Diplomarbeit

# Generierung hybrider Gitter für die Strömungssimulation auf komplexen anatomischen Geometrien

29.6.2009

Bertram Wölk

Betreuer: Prof. Dr. Marc Alexa, Technische Universität Berlin
Dr. Stefan Zachow, Konrad Zuse Institut Berlin

Technische Universität Berlin, Fachgebiet Computer Graphics
Institut für Technische Informatik und Mikroelektronik
Fakultät IV - Elektrotechnik und Informatik

## Erklärung

Hiermit versichere ich, Bertram Wölk, dass ich die vorliegende Diplomarbeit ohne fremde Hilfe und nur unter Verwendung der angegebenen Hilfsmittel verfasst habe.

---

Berlin, den 29.6.2009

## Zusammenfassung

Basierend auf einem vorhandenen Ansatz zur Einführung von anisotropen Tetraedern im Randbereich eines reinen Tetraedergitters wird ein Gittergenerator für hybride Gitter implementiert. Das hybride Gitter besteht in Randnähe primär aus anisotropen Prismen und im Inneren der Geometrie aus isotropen Tetraedern. Eine erhöhte Auflösung im Randbereich soll zu besseren Ergebnissen von numerischen Strömungssimulationen führen, für welche eine problemangepasste Diskretisierung des zu untersuchenden Gebietes benötigt wird. In dem zuvor genannten Ansatz wird eine Reihe von Übergangselementen vorgeschlagen, die an scharfen Kanten der Oberfläche platziert werden sollen. Im Rahmen dieser Diplomarbeit wird die Idee der Übergangselemente aufgegriffen und bei hybriden Gittern eingesetzt, um auch komplexe Eingabegeometrien vergittern zu können. Der ursprüngliche Gittergenerierungprozess wird überarbeitet und erweitert. Eine neue Menge an Übergangselementen wird eingeführt, es werden gekrümmte Extrusionsvektoren verwendet und es wird die Auswertung der medialen Oberfläche vorgenommen, um Überschneidungen im hybriden Gitter zu vermeiden. Der Gittergenerator wird als Modul in das Visualisierungs- und Analyseprogramm Amira implementiert und die erstellten hybriden Gitter werden auf ihre Elementqualität und die Güte der Strömungssimulationsergebnisse hin überprüft.



## Abstract

Based on an existing approach for the introduction of anisotropic tetrahedra near the surface boundary of a tetrahedral grid a grid generator for hybrid grids is implemented. The hybrid grid consists near the surface boundary primarily of anisotropic prisms and inside the geometry of isotropic tetrahedra. An increased resolution near the boundary should lead to better results of numerical flow simulations, which needs a problem specific discretization of the analyzed domain. In the aforementioned approach a set of transition elements is suggested, which should be placed at sharp surface corners. As a part of this diploma thesis the concept of using transition elements is applied for creating hybrid grids even for very complex input geometries. The initial grid generation process is revised and enhanced. A new set of transition elements is introduced, curved extrusion vectors are used and the medial surface is evaluated to avoid intersections in the hybrid grid. The grid generator is implemented as a module for the visualization and analysis tool Amira and the element quality of the generated hybrid grids and the quality of flow simulations performed on the grids are tested.

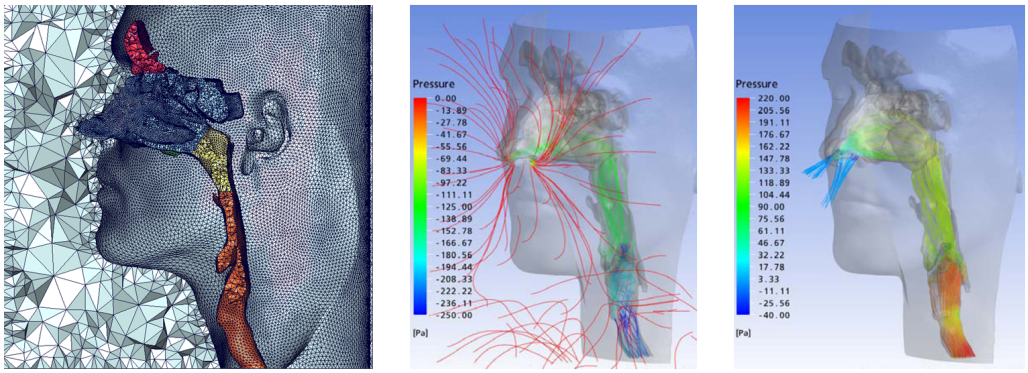
# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Problem

Die Gittergenerierung ist ein sehr wichtiger Schritt für die numerische Analyse mittels der Methode der finiten Elemente - von ihr hängt die Qualität der späteren numerischen Simulation ab. Obwohl der Bereich der automatischen zwei- und dreidimensionalen Gittergenerierung in den letzten Jahrzehnten ausgiebig erforscht wurde, ist in vielen Fällen eine zeitaufwändige, manuelle Vor- und Nachbearbeitung der Gitter von Personen mit Simulationserfahrung und Fachwissen über die relevanten Gittereigenschaften erforderlich [Shimada, 2006]. Des Weiteren sind die Gittergenerierungsalgorithmen auf Modelle aus CAD-Programmen ausgerichtet und haben Schwierigkeiten mit komplexen, nichtmannigfaltigen Geometrien, wie sie bei anatomischen Strukturen im medizinischen Bereich anzutreffen sind.

Für die Luftströmungssimulation in den oberen Atemwegen zwecks Therapieplanung für die Rhinochirurgie ist es notwendig, die Anatomie als volumetrisches Gitter bestehend aus einer Menge von geometrischen Elementen (Hexaeder, Tetraeder, Prismen, Pyramiden, etc.) darzustellen (Abbildung 1.1). Für die komplexe, nichtlineare Strömungsanalyse ist ein hochqualitatives, hybrides Gitter gefordert, welches u.a. im Randbereich anisotrope Prismenschichten und im Inneren Tetraeder aufweist.




(a) (b) (c)

Abbildung 1.1: (a) Schnitt durch ein numerisches Volumengitter der oberen Atemwege; (b) Strömung beim Einatmen; (c) Strömung beim Ausatmen

## 1.2 Zielsetzung

Das am Konrad-Zuse-Zentrum (ZIB) entwickelte 3D-Visualisierungs-Framework Amira soll um ein Modul erweitert werden, welches eine triangulierte Oberfläche mit Prismenrandschichten versieht. Eine GUI soll die leichte Nutzerinteraktion und Auswahl von Rändern ermöglichen, auf denen Prismenschichten erstellt werden. Neben der Angabe der

Dicke der Randschicht soll auch das Abstandsverhältnis zwischen den einzelnen Prismenschichten wählbar sein. Da keine hybriden Gitter in Amira bearbeitet werden können, soll das fertige Gitter ins CGNS-Format exportiert werden.

Der Gittergenerierungsalgorithmus muss mit komplexen, nichtmannigfaltigen Geometrien zurechtkommen und ein hybrides Gitter mit minimaler Nutzerinteraktion erstellen. Dabei sollen Prismen auf einer vorgegebenen Dreiecksoberfläche erstellt und der verbleibende Raum mit Tetraedern gefüllt werden.

Die Fähigkeiten des Moduls sollen an Geometrien aus Projekten am ZIB dargestellt werden. Des Weiteren soll ein Vergleich von numerischen Ergebnissen, die mit einem Gitter des Moduls bestimmt wurden, mit analytisch gewonnenen erfolgen. Das Programm soll mit anderen kommerziellen Gittergeneratoren in Bezug auf die Elementqualität und die Qualität der numerischen Simulationsergebnisse verglichen werden.

Der Gittergenerator soll - unter Verwendung vorhandener Module zur Tetraedergenerierung, Gitterverfeinerung und Visualisierung - schließlich so in Amira integriert werden, dass der Nutzer ohne viel Vorwissen ausgehend von z.B. volumetrischen Bilddaten oder einer vorhandenen Oberfläche ein hochqualitatives, hybrides Gitter erstellen kann.

## 1.3 Beitrag und Aufbau der Arbeit

Als Basis für den in dieser Diplomarbeit implementierten Gittergenerator dient der von Garimella [1999] in seiner Doktorarbeit entwickelte Ansatz zur Einführung von anisotropen Tetraedern im Randbereich eines reinen Tetraedergitters. In diesem Ansatz wird eine Reihe von Übergangselementen vorgeschlagen, die an scharfen Kanten platziert werden sollen. Diese Übergangselemente wurden jedoch nicht in dem zur Doktorarbeit gehörenden Gittergenerator implementiert. Im Rahmen dieser Diplomarbeit wird die Idee der Übergangselemente aufgegriffen und bei hybriden Gittern eingesetzt, um auch komplexe Eingabegeometrien vergittern zu können. Der von Garimella [1999] vorgeschlagene Gittergenerierungprozess wird überarbeitet und erweitert. Es wird eine neue Menge an Übergangselementen eingeführt, gekrümmte Extrusionsvektoren verwendet und die Auswertung der medialen Oberfläche, um Überschneidungen im hybriden Gitter zu vermeiden, vorgenommen. Der Gittergenerator wird als Modul in das Visualisierungs- und Analyseprogramm Amira implementiert und die erstellten hybriden Gitter werden auf ihre Elementqualität und die Güte der Strömungssimulationsergebnisse hin überprüft.

Die Diplomarbeit gliedert sich wie folgt:

In Kapitel 2 *Grundlagen* werden zuerst - ausgehend von den Zellkomplexen - die Eigenschaften von Polygongittern erläutert. Begriffe, die häufig in den folgenden Kapiteln Verwendung finden, wie Nichtmannigfaltigkeit und Dreiecksstern, werden hier definiert. Nach der Darstellung der Polygongitter werden die unterschiedlichen Typen von Polyedergittern beschrieben, die für numerische Berechnungen genutzt werden. Warum anisotrope Elemente in Wandnähe des Gitters bei der Simulation von Strömung notwendig sein können, lässt sich im Abschnitt über die Strömungsmechanik nachlesen. Da sich die Gitterqualität auf die Strömungssimulationsergebnisse auswirkt, muss bei der Generierung des Gitters auf die Qualität der Elemente geachtet werden. Kriterien zur Beurteilung der Elementqualität werden im Abschnitt über die Gitterqualität beschrieben.

Das Kapitel 3 *Stand der Forschung* beschreibt die wichtigsten Forschungsergebnisse aus dem Bereich der anisotropen Gittergenerierung. Dabei wird nicht nur auf die Erzeugung

von hybriden Gittern eingegangen, sondern auch auf relevante Ergebnisse aus angrenzenden Bereichen.

Welche Vorgehensweise in dieser Diplomarbeit zur Generierung des hybriden Gitters genutzt wird, kann im Kapitel 4 über die *Generierung hybrider Gitter* nachgelesen werden. In diesem Kapitel wird auch auf bestimmte Aspekte der Implementierung des Gittergenerators eingegangen.

Eine Evaluation der vom Gittergenerator erzeugten Gitter wird im Kapitel 5 *Ergebnisse* vorgenommen. Neben Darstellungen von Randschichtgittern für einige Testgeometrien werden auch die Ergebnisse von Strömungssimulationen analysiert.

In Kapitel 6 *Diskussion und Ausblick* werden die Ergebnisse dieser Diplomarbeit diskutiert und Ansatzpunkte für zukünftige Entwicklungen am Gittergenerator vorgestellt.

# 2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die für das Verständnis der folgenden Kapitel nötig sind. In Abschnitt 2.1 werden ausgehend von den Zellkomplexen Begriffe eingeführt, um topologische Eigenschaften von Gittern genauer spezifizieren zu können. Abschnitt 2.2 beschreibt die Gitter- und Elementtypen, nach denen ein Rechengitter charakterisiert wird, und die Vor- und Nachteile, die ein jeweiliger Gittertyp besitzt. Da in Kapitel 5 Strömungssimulationen auf den vom Gittergenerator erstellten hybriden Gittern durchgeführt werden, wird in Abschnitt 2.3 eine Übersicht der wichtigsten Strömungseigenschaften gegeben. Das Grundlagenkapitel wird durch eine Erläuterung (Abschnitt 2.4) der für ein numerisches Gitter wichtigen Qualitätseigenschaften abgeschlossen.

## 2.1 Zellkomplexe

Für die rechnergestützte Strömungssimulation muss die zu untersuchende Geometrie samt des beinhalteten oder umliegenden Raumes diskretisiert werden. Das Gebiet wird in Zellen unterteilt: Oberflächen setzen sich aus Polygonen zusammen, während Volumen mit Polyedern gefüllt werden. Die entstehende Menge von Zellen wird als Gitter oder auch als Zellkomplex bezeichnet. In diesem Abschnitt sollen die topologischen Eigenschaften von Zellkomplexen erläutert werden, wie z.B. nichtmannigfaltige Nachbarschaften, die bei der Konzeption des Gittergenerators berücksichtigt werden müssen. Außerdem sollen Begrifflichkeiten beschrieben werden, die in den späteren Kapiteln häufige Verwendung finden.

Ein *Zellkomplex* $K$ besteht aus einer finiten Anzahl von $n$-dimensionalen Zellen:

$$K = \bigcup \{\sigma : \sigma \text{ ist eine Zelle}\}$$

Die Dimension $n$ des Komplexes ist gleich der Zelle mit der höchsten Dimension. Eine *Zelle* ist dabei eine Menge, dessen Inneres homeomorph zu einer $n$-dimensionalen Scheibe ist. Der Zellrand setzt sich aus Zellen mit niedrigerer Dimension zusammen, die Facetten der $n$-dimensionalen Zelle genannt werden. Dabei schreibt man $\sigma < \tau$, wenn die Zelle $\sigma$ eine Facette der Zelle $\tau$ ist. Ein Polyeder ist z.B. ein 2-Zellkomplex, welcher Polygone (2-dimensionale Zellen), Kanten (1-dimensionale Zellen) und Knoten (0-dimensionale Zellen) als Facetten besitzt [Kinsey, 1993].

Um Komplexe zu formen, werden Zellen mit gleicher Dimension miteinander verbunden: Knoten werden mit Knoten vereint und Kanten mit Kanten. Dabei sind Einschränkungen zu beachten, um einen validen Zellkomplex bzw. valides Gitter zu erhalten: So sollen z.B. dreieckige nicht mit viereckigen Facetten zusammengefügt werden.

$|K|$ bezeichnet die Menge aller Punkte vom Komplex $K$:

$$|K| = \{x : x \in \sigma \in K, \sigma \text{ ist eine Zelle in } K\}$$

$|K|$ wird auch als unterliegender Raum von $K$ bezeichnet. Um eine topologische Struktur für Komplexe zu definieren, muss für jeden Punkt in $|K|$ eine Nachbarschaft feststehen. Wie sich die Nachbarschaft von einem Punkt in einem Komplex ergibt, soll für 2-Komplexe gezeigt werden. 2-Komplexe setzen sich aus Polygonen zusammen, die über Kanten und Vertices miteinander verbunden werden, wobei zu beachten ist, dass ein Punkt im Komplex zu den Punkten der Zellen korrespondiert, aus denen er konstruiert wurde. Bei 2-Komplexen unterscheidet man zwischen Punkten im Inneren des Polygons, Punkten auf einer Kante und Vertices.

Für einen Punkt, der in einem Polygon liegt, ist die Nachbarschaft des Punktes eine beliebige Scheibe, die vollständig im Inneren des Polygons enthalten ist. Liegt der Punkt $x$ auf einer Kante $e$, die ursprünglich durch Vereinen von mehreren anderen Polygonkanten $e_0 \dots e_n$ entstanden ist, so muss der zu $x$ korrespondierende Punkt $x_i$ auf jeder Kante $e_i$ gefunden werden. Das Vereinen der Halbkreisnachbarschaften der Punkte $x_0 \dots x_n$ ergibt die Nachbarschaft vom Punkt $x$ (Abbildung 2.1). Handelt es sich bei dem Punkt $x$ um einen Vertex, der sich aus den Polygonvertices $x_0 \dots x_n$ ergeben hat, so setzt sich die Nachbarschaft von $x$ aus den Kreissektoren der Vertices $x_0 \dots x_n$ zusammen.

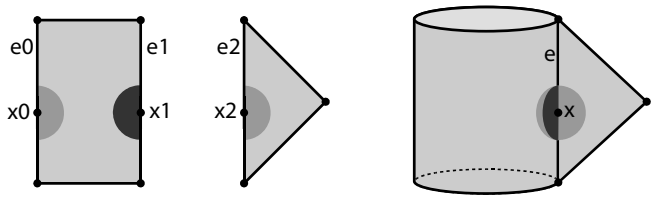



Abbildung 2.1: Zusammensetzung der Nachbarschaft eines Punktes $x$ auf einer Kante $e$ [Kinsey, 1993].

Durch die Einführung des Nachbarschaftsbegriffes können Zellkomplexe nach den Kriterien, die im nächsten Abschnitt genannt werden, als mannigfaltig oder nichtmannigfaltig kategorisiert werden.

### 2.1.1 Oberflächen

Eine $n$-dimensionale *Mannigfaltigkeit* ist ein topologischer Raum, in dem jeder Punkt eine Nachbarschaft topologisch äquivalent zu einer $n$-dimensionalen offenen Scheibe besitzt. Eine 2-Mannigfaltigkeit wird oft auch als Oberfläche bezeichnet. Handelt es sich um eine Mannigfaltigkeit mit Rand, so existieren Punkte, die eine zur offenen $n$-dimensionalen Scheibe oder Halbscheibe topologisch äquivalente Nachbarschaft besitzen. Ein Punkt, der weder eine offene noch eine halboffene Scheibe als Nachbarschaft besitzt, wird als *nichtmannigfaltig* bezeichnet. Enthält eine Oberfläche mindestens einen nichtmannigfaltigen Punkt, so wird sie als nichtmannigfaltige Oberfläche deklariert (Abbildung 2.2).

Eine Mannigfaltigkeit mit oder ohne Rand kann entweder *orientierbar* oder *nicht orientierbar* sein. Dabei ist die Orientierbarkeit ein globales Kriterium, welches nicht durch lokale Betrachtung ermittelt werden kann. Stellt man sich vor, dass sich ein $n+1$-dimensionales Objekt auf der $n$-Mannigfaltigkeit bewegt und sich dabei auf einer Seite der lokalen Nachbarschaft befindet, so ist die Mannigfaltigkeit nicht orientierbar, wenn es einen Weg gibt, der zur gleichen zuvor besuchten Nachbarschaft führt - diesmal jedoch auf der anderen Seite. Die Fläche ist orientierbar, wenn kein wie zuvor beschriebener Pfad bzw. kein Möbius-Band vorzufinden ist [Edelsbrunner, 2001].

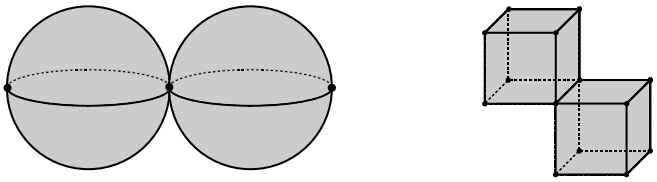


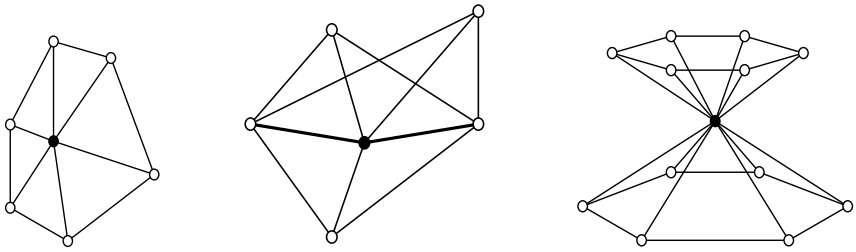
Abbildung 2.2: Die beiden dargestellten Zellkomplexe sind nichtmannigfaltig [Kinsey, 1993].

Die in dieser Arbeit vorkommenden Flächen sind orientierbare 2-Komplexe, die auch nichtmannigfaltig sein dürfen. Die 2-dimensionalen Zellen des Komplexes sind Dreiecke und Vierecke, die über Vertices oder Kanten verbunden sein können. Punkte oder Kanten, die nicht Teil eines Polygons sind, sollten nicht im Gitter auftreten. Aus der Graphentheorie lässt sich die Adjazenz- und Inzidenzrelation übernehmen. *Adjazenz* ist die Nachbarschaftsbeziehung zwischen gleichartigen Gitterelementen, während die *Inzidenz* zwischen unterschiedlichen Elementen definiert ist. So sind z.B. zwei Polygone adjazent zueinander, wenn eine Kante existiert, die zu beiden inzident ist. Über die Inzidenz lässt sich der Begriff des *Sterns* definieren: Der Stern eines Vertices beinhaltet alle zum Vertex inzidenten Kanten und Polygone. Dies kann allgemein für den Zellkomplex $K$ formuliert werden:

$$\text{Stern } \tau = \{\sigma \in K | \tau < \sigma\}$$

Die *Valenz* bezeichnet die Anzahl der Vertices, die zu einem betrachteten Vertex adjazent sind, also über eine Kante mit ihm verbunden sind.

Der Begriff der Mannigfaltigkeit auf Polygonoberflächen bezogen bedeutet, dass zu jeder Kante höchstens zwei Flächen inzident sind, während um einen Vertex nur ein einziger Ring aus Flächen existieren darf. Des Weiteren gilt, dass für mannigfaltige Flächen die Euler-Pointcaré-Formel erfüllt sein muss. Polygonflächen, die diese Eigenschaften nicht besitzen, sind nichtmannigfaltig (Abbildung 2.3). Nichtmannigfaltigkeiten in anatomischen Geometrien entstehen durch innere Flächen, die an der Schnittstelle von zwei unterschiedlichen Materialien auftreten.



(a) (b) (c)

Abbildung 2.3: (a) mannigfaltiger Dreiecksstern; (b) Nichtmannigfaltigkeit an einer Kante; (c) Nichtmannigfaltigkeit an einem Vertex

Bis jetzt wurde nur die Topologie bzw. die Konnektivität der Polygonoberfläche behandelt. Damit 3D-Objekte durch die Oberflächen repräsentiert werden können, muss eine *geometrische Einbettung* des Polygonnetzes angegeben werden. Dazu wird jedem Vertex

eine Position im $\mathbb{R}^3$ zugeteilt. Neben den Polygonnetzen, die aus Dreiecken und Vierecken bestehen, werden in dieser Diplomarbeit auch 3-Komplexe mit den Zelltypen Tetraeder, Pyramide, Prisma und Hexaeder verwendet. Die zuvor für 2-Komplexe aufgeführten Einschränkungen und Definitionen können leicht auf Polyedernetze erweitert werden.

## 2.2 Rechengitter

Im Abschnitt 2.1 wurden die Zellkomplexe und ihre topologischen Eigenschaften erläutert. Nutzt man Zellen zur Unterteilung eines physikalischen Gebietes in kleinere Untergebiete, um numerische Simulationen auf diesem Gebiet zu ermöglichen, so wird in der Literatur der Begriff des Gitters (engl.: Mesh, Grid), Rechengitters oder auch numerischen Gitters anstatt des Begriffes der Zellkomplexe verwendet. Der Prozess, in dem das Gitter entsteht, wird hierbei als Gittergenerierung bezeichnet [Lee und Xu, 2005]. Eine Unterteilung des zu untersuchenden Gebietes ist nötig, da partielle Differentialgleichungen, die Strömung oder Hitzeübertragung beschreiben, nur für sehr einfache Gebiete analytisch gelöst werden können. In komplizierteren Fällen muss daher zur numerischen Lösung das Gebiet unterteilt und die Gleichungssysteme diskretisiert werden. Die Lösung wird dann für jedes einzelne Untergebiet bzw. Element ermittelt [Shimada, 2006]. Üblicherweise wird die Methode der finiten Elemente, die Methode der finiten Volumen oder die Methode der finiten Differenzen verwendet, um die angenäherte Version des Gleichungssystems zu lösen. Dabei muss auf die Kontinuität der Lösung zwischen den Flächen zweier Elemente geachtet werden, damit ein späteres Zusammensetzen der einzelnen Lösungen ein genaues Ergebnis für das gesamte Gebiet liefert (Abschnitt 2.4: Gitterqualität).

Das Konzept, mit einem Gitter einen Raum zu diskretisieren, ist unmittelbar mit den ersten Versuchen verbunden, numerisch Lösungen für partielle Differenzialgleichungen zu finden. L. Richardson beschrieb um 1920 die Möglichkeit zur Verwendung der Methode der finiten Differenzen in der Meteorologie [Richardson, 1922]. Seine Erkenntnisse konnten jedoch erst mit der Erfindung des Computers sinnvoll genutzt werden. Ab diesem Zeitpunkt wurden Strömungssimulationen vermehrt computergestützt durchgeführt. Anfangs wurden noch Simulationen in 2D mit einfacher Berandung simuliert. Durch die steigende Rechenleistung konnten jedoch schon bald Strömungen auf komplexeren 3D-Gebieten berechnet werden. Durch die steigende Komplexität der zu untersuchenden Geometrien rückte der Prozess der Gittergenerierung - vormals eine kleine Nebenaufgabe bei numerischen Simulationen - in den Fokus. Man suchte nach Methoden, um die ursprünglich manuell erstellten Gitter automatisch generieren zu können. Der Bereich der Gittergenerierung wurde schnell zu einem eigenen Forschungsgebiet mit Einflüssen aus einer Reihe von Forschungsgebieten, speziell der Mathematik und Informatik. Heute existieren mehrere internationale Konferenzen, die sich ausschließlich dem Thema der Gittergenerierung und -adaptation widmen, außerdem finden sich auf fast jeder Konferenz zum Thema computergestützter Simulation Sessions zum Thema Rechengitter [Baker, 2005]. Als Konferenzen seien der *International Meshing Roundtable*[1], die *ISGG Conference on Numerical Grid Generation*[2], die *AIAA Computational Fluid Dynamics Conference*, das *AIAA Aerospace Sciences Meeting and Exhibit*[3] und das *Symposium on Trends in Unstructured Mesh Generation*[4] zu nennen.

[1] http://www.imr.sandia.gov/ (Stand: 28.10.2008)
[2] http://me-wiki.eng.uab.edu/isgg/ (Stand: 28.10.2008)
[3] http://www.aiaa.org/ (Stand: 28.10.2008)
[4] http://www.esc.sandia.gov/usnccm.html (Stand: 28.10.2008)

Im Folgenden werden strukturierte (Abschnitt 2.2.1) und unstrukturierte Gitter (Abschnitt 2.2.2) näher erläutert. Die Kombination aus beiden, das hybride Gitter, wird in Abschnitt 2.2.3 dargestellt. Neben der Beschreibung der Eigenschaften der verschiedenen Gittertypen sollen die unterschiedlichen Algorithmen zur Gittergenerierung auch auf die Möglichkeit zur Verwendung in dem zu implementierenden Gittergenerator untersucht werden.

Die Wahl eines bestimmten Gittertyps in Kombination mit bestimmten Elementtypen ist abhängig vom Anwendungsszenario. In die Entscheidung für einen bestimmten Gittertyp fließen Eigenschaften des zu lösenden physikalischen Phänomens, verfügbare Rechenzeit und gewünschte Genauigkeit, ggf. Ergebnisse aus zuvor durchgeführten Simulationen, Eigenschaften des eingesetzten numerischen Verfahrens sowie die Komplexität des zu diskretisierenden Gebietes ein. In allen Fällen wird angestrebt ein Gitter zu generieren, welches zu sehr genauen Simulationsergebnissen bei minimalem Speicherbedarf und minimaler Zeitkomplexität führt.

### 2.2.1 Strukturiert

Da der zu implementierende Gittergenerator semi-strukturierte Prismen im Randbereich generieren soll, ist zu überprüfen, ob Methoden zur strukturierten Gittergewinnung für die Erstellung der Randschicht verwendet werden können.

Bei strukturierten Gittern (Abbildung 2.4) ist die Knotenvalenz an allen Stellen im Gitter gleich (reguläre Konnektivität). Dadurch können die Gitterpunkte in einem festen Array gespeichert werden, welches wenig Speicherplatz erfordert, da auch die Nachbarschaftsbeziehungen nicht explizit gespeichert werden müssen. Der Nachteil einer solchen Datenstruktur ist, dass nachträgliches Verfeinern des Gitters durch das Hinzufügen von Punkten nicht möglich ist. Wird eine spätere adaptive Verfeinerung des Gitters gewünscht, muss dafür eine erweiterte Datenstruktur bereitgestellt werden, die dann jedoch den Vorteil der effizienten Speicherung der Konnektivität von strukturierten Gittern zunichte macht. Als Elementtypen in strukturierten Gittern sind vornehmlich Vierecke bzw. in 3D Hexaeder anzutreffen. Auch strukturierte Gitter aus Elementen, die eher in unstrukturierten Gittern Verwendung finden, wie z.B. Tetraeder, sind möglich jedoch eher unüblich. Der Grund dafür ist, dass Tetradergitter mehr Elemente benötigen, um ein Gebiet zu füllen, als Hexaedergitter. Da man bestrebt ist ein Gitter mit minimaler Elementanzahl zu erreichen, wird man ein Hexaedergitter - sofern dieses mit guter Qualität für das Gebiet generiert werden kann - einem Tetraedergitter vorziehen.

Um strukturierte Gitter zu erstellen, existieren eine Reihe von Gittergenerierungsverfahren. Bei der algebraischen Gittergenerierung werden die Knotenpositionen durch Interpolation bestimmt. Die algebraische Gittergenerierung ist schnell und bietet eine gute Kontrolle über die Verteilung der Randknoten, hat jedoch den Nachteil, dass im Gitter Unstetigkeiten vom Rand aus ins Innere propagiert werden, die zu Verzerrungen und Überschneidungen der Gitterlinien führen können [Thompson und Soni, 1999](Abbildung 2.5).

Die Generierung von strukturierten Gittern kann auch durch numerische Algorithmen erfolgen, in denen mit Hilfe von Differentialgleichungen basierend auf einem initialen Gitter die Knoten neu positioniert werden. Zur Generierung der Gitter kommen elliptische oder auch hyperbolische partielle Differentialgleichungen zum Einsatz. Die Probleme der algebraischen Gittergenerierung treten bei der elliptischen nicht auf. Die Übergänge im Gitter sind ebenmäßiger und Überschneidungen bei starker Krümmung sind unwahrscheinlicher.

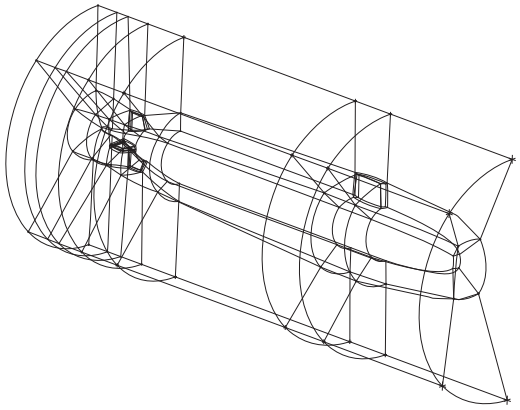


Abbildung 2.4: Zu sehen sind die groben Blöcke eines blockstrukturierten Gitters für eine U-Boot-Geometrie. Da das U-Boot symmetrisch ist, muss ein Gitter für nur eine Hälfte der Geometrie erstellt werden. Die Blöcke werden schließlich mit Hexaedern gefüllt, so dass ein strukturiertes Hexaedergitter entsteht.

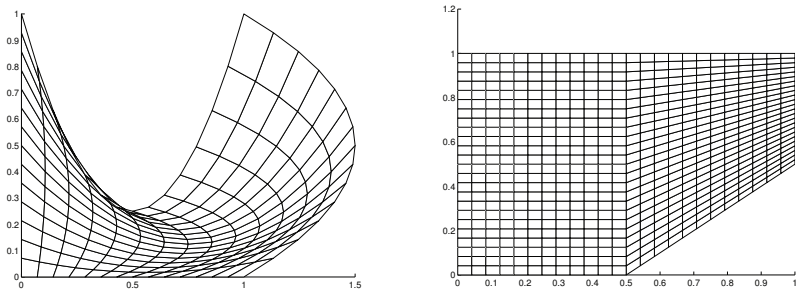



Abbildung 2.5: linkes Bild: Ist der Rand zu stark gekrümmt, treten Überschneidungen der Gitterlinien auf. Rechtes Bild: Unstetigkeiten werden vom Rand ins Gebietsinnere propagiert: Die Ecke ist auch bei zunehmender Distanz vom Rand deutlich zu erkennen.

Nachteilig ist der erhöhte Rechenaufwand und die etwaige Veränderung der Punktabstände auf den Rändern. Hinzu kommt der Kontrollverlust über die Gittereigenschaften, dem mit Einführung von Kontrolltermen in die Gleichungssysteme begegnet wird, jedoch bleibt die Schwierigkeit bei komplexen Gebieten, gute Gittereigenschaften über das gesamte Gitter zu erhalten [Baker, 2005].

Da die zuvor genannten Verfahren nur für einfache Berandungen ein Gitter mit guter Qualität ohne Überschneidungen liefern, müssen komplexere Gebiete durch Unterteilung in Untergebiete vereinfacht werden - in den Untergebieten kann nun wieder algebraisch oder numerisch ein Gitter erstellt werden. Dabei gibt es unterschiedliche Typen von zusammengesetzten Gittern, wie das *Overset-* (auch *Chimera-Gitter*), *Patch-* und *Multiblock-Gitter*, die sich dadurch unterscheiden, wie oder ob überhaupt Konnektivität zwischen benachbarten Zellen hergestellt wird [Thompson und Soni, 1999]. Ein Nachteil von Patch- und Mehrblockgittern ist der manuelle Aufwand, der nötig ist, die Blöcke an die vorhandene Geometrie anzupassen [Baker, 2005]. Eine mögliche Vorgehensweise, um die Gebietsdekomposition zu automatisieren, wird von Park und Lee [1998] beschrieben.

Eine weitere Möglichkeit zur Gittergenerierung ist das *Sweeping*, in der ein 2D-Gitter entlang einer Kurve extrudiert wird und dadurch ein 3D-Gitter entsteht. Das Verfahren kann nur auf bestimmte Geometrien angewandt werden, für die es möglich ist, ein Ausgangs- und Zielgitter mit gleicher Topologie anzugeben, welche durch einfache Oberflächen miteinander verbunden sind. Blacker [1996] führt mit dem *Cooper Tool* eine Methode ein, die den Sweeping-Ansatz verallgemeinert. Die Sweeping-Flächen können sich aufteilen und wieder verschmelzen - dadurch können auch komplexere Geometrien vergittert werden [Owen, 1998].

Durch die zuvor aufgezählten Nachteile der einzelnen Verfahren ist von einer Nutzung der Algorithmen zur Gewinnung einer Randschicht abzusehen. Die Hauptnachteile sind die nötige Unterteilung der Randschicht in einfache viereckige bzw. quaderförmige Bereiche, die Anfälligkeit der Algorithmen in Bezug auf komplexe Oberflächen und der geringe Automationsgrad der Methoden.

### 2.2.2 Unstrukturiert

Bei unstrukturierten Gittern (Abbildung 2.6) müssen die Knoten keine bestimmte Valenz besitzen (nichtreguläre Konnektivität) - beliebig viele Elemente können inzident zu einem Knoten sein [Owen, 1998]. Durch die nicht festgelegte Knotenvalenz haben die Gittergenerierungsalgorithmen mehr Freiheit bei der Elementplatzierung, welches dazu führt, dass auch komplexere Geometrien mit einem Gitter versehen werden können. Außerdem verfügen Algorithmen für unstrukturierte Gitter generell über einen hohen Automationsgrad und benötigen nur sehr wenige Eingabeparameter, welches auch unerfahrenen Nutzern mit geringem Zeitaufwand erlaubt, große Gebiete mit einem validen Gitter zu versehen.

Tetraeder und Dreiecke sind der am häufigsten für unstrukturierte Gitter verwendete Elementtyp, sie besitzen im Vergleich zu Vierecken und Hexaedern eine bessere Anpassungsfähigkeit an komplexe Oberflächen bei Wahrung von guter Elementqualität. Die etablierten Gittergenerierungsmethoden sind hierbei das *Octree-*, *Delaunay-* und *Advancing-Front-Verfahren*. Diese Verfahren können in 2D sowie in 3D vom Konzept her fast unverändert genutzt werden und unterscheiden sich dabei nur im Komplexitätsanstieg, wenn die 2D-Methoden auf den 3D-Fall verallgemeinert werden [Owen, 1998].

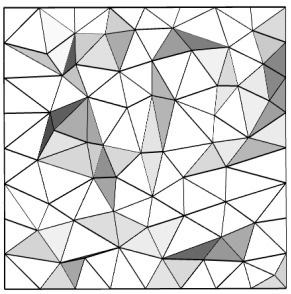


Abbildung 2.6: Schnitt durch ein Tetraedergitter, welches für einen Würfel erzeugt wurde

Auch Vierecke und Hexaeder können unstrukturierte Gitter formen. In *indirekten Verfahren* wird ein zuvor erstelltes Dreiecksgitter in ein Viereckgitter transformiert, während bei *direkten Verfahren* ohne Zwischenschritt mit Vierecken bzw. Hexaedern gearbeitet wird.

Ein Nachteil bei unstrukturierten Gittern ist der erhöhte Speicherplatzbedarf, der durch die explizite Speicherung der Nachbarschaftsrelationen hervorgerufen wird. Durch die unterschiedlichen Knotenvalenzen kann man über die Arrayposition eines Knotenpunktes nicht wie bei strukturierten Gittern auf die Nachbarknoten schließen. Des Weiteren ist die Einflussnahme auf den Generierungsprozess von unstrukturierten Gittern von Seiten des Nutzers nicht so einfach möglich wie bei strukturierten, bei denen man die Knotenverteilung über das gesamte Gebiet relativ gut kontrollieren kann. Über das Oberflächengitter kann das zu erstellende unstrukturierte, volumetrische Gitter indirekt beeinflußt werden. Hat man genaue Vorstellungen über die Elementgröße, Knotendichte oder den Grad der Anisotropie, können Verfahren wie von Shimada [2006] beschrieben angewandt werden. Nachteilig ist auch die erhöhte Anzahl an Elementen im Vergleich zu strukturierten Gittern (bei Verwendung der für strukturierte und unstrukturierte Gitter typischen Elemente), die zur Vergitterung eines Gebietes gebraucht werden. Dadurch wird der benötigte Speicherplatz und auch die Dauer der numerischen Simulation erhöht. Daher sollte nur bei komplexen Geometrien, für die kein strukturiertes Gitter erstellt werden kann, ein unstrukturiertes Gitter gewählt werden.

Im Folgenden wird das Advancing-Front-Verfahren näher erläutert, da es in dem zu implementierenden Gittergenerator zur Erstellung des Tetraedergitterkerns verwendet werden soll. Des Weiteren werden Methoden vorgestellt, wie Vierecke in ein Dreiecksgitter nachträglich eingefügt werden können. Außerdem wird ein Verfahren beschrieben, welches die mediale Oberfläche zur Gittergenerierung nutzt. Beides sind Themenbereiche, die für die Konzeption eines Gittergenerators für hybride Gitter von Interesse sind. Abschließend wird dargestellt, wie man anisotrope Elemente im Randbereich eines unstrukturierten Gitters erreichen kann.

#### Advancing-Front

In der *Advancing-Front-Methode* (auch *Moving-Front-Methode* genannt) wird das Gebiet ausgehend von der Randdreiecksfläche mit Tetraedern gefüllt. Ein Teil der Dreiecke der zuletzt erstellten Tetraeder bildet dabei die Front, von der ausgehend die optimale Position für einen neuen Punkt gesucht wird, der mit den schon vorhandenen Punkten der Front ein neues Tetraeder formen soll (Abbildung 2.7). Bilden die schon vorhandenen Punkte dabei die Möglichkeit ein Tetraeder mit besserer Qualität zu erstellen, wird auf das Einfügen eines neuen Punktes verzichtet. Neben der Suche nach einem neuen Tetrader mit optimaler

Qualität wird auch eine Reihe von Schnitttests durchgeführt, um ein Überlappen der aufeinander zulaufenden Fronten zu verhindern. Die Auflösung des Gitters kann in einigen Implementierungen gesteuert werden, so bestimmt der Nutzer in der Metode von Pirzadeh [1993a] die Elementbeschaffenheit durch ein Skalarfeld.

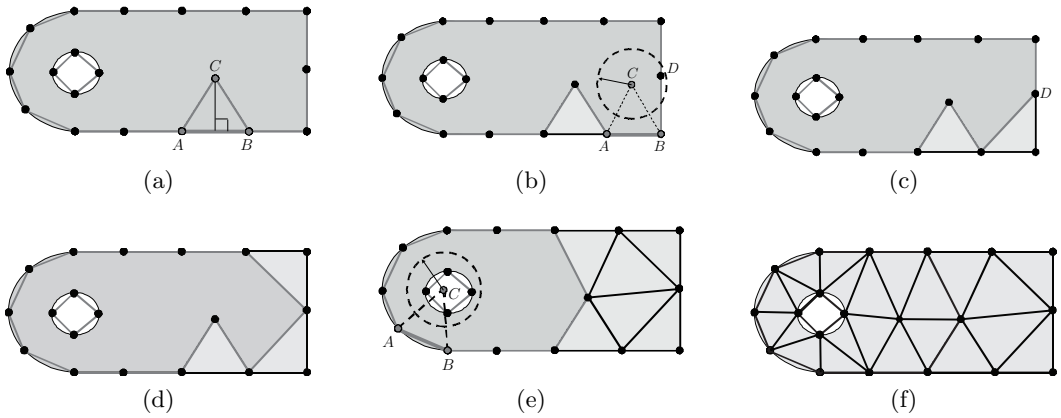



Abbildung 2.7: Schritte im Ablauf der Advancing-Front-Methode: (a) Ein isotropes Dreieck wird erstellt und die Kante $AB$ aus der Front entfernt, während $AC$ und $BC$ zur Front hinzugefügt werden. (b) Ein neues Element wird entweder durch Verbinden mit einem neu eingefügten oder durch Verbinden mit einem bestehenden Punkt gewonnen. Liegt kein Punkt in dem Auswahlradius, so wird ein neuer Punkt eingefügt. (c) Das neue Dreieck wurde mittels Punkt $D$ erstellt. (d) Ein weiteres Dreieck wurde hinzugefügt. (e) Liegen mehrere Punkte im Auswahlradius, so wird derjenige genommen, der beim Verbinden das Dreieck mit der besten Qualität liefert. (f) Das Advancing-Front-Verfahren ist abgeschlossen - es existieren keine Kanten mehr, die zur Front gehören.

Ein Problem dieses Verfahrens entsteht, wenn die Fronten aufeinander treffen und der noch freie Raum vergittert werden muss. In 2D gibt es eher keine Probleme den Hohlraum zu füllen, in 3D können jedoch beliebig komplexe Freiräume entstehen. Dem wird durch Backtracking und Füllmuster versucht zu begegnen [Baker, 2005]. Eine Kombination aus Advancing-Front- und Octree-Verfahren wurde von McMorris und Kallinderis [1997] entwickelt.

### Generierung von Vierecks- und Hexaedergittern

Auch Vierecke und Hexaeder können unstrukturierte Gitter formen. In *indirekten Verfahren* wird ein zuvor erstelltes Dreiecksgitter in ein Viereckgitter transformiert (Abbildung 2.8), während bei *direkten Verfahren* ohne Zwischenschritt mit Vierecken gearbeitet wird. Indirekte Verfahren sind schnell, da sie lokal arbeiten, besitzen jedoch eine schlechte Elementqualität und viele irreguläre Punkte, die im Gitter verbleiben. Die Anzahl von irregulären Punkten in unstrukturierten Gittern sollte minimal gehalten werden, da an diesen die inzidenten Elemente durch die Verzerrung eine schlechtere Qualität besitzen [Baker, 2005]. Statt die Dreiecke jeweils in drei Vierecke aufzuteilen, kann durch Kantenkippen (*Quad-Morphing*) ein Viereckgitter von besserer Qualität gewonnen werden [Owen, 1998].

In der direkten Methode von Schneiders u. a. [1994] wird automatisch ein Hexaedergitter erstellt. Bei komplexen Geometrien wird hierbei der Großteil der Geometrie mit einem

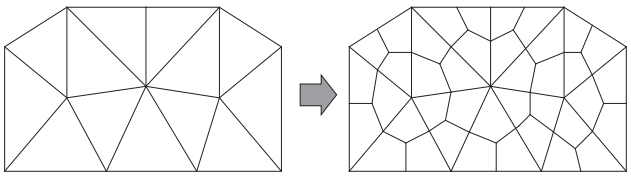


Abbildung 2.8: Durch Verbinden von Punkten im Mittelpunkt des Dreiecks und in den Mittelpunkten der drei Dreieckskanten entstehen aus einem Dreieck drei Vierecke.

strukturierten Hexaedergitter versehen, während der verbleibende Freiraum zur Oberfläche mit unstrukturiert angeordneten Hexaedern gefüllt wird. Eine weitere Möglichkeit ein Hexaedergitter zu erstellen, ist das *Plastering* bzw. *Paving*, welches dem Advancing-Front-Verfahren ähnelt. Beim Plastering wird das Gebiet der Reihe nach mit Elementen gefüllt, wie in Abbildung 2.9a dargestellt [Baker, 2005].

Die mediale Achse bzw. in 3D die mediale Oberfläche kann als Zwischenschritt genutzt werden, um automatisch ein Hexaedergitter zu generieren [Sheehy u. a., 1995; Baker, 2005]. Die mediale Achse kann als Sammlung von Linien und Kurven dargestellt werden, die durch Nachziehen des Mittelpunktes eines Kreises entstehen, wobei der Kreis durch die Geometrie gerollt wird und maximal viel Platz einnimmt (Abbildung 2.9b). Die dadurch entstehenden Unterteilungen des Gebietes in simplere Formen können nun z.B. mittels Plastering mit Vierecken bzw. Hexaedern gefüllt werden [Owen, 1998].

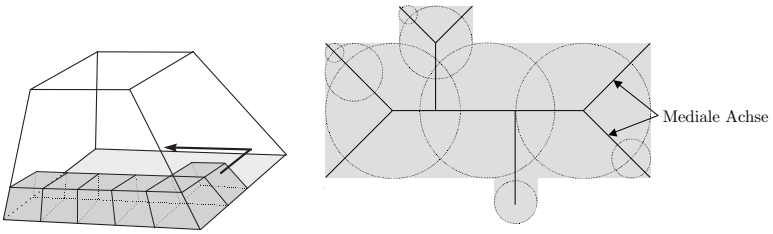



Abbildung 2.9: (a) Ablauf des Plastering-Verfahrens; (b) Dekomposition der Geomtrie in einfachere Untergebiete, die dann mit Vierecken gefüllt werden können

#### Anisotropie im Randbereich

In Bereichen, in denen Randschichtströmung vorliegt (Abschnitt 2.3.2), wird eine erhöhte Auflösung im Gitter benötigt, da sich der Geschwindigkeitsgradient in Richtung der Oberflächennormalen stark verändert. Eine Verfeinerung der Gitterauflösung tangential zur Oberfläche ist im Randschichtströmungsbereich jedoch nicht notwendig, da hier nur eine geringe Änderung in der Strömungsgeschwindigkeit vorliegt [Kallinderis und Ward, 1993]. Eine isotrope Verfeinerung des Gitters würde auch automatisch Verfeinerungen in anderen Richtungen nach sich ziehen, in denen eine hohe Auflösung des Gitters nicht notwendig ist. Der Einsatz von anisotropen Randelementen ist hier sinnvoll, jedoch können im Inneren der Geometrie wieder isotrope Elemente verwendet werden. Der anisotrope Bereich sollte hierbei weich in den isotropen Gitterbereich übergehen, da spontane Änderungen in der Elementgröße die Lösung verfälschen oder auch zur Instabilität des Lösers führen können. Die optimale Punkteverteilung kann am besten durch auf Fehlerabschätzung basierende, adaptive Verfeinerung erreicht werden. Trotz dieser Möglichkeit sollte

das Gitter schon vor der Verfeinerung eine problemangepasste Punkteverteilung aufweisen, damit die Iterationsanzahl der adaptiven Verfeinerung gering gehalten werden kann [Garimella, 1999].

Um wandnahe Anisotropie in Tetraedergittern zu erreichen, wurden mehrere Algorithmen konzipiert. Ein Überblick über die wichtigsten wird in Abschnitt 3.1 gegeben. Anisotropie im Randbereich mit geringerer Elementanzahl als bei reinen Tetraedergitterlösungen kann durch die Verwendung von hybriden Gittern erreicht werden. Ein Überblick über die Eigenschaften von hybriden Gittern soll im folgenden Abschnitt gegeben werden.

### 2.2.3 Hybrid

Hybride Gitter bestehen aus einem strukturierten und einem unstrukturierten Teil und kombinieren die guten Eigenschaften der beiden Gittertypen (Abbildung 2.10). Dies sind bei strukturierten Gittern die leichte Steuerbarkeit der Gittereigenschaften und die genaueren numerischen Simulationsergebnisse, während unstrukturierte Gitter größere Flexibilität bei komplexen Geometrien aufweisen. Gelegentlich werden die Begriffe hybrid und gemischte Gitter (engl.: Hybrid Grid, Mixed Grid) in der Literatur synonym verwendet, wobei sich der Ausdruck *gemischtes Gitter* darauf bezieht, ob unterschiedliche Elementtypen im Gitter verwendet wurden [Thompson und Soni, 1999]. Verwendet man statt des strukturierten Teils Prismen oder Hexaeder, die unstrukturiert angeordnet sind, so spricht man auch von semistrukturierten Gittern - das hybride Gitter besteht dann z.B. aus einem semistrukturierten Prismen- und einem unstrukturierten Tetraederteil.

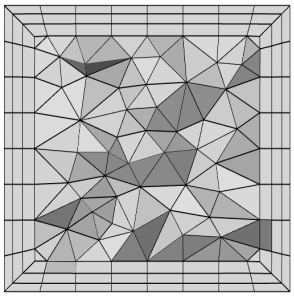


Abbildung 2.10: Schnitt durch ein hybrides Gitter, welches für einen Würfel erzeugt wurde

Wird ein Gitter mit anisotropen Gebieten im Randbereich für eine komplexe Geometrie benötigt, um die Grenzschichtströmung und Wandschubspannung adäquat zu erfassen, können - wie im vorigen Abschnitt beschrieben - auch Tetraedergitter genutzt werden. Die starken Gradienten normal zur Oberfläche erfordern jedoch so enge Gitterabstände in Randnähe, dass sich bei einer reinen Tetraedervernetzung eine nicht mehr akzeptable Anzahl von Elementen ergeben würde. In diesen Fällen ist der Einsatz eines hybriden Gitters von Vorteil - mit einem anisotropen, strukturierten Gitter im Randbereich, welches eine hohe Auflösung bei geringer Elementanzahl gewährleistet, und einem anpassungsfähigen, isotropen, unstrukturierten Gitter in den verbleibenden Bereichen [Kallinderis und Ward, 1993].

Eine ausführliche Übersicht von Verfahren zur Generierung von hybriden Gittern ist im Abschnitt 3.4 gegeben.

### 2.2.4 Elementtypen

Neben seiner Konnektivität wird ein Rechengitter mittels der verwendeten Elementtypen klassifiziert. Dabei werden in 2D Dreiecke und Vierecke verwendet, während in 3D Pyramiden, Prismen, Tetraeder und Hexaeder zum Einsatz kommen (Abbildung 2.11). Die 3D-Elemente besitzen 2D-Elemente als Begrenzungsflächen (Abschnitt 2.1: Zellkomplexe), die in einem Gitter Teil der Randfläche bzw. Teil von inneren Rändern sein können. Beliebige Polygone bzw. Polyeder können auch Gitter formen. Da in dieser Diplomarbeit, aufgrund der in der Einleitung genannten Rahmenbedingungen (Abschnitt 1), jedoch ein hybrides Gitter mit den üblichen, zuvor genannten Elementtypen erstellt werden soll, wird auf die Betrachtung von weiteren Elementtypen und Verfahren zur Generierung von Gittern, die diese allgemeinen Elementtypen verwenden, verzichtet.

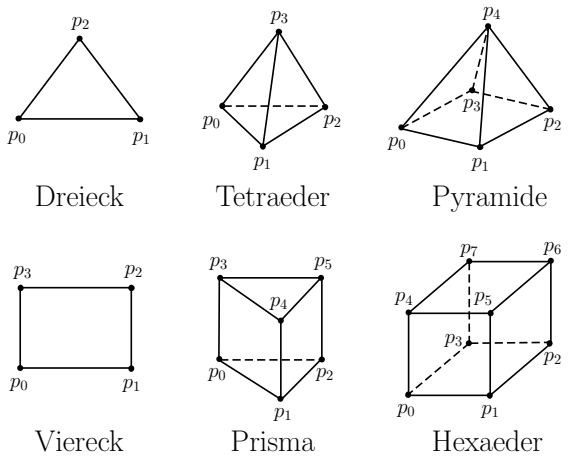



Abbildung 2.11: Elementtypen

## 2.3 Strömungsmechanik

Die Strömungslehre beschäftigt sich mit der Beschreibung und Vorausberechnung des kinematischen und dynamischen Verhaltens von Fluiden, weshalb sie auch als Fluid- oder Strömungsmechanik bezeichnet wird. *Fluid* ist dabei der Oberbegriff für Flüssigkeiten und Gase. Der Oberbegriff hat sich etabliert, weil in bestimmten thermodynamischen Zustandsbereichen keine klare Trennung zwischen einem flüssigem und einem gasförmigen Zustand möglich ist [Herwig, 2006]. Im Gegensatz zu einem Festkörper ist ein Fluid dadurch definiert, dass ein Fluidelement durch Schubspannungen verformt wird und nicht zur Ruhe kommt. In der technischen Strömungslehre kann man davon absehen, dass Fluide aus Molekülen bestehen. Man benutzt die Kontinuumshypothese, die besagt, dass die Masse stetig über das Volumen verteilt ist [Böswirth, 2007].

Die Strömungslehre ist ein kompliziertes Gebiet: Im Vergleich zur Massenpunktdynamik, wo einige Parameter genügen, um z.B. ein Momentbild einer Planetenbewegung zu bestimmen, benötigt ein Momentbild der Umströmung eines Körpers die Kenntnis der Geschwindigkeiten und Drücke in unendlich vielen Raumpunkten. Dies erklärt auch, warum Versuche in der Strömungslehre eine weitaus wichtigere Rolle einnehmen als in anderen Disziplinen wie z.B. in der Festkörpermechanik. In technischen Fragestellungen sind nicht so sehr die bewegten Teilchen von Interesse, sondern die ruhenden bzw. gleichförmig bewegten umströmten Körper (Autos, Flugzeuge).

Analog wie die Festkörpermechanik kann auch die Fluidmechanik in *Fluidstatik*, in welcher man sich mit ruhenden Fluiden befasst, und *Fluiddynamik*, in der bewegte Fluide Mittelpunkt des Interesses sind, eingeteilt werden. Im Gegensatz zur Festkörperstatik spielt die Fluidstatik eine weniger wichtige Rolle. Innerhalb der Fluiddynamik werden Anwendungsfälle durch die Nennung der auftretenden Strömungsaspekte (Abschnitt 2.3.1) und Strömungsphänomene (Abschnitt 2.3.2) näher charakterisiert.

### 2.3.1 Strömungsaspekte

Bei den Strömungsaspekten unterscheidet man zwischen laminarer und turbulenter, stationärer und instationärer, kompressibler und inkompressibler sowie viskoser und reibungsfreier Strömung.

#### Laminare und turbulente Strömung

Strömungen können *laminares* oder *turbulentes* Verhalten aufweisen. Bei laminarer Strömung folgen die einzelnen Fluidelemente ungestörten, glatten Strömungsbahnen, wobei die Impulsübertragung zwischen benachbarten Fluidbereichen durch molekulare Wechselwirkungen erfolgt. Turbulente Strömungen sind durch stark schwankende Strömungsgeschwindigkeiten gekennzeichnet, deren Schwankungskomponenten in alle drei Raumrichtungen weisen. Durch die Schwankungsbewegungen wird die Impulsübertragung stark erhöht, weil lokal große Geschwindigkeitsunterschiede auftreten. Dies führt z.B. dazu, dass turbulent umströmte Körper einen höheren Reibungswiderstand aufweisen als bei laminarer Umströmung [Herwig, 2006]. Turbulente Strömungen treten vor allem bei technischen Rohrströmungen und in Grenzschichten häufig auf, jedoch können auch freie Strömungen ohne Begrenzungswände Turbulenzen aufweisen [Böswirth, 2007].

Welche der beiden Strömungsformen vorliegt, entscheidet sich anhand der Reynolds-Zahl, einem dimensionslosen Parameter. Osborne Reynolds zeigte in einem Versuch, dass Strömungen eines bestimmten Fluids mit niedriger Geschwindigkeit in der Regel laminar sind, während bei hohen Geschwindigkeiten turbulente Strömungen vorliegen. Technisch relevante Strömungen sind fast immer turbulent [Herwig, 2008].

#### Stationäre und instationäre Strömung

Man unterscheidet zwischen *stationärer* und *instationärer Strömung*. Erstere liegt vor, wenn für einen ortsfesten Beobachter alle Strömungsgrößen zeitunabhängige Werte aufweisen. Lässt sich bei den zuvor beschriebenen Größen eine Zeitabhängigkeit feststellen, so liegt instationäre Strömung vor. Sonderformen von instationärer Strömung treten auf, falls die Strömungsgrößen ein Zeitverhalten aufweisen, welches periodisch ist oder sich asymptotisch einem stationären Verhalten annähert. Im letzten Fall spricht man von transienter Strömung. Bei technischen Anwendungen kommt man meist mit den einfacheren stationären Strömungen aus. Bei turbulenten Strömungen, die aufgrund der überlagerten Schwankungsbewegungen zunächst von Natur aus instationär sind, bezieht sich die Unterscheidung zwischen stationär und instationär auf eine über einen kurzen Zeitraum gemittelte Strömung [Herwig, 2006].

#### Kompressible und inkompressible Strömung

Neben Fluiden, welche die thermodynamische Eigenschaft besitzen, dass ihre Dichte mit dem Druck und der Temperatur stark veränderlich ist, existieren auch solche, deren Dichte fast gar nicht vom Druck bzw. der Temperatur abhängt. Dadurch können Fluide als kompressibel oder inkompressibel klassifiziert werden. Neben dem Fluid kann auch die Strömung als kompressibel bezeichnet werden, wenn durch die Strömung im Strömungsfeld Dichteunterschiede auftreten, die nicht vernachlässigt werden können. Existieren keine oder nur vernachlässigbar kleine strömungsbedingte Dichteunterschiede, so spricht man von inkompressibler Strömung [Herwig, 2008].

Die thermodynamische Eigenschaft eines Fluids ist nur eine notwendige Bedingung dafür, dass eine Strömung kompressibel ist. Erst wenn in dem Strömungsfeld tatsächlich erhebliche Dichteänderungen auftreten, liegt auch eine kompressible Strömung vor. Demnach kann es z.B. auch eine inkompressible Strömung eines kompressiblen Fluides geben.

#### Viskose und reibungsfreie Strömung

Zur Ausbildung einer Strömung kommt es, wenn Reibungs-, Druck- oder Volumenkräfte auf die einzelnen Fluidpartikel wirken. Soll das Fluid nicht in Ruhe oder gleichförmiger Bewegung verharren, muss mindestens eine dieser Kräfte wirken. Reibungskräfte treten auf, wenn Fluidpartikel eine Relativbewegung zueinander besitzen und sich aufgrund von molekularen Wechselwirkungen gegenseitig mitreißen. Aus makroskopischer Sicht ist dafür die Viskosität verantwortlich. Sind keine großen Relativbewegungen vorhanden und die Reibungskräfte sehr viel kleiner als die Druck- und Volumenkräfte, können die Reibungskräfte vernachlässigt werden. Reibungseffekte treten in realer Strömung immer auf, sind sie jedoch von weniger großer Bedeutung, so kann zur Modellierung die *reibungsfreie* Strömung genutzt werden.

Auch hier ist wieder die Viskosität eines Fluides als Stoffeigenschaft nur die notwendige Voraussetzung für das Auftreten einer reibungsbehafteten, also durch Reibungskräfte beeinflußten Strömung.

### 2.3.2 Strömungsphänomene

Das Verhalten von Strömungen kann neben den Strömungsaspekten noch durch die Angabe von Strömungsphänomenen genauer beschrieben werden.

#### Wandeinfluss und Grenzschichtströmung

Teile eines Strömungsfeldes sind meistens durch starre, unbewegliche und undurchlässige Wände begrenzt. Basierend auf den physikalischen Eigenschaften werden an den Wänden Randbedingungen für die Strömung formuliert. Die Fluidmoleküle stehen untereinander und mit den Molekülen der begrenzenden Wände in Wechselwirkung. Bei Gasen, deren Moleküle frei beweglich sind, besteht diese Wechselwirkung aus Stößen untereinander oder mit den Wandmolekülen. Da die Moleküle von Flüssigkeiten in einem flexiblen Gitterverband eingebunden sind, besteht die Wechselwirkung aus einer gegenseitigen Beeinflussung im Gitterverband benachbarter Flüssigkeits - bzw. Wandmoleküle. Makroskopisch führt

dies zu einer stetigen Verteilung aller Strömungsgrößen - somit auch der Strömungsgeschwindigkeit. An den Rändern von Strömungsgebieten, die durch feste Wände begrenzt sind, führt die Wechselwirkung mit den Wandmolekülen zu einem Verlauf der Geschwindigkeitsverteilung vom Wert Null an der Wand auf von Null verschiedene Werte. Diesen speziellen Aspekt des Strömungsverhaltens an der Wand nennt man Haftbedingung [Herwig, 2006].

Über den Mechanismus der Haftbedingung wird zwischen der Wand und der angrenzenden Strömung eine Schubspannung, die sog. Wandschubspannung, übertragen, die letztendlich zum Reibungswiderstand bei umströmten Körpern führt. Diese Wandschubspannung ist mit dem Geschwindigkeitsprofil unmittelbar an der Wand über Gleichung 2.1 verbunden. Wobei $\tau$ die Wandschubspannung, $(\partial u/\partial n)_W$ der Geschwindigkeitsgradient senkrecht zur Wand und $\eta$ die dynamische Viskosität des Fluids ist. Diese Größe ist eine Stoffeigenschaft und beschreibt die Fähigkeit des Fluides zur Impulsübertragung in einer Scherströmung.

$$\tau_w = \eta \left. \frac{\partial u}{\partial n} \right|_W \tag{2.1}$$

In Wandnähe liegt ein steiler Geschwindigkeitsanstieg vor, der sich bis zum Erreichen der Geschwindigkeit außerhalb der Wandnähe fortsetzt. Daraus ergibt sich das in Abbildung 2.12 dargestellte Geschwindigkeitsprofil. Dieser wandnahe Bereich wird Grenzschicht oder auch Randschicht genannt. Der Grenzschichtcharakter der Strömung wird umso ausgeprägter, d.h. der Wandabstand, in dem die Geschwindigkeit der Außenströmung erreicht wird, wird kleiner, je größer die Außengeschwindigkeit $u_\infty$ ist.

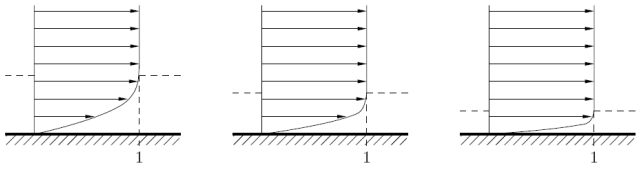



Abbildung 2.12: Geschwindigkeitsprofil mit steigendem $u_\infty$ in Wandnähe

Diese Strömungsgrenzschicht mit der Dicke $\delta$ kann getrennt vom übrigen Strömungsfeld betrachtet werden. In der Grenzschicht spielen Reibungseffekte eine entscheidene Rolle, während diese außerhalb der Grenzschicht vernachlässigt werden können. Zur Ausbildung einer Grenzschicht kommt es bei hoher Strömungsgeschwindigkeit bzw. ab einer bestimmten Reynolds-Zahl. Abhängig von der Reynolds-Zahl ist auch die Grenzschichtdicke. Die Reynolds-Zahl wird benutzt, um Strömungen in verschiedene Kategorien einzuteilen. Strömungen bei sehr kleinen Reynolds-Zahlen verhalten sich physikalisch sehr verschieden von solchen mit hoher Zahl, die den beschriebenen Grenzschichtcharakter besitzen. Die Grenzschichten sind in vielen Fällen extrem dünn, spielen aber trotzdem eine wesentliche Rolle, weil in ihnen die Reibungseffekte des Strömungsfeldes konzentriert sind. Der Reibungswiderstand eines Tragflügels kann z.B. nur korrekt ermittelt werden, wenn die Vorgänge in der meist wenige Millimeter messenden Grenzschicht richtig erfasst werden [Herwig, 2008].

Es existieren eine Reihe von physikalischen Modellen ohne Haftbedingung, wie z.B. die Potentialtheorie. In diesen Theorien liegt an der Wand ein Sprung von einer Geschwindigkeit von Null auf einen endlichen Wert vor. Dies bedeutet, dass Effekte im Zusammenhang mit der Gesamtverteilung der Geschwindigkeit im Strömungsfeld erfasst werden können,

aber Effekte, die auf Haftbedingungen basieren, können nicht beschrieben werden. Ohne die Haftbedingung wird keine Schubspannung an die Wand übertragen und somit ist der Reibungswiderstand eines umströmten Körpers gleich Null. Es handelt sich dann um eine Simulation mit reibungsfreier Strömung.

#### Ablösung

Bei der Ablösung handelt es sich um ein Grenzschichtphänomen. Wird ein Kreiszylinder umströmt, so kann die Strömungsgrenzschicht nicht beliebig weit der Körperkontur folgen, weil sie durch die Reibungseffekte kinetische Energie verloren hat und nicht mehr gegen den ansteigenden Druck anströmen kann. Dadurch folgt eine Ablösung vom Körper und es kommt zu komplexen, häufig instationären Rückstromgebieten im Strömungsfeld. Die Auswirkungen der Ablösung auf das gesamte Strömungsfeld sind erheblich. Man unterscheidet zwischen der druckinduzierten und der geometrieinduzierten Strömungsablösung (Abbildung 2.13), wobei letztere durch Kanten oder Stufen im Profil verursacht wird.

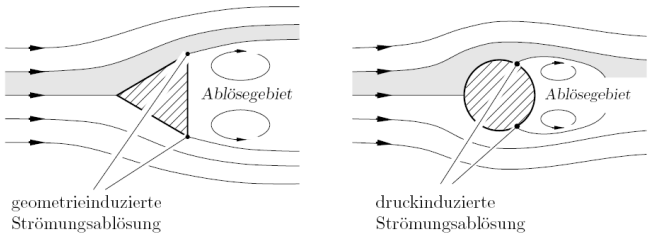



Abbildung 2.13: Geometrie- und druckinduzierte Strömungsablösung [Herwig, 2006]

## 2.4 Gitterqualität

In diesem Abschnitt soll der Begriff der Gitterqualität näher beschrieben werden, da Gitterqualitätskriterien bei der Generierung eines hybriden Gitters berücksichtigt werden müssen (Abschnitt 4: Methoden) und die Ergebnisse des Gittergenerators in dieser Diplomarbeit anhand von Qualitätskriterien bewertet und mit anderen Gittergenerierungstechniken verglichen werden sollen (Abschnitt 5: Ergebnisse).

Der Forschungsbereich der Gitterqualität beschreibt eine Menge von Kriterien, die zur Quantifizierung von Gittereigenschaften eingesetzt werden können - jedem Element wird dabei von dem Qualitätskriterium eine reelle Zahl zugewiesen. Dabei sind die Wertebereiche der Gitterqualitätskriterien, die eine gute Gitterqualität darstellen, abhängig vom spezifischen Anwendungsfall. Knupp [2007] definiert den Begriff der Gitterqualität für Simulationen, die die Lösung von partiellen Differentialgleichungen erfordern:

> Mesh Quality concerns the characteristics of a mesh that permit a particular numerical PDE simulation to be efficiently performed, with fidelity to the underlying physics, and with the accuracy required for the problem.

Knupp [2007] hebt in den Erläuterungen zu seiner Definition die folgenden Punkte hervor. Da die Gitterqualität abhängig von den durchzuführenden Berechnungen ist, ist daran zu denken, dass sich bei Verwendung des Gitters für eine andere physikalische Simulation auch die Anforderungen an die Ausprägungen der Gitterqualitätskriterien ändern. Des Weiteren

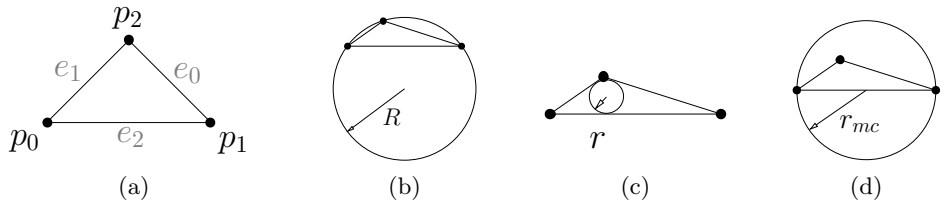
sollten die Gittereigenschaften nicht zu einer Senkung der Konvergenzgeschwindigkeit des Lösers führen - es sind Elemente zu vermeiden, die eine schlechte Matrixkonditionierung oder Cluster von großen Eigenwerten zur Folge haben. Das Gitter sollte zu genauen Simulationsergebnissen beitragen, indem es den lokalen und globalen Fehler bis unter eine angegebene Schranke senkt. Letztendlich muss das Gitter zusammen mit der ausgewählten Diskretisierungsmethode der partiellen Differentialgleichungen gewährleisten, dass sich der durch die Problemdiskretisierung bedingte Fehler in einem akzeptablen Rahmen hält [Knupp, 2007].

Qualitätskriterien besitzen ein breites Anwendungsfeld [Knupp, 2007], so können sie zur Ermittlung von Defekten im Gitter, wie invertierte Elemente, sehr große oder kleine Winkel oder fehlerhafte Topologie, eingesetzt werden. Wurden solche Defekte festgestellt, kann das Gitter verworfen oder es können die betroffenen Stellen repariert werden. Ein weiterer Verwendungszweck ist die zielgerichtete Abstimmung des Gitters auf die durchzuführende Simulation. Dies ist jedoch nur möglich, wenn Wissen über die für die Simulation positiven Gittereigenschaften vorliegt. Auch Gitteroptimierungstechniken nutzen Qualitätskriterien um zu beurteilen, ob die vorgenommenen geometrischen oder topologischen Veränderungen Vorteile in Bezug auf die Gitterqualität haben. Ein weiteres Anwendungsbeispiel sind Gittergeneratoren, die neuerstellte Elemente auf Qualität und Defekte überprüfen und anhand von Qualitätskriterien den weiteren Verlauf im Gittergenerierungsprozess bestimmen.

In den beiden folgenden Abschnitten 2.4.1 und 2.4.2 werden Dreiecks- und Viereckskriterien aufgeführt. Der Wertebereich der Qualitätskriterien ist - sofern nicht anders angegeben - so skaliert, dass das Einheitsdreieck und das Einheitsquadrat den Wert eins erhalten. Die Angabe bei den Qualitätskriterien, die für ein Element mit guter Qualität steht, ist auf das Ziel bezogen, ein möglichst isotropes Element zu erhalten. Möchte man die Qualität von anisotropen Elementen bewerten und soll der Wertebereich so skaliert werden, dass der Wert eins für ein Element mit optimaler Qualität steht, so können *explizite Qualitätskriterien* verwendet werden [George und Borouchaki, 1998; Frey und George, 2000]. Diese nutzen isotrope Qualitätskriterien, erhalten jedoch als Parameter ein anisotropes Referenzelement, um den Wertebereich skalieren zu können. Im Abschnitt 2.4.3 wird auf die Bestimmung der Qualität bei Tetraedern, Pyramiden, Prismen und Hexaedern eingegangen. Da die in Abschnitt 2.4.1, 2.4.2 und 2.4.3 vorgestellten Qualitätskriterien nur abhängig von geometrischen Parametern sind, können in den Arbeiten von Shewchuk [2002a,b] Kriterien nachgelesen werden, in die auch Informationen über die Eigenschaften der physikalischen Simulation einfließen.

### 2.4.1 Dreiecksqualitätskriterien

Die Qualitätskriterien in diesem Abschnitt beziehen sich auf ein Dreieck, wie es in Abbildung 2.14a dargestellt ist. $\vec{e}_0$, $\vec{e}_1$ und $\vec{e}_2$ sind die Vektoren zur jeweiligen Kante. $e_{min}$ steht für die kürzeste Kante, $e_{max}$ für die längste und $e_{med}$ für die durchschnittliche Kantenlänge. Die Fläche $A$ berechnet sich aus der Hälfte des Kreuzproduktes von einem beliebigen Paar von adjazenten Kanten (z.B. $A = 1/2 \, \|\vec{e}_0 \times \vec{e}_1\|$). Bei einem Einheitsdreieck gilt $A = \sqrt{3}/4$. Der Inkreisradius wird durch die Formel 2.2 und der Umkreisradius durch die Gleichung 2.3 bestimmt (Abbildung 2.14c und 2.14b). Der minimale umschließende Kreis wird mit $r_{mc}$ bezeichnet (Abbildung 2.14d). Er ist bei einem spitzwinkligen Dreieck gleich dem Umkreisradius, während er sich bei einem stumpfwinkligen durch die längste Kante $e_{max}$ ergibt, welche in diesem Fall zum Durchmesser von $r_{mc}$ wird.




(a) (b) (c) (d)

Abbildung 2.14: (a) Beschriftungskonvention für ein Dreieck in diesem Abschnitt; (b) Umkreisradius $R$; (c) Inkreisradius $r$; (d) minimaler umschließender Kreis $r_{mc}$

$$r = \frac{2A}{\|\vec{e}_0\| + \|\vec{e}_1\| + \|\vec{e}_2\|} \tag{2.2}$$

$$R = \frac{\|\vec{e}_0\| \, \|\vec{e}_1\| \, \|\vec{e}_2\|}{2r(\|\vec{e}_0\| + \|\vec{e}_1\| + \|\vec{e}_2\|)} \tag{2.3}$$

#### Aspektverhältnis

$$q_{ar} = \frac{e_{max}}{2\sqrt{3}r} \tag{2.4}$$

$$q_{ar} = \frac{e_{max}(\|\vec{e}_0\| + \|\vec{e}_1\| + \|\vec{e}_2\|)}{4\sqrt{3}A} \tag{2.5}$$

Das *Aspektverhältnis* (engl.: Aspect Ratio) kann durch Gleichung 2.4 oder alternativ durch Gleichung 2.5 bestimmt werden. Werte zwischen eins und 1,3 stehen für eine akzeptable Ausprägung des Qualitätskriteriums $q_{ar}$ [Stimpson u. a., 2007]. Oft wird unter Aspektverhältnis auch das Verhältnis zwischen Inkreisradius und Umkreisradius verstanden, welches jedoch unter dem Namen *Radienverhältnis* eingeführt werden soll.

#### Radien- und Kantenverhältnis

$$q_{rr} = \frac{R}{2r} \tag{2.6}$$

Das *Radienverhältnis* (engl.: Radius Ratio) wird mittels Gleichung 2.6 bestimmt. Dreiecke mit gutem Radienverhältnis besitzen einen Qualitätswert $q_{rr}$ zwischen eins und drei [Stimpson u. a., 2007].

$$q_{e\_rat} = \frac{e_{max}}{e_{min}} \tag{2.7}$$

Das *Kantenverhältnis* kann durch Gleichung 2.7 errechnet werden. Dreiecke mit akzeptabler Qualität besitzen Werte zwischen eins und 1,3.

#### Skalierte Jacobi-Determinante

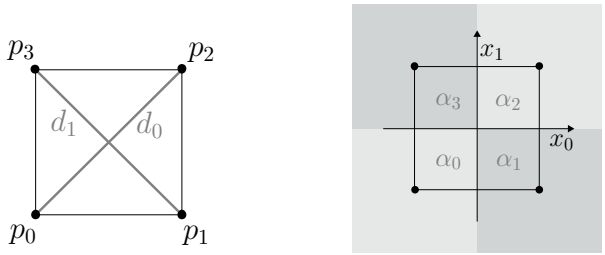
$$q_{s\_jacob} = \frac{2\sqrt{3}}{3}\frac{J}{E_{max}} \tag{2.8}$$

$$E_{max} = \max\left\{\|\vec{e}_0\|\,\|\vec{e}_1\|, \|\vec{e}_0\|\,\|\vec{e}_2\|, \|\vec{e}_1\|\,\|\vec{e}_2\|\right\} \tag{2.9}$$

Sei $E_{max}$ das Produkt der Länge der zwei längsten Kanten (Gleichung 2.9) und $J$ die Jacobi-Determinante des Dreiecks, die im Fall von $\vec{n}\cdot(\vec{e}_2\times\vec{e}_1) < 0$ ($\vec{n}$ ist die Dreiecksnormale) den Wert $-J$ annimmt, dann ergibt sich die *skalierte Jacobi-Determinante* wie in Gleichung 2.8 beschrieben. Nimmt $q_{s\_jacob}$ Werte zwischen 0,5 und $2\sqrt{3}/3$ an, so handelt sich es um ein Dreieck mit guter Qualität [Stimpson u. a., 2007].

### 2.4.2 Viereckqualitätskriterien

Die in diesem Abschnitt erwähnten Qualitätskriterien beziehen sich auf ein Viereck, wie es in Abbildung 2.15a dargestellt ist. Dabei sind $\vec{e}_0$, $\vec{e}_1$, $\vec{e}_2$ und $\vec{e}_3$ die Vektoren der jeweiligen Kanten. $e_{min}$ steht für die kürzeste Kante, während $e_{max}$ ein Verweis auf die längste ist. Die Diagonalen des Vierecks sind $\vec{d_0}$ mit $\vec{d_0} = \vec{p}_2 - \vec{p}_0$ und $\vec{d_1}$ mit $\vec{d_1} = \vec{p}_3 - \vec{p}_1$, dabei steht $d_{max}$ für die längere Diagonale. Jedes Viereck besitzt zwei Achsen $\vec{x}_0$ mit $\vec{x}_0 = (\vec{p}_1 - \vec{p}_0) + (\vec{p}_2 - \vec{p}_3)$ und $\vec{x}_1$ mit $\vec{x}_1 = (\vec{p}_2 - \vec{p}_1) + (\vec{p}_3 - \vec{p}_0)$ (Abbildung 2.15b). Jeder Punkt $\vec{p}_n$ ($n \in \{1,2,3,4\}$) besitzt eine Normale $\vec{n}_n$, die sich aus dem Kreuzprodukt der anliegenden Kanten ergibt. Die Normale $\vec{n}$ des Vierecks ergibt sich aus dem Kreuzprodukt der zwei Achsen $\vec{x}_0$ und $\vec{x}_1$. Die normalisierten Varianten von Vektoren werden durch ein Dach über dem Vektornamen symbolsiert - z.B. im Fall der Normalen $\hat{n}$ bzw. $\hat{n}_n$. Alle Normaleneinheitsvektoren sind identisch, wenn alle Punkte des Vierecks in einer gemeinsamen Ebene liegen. Das Viereck kann in vier Regionen $\alpha_{0..3}$ partitioniert werden (Abbildung 2.15b), wobei $\alpha_n$ sich aus $\hat{n}\cdot\vec{n}_n$ ergibt.




(a) (b)

Abbildung 2.15: (a) Namenskonvention für ein Viereck in diesem Abschnitt; (b) die zwei Achsen $x_0$ und $x_1$ und die vier Partitionen $\alpha_{0..3}$ des Vierecks

Der vorzeichenbehaftete Flächeninhalt $A$ kann über $\frac{1}{4}\sum_{i=0}^{3}\alpha_i$ ermittelt werden. $A$ kann auch als Qualitätskriterium genutzt werden, mit welchem sehr kleine oder konkave Vierecke bzw. falsche Punktnummerierungen gefunden werden können.

#### Aspektverhältnis

$$q_{ar} = \frac{e_{max}\left(\|\vec{e}_0\| + \|\vec{e}_1\| + \|\vec{e}_2\| + \|\vec{e}_3\|\right)}{4A} \tag{2.10}$$

Das *Aspektverhältnis* kann durch Formel 2.10 bestimmt werden. Es ist normalerweise als Verhältnis zwischen der längsten Kante und dem Inkreisradius definiert. Für ein Viereck existiert nur im Fall von $\|\vec{e}_0\| + \|\vec{e}_2\| = \|\vec{e}_1\| + \|\vec{e}_3\|$ ein Inkreisradius, so dass das Aspektverhältnis alternativ auch wie in Gleichnung 2.5 für das Dreieck definiert werden kann. Vierecke, die einen Wert von $q_{ar}$ zwischen eins und 1,3 besitzen, sind von guter Qualität [Stimpson u. a., 2007].

### (Maximales) Kantenverhältnis

$$q_{er} = \frac{e_{max}}{e_{min}} \tag{2.11}$$

$$q_{max\_er} = \max\left\{\frac{\|\vec{x}_0\|}{\|\vec{x}_1\|}, \frac{\|\vec{x}_1\|}{\|\vec{x}_0\|}\right\} \tag{2.12}$$

Das *Kantenverhältnis* $q_{er}$ (Gleichung 2.11) und das *maximale Kantenverhältnis* $q_{max\_er}$ (Gleichung 2.12) bewegen sich bei guter Viereckqualität im Bereich von eins bis 1,3 [Stimpson u. a., 2007].

### (Skalierte) Jacobi-Determinante und Scherung

$$q_{jacob} = \min_{i\in\{0,1,2,3\}} \{\alpha_i\} \tag{2.13}$$

$$q_{s\_jacob} = \min\left\{\frac{\alpha_0}{\|\vec{e}_0\|\,\|\vec{e}_3\|}, \frac{\alpha_1}{\|\vec{e}_1\|\,\|\vec{e}_0\|}, \frac{\alpha_2}{\|\vec{e}_2\|\,\|\vec{e}_1\|}, \frac{\alpha_3}{\|\vec{e}_3\|\,\|\vec{e}_2\|}\right\} \tag{2.14}$$

Die *Jacobi-Determinante* kann an jedem Knoten durch Gleichung 2.13 bestimmt werden. Die *skalierte Jacobi-Determinante* ist die minimale Jacobi-Determinante dividiert durch die Länge der zwei anliegenden Kantenvektoren (Gleichung 2.14). Gute Qualität im skalierten Fall bedeutet Werte zwischen 0,3 und eins [Stimpson u. a., 2007].

Der Wert des Qualitätskriteriums *Scherung* $q_{shear}$ (engl.: Shear) ist identisch mit dem der skalierten Jacobi-Determinante.

### Schiefe und Windschiefe

$$q_{skew} = |\hat{x}_0 \cdot \hat{x}_1| \tag{2.15}$$

$$q_{wp} = 1 - \min\left\{(\hat{n}_0 \cdot \hat{n}_2)^3, (\hat{n}_1 \cdot \hat{n}_3)^3\right\} \tag{2.16}$$

Die *Schiefe* $q_{skew}$ (engl.: Skew) eines Vierecks, welche den Winkel zwischen den beiden Viereckseiten misst, wird durch Gleichung 2.15 ermittelt. Akzeptable Werte liegen im Bereich von 0,5 bis eins.

Die *Windschiefe* $q_{wp}$ (engl.: Warpage, Gleichung 2.16) errechnet sich über den minimalen Winkel von zwei Ebenen, die sich entlang der Viereckdiagonalen $\vec{d}_0$ oder $\vec{d}_1$ schneiden. Vierecke mit guter Qualität besitzen einen Wert zwischen null und 0,7. Für das Einheitsviereck mit rechten Winkeln ist $q_{wp}$ gleich null [Stimpson u. a., 2007].

### 2.4.3 Qualitätskriterien für weitere Elementtypen

Alle Qualitätskriterien für das Dreieck, die in Abschnitt 2.4.1 aufgeführt wurden, können auch auf andere Simplices, wie das Tetraeder, verallgemeinert werden. Auch die Viereckskriterien aus Abschnitt 2.4.2 lassen sich nach leichter Modifikation auf das Hexaeder anwenden. Eine ausführliche Auflistung von Qualitätskriterien für Tetraeder und Hexaeder bietet die Arbeit von Stimpson u. a. [2007].

Qualitätskriterien für Pyramiden und Prismen können in den Arbeiten von Dyedov u. a. [2009] und Stimpson u. a. [2007] nachgeschlagen werden.

Zur Auswertung der Qualität von Elementen, die keine Tetraeder sind, können diese auch in Tetraeder unterteilt werden. Durch Auswertung des Tetraeders mit der schlechtesten Qualität in Kombination mit der Qualität der Facetten des zu untersuchenden Elements kann ein Bild über die Gesamtqualität des Elements gewonnen werden [Frey und George, 2000].

# 3 Stand der Forschung

Dieses Kapitel beschreibt die wichtigsten Forschungsergebnisse aus dem Bereich der anisotropen Gittergenerierung. Dabei wird nicht nur auf die Erzeugung von hybriden Gittern eingegangen (Abschnitt 3.4), sondern auch auf relevante Ergebnisse aus angrenzenden Bereichen wie die Generierung von anisotropen Tetraedergittern (Abschnitt 3.1), dünn- bzw. dickwandigen Gittern (Abschnitt 3.2) und reinen Prismengittern (Abschnitt 3.3). Am Ende jedes Abschnitts wird eine Bewertung der vorgestellten Verfahren vorgenommen: Es wird erläutert, ob Ideen eines Verfahrens für den Gittergenerator in dieser Diplomarbeit verwendet werden können.

## 3.1 Anisotropie in Tetraedergittern

Die bekanntesten Verfahren zur anisotropen Tetraedergittergenerierung sind die Advancing-Layers- und die Advancing-Normals-Methode (Abschnitt 3.1.1). Die Generalized-Advancing-Layers-Methode, die im Rahmen einer Doktorarbeit entwickelt wurde [Garimella, 1999], stellt eine Verallgemeinerung der zuvor genannten Methoden dar. Die Inhalte dieser Doktorarbeit werden detailliert in Abschnitt 3.1.2 dargestellt, da sie einen guten Überblick über die aufkommenden Probleme bei der Randschichterzeugung für komplexe nichtmannigfaltige Geometrien geben und einige Lösungsvorschläge aufführen, die im Abschnitt 4 dieser Diplomarbeit aufgegriffen werden.

### 3.1.1 Advancing-Layers / Advancing-Normals

Von Pirzadeh [1993b] wird das Advancing-Layers-Verfahren vorgestellt, welches ein Tetraedergitter mit anisotropen Zellen im Randbereich unter Verwendung eines modifizierten Advancing-Front-Verfahrens generiert. Dabei werden im Randbereich die Tetraeder schichtweise auf der Front erstellt. Neue Punkte werden entlang der geglätteten Punktnormalen platziert (Abbildung 3.1). Der ideale Punkt wird durch ein physikalisches Modell basierend auf Federn ermittelt, da sich die Qualitätskriterien des Advancing-Front-Verfahrens für gestreckte Tetraeder nicht eignen. Die Nutzung des Federmodells präferiert den Punkt, der am Nähsten am Dreieck liegt und eine minimale Dehnung der Federn verursacht. Des Weiteren sorgt das Kriterium für die Überschneidungsfreiheit in der Randschicht. Ist die gewünschte Anzahl an gestreckten Tetraedern erreicht worden, wird die herkömmliche Advancing-Layers-Methode verwendet, um den restlichen Bereich mit isotropen Tetraedern auszufüllen.

Einen ähnlichen Ansatz wie Pirzadeh [1993b] verfolgen auch Hassan u. a. [1995]. Es werden Schichten mit Hilfe des Advancing-Front-Verfahrens und der üblichen Qualitätskriterien für isotrope Tetraeder erzeugt. Nach Fertigstellung der Schicht werden die Punkte der neuen Front unter Beibehaltung der Konnektivität der Elemente entlang der Tetraederkanten in Richtung des Randes verschoben, bis der vom Nutzer gewünschte Abstand erreicht ist (Abbildung 3.2). Sind auf der neuen Front weniger Punkte als auf der vorherigen, so muss entschieden werden, in welche Richtung der neue Punkt verschoben werden

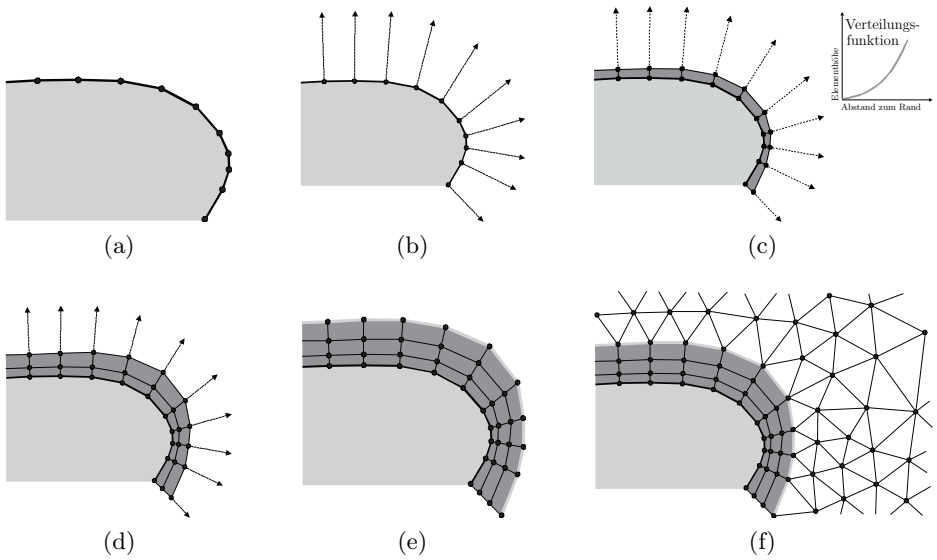



Abbildung 3.1: Ablauf des Advancing-Layers-Verfahrens [Pirzadeh, 1993b]: (a) die Oberfläche; (b) Ermittlung der Vertexnormalen; (c) - (e) schrittweise Generierung der Prismenschichten; (f) Erstellen des Tetraedergitters

soll. Die Autoren versuchen die Ähnlichkeit zwischen den einzelnen Dreiecksfronten zu wahren. Eine Auswahl einer bestimmten Richtung unter Berücksichtigung eines optimalen Elementvolumens ist nichttrivial. Im umgekehrten Fall - es sind mehr Punkte auf der neuen Front vorhanden - werden Punkte vereint, die durch das Zusammenschieben der Tetraeder zu nahe beeinanderliegen. Im 2D-Fall führt die Methode zu guten Ergebnissen, während im 3D-Fall bei komplexeren Geometrien Schwierigkeiten auftraten [Hassan u. a., 1996].

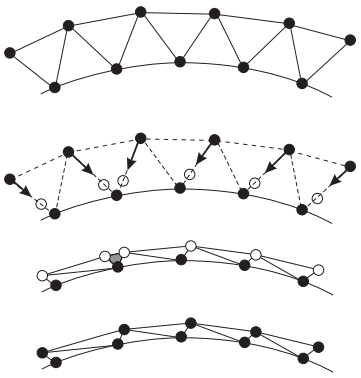


Abbildung 3.2: Ablauf des Advancing-Layers-Verfahrens im Randbereich nach dem Ansatz von Hassan u. a. [1995]

Durch die Weiterentwicklung zum Advancing-Normals-Verfahren begegnen Hassan u. a. [1996] den vorher genannten Problemen in 3D. Bei dem Verfahren werden Punkte der neuen Front entlang der Punktnormalen der vorherigen Front platziert. Danach erfolgen mehrere Glättungsiterationen, deren Anzahl pro Schicht zunimmt. Merkmalslinien können indirekt berücksichtigt werden, da Punktnormalen an Nähten von Patches nicht geglättet werden. Die Verfahren von Pirzadeh [1993b] und Hassan u. a. [1996] weisen starke Ähnlichkeiten auf und werden in der Literatur beide als die Advancing-Normals-Methode

bezeichnet.

Connell und Braaten [1995] stellen einige Verbesserungsvorschläge für das Advancing-Layers-Verfahren vor. So wird neben der expliziten Kontrolle auf Überschneidungsfreiheit der Prismen die Möglichkeit der Reduktion der Schichtdicke in engen Regionen der Geometrie implementiert.

Marcum und Weatherill [1995] beschreiben eine Methode, die eine Weiterentwicklung des Advancing-Normals-Verfahrens darstellt. Im Unterschied zur Methode von Hassan u. a. [1996] werden an scharfen Kanten zusätzliche Punktnormalen eingefügt. Dieses Vorgehen weist Parallelen zu dem allgemeineren Ansatz mit mehreren Richtungsvektoren pro Punkt, beschrieben von Garimella [1999], auf.

### 3.1.2 Generalized-Advancing-Layers

Da die Advancing-Layers- bzw. die Advancing-Normals-Methode bei komplexen Geometrien zu defekten Gittern und an scharfen Kanten zu Elementen mit sehr schlechter Qualität führt, stellt Garimella [1999] eine Verallgemeinerung der Methode vor, die für komplexe nichtmannigfaltige Geometrien Tetraedergitter mit anisotropen Randelementen generiert. Die Randschichten werden dabei ausgehend von einer Dreiecksoberfläche aus Prismen und Übergangspolyhedra aufgebaut, die anschließend in Tetraeder unterteilt werden. Durch verschiedene Prozeduren wird gewährleistet, dass ein valides Gitter mit überschneidungsfreien Elementen erstellt wird.

Um die Topologie einer nichtmannigfaltigen Fläche abbilden zu können, wird eine Datenstruktur konstruiert, die eine reduzierte Variante der Radial-Edge-Datenstruktur [Weiler, 1988] darstellt und den Namen Minimal-Use-Datenstruktur [Beall, 1998] trägt. Die Grundidee der Datenstruktur ist es, die nichtmannigfaltige Geometrie in mannigfaltige Abschnitte zu unterteilen. Nach Repräsentierung der Geometrie kann die Generierung der Randschicht erfolgen. Bei der Advancing-Normals-Methode wurde hierbei nur ein Richtungsvektor pro Punkt verwendet, wodurch der Einsatz des Verfahrens auf mannigfaltige Geometrien beschränkt wird. Daher werden in der Generalized-Advancing-Layers-Methode [Garimella, 1999] mehrere Richtungsvektoren, die beliebige Formen annehmen können und als Splines repräsentiert werden, pro Punkt verwendet. Mehrere Richtungsvektoren erlauben es, dass Prismen, die aus benachbarten Dreiecken extrudiert wurden, nicht an den Seitenflächen miteinander verbunden sein müssen. Dies führt zu gutgeformten Prismen auch bei großen Dihedralwinkeln. Auftretende Spalten in der Randschicht werden mit Übergangselementen gefüllt, um die anisotropen Dreieckselemente der in Tetraeder unterteilten Prismen nicht dem isotropen Gittergenerator als Ausgangsfront zu übergeben, welches verzerrte Tetraeder zur Folge hätte.

Liegt ein Richtungsvektor auf einer inneren Fläche oder auf einem anderen Rand, so wird die betreffende Fläche neu tesseliert, die Richtungsvektoren werden zu Kanten der Oberfläche und bilden zusammen mit benachbarten Richtungsvektoren die viereckigen Seiten der Prismen. Die einzelnen Schritte des Algorithmus sind in Abbildung 3.3 dargestellt.

Eine kritische Komponente in jedem Gittergenerator ist das Verfahren zur Platzierung der Gitterpunkte. Eine gute Verteilung der Punkte verbessert die Gitterqualität und verringert die Anzahl der nachträglich erforderlichen Optimierungsschritte. Die Punktplatzierung wird entlang der Richtungsvektoren vorgenommen. Das Abstandsverhältnis zwischen den einzelnen Punkten kann vom Nutzer vorgegeben werden, so dass z.B. Randschichten mit steigender Dicke generiert werden können. Liegt ein Richtungsvektor auf einem Rand,

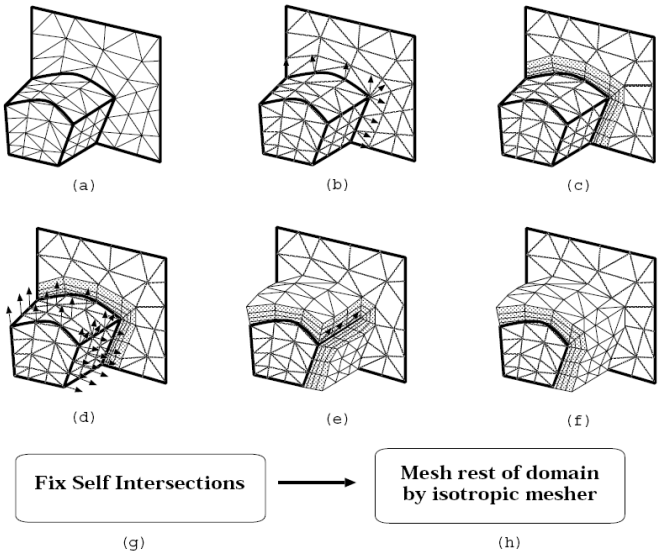



Abbildung 3.3: Schritte in der Gittergenerierung [Garimella, 1999]

so passt er sich diesem an und kann dabei beliebig gekrümmte Formen annehmen, liegt er hingegen im Inneren der Geometrie, so ist er gerade. Es wird also zwischen inneren Richtungsvektoren und Richtungsvektoren auf Rändern unterschieden. Unter speziellen Umständen können Richtungsvektoren zu beiden Kategorien gehören.

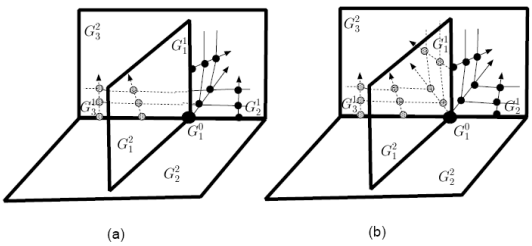



Abbildung 3.4: Bei einem inneren Rand werden mehrere Vektoren benötigt [Garimella, 1999].

Die Anzahl der Richtungsvektoren pro Randgitterpunkt hängt von der Oberflächengeometrie ab. Zumeist ist ein Richtungsvektor pro Punkt ausreichend, nur an nichtmannigfaltigen Stellen (z.B. innere Flächen (Abbildung 3.4)) und scharfen Kanten auf der Oberfläche sind mehrere Richtungsvektoren notwendig, um ein valides Gitter zu erhalten.

Mehrere Richtungsvektoren an einem Vertex werden auch benötigt, wenn eine komplexe Oberflächenbeschaffenheit bzw. eine grobe Diskretisierung des Ausgangsmodells vorliegt, dann kann zumeist kein gemeinsamer Richtungsvektor gefunden werden, da die einzelnen Dreiecksnormalen zu stark variieren. Der Grund für die Einführung von mehreren Richtungsvektoren ist, dass für ein überschneidungsfreies Prismengitter die Punkte auf dem Richtungsvektor von jedem Vertex aus sichtbar sein müssen. Mit der Sichtbarkeit ist gemeint, dass jedes Tetraeder, welches durch einen Punkt auf dem Richtungsvektor und den Punkten von einem Dreieck des Sterns geformt wird, positives Volumen haben muss (Abbildung 3.5). In Fällen, in denen aus stark variierenden Dreiecksnormalen noch ein einziger sichtbarer Richtungsvektor errechnet werden kann, ist dennoch die Verwendung von

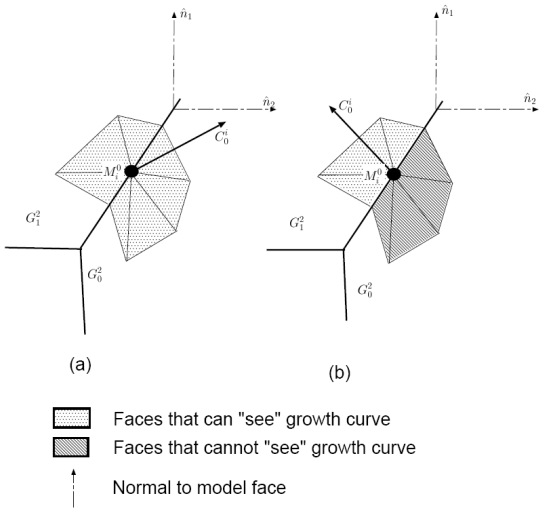



Abbildung 3.5: Verletzung des Sichtbarkeitskriteriums [Garimella, 1999]

mehreren Richtungsvektoren angebracht, da eine starke Abweichung zwischen den Dreiecksnormalen und dem Richtungsvektor zu großen dihedralen Winkeln in den Elementen der Randschicht führt [Garimella, 1999].

Die Anzahl der Richtungsvektoren pro Vertex ist dabei von der Oberflächenbeschaffenheit abhängig. Für jeden Stern werden je nach Anzahl der scharfen Kanten Dreiecksmengen gebildet, wobei jede Menge einen eigenen Richtungsvektor erhält. Damit hängt die Summe der Richtungsvektoren von der Zahl der vorliegenden Mannigfaltigkeiten und deren Oberflächenbeschaffenheit ab.

Die Überprüfung auf ein korrektes Randgitter findet nach der Berechnung der Richtungsvektoren statt, da Probleme wie Überschneidungen von Elementen durch die ungünstige Platzierung von Richtungsvektoren hervorgerufen werden. Es gibt zwei Gründe für invalide Elemente im Randbereich, zum einen ist dies die bereits genannte Verletzung der Sichtbarkeit eines Punktes auf dem Richtungsvektor und zum anderen verursachen Überschneidungen von Richtungsvektoren defekte bzw. invertierte Elemente (Abbildung 3.6). Das erstgenannte Problem wird bei der Berechnung der Richtungsvektoren adressiert, das zweitgenannte wird in einem eigenen Gittergenerierungsschritt behandelt und nachfolgend näher beschrieben.

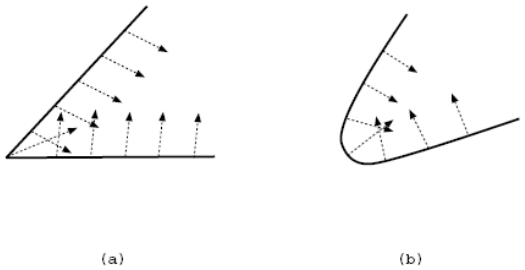



Abbildung 3.6: Überschneidung von benachbarten Richtungsvektoren an konkaven Stellen

Überschneidungen treten an scharfen Kanten sowie in Bereichen auf, in denen benachbarte Oberflächen, die jeweils mit einer Randschicht versehen werden sollen, nahe beieinander liegen. Der Überschneidung wird durch Glättung, Verringerung der Schichtendicke und

Entfernung von Richtungsvektoren in genau dieser Reihenfolge begegnet. Durch die gewichtete Laplace-Glättung können die meisten Fälle von Richtungsvektorüberschneidung, die aus der starken Krümmung der Oberfläche resultieren, eliminiert werden. Dies kann sich jedoch auf die Gitterqualität auswirken und zuvor gutgeformte Elemente verzerren. Die Verringerung der Randschichtdicke kommt dort zum Einsatz, wo die Schichtdicke relativ hoch im Verhältnis zur Krümmung der Oberfläche ist. Die Richtungsvektoren werden dabei lokal verkleinert, falls sich dadurch valide Elemente erreichen lassen. In einigen speziellen Fällen führen die zuvor erwähnten Methoden zu keinen korrekten Randelementen, hier kommt die Entfernung von Richtungsvektorsegmenten zum Einsatz, welche solange fortgeführt wird, bis nur noch Vektorsegmente existieren, die zu gültigen Elementen führen. Dies zieht die Verwendung von Übergangselementen nach sich, um Unterschiede in der Anzahl der Schichten zu überbrücken. Die Verkleinerung der Richtungsvektoren basiert auf der Erkenntnis, dass Probleme in den Randschichten zumeist in den oberen Schichten auftreten.

Ob es Überschneidungen zwischen benachbarten Randschichten gibt, wird nach der Generierung der Randschichtelemente mittels eines Octree-Verfahrens geprüft. Hierzu werden die Flächen, die später dem isotropen Tetraedergenerator übergeben werden, gegenseitig auf Überschneidung getestet und bei Bedarf wird die Länge der Richtungsvektoren verkleinert oder Elemente bzw. Richtungsvektoren werden entfernt. Da es durch Verkleinerung der Elemente auch zu Überschneidungen von Flächen kommen kann, die innerhalb der Randschicht liegen, muss das Octree-Verfahren iterativ arbeiten - solange bis keine Überschneidungen mehr vorhanden sind. Dabei ist dieses Verfahren relativ komplex: Die Operationen zur Auflösung der Überschneidungen, wie z.B. Verkleinern der Elemente, können adjazente Flächen in benachbarte Octree-Zellen schieben, so dass eine Aktualisierung des Octrees nötig wird. Auch bei dem Entfernen von Richtungsvektorsegmenten, welches das Einfügen von Übergangselementen nach sich zieht, muss die Information über die Elemente im Octree erneuert werden.

Durch Tests werden Richtungsvektoren ermittelt, die zu ungültigen Elementen oder schlechter Elementqualität führen. So müssen z.B. die in zwei Dreiecke unterteilten Rechtecke der Randprismenseiten positiven Flächeninhalt aufweisen, adjazente Dreiecke dürfen einen zuvor definierten Dihedralwinkel nicht überschreiten und benachbarte Richtungsvektoren werden auf Überschneidungen hin untersucht. Außerdem muss das Volumen der drei Tetraeder, in die jedes Prisma zerlegt werden kann, ebenfalls positiv sein [Garimella, 1999].

Durch die Verwendung von mehreren Richtungsvektoren pro Punkt müssen eine Reihe von Übergangselementen eingesetzt werden (Abbildung 3.7), welche die entstehenden Spalten füllen. Es wäre auch denkbar, die Spalten frei und sie später von dem isotropen Tetraedergenerator füllen zu lassen, jedoch würde dieser aufgrund der stark anisotropen Prismenseiten (die aus zwei gestreckten Dreiecken bestehen) nahe der Randschicht Elemente von schlechter Qualität erzeugen. Daher kommt die Verwendung von Übergangselementen der Gitterqualität zugute. Übergangselemente kommen an Kanten und Vertices zum Einsatz und müssen je nach Beschaffenheit der Spalte richtig gewählt werden, um ein gültiges Gitter zu gewährleisten. So setzt sich das letztendliche Übergangselement für eine Kante aus einzelnen Elementen zusammen, die z.B. dafür sorgen, dass der Höhenunterschied zwischen zwei benachbarten Schichten überbrückt wird. Ein Teil der Übergangselemente aus der Arbeit von Garimella [1999] ist als Vorschlag zu sehen und wurde nicht in den zur Doktorarbeit zugehörigen Gittergenerator implementiert, so dass sich aus den erwähnten numerischen Simulationen nicht ableiten lässt, was für Auswirkungen die Verwendung zahlreicher Übergangselemente auf die Simulationsergebnisse hat. Ito u. a. [2007] bemän-

geln, dass die Übergangselemente u.U. von schlechter Qualität sein können (Sliver) und nicht aus den üblichen Elementen wie Tetraedern, Pyramiden, Prismen und Hexaedern aufgebaut sind.

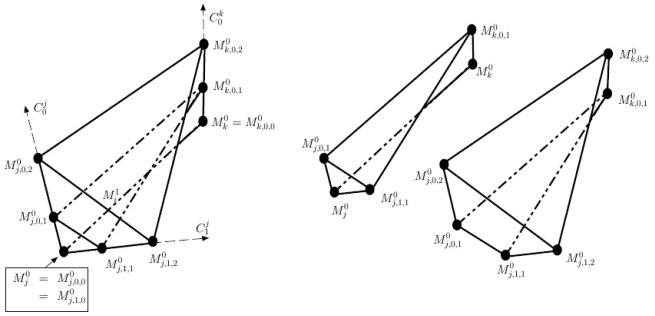


Abbildung 3.7: Das Bild zeigt einige der zahlreichen Übergangselemente, die an einer Kante platziert werden können. Es handelt sich um Polyeder, die letztendlich in Tetraeder unterteilt werden [Garimella, 1999]

Nach Aufbau der Randschicht werden deren Elemente in Tetraeder unterteilt. Teilt man dazu jedes Prisma in drei Tetraeder, müssen auch die viereckigen Seiten des Prismas durch eine Diagonale in zwei Dreiecke aufgespalten werden. Es existieren acht Möglichkeiten, wie ein Prisma mit Diagonalen versehen werden kann - sechs davon ergeben jedoch nur einen gültigen Ausgangspunkt für die Tetraedisierung des Prismas. Des Weiteren muss auf sich deckende Diagonalen bei benachbarten Prismenseiten geachtet werden, da sonst eine ungültige Konnektivität im Gitter entsteht. Gültige Diagonalen können in der Randschicht gewährleistet werden, indem man die Vertices durchnummeriert, danach den Knoten eines Dreiecks mit der kleinsten Nummer auswählt und von diesem die Diagonalen aufsteigend mit den über den beiden anderen liegenden Vertices verbindet [Bauer, 1994]. Dies hat die konsistente Diagonalisierung zur Folge, weil bei den Vertices eines Dreiecks $(V_1, V_2, V_3)$ immer ein Vertex existiert, dessen Nummer kleiner als die der beiden anderen ist $(N_{V_1} < N_{V_2} < N_{V_3})$. Die Tetraedisierung kann dabei durch die Verwendung von Mustern erfolgen, wobei die sechs möglichen Unterteilungsmuster des Prismas durch Drehung auf zwei Muster (Abbildung 3.8) reduziert werden können [Garimella, 1999].

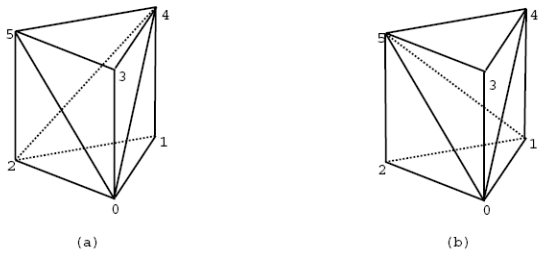



Abbildung 3.8: Zwei Muster zur Zerlegung eines Prismas in drei Tetraeder [Garimella, 1999]

### 3.1.3 Andere Verfahren

Die Hauptprobleme von semistrukturierten Gittergeneratoren treten an scharfen Kanten auf. Hier können leicht gestreckte bzw. defekte Elemente entstehen. Ziel des von Sharov

u. a. [2003] beschriebenen Algorithmus ist es, eine automatische Vergitterung des Gebietes basierend auf Methoden zur Erzeugung von Prismengittern zu erreichen und dabei gute Gitterqualität an scharfen Kanten zu gewährleisten. Die Methode verändert dazu die Oberflächentriangulierung, so dass in der Nähe von Ecken nicht verzerrte, isotrope Tetraeder verwendet werden können. Ohne diese Verfeinerung tendieren die Gittergeneratoren dazu, zu große Tetraeder an Ecken zu produzieren. Dabei wird die Oberfläche an Ecken verfeinert, so dass in Eckennähe eine höhere Auflösung entsteht. Dies hat zur Folge, dass mit der Verwendung von nur einem Richtungsvektor auch komplexere Oberflächen mit einem anisotropen volumetrischen Gitter in Eckennähe versehen werden können. In der Umgebung um Kanten werden die Oberflächendreiecke gestreckt. Die beiden Vorgehensweisen basieren auf der Idee, dass man durch eine durchdachte Oberflächentriangulierung bei der volumetrischen Gittergenerierung auftretende Probleme weitestgehend vermeiden kann. Auf der Dreiecksoberfläche werden schließlich Prismen extrudiert, die später in Tetraeder unterteilt werden, dabei werden Ecken und scharfe Kanten ausgelassen, da hier der isotrope Gittergenerator zum Einsatz kommt [Sharov u. a., 2003]. Ein Nachteil nach Ito u. a. [2007] ist der Mehraufwand des Nutzers, der in den Oberflächenvergitterungsprozess Einfluss nehmen muss, um gute Ergebnisse zu gewährleisten, und die hohe Anzahl von zusätzlichen Vertices in Randnähe.

Detomi [2004] verwendet ein Federmodell, um Randschichten in sein Tetraedergitter einzufügen und indirekt die maximale Extrusionsdicke zu ermitteln (Abbildung 3.9). Als Ergebnis gewinnt er ein reines Tetraedergitter mit anisotropen Tetraedern im Randbereich. Als Eingabe erhält sein Algorithmus ein Tetraedergitter, in welches in Tetraeder unterteilte Prismen mit sehr geringen Volumen eingefügt werden. Alle Tetraederkanten werden nun mit Federn versehen. Um degenerierte Tetraeder zu vermeiden, werden zusätzlich Federn im Tetraeder eingefügt. Diese verbinden jeden Knoten des Tetraeders mit dem gegenüberliegenden Dreieck. Durch den Einsatz eines Deformationsalgorithmus wird versucht die Randschichttetraeder auf eine vom Nutzer bestimmte Höhe zu bringen. Die Randschichttetraeder entfalten sich, während die Federkräfte eine Überschneidung, Deformation und eine zu große Höhe der Tetraeder in konkaven Regionen verhindern.

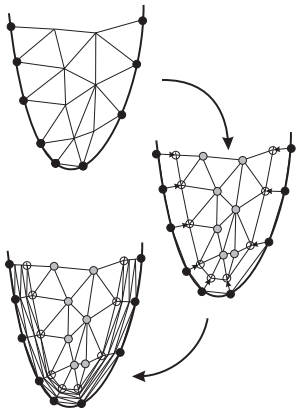


Abbildung 3.9: Durch die Deformation des Tetraedergitters wird Freiraum im Randbereich geschaffen, der mit Randschichtelementen gefüllt werden kann.

### 3.1.4 Bewertung

Die Verfahren von Pirzadeh [1993b] und Hassan u. a. [1995] stellen die Anfänge der Entwicklung von Gittergeneratoren für anisotrope Tetraedergitter dar. Garimella [1999] greift

die Probleme der vorigen Verfahren, wie die Behandlung von komplexen, nichtmannigfaltigen Flächen, auf und schlägt Methoden vor, um diese Probleme zu behandeln. So werden eine Reihe von Übergangselementen, eine Datenstruktur um nichtmannigfaltige Topologie abzubilden und mehrere Vorgehensweisen zur Behandlung von Defekten im Gitter genannt. Die Methoden der drei Autorengruppen eignen sich auch für einen Gittergenerator, der hybride Gitter erstellen soll. Schließlich wird zuerst durch Extrusion eine Prismenrandschicht erzeugt, die später in Tetraeder unterteilt wird. Verzichtet man auf die Unterteilung und wendet die Methoden zur Vermeidung von Defekten, die von Garimella [1999] entwickelt wurden, auf Prismenrandschichten an, so erhält man eine gute Grundlage für einen Gittergenerator für hybride Gitter.

Die Idee von Sharov u. a. [2003] an scharfen Kanten, an denen Schwierigkeiten beim Erstellen von anisotropen Randelementen auftreten, auf gestreckte Elemente zu verzichten, ist eine Möglichkeit Defekte im Gitter zu umgehen. Jedoch hat dies den Nachteil, dass das Gitter in diesen Bereichen nicht an das Strömungsverhalten angepasst ist, welches zu Ungenauigkeiten bei der Strömungssimulation führen könnte. Sharov u. a. [2003] modifiziert die Triangulierung des Eingabegitters, um eine optimale Basis für das spätere volumetrische Gitter zu ermöglichen. Die zuvor genannten Verfahren nutzen eine Dreiecksoberfläche, um danach Schichten aus anisotropen Tetraedern zu formen und schließlich den noch vorhandenen Freiraum mit isotropen Tetraedern zu füllen. Detomi [2004] geht von einem Tetraedergitter aus, in welches nachträglich gestreckte Tetraeder in Randnähe eingefügt werden. Ein Tetraedergitter als Eingabe zu nehmen und nachträglich Randschichten einzufügen, hat den Vorteil kein neues Gitter erzeugen zu müssen. Nachteilig ist, dass das Tetraedergitter berücksichtigt werden muss und man damit - gerade wenn man Übergangselemente an scharfen Kanten nutzen möchte - bei der Randschichtgenerierung weniger Freiheit besitzt bzw. das Tetraedergitter an die Randschicht angepasst werden muss (Abbildung 3.10).

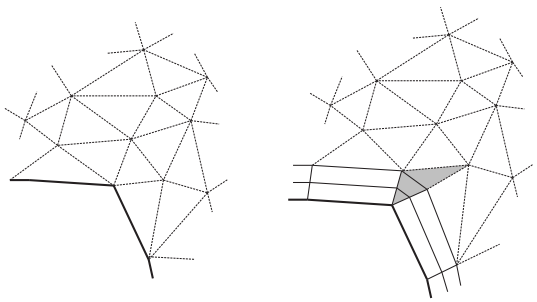


Abbildung 3.10: Werden beim Deformationsansatz Übergangselemente in der Randschicht verwendet, so müssen zusätzliche Tetraeder im Tetraedergitter eingefügt werden, um die Konnektivität zwischen Randschicht und Tetraederkern zu erhalten.

## 3.2 Thin-Shell / Thick-Shell

Die Analyse von dünnwandigen Strukturen (Thin-Walled-Structures) wie z.B. von im Spritzgussverfahren gefertigten Plastikteilen findet sich in vielen Bereichen der Industrie. Dabei wird eine Struktur als dünnwandig angesehen, wenn sich die ersten beiden Dimensionen, Länge und Breite, im Verhältnis zur dritten Dimension, Dicke, stark unterscheiden. Von Lee und Xu [2005] wird ein Algorithmus beschrieben, der durch Extrusion in

Richtung der Dreiecksnormalen dünne Prismenschichten bzw. Hexaederschichten erzeugt. Dabei heben die Autoren hervor, dass auch Nichtmannigfaltigkeiten, die an Kanten auftreten können, von ihrem Verfahren berücksichtigt werden. Für Nichtmannigfaltigkeiten an Vertices und komplexe Geometrien ist der beschriebene Algorithmus nicht geeignet. Die Extrusionsrichtung ist dabei nicht die Punktnormale, sondern die Flächennormale. Alle Flächen werden zuerst extrudiert, danach werden die Überschneidungen zwischen Nachbarelementen ermittelt und es wird eine korrekte Konnektivität zwischen den Elementen unter Zuhilfenahme der Schnittgeraden erzeugt, so dass keine Überschneidungen mehr auftreten (Abbildung 3.11). Auch das mögliche Auftreten von invaliden Elementen durch starke Unterschiede in der Orientierung der Seitenkanten eines Elements wird behandelt (Abbildung 3.12). Zur Korrektur werden die Winkel der Seitenkanten eines Elements und seiner Nachbarelemente zur Oberfläche sukzessiv verändert [Lee und Xu, 2005].

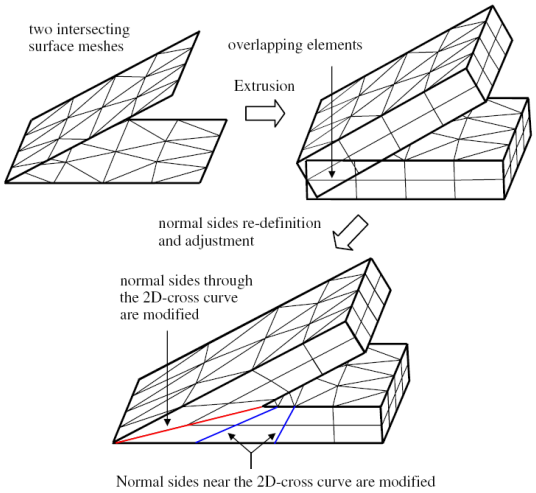



Abbildung 3.11: Herstellung der Konnektivität zwischen Prismen mittels Schnittberechnung [Lee und Xu, 2005]

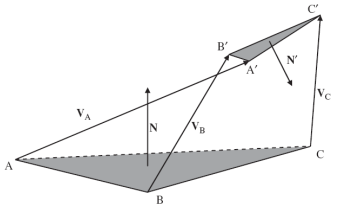



Abbildung 3.12: Entstehen eines invaliden Prismas durch sehr unterschiedliche Richtungsvektoren [Lee und Xu, 2005]

Erleben u. a. [2005] beschreiben eine Methode, um eine Oberfläche zwecks physikalischer Simulationen mit einer Schicht aus Prismen zu überziehen, die schließlich in Tetraeder unterteilt werden. Dabei werden invalide Prismen vermieden, indem darauf geachtet wird, dass die Winkel zwischen den Punktnormalen des Ausgangsdreiecks und der Normalen des extrudierten Dreiecks jeweils kleiner als 90° sind. Die maximale Extrusionsweite wird für alle Punktnormalen des Ausgangsdreiecks berechnet. Durch dieses Vorgehen werden Überschneidungen des Prismas mit seinen Nachbarn sowie ein negatives Volumen durch die falsche Orientierung des extrudierten Dreiecks vermieden. Möchte man gleichhohe Prismen für alle Dreiecke generieren, kann durch Iteration über die Prismen die minimale Extrusionslänge und somit die maximale globale Extrusionslänge ermittelt werden.

Dabei ist bei starken Schwankungen in der maximalen Extrusionslänge eine adaptive Extrusion, bei der für jedes Dreieck die lokal errechnete Extrusionslänge verwendet wird, der globalen maximalen Länge vorzuziehen. Verkürzt man bei der adaptiven Höhenanpassung ein Prisma entlang der Extrusionsvektoren, so bleibt bei zuvor konvexen Prismen die Eigenschaft der Konvexität erhalten. Um die Überlappung von sich gegenüberliegenden Prismenschichten zu verhindern, wird mit Hilfe einer räumlichen Hashing-Datenstruktur auf Überschneidung zwischen den Extrusionsvektoren und den durch die Prismen aufgespannten Bounding-Boxes geprüft und bei positivem Ergebnis die Extrusionslänge verkürzt. Das Problem die Prismen in Tetraeder zu unterteilen, so dass die Seitendreiecke von benachbarten Prismen zusammenpassen, wird hier wie folgt gelöst: Ein Startdreieck auf der Oberfläche wird ausgewählt und mit einer beliebigen Prismentetraedisierung versehen, danach werden die benachbarten Oberflächendreiecke mit Tetraedisierungen versehen, die zu den Seitendreiecken des Startprismas passen. Es wird also immer darauf geachtet, dass die Seitendreiecke der schon tetraedisierten Nachbarn mit denen des gerade bearbeiteten Prismas übereinstimmen. Dies wird so lange fortgeführt, bis die ganze Oberfläche mit in Tetraeder unterteilten Prismen versehen ist. Während des Unterteilungsprozesses kann es natürlich auch dazu kommen, dass benachbarte Tetraedisierungen nicht zusammenpassen, dies kann jedoch durch lokale Kippoperationen der Seitendiagonalen in den meisten Fällen korrigiert werden. Ist eine Korrektur nicht möglich, so wird das Problem wellenartig über die benachbarten Prismen propagiert, bis ein Prisma gefunden wurde, dessen Tetraedisierung verändert werden kann, ohne eine ungültige Überschneidung der Seitendiagonalen hervorzurufen [Erleben u. a., 2005].

Erleben und Sporring [2007] publizierten ein Verfahren, welches auf ihrer vorangegangenen Publikation aufbaut [Erleben u. a., 2005], um dickwandige Gitter (engl.: Thick-Shell-Meshes) für Deformationssimulationen zu produzieren. Dabei soll für ein Dreiecksgitter ein dickwandiges Tetraedergitter mit einer minimalen Anzahl von Elementen erstellt werden. Da ein Überlappen der gegenüberliegenden Fronten vermieden werden muss, wird die im Distanzfeld implizit vorhandene mediale Oberfläche als Stoppkriterium bei der Punktnormalenextrusion herangezogen. Man kann die mediale Oberfläche auch explizit errechnen, für komplexe Oberflächen eignen sich diese Verfahren jedoch nicht, sodass Erleben und Sporring [2007] ein implizites Verfahren wählten. Der Gradient des Distanzfeldes ist nahe der Oberfläche parallel zur Punktnormalen der Dreiecksoberfläche. Mit zunehmendem Abstand zur Oberfläche laufen die Gradienten auf singuläre Punkte zu, diese ergeben sich an Stellen, die zu zwei oder mehreren Punkten auf der Oberfläche den gleichen Abstand besitzen. Die singulären Punkte liegen auf der medialen Oberfläche bzw. auf der Skelettrepräsentation der Oberfläche.

Um die mediale Oberfläche zu finden, ermitteln die Autoren in ihrem ersten Ansatz mit einem Bisektionsverfahren auf der Punktnormalen $n$ vom Vertex $p$ die Position, für die die Gleichung $\epsilon = -\phi(p - \epsilon n)$ nicht mehr gilt. Die Distanzkriteriumsgleichung ändert sich von $\epsilon = -\phi$ in $\epsilon \neq -\phi$, sobald die mediale Oberfläche überschritten wird. Dieses Vorgehen führte in Tests besonders an Symmetrielinien zu schlechten Ergebnissen: Die Normalenextrusion wurde an einigen Stellen zu früh abgebrochen. Deshalb wurde ein zweiter Ansatz vorgeschlagen, in dem abhängig von der Voxelauflösung schrittweise entlang des Gradienten des Distanzfeldes ausgehend von einem Vertex gelaufen wird. Erleben und Sporring [2007] nennen dies Line-Stepping. Als Stoppkriterium wird ein Normalentest $\nabla\phi(p) \cdot \nabla\phi(q) > \rho$ durchgeführt, wobei $p$ die Vertexposition und $q$ die aktuelle Position im Distanzfeld ist - des Weiteren gilt für den nutzergewählten Wert $\rho > 0$. Die Idee ist, dass sich die Richtung des Gradienten ändert, sobald die mediale Oberfläche überschritten wird. Jedoch besitzt der Normalentest Probleme. Beginnt das Verfahren auf einer medialen Oberfläche, so kann der Extrusionsvektor sehr lang werden, so dass bei Verbinden der Extrusionspunkte, um

ein Prisma zu formen, invalide Elemente entstehen können (Abbildung 3.13). Des Weiteren kann die mediale Oberfläche überquert werden, wenn die Oberflächenkrümmung geringer ist als der Kosinuswert $\rho$ (Abbildung 3.13). Auch muss man auf eine ausreichende Auflösung der Dreiecksoberfläche achten, da sonst Überschneidungen von gegenüberliegenden Fronten entstehen können (Abbildung 3.14).

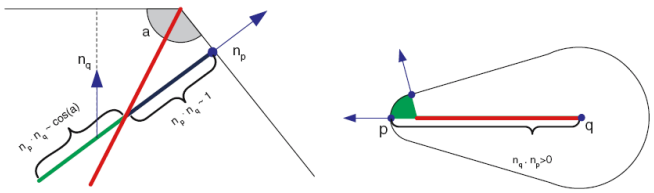



Abbildung 3.13: links: Überlaufen der medialen Oberfläche bei größerem Wert von $\rho$ als die Oberflächenkrümmung; rechts: zu langer Extrusionsvektor bedingt durch Ablaufen des Gradienten auf der medialen Achse [Erleben und Sporring, 2007]

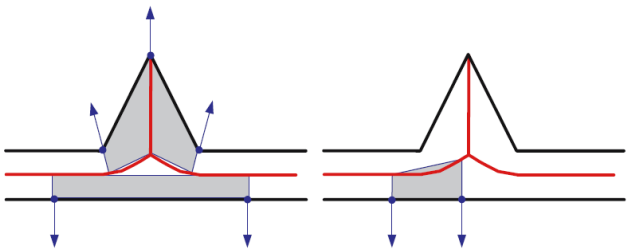


Abbildung 3.14: Sampling-Problem beim Line-Stepping-Verfahren [Erleben und Sporring, 2007]

### 3.2.1 Bewertung

Erleben u. a. [2005] sowie Lee und Xu [2005] erläutern das Problem, dass stark variierende Richtungsvektoren bei der Extrusion zu defekten Prismen führen und schlagen Methoden vor, um dies zu verhindern. In beiden Ansätzen wird auf ein *Umklappen* des oberen Dreiecks des Prismas geachtet und ggf. eine Veringerung der Extrusionshöhe durchgeführt. Des Weiteren beschreibt Erleben u. a. [2005] eine Möglichkeit Prismen in Tetraeder zu unterteilen. Jedoch terminiert der Unterteilungsalgorithmus nicht zwangsläufig, so dass - falls eine Unterteilung der Prismenrandschichten erfolgen soll - lieber auf das Verfahren von Garimella [1999] zurückgegriffen werden sollte.

Die mediale Oberfläche zu nutzen, um die maximale Dicke der Tetraederschicht zu ermitteln, wird von Erleben u. a. [2005] vorgeschlagen. Dazu nutzen sie ein Distanzfeld, in welchem die mediale Oberfläche implizit enthalten ist. Durch Ablaufen des Distanzfeldes in Gradientenrichtung wird die mediale Oberfläche ermittelt. Jedoch wird in bestimmten Fällen die mediale Oberfläche überlaufen, welches zu defekten Randzellen führen kann. Das Distanzfeld bietet nahezu die Information, die benötigt wird, um die maximal mögliche Randschichtdicke zu erhalten. Nutzt man zuerst das Distanzfeld, um eine Annäherung an die maximale Extrusionshöhe zu gewinnen, und prüft danach mit einem Octree auf Überschneidungen der Randschichten, so können Defekte im Gitter und eine zu große Zahl an

Schnitttests vermieden werden. Ein zusätzlicher Vorteil ist, dass durch die Distanzfeldinformation ein bestimmter Abstand zwischen gegenüberliegenden Randelementen eingehalten werden kann.

## 3.3 Prismengitter

Kallinderis und Ward [1993] erzeugen ein Prismengitter, indem für jede Schicht die Viereckssseiten eines geglätteten Voxelgitters mit den Punktnormalen der vorherigen Schicht geschnitten werden. Ein gleichmäßiger Übergang innerhalb des Gitters wird durch Laplaceähnliche Glättungsschritte erreicht.

Der Programmablauf kann in zwei Phasen unterteilt werden: Im ersten Schritt werden die Richtungsvektoren errechnet und geglättet, während im zweiten die Vektoren mit der geglätteten, aus Vierecken bestehenden Oberfläche eines Voxelgitters geschnitten werden. Die Glättung des Voxelgitters verringert die Krümmung des Ausgangsgitters und minimiert dadurch die Wahrscheinlichkeit von Elementüberschneidungen. Durch den Schnitt der Vektoren mit der Voxeloberfläche gewinnt man ein Dreiecksgitter, welches die Ausgangsfläche für die Generierung der nächsten Prismenschicht darstellt [Kallinderis und Ward, 1993].

Besondere Aufmerksamkeit wurde auf den Algorithmus zur Ermittlung der Richtungsvektoren gelegt. Von den Richtungsvektoren hängt ab, ob die spätere Prismenschicht aus überschneidungsfreien Elementen mit positiven Volumen besteht oder Defekte aufweist. Für kontinuierliche, analytische Oberflächen ist die Berechnung der Oberflächennormalen trivial, arbeitet man jedoch mit diskretisierten Geometrien ist es schwierig einen Algorithmus zu finden, der eine gültige Punktnormale garantiert. Die einfache Berechnung der Punktnormalen durch Summierung der einzelnen Dreiecksnormalen des Sterns reicht hierbei nicht aus. Kallinderis und Ward [1993] schlagen die Einführung eines Sichtbarkeitskriteriums vor, welches bedeutet, dass ein Richtungsvektor bzw. die Punktnormale mit keiner der Normalen der inzidenten Dreiecke einen Winkel von mehr als 90° aufspannen darf. Der erlaubte Bereich, in dem sich der Vektor befinden darf, hat die Form eines kegelförmigen Polyhedrons und wird als Sichtbarkeitskegel bezeichnet. Um die Berechnungen im Verfahren zu vereinfachen, wird das Sichtbarkeitspolyhedron zu einem Kegel vereinfacht, der als maximaler Winkel, von dem der Vektor abweichen darf, zu jedem Richtungsvektor gespeichert werden kann.

Der Normalenvektor $\hat{N}_i$ von Stern $i$ kann leicht aus dem gewichteten Durchschnitt der Dreiecksnormalen $\hat{n}_k$ des Sterns berechnet werden (Gleichung 3.1), wobei $W_k$ ein Gewichtungsfaktor darstellt, der gleich zum Winkel zwischen der Kante $k$ und $k-1$ ist. Diese Methode produziert für glatte Oberflächen korrekte Punktnormalen, ist der Stern jedoch eine Diskretisierung einer komplexen Oberfläche wie z.B. dem Übergang zwischen Flügel und Flugzeugrumpf, so kommt es leicht zur Verletzung des Sichtbarkeitskriteriums. Aus diesem Anlass entwarfen Kallinderis und Ward [1993] einen Algorithmus, der in vielen Fällen auch für komplexe Geometrien valide Punktnormalen berechnet, indem er den kleinsten Winkel zwischen der Punktnormalen und allen Dreiecksflächen maximiert. Ein gewisser Grad an Orthogonalität der Punktnormalen kann durch das Festlegen eines maximalen Winkels $\beta_{max}$ für den Halbkegelwinkel $\beta_i$ erreicht werden [Kallinderis und Ward, 1993].

$$\hat{N}_i = \frac{\sum_k W_k \hat{n}_k}{\sum_k W_k} \tag{3.1}$$

Die so errechneten Punktnormalen können sich jedoch in konkaven Regionen, wo die Punkte nahe beieinander liegen, überschneiden und somit ein ungültiges Gitter verursachen. Um dies zu verhindern, werden die Punktnormalen in mehreren Iterationen geglättet. Nach jeder Iteration wird das Sichtbarkeitskriterium überprüft, hat ein Vektor den Sichtbarkeitskegel verlassen und somit das Kriterium verletzt, so wird der Vektor auf den Kegel projiziert und als *fest* für die folgenden Glättungsoperationen markiert. Nachteil der Methode ist die Annahme, dass die durchgeführten Operationen ein gültiges Gitter erzeugen - eine Prüfung auf Überschneidungsfreiheit oder negative Volumen der Elemente im Randbereich wird nicht vorgenommen [Garimella, 1999]. Die Methode geht von einer mannigfaltigen Dreiecksoberfläche aus und verursacht nahe scharfer Kanten Elemente mit geringer Qualität. Das Sichtbarkeitskriterium liefert einen ersten Ansatz für eine robustere Punktnormalenberechnung auf komplexen Oberflächen, garantiert jedoch nicht in allen Fällen eine korrekte Punktnormale, da für manche Sterne die Sichtbarkeit nur durch die Verwendung von mehreren Normalen pro Punkt erfüllt werden kann [Garimella, 1999].

### 3.3.1 Bewertung

Bei der Berechnung von Richtungsvektoren zur Dreiecksextrusion muss darauf geachtet werden, dass die Richtungsvektoren zu gültigen Prismen führen. Dies ist bei komplexen Oberflächen, bei welchen die Dihedralwinkel der Dreiecke eines Dreieckssterns stark variieren, ein Problem. Kallinderis und Ward [1993] betiteln das Problem als Verletzung des Sichtbarkeitskriteriums und stellen eine Methode vor, mit der durch eine Korrektur von Richtungsvektoren defekte Randelemente vermieden werden können. Das Verfahren stößt jedoch bei bestimmten Konstellationen an seine Grenzen, so dass eine Kombination des Sichtbarkeitskriteriums mit der Verwendung von mehreren Richtungsvektoren sinnvoll ist.

## 3.4 Hybride Gitter

Basierend auf ihrer früheren Arbeit [Kallinderis und Ward, 1993], in der ein reines Prismengitter erzeugt wurde, konzipierten Kallinderis u. a. [1997] nun einen Ansatz für einen Generator für hybride Gitter. Es werden die Dreiecke unter Berücksichtigung des Sichtbarkeitskriteriums extrudiert. Dabei ist die Länge jeder einzelnen Extrusionspunktnormalen von der Krümmung des Sterns abhängig. Die Krümmung ist der gemittelte Dihedralwinkel zwischen allen Dreieckspaaren des Sterns. Nach Berechnung der Extrusionsvektoren erfolgt die Glättung der Punktnormalen in mehreren Iterationen. In engen Bereichen der Geometrie wird die Anzahl der Prismenschichten sukzessiv verringert. Nach Generierung der Prismenschichten wird das Tetraedergitter durch ein Verfahren erzeugt, welches eine Kombination aus der Octree- und Advancing-Front-Methode darstellt [McMorris und Kallinderis, 1997].

Sharov und Nakahashi [1996] verfolgen einen ähnlichen Ansatz mit einigen Modifikationen, die eine bessere Elementqualität versprechen, verwenden jedoch nach Generierung der Randschicht einen Delaunay-Tetraedergenerator. Auch Lohner [1993] nutzt die Dreiecksextrusion zur Prismengenerierung. Im Unterschied zu den zuvor genannten Methoden werden Elemente in der Randschicht, die sich überschneiden oder von schlechter Qualität sind, aus dem Gitter entfernt. Dies wird durch einen Suchbaum beschleunigt. Bei

Sharov und Nakahashi [1996] und Lohner [1993] ist eine Unterteilung der Prismen in Tetraeder vorgesehen. Lohner [1993] nutzt eine iterative Methode, um die Viereckssseiten korrekt in Dreiecke zu unterteilen und damit eine gültige Konnektivität zwischen benachbarten, in Tetraeder unterteilte, Prismen zu erhalten. In dem folgenden Paper bemängelt Lohner [1999] die Schwierigkeiten der Dreiecksextrusion bei komplexen Geometrien und stellt ein Verfahren zur nachträglichen anisotropen Verfeinerung eines isotropen Delaunay-Tetraedergitters vor.

Der von Ito u. a. [2007] gewählte Ansatz produziert Prismenelemente im Randbereich, während das restliche Gebiet mit Tetraedern versehen wird. Die Weiterentwicklung zur herkömmlichen Extrusion der Oberflächendreiecke ist die Verwendung von zwei Richtungsvektoren an scharfen konvexen Kanten. Dieses Vorgehen erhöht die Elementqualität in diesen Bereichen, stößt jedoch an seine Grenzen, falls Gebiete mehr als zwei Richtungsvektoren benötigen. Dies ist den Autoren bewusst, jedoch führen sie an, dass der Einsatz von mehr als zwei Richtungsvektoren aufgrund von topologischen Bedingungen, die im Umfeld der Prismen eingehalten werden müssen, eine schwierige Aufgabe ist. Ein Alternating-Digital-Tree wird als Suchdatenstruktur genutzt, um invalide Randelemente zu vermeiden.

Oft verhindert nur ein kleiner Bereich der Oberfläche, dass eine korrekte Prismenschicht erstellt werden kann. Solche Bereiche treten z.B. um einen singulären Punkt auf und zwingen Verfahren mit nur einem Richtungsvektor zum Abbruch. Durch Einführung von zwei Richtungsvektoren stellen singuläre Punkte kein Problem mehr dar. Das Verwenden von mehreren Richtungsvektoren zieht normalerweise den Einsatz von generalisierten Prismen zum Abschluss der Übergangselemente an Kanten nach sich, dies ist insofern kritisch, da die Qualität dieser Elemente - bedingt durch die zwei gekrümmten Viereckssseiten - schwer zu evaluieren ist. Um auf andere Elemente als Tetraeder, Prismen und Hexaeder verzichten zu können, wird die erste Randschicht an Abschlusspunkten nicht extrudiert, so dass an dieser Stelle eine Randschicht weniger und kein generisches Prisma erstellt wird - die weiteren Schichten bestehen dann wieder aus Prismen.

Die Konstellation der Übergangselemente an einer scharfen Kante wird ermittelt, indem pro Punkt dieser Kanten die Zahl ausgehender Kanten ($n_{es}$) ermittelt wird. Bei $n_{es} = 1$ ist der Punkt das Ende einer scharfen Kante und es wird überprüft, ob ein Abschluss der scharfen Kante zu Elementen mit guter Qualität führt. Kann keine gute Qualität erreicht werden, wird die Kante durch die Modifikation der Umgebung verlängert und erst eine Kante weiter der Abschluss vorgenommen. Bei $n_{es} = 2$ werden pro Punkt der Kante zwei Richtungsvektoren erstellt, während sich bei $n_{es} > 2$ mehrere scharfe Kanten treffen und am Schnittpunkt miteinander vereint werden. Dabei ist zu ergänzen, dass die Übergangselemente zuerst als degenerierte Elemente mit keinem Volumen generiert und erst, nachdem die gesamte Front mit einer Randschicht versehen ist, entfaltet werden.

Innere Ränder werden als planare Ebenen aufgefasst und im Zuge der Advancing-Front-Prismengenerierung modifiziert, um mit den Seitenflächen der Randschichten zu korrespondieren. Hierbei werden die Dreiecke des inneren Randes weitläufig im Bereich um die zu erwartende Randschichthöhe entfernt und temporär, um die Fläche zu erhalten, mit einer groben Delaunay-Triangulierung ohne zusätzliches Einsetzen von Punkten versehen. Nach Generierung der Randschichten werden die verbleibenden gestreckten Dreiecke entfernt und der Bereich wird mittels eines Advancing-Front-Algoritmus trianguliert. Es wird versucht, die zuvor vom Nutzer angegebene Schichtenanzahl zu erreichen. Tritt der Fall ein, dass die Elementqualität der zu erzeugenden Elemente sehr gering ist, die Sichtbarkeit verletzt wird oder Überschneidungen mit einer gegenüberliegenden Randschicht

auftreten können, wird die Randschichtextrusion an dieser Stelle abgebrochen [Ito u. a., 2007].

Athanasiadis und Deconinck [2003] stellen einen Algorithmus zur Erstellung von hybriden Gittern vor, der auch mit bestimmten Nichtmannigfaltigkeiten (bedingt durch das genutzte CAD-Austauschformat STEP) umgehen kann. Anders als z.B. bei Garimella [1999], der von einer triangulierten Oberfläche ausgeht und daher innere Flächen, die an die Randfläche angrenzen, neu vergittern muss, arbeiten Athanasiadis und Deconinck [2003] auf einer Parametrisierung der Ausgangsfläche (Topologie- und Geometrierepräsentation) und vergittern die Oberflächen schon standardmäßig so, dass es später keine Interferenzen zwischen inneren Flächen und generierter Randschicht gibt.

Um die Validität der Prismenschicht zu garantieren, wird ein Prisma erst erstellt, wenn es alle Kriterien wie Qualität, Überschneidungsfreiheit, maximaler Unterschied von nur einer Schicht zu den benachbarten Prismen und ein vorgegebenes anisotropes Seitenverhältnis erfüllt. Da die Oberfläche schon auf die Randschicht zugeschnitten ist, können die Randelemente ohne Analyse der Umgebung lokal erstellt werden. Die semistrukturierte Prismenschicht wird schichtweise aufgebaut. Nachdem eine Schicht erstellt wurde, werden die Prismen einzeln überprüft und falls notwendig aus der Schicht entfernt, nach diesem Schritt wird die nächste Randschicht auf der vorigen erstellt und ebenfalls überprüft. Dieses Vorgehen wiederholt sich bei sukzessiv steigender Schichtdicke, bis die Prismen ein isotropes Seitenverhältnis besitzen und damit ein weicher Übergang in das isotrope Tetraedergitter gewährleistet ist.

Nach der Prismengenerierung wird der restliche Bereich mit einem Tetraedergenerator vergittert. Die Möglichkeit, alle Schichten in einem Schritt zu erzeugen, wurde vermieden, da hierdurch leicht bei komplexen Geometrien defekte Elemente erzeugt werden können, die schwer wieder zu reparieren sind. Durch die schrittweise Gittererzeugung werden Unzulänglichkeiten in einer Schicht umgehend beseitigt, so dass der Fehler nicht in die folgenden Schichten propagiert wird. Die geglätteten Richtungsvektoren bleiben für alle Schichten gleich. Das Sichtbarkeitskriterium eines Richtungsvektors wird nicht überprüft, stattdessen werden invalide Prismen entfernt, um dem Problem, dass für manche Konstellationen kein gültiger Richtungsvektor existiert, zu begegnen. Das genannte Abbruchkriterium kann zu einzelnen Löchern bzw. freistehenden Prismen in der obersten Randschicht führen, so dass nach Fertigstellung der letzten Schicht diese geglättet wird, um gleichmäßige Übergänge zwischen den einzelnen Schichten zu ermöglichen. Eine Octree-Datenstruktur wird zur Überprüfung auf Überschneidungen genutzt.

In der Methode werden nicht mehrere Richtungsvektoren pro Vertex eingesetzt, da dies nach Aussage von Athanasiadis und Deconinck [2003] nicht unbedingt bessere Ergebnisse als die traditionellen Methoden hervorbringt, weil starke Unterschiede in der Elementgröße zwischen Prismen und anliegenden Übergangselementen auftreten.

Khawaja und Kallinderis [2000] beschreiben ein Multi-Zonen-Gitterverfahren und die Idee der lokalen Reduzierung der Schichtenanzahl, welche dazu dient sehr schmale Elemente in engen Regionen zu vermeiden, da sie negative Auswirkungen auf die numerische Simulation haben können. Das Multi-Zonen-Gitterverfahren zielt darauf ab, unterschiedliche Randschichtauflösungen innerhalb der gleichen Geometrie zu erlauben.

Basierend auf einem CAD-Modell wird die Oberfläche so trianguliert, dass in der Umgebung von Kanten und besonderen Merkmalen die Auflösung erhöht wird, um eine gute Basis zur Generierung der Prismenschicht zu erreichen. Um das Prismengitter zu erstellen, werden zuerst die Richtungsvektoren ermittelt, dann wird die Länge der Extrusion

berechnet und schließlich wird durch Glättung ein weicher Übergang zwischen benachbarten Richtungsvektoren erreicht. Die Prismenschichtgenerierung erfolgt dabei schrittweise. Der Richtungsvektor ist hierbei ein zur diskretisierten Oberfläche orthogonaler Vektor, dessen Grad der Orthogonalität zuvor spezifiziert werden kann. Die Extrusionslänge wird so berechnet, dass die Krümmung der Oberfläche in jeder Prismenschicht verringert wird. Dabei können die Richtungsvektoren entlang beliebig geformter Spline-Oberflächen laufen.

Der Teraedergenerator basiert auf einem Octree-Verfahren, welches das Längenverhältnis der letzten Prismenschicht als Eingabe bekommt, damit ein gleichmäßiger Übergang von der Prismen- zur Tetraederschicht erfolgt. Eine Reduzierung der Schichtenanzahl kann auch an Nichtmannigfaltigkeiten eingesetzt werden oder bei einer Fläche, die keine Dicke besitzt und mit einem Prismengitter versehen werden soll. Hier kann an Kanten dieser Fläche, an welchen die Randschicht von einer Seite auf die andere Seite verlaufen soll, die Schichtenanzahl einfach auf Null verringert werden, während sie auf der anderen Seite von Null an erhöht wird. Durch die unterschiedliche Anzahl von Prismenschichten müssen Übergangselemente wie Tetraeder und Pyramiden genutzt werden, damit die Viereckksseiten der Prismen nicht dem Tetraedergenerator übergeben werden, der ein Dreiecksgitter erwartet [Khawaja und Kallinderis, 2000].

Wang und Murgie [2006] bestimmen die Extrusionsvektoren pro Vertex durch ein Distanzfeld, welches durch die Lösung der Eikonal-Gleichung mit einem Sweeping-Verfahren errechnet wird. Die Punktnormalen ergeben sich durch die Gleichung $n = \nabla\phi/|\nabla\phi|$. Nach Berechnung der Punktnormalen werden diese durch die Lösung von $x_t = Fn$ ($F$ entspricht der Geschwindigkeit, die hier $F = 1$ an jedem Vertex ist) mit einem Runge-Kutta-Verfahren vierter Ordnung ins Innere der Geometrie propagiert, bis die gewünschte Randschichtdicke erreicht wurde. Die propagierten Punkte werden so verbunden, dass sich Prismen ergeben. Zum Schluss findet die Tetraedisierung des verbleibenden Raumes statt.

Theoretisch kann das Dreiecksausgangsgitter beliebig weit ins Innere verschoben werden, ohne das sich die Konnektivität zwischen dem Ausgangsgitter und dem obersten Gitter der Prismenschicht verändert. Bei zunehmender Randschichtdicke muss man jedoch darauf achten, dass z.B. an konkaven Kanten keine Prismen mit sehr geringem Volumen entstehen (Abbildung 3.15). Um dies zu vermeiden, ist nach der Randschichtgenerierung eine Punktrepositionierung erforderlich. Die Autoren heben hervor, dass das Verfahren auch mit Singularitäten in der Geometrie zurechtkommt.

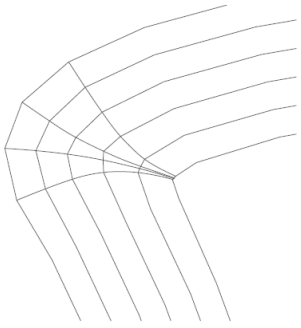


Abbildung 3.15: Distanzpropagation an einer konkaven Kante lässt Prismen mit geringem Volumen entstehen [Wang und Murgie, 2006]

### 3.4.1 Offset-Oberflächen und Bewegungsplanung

Der Prozess der Randschichtgenerierung lässt sich als Offset-Operation beschreiben. Die Oberfläche wird verschoben und die Dreieckspunkte der neugewonnenen Offset-Oberfläche und der Ausgangsoberfläche werden verbunden - man erhält eine Prismenschicht. Offset-Algorithmen berücksichtigen nicht den Abstand zur gegenüberliegenden Offset-Oberfläche - das Ergebnis kann dennoch überschneidungsfrei sein (Abbildung 3.16). Bei den überschneidungsfreien Methoden werden die Oberflächen beim Aufeinandertreffen vereint - ist die Offset-Distanz zu groß, so erhält man im Innern der Oberfläche keine Offset-Oberfläche. Für den äußeren Bereich der Oberfläche, bei welchem die Offset-Distanz unbeschränkt ist, lässt sich immer eine Offset-Oberfläche finden. Eine überschneidungsfreie Offset-Oberfläche lässt sich einfach durch eine Isosurface gewinnen, die trianguliert wird.

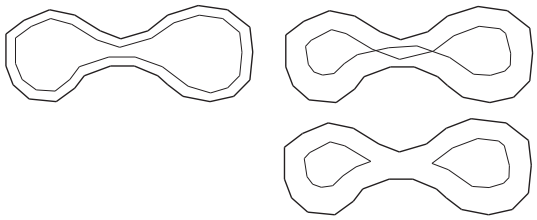


Abbildung 3.16: Wird eine zu große Offset-Distanz gewählt, so kommt es zu Überschneidungen bzw. zur Vereinigung der Offset-Oberflächen.

Auch bei der Bewegungsplanung von Robotern muss ein bestimmter Abstand zu Wänden eingehalten werden. Neben Offset-Algorithmen oder der Berechnung der Minkowski-Summe können auch trapezförmig unterteilte Räume (engl.: Trapezoidal Maps) zum Einsatz kommen [De Berg u. a., 2008]. Die Minkowski-Summe fällt in die Kategorie der Offset-Algorithmen, während die Wegbestimmung auf einer trapezförmig unterteilten Karte eine Annäherung an die mediale Achse darstellt.

### 3.4.2 Bewertung

Ito u. a. [2007] nutzen maximal zwei Richtungsvektoren an scharfen Kanten und vermeiden das degenerierte Prisma als Übergangselement. Die Prismenschicht wird erst mit einer Dicke von Null generiert und nach Fertigstellung entfaltet. Während alle bisher vorgeschlagenen Gittergeneratoren von einer triangulierten Fläche oder einem Tetraedergitter ausgegangen sind, arbeiten die Algorithmen von Athanasiadis und Deconinck [2003] auf einer Parametrisierung der Ausgangsfläche, um bestimmte Probleme zu umgehen, wie z.B. das Einfügen von Viereckssseiten in schon triangulierte innere Ränder. Daher ist das Verfahren vornehmlich für CAD-Geometrien geeignet. Wang und Murgie [2006] nutzen den Gradientenabstieg zur Generierung eines hybriden Gitters. Das Verfahren ist dabei fast identisch zu dem Line-Stepping-Verfahren von Erleben u. a. [2005] mit dem Unterschied, dass kein reines Tetraedergitter erstellt wird. Die Randschichtgenerierung lässt sich auch als Offset-Operation angewandt auf die Eingabeoberfläche auffassen, wobei auf Überschneidungen bzw. bei überschneidungsfreien Offset-Algorithmen auf Verschmelzung der Fronten geachtet werden muss. Nutzt man Offset-Methoden, so sollte eine variable Offset-Distanz möglich sein, damit die Dicke der Randschicht sich an den lokalen Freiraum anpassen kann, da sich die Strömung in Bereichen größeren Geometriedurchmessers

auch verändert und damit ein problemangepasstes volumetrisches Gitter ermöglicht wird (Abbildung 3.17).

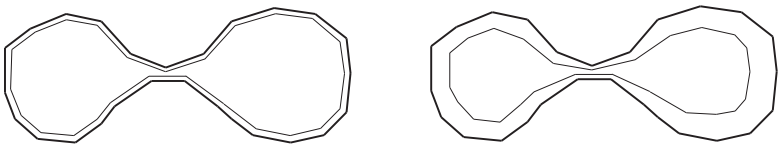


Abbildung 3.17: Wird keine variable Offset-Distanz gewählt, kann die Randschicht in großen Bereichen der Geometrie zu dünn sein.

## 3.5 Zusammenfassung

In der Literatur existieren mehrere Ansätze wie Prismen, teilweise nachträglich in Tetraeder unterteilt, in Randnähe erstellt werden können. So arbeiten einige Gittergeneratoren auf einer triangulierten Oberfläche, andere auf einem Tetraedergitter oder einer parametrisierten Eingabeoberfläche. Um randnahe Zellen zu formen, können die Dreiecke einfach extrudiert oder aber auch durch ein Federmodell nachträglich aufgespannt werden. Bei komplexen Oberflächen entstehen leicht defekte Prismen, so dass mehrere Methoden zur Überprüfung auf Defekte und zur Korrektur von invaliden Prismen recherchiert wurden. Übergangselemente und mehrere Richtungsvektoren ermöglichen es, sehr komplexe Oberflächen mit einer Randschicht zu versehen. Ist die Eingabegeometrie nichtmannigfaltig, so muss auch dies entsprechend berücksichtigt werden - z.B. durch eine Datenstruktur, die nichtmannigfaltige Topologie repräsentieren kann. Damit sich gegenüberliegende Randschichten nicht überschneiden, können Schnitttests durchgeführt oder aber auch die mediale Oberfläche ausgewertet werden.

Für den zu implementierenden Gittergenerator, der komplexe nichtmannigfaltige Geometrien mit Randschichten versehen soll, ist die Verwendung von mehreren Richtungsvektoren pro Vertex und den damit verbundenen Übergangselementen sinnvoll [Garimella, 1999; Ito u. a., 2007]. Um mehr Freiheit bei der Platzierung von Übergangselementen zu besitzen, wird als Eingabe ein Dreiecksgitter akzeptiert, welches zunächst mit einer Prismenschicht versehen wird. Danach wird der verbleibende Raum mit Tetraedern gefüllt. Um die Anzahl der Schnitttests gering zu halten, kann der Abstand zur medialen Oberfläche ausgewertet werden [Wang und Murgie, 2006; Erleben und Sporring, 2007]. Die Verwendung des Sichtbarkeitskriteriums und der Test auf überfaltete Prismen verhindern Defekte im hybriden Gitter [Kallinderis und Ward, 1993; Erleben u. a., 2005]. Die von Beall [1998] beschriebene Datenstruktur wird zur Speicherung der nichtmannigfaltigen Geometrien genutzt. Dies sind die Kernmethoden, welche für den Gittergenerator dieser Diplomarbeit genutzt werden sollen. Der genaue Gittergenerierungsprozess ist im nächsten Kapitel beschrieben.

# 4 Generierung hybrider Gitter

In diesem Kapitel werden - ausgehend von den Anforderungen an die Software (Abschnitt 4.1) - die Probleme erläutert, die bei der Generierung eines hybriden Gitters auftreten können (Abschnitt 4.2). Im Abschnitt 4.3 wird das Verfahren zur Gittergenerierung beschrieben, welches in Form eines Moduls für das Programmpaket Amira implementiert wurde (Abschnitt 4.4).

## 4.1 Anforderungen

Die Anforderungen an das Gittergenerierungsmodul, welches in das Visualisierungs-Framework Amira integriert werden soll, wurden in der Einleitung genannt (Abschnitt 1.2). Zwecks besserer Übersicht werden die Anforderungen nochmals aufgelistet:

#### GUI

- Geringe Anzahl an Parametern, die vom Nutzer angegeben werden sollen
- Auswahl der Geometriebereiche, auf welchen Prismenrandschichten erstellt werden sollen
- Angabe der Schichtdicke und des Wachstumsfaktors der Prismenschichten

#### Gittergenerator

- Hoher Automationsgrad des Gittergenerierungsprozesses
- Komplexe nichtmannigfaltige Geometrien sollen verarbeitet werden können
  - Der Einsatz von Übergangselementen an scharfen Kanten soll erprobt werden
- Generierung eines hybriden Gitters, welches im Inneren aus Tetraedern und im Randbereich primär aus Prismen besteht

#### Eingabe

- Triangulierungen mit Informationen über Materialzugehörigkeit (*Material ID*) und Randeigenschaft (*Boundary ID*) für jedes Dreieck

#### Ausgabe

- Export des hybriden Gitters im CGNS-Dateiformat

## 4.2 Probleme

Das Gesamtproblem einen Gittergenerator zu implementieren, der die zuvor genannten Anforderungen erfüllt, soll in Teilprobleme aufgelöst werden. Aufkommende Probleme bei der Randschichtgenerierung wurden in Abschnitt 3 - insbesondere in der Beschreibung des Generalized-Advancing-Layers-Verfahrens (Abschnitt 3.1.2) - aufgeführt und sollen hier wieder aufgegriffen werden.

In diesem Abschnitt wird - anhand eines sehr einfachen Algorithmus zur Generierung eines hybriden Gitters - aufgezeigt, welche Probleme im Gittergenerierungsprozess auftreten können (Algorithmus 1). Der Algorithmus erzeugt nur eine Prismenschicht für alle Dreiecke der Eingabe.

**Algorithmus 1** Grober Ablauf der Generierung eines hybriden Gitters

```
SIMPLE-PRISM-GEN(TRIANGULIERUNG Surf, REELLEZAHL h)
 1  for each v^0 ∈ Surf
 2       do ▷ Berechne die Vertexnormale und verschiebe v^0 um h Längeneinheiten
 3          v^0' ← CALC-NORMAL(v^0) · h + v^0
 4          ▷ Speichere die Zugehörigkeit zwischen v^0 und v^0'
 5          nextPoint(v^0) ← v^0'
 6  for each t^2 ∈ Surf
 7       do ▷ Erstelle ein Dreieck t^2' mittels der Funktion nextPoint aus den
 8          ▷ Vertices v^0_{0,1,2} des Dreiecks t^2
 9          t^2' ← (nextPoint(v^0_0), nextPoint(v^0_1), nextPoint(v^0_2))
10          ▷ Erzeuge die Dreiecksfront für den Tetraedergenerator
11          TetraFront ∪ {t^2'}
12          ▷ Speichere die sechs Prismenpunkte
13          PrismGrid ∪ {(v^0_0, ..., v^0_2, nextPoint(v^0_0), ..., nextPoint(v^0_2))}
14  ▷ Erstelle das Tetraedergitter
15  TetraGrid ← TETRA-GEN(TetraFront)
16  ▷ Vereinige die einzelnen Gitter zu einem hybriden Gitter
17  return HybridGrid ← MERGE-GRIDS(Surf, PrismGrid, TetraGrid)
```

Bei der Ausführung des Algorithmus können die folgenden Probleme auftreten:

### 4.2.1 Globale Überschneidungen

Der Gittergenerator führt im beschriebenen Programmablauf eine Offset-Operation aus, gefolgt von der Tetraedergenerierung und der Speicherung im CGNS-Format. Übergibt man dem Generator eine einfache Geometrie, z.B. eine Sphäre, so wird ein valides hybrides Gitter erstellt, solange die gewünschte Schichtdicke $h$ nicht zu groß ist. Bei zu groß gewählter Schichtdicke überschneiden sich die gegenüberliegenden Schichten (Abbildung 4.1). Unsere erste Problemklasse sind daher globale Überschneidungen.

### 4.2.2 Lokale Überschneidungen

Neben globalen können auch lokale Überschneidungen auftreten. Besitzt das Eingabegitter scharfe Kanten, wie z.B. bei einem Würfel, so kann es ab einer bestimmten Schichtdicke zu Überschneidungen von Richtungsvektoren kommen (Abbildung 4.2). Dies hat defekte Prismen zur Folge.

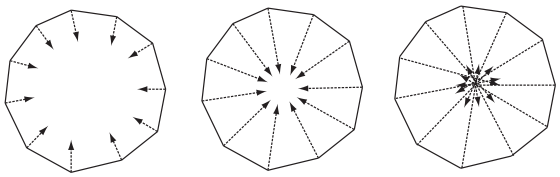


Abbildung 4.1: Die Bilder zeigen jeweils eine Sphäre im Querschnitt. Es werden Richtungsvektoren berechnet, die für die Extrusion der Randschicht notwendig sind. Ab einer bestimmten Vektorlänge kommt es zu Überschneidungen der Richtungsvektoren.

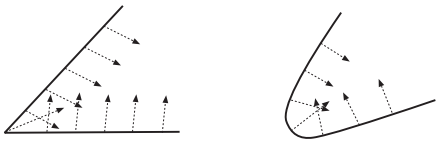


Abbildung 4.2: In der Nähe von scharfen Kanten kann es zu Überschneidungen der Richtungsvektoren kommen.

### 4.2.3 Komplexe Geometrien

Besitzt eine Dreiecksoberfläche stark variierende Dihedralwinkel können defekte Prismen entstehen. Dies ist auf eine Verletzung des *Sichtbarkeitskriteriums* zurückzuführen, welches im Abschnitt 3.3 erklärt wurde. Neben der Beschreibung des Begriffs wurde auch ein Verfahren vorgestellt, durch welches die Wahrscheinlichkeit erhöht wird einen Richtungsvektor zu erhalten, der nicht das *Sichtbarkeitskriterium* verletzt. Der Grundgedanke des Kriteriums kann für die Eingabeoberfläche $Surf$ wie folgt formuliert werden:

$$\forall v^0 \in Surf : \exists \vec{g}\, \forall t^2 \in Star(v^0) : \vec{g} \cdot Normal(t^2) > 0$$

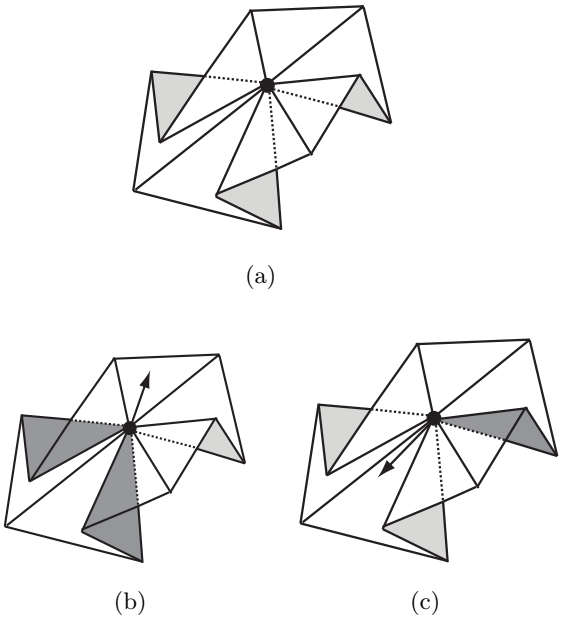
Der Winkel zwischen dem Richtungsvektor $\vec{g}$, welcher zum Vertex $v^0$ gehört, und aller zum Vertex $v^0$ inzidenten Dreiecke muss also kleiner als 90° sein, damit die später auf den Dreiecken des Sterns erstellten Prismen gültig sind und keine Überfaltungen auftreten. Für den Dreiecksstern in Abbildung 4.3 kann diese Bedingung nicht erfüllt werden. Es muss also ein Verfahren genutzt werden, welches valide Prismen auch für komplexe Geometrien garantiert.

### 4.2.4 Nichtmannigfaltigkeiten

Die Eingabedreiecksgitter können nichtmannigfaltig sein. Daher muss es für den Gittergenerator möglich sein nichtmannigfaltige Stellen zu erkennen, um Defekte im Gitter zu vermeiden. Nichtmannigfaltigkeiten können an Vertices und an Kanten auftreten (Abbildung 4.4).

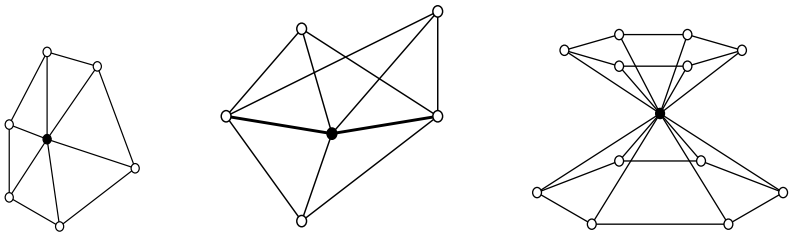
## 4.3 Gittergenerierungsprozess

In diesem Abschnitt wird der Ablauf der Gittergenerierung beginnend vom Einlesen der Oberfläche bis zur Ausgabe des fertigen hybriden Gitters erläutert. Dabei werden die Me-



(a) (b) (c)

Abbildung 4.3: (a) Das Bild zeigt einen Dreiecksstern mit drei scharfen Kanten. Durch die scharfen Kanten sind die Rückseiten (hellgrau) von drei Dreiecken zu sehen. Benutzt man nur einen Richtungsvektor an dem Vertex des Dreiecksstern, so würden bei der Extrusion defekte Prismen entstehen. Es existieren immer Dreiecke (dunkelgrau), für die der Winkel zwischen der Dreiecksnormalen und dem Richtungsvektor größer als 90° ist (Abbildung (b) und (c)).



(a) (b) (c)

Abbildung 4.4: (a) Mannigfaltiger Dreiecksstern; (b) Nichtmannigfaltigkeit an einer Kante; (c) Nichtmannigfaltigkeit an einem Vertex

thoden zur Lösung der in Abschnitt 4.2 aufgelisteten Probleme dargestellt. Eine Übersicht der Schritte, die im Gittergenerierungsprozess auftreten, ist in Abbildung 4.5 dargestellt. Jeder Schritt wird in einem eigenen Abschnitt erläutert.

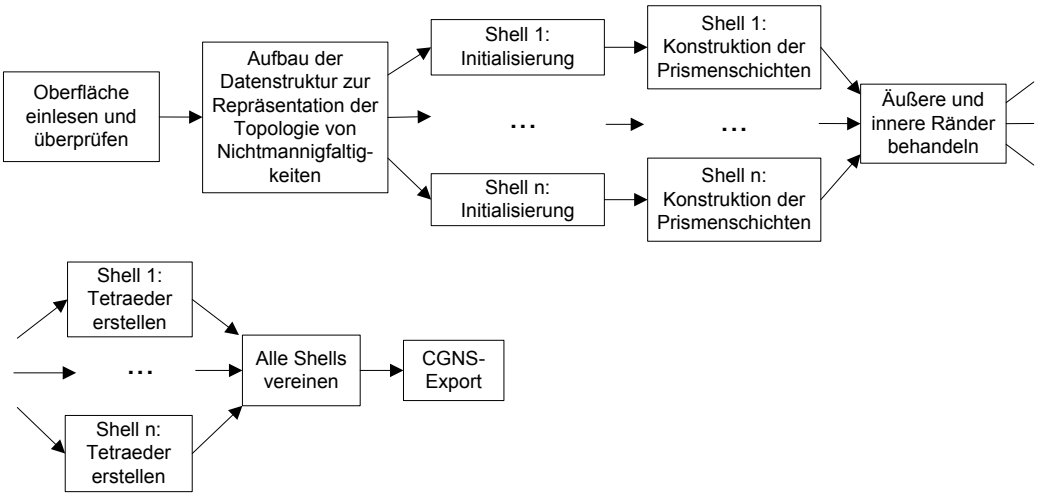



Abbildung 4.5: Es werden die einzelnen Schritte im Gittergenerierungsprozess gezeigt. Eine *Shell* bezeichnet dabei ein mannigfaltiges Gebiet der nichtmannigfaltigen Eingabegeometrie. *Shells* werden im folgenden Abschnitt näher beschrieben.

### 4.3.1 Eingabe der Oberfläche

Nach Übergabe eines Dreiecksgitters an den Gittergenerator wird geprüft, ob Löcher (Kanten mit genau einem inzidenten Dreieck) im Gitter vorhanden sind. Löcher dürfen im Gitter nicht vorhanden sein, da der Tetraedergittergenerator eine geschlossene Dreiecksfront benötigt. Des Weiteren müssen die Informationen aus der Oberfläche gewonnen werden, die dem Nutzer auf der GUI angezeigt werden - wie z.B. die Regionen, auf denen eine Randschicht erstellt werden kann.

Die Eingabeoberfläche besitzt neben der Konnektivitätsinformation und den Dreieckselementen auch Informationen darüber, welche *Boundary ID* einem Dreieck zugeordnet ist und welche *Material IDs* es besitzt. Ein Dreieck besitzt für seine Vorder- und Rückseite jeweils eine eigene *Material ID*. Diese wird später genutzt, um jedem Element des volumetrischen Gitters ein Material zuzuweisen.

### 4.3.2 Nichtmannigfaltigkeiten

Nach der Oberflächenanalyse wird die übergebende Dreiecksdatenstruktur mit Informationen über die Topologie von nichtmannigfaltigen Gebieten angereichert. Als Datenstruktur zur Repräsentation der nichtmannigfaltigen Topologien wird die *Minimal Use Datenstruktur* verwendet, die im Folgenden näher beschrieben wird.

Die *Minimal Use Datenstruktur* [Beall, 1998] basiert auf der *Radial Edge Datenstruktur* [Weiler, 1988], welche Parallelen zu den *Separation Surfaces* vom gleichen Autor aufweist (Abbildung 4.6). Grundidee ist die Einführung von *Uses*, dabei besitzt ein Dreieck genau zwei *Face Uses*, die jeweils auf einer Seite des Dreiecks liegen. Eine Kante erhält pro inzidentem Dreieck ein *Edge Use*, welches auf zwei *Vertex Uses* verweist. Die *Vertex Uses*

stehen in Bezug zu den Vertices, aus denen sie hervorgegangen sind. Die einzelnen *Uses* sind untereinander verbunden, so dass z.B. einfach über die benachbarten *Edge Uses* eines *Vertex Uses* iteriert werden kann. Der Unterschied zwischen der *Minimal Use Datenstruktur* und der *Radial Edge Datenstruktur* ist, dass letztere viel mehr Informationen bereitstellt, als für die meisten Anwendungsfälle nötig wäre [Garimella, 1999].

Die Verwendung von *Uses* hat zur Folge, dass die nichtmannigfaltige Geometrie durch die verbundenen *Face Uses* in Mannigfaltigkeiten unterteilt wird. Somit werden Algorithmen, die mannigfaltige Geometrien voraussetzen (wie z.B. eine Halbkantendatenstruktur), auch für Nichtmannigfaltigkeiten verwendbar.

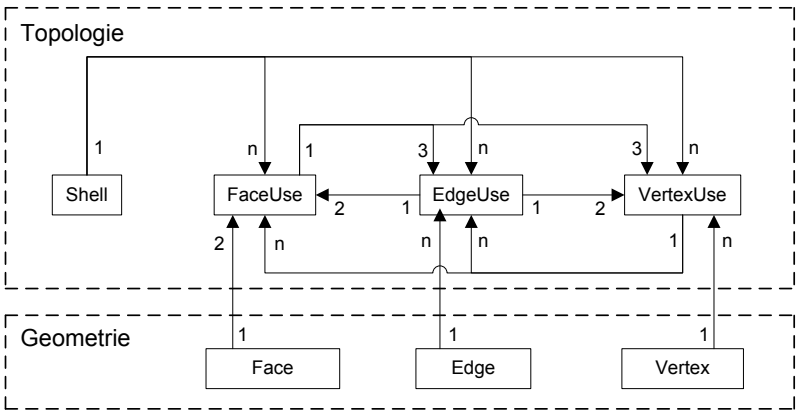



Abbildung 4.6: Bei der Konzeption der *Minimal Use Datenstruktur* wurde darauf geachtet, dass Geometrie- und Topologiedaten voneinander getrennt sind. Die Datenstruktur besitzt neben den erwähnten *Uses* auch *Shells*, die einzelne mannigfaltige Gebiete darstellen.

Der Ablauf zum Aufbau der Datenstruktur soll nun beschrieben werden (Algorithmus 2). Im ersten Schritt bekommt der Algorithmus ein Dreiecksgitter als Eingabe. Für jedes Dreieck werden zwei *Face Uses* mit entgegengesetzten *Face Use* Normalen erzeugt. Danach wird über die Kanten des Eingabegitters iteriert, wobei eine Einteilung der Kanten in drei Kategorien stattfindet (Abbildung 4.7):

- Kanten mit nur einem inzidenten Dreieck stellen Ränder dar. Für die Kante wird ein *Edge Use* erstellt, welches mit dem *Face Use* der Vorder- und Rückseite des Dreiecks verbunden wird.
- Besitzt die Kante genau zwei Dreiecke, so werden für beide Dreiecke insgesamt vier *Face Uses* vereint. Für die Kante sind nun zwei *Edge Uses* erforderlich, die jeweils zwei *Face Uses* miteinander verbinden.
- Handelt es sich um eine nichtmannigfaltige Kante, kann die Konnektivität der *Face Uses* wie folgt gefunden werden: Paare von adjazenten *Face Uses* in einem Gitter nutzen ihre gemeinsame Kante immer in entgegengesetzter Richtung [Garimella, 1999]. Gibt es mehrere *Face Uses*, die in Frage kommmen, so werden diese um die gemeinsame Kante geordnet und das *Face Use* mit dem kleinsten dihedralen Winkel zum gerade betrachteten *Face Use* wird mit diesem vereint (Abbildung 4.8). Algorithmus 3 beschreibt das Vorgehen im Detail.

Wurden nach der Generation der *Edge Uses* auch die *Vertex Uses* und *Shells* erzeugt, so erhält man als Ergebnis ein in mannigfaltige Dreiecksnetze (*Shells*) unterteiltes nichtmannigfaltiges Dreiecksgitter (Abbildung 4.9). Ein Vorteil der *Shells* ist, dass in jeder *Shell*

**Algorithmus 2** Aufbau der Datenstruktur zur Repräsentation von nichtmannigfaltiger Topologie

```
CREATE-NMSURFACE(TRIANGULIERUNG Surf)
 1  ▷ Erzeuge die leere Datenstruktur
 2  NMSurf ← 0
 3  ▷ Erstelle für jedes Dreieck der Eingabeoberfläche zwei Face Uses mit
 4  ▷ entgegengesetzten Face Use Normalen
 5  CREATE-FACE-USES(Surf, NMSurf)
 6  ▷ Erstelle Edge Uses in Abhängigkeit von der Anzahl der zur Kante
 7  ▷ inzidenten Dreiecke
 8  CREATE-EDGE-USES(Surf, NMSurf)
 9  ▷ Erstelle die Vertex Uses
10  CREATE-VERTEX-USES(Surf, NMSurf)
11  ▷ Speichere mannigfaltige Gebiete in Form von Shells
12  CREATE-SHELLS(Surf, NMSurf)
13  return NMSurface

CREATE-EDGE-USES(TRIANGULIERUNG Surf, NMDATENSTRUKTUR NMSurf)
1  for edgeNum ← 0 to length[Surf.edges]
2       do
3          if length[Surf.edges[edgeNum].triangles] = 1
4             then PROCESS-BOUNDARY-EDGE(edgeNum, Surf, NMSurf)
5             else if length[Surf.edges[edgeNum].triangles] = 2
6                     then PROCESS-MANIFOLD-EDGE(edgeNum, Surf,
7                          NMSurf)
8                     else PROCESS-NON-MANIFOLD-EDGE(edgeNum,
9                          Surf, NMSurf)
```

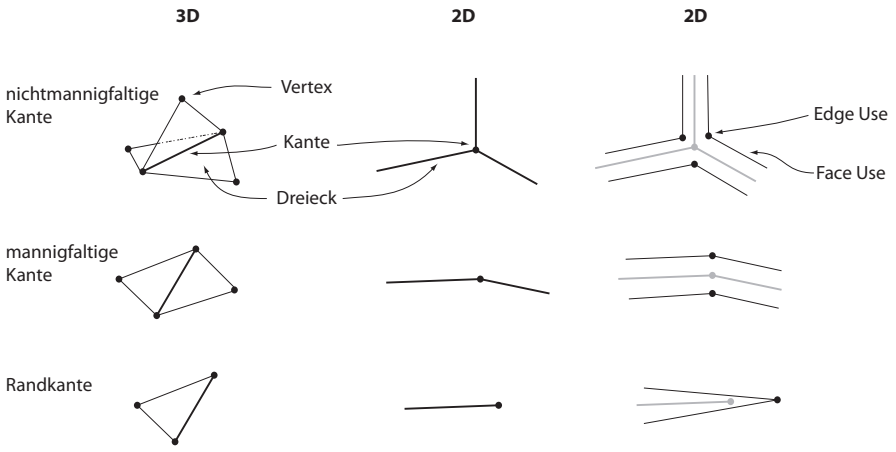



Abbildung 4.7: Die Anzahl der inzidenten Dreiecke an einer Kante bestimmt, wie die *Face Uses* erstellt werden.

**Algorithmus 3** Generation der *Edge Uses* an einer nichtmannigfaltigen Kante

```
PROCESS-NON-MANIFOLD-EDGE(GANZEZAHL edgeNum, TRIANGULIERUNG Surf, NMDATENSTRUKTUR NMSurf)
 1  edge ← Surf.edges[edgeNum]
 2  ▷ Hole alle Face Uses der zur Kante edge inzidenten Dreiecke
 3  FaceUses ← {fUse ∈ NMSurf.faceUses | fUse.triangleIdx ∈ edge.triangleIndices}
 4  for each currFaceUse ∈ FaceUses
 5       do
 6          currFaceUseTriangle ← Surf.triangles[currFaceUse.triangleIdx]
 7          ▷ Errechne den gerichteten Kantenvektor
 8          commonEdgeVector ← GET-CCW-EDGE-VECTOR(currFaceUseTriangle, edge)
 9          if currFaceUse.normal ≠ currFaceUseTriangle.normal
10            then commonEdgeVector ← commonEdgeVector · −1
11          PartnerFaceUses ← ∅
12          for each currFaceUse2 ∈ FaceUses
13               do
14                  if (currFaceUse = currFaceUse2)∨
15                  (currFaceUse.triangleIdx = currFaceUse2.triangleIdx)
16                    then continue
17                  currFaceUse2Triangle ← Surf.triangles[currFaceUse2.triangleIdx]
18                  ▷ Errechne den gerichteten Kantenvektor
19                  pEdgeVector ← GET-CCW-EDGE-VECTOR(currFaceUse2Triangle, edge)
20                  if currFaceUse2.normal ≠ currFaceUse2Triangle.normal
21                    then pEdgeVector ← pEdgeVector · −1
22                  if commonEdgeVector ≠ pEdgeVector
23                  ▷ Die beiden Face Uses teilen sich eine Kante in entgegengesetzter
24                  ▷ Richtung ⇒ currFaceUse2 ist ein möglicher Partner von currFaceUse
25                    then PartnerFaceUses ∪ {currFaceUse2}
26          ▷ Ordne alle möglichen Partner FaceUses um die gemeinsame Kante
27          minDihedralAngle ← ∞
28          minDihedralAngleFUse ← 0
29          for each pFaceUse ∈ PartnerFaceUses
30               do
31                  n1n2CrossProduct ← pFaceUse.normal × currFaceUse.normal
32                  n1n2AngleACos ← acos(pFaceUse.normal · currFaceUse.normal)
33                  alphaCos ← commonEdgeVector · n1n2CrossProduct
34                  dihedralAngle ← 0
35                  if alphaCos ≥ 0
36                    then dihedralAngle ← π − n1n2AngleACos
37                    else  dihedralAngle ← n1n2AngleACos + π
38                  if dihedralAngle < minDihedralAngle
39                    then minDihedralAngle ← dihedralAngle
40                         minDihedralAngleFUse ← pFaceUse
41          ▷ Verbinde die beiden Face Uses durch ein Edge Use
42          CREATE-EDGE-USE-FOR-FACE-USES(currFaceUse, minDihedralAngleFUse, Surf)
43          ▷ Entferne die verbundenen Face Uses aus der Suchmenge FaceUses
44          FaceUses\{currFaceUse, minDihedralAngleFUse}
```

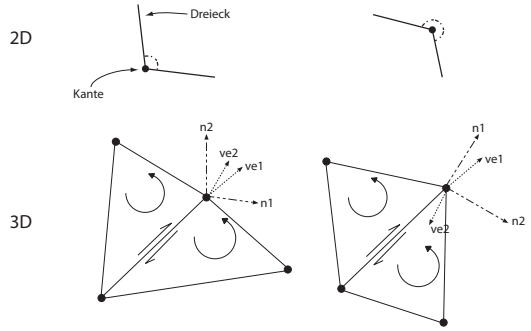



Abbildung 4.8: Handelt sich es um eine nichtmannigfaltige Kante, so müssen die *Face Uses* ermittelt werden, die über ein *Edge Use* miteinander verbunden werden sollen. Um das Paar von *Face Uses* zu bestimmen, werden zuerst die *Face Uses* aussortiert, deren Halbkanten in die gleiche Richtung zeigen. Zwischen den verbleibenden Kanten wird der Winkel mit Hilfe des Skalarprodukts bestimmt. Um Winkel über 180° zu erhalten, kann das Kreuzprodukt verwendet werden. Dazu wird der Kantenvektor $ve1$ bestimmt, der in Richtung eines vorher festgelegten Vertex zeigt. Nun wird das Kreuzprodukt $ve2$ aus den beiden *Face Use* Normalen berechnet ($n2 \times n1$). Ist das Skalarprodukt aus $ve1$ und $ve2$ negativ, so handelt es sich um eine konvexe Kante [Garimella, 1999].

alle *Face Use* Normalen konsistent orientiert sind. Dies ist auch bei inkonsistenter Orientierung der Normalen des Ausgangsdreiecksgitters gewährleistet. Diese Eigenschaft ist bei inneren Rändern wichtig, da es für sie keine eindeutige Orientierung gibt. Des Weiteren ist eine konsistente Normalenorientierung bei der Extrusion der Randschicht von großer Wichtigkeit.

Die einzelnen *Face Uses* der *Shells* sind mit Nachbarschaftsinformationen versehen, die nicht nur Operationen auf Elementen innerhalb der eigenen *Shell*, sondern auch den Zugriff auf Elemente von adjazenten *Shells* ermöglichen. D.h. das Eingabegitter zerfällt nicht unwiderbringlich in Mannigfaltigkeiten bzw. *Shells* - das ursprüngliche Eingabegitter kann aus den Nachbarschaftsinformationen zurückgewonnen werden bzw. ist fester Teil der Datenstruktur.

### 4.3.3 Initialisierung der Front

Bevor die Initialisierung der Oberfläche erfolgt, wird jede *Shell* in eine Datenstruktur kopiert, die beliebige Polygone speichert und gleichzeitig auch einzelne Parameter für jedes Gitterelement aufnehmen kann. Ersteres ist notwendig, da das ursprüngliche Dreiecksgitter bei Vorliegen von inneren oder äußeren Rändern mit Viereckselementen versehen wird (Abschnitt 4.3.5).

Der Initialisierungsschritt nimmt - basierend auf den vom Nutzer gewählten Bereichen - eine Klassifikation jedes Gitterelementes vor. So wird u.a. für die Gitterelemente gespeichert, welche volumetrischen Elemente auf ihnen erzeugt werden sollen. Bevor jedoch die Initialisierungsschritte genauer erläutert werden können, muss zuerst das Problem *Komplexe Geometrien*, erwähnt in Abschnitt 4.2.3, gelöst werden: Es konnten keine gültigen Prismen für Oberflächen mit stark variierenden Dihedralwinkeln erzeugt werden. Erst nach Lösung des Problems ist klar, welche Elementtypen in der Randschicht Verwendung finden.

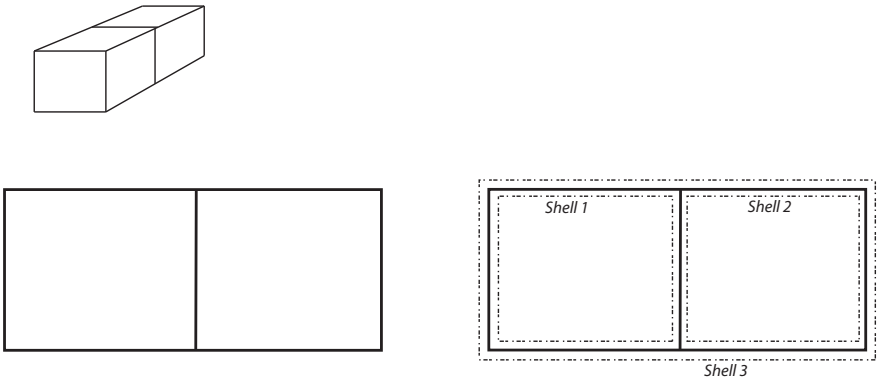



Abbildung 4.9: Eine einfache nichtmannigfaltige Geometrie kann man gewinnen, indem man zwei Würfel miteinander vereint. Die resultierende Geometrie ist ein Würfel mit einem inneren Rand. Erstellt man eine nichtmannigfaltige Repräsentation der Geometrie, so werden drei *Shells* erzeugt. Die Generierung der nichtmannigfaltigen Repräsentation kann man sich auch als Offset-Operation vorstellen, in der Offset-Oberflächen mit der Distanz Null innen und außen von der Eingabegeometrie erstellt werden.

### Mehrere Richtungsvektoren

Für den Dreiecksstern mit stark variierenden Dihedralwinkeln in Abbildung 4.3 konnten bei der Verwendung von nur einem Richtungsvektor keine gültigen Prismen erstellt werden. Eine Menge von intakten Randelementen kann nur gewonnen werden, wenn mehrere Richtungsvektoren $\vec{g}$ pro Vertex $v^0$ des Dreiecksgitters $Surf$ verwendet werden (Abbildung 4.10):

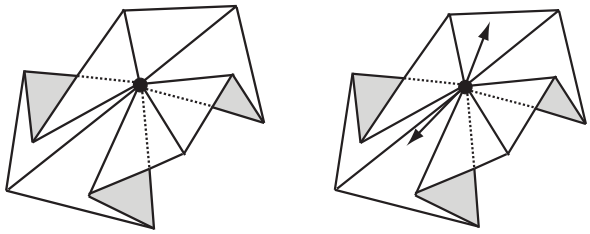
$$\forall v^0 \in Surf : \forall t^2 \in Star(v^0) \exists \vec{g} : \vec{g} \cdot Normal(t^2) > 0$$

Durch die Nutzung von mehreren Richtungsvektoren können in der Randschicht nicht nur Prismen verwendet werden. Es müssen *Übergangselemente* an den scharfen Kanten verwendet werden, die sich aus Tetraedern, Pyramiden und Hexaedern zusammensetzen. Die Auswirkungen der Verwendung von weiteren Elementtypen auf den Gittergenerierungsprozess ist in den folgenden Abschnitten beschrieben.

### Übergangselemente

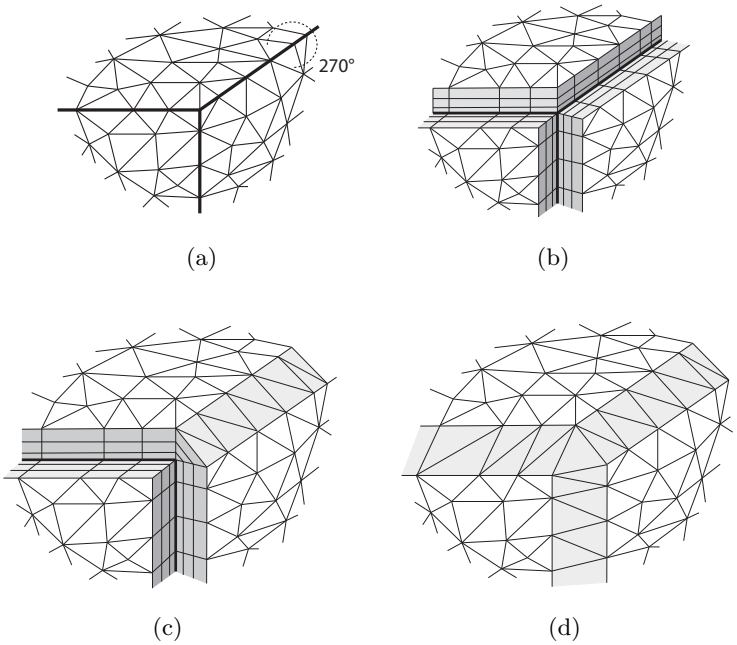
Die Verwendung von reinen Prismenrandschichten ist durch den Einsatz von mehreren Richtungsvektoren pro Vertex nicht mehr möglich (Abbildung 4.11). An Kanten des Dreiecksgitters, an denen mindestens ein Punkt der Kante mehr als einen Richtungsvektor aufweist, müssen Kantenübergangselemente eingefügt werden. An Vertices mit mehreren Richtungsvektoren werden Vertexübergangselemente eingesetzt.

Die Richtung jedes einzelnen Richtungsvektors errechnet sich aus den ihm zugehörigen Dreiecksnormalen. Übergangselemente werden bei der ersten Randschicht an Vertices und Kanten benötigt. In Abbildung 4.12 ist eine Übersicht der verwendeten Übergangselemente zu sehen. Die einzelnen Gruppen von Übergangselementen sollen nun näher erläutert werden.



(a) (b)

Abbildung 4.10: (a) Das Bild zeigt den Dreiecksstern, für den mit nur einem Richtungsvektor keine gültigen Prismen erzeugt werden konnten. (b) Erst durch den Einsatz von zwei Richtungsvektoren können defekte Randelemente vermieden werden. Dabei wird jedem Dreieck einer der beiden Richtungsvektoren zugewiesen - es bilden sich also zwei Dreiecksgruppen, für welche die Winkel zwischen den Dreiecksnormalen und dem zugeordneten Richtungsvektor kleiner als 90° sind.




(a) (b)

(c) (d)

Abbildung 4.11: (a) Auf der Abbildung ist ein Teil einer Dreiecksoberfläche zu sehen, der mit einer Randschicht versehen werden soll. Die Oberfläche besitzt scharfe Kanten mit einem Winkel von 270°. (b) Auf der Oberfläche werden Prismen erstellt. Es verbleiben die scharfen Kanten als Spalte zwischen den Prismenschichten. (c) An einem Kantenzug werden Übergangselemente eingefügt. (d) Die verbleibenden scharfen Kanten werden mit Übergangselementen versehen. Anzumerken ist, dass in dem dargestellten Beispiel die Übergangselemente keine Vorraussetzung für ein gültiges Gitter sind.

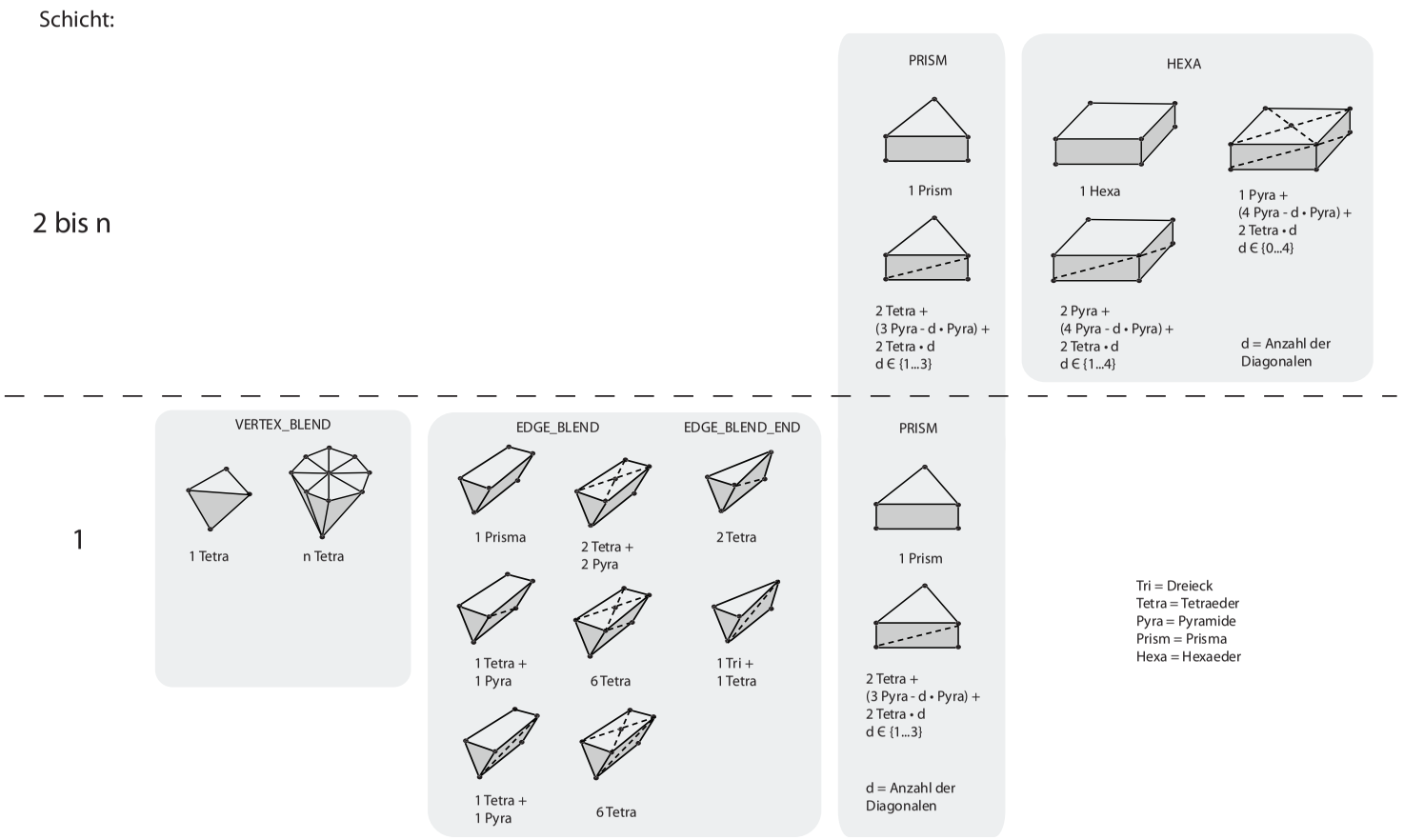



Abbildung 4.12: Die Übergangselemente sind nach ihrem Vorkommen in bestimmten Schichten geordnet. Einige Elemente wie `VERTEX_BLENDs` und `EDGE_BLENDs` sind nur in der ersten Randschicht anzutreffen. Des Weiteren findet eine Einteilung nach Elementtyp statt. `VERTEX_BLENDs` stellen Übergangselemente an Vertices dar, während `EDGE_BLENDs` Übergangselemente an Kanten sind. Neben dem `EDGE_BLEND` (Prisma) gibt es noch das `EDGE_BLEND_END`-Element (degeneriertes Prisma). Zusätzlich zu den Übergangselementen besteht die Randschicht aus Prismen (`PRISM`) und Hexaedern (`HEXA`) - letztere werden nicht in der ersten Randschicht verwendet.

### Übergangselemente an Vertices

Als Übergangselemente an Vertices (`VERTEX_BLENDs`) werden Tetraeder verwendet (Abbildung 4.12). Je nach Anzahl der am Vertex inzidenten Kantenübergangselemente verwendet man nur ein Tetraeder oder aber auch mehrere Tetraeder: Drei Kantenübergangselemente verlangen ein Tetraeder als Übergangselement am Vertex, während mehr als drei inzidente Kantenübergangselemente die gleiche Anzahl an Tetraedern am Vertex benötigen. Die dem Vertex gegenüberliegende Dreiecksseite des Tetraeders bildet ein Dreieck, auf welchem die nächste Randschicht erstellt wird. Ist vom Nutzer nur eine Randschicht gewünscht, so stellt das Dreieck ein Teil des Abschlusses der Randschicht dar und wird dem Tetraedergenerator als Teil der Ausgangsfront übergeben. Sollen jedoch mehr als eine Randschicht erstellt werden, so wird auf dem Dreieck ein Prisma erzeugt, welches dann Teil der nächsten Randschicht ist.

### Übergangselemente an Kanten

Es gibt zwei Typen von Übergangselementen an Kanten: Dies sind Kantenübergangselemente, die die Form eines Prismas besitzen (`EDGE_BLENDs`) und Übergangselemente in der Gestalt eines degenerierten Prismas `EDGE_BLEND_ENDs`. Ein degeneriertes Prisma wird als Abschluss einer Reihe von Prismenübergangselementen genutzt: Unterschreitet die als zuvor scharf markierte Kante den Schwellwert für die maximale Konvexität, so wird an dieser Stelle ein degeneriertes Prisma platziert. Da das fertige hybride Gitter im CGNS-Dateiformat exportiert werden soll, welches keine beliebigen Polyeder unterstützt, muss das degenerierte Prisma in andere Elementtypen unterteilt werden.

Da die beiden Typen von Kantenübergangselementen Viereckssseiten besitzen, muss es möglich sein, diese bei Bedarf in Dreiecke zu unterteilen. Eine Unterteilung der Viereckssseiten wird erforderlich, wenn die benachbarten Elemente Dreiecksseiten besitzen. Das Prisma besitzt ein Viereckselement, welches die Basis für ein Hexaeder der nächsten Schicht ist, während auf das Dreieck des degenerierten Prismas ein Prisma gesetzt wird. Für den Fall, dass nur eine Randschicht erstellt werden soll, muss die obere Viereckssseite des Prismas in zwei Dreiecke unterteilt werden (Abbildung 4.13).

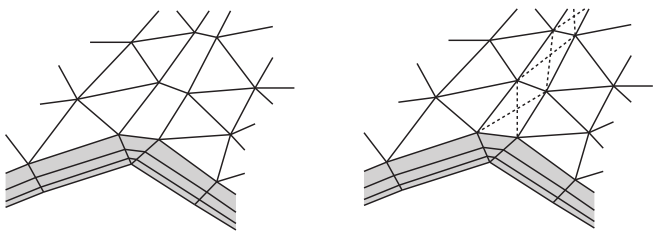


Abbildung 4.13: Der Abschluss der letzten Randschicht darf nur aus Dreiecken bestehen.

### Andere Übergangselemente

Der Einsatz von Übergangselementen hat auch Auswirkungen auf die benachbarten Prismen. Das Übergangselement an einem Vertex (`VERTEX_BLEND`) sowie das Prisma an einer Kante (`EDGE_BLEND`) haben keine Auswirkungen auf die Nachbarschaft, da sie die gleichen Seitenelementtypen besitzen. Die notwendige Unterteilung des degenerierten Prismas

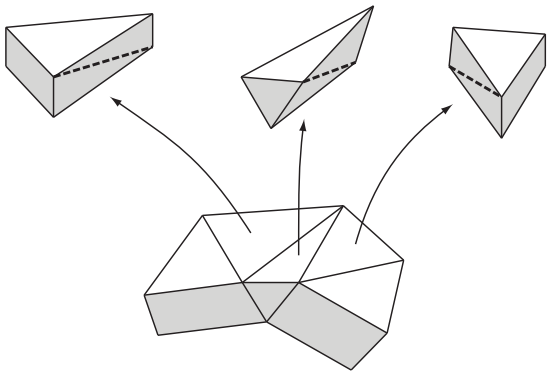


Abbildung 4.14: Das Bild zeigt einen Teil einer Randschicht, der aus vier Prismen und einem degenerierten Prisma besteht. Da das degenerierte Prisma unterteilt wird und daher die zwei Viereckssseiten in Dreiecke unterteilt werden, müssen auch die Viereckssseiten der zwei anliegenden Prismen passende Kanten aufweisen.

(`EDGE_BLEND_END`) führt jedoch zu Diagonalen, die auch in den benachbarten Prismenelementen vorhanden sein müssen. D.h. die Prismen, die eine Viereckssseite mit dem degenerierten Prisma gemeinsam haben, müssen auch unterteilt werden, um eine korrespondierende Diagonale zu erhalten. Deshalb wird eine spezielle Unterteilung für diese Prismen verwendet. Diese Unterteilung muss in der Lage sein, die Viereckssseiten des Prismas - je nach Anzahl der benachbarten degenerierten Prismen - mit Diagonalen versehen zu können (Abbildung 4.14). Die Unterteilung sollte am Besten keine weiteren Unterteilungen von Nachbarelementen zur Folge haben. Insbesondere bedeutet dies, dass die zwei Dreiecke des Prismas nicht modifiziert werden sollten. Eine Unterteilung des Dreiecks, auf welchem das Prisma der nächsten Schicht erstellt werden soll, wäre nicht besonders aufwändig zu behandeln - die Unterteilung zieht sich einfach durch die folgenden Randschichten - jedoch ist die Unterteilung des Basisdreiecks schwieriger zu behandeln. Unterteilt man das Basisdreieck, so muss diese Unterteilung auch in der möglicherweise anliegenden *Shell* weitergeführt werden. Deshalb wird die in Abbildung 4.12 dargestellte Unterteilung verwendet. Diese hat die Eigenschaft, dass die Nachbarschaft nicht weiter modifiziert werden muss und dass für jede Viereckssseite unabhängig eine Diagonale erstellt werden kann.

Das Hexaeder (`HEXA`) tritt nicht in der ersten Randschicht auf. Es wird ab der zweiten Randschicht auf die Prismenübergangselemente der Kanten gesetzt (`EDGE_BLENDs`). Auch die Seitenflächen des Hexaeders müssen die Möglichkeit besitzen in Dreiecke unterteilt zu werden, falls das Hexaeder an einem inneren oder äußeren Rand liegt oder wenn es am Ende der Randschicht positioniert ist. Hier kann fast das gleiche Unterteilungsmuster wie beim Prisma angewandt werden. Ist ein Hexaeder Teil der letzten Schicht, so muss das obere Viereck in Dreiecke unterteilt werden, da es schließlich dem Tetraedergenerator als Teil der Front übergeben wird.

#### Initialisierungsschritte

Da nun die möglichen Elementtypen feststehen, die in der Randschicht zum Einsatz kommen sollen, kann das Vorgehen im Initialisierungsprozess, welches für jede *Shell* separat ausgeführt wird, näher erläutert werden. Welche einzelnen Methoden im Ablauf der Initialisierung ausgeführt werden, ist aus Algorithmus 4 ersichtlich. Wie bereits erwähnt, werden die Dreiecke des Eingabegitters in eine Datenstruktur für beliebige Polygone kopiert

$$
\begin{array}{lcl}
cell = point^0 \vee edge^1 \vee poly^2 & & \\
cell.frontType & \in & \{NON\_FRONT, FRONT, BOUNDARY, \\
 & & FRONT\_END\_NON\_FRONT, \\
 & & FRONT\_END\_BOUNDARY\} \\
point^0.elementType & \in & \{NO\_ELEMENT, \\
 & & NO\_ELEMENT\_EDGE\_BLEND\_END, \\
 & & NO\_ELEMENT\_EDGE\_BLEND, \\
 & & VERTEX\_BLEND\} \\
edge^1.elementType & \in & \{NO\_ELEMENT, EDGE\_BLEND, \\
 & & EDGE\_BLEND\_END\} \\
poly^2.elementType & \in & \{NO\_ELEMENT, PRISM, HEXAEDRON\}
\end{array}
$$

Tabelle 4.1: In der Auflistung sind alle möglichen Front- und Elementtypen dargestellt, die einem Oberflächenelement zugewiesen werden können.

(Methode: Transfer-Use-Mesh-To-Poly-Mesh). Danach erfolgt - durch die Methode Init-Front - die Kategorisierung der Oberflächenelemente. Allen Dreiecken, Kanten und Vetrices wird eine Kombination aus Fronttyp und Elementtyp zugewiesen. Die Dreiecke wurden schon vom Nutzer per GUI in die Kategorien `NON_FRONT`, `FRONT` und `BOUNDARY` eingeteilt. Die Kategorie `NON_FRONT` bezeichnet Dreiecksbereiche, auf denen keine Randschicht erstellt werden soll, während die Markierung `FRONT` bedeutet, dass eine Randschicht generiert werden soll. Die Markierung `BOUNDARY` bezeichnet innere und äußere Ränder, durch welche die Randschicht weitergeführt werden soll.

Bis jetzt wurden nur Dreiecke markiert. Für die Randschichtgenerierung ist es jedoch auch wichtig die Vertices und Kanten geeignet zu kategorisieren - Kanten zwischen Bereichen wie der `FRONT` und der `BOUNDARY` werden z.B. mit `FRONT_END_BOUNDARY` markiert, während scharfe Kanten das Flag `EDGE_BLEND` oder `EDGE_BLEND_END` erhalten. Wurde allen Gitterelementen ein Fronttyp zugeteilt, kann - in Abhängigkeit von im Gitter vorliegenden scharfen Kanten - bestimmt werden, welche volumetrischen Elementtypen auf das jeweilige Oberflächengitterelement gesetzt werden sollen. Eine Übersicht aller möglichen Element- und Fronttypen ist in Liste 4.1 zu sehen. Eine Einteilung eines Dreiecksgitters in die genannten Kategorien ist beispielhaft in Abbildung 4.15 dargestellt.

**Algorithmus 4** Einzelne Schritte im Initialisierungsprozess

Boundary-Layer-Init(BLDatenstruktur *BLayer*)
1 ▷ Kopiere einzelne *Shells* in eine Datenstruktur für beliebige Polygone
2 Transfer-Use-Mesh-To-Poly-Mesh(*BLayer*)
3 ▷ Klassifiziere die Polygone, Kanten und Punkte
4 Init-Front(*BLayer*)
5 ▷ Füge die Übergangselemente als *virtuelle Polygone* ein
6 Create-Virtual-Elements(*BLayer*)
7 ▷ Ermittle innere Ränder
8 Find-Inner-Boundaries(*BLayer*)

In der ersten Randschicht treten Randzellen an Vertices, Kanten und Polygonen auf, während in den folgenden Schichten nur noch Zellen auf Polygonen erstellt werden. Um eine einheitliche Generierung der Randzellen zu ermöglichen, werden die Übergangselemente der ersten Randschicht nicht mehr an Punkten oder Kanten gespeichert, sondern durch die Methode Create-Virtual-Elements in Form von virtuellen Polygonen in die Polygonfront der ersten Randschicht eingefügt (Abbildung 4.16). Virtuell bedeutet, dass die

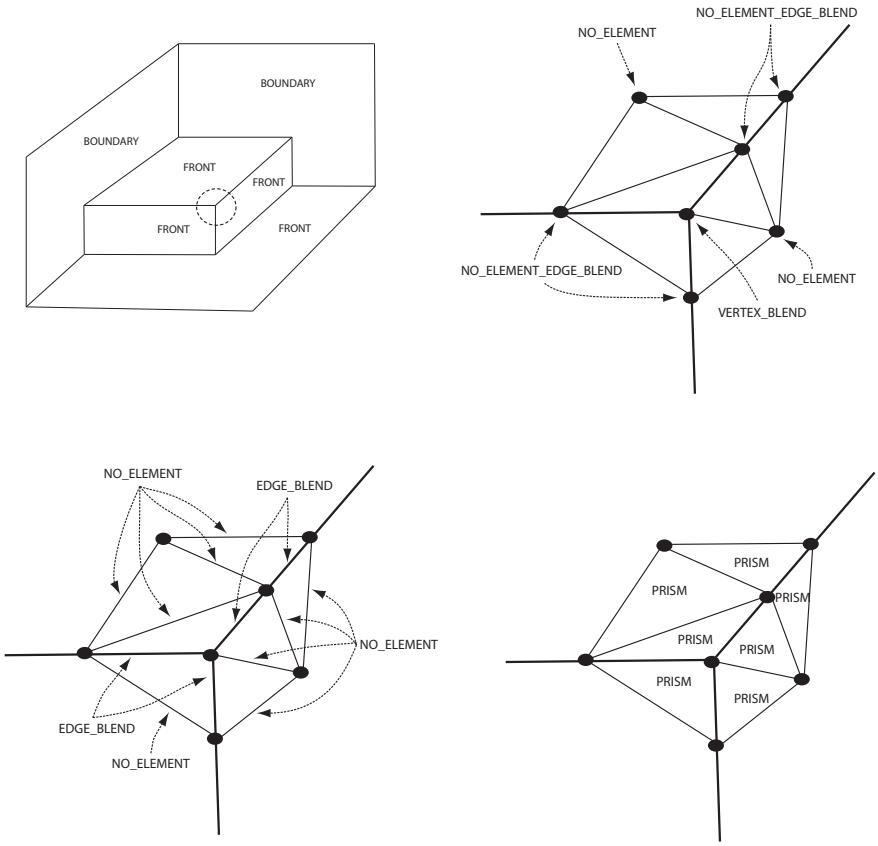



Abbildung 4.15: Jedem Element der Eingabegeometrie wird ein *Fronttyp* zugeordnet.

Polygone nur temporär vorhanden sind und in späteren Schritten, wie der Vereinigung der *Shells*, ignoriert werden. Dadurch ergibt sich, dass jedem Vertex der Front genau ein Richtungsvektor zugeteilt ist. Erst beim Entfernen der virtuellen Polygone sind mehrere Richtungsvektoren einem Vertex zugeordnet.

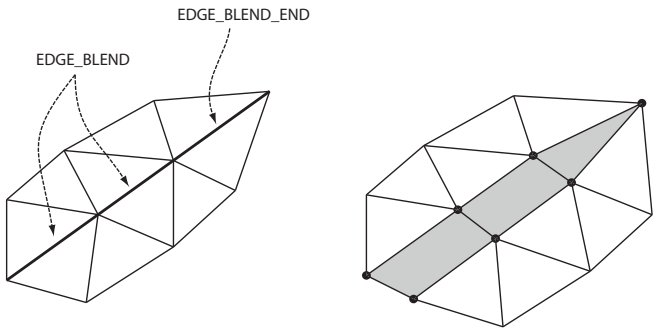



Abbildung 4.16: In der ersten Polygonschicht werden virtuelle Polygone eingefügt, um die Übergangselemente an Vertices und Kanten zu repräsentieren.

In der Methode Create-Virtual-Elements werden auch die Richtungsvektoren der ersten Randschicht errechnet. Die Richtung des Vektors ergibt sich aus allen ihm zugehörigen Dreiecken. Diese lassen sich ermitteln, indem man ausgehend von einer scharfen Kante des Sterns alle Dreiecke abläuft, bis man die nächste scharfe Kante oder wieder die Startkante erreicht - für alle Dreiecke zwischen zwei scharfen Kanten wird ein eigener Richtungsvektor erstellt, dessen initiale Richtung sich aus den zugehörigen Dreiecksnormalen ergibt (Abbildung 4.17).

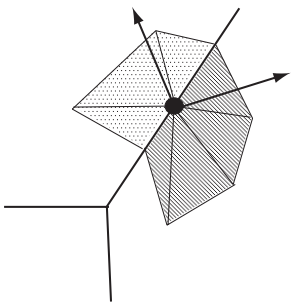


Abbildung 4.17: Die zwei Richtungsvektoren auf der scharfen Kante besitzen jeweils disjunkte Mengen an zugehörigen Dreiecken. Jede Dreiecksmenge ist durch einen anderen Grauton dargestellt.

Als letzter Schritt in der Initialisierungsphase werden die inneren bzw. äußeren Ränder in der *Shell* ermittelt (Methode: Find-Inner-Boundaries). Der Nutzer hat zwar schon die Randdreiecke mit `BOUNDARY` markiert, jedoch lässt sich aus dieser Information nicht ableiten, ob es sich um eine einzige Randregion innerhalb der *Shell* oder mehrere Regionen handelt (Abbildung 4.18). Deshalb werden mittels Breitensuche zusammenhängende Randregionen ermittelt und gespeichert.

Durch den Initialisierungsprozess sind alle nötigen Informationen ermittelt worden, die für die Generierung der ersten Randschicht erforderlich sind. Diese und die folgenden Randschichten werden im Konstruktionsprozess erstellt, der im nächsten Abschnitt beschrieben wird.

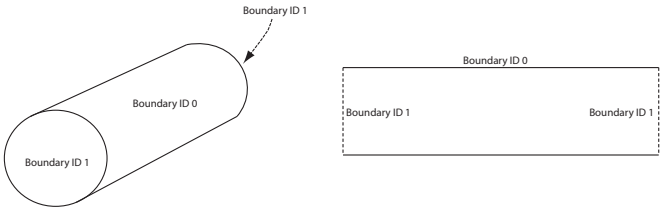



Abbildung 4.18: Die *Boundary ID 1* ist zweimal in der Geometrie vorhanden. Die einzelnen Regionen mit der *Boundary ID 1* sind jedoch nicht zusammenhängend.

### 4.3.4 Konstruktion der Prismenschichten

Das Ergebnis der Initialisierungsphase ist ein Polygongitter mit Informationen über den Front- und Elementtyp der einzelnen Polygone. Im Konstruktionsprozess werden schrittweise die neuen Randschichtelemente erzeugt. Die Offset-Polygone der aktuellen Randschicht bilden dabei die Polygonfront, auf der die nächste Schicht von Randelementen erstellt wird - eine Anzahl von $n$ Randschichten führt zu $n+1$ Polygongittern. Für jedes Polygon wird ein Volumenelement gespeichert, welches sich jeweils aus dem Polygon $poly_i$ des Polygongitters $i$ und dem darüberliegenden Polygon $poly_{i+1}$ des Polygongitters $i+1$ ergibt. Das letzte Polygongitter ist die Front für das Tetraedergitter und darf nur aus Dreiecken bestehen (Abbildung 4.19). Die einzelnen Polygongitter sind durch die Richtungsvektoren miteinander verbunden (Abbildung 4.20).

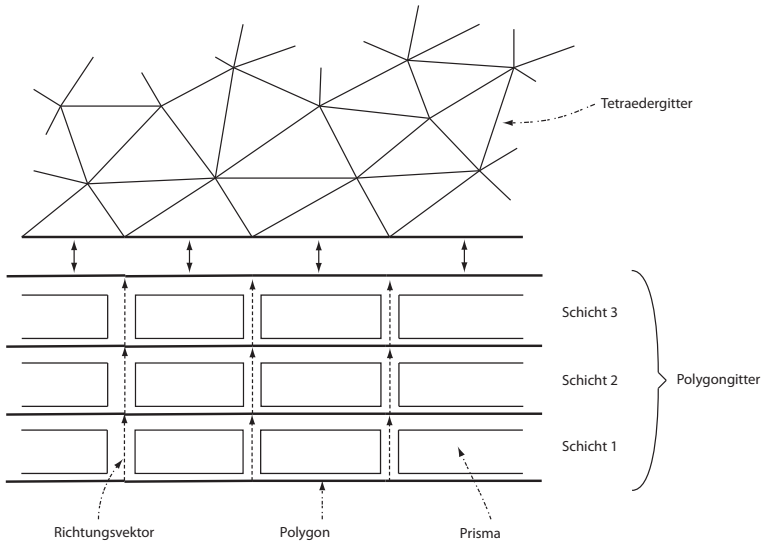



Abbildung 4.19: Auf jedem Frontpolygon werden volumetrische Elemente erstellt. Für die drei Randschichten in der Abbildung werden vier Polygonflächen erstellt. Die Richtungsvektoren verbinden die einzelnen Polygonflächen. Die letzte Polygonfläche wird dem Tetraedergenerator übergeben.

Der Ablauf der Konstruktionsphase ist in Algorithmus 5 dargestellt. Die Polygongitter werden einzeln durch die Methode Create-One-Layer erstellt. Danach wird die letzte Polygonfront der Randschichten auf Überschneidung mit der gegenüberliegenden Front geprüft (Fix-Front-Intersections). Bei Auftreten von Überschneidungen wird die Schichtdicke in den betroffenen Bereichen verringert. Sind alle Polygonnetze generiert worden und keine Überschneidungen mehr vorhanden, werden für jedes Polygon der Polygonnetze 0 bis $n-1$ Volumenelemente erzeugt (Create-Cells). Die Polygone des letzten Polygon-

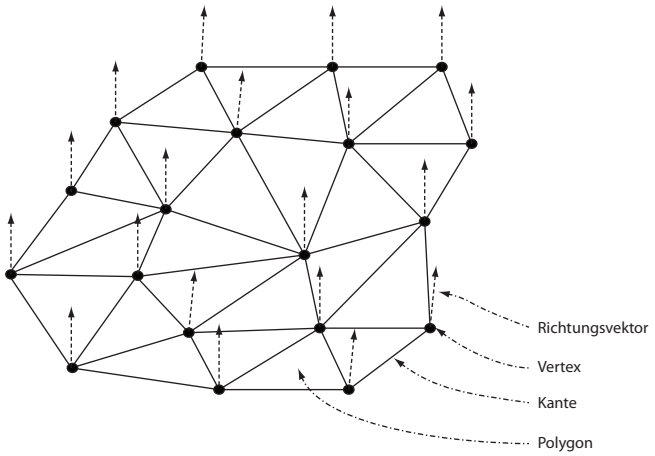



Abbildung 4.20: Auf jedem Frontvertex des Polygongitters wird ein Richtungsvektor erstellt.

gitters $n$ erhalten keine Volumenelemente, da sie die Front für den Tetraedergenerator bilden.

**Algorithmus 5** Ablauf des Konstruktionsprozesses

```
CREATE-LAYERS(BLDATENSTRUKTUR BLayer)
1  for layNum ← 0 to MAXNUMLAYERS
2       do ▷ Erstelle eine Randschicht
3          CREATE-ONE-LAYER(layNum, BLayer)
4  ▷ Behandle Überschneidungen der Randschichtfronten
5  FIX-FRONT-INTERSECTIONS(BLayer)
6  ▷ Generiere die Volumenelemente
7  CREATE-CELLS(BLayer)
```

Die Methode CREATE-ONE-LAYER wird ausführlich in Algorithmus 6 erläutert. CALC-HEIGHT-OF-NEXT-LAYER wird in den nächsten Abschnitten näher betrachtet.

### Lokale und globale Überschneidungen

In der Methode CALC-HEIGHT-OF-NEXT-LAYER aus Algorithmus 6 wird die Richtung und Länge der Richtungsvektoren bestimmt. Damit keine Defekte im späteren hybriden Gitter entstehen, muss das Problem der lokalen und globalen Überschneidungen gelöst werden. Um Überschneidungen zu verhindern, werden vier Vorgehensweisen genutzt: Auswerten der *Feature Size*, Vermeiden von Überkreuzungen der Richtungsvektoren, Glätten der Richtungsvektoren und Überprüfen auf Überschneidungen der gegenüberliegenden Polygonfronten.

Die Wahrscheinlichkeit für das Auftreten von lokalen und globalen Überschneidungen wird durch das Auswerten der *Gradient Limit Feature Size* verringert [Dyedov u. a., 2009]. Des Weiteren ermöglicht es die *Feature Size* zu geringe Abstände zwischen gegenüberliegenden Randschichten zu vermeiden. Dies hätte nämlich stark gestreckte Tetraeder zur Folge bzw. kann u.U. zum Abbruch des Tetraedergenerationsprozesses führen (Abbildung 4.21). Auch wenn keine Überschneidungen aufgetreten sind, sollte also ein Minimalabstand zwischen den Fronten eingehalten werden.

**Algorithmus 6** Generierung einer Randschicht

```
CREATE-ONE-LAYER(GANZEZAHL layNum, BLDATENSTRUKTUR BLayer)
 1  if layNum ≠ 0
 2     then ▷ die Richtungsvektoren der ersten Schicht wurden schon in der
 3          ▷ Initialisierungsphase erzeugt
 4          for each blPoint^0 ∈ BLayer.points_layNum
 5               do ▷ Erstelle einen Richtungsvektor für blPoint^0
 6                  CREATE-GROWTHLINE(blPoint^0, BLayer)
 7          ▷ Ermittle Nachbarschaftsinformationen für den Richtungsvektor
 8          CALC-GROWTHLINE-NEIGHBORS(layNum, BLayer)
 9  ▷ Errechne die Länge und Richtung jedes Richtungsvektors
10  CALC-HEIGHT-OF-NEXT-LAYER(layNum, BLayer)
11  ▷ Ertstelle die Vertices der nächsten Polygonschicht
12  for gLine ∈ BLayer.growthlines_layNum
13        do ▷ Verschiebe den Vertex der vorigen Polygonschicht, um einen Vertex der nächsten
14           ▷ Polygonschicht zu erhalten
15           gLinePoi0Idx ← gLine.pointIndices[0]
16           layerPoint ← BLayer.points[gLinePoi0Idx].coord + gLine.direction · gLine.length
17           nextLayPoint.coord ← layerPoint
18           nextLayPoint.frontType ← BLayer.points[gLinePoi0Idx].frontType
19           nextLayPointIdx ← length[BLayer.points]
20           gLine.pointIndices[1] ← nextLayPointIdx
21           BLayer.points[nextLayPointIdx]_layNum ← nextLayPoint
22  ▷ Erzeuge die Polygone für die nächste Randschicht mittels der Richtungsvektoren
23  ▷ jedes Polygons polygon^2
24  for polygon^2 ∈ BLayer.polygons_layNum
25        do
26           elemType ← polygon^2.elementType
27           switch(elemType)
28                  case NO_ELEMENT
29                       continue
30                  case PRISM
31                  case EDGE_BLEND_END
32                  case VERTEX_BLEND
33                       ▷ Erstelle das Dreieck für die nächste Randschicht
34                       CREATE-NEXT-TRIANGLE(layNum, polygon^2, BLayer)
35                       break
36                  case HEXAEDRON
37                  case EDGE_BLEND
38                       ▷ Erstelle das Viereck für die nächste Randschicht
39                       CREATE-NEXT-QUAD(layNum, polygon^2, BLayer)
40                       break
```

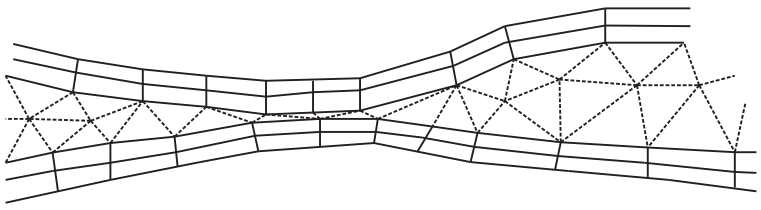


Abbildung 4.21: Ein zu geringer Abstand zwischen den Randschichtfronten führt zu stark deformierten Tetraedern.

Die *Feature Size* stellt den Abstand zwischen einem Punkt auf der Oberfläche und dem nächstliegenden Punkt auf der medialen Fläche der Oberfläche dar. Da der Abstand zur medialen Fläche zur Randschichtgenerierung genutzt werden soll und abrupte Höhenunterschiede vermieden werden sollen, wird der Gradient des *Feature Size* beschreibenden Skalarfeldes über die geometrische Auflösung der Randflächen beschränkt. Neben der Gradientenbeschränkung kann auch der minimale und maximale Wert des Skalarfeldes angegeben werden. Die eigentliche angenäherte *Feature Size* ist die Hälfte des lokalen Durchmessers $Diam$ der Oberfläche. Der Durchmesser an einem Vertex $p^0$ der Oberfläche $Surf$ wird durch das Schneiden der ins Oberflächeninneren orientierten Punktnormalen $\vec{n}_{p^0}$ mit der Oberfläche bestimmt. Ergeben sich mehrere Schnittpunkte, so wird derjenige mit dem geringsten Abstand zu $p^0$ genommen:

$$Diam[p^0] = \min\{\lambda | p^0 + \lambda \vec{n}_{p^0} \in Surf\}$$

Der Unterschied im Durchmesser $Diam$ zwischen den Kantenpunkten jeder Kante $edge^1$ wird in eine *Priority Queue* einsortiert, wobei die Schranke $G$ vom Nutzer gewählt werden kann. Der Gewichtungsschlüssel ergibt sich aus:

$$Key_{edge^1} = Diam[p_0^0] - (Diam[p_1^0] + G\|p_0^0 - p_1^0\|) \quad p_0^0, p_1^0 \in edge^1 \in Surf, G \in \mathbb{R}$$

Der genaue Ablauf der *Feature Size* Bestimmung ist in Algorithmus 7 dargestellt. Die Schnitttests der Methode Ray-Surface-Intersections können z.B. durch die Nutzung eines *Octrees* beschleunigt werden. Die Abbildung 4.22 zeigt das Ergebnis des Algorithmus für die Oberfläche eines Aneurysmas. Die *Feature Size* wird nicht für äußere und innere Randflächen bestimmt.

**Algorithmus 7** Bestimmung der *Feature Size*

```
Calc-Feature-Size(Triangulierung Surf, ReeleZahl G, ReeleZahl fsMin, ReeleZahl fsMax)
1  scalarField ← 0
2  ▷ Schneide die Punktnormalen mit der Oberfläche Surf
3  Ray-Surface-Intersections(Surf, scalarField, fsMin, fsMax)
4  ▷ Beschränke den Gradienten des Skalarfeldes scalarField
5  Limit-Gradient(Surf, scalarField, G)
6  return scalarField
```

Lokale Überschneidungen werden durch ein *Überkreuzen* der Richtungsvektoren eines Polygons verursacht. Durch Glättung der Richtungsvektoren kann die Wahrscheinlichkeit für eine Überkreuzung verringert werden. Eine gänzliche Vermeidung kann nur erreicht werden, wenn darauf geachtet wird, dass jedes Prisma konvex ist und positives Volumen besitzt. Stehen die einzelnen Extrusionsvektoren nicht parallel zu einander, so kann ab einer bestimmten Extrusionshöhe ein Verdrehen des Prismas auftreten - der Winkel zwischen der Normalen des Ausgangsdreiecks und der Normalen des Offset-Dreiecks ist größer als 90° [Erleben u. a., 2005]. Um eine Grenze für die maximale Prismenhöhe bzw. Extrusionshöhe zu erhalten, kann wie folgt vorgegangen werden. Die Positionen, der durch die Richtungsvektoren verschobenen Punkte, sind:

$$\vec{p'}_i(\lambda) = \vec{p}_i + \vec{n}_i\lambda \quad i \in \{0, 1, 2\}$$

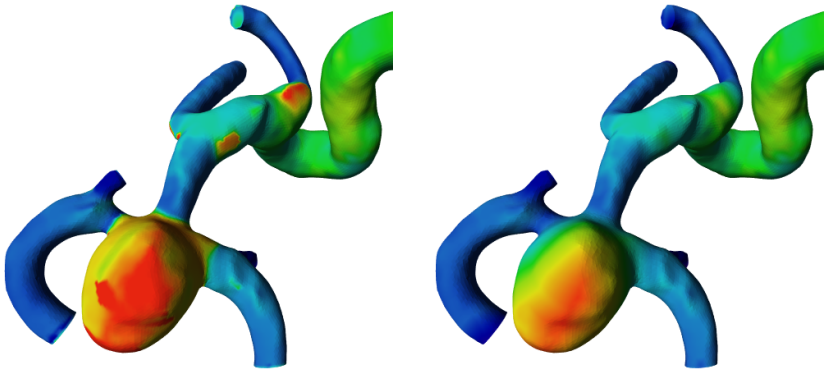
**Algorithmus 8** Hilfsmethoden zur Bestimmung der *Feature Size*

GRADIENT-LIMIT-VIOLATED(KANTE $edge$, TRIANGULIERUNG $Surf$, SKALARFELD $scalarField$, REELLE-ZAHL $keyValue$)

```
1  poi0Idx ← edge^1.points[0]
2  poi1Idx ← edge^1.points[1]
3  fsDiff ← |scalarField[poi0Idx] − scalarField[poi1Idx]|
4  maxAllowedDiff ← G · ||Surf.points[poi0Idx] − Surf.points[poi1Idx]||
5  keyValue ← fsDiff − maxAllowedDiff
6  if keyValue > 0
7     then ▷ Gradient ist zu stark
8          return TRUE
9  return FALSE
```

LIMIT-GRADIENT(TRIANGULIERUNG $Surf$, SKALARFELD $scalarField$, REELEZAHL $G$)

```
 1  ▷ Erstelle Priority Queue
 2  pHeap ← 0
 3  for each edge^1 ∈ Surf.edges
 4       do ▷ Überprüfe jede Kante auf zu großen Gradienten
 5          if GRADIENT-LIMIT-VIOLATED(edge^1, Surf, scalarField, keyValue)
 6             then pHeap.insert(keyValue, edge^1)
 7  while ¬pHeap.empty()
 8       do ▷ Hole Kante mit dem größten Gradienten
 9          maxEdge^1 ← pHeap.popMax()
10          poi0Idx ← maxEdge^1.points[0]
11          poi1Idx ← maxEdge^1.points[1]
12          maxAllowedDiff ← G · ||Surf.points[poi0Idx] − Surf.points[poi1Idx]||
13          ▷ Verringere den Wert an einem Kantenpunkt
14          if scalarField[poi0Idx] > scalarField[poi1Idx]
15             then lowPoiIdx ← poi0Idx
16                  scalarField[poi0Idx] ← scalarField[poi1Idx] + maxAllowedDiff
17             else lowPoiIdx ← poi1Idx
18                  scalarField[poi1Idx] ← scalarField[poi0Idx] + maxAllowedDiff
19          for epp^1 ∈ Star(Surf.points[lowPoiIdx])
20               do ▷ Überprüfe alle Kanten inzident zum veränderten Kantenpunkt lowPoiIdx
21                  if maxEdge^1 = epp^1
22                     then continue
23                  if GRADIENT-LIMIT-VIOLATED(epp^1, Surf, scalarField, keyValue)
24                     then
25                          if epp^1 ∈ pHeap
26                             then pHeap.rerank(keyValue, epp^1)
27                             else pHeap.insert(keyValue, epp^1)
28                     else
29                          if epp^1 ∈ pHeap
30                             then pHeap.delete(epp^1)
```



(a) (b)

Abbildung 4.22: (a) farbcodierte Darstellung des genäherten Abstandes von Rand und medialer Oberfläche; (b) gleiches Feld, nun mit beschränktem Gradienten

Die Normale des Offset-Dreiecks erhält man aus:

$$\vec{n'}(\lambda) = (\vec{p'}_1(\lambda) - \vec{p'}_0(\lambda)) \times (\vec{p'}_2(\lambda) - \vec{p'}_0(\lambda))$$

Umformen der Gleichung ergibt ein Polynom zweiter Ordnung:

$$\begin{aligned}\vec{n'}(\lambda) = &\underbrace{(\vec{n}_0 - \vec{n}_1) \times (\vec{n}_0 - \vec{n}_2)}_{\vec{c}} \\ &+ \underbrace{((\vec{p}_1 - \vec{p}_0) \times (\vec{n}_0 - \vec{n}_2) + (\vec{n}_0 - \vec{n}_1) \times (\vec{p}_2 - \vec{p}_0))\lambda}_{\vec{b}} \\ &+ \underbrace{((\vec{p}_1 - \vec{p}_0) \times (\vec{p}_2 - \vec{p}_0))\lambda^2}_{\vec{a}}\end{aligned}$$

Um nun ein positives Volumen und die Konvexität des Prismas zu gewährleisten, muss das Skalarprodukt zwischen der Normalen des Offset-Dreiecks und den Richtungsvektoren positiv sein.

$$\vec{n}_{0\ldots2} \cdot \vec{n'}(\lambda) > 0$$

Dies lässt sich als ein System von Ungleichungen formulieren:

$$\begin{bmatrix} \vec{n}_0 \cdot \vec{a} & \vec{n}_0 \cdot \vec{b} & \vec{n}_0 \cdot \vec{c} \\ \vec{n}_1 \cdot \vec{a} & \vec{n}_1 \cdot \vec{b} & \vec{n}_1 \cdot \vec{c} \\ \vec{n}_2 \cdot \vec{a} & \vec{n}_2 \cdot \vec{b} & \vec{n}_2 \cdot \vec{c} \end{bmatrix} \begin{bmatrix} \lambda^2 \\ \lambda \\ 1 \end{bmatrix} > 0$$

Der größte positive Wert für $\lambda$, der das Ungleichungssystem erfüllt, ist die maximale Prismenhöhe. Die Verwendung der maximalen Prismenhöhe kann in einigen Fällen zu einem Offset-Dreieck mit einem Flächeninhalt von Null führen. Dies sollte überprüft bzw. die maximale Extrusionshöhe sollte unterschritten werden. Das Verfahren kann auch auf andere Elementtypen als Prismen angewendet werden. Für Kantenübergangselemente kann es unverändert genutzt werden. Hexaeder müssen in zwei Prismen unterteilt werden, wobei der

kleinere Wert der zwei maximalen Extrusionshöhen als obere Grenze für die Hexaederhöhe genutzt wird.

Da nur eine Näherung der *Feature Size* errechnet wurde, können globale Überschneidungen weiterhin auftreten. Um dies gänzlich zu vermeiden, muss auf eine Überschneidungsfreiheit der Dreiecksfront geachtet werden, die dem Tetraedergenerator übergeben wird. Das genaue Vorgehen wird u.a. im nächsten Abschnitt erläutert.

#### Konstruktionsschritte

In diesem Abschnitt soll die Methode Calc-Height-Of-Next-Layer aus Algorithmus 6 näher beschrieben werden. Sie ist für die Bestimmung der Extrusionsvektorenrichtungen und Platzierung der Knotenpunkte der später durch die Methode Create-Cells erstellten volumetrischen Randzellen zuständig. Bei der Positionierung der Zellknoten müssen Überschneidungen vermieden werden. Dazu werden die im vorigen Abschnitt genannten Verfahren eingesetzt.

Algorithmus 9 zeigt die zwei Hauptschritte im Verfahren der Randschichtgenerierung. Zuerst wird in der Methode Init-Height-Of-Layer-0 die Höhe der ersten Randschicht bestimmt, danach wird die Position der Knotenpunkte des nächsten Polygongitters errechnet (Calc-Height-Of-Next-Layer-N).

In der Methode Init-Height-Of-Layer-0 wird zuerst die *Feature Size* errechnet. Danach wird die vom Nutzer vorgegebene Gesamtrandschichtdicke in Abhängigkeit der *Feature Size* pro Polygon leicht verringert bzw. erhöht (Algorithmus 10). Durch die zusätzliche Angabe einer minimal und maximal einzuhaltenen Schichtdicke kann sich so die Dicke der Randschicht an dem lokal vorhandenen Freiraum anpassen. Ob die vom Nutzer gewünschte Schichtdicke zu erreichen ist, wird über die *Feature Size* ermittelt. Durch die Wachstumsfunktion der Randschicht und der Gesamtschichtdicke kann die Höhe der ersten Randschicht bestimmt und in den einzelnen Polygonen gespeichert werden. Die Methode Calc-GLine-Dir berechnet die Richtungsvektoren, welche sich aus den durch den Innenwinkel gewichteten Normalen der zu den Richtungsvektoren inzidenten Polygonen errechnen:

$$gline.direction = \left\| \sum_{poly \in P} poly.normal \cdot \angle e_0, e_1 \right\|$$

$$P = \{BLayer.polygons[pIdx] | pIdx \in gline.polygonIndices\}$$
$$e_{0,1} \in poly \wedge e_0 \neq e_1 \wedge gline.pointIndices[0] \in e_{0,1}.pointIndices$$

Nach Berechnung der initialen Richtung wird überprüft, ob der Richtungsvektor das Sichtbarkeitskriterium verletzt. Ist dies der Fall, wird durch die Maximierung des minimalen Winkels zwischen dem Richtungsvektor und den Normalen des Polygonsterns versucht, dem Sichtbarkeitskriterium zu genügen. Ist dies nicht möglich, so muss der Gittergenerierungsprozess mit einem kleineren Wert gestartet werden, ab welchem eine Kante als scharf kategorisiert wird. Erfüllt der Richtungsvektor das Sichtbarkeitskriterium, so wird als letzter Schritt - falls der Richtungsvektor auf einem inneren oder äußeren Rand liegt - die Projektion des Richtungsvektors auf die Randfläche durchgeführt. Nach dem Ausführen der Methode Calc-GLine-Dir wird meistens eine Glättung der Richtungsvektoren durchgeführt (Smooth-GLines). Die Glättung erfolgt analog zur Bestimmung des Richtungsvektors - diesmal jedoch nicht durch Mittelung der inzidenten Polygone, sondern

durch Mittelung der benachbarten Richtungsvektoren. Auch hier wird eine Gewichtung durch die beiden inneren Winkel der zu den beiden Richtungsvektoren inzidenten Polygone durchgeführt (Abbildung 4.23). Liegt ein Richtungsvektor auf einem inneren oder äußeren Rand wird er nach jedem Glättungsschritt auf die Randfläche zurückprojektiert. Die Glättungsergebnisse in jeder Glättungsiteration werden in einem temporären Array gespeichert, so dass neu ermittelte Richtungsvektoren nicht schon den aktuellen Glättungsvorgang beeinflussen.

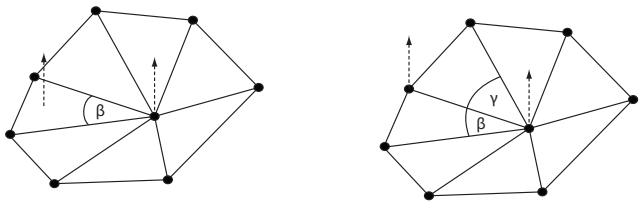



Abbildung 4.23: links: Winkelgewichtung der Dreiecksnormalen für die Vertexnormalenberechnung; rechts: Winkelgewichtung von einem Richtungsvektor zur Richtungsvektorglättung

Nun kann in der Methode Calc-Height-Of-Next-Layer-N basierend auf der Dicke der vorigen Randschicht (Compute-Cell-Height-From-Previous-Cell) die aktuelle Schicht $layNum$ bestimmt werden (Algorithmus 11). Im nächsten Schritt der Methode wird die Beschränkung des Gradienten der Höhenwerte durchgeführt, gefolgt von der Laplace-Glättung der Höhenwerte. Die Gradientenbeschränkung geschieht wie in Algorithmus 8 zur Beschränkung der *Feature Size*. Danach wird die Position der Punkte des nächsten Polygongitters bestimmt - kann die vorgegebene Zellenhöhe nicht erreicht werden, werden die betreffenden Polygone gespeichert und der Generierungsprozess mit geringeren Höhen für die markierten Polygone neu gestartet. Neustart bedeutet hier nicht, dass die aktuelle Randschicht neu berechnet wird, sondern alle Zellen neu bestimmt werden, welche sich unter den Polygonen befinden, die einen zu geringen Höhenwert aufweisen. Nur die Höhenwerte der letzten Randschicht neu zu berechnen ist nicht möglich, weil dadurch die vorgeschriebene Wachstumsfunktion verletzt werden würde - wären alle Randschichten gleich hoch, könnte nur die letzte Randschicht neu berechnet werden. Welche Polygone bei einer Neuberechnung der Höhe verändert werden müssen, zeigt Abbildung 4.24.

**Algorithmus 9** Bestimmung der maximalen Dicke einer Randschicht

```
Calc-Height-Of-Next-Layer(GanzeZahl layNum, BLDatenstruktur BLayer)
1  if layNum = 0
2     then Init-Height-Of-Layer-0(BLayer)
3  Calc-Height-Of-Next-Layer-N(layNum, BLayer)
```

In der Methode Recompute-Cells-Of-Polygons wird die Richtung der Richtungsvektoren aktualisiert und die Länge des Richtungsvektors neu bestimmt. Die Länge ergibt sich aus dem Mittelwert der Höhenwerte der zum Richtungsvektor inzidenten Polygone. Bei diesem Vorgang muss darauf geachtet werden, dass keine der zukünftig auf einem der inzidenten Polygone erstellten Zellen durch einen zu großen Höhenwert verdreht ist. Dies kann durch das Anwenden der in Abschnitt 4.3.4 vorgestellten Methode erreicht werden. Kann die im Polygon vorgegebene Höhe nicht erreicht werden, wird der Index des betreffenden Polygons in dem Array *shrinkPolyIndices* gespeichert. Alle Höhenwerte der Polygone

**Algorithmus 10** Errechnen der initialen Höhe der ersten Randschicht

```
INIT-HEIGHT-OF-LAYER-0(BLDATENSTRUKTUR BLayer)
 1  ▷ Erstelle ein Dreiecksgitter des ersten Polygongitters (ohne Polygone von Übergangselementen)
 2  Surf ← CREATE-SURFACE(BLayer, 0)
 3  scalarField ← CALC-FEATURE-SIZE(Surf, GLIMIT, GMIN, GMAX)
 4  polyFeatSize[] ← 0
 5  for each poly² ∈ BLayer.polygons₀
 6       do ▷ Berechne Feature Size für jedes Polygon
 7          if poly² ≠ FRONT
 8            then continue
 9          for each pIdx ∈ poly².pointIndices
10               do ▷ Der Durchmesser wird durch drei geteilt: 1/3 für die Randschicht,
11                  ▷ 1/3 für die Tetraeder und 1/3 für die gegenüberliegende Randschicht
12                  polyFeatSize[poly²] ← polyFeatSize[poly²] + scalarField[pIdx]/3
13          polyFeatSize[poly²] ← polyFeatSize[poly²]/length[poly².pointIndices]
14          ▷ Errechne die durchschnittliche Feature Size aller Polygone
15          averageFeatSize ← averageFeatSize + polyFeatSize[poly²]
16          featSizeValues ← featSizeValues + 1
17  averageFeatSize ← averageFeatSize/featSizeValues
18  for each poly² ∈ BLayer.polygons₀
19       do ▷ Berechne die angestrebte Höhe aller Zellen eines Polygons prefTotalThickness
20          if poly² ≠ FRONT
21            then continue
22          scaleFactor ← polyFeatSize[poly²]/averageFeatSize
23          ▷ Variiere die vom Nutzer vorgegebene Gesamtdicke TOTAL_THICKNESS in
24          ▷ Abhängigkeit von der Feature Size
25          prefTotalThickness ← TOTAL_THICKNESS · scaleFactor
26          if prefTotalThickness < MIN_TOTAL_THICKNESS
27            then prefTotalThickness ← MIN_TOTAL_THICKNESS
28          if prefTotalThickness > MAX_TOTAL_THICKNESS
29            then prefTotalThickness ← MAX_TOTAL_THICKNESS
30          ▷ Ist die angestrebte Gesamtdicke prefTotalThickness am Polygon zu erreichen?
31          totalLayThickness ← min(polyFeatSize[poly²], prefTotalThickness)
32          guess ← totalLayThickness/NUM_LAYERS
33          ▷ Errechne die Dicke der ersten Zelle des Polygons mit der vorgegebenen
34          ▷ Wachstumsfunktion func
35          poly².cellHeight ← NEWTON-RAPHSON(func, 0, totalLayThickness, guess)
36  gLineIndices[] ← 0
37  for gLineNum ← 0 to length[BLayer.growthlines₀]
38       do ▷ Bestimme die Richtung der Richtungsvektoren
39          CALC-GLINE-DIR(BLayer, gLineNum)
40          gLineIndices[length[gLineIndices]] ← gLineNum
41  ▷ Glätte alle Richtungsvektoren
42  SMOOTH-GLINES(GL_SMOOTH_NUM, gLineIndices)
```

im Array werden anschließend reduziert und die Berechnung der Richtungsvektoren - wie oben beschrieben - erneut ausgeführt.

Nach dem Erstellen der Polygonnetze samt der Randzelleninformationen folgt der nächste Schritt im Generierungsprozess: Die Viereckssseiten der Prismenrandschichten müssen ggf. in die Triangulierung von äußeren und inneren Rändern eingefügt werden. Das genaue Vorgehen beschreibt der nächste Abschnitt.

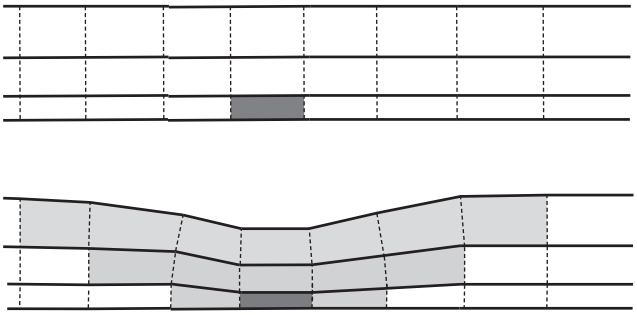


Abbildung 4.24: Die Höhenänderung eines Prismas hat die Neuberechnung der darüberliegenden Prismenzellen zur Folge.

### 4.3.5 Äußere und innere Ränder

Dadurch dass nichtmannigfaltige Geometrien als Eingabe akzeptiert werden, können innere Ränder, z.B. an Materialgrenzen, auftreten. Wird keine Randschicht auf dem inneren Rand gewünscht, so muss die Konnektivität der Elemente durch den inneren Rand weitergeführt werden (Abbildung 4.25). An inneren Rändern müssen also die beiden adjazenten *Shells* aufeinander abgestimmt werden. Äußere Ränder treten auf, wenn der Nutzer eine Weiterführung der Randschichtseitenelemente auf die Eingabegeometrie wünscht. Dies ist ein Spezialfall der inneren Ränder, bei dem jedoch nur eine *Shell* beteiligt ist und somit eine Synchronisation zwischen einer weiteren *Shell* entfällt. Innere bzw. äußere Ränder wurden in jeder *Shell* durch die Methode FIND-INNER-BOUNDARIES (Algorithmus 4) ermittelt.

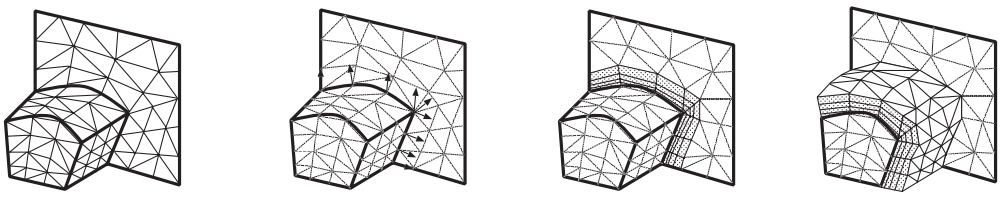


Abbildung 4.25: Die Viereckssseiten der Randschicht müssen in die innere Randfläche integriert werden.

Im Weiteren soll nur noch von inneren Rändern gesprochen werden, da der beschriebene Prozess für beide Rändertypen identisch ist. Ursprünglich bestehen die inneren Ränder nur aus Dreiecken. Mit Generierung der Randschicht müssen Viereckssseiten in die Vergitterung des inneren Randes eingefügt werden. Als Möglichkeiten bieten sich hier eine Neuvergitterung des inneren Randes oder aber eine Integration der Konnektivität in die vorhandene Vergitterung an. In dieser Arbeit wird das Integrationsverfahren verwendet,

**Algorithmus 11** Bestimmung der Höhe

```
CALC-HEIGHT-OF-NEXT-LAYER-N(GANZEZAHL layNum, BLDATENSTRUKTUR BLayer)
 1  fstPolyIndicesToRecompute[] ← 0
 2  currentLayPolyIndices[] ← 0
 3  ▷ Alle Polygonindices der aktuellen Schicht layNum werden in currentLayPolyIndices eingefügt
 4  for each pidx ∈ BLayer.polygons_layNum
 5        do
 6           if BLayer.polygons[pidx].frontType ≠ FRONT
 7              then continue
 8           currentLayPolyIndices[length[currentLayPolyIndices]] ← pidx
 9  for lay ← layNum to layNum
10        do ▷ Berechne die aktuelle Schichtdicke ausgehend von der Dicke der vorigen Schicht
11           if lay ≠ 0
12              then COMPUTE-CELL-HEIGHT-FROM-PREVIOUS-CELL(Blayer, currentLayPolyIndices)
13           ▷ Beschränke den Gradienten und speichere Polygone, deren Höhenwert verändert wurde
14           ▷ im Array alteredPolyIndices
15           alteredPolyIndices[] ← 0
16           LIMIT-GRADIENT-OF-LAYER-HEIGHT(BLayer, currentLayPolyIndices, alteredPolyIndices)
17           ▷ Füge die Indices des alteredPolyIndices zum currentLayPolyIndices Array hinzu
18           currentLayPolyIndices ∪ alteredPolyIndices
19           ▷ Glätte die Höhenwerte
20           SMOOTH-LAYER-THICKNESS(BLayer, H_SMOOTH_NUM, currentLayPolyIndices)
21           ▷ Aktualisiere die zum Polygon gehörenden Randzellen
22           shrinkPolyIndices[] ← 0
23           RECOMPUTE-CELLS-OF-POLYGONS(Blayer, lay, currentLayPolyIndices, shrinkPolyIndices)
24           if length[shrinkPolyIndices] > 0
25              then ▷ Einige Zellen konnten die vorgeschriebene Höhe nicht erreichen
26                   resFstPolyIndices[] ← 0
27                   GET-FST-LAYER-POLY-INDICES(BLayer, lay, shrinkPolyIndices, resFstPolyIndices)
28                   ▷ Verringere die Höhe aller Zellen der Polygone in shrinkPolyIndices bzw.
29                   ▷ resFstPolyIndices
30                   polyShrinkPercentages.fill(0.1, length[resFstPolyIndices])
31                   SHRINK-TOTAL-HEIGHT(BLayer, resFstPolyIndices, polyShrinkPercentages)
32                   if lay = layNum
33                   ▷ Die letzte Schicht wurde erreicht; alle Zellen mit zu geringer Höhe wurden neu
34                   ▷ berechnet; das fstPolyIndicesToRecompute Array kann zurückgesetzt werden
35                      then fstPolyIndicesToRecompute[] ← 0
36                   ▷ Alle neuen Zellen mit zu geringer Höhe werden in das
37                   ▷ fstPolyIndicesToRecompute Array eingefügt
38                   fstPolyIndicesToRecompute ∪ resFstPolyIndices
39                   ▷ Lösche das currentLayPolyIndices Array und starte die for Scheife neu
40                   currentLayPolyIndices[] ← 0
41                   currentLayPolyIndices ← fstPolyIndicesToRecompute
42                   lay ← −1
43              else
44                   if lay = layNum
45                      then break ▷ Letzte Schicht und alle Zellen habe die richtige Höhe
46                   ▷ Polygonindices der nächsten Schicht
47                   resNextLayPolyIndices[] ← 0
48                   ▷ Durch die Änderungen in der Höhe der aktuellen Schicht müssen Zellen der
49                   ▷ nächsten Schicht aktualisiert werden
50                   for clp ∈ currentLayPolyIndices
51                        do
52                           for gl ∈ BLayer.polygons[clp].growthLineIndices
53                                do
54                                   nextLayPoiIdx ← BLayer.growthLines[gl].pointIndices[1]
55                                   for poipoly ∈ BLayer.points[nextLayPoiIdx].polygonIndices
56                                        do
57                                           if poipoly ∉ resNextLayPolyIndices
58                                           then resNextLayPolyIndices[length[resNextLayPolyIndices]]
59                                           ← poipoly
60                   currentLayPolyIndices ← resNextLayPolyIndices
```

da somit ein Teil der ursprünglichen Vergitterung erhalten bleibt, an welche der Nutzer u.U. bestimmte Anforderungen gestellt hat, wie in bestimmten Bereichen gestreckte Dreiecke, die durch eine Neuvergitterung verloren gehen könnten. Außerdem können die äußeren Ränder relativ groß sein, so dass eine komplette Neuvergitterung zeitaufwändig wäre.

Ziel ist es also, den Kantenzug der letzten Prismenschicht in die Triangulierung einzufügen, die Dreiecke, die sich mit den Randschichtvierecken überlagern, zu entfernen und schließlich die Vierecke mit dem übriggebliebenen Dreieckskern zu verbinden. Um einen Kantenzug in eine vorgegebene Triangulierung einzubetten, gibt es zwei Ansätze. Die erste Möglichkeit ist es, die Knoten des Kantenzuges in die Triangulierung einzufügen und sie dann mit neuen Kanten zu verbinden. Neu erstellte Kanten, die vorhandene schneiden, müssen unterteilt werden, wodurch neue Knoten entstehen. Weitere Knoten sind in dem Fall der Randschichtgenerierung nicht erwünscht, da sie Auswirkung auf die Randschicht und die Ausgangsgeometrie haben: Es müssten neue Prismen und Vertices eingefügt werden. Um dies zu vermeiden, kann eine andere Methoden zur Kantenwiederherstellung genutzt werde. Hierbei werden die Knoten des Kantenzuges eingefügt, um danach durch Kippen der vorhandenen Kanten eine Kante zwischen zwei neu eingefügten Knoten zu erhalten. Es wurde bewiesen, dass die Kantenwiederherstellung mit letzterer Methode in 2D immer möglich ist. Da die inneren Ränder jedoch auch nicht planar sein können, wird das von Karamete u. a. [2001] vorgeschlagene Verfahren genutzt, welche eine Verallgemeinerung der zuvor genannten Methode für beliebige Oberflächen in 3D darstellt.

Das Einfügen der Seitenelemente des Randes wird in mehreren Schritten vollzogen (Abbildung 4.26). Zuerst werden die Punkte des obersten Kantenzuges der Seitenelemente in die Randtriangulierung eingefügt. Die Punkte können durch eine Projektion wieder auf einem Punkt zu liegen kommen oder auch Kanten oder Dreiecke unterteilen. Hier werden Toleranzbereiche genutzt, damit ein neuer Punkt nicht zu nahe an einem vorhandenen Punkt eingefügt wird, welches gestreckte Dreiecke zur Folge hätte. Nun werden alle Kanten ermittelt, die auf einer Linie zwischen zwei neu eingefügten Knoten liegen. Die Kanten werden nacheinander gekippt bis eine Kante zwischen den zwei Knoten vorhanden ist. Dabei darf eine gekippte Kante keine Überschneidungen im Dreiecksgitter hervorrufen und sich auch nicht mit der zukünftigen Kante überschneiden. Der genaue Ablauf des Algorithmus ist im Paper von Karamete u. a. [2001] nachzulesen. Wurden alle Kanten wieder hergestellt, können die Elemente des inneren Randes, welche unterhalb des eingefügten Kantenzuges liegen, entfernt werden. Anschließend wird der Dreieckskern geglättet [Zilske u. a., 2008] und die Vierecke mit dem Dreieckskern verbunden. Bei äußeren Rändern ist nun die Behandlung des Randes abgeschlossen. Innere Ränder erfordern eine zusätzliche Ermittlung der Korrespondenz zwischen den Knoten des Randes und beiden *Shells*.

### 4.3.6 Erstellen der Tetraeder

Nachdem die Seitenelemente der Randschichten in die zugehörigen Polygongitter der inneren und äußeren Ränder eingefügt wurden, ist die Dreiecksfront vollständig, die für die Generierung des Tetraedergitters nötig ist. Nun kann mit dem Tetraedergenerator von Amira, welcher im Advancing-Front-Verfahren arbeitet, ein Gitter generiert werden (Abbildung 4.27). Nach diesem Schritt ist der Gittergenerierungsprozess für eine *Shell* abgeschlossen: Sowohl die Zellen der Randschicht als auch das innere Tetraedergitter wurden erstellt. Was als Operation folgt, ist die Vereinigung der einzelnen *Shells* zu einem einzigen hybriden Gitter.

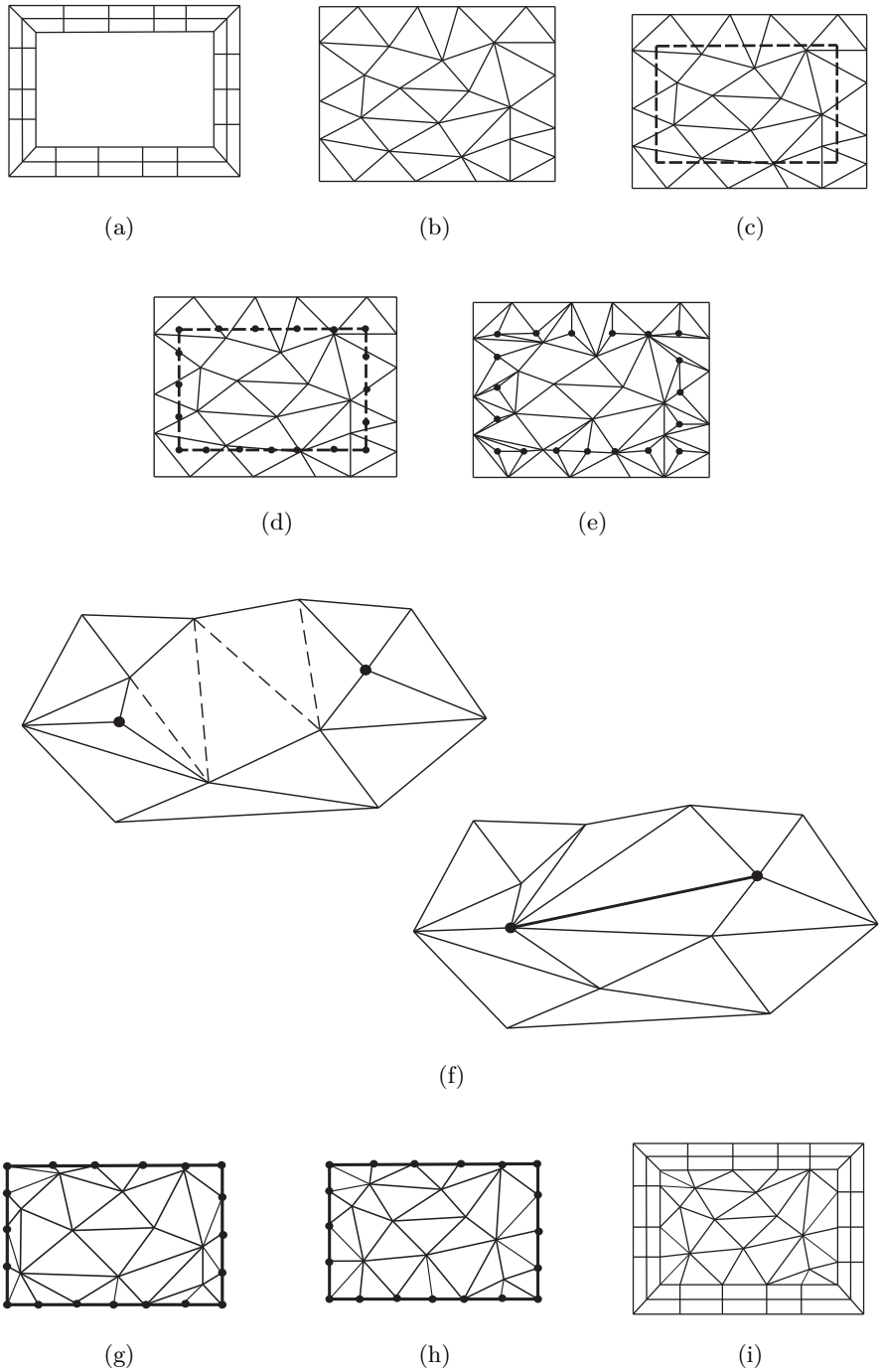



Abbildung 4.26: (a) Viereckssseiten der Randschicht; (b) innerer Rand der Eingabeoberfläche; (c) Auftragen des obersten Kantenzuges der Randschicht; (d) Auftragen der Punkte des Kantenzuges; (e) Einbetten der Punkte des Kantenzuges mittels Kanten- und Polygonteilung; (f) Kantenwiederherstellung durch Kantenkippen; (g) Entfernen der nicht benötigten Dreiecke; (h) Glättung des Gitters; (i) Polygongitter nach Vereinigung des Dreiecksmit dem Viereckssgitter

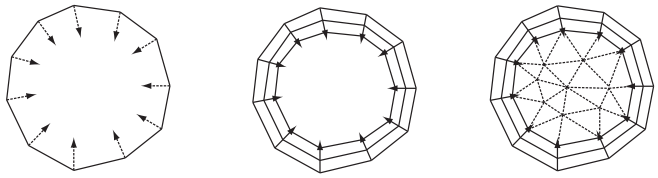


Abbildung 4.27: Nach Ermittlung der Richtungsvektoren und Konstruktion der Randschicht kann das Tetraedergitter generiert werden.

### 4.3.7 Vereinigung der Shells und CGNS-Export

Durch die Datenstruktur zur Repräsentation von nichtmannigfaltigen Oberflächen wurde die Eingabeoberfläche in *Shells* unterteilt, die separat und größtenteils unabhängig voneinander bearbeitet wurden. Nun müssen diese jedoch vereint werden. Hierzu werden die Eingangsoberfläche und die Informationen über innere Ränder hinzugezogen, um eine korrekte Konnektivität zu ermöglichen und duplizierte Punkte zu vermeiden. Mögliche Schnittstellen sind in Abbildung 4.28 dargestellt. Insbesondere die Dreiecksfront für das Tetraedergitter kann sich aus mehreren Dreiecksgittern zusammensetzen.

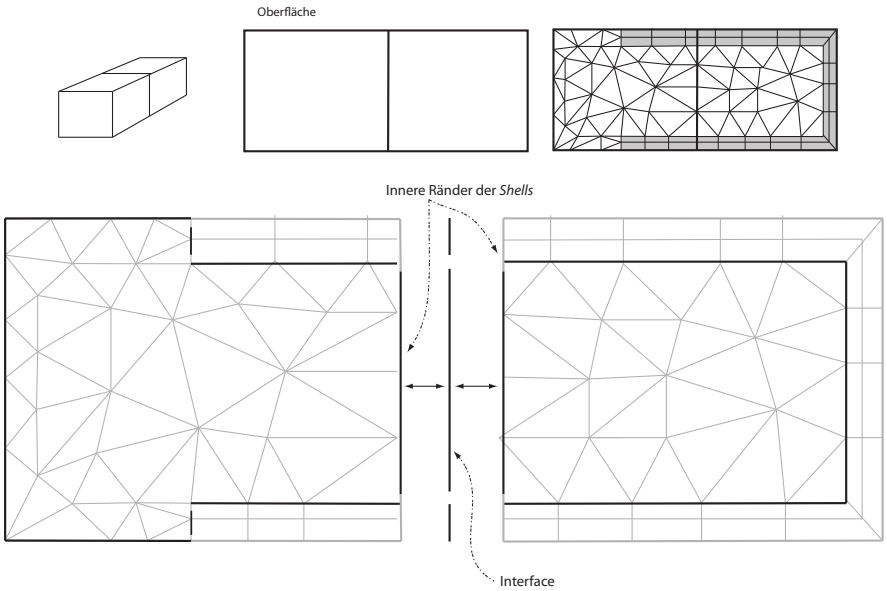



Abbildung 4.28: Bevor das Tetraedergitter erstellt werden kann, müssen die einzelnen Dreiecksoberflächen zusammengefügt werden.

Falls gewünscht, wird das Gitter in eine CGNS-Datei geschrieben. Das CGNS-Format ist ein portables, erweiterbares und gut dokumentiertes Dateiformat zur Speicherung von Daten, die bei einer Fluidsimulation auftreten (Gitter, Simulationsergebnisse, etc.). Die Etablierung eines Standards zum Austausch von CFD-Daten wurde von der NASA und Boeing initiiert. Die Entscheidung über Veränderungen und die Pflege des CGNS-Standards unterliegt heute dem *CGNS Steering Committee*, welches sich aus einer Reihe von internationalen Repräsentanten aus der Wirtschaft und Forschung zusammensetzt. Die Bibliothek, die zur Speicherung der CGNS-Datei benötigt wird, wurde in mehreren Sprachen als Open-Source-Software implementiert.

In die CGNS-Datei wird das Randpolygongitter samt der zugehörigen *Boundary Ids* geschrieben. Anschließend werden die einzelnen Zellen zusammen mit der *Material ID* gespei-

chert. Damit ist die Sicherung des hybriden Gitters abgeschlossen.

## 4.4 Implementierung

Der Gittergenerator zur Generierung von hybriden Gittern wurde als Modul in das Visualisierungs- und Analysewerkzeug Amira implementiert. Es wurde die Programmiersprache C++ und die Bibliotheken Qt, STL und Boost verwendet. Neben der objektorientierten Programmierung wurde auf die Wiederverwendbarkeit der beteiligten Klassen, wie der Datenstruktur für nichtmannigfaltige Oberflächen, geachtet. Die Verwendung von Smartpointern und Iteratoren führt zu übersichtlichem und sicherem Code.

### 4.4.1 Übersicht der verwendeten Klassen

In Abbildung 4.29 ist ein UML-Klassendiagramm des implementierten Gittergenerators zu sehen. Die grau hinterlegten Klassen waren bereits in Amira vorhanden. Objekte der Klassen, deren Namen ein `Hx` vorangestellt ist, verfügen über eine GUI und können direkt im Amira-Objektpool erstellt und verwendet werden. Möchte man die Funktionalität der anderen Klassen nutzen, so ist dies durch Aufrufen über Tcl-Methoden oder C++ Code möglich.

Möchte man ein hybrides Gitter erstellen, so erzeugt man sich für eine Oberfläche (`HxSurface`) ein `HxBoundaryLayerGen2` Modul. Über die GUI des Moduls können die Benutzereinstellungen, wie Anzahl der Schichten, vorgenommen werden und die Regionen, auf denen eine Randschicht extrudiert werden soll, eingestellt werden. Ist dies abgeschlossen, wird der Gittergenerierungsprozess gestartet. Dazu wird ein `SurfaceLayerGen` Objekt erstellt. Dieses analysiert die Eingabeoberfläche, erzeugt das `NMSurface` Objekt, welches die Topologie einer nichtmannigfaltigen Oberfläche repräsentieren kann, und sammelt alle für den Generierungsprozess wichtigen Parameter. Dies wurde in der `SurfaceLayerGen` Klasse implementiert und nicht in der `HxBoundaryLayerGen2` Klasse, da letztere eher als eine GUI-Klasse anzusehen ist, in der keine eigentliche Programmfunktionalität implementiert werden sollte. Des Weiteren könnte z.B. an dieser Stelle eine `TetraLayerGen` Klasse eingefügt werden, um das Modul um einen Gittergenerationsansatz zu erweitern, der ein Tetraeder als Eingabe erhält und mittels Deformation eine Randschicht einfügt, um dadurch ein hybrides Gitter zu generieren. Das `SurfaceLayerGen` Objekt erstellt für jede *Shell* ein `BoundaryLayer` Objekt. Die `BoundaryLayer` Klasse ist eine der Kernklassen. Sie repräsentiert das gesamte Gitter einer *Shell*, welches sich aus den einzelnen Randschichten (`BLPolygonMesh`) und dem Tetraedergitter (`HxTetraGrid`) zusammensetzen kann - der Nutzer kann jedoch auch ein reines Tetraedergitter oder kein Gitter für diese *Shell* verlangen. Das `BoundaryLayer` Objekt nutzt ein modifiziertes Polygongitter (`BLPolygonMesh`), um die Ober- und Unterseiten der Randschichten in Form von Polygonen zu speichern. Die Bestimmung der maximalen Schichtdicke geschieht durch das `BoundaryLayerGLFS` Objekt, welches vom `BoundaryLayer` Objekt abgeleitet ist. So kann der Gittergenerator bei Bedarf durch eine weitere Methode zur Schichtdickenbestimmung erweitert werden. Mit dem `HxFeatureSize` Objekt errechnet das `BoundaryLayerGLFS` Objekt die angenäherte *Feature Size*. Das `BoundaryLayer` Objekt speichert die in einer *Shell* gefundenen inneren und äußeren Ränder jeweils in einem `InnerBoundary` Objekt ab. Nachdem alle *Shells* mit einem Gitter versehen wurden, ermittelt das `SurfaceLayerGen` Objekt die gemeinsamen inneren Ränder unter Nutzung der `InnerBoundary` Objekte und speichert die gefundenen korrespondierenden Ränder in einem `InnerBoundaryInterface` ab. Danach vereint das

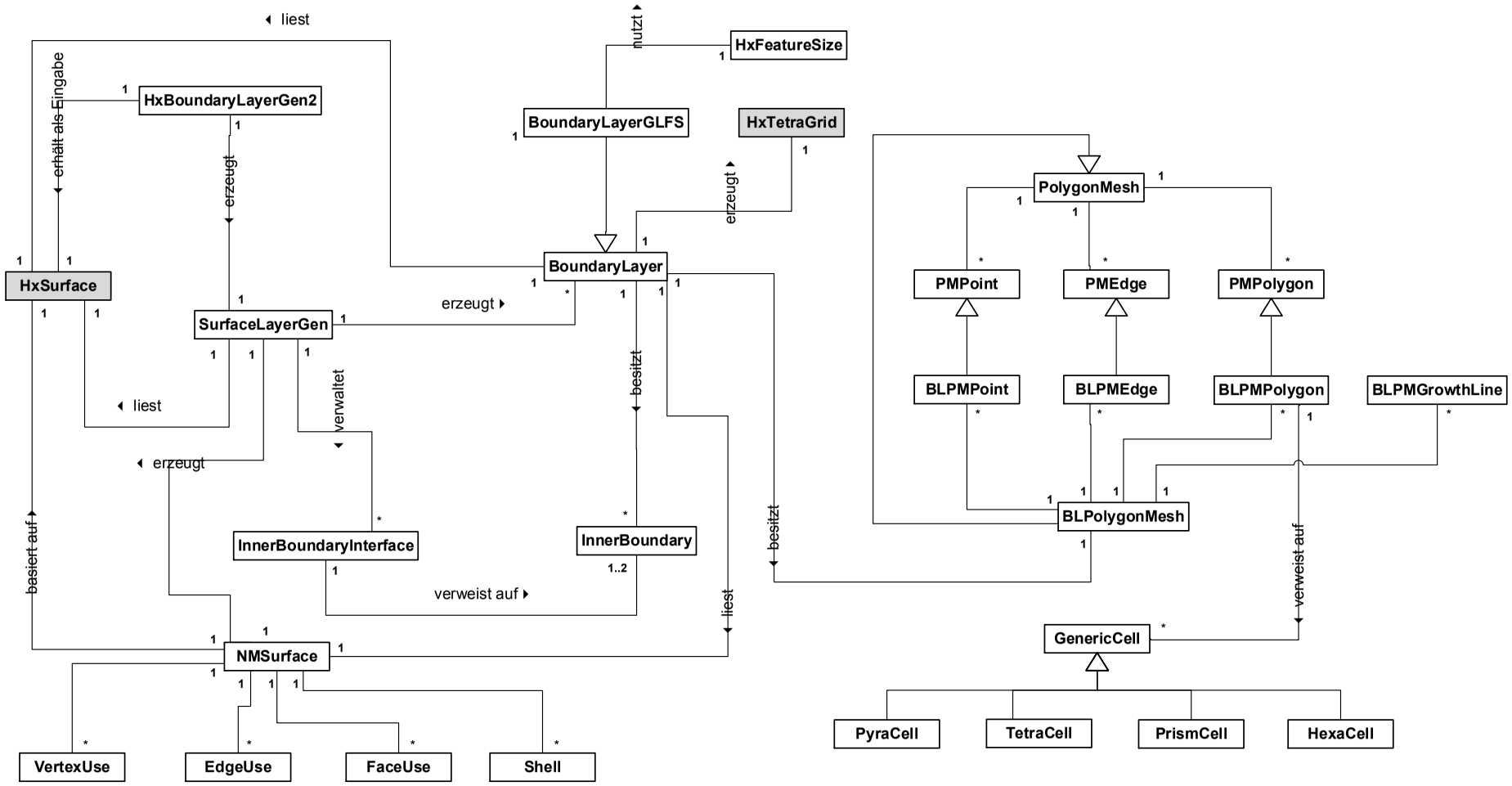



Abbildung 4.29: Die Abbildung zeigt ein UML-Klassendiagramm. Es wurden nur die wichtigsten Klassen abgebildet - Klassen, die zur GUI gehören oder Hilfsfunktionalität implementieren, sind nicht dargestellt.

SurfaceLayerGen Objekt die einzelnen *Shells* zu einem hybriden Gitter, welches nun als CGNS-Datei gespeichert werden kann.

### 4.4.2 Repräsentation des hybriden Gitters

Das Amira-Framework speichert Geometrien nach dem Prinzip der Randrepräsentation (Hierbei wird ein Körper über seine Begrenzungen beschrieben) als reine Dreiecksgitter in einer indexbasierten Datenstruktur (HxSurface), die Nichmannigfaltigkeiten abbilden kann. Die Datenstruktur legt der Geometrie keine topologischen Beschränkungen auf, wie es bei anderen Datenstrukturen der Fall ist [Campagna u. a., 1998], und kann daher universell eingesetzt werden.

Es wurde schon erwähnt, dass basierend auf einem HxSurface und einem NMSurface ein BoundaryLayer Objekt für jede *Shell* erstellt wird. In der *Shell* wird ein BLPolygonMesh Objekt zur Repräsentation der Randschicht und ein HxTetragrid Objekt zur Speicherung des Tetraedergitters verwendet. Die genaue Repräsentation der Randschicht soll in diesem Abschnitt beschrieben werden.

In Abbildung 4.30 ist das Zusammenspiel der einzelnen Klassen dargestellt. Eine HxSurface wird als Eingabe entgegengenommen. Dieses Dreiecksgitter wird mit Informationen über nichtmannigfaltige Regionen durch die NMSurface Klasse angereichert. Die Dreiecksoberfläche wird in einzelne Shells aufgeteilt.

Das BLPolygonMesh kann im Gegensatz zum HxSurface und NMSurface mit beliebigen Polygonen umgehen, da es von der PolygonMesh Klasse abgeleitet wurde. Die PolygonMesh Klasse kann Polygongitter speichern und bietet eine Reihe von grundlegenden Funktionen an, die die Arbeit mit Polygonen erleichern. Die einzelnen Elemente des Polygongitters sind der PMPoint, die PMEdge und das PMPolygon. Wichtig bei der Konzeption der PMPolygon Klasse war die Erweiterbarkeit der einzelnen Gitterelemente um weitere Attribute. Ein Polygon sollte z.B. leicht um ein Attribut *FrontType* erweitert werden können, um zu speichern, ob das Element Teil der Randschicht, eines inneren Randes oder einer Region ist, auf der keine Randschicht erstellt werden soll. D.h. die Gitterelemente sollten statisch um neue Attribute angereichert werden können. Um dies zu realisieren, kann das Factory-Pattern oder die Template-Programmierung genutzt werden. Die Wahl fiel auf die Template-Programmierung: Die einzelnen Elementklassen können als Templateparameter übergeben werden. Die Gitterelemente werden zusammenhängend im Speicher in Arrays gespeichert, um eine schnellen Zugriff bzw. Iteration über die Elemente zu ermöglichen. Das Factory-Pattern wurde nicht verwendet, da es auf Pointern basiert und damit der Vorteil des zusammenhängenden Arrayspeichers zunichte gemacht werden würde, außerdem ist die new-Operation in C++ relativ teuer und daher bei sehr häufiger Nutzung zu vermeiden. Durch einfache Ableitung der PolygonMesh Gitterelemente und Ergänzung der gewünschten Attribute können die Elemente für die BLPolygonMesh Datenstruktur gewonnen werden, welche ihrerseits von der PolygonMesh Template-Klasse abgeleitet wurde.

Die Datenstrukturen wurden so konzipiert, dass andere Klassen, die auf die Datenstruktur zugreifen möchten, nur wenig über die Form der Speicherung wissen müssen. Es wird versucht, die Daten von den Algorithmen zu trennen. Dies kann durch Iteratoren gewährleistet werden und durch Schnittstellen, die nur sehr wenig über die Datenstruktur preisgeben. So kann das SurfaceLayerGen Objekt das hybride Gitter ins CGNS-Dateiformat exportieren, indem es über ein Array aus Zellen (GenericCell) iteriert - es muss also relativ wenig über die BLPolygonbMesh Datenstruktur bekannt sein.

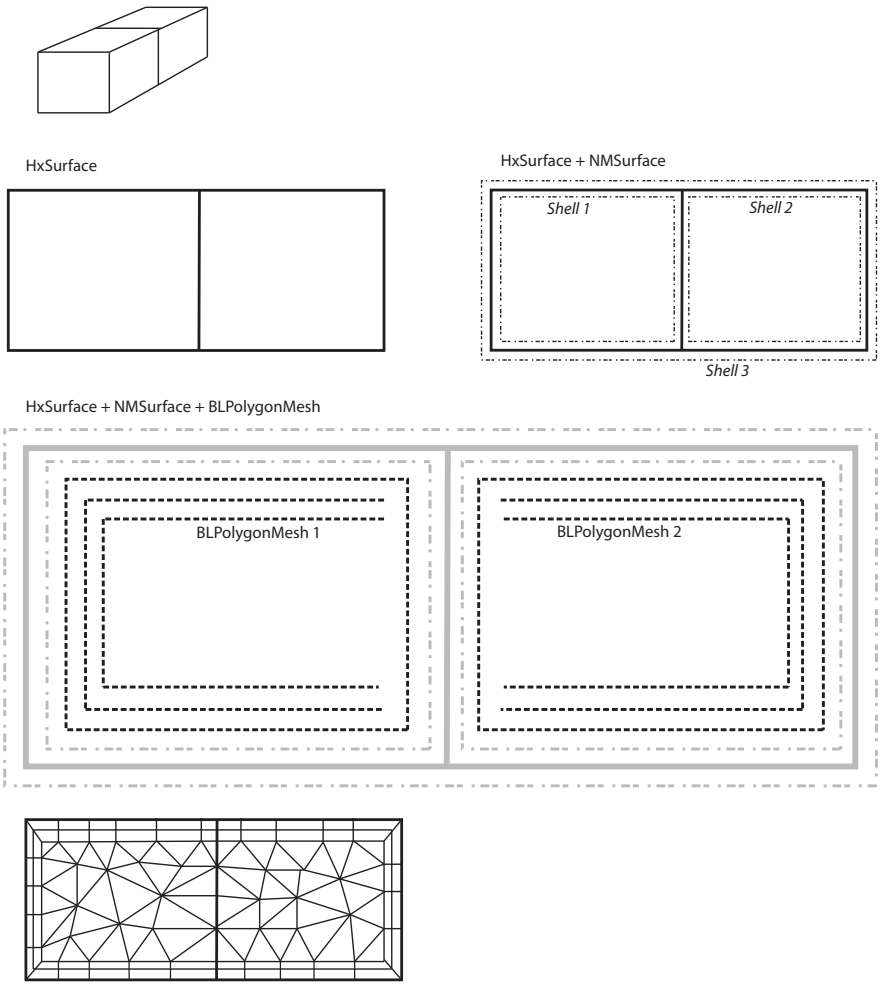



Abbildung 4.30: Die Eingabeoberfläche liegt als `HxSurface` vor und wird mit Informationen über Nichtmannigfaltigkeiten angereichert (`NMSurface`). Die Randschichten der einzelnen *Shells* werden mit Hilfe eines `BLPolygonMesh` repräsentiert.

### 4.4.3 GUI

Der Generator für hybride Gitter wurde als Modul in das Visualisierungsprogramm Amira integriert. Als Eingabe erhält das Modul Oberflächentriangulierungen des Typs `HxSurface`, welche durch die von Amira bereitgestellten Module für den Gittergenerierungsprozess vorbereitet werden können - z.B. durch Anwendung von Glättungsoperationen zur Erhöhung der Dreiecksqualität. Auch kann die Oberfläche mittels Segmentierung aus einem volumetrischen Datensatz gewonnen werden.

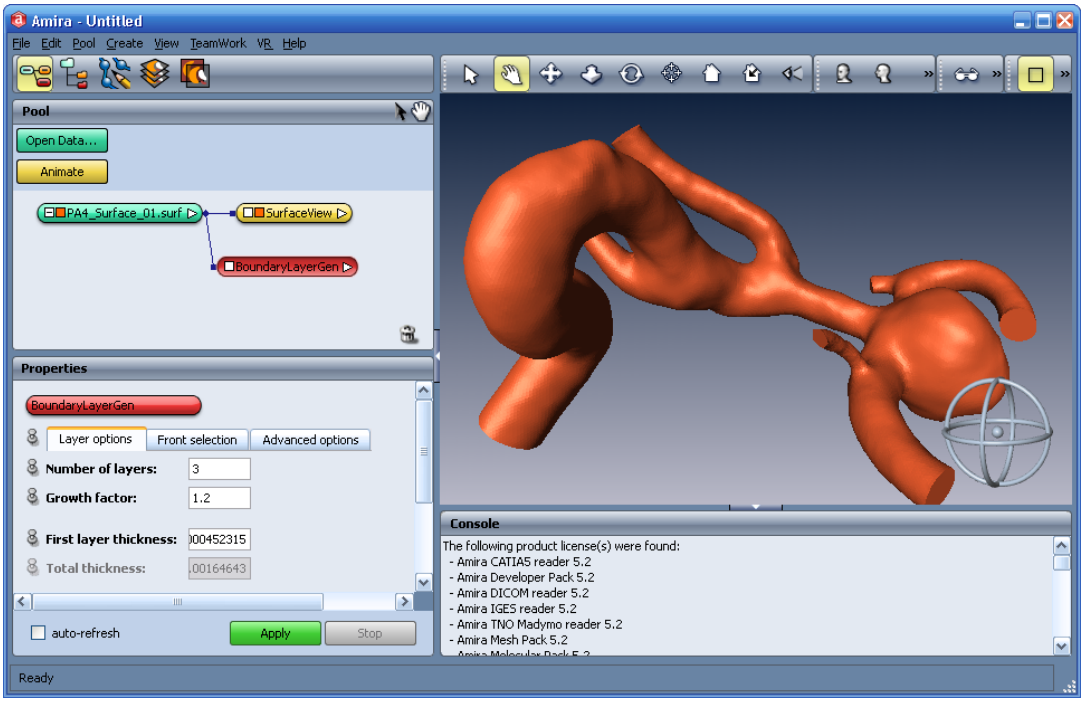



Abbildung 4.31: Das Bild zeigt Amira mit einer geladenen Aneurysmenoberfläche und aktivem Randschichtgenerator. Die GUI von Amira ist in die Bereiche Objektpool (links-oben), Modul-GUI (links-unten), Grafikausgabe (rechts-oben) und Kommandozeile (rechts-unten) unterteilt.

Die Oberfläche des Moduls ist durch Karteireiter in die Bereiche *Layer Options*, *Front Selection* und *Advanced Options* unterteilt. Die Optionen der einzelnen Gruppen werden nachstehend näher beschrieben:

**Layer options:** Der Nutzer übergibt die grundlegenden Parameter an das Modul. Dies sind die Anzahl der Prismenschichten, der Anstiegsfaktor der Prismenhöhe und der Name der CGNS-Datei.

**Front selection:** In diesem Abschnitt werden die Randbereiche, auf denen eine Prismenschicht erstellt werden soll, über die *Material Id* und *Boundary Id* bestimmt. Des Weiteren kann für jede *Shell* die Beschaffenheit des Tetraedergitters durch Angabe der *Meshsize* (Volumenerhöhung mit zunehmendem Abstand vom Rand) und einer oberen Schranke für die Tetraederqualität beeinflußt werden.

**Advanced options:** In den erweiterten Optionen können Feineinstellungen für den Generierungsprozess vorgenommen werden.

**Sharp edge angle:** Der Parameter bestimmt den Winkel, ab welchem eine Kante als scharf angesehen wird. An diesen Kanten werden dann Übergangselemente

erstellt. Der Default-Wert liegt bei 270°. Wählt man einen Wert von 360° werden keine Übergangselemente genutzt, welches jedoch auch zum Abbruch des Generierungsprozesses führen kann.

**Growthline smoothing:** Anzahl der Glättungsiterationen für die Richtungsvektoren

**Maximum growthline face angle:** Der Wert bestimmt den maximal erlaubten Winkel zwischen dem Richtungsvektor und einem seiner zugehörigen Polygonnormalen. Hierdurch kann die Orthogonalität der Randzellen beeinflußt werden.

**Lower/Upper thickness deviation:** Maximale Abweichung der Schichtdicke von dem vorgegebenen Wert

**Height smoothing:** Anzahl der Glättungsiterationen der Prismenhöhenwerte

**Max height difference:** Maximal erlaubter Unterschied in der Höhe zwischen zwei benachbarten Prismenelementen

**Max tetra diam ratio:** Obere Grenze für die Tetraederqualität

**Tetra smoothing:** Anzahl der Glättungsiterationen für den Tetraederkern

**Mesh up/down scale:** Skaliert das Gitter, damit Rundungsfehler bei sehr geringen Geometrieabmessungen minimiert werden können.

## 4.5 Zusammenfassung

Als Hauptprobleme bei der Generierung eines hybriden Gitters lassen sich lokale und globale Überschneidungen, komplexe Geometrien und nichtmannigfaltige Oberflächen nennen. Überschneidungen können durch Auswerten der *Feature Size*, Ermitteln von Überkreuzungen der Richtungsvektoren, Glätten der Richtungsvektoren und Überprüfen auf Überschneidungen der gegenüberliegenden Polygonfronten vermieden werden. Wurden Überschneidungen gefunden, wird die Randschicht in der Höhe verringert. Die Beschränkung des maximalen Höhenunterschieds zwischen benachbarten Randzellen und die Glättung der Randzellen führen zu ebenmäßigen Randschichten. Eine Menge von Übergangselementen und mehrere Richtungsvektoren pro Vertex ermöglicht es, Randschichten auch für sehr komplexe Geometrien zu erstellen. Die Topologie nichtmannigfaltiger Oberflächen wird durch die *Minimal Use* Datenstruktur repräsentiert. Die Lösungskonzepte wurden im Rahmen eines Gittergenerierungsmoduls in Amira implementiert, welches hybride Gitter im CGNS-Format speichern kann.

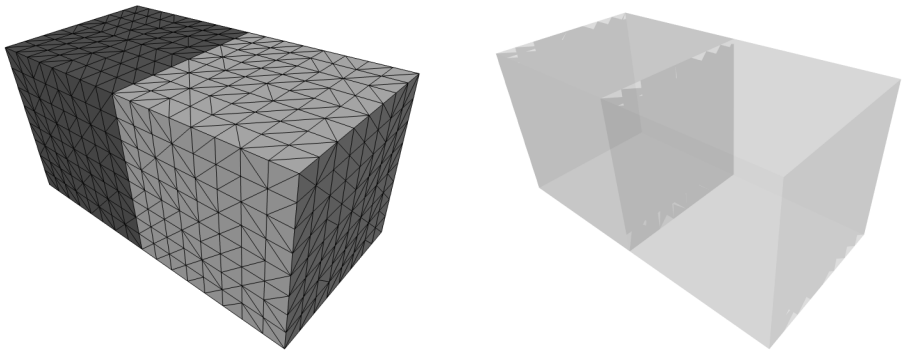
# 5 Ergebnisse

In diesem Kapitel werden hybride Gitter, die durch den in dieser Diplomarbeit implementierten Gittergenerator erzeugt wurden, dargestellt (Abschnitt 5.1) und näher analysiert. In Abschnitt 5.2 werden numerisch gewonnene Strömungsergebnisse mit analytischen verglichen. Die Elementqualität und die Eigenschaft bei einer Strömungssimulation von einem Gitter, das mit einem aktuellen kommerziellen Gittergenerator erstellt wurde, wird in Abschnitt 5.3 überprüft. Der Einfluss von Übergangselementen auf die Konvergenz der Strömungssimulation wird in Abschnitt 5.4 betrachtet. Abschnitt 5.5 zeigt die Ergebnisse einer Strömungssiumlation, die auf einem hybriden Gitter der oberen Atemwege durchgeführt wurde.

## 5.1 Beispielgeometrien

Im Kapitel 4 wurden Probleme aufgezeigt, die im Gittergenerierungsprozess auftreten können. In diesem Abschnitt werden die Fähigkeiten der Gittergenerierungsansätze anhand von Beispielgeometrien gezeigt, wobei jede Geometrie eine bestimmte Problemklasse adressiert. Im Folgenden sind die Geometrien aufgelistet und welche Probleme diese behandeln:

- Zwei Würfel (Abbildung 5.1): Nichtmannigfaltigkeiten
- Metaballs (Abbildung 5.2): Globale Überschneidungen
- Ecke (Abbildung 5.3): Übergangselemente an Kanten und Vertices
- Würfel (Abbildung 5.4): Äußere Ränder



(a) (b)

Abbildung 5.1: Werden zwei Würfel durch jeweils eine Seitenfläche miteinander verbunden, enthält die Geometrie einen inneren Rand und ist somit nichtmannigfaltig. Erzeugt ein Gittergenerator ein hybrides Gitter für diese Geometrie, so muss er Nichtmannigfaltigkeiten verarbeiