauptsächlich auf die Kegelspitze auswirkt. Eine genauere Approximation ist hier aber nicht wünschenswert.

Kugel

Das nachfolgende Lemma wird benutzt, um eine intuitive Begründung für die Berechnung der Längenbeschränkung μ zu liefern. Um gute Ergebnisse bei der Flächenapproximation zu erhalten, ist es nicht erforderlich, eine wirkliche Schranke der Kantenlänge zu berechnen. Es wird lediglich argumentiert, warum die getroffene Wahl keine allzu schlechten Ergebnisse liefert.

Lemma 5.2 *Seien K_1 und K_2 zwei Kreise desselben Körpers mit Radius r_1 und r_2. Seien Δt_1 und Δt_2 die Schrittweite, mit der die Kreise approximiert werden. Falls $r_1 < r_2$, dann gilt $\Delta t_1 \geq \Delta t_2$.*

Beweis Nach Lemma 5.1 gilt $\Delta t_i = 2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_i}\right\}\right)$. Sei

$$r_1 < r_2$$

dann gilt

$$\begin{aligned} 1 - \frac{\rho}{r_1} &< 1 - \frac{\rho}{r_2}, \\ \max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_1}\right\} &\leq \max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_2}\right\} \\ \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_1}\right\}\right) &\geq \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_2}\right\}\right) \\ \Delta t_1 &\geq \Delta t_2. \end{aligned}$$

□

Beispiel 5.4 In Abbildung 5.13 sieht man ein Beispiel für Lemma 5.2: ρ ist die Schranke für die Hausdorff-Distanz; sie ist für beide Kreise gleich. Für den kleineren Kreis ergibt sich daraus eine größere Schrittweite Δt_1.

Es ist recht schwierig für allgemeine Kanten eine Beschränkung der Kantenlänge im Parameter der Kugel zu bestimmen. Deswegen werden zunächst die besonderen Kanten $u = a$ bzw. $v = a$, mit a konstant betrachtet. Für diese gilt, dass die

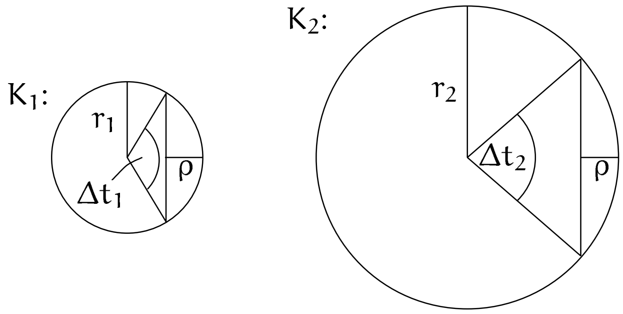


Abbildung 5.13.: Beispiel für Lemma 5.2.

Längenbeschränkung μ im Parameterraum gerade gleich der Schrittweite Δt ist, mit der diese Kante approximiert wurde.

Sei im Nachfolgenden r_K der Radius der Kugel, die approximiert werden soll. Für $v = a$ ist die entsprechende Kante ein Teil eines Großkreises; d.h. der Radius der Kreiskante ist gleich dem Radius r_K der Kugel.

Betrachtet man hingegen die Kanten $u = a$, und r_a bezeichne den Radius der Kreiskante, so gilt $r_a \leq r_K$. Die Gleichheit $r_a = r_K$ gilt genau für $a = 0$, und für $|a| < |b| < \frac{\pi}{2}$ gilt $r_a < r_b$. Denkt man an die Schnittkanten für die Kugel zurück (z.B. Abbildung 5.7), so sieht man, dass es keine Kante der Form $u = a$ für $a > \alpha$ geben kann (α ist hierbei der Winkel aus Abbildung 5.8); d.h. für diesen Wert ist der Radius minimal. Nach Lemma 5.2 ist die Schrittweite somit maximal, und nach obiger Überlegung somit auch die Längenbeschränkung μ.

Für den Radius r_α dieser Kante gilt $r_\alpha = r_K \cdot \cos\alpha$. Nach Lemma 5.1 gilt dann $\mu_\alpha = \Delta t_\alpha = 2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_K \cdot \cos\alpha}\right\}\right)$. Diese Längenbeschränkung der Segmente der Kante $u = \alpha$ wurde bei der Implementierung verwendet. Nach Gleichung 5.10 wird der Genauigkeitsparameter ρ_F für die Kugel somit als

$$\rho_F := \frac{\pi}{6}\left(2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_K \cdot \cos\alpha}\right\}\right)\right)^2 \tag{5.12}$$

berechnet.

Diese ganze Überlegung liefert allerdings nur eine Schranke für die Kanten der Form $u = a$ bzw. $v = a$. Für allgemeine Kanten muss dies nicht erfüllt sein. Es besteht aber die durchaus berechtigte Hoffnung, dass dies die Lösung nicht allzu schlecht macht, da eine „schräge“ Kante als Linearkombination zweier achsenparalleler Kanten dargestellt werden kann.

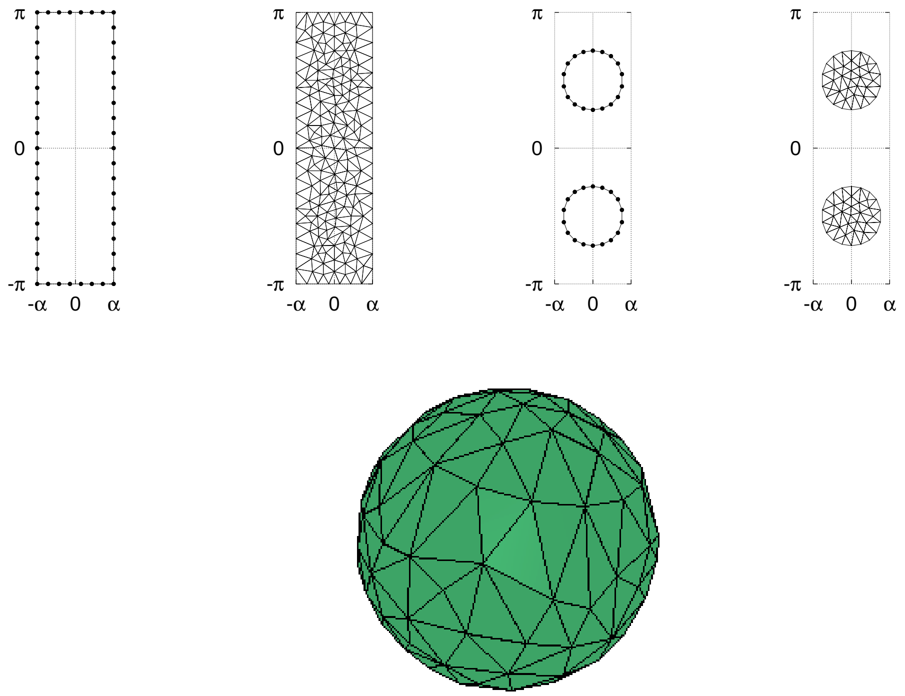


Abbildung 5.14.: Beispielapproximation einer Kugel.

Bisher wurde bei den Überlegungen nur die „standard“ Parametrisierung aus Abschnitt 4.3.2 verwendet. Die Kugelkappen werden aber in einem „gedrehten“ Parameterraum approximiert. Da der Parameterraum aber ansonsten identisch ist, wird auch für diese Approximation Gleichung 5.12 verwendet.

Beispiel 5.5 In Abbildung 5.14 sieht man ein Beispiel für die Approximation einer Kugel. Für beide Parametrisierungen sieht man jeweils die Eingabe- und Ausgabebilder. Dabei geht die u-Achse von links nach rechts und die v-Achse von oben nach unten. Man kann recht gut erkennen, dass die Dreiecke ziemlich wohlgeformt sind. Darunter ist dann die Kugel im Modellraum. Man kann die obere Schnittkante, die die Kappen abschneidet, recht gut erkennen.

Torus

Ähnlich wie bei der Kugel, betrachtet man beim Torus auch nur die Kanten der Form $u = a$ bzw. $v = a$ mit konstantem a. Auch für die Parametrisierung des Torus gilt, dass die Schrittweite Δt mit der diese Kanten approximiert werden, der Länge μ der entsprechenden Segmente im Parameterraum entsprechen.

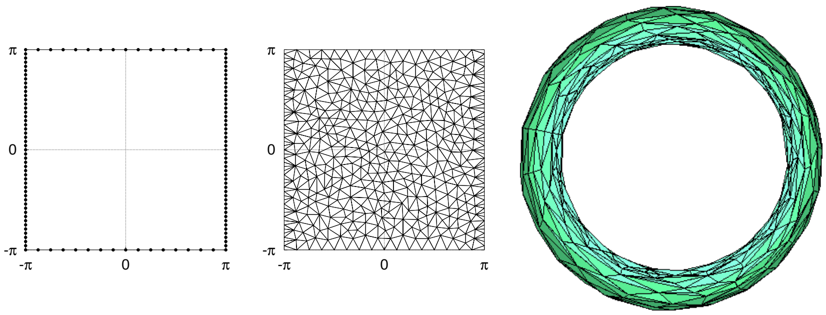


Abbildung 5.15.: Beispielapproximation eines Torus mit kleinem Nebenradius.

Die Frage, die es noch zu klären gibt, ist, welche von diesen Kanten man nehmen soll, um einen guten Wert zu erhalten. Im Gegensatz zur Kugel bestehen die Kanten beim Torus aus Kreisen mit evtl. sehr unterschiedlichen Radien. Konkret bedeutet dies, dass bei der Wahl von μ der Kreis entscheidend ist, der den kleinsten Radius besitzt (nach Lemma 5.2 ist dann Δt maximal).

Dies kann entweder der Nebenradius sein, oder aber die Differenz aus Hauptradius und Nebenradius. Sei also r der Neben- und R der Hauptradius des Torus; d.h. der relevante Radius $r_{\min}$ ist $r_{\min} = \min\{r, R - r\}$.

Nach Lemma 5.1 und obigen Überlegungen, erhält man, nach Gleichung 5.10, den Genauigkeitsparameter ρ_F für den Torus als

$$\rho_F := \frac{\pi}{6}\left(2 \cdot \arccos\left(\max\left\{\frac{1}{2}, 1 - \frac{\rho}{r_{\min}}\right\}\right)\right)^2, \text{ mit } r_{\min} = \min\{r, R - r\}. \tag{5.13}$$

Beispiel 5.6 In Abbildung 5.15 sieht man ein Beispiel für die Approximation eines Torus mit kleinem Nebenradius. Man kann sehr deutlich sehen, dass auf der ν-Achse die Kanten wesentlich feiner approximiert wurden als auf der u-Achse. Für die Approximation der Fläche war die Aufteilung auf der u-Achse entscheidend. Man sieht recht deutlich, dass die Dreiecke – bis auf den Rand entlang der ν-Achse – wohlgeformt sind.

Beispiel 5.7 In Abbildung 5.16 ist ein Beispiel für den zweiten Fall dargestellt. Hier ist der Nebenradius so groß gewählt, dass $R - r < r$ gilt. Konkret bedeutet dies, dass die Approximation der Schnittgeraden $\nu = \pi$ feiner als die Approximation der Schnittgeraden $u = \pi$ ist. Deswegen wurde der Radius $R - r$ für die Bestimmung der Genauigkeit gewählt. Man erhält sozusagen eine gespiegelte Version von Beispiel 5.6.

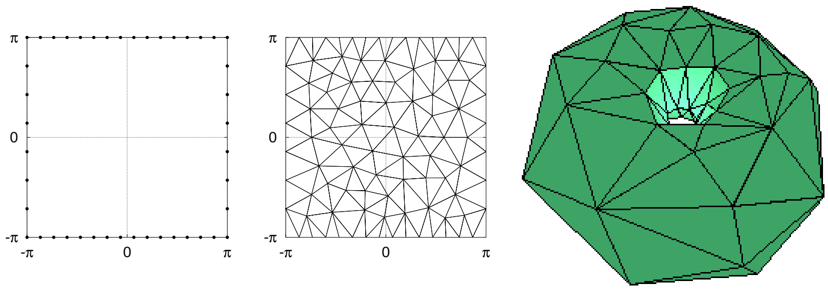


Abbildung 5.16.: Beispielapproximation eines Torus mit großem Nebenradius.

5.3.6. Eingabedaten für Triangle erzeugen

In diesem Unterabschnitt werden einige Details erläutert, wie das Eingabepolygon für den Triangulationsalgorithmus erzeugt wird; insbesondere wird auch erwähnt, in welcher Form dieser die Eingabe erwartet und wie man sie aus dem ACIS-Format erhalten kann. Da die meisten dieser Details an dieser Stelle schon intuitiv klar sein dürften, wird nur kurz erwähnt, auf was man achten muss.

Fast alle Schnittkanten[14] müssen im Parameterraum doppelt hinzugefügt werden. Dies hat insbesondere für Punkte, die mit einem Punkt auf der Schnittkante verbunden sind, die Konsequenz, dass man entscheiden muss, mit welchem der möglichen Punkte im Parameterraum er verbunden werden muss. Dies kann man aber leicht lösen, indem man sich alle möglichen Kombinationen anschaut, und diejenige mit der kleinsten Distanz nimmt.

Schaut man sich Abbildung 5.6 auf Seite 38 an, so sieht man, dass man auch Polygone mit Löchern spezifizieren muss. Bei Triangle geschieht dies, indem man einen Punkt angibt, der innerhalb des Lochs liegt. Programmtechnisch bedeutet dies, dass man anhand der ACIS-Daten einen solchen Punkt bestimmen muss. Dies ist aber nicht sonderlich schwierig, da eine Schleife in ACIS einen zusammenhängenden Kantenzug definiert. Eine Schleife ist auch orientiert, so dass man anhand dieser Orientierung das Innere und das Äußere bestimmen kann.

Konkret geht der Algorithmus alle Schleifen durch und fügt für jede Schleife einen Punkt als Loch hinzu; dieser Punkt wird bestimmt, indem man von einer Kante der Schleife aus rechtwinklig einen kleinen Schritt ins Äußere geht[15]. Dabei wird auch ein Punkt für die äußere Schleife eingefügt. Dies ist aber unproblematisch, da Triangle mit Löchern, die außerhalb des Eingabepolygons liegen, keine Probleme hat.

[14]Die Schnittkanten bei der Kugel, die die Kappen abschneiden, bilden hierbei eine Ausnahme.
[15]Dies kann man auf aufgrund der Orientierung der Schleife leicht bestimmen.

5.3.7. Interpretation der Ausgabe von Triangle

Die Ausgabe, die Triangle produziert, kann man leicht interpretieren, da sie aus einer Dreiecksliste besteht, in der die Dreiecke schon richtig orientiert sind. Diese Daten kann man direkt in entsprechende ACIS-Daten umwandeln. Die einzige Stelle, die etwas Sorgfalt benötigt, ist, zu verhindern, dass gleiche Knoten mehrfach eingefügt werden. Techniken hierfür werden im Abschnitt 5.3.8 näher beschrieben.

5.3.8. Implementierungsdetails

Bei einer konkreten Implementierung sind noch weitere Details zu beachten. Einige, mir wichtig erscheinende Details werden in diesem Abschnitt noch erwähnt und kurz erläutert.

Algorithmus 1 wurde nicht in dieser expliziten Form implementiert. Dies gilt vor allem für den Block zur Bestimmung der Schnittkanten. Diese werden nicht ganz am Anfang bestimmt, sondern erst bei der Approximation der zugehörigen Fläche berechnet. Man muss sie allerdings auch schon bei der Approximation der Kanten berücksichtigen, da gegebenenfalls Schnittpunkte der Schnittkanten mit Objektkanten in der Kantenapproximation auftreten müssen.

Diese Vorgehensweise liefert die gleichen Ergebnisse wie Algorithmus 1, besitzt aber den Vorteil, dass man keine Euler-Operatoren für das Hinzufügen von Kanten zu dem Objekt implementieren muss[16]. Allerdings ist die Darstellung in Algorithmus 1 intuitiver und übersichtlicher.

Da zu einer Kante zwei Flächen adjazent sind und da für diese beiden Flächen die gleiche Kantenapproximation benutzt werden muss, ist es notwendig von einer Kante zu deren Approximation zu gelangen (die Kantenapproximation wird ja *vor* der Flächenapproximation berechnet). Dies wird erreicht, indem die einzelnen Approximationssegmente bei der Kante als Liste gespeichert werden. So wird ein einfacher Zugriff darauf ermöglicht.

Hat man die Approximation im Parameterraum durchgeführt, so muss man die Punkte wieder in den Modellraum zurückrechnen. Da die Punkte auf den Grenzen schon im Modellraum vorhanden sind und es nicht sinnvoll ist, Duplikate davon im Modellraum zu erzeugen, sollte man die Koordinaten im Parameterraum nicht einfach in Koordinaten im Modellraum zurückrechnen.

[16] Es wird ja an keiner Stelle wirklich gebraucht, dass die Schnittkanten zum Objekt hinzugefügt werden.

Ein weiterer Nachteil einer einfachen „Hin- und Zurückrechnung“ ist auch, dass die Umrechnung von Punktkoordinaten in den Parameterraum und zurück mit normaler Fließkommaarithmetik erfolgt. Da aber Grenzpunkte in zwei verschiedene Parameterräume gerechnet werden müssen, würden die Duplikate aufgrund numerischer Ungenauigkeiten sowieso nicht auf die gleichen Koordinaten abgebildet.

D.h. in jedem Fall sollten bei den Punkten, die das Eingabepolygon erzeugen, ein Verweis auf die zugehörige Punkte im Modellraum gespeichert werden. Auf diese Weise kann man die beiden Nachteile ganz leicht vermeiden. Triangle erlaubt diese Vorgehensweise mit der Möglichkeit, bei den Punkten zusätzliche Attribute zu speichern.

Bisher wurde auch noch nicht erwähnt, wie die Schnittkanten eigentlich bestimmt werden. Es werden ja die Schnittpunkte der Objektkanten mit den Schnittgeraden ausgerechnet. Anhand dieser Schnittpunkte muss man dann die Schnittkanten bestimmen. Man könnte also einfach zwei „benachbarte“[17] Schnittpunkte mit einer Kante verbinden. Dies hat aber den Nachteil, dass diese Kante außerhalb des Objekts liegen kann. D.h. man muss feststellen, ob eine Schnittkante zum Objekt gehört oder nicht.

Bei dem Kegel/Zylinder ist dies denkbar einfach, da hier immer abwechselnd eine Kante dazugehört und die nächste[18] nicht dazugehört.

In den anderen Fällen ist dies nicht ganz so einfach. Zum Beispiel kann der Fall auftreten, dass zwei aufeinanderfolgende Kanten dazugehören; dies kann bei den Schnittpunkten der Schnittkanten passieren. Auch hat man anders als bei dem Kegel/Zylinder keine Kante, von der man weiß, dass sie sicher dazugehört. Deswegen wird einfach für jede Schnittkante nachgesehen, ob ihr Mittelpunkt in der Fläche liegt oder nicht (dieses Verfahren wäre natürlich auch für den Fall des Kegels/Zylinders anwendbar; das oben vorgestellte Verfahren ist aber nicht von numerischen Ungenauigkeiten abhängig und wurde deswegen bevorzugt). Dafür ist eine Testroutine nötig, die prüft, ob ein Punkt in einer Fläche liegt. In Abschnitt 5.3.9 wird so eine Testroutine vorgestellt.

Es kann auch der Spezialfall auftreten, dass eine Schnittkante mit einer Objektkante zusammenfällt (dieser Fall wird erkannt, indem man feststellt, dass alle Punkte in einer ϵ-Umgebung zu der Parameterkante liegen). In diesem Fall ist es nicht notwendig, dass

[17] Benachbarte Schnittpunkte kann man finden, indem man die Punkte im Parameterraum sortiert – je nachdem um welche Schnittgerade es sich handelt ggf. nach anderen Parametern.

[18] Das „nächste“ bezieht sich hier auf eine Sortierung der Punkte mit dem u-Parameterwert als Schlüssel.

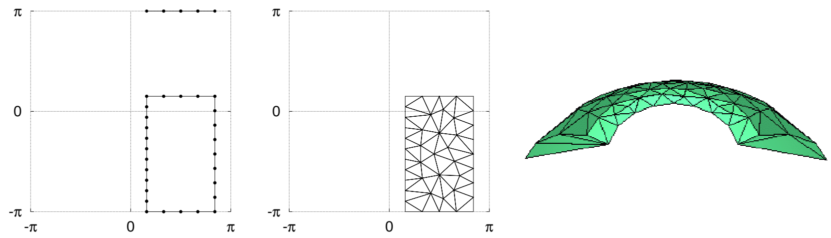


Abbildung 5.17.: Beispiel für den Spezialfall, dass eine Objektkante mit der Schnittkante zusammenfällt. Im linken Bild kann man deutlich sehen, dass oben eine einzelne Kante erzeugt wird. Triangle entfernt diese aber und das Ergebnis entspricht der Vorstellung.

die Schnittkante doppelt (d.h. bei beiden Grenzen des Parameterraums) hinzugefügt wird; allerdings ist es wesentlich einfacher dies zu tun, da Triangle einzelne Kanten einfach entfernt. Ein Beispiel für diesen Spezialfall kann man in Abbildung 5.17 sehen.

5.3.9. Schnittprobleme und Enthaltenseintests

Wie man in Abschnitt 5.3.8 gesehen hat, sind für das Hinzufügen der Schnittkanten Operationen notwendig, die den Schnitt mit Kanten berechnen und die prüfen, ob ein Punkt in einer Fläche liegt. In diesem Abschnitt wird erläutert, wie man dies bewerkstelligen kann.

Schnitt zwischen Ebene und Ellipse

Eigentlich muss man für die Berechnung der Schnittpunkte den Schnitt der Schnittgeraden mit den Objektkanten berechnen; d.h. den Schnitt zwischen Ellipsen und/oder Geraden. Es ist klar, dass ein solches Verfahren numerisch anfällig wäre. Deswegen wurden die Objektkanten mit den Ebenen geschnitten, in denen die Schnittgeraden liegen. Da man auf diese Weise eventuell falsche Schnittpunkte bekommt (da z.B. bei der Kugel die Schnittgerade der Form $v = \pi$ nur ein Halbkreis ist), muss man noch verifizieren, ob er überhaupt einer ist. Dies kann man aber sehr leicht realisieren.

D.h. als Operation für Schnittberechnungen ist lediglich der Schnitt zwischen einer Ebene und einer Ellipse notwendig.

Sei $\vec{r}_e(t) = \vec{C} + \cos t \cdot \vec{a} + \sin t \cdot \vec{b}$ die Parametrisierung der Ellipse, und sei die Ebene gegeben durch $\vec{n} \cdot \vec{x} = n_0$. Setzt man die Parametrisierung der Ellipse in die Ebenengleichung ein, so erhält man

$$f_1 \cos t + f_2 \sin t + f_0 = 0 \quad \text{mit} \quad f_0 := \vec{n} \cdot \vec{C} - n_0, \quad f_1 := \vec{n} \cdot \vec{a}, \quad f_2 := \vec{n} \cdot \vec{b}.$$

Setzt man $\cos t = \sqrt{1-\sin^2 t}$ bzw. $\sin t = \sqrt{1-\cos^2 t}$ ein und löst nach $\sin t$ bzw. $\cos t$ auf, so erhält man als Lösung (falls $f_1 \neq 0$ und $f_2 \neq 0$)

$$\sin t_{1,2} = \frac{-f_2 f_0 \pm f_1 \sqrt{f_1^2 + f_2^2 - f_0^2}}{f_1^2 + f_2^2}$$
$$\cos t_{1,2} = \frac{-f_1 f_0 \mp f_2 \sqrt{f_1^2 + f_2^2 - f_0^2}}{f_1^2 + f_2^2}.$$

Ist die Diskriminante negativ, so gibt es keinen Schnittpunkt. Falls sie Null wird, so liegt ein Berührpunkt vor. Im Falle $f_1 \neq 0$ und $f_2 \neq 0$ liegt die Ellipse in der Ebene. Ansonsten besitzt dieses Gleichungssystem für $-\pi < t \leq \pi$ zwei Lösungen, die die Schnittpunkte der Ellipse mit der Ebene liefern.

Auf diese Weise kann man also den Schnitt einer Ebene mit einer Ellipse berechnen. Im allgemeinen ist man aber an dem Schnitt einer Ebene mit einer Kante interessiert. Und die Kante muss nicht unbedingt die komplette Ellipse sein. D.h. man muss nach dieser Berechnung noch prüfen, ob die berechneten Schnittpunkte auf der Kante liegen. Das geschieht, indem man den Anfangs- und Endpunkt der Kante in den Parameterraum rechnet und schaut, ob die Schnittpunkte im entsprechenden Intervall im Parameterraum liegen. Dabei muss man natürlich die Orientierung der Kante berücksichtigen (zur Erinnerung siehe Bild 3.6 auf Seite 18).

Punkt-auf-Fläche-Test

In diesem Abschnitt wird beschrieben, wie ein Test, ob ein Punkt $\vec{P}$ in einer Fläche f liegt, aussehen kann. Dabei wird benutzt, dass die Begrenzungskanten bezüglich der Normalen im Gegenuhrzeigersinn orientiert sind.

Das Verfahren sieht insgesamt so aus, dass von dem zu testenden Punkt $\vec{P}$ aus ein Strahl $\vec{g}(t)$ auf der Oberfläche ausgeschickt wird und der Punkt $\vec{S}$ bestimmt wird, in dem sich dieser Strahl zum ersten Mal mit einer Begrenzungskante $\vec{e}(t)$ schneidet. In diesem Schnittpunkt $\vec{S}$ wird dann die Orientierung von dem Dreibein aus der Tangente an $\vec{e}$, der Normalen im Punkt $\vec{S}$ und der Tangenten an den Strahl $\vec{g}$ bestimmt. Anhand dieser Orientierung kann man dann entscheiden, ob $\vec{P}$ innerhalb oder außerhalb der Fläche liegt. Siehe hierzu auch Abbildung 5.18.

Konkret heißt das

$$\vec{P} \in f \iff \det\left[\vec{t}^{\,\vec{e}}_{\vec{S}}, \vec{n}^{\,f}_{\vec{S}}, \vec{t}^{\,\vec{g}}_{\vec{S}}\right] > 0.$$

Dabei ist $\vec{t}^{\,\vec{e}}_{\vec{S}}$ bzw. $\vec{t}^{\,\vec{g}}_{\vec{S}}$ die Tangente im Punkt $\vec{S}$ an die Kurve $\vec{e}(t)$ bzw. $\vec{g}(t)$, und $\vec{n}^{\,f}_{\vec{S}}$ bezeichnet die Normale im Punkt $\vec{S}$ an die Oberfläche von f.

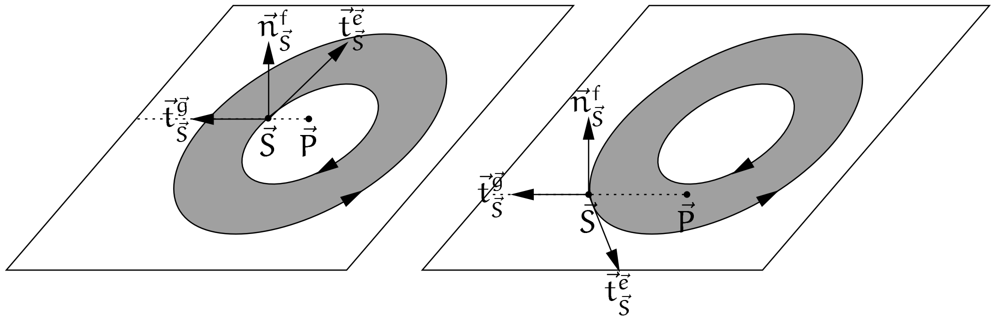


Abbildung 5.18.: Beispiel für Punkt-auf-Fläche-Test in der Ebene: Anhand der Orientierung des Dreibeins kann entschieden werden, ob der Punkt $\vec{P}$ innerhalb oder außerhalb der Fläche liegt. (Die gepunktete Linie ist der Strahl $\vec{g}(t)$.)

Der Algorithmus für den Test geht also so vor, dass alle Schnittpunkte der Kurve $\vec{g}(t)$ mit den Begrenzungskanten der Fläche berechnet werden. Von diesen Schnittpunkten wird der „früheste" Schnittpunkt bestimmt. Dies kann man machen, indem man den kleinsten positiven Parameterwert t_{min} und dessen zugehörigen Punkt nimmt[19].

Im Nachfolgenden wird jetzt für die einzelnen Oberflächentypen erläutert, welcher Strahl $\vec{g}(t)$ verwendet wird und wie man dafür die Schnittberechnung mit den Begrenzungskanten numerisch stabil gestalten kann.

Ebene

Im Falle der Ebene wird als Strahl einfach eine beliebige Halbgerade verwendet, die in der Ebene liegt und in $\vec{P}$ anfängt; d.h.

$$\vec{g}(t) := \vec{P} + t\vec{w}, \quad t \geq 0 \quad \text{mit} \quad \vec{n} \cdot \vec{w} = 0$$

Hierbei sei $\vec{n}$ die Normale der Ebene.

Um den Schnitt zwischen der Halbgeraden und den Kanten zu berechnen, wird aus numerischen Gründen der Schnitt zwischen der Ebene, die senkrecht zu der Ebene, die die Oberfläche definiert, steht und in der $\vec{g}(t)$ liegt, berechnet (d.h. die Ebene mit der Normalen $\vec{n} \times \vec{w}$ durch den Punkt $\vec{P}$).

[19]Die negativen Parameterwerte werden ignoriert, da $\vec{g}(t)$ als Strahl, d.h. gerichtet, interpretiert wird. Allerdings muss man gegebenenfalls bei Ellipsen aufpassen, dass man den Parameterbereich richtig wählt; d.h. $0 \leq t < 2\pi$.

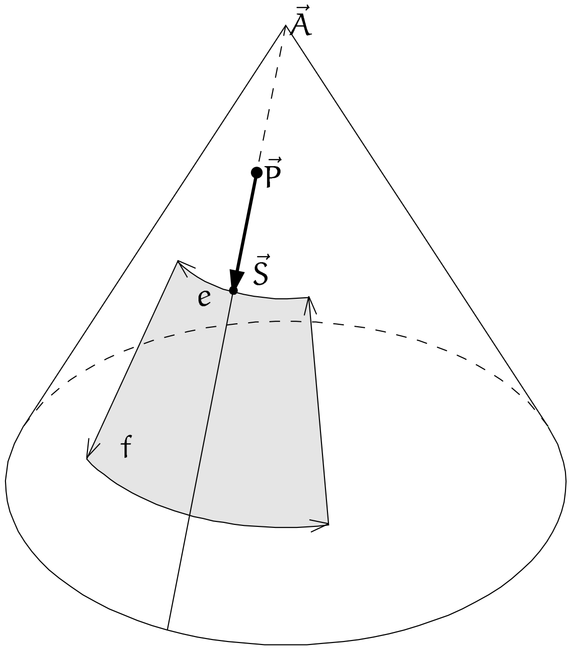


Abbildung 5.19.: Der Verfolgungsstrahl beim Punkt-auf-Fläche-Test für den Kegel.

Kegel/Zylinder

Beim Kegel und Zylinder wird der Strahl so gewählt, dass er eine Halbgerade ist; d.h. man geht auf dem Mantel nach „oben“ oder nach „unten“.

Sei $\vec{C}$ der Mittelpunkt der Basisellipse des Kegels bzw. des Zylinders. Sei $\vec{n}$ der Normalenvektor der Ebene, in der die Basisellipse liegt. Seien s bzw. c der Sinus- bzw. Kosinuswert des halben Hauptwinkels. Sei r die Länge der großen Halbachse.

Mit diesen Notationen ergibt sich für den Fall eines *Zylinders* der Strahl als

$$\vec{g}(t) := \vec{P} + t\vec{n}, \quad t \geq 0.$$

Im Falle eines *Kegels* ist es etwas komplizierter. Als erstes wird die Kegelspitze $\vec{A}$ berechnet. Sei $\vec{F}_K(u, v)$ die Parametrisierung des Kegels. Die Kegelspitze wird gerade für $u = -\frac{1}{s}$ angenommen. Somit ergibt sie sich als $\vec{A} = \vec{F}_K(-\frac{1}{s}, v) = \vec{C} - \frac{c}{s} r \cdot \vec{n}$. Den Richtungsvektor für den Strahl erhält man nun einfach, indem man von $\vec{A}$ in Richtung $\vec{P}$ geht (vgl. Abbildung 5.19); d.h.

$$\vec{g}(t) := \vec{P} + t(\vec{P} - \vec{A}), \quad t \geq 0.$$

Kugel

Bei der Kugel wird ein Großkreis als Strahl verwendet. Hier empfiehlt es sich nicht, einen beliebigen Großkreis zu verwenden, da dieser eventuell keine Kante schneiden wird, da die Kugel zyklisch ist. Um sicherzustellen, dass dieser Fall nicht auftritt, wird der Großkreis so gewählt, dass er durch einen beliebigen Punkt auf dem Rand[20] geht.

Sei also $\vec{C}$ der Mittelpunkt der Kugel und sei $\vec{Q}$ ein beliebiger Punkt auf dem Rand. Der gesuchte Großkreis liegt somit in der Ebene, die von den Punkten $\vec{C}$, $\vec{P}$ und $\vec{Q}$ aufgespannt wird. Sei $\vec{n}$ der Einheits-Normalenvektor dieser Ebene. Dann wird als Strahl

$$\vec{g}(t) := \vec{C} + \cos t \cdot (\vec{P} - \vec{C}) + \sin t \cdot \vec{n} \times (\vec{P} - \vec{C})$$

verwendet. Da der Schnittpunkt anhand der Parametrisierung ausgewählt wird, ist es wichtig, dass die Parametrisierung so gewählt wird, dass $\vec{g}(0) = \vec{P}$ ist, was bei obiger Parametrisierung offensichtlich gilt.

Für die Bestimmung der Schnittpunkte wird jede Objektkante mit der Ebene, die von den Punkten $\vec{C}$, $\vec{P}$ und $\vec{Q}$ aufgespannt wird, geschnitten.

Torus

Schließlich muss noch der Torus geklärt werden. Um einen einfachen Test zu gewährleisten, möchte man sich gerne auf Kreise als Strahl beschränken. Deswegen wird als Strahl einfach der „Nebenkreis" durch den Punkt $\vec{P}$ gewählt. Da der Torus aber zyklisch ist, kann es passieren, dass dieser Strahl gar keine Objektkante schneidet, obwohl die Fläche gar nicht der komplette Torus ist. Sollte dieser Fall auftreten, muss man einen anderen Strahl wählen. Hierbei wird ein Strahl gewählt, der in einer Ebene liegt, die parallel zu der Ebene ist, in der der Hauptkreis liegt. Damit sichergestellt ist, dass eine Objektkante getroffen wird, wird der Strahl so gewählt, dass er durch einen Randpunkt geht. Das hat zur Folge, dass der Strahl nicht durch $\vec{P}$ gehen muss. Dies ist aber insofern unproblematisch, da es von $\vec{P}$ aus einen Weg zu einem Punkt auf dem zweiten Strahl existiert, ohne dass eine Kante geschnitten wird (nämlich gerade ein Weg auf dem ersten Strahl). Siehe hierzu auch Abbildung 5.20.

Sei $\vec{C}$ der Mittelpunkt des Torus. Sei $\vec{n}$ die Normale der Ebene, in der der Hauptkreis liegt. Sei R der Wert des Hauptradius. Ferner sei $\vec{Q}$ ein beliebiger Punkt auf einer Objektkante.

Für den ersten Strahl erhält man den Kreis

$$\vec{g}_1(t) := \vec{C}_1 + \cos t \cdot \vec{a}_1 + \sin t \cdot \vec{b}_1,$$

[20] Anfangs- oder Endpunkt einer Kante sollte man nicht nehmen, da dies numerisch nicht besonders stabil wäre. In der Implementierung wird einfach der Mittelpunkt einer beliebigen Kante verwendet.

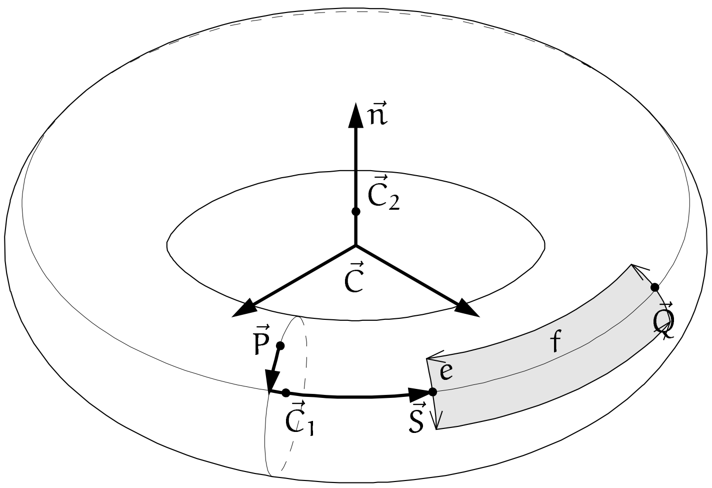


Abbildung 5.20.: Die Verfolgungsstrahlen beim Punkt-auf-Fläche-Test für den Torus.

mit

$$\begin{aligned}
\vec{C}_1 &:= \vec{C} + R\frac{\vec{P}-\vec{C}-[\vec{n}\cdot(\vec{P}-\vec{C})]\vec{n}}{\left\|\vec{P}-\vec{C}-[\vec{n}\cdot(\vec{P}-\vec{C})]\vec{n}\right\|}\\
\vec{a}_1 &:= \vec{P}-\vec{C}_1\\
\vec{b}_1 &:= \frac{\vec{n}\times(\vec{P}-\vec{C})}{\left\|\vec{n}\times(\vec{P}-\vec{C})\right\|}\times\vec{a}_1.
\end{aligned}$$

Für den zweiten Strahl erhält man den Kreis

$$\vec{g}_2(t) := \vec{C}_2 + \cos t\cdot\vec{a}_2 + \sin t\cdot\vec{b}_2,$$

mit

$$\begin{aligned}
\vec{C}_2 &:= \vec{C} + (\vec{n}\cdot\vec{Q})\vec{n}\\
\vec{a}_2 &:= \frac{\left\|\vec{Q}-\vec{C}_2\right\|}{R}(\vec{C}_1-\vec{C})\\
\vec{b}_2 &:= \vec{n}\times\vec{a}_2.
\end{aligned}$$

5.4. Erweiterbarkeit

Nachdem nun der Algorithmus detailliert beschrieben wurde, ist es eine Überlegung wert, was zu tun ist, wenn man den Algorithmus um neue Kurven- oder Oberflächentypen erweitern will. Geht man die Beschreibung des Algorithmus durch, so sind diese

Punkte eigentlich klar. In diesem Abschnitt werden sie deswegen nur noch einmal übersichtlich zusammengestellt.

5.4.1. Erweitern um neue Kurventypen

Will man den Algorithmus um neue Kurventypen erweitern, so sind zwei Aufgaben zu bewältigen. Zum einen muss man den Schnitt zwischen dem neuen Kurventyp und den Schnittkanten berechnen. Bei den bisherigen Schnittkanten, reicht es aus, den Schnitt zwischen dem neuen Kurventyp und einer Ebene zu berechnen. Diese Operation muss implementiert werden.

Zum anderen muss die Kantenapproximation für den neuen Kurventyp implementiert werden. Dies stellt vermutlich die größte Schwierigkeit dar, da der Genauigkeitsparameter eine Schranke für die Hausdorff-Distanz darstellt. Das bedeutet, dass die Kantenapproximation dies gewährleisten muss. Darüberhinaus ist es sogar wünschenswert, dass der Genauigkeitsparameter nicht nur eine Schranke für die Hausdorff-Distanz ist, sondern dass die Hausdorff-Distanz nicht allzuviel davon abweicht.

Im Falle der Ellipse war dies noch recht einfach zu machen. Für kompliziertere Kurventypen könnte dies mit einigen Schwierigkeiten verbunden sein. Ein Anhaltspunkt für einen Lösungsansatz liefert die Arbeit von Cuillière [Cui97] und die Abschätzung aus [FMM86].

5.4.2. Erweitern um neue Oberflächentypen

Um den Algorithmus um neue Oberflächentypen zu erweitern, ist etwas mehr Arbeit notwendig. Insbesondere kommen auch einige Details hinzu, die man beachten muss und die einem das Leben schwer machen können.

Zunächst ist es klar, dass man eine Parametrisierung für den Oberflächentyp benötigt. Damit man die Begrenzungskanten in den Parameterraum transferieren kann, ist es notwendig die Inverse zur Parametrisierung zu bestimmen. Je nach vorgegebener Parametrisierung kann dies schwierig sein. Cuillière gibt in [Cui98] ein Verfahren zur Berechnung an, falls dies analytisch nicht möglich ist.

Daneben muss man sich auch überlegen, wie man den Genauigkeitsparameter für die Flächenapproximation für diesen Oberflächentyp bestimmen kann. Unter Umständen kann es sein, dass für diesen Oberflächentyp die Vorgehensweise über eine Triangulierung im Parameterraum mit einfachen Qualitätskriterien keine guten Ergebnisse liefert. In diesem Fall muss man sich überlegen, ob man eine andere Punktauswahl treffen soll. In Abschnitt 6.2 werden einige Erweiterungen kurz diskutiert.

Unter Umständen ist auch eine Erweiterung der Routine notwendig, die die Approximation gerader Kanten übernimmt. Diese Routine wird gebraucht, um gerade

Kanten, die zu einer gekrümmten Oberfläche adjazent sind, zu approximieren, so dass die gekrümmte Oberfläche adäquat approximiert werden kann. Bisher war dies nur für den Fall des Kegels/Zylinders notwendig. Es kann aber sein, dass es bei neuen Oberflächen auch gerade Kanten geben kann. In diesem Fall, muss man diese Routine so erweitern, dass sie auch damit arbeiten kann.

Der letzte Fall, den man behandeln muss, sind die Schnittkanten. Hier muss man besondere Sorgfalt walten lassen, da einige Details beachtet werden müssen. Ist die Oberfläche nicht periodisch in einem oder mehreren Parametern oder besitzt die Parametrisierung keine Singularitäten, so benötigt man keine Schnittkanten.

Im Falle, dass es Singularitäten gibt, muss man sich überlegen, wie man damit zurechtkommen kann. Eine Möglichkeit hat man ja bei der Diskussion der Kugel gesehen. Dort wurden Schnittkanten eingefügt, die dazu dienten, die Singularitäten aufzuheben. Ist ein ähnliches Verfahren nicht möglich, so muss man die Singularitäten als gesonderte Spezialfälle behandeln.

Falls Schnittkanten gebraucht werden, muss man sich als erstes überlegen, wo man diese am Besten anbringt und welche Form sie haben. Sind die Schnittkanten Geraden oder Ellipsen, so ist dies kein Problem. Haben sie aber eine andere Gestalt, so muss man sich das etwas genauer ansehen. Liegt der Schnittkantentyp in einer Ebene, so ist dies auch unproblematisch, da für die Berechnung des Schnitts mit den Objektkanten immer die Ebene benutzt wurde, in der die Schnittkante liegt. Falls das die Schnittkante nicht mehr zulässt, müssen alle Kurventypen so erweitert werden, dass man den Schnitt mit dem neuen Schnittkantentyp berechnen kann.

Ferner benötigt man für die Schnittkantenberechnung noch eine Punkt-auf-Fläche-Testroutine, da damit bestimmt wird, ob der entsprechende Teil der Schnittkante überhaupt in der Fläche liegt.

6. Vergleich mit anderen Algorithmen und Ausblick

In diesem Kapitel wird der vorgestellte Algorithmus mit ähnlichen Algorithmen aus den Arbeiten [Che93, Cui97, Cui98, DM91, PT98, SH92] verglichen. Diese Liste an Arbeiten erhebt keinesfalls den Anspruch vollständig zu sein. Bei dem Vergleich werden vor allem die Aspekte aus Abschnitt 5.2 angesprochen, und es wird diskutiert, was diese Algorithmen mehr leisten können und wo sie gegebenenfalls Schwächen besitzen.

Einige der Algorithmen garantieren gewisse Schranken für die Genauigkeit der Approximation. Allerdings wird nicht genau angegeben, wieviele Dreiecke man z.B. braucht, um diese Bedingung zu erzielen. Das bedeutet, dass ein Vergleich, der die „Güte“ der Approximation beinhalten würde, nicht möglich ist, ohne die vorgeschlagenen Algorithmen nachzuimplementieren. Deswegen wurde auf einen solchen Vergleich verzichtet.

Der zweite Teil dieses Kapitel besteht aus einem Ausblick, der aufzeigen soll, in welche Bereiche man noch Arbeit stecken kann, um das Ergebnis zu verbessern. Insbesondere wird auch versucht, darzulegen, welche Ideen aus den anderen Arbeiten verwendet werden können, um ein besseres Ergebnis zu erreichen.

6.1. Vergleich mit anderen Algorithmen

Im nachfolgenden werden die Verfahren aus [Che93, Cui97, Cui98, DM91, PT98, SH92] kurz beschrieben, und Unterschiede, Stärken, Schwächen, etc. zu dem Algorithmus, der in dieser Arbeit erläutert wurde, werden aufgezeigt.

Chew [Che93]
Chew beschreibt in [Che93] einen Algorithmus zur Approximation von gekrümmten Oberflächen, der im Gegensatz zu allen anderen mir bekannten Algorithmen die Triangulierung nicht im Parameterraum durchführt. Vielmehr erweitert er den Begriff der Delaunay-Triangulierung auf gekrümmte Oberflächen. Hierzu muss er unter anderem den Begriff „Umkreis von einem Dreieck“ auf gekrümmte Oberflächen erweitern.

Der Algorithmus benötigt eine (gegebenenfalls schlechte) Ausgangstriangulierung der Oberfläche, die anschließend verfeinert wird. Der Algorithmus benutzt somit ein a posteriori Verfahren, das inkrementell die Triangulierung verfeinert, bis die gewünschten Gütekriterien erfüllt werden. Der Algorithmus kann garantieren, dass die Dreiecke wohlgeformt sind (d.h. keine Winkel kleiner als 30° vorkommen) und dass die Dreiecke nicht zu groß sind (hier kann man prinzipiell jede Eigenschaft einsetzen, die durch Verkleinerung der Dreiecksgröße erreicht werden kann).

Das Problem der verallgemeinerten Polyeder spricht der Algorithmus nicht direkt an; d.h. die Fragestellung, wie man sicherstellen kann, dass an den Begrenzungskanten keine Lücken entstehen oder Selbstdurchdringung stattfindet. Der Algorithmus führt aber eine constraint-basierte Delaunay-Triangulierung durch und kann somit gekrümmte Oberflächen mit Begrenzungskanten approximieren. Es ist somit denkbar, diesen Algorithmus als alternative Punktauswahl zu der in dieser Arbeit vorgestellten Methode zu verwenden. Dies hätte dann auch den Vorteil, dass man die Gütekriterien direkter kontrollieren könnte.

Cuillière [Cui97, Cui98]

Cuillière beschreibt in [Cui98] einen Algorithmus zur Triangulierung verallgemeinerter Polyeder. Er schlägt hierzu einen kantenorientierten Algorithmus vor. Wie man die Kanten hierfür approximiert, beschreibt er in [Cui97]. Dort zeigt er auch die Wichtigkeit für die kantenorientierte Vorgehensweise, dass die Kanten so genau approximiert wurden, so dass die Approximation der Fläche mit der gewünschten Genauigkeit möglich ist.

Sowohl für die Approximation der Kanten als auch für die der Flächen schlägt er einen a priori Ansatz vor, indem er eine Dichtefunktion für die Punktverteilung bestimmt. Auf diese Weise kann er auch Oberflächen mit starker Krümmungsänderung behandeln.

Die Güte der Approximation bewertet er mittels eines Qualitätsindikators, der sich aus zwei Faktoren zusammensetzt. Zum einen sollen die Dreiecke wohlgeformt sein; d.h. am besten sind gleichseitige Dreiecke. Zum anderen soll aber auch die Dichtefunktion möglichst exakt eingehalten werden.

Die eigentliche Triangulierung wird im Parameterraum der Oberfläche durchgeführt. Dafür wird ein Advancing-Front Algorithmus verwendet (für einen Überblick über verschiedene Triangulierungsalgorithmen sei auf [BE95] verwiesen).

Cuillière spricht u.a. auch das Problem von periodischen Oberflächentypen (z.B. Zylinder) an. Die angedeutete Lösung benutzt eine „lokale Parametrisierung“. Dies funktioniert, da ein Advancing-Front Triangulierungsalgorithmus verwendet wird, der immer nur lokal operiert. Leider wird diese Lösung nicht genauer erläutert.

Insgesamt kann man sagen, dass die in den beiden Arbeiten von Cuillière vorgestellte Vorgehensweise sehr ähnlich zu dem in dieser Arbeit präsentierten Algorithmus ist. Die wichtigsten Unterschiede sind, dass Cuillière auch Oberflächen mit starker Krümmungsänderung gut behandeln kann. Ferner benötigt er zum Approximieren periodischer Oberflächen keine Schnittkanten, da er diesen Fall durch eine lokale Triangulierung erschlagen kann.

Dolenc und Mäkelä [DM91]

Dolenc und Mäkelä stellen in [DM91] einen Algorithmus vor, der verallgemeinerte Polyeder trianguliert. Die Anwendung, die sie für ihren Algorithmus nennen ist die Stereolithographie. Daraus leiten sie zwei wichtige Kriterien ab, die erfüllt werden müssen:

- Die Anzahl der Dreiecke soll minimiert werden, da Apparate zur Stereolithographie nur eine beschränkte Anzahl von Dreiecken zulassen.
- Die Triangulierung muss korrekt sein; d.h. Lücken zwischen Flächen oder Durchdringungen von Flächen dürfen nicht auftreten. Dies ist eine Forderung, die Apparate zur Stereolithographie an die Eingabedaten stellen.

Der Algorithmus arbeitet flächenorientiert. Das bedeutet, dass am Ende die Lücken, die eventuell zwischen den Flächen entstanden sind, ausgefüllt werden müssen. Leider wird nur angedeutet, wie dies erreicht werden kann. Der Vorteil dieses Vorgehens ist, dass die Ausgangsdaten nicht zwangsläufig einen korrekten festen Körper repräsentieren müssen.

Für die Triangulierung der Oberfläche wird ein einfacher adaptiver Algorithmus verwendet, der die Fläche solange in Rechtecke unterteilt, bis die gewünschte Genauigkeit erreicht ist. Als Kriterium für die Genauigkeit wird ein Flachheitstest verwendet. Die Rechtecke kann man anschließend recht einfach triangulieren.

Die Probleme, die beim Approximieren periodischer Oberflächen entstehen, werden nicht angesprochen. Aufgrund einer Beispielillustration kann man aber vermuten, dass sie – ähnlich wie in dem hier vorgestellten Algorithmus – Schnittkanten verwenden, um die Periodizität aufzuheben. Diese Schnittkanten dürften für sie auch kein Problem darstellen, da sie grundsätzlich mit einer rechteckigen Unterteilung des Parameterraums arbeiten.

Piegl und Tiller [PT98]

Die Herangehensweise von Piegl und Tiller in [PT98] an das Problem der Approximation verallgemeinerter Polyeder ist flächenorientiert: die Oberflächen und deren Begrenzungskanten werden unabhängig voneinander approximiert und in einem weiteren Schritt vereinigt.

Sowohl die Approximation der Kanten als auch die Approximation der Flächen erfolgt adaptiv; die Kanten werden solange in der Mitte unterteilt, bis die Genauigkeitskriterien erfüllt sind. Bei den Flächen wird wie in [DM91] eine Unterteilung in Rechtecke vorgenommen, die anschließend recht einfach trianguliert werden können.

Die Genauigkeitsparameter für die Approximation der Kanten und der Flächen sind unabhängig voneinander. Es sind auch einige Beispiele angegeben, wie es sich auswirkt, wenn nur einer der beiden Parameter verändert wird. Dies ist bei dem hier vorgestellten Algorithmus prinzipiell auch möglich. Allerdings wurde eine Verbindung der beiden Parameter hergeleitet, so dass man nur noch einen Parameter angeben muss, da dies in der Regel eine für den Benutzer intuitivere Bedienweise ist. Piegl und Tiller verzichten in [PT98] auf eine solche Verbindung.

Der Algorithmus wird für NURBS-Oberflächen beschrieben und ist geometriebasiert. Geometrie-basiert bedeutet in diesem Fall, dass der Algorithmus für die Einhaltung der Genauigkeitsparameter nicht auf besondere Eigenschaften der Parametrisierung der Oberfläche angewiesen ist. Die Algorithmen aus [SH92] und [Cui98] benutzen nämlich Sätze aus [FMM86] und [SH92] für die Genauigkeitsparameter. Diese Sätze erfordern C^2-Oberflächen, wohingegen der Algorithmus von Piegl und Tiller lediglich C^0-Oberflächen voraussetzt.

Sheng und Hirsch [SH92]

Der Algorithmus von Sheng und Hirsch in [SH92] ist auch als flächenorientiert anzusehen. Die komplette Triangulierung wird im Parameterraum durchgeführt. Hierzu wird für jede Fläche ein entsprechendes Polygon im Parameterraum berechnet, das dort mit einer constraint-basierten Delaunay-Triangulierung zunächst trianguliert wird. Diese Ausgangstriangulierung wird so lange verfeinert, bis die Genauigkeitskriterien erfüllt werden.

Nachdem so alle Flächen approximiert wurden, müssen eventuell entstandene Lücken geschlossen werden. Dies geschieht, indem Punkte bzw. Kanten, die nahe beieinander liegen und eine Lücke bilden, verschoben werden, so dass sie danach aufeinander liegen.

Um eine Schranke für die Dreiecksgröße zu bestimmen, wird eine Verallgemeinerung eines Satzes aus [FMM86] verwendet, der es erlaubt, eine Schranke für den parametrischen Abstand der Linearisierung zu der Originalfläche anzugeben.

6.2. Ausblick

Man konnte an einigen Stellen sehen, dass der Algorithmus nicht für alle Fälle gut geeignet ist. Insbesondere im Rahmen der Punktauswahl kann man sicher noch einiges verbessern. Das wird in diesem Abschnitt noch gemacht.

Der Grund, dass die Punktauswahl nicht immer gut ist, liegt daran, dass ein recht einfaches Verfahren dafür verwendet wurde, das nur über Längen bzw. Flächeninhalte der Triangulierung im Parameterraum Aussagen macht. Das eigentlich Interessante ist aber die Gestalt der Approximation im Modellraum.

Die vorgestellten Arbeiten im vorigen Abschnitt haben immer das Aussehen im Modellraum mitberücksichtigt. Insbesondere wäre die Punktauswahl von Cuillière [Cui98] leicht auf den in dieser Arbeit vorgestellten Algorithmus zu übertragen, da er auch eine a priori Punktauswahl trifft.

Eine andere Idee, wie man die Punktauswahl verbessern kann, ist an den Artikel von O'Rourke [O'R97] angelehnt, in dem er unter anderem beschreibt, wie man Punkte möglichst „gleichmäßig" auf der Kugel verteilen kann. Ein Vorschlag ist hierbei, alle Punkte mit einer „elektrischen Ladung" zu versehen und dann in einem iterativen Prozeß die abstoßenden Kräfte zu simulieren und die Punkte entsprechend auf der Oberfläche zu verschieben; d.h. es wird der schließlich Zustand angenommen, in dem die Energie am kleinsten ist und somit die Punkte möglichst gleichmäßig verteilt sind.

Diese Vorgehensweise muss man nur auf die übrigen Oberflächentypen ausweiten; beim Kegel/Zylinder muss man dabei natürlich beachten, dass sie in einem Parameter nicht begrenzt sind. Deswegen müsste man verhindern, dass die Punkte ins Unendliche verschwinden. Das könnte man aber erreichen, indem man keine Punkte über einen festen Rand hinaus bewegt.

Der Algorithmus, so wie er in dieser Arbeit vorgestellt wurde, ist nicht in der Lage irgendwelche Qualitätskriterien (wie z.B. Schranke für die Hausdorff-Distanz der Approximation) zu garantieren. Dies liegt unter anderem auch daran, dass der Triangulierungsalgorithmus keine Garantien dafür gibt, in welchen Fällen die geforderten Qualitätskriterien eingehalten werden können. Auch sonst wurde hauptsächlich nach heuristischen Ergebnissen vorgegangen, in der Hoffnung, dass die Ergebnisse nicht zu schlecht werden.

Es ist aber für einige Anwendungsfälle sicher nicht uninteressant, eine garantierte Genauigkeit zu erzielen. Dies könnte in einer verbesserten Punktauswahl berücksichtigt werden.

Abschließend kann man sagen, dass der vorgestellte Algorithmus für die behandelten Fälle ganz gute Ergebnisse liefert. Es besteht aber durchaus noch Verbesserungsbedarf. Insbesondere eine Erweiterung des Algorithmus um neue Oberflächentypen macht eine etwas durchdachtere Punktauswahl erforderlich. In dieser Richtung kann man sicher noch einiges erreichen.

A. Beispiele

In diesem Anhang sind einige Beispielbilder zu finden, die mit dem in dieser Arbeit vorgestellten Algorithmus approximiert wurden. Sie sollen den Algorithmus verdeutlichen und werden größtenteils ohne weitere Bemerkungen präsentiert.

Die ersten Bilder sind nach verschiedenen Oberflächentypen aufgeteilt. Dabei ist immer nur eine gekrümmte Oberfläche beteiligt. Diese Abbildungen bestehen immer aus drei Teilen: dem Eingabepolygon im Parameterraum, der Triangulierung im Parameterraum und der Approximation im Objektraum.

In Abschnitt A.5 sind dann einige größere Beispiele zu sehen. Sie bestehen auch aus mehreren gekrümmten Oberflächen. Aus diesem Grund wurde auf die Darstellungen des Parameterraums verzichtet. Allerdings sind die Objekte mit verschiedenen Genauigkeitsparametern berechnet worden.

A.1. Zylinder

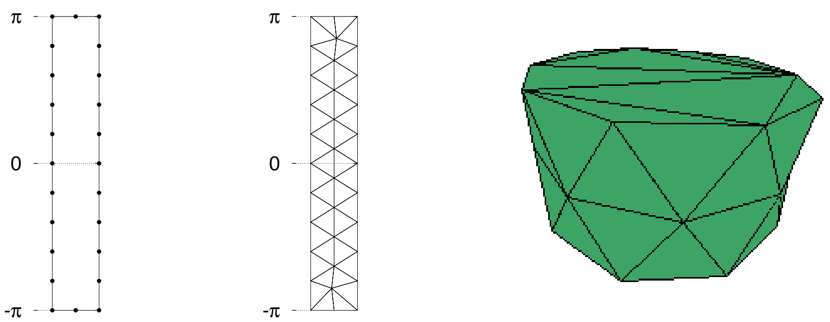


Abbildung A.1.: Zylinder mit Höhe und Radius 1; Genauigkeitsparameter $\rho = 0.05$.

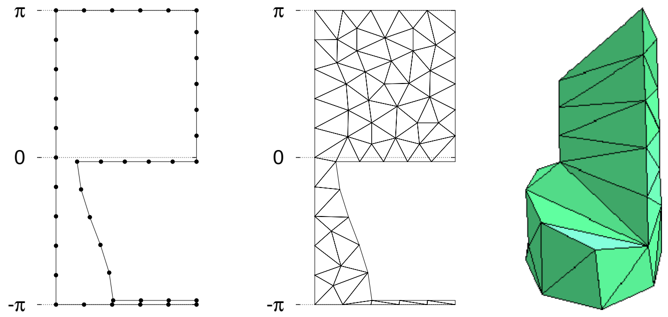


Abbildung A.2.: Zylinder mit Schnitt.

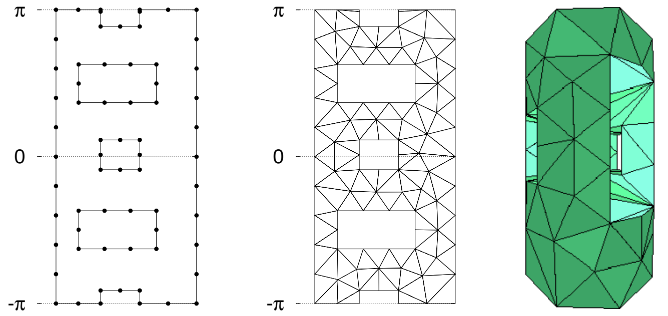


Abbildung A.3.: Zylinder mit zwei Durchbohrungen.

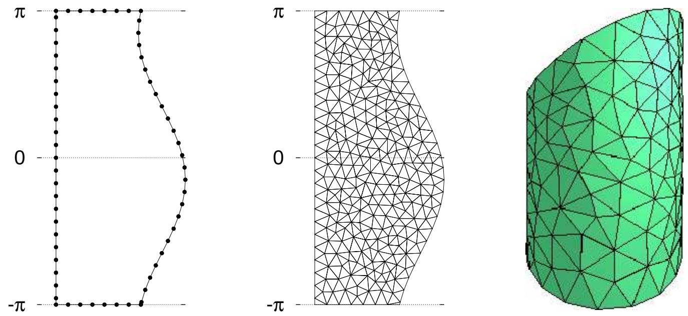


Abbildung A.4.: Schräg abgesägter Zylinder.

A.2. Kegel

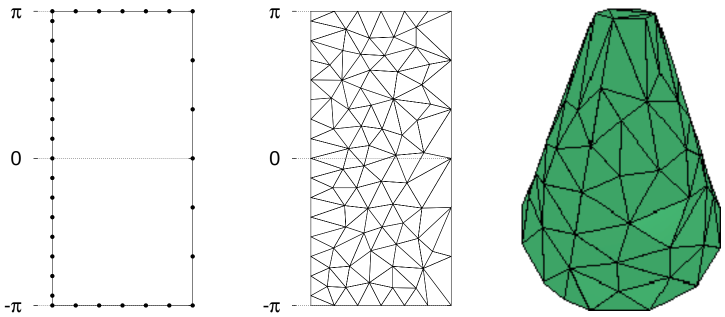


Abbildung A.5.: Kegelstumpf.

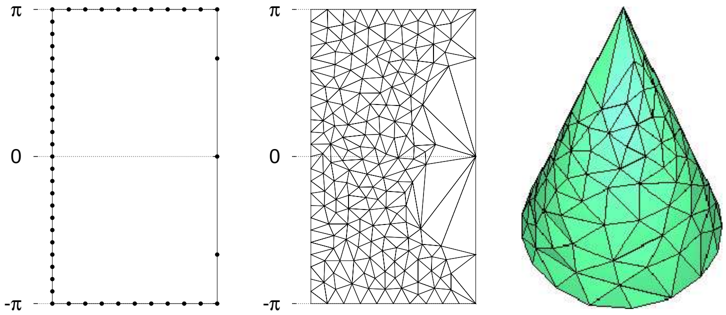


Abbildung A.6.: Kegel mit Spitze.

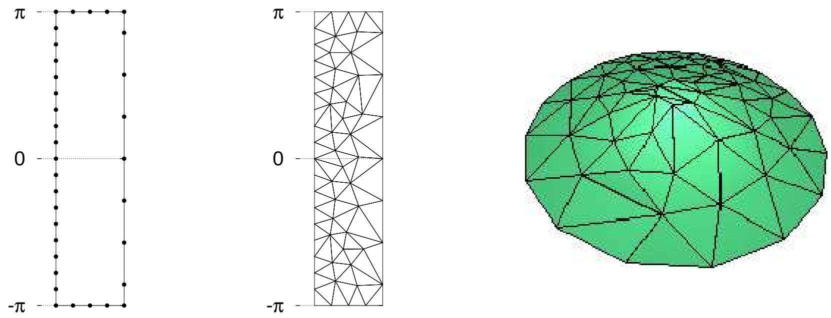


Abbildung A.7.: Sehr flacher Kegel.

A.3. Tori

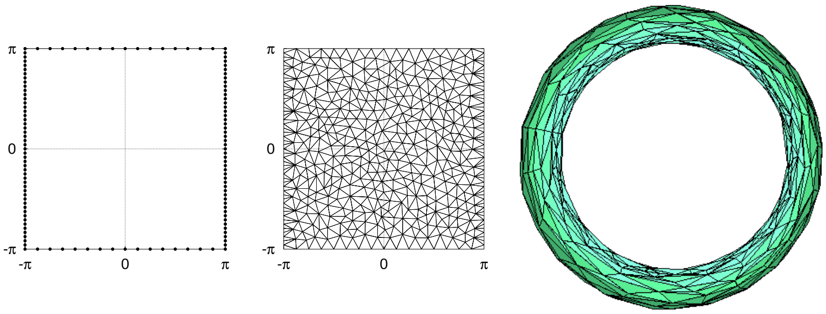


Abbildung A.8.: Torus mit kleinem Nebenradius.

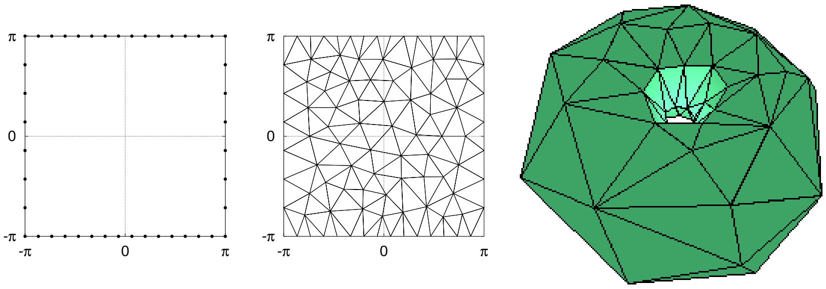


Abbildung A.9.: Torus mit großem Nebenradius.

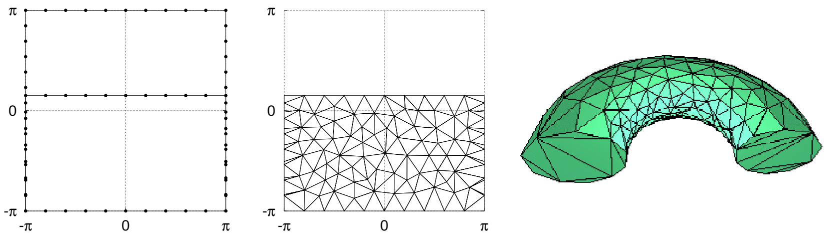


Abbildung A.10.: Ein „Henkel“.

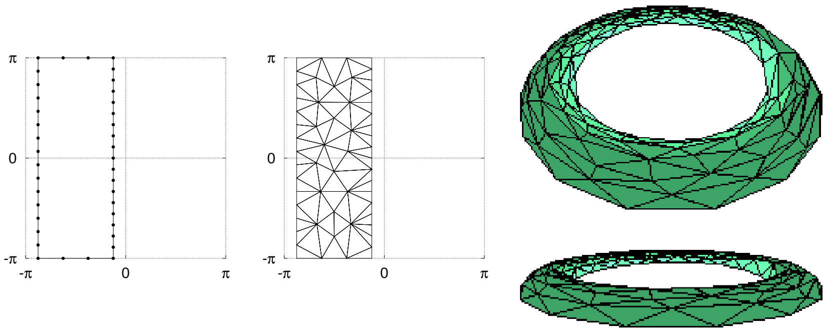


Abbildung A.11.: Eine „Toruskappe“

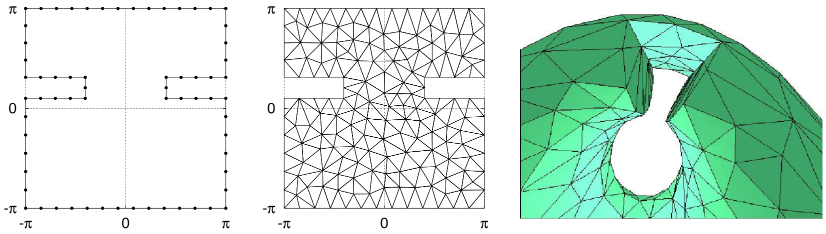


Abbildung A.12.: Torus mit Aussparung.

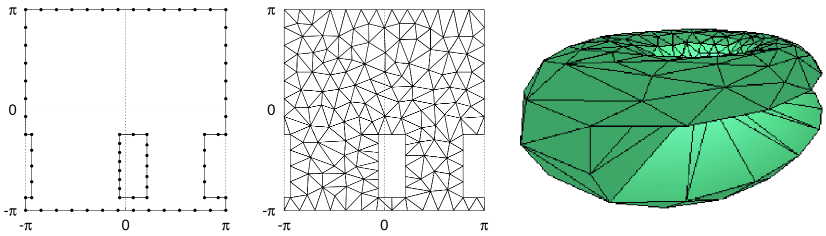


Abbildung A.13.: Torus mit Durchbohrung.

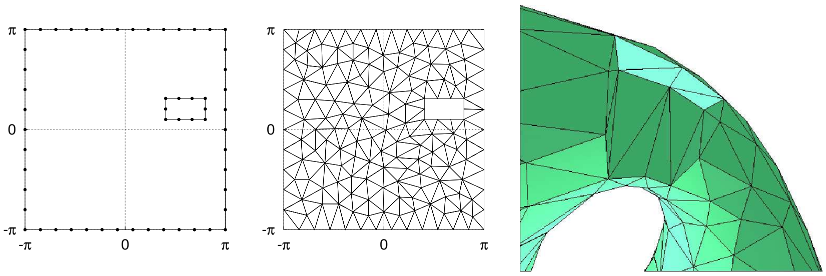


Abbildung A.14.: Torus mit „Fräsung“.

A.4. Kugeln

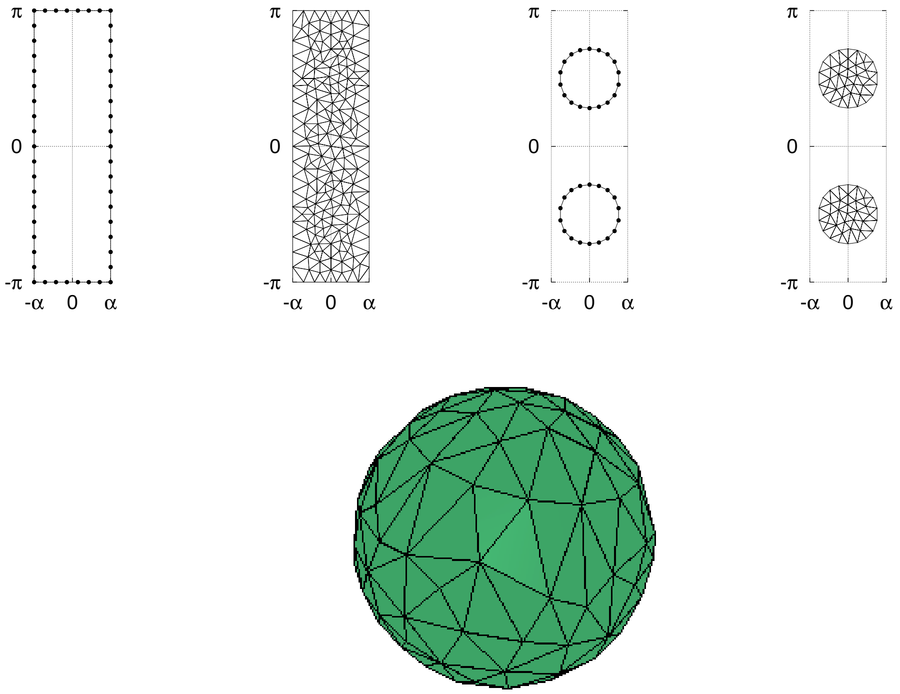


Abbildung A.15.: Einheitskugel; Genauigkeitsparameter $\rho = 0.01$.

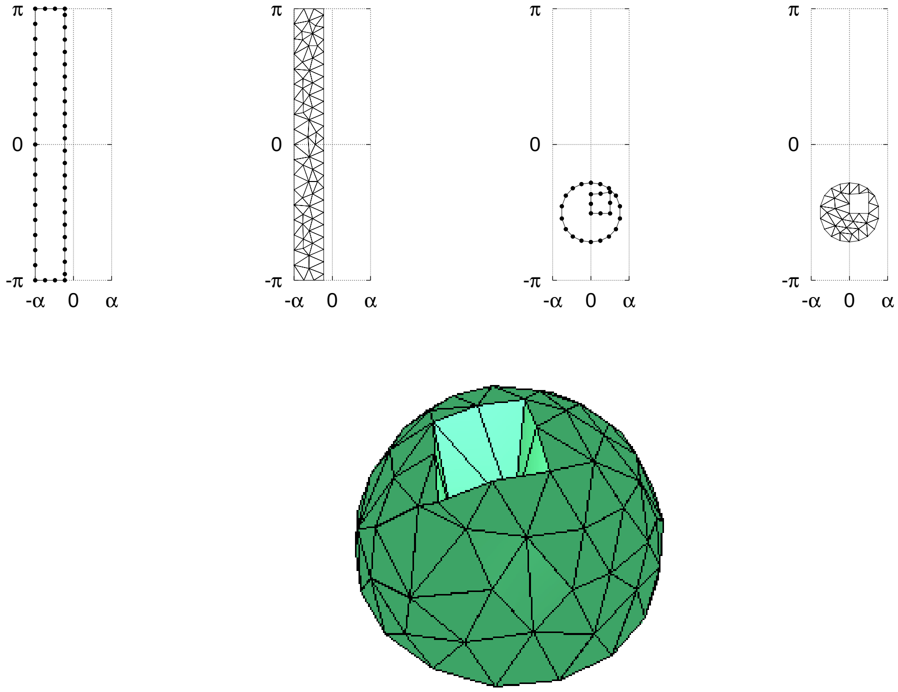


Abbildung A.16.: Kugelkappe mit Loch.

A.5. Weitere Beispiele

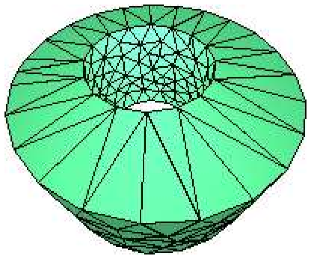
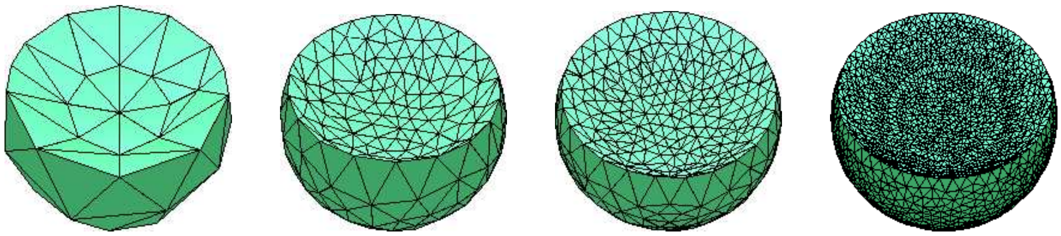
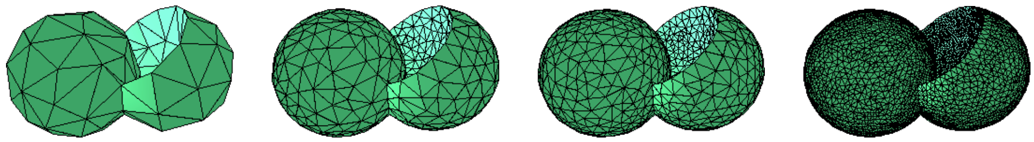


Abbildung A.17.: Zylinder mit zylindrischem Loch.



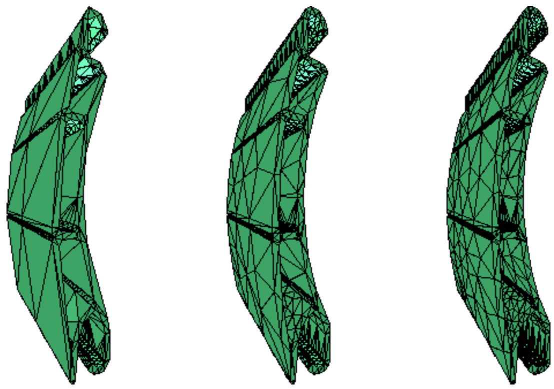
(a) $\rho = 0.05$ (b) $\rho = 0.01$ (c) $\rho = 0.005$ (d) $\rho = 0.001$

Abbildung A.18.: Kugel mit kugelförmiger Ausfräsung.



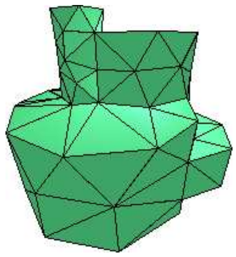
(a) $\rho = 0.05$ (b) $\rho = 0.01$ (c) $\rho = 0.005$ (d) $\rho = 0.001$

Abbildung A.19.: Komplexes Beispiel mit Kugeln.

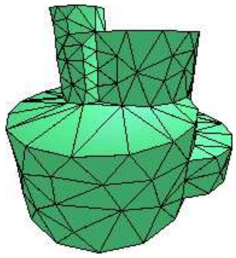


(a) $\rho = 0.5$ (b) $\rho = 0.1$ (c) $\rho = 0.05$

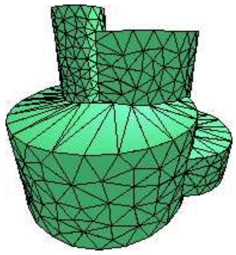
Abbildung A.20.: Ein Rollladenprofil (schräg von der Seite).



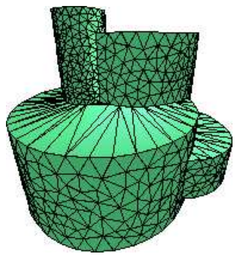
(a) $\rho = 0.1$



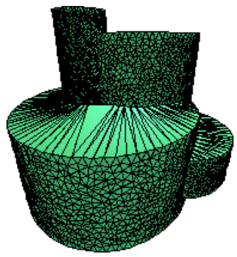
(b) $\rho = 0.03$



(c) $\rho = 0.01$



(d) $\rho = 0.005$



(e) $\rho = 0.001$

Abbildung A.21.: Komplexes Beispiel mit Zylindern.

Literaturverzeichnis

[ABGW90] H. Alt, J. Blömer, M. Godau, and H. Wagener. Approximation of convex polygons. In M. S. Paterson, editor, *ICALP 90 Automata, Languages and Programming: 17th international Colloquium, Warwick University, England, July 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 703–716, Berlin;Heidelberg;New York, 1990. Springer.

[Bar89] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Comput. Graph*, 23(3):223–232, 1989. Proc. SIGGRAPH '89.

[BE95] M. W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 47–123. World Scientific, second edition, 1995.

[Che87] L. P. Chew. Constrained Delaunay triangulations. In D. Wood, editor, *Proceedings of the 3rd Annual Symposium on Computational Geometry (SCG '87)*, pages 215–222, Waterloo, ON, Canada, June 1987. ACM Press.

[Che93] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 274–280. ACM, May 1993.

[Cui97] J.-C. Cuillière. A direct method for the automatic discretization of 3D prametric curves. *Computer-Aided Design*, 29(9):639–647, 1997.

[Cui98] J.-C. Cuillière. An adaptive method for the automatic triangulation of 3D prametric surfaces. *Computer-Aided Design*, 30(2):139–149, 1998.

[DM91] A. Dolenc and I. Mäkelä. Optimized triangulation of parametric surfaces. Tko-b74, University of Technology, Helsinki, 1991.

[FMM86] D. Filip, R. Magedson, and R. Markot. Surface algorithm using bounds on derivatives. *Computer Aided Geometric Design (CAGD)*, 3:295–311, 1986.

[FP94] T.-P. Fang and L. A. Piegl. Algorithm for constrained Delaunay triangulation. *The Visual Computer*, 10:255–265, 1994.

[Fuh98] A. Fuhrmann. Approximation konvexer Polyeder in der Hausdorff-Metrik. Diplomarbeit, Universität des Saarlandes, Juli 1998.

[FvDFH96] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice, second edition in C*, chapter 12, pages 533–562. Addison-Wesley, Singapore;Tokyo;Madrid, 1996.

[HKL+99] G. Hotz, A. Kerzmann, C. Lennerz, R. Schmid, E. Schömer, and T. Warken. SiLVIA – a simulation library for virtual reality applications. In *Virtual Reality '99 Conference*, page 82. IEEE, March 1999.

[JCL97] S. Jayaram, H. I. Connacher, and K. W. Lyons. Virtual assembly using virtual reality techniques. *Computer-Aided Design*, 29(8):575–584, 1997.

[Len99] C. Lennerz. Impulsbasierte Dynamiksimulation starrer Körper unter Verwendung von Hüllkörperhierarchien. Diplomarbeit, Universität des Saarlandes, 1999. in Vorbereitung.

[Mir96] B. Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD thesis, University of California, Berkeley, 1996.

[O'R97] J. O'Rourke. Computational geometry column 31. *International Journal of Computational Geometry and Applications*, 7(4), 1997.

[PT98] L. A. Piegl and W. Tiller. Geometry-based triangulation of trimmed NURBS surfaces. *Computer-Aided Design*, 30(1):11–18, 1998.

[Rup95] J. Ruppert. A Delauny refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, May 1995.

[SH92] X. Sheng and B. E. Hirsch. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design*, 24(8):437–444, 1992.

[She96] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133. Association for Computing Machinery, May 1996.

[Spa96] Spatial Technology Inc., 2425 55th Street, Building A, Boulder, Colorado 80301-5704. *ACIS Save File Format Manual*, 2.0 edition, 1996.

[War99] T. Warken. Berechnung von Kontaktkräften für eine zwangsbasierte Dynamiksimulation. Diplomarbeit, Universität des Saarlandes, April 1999.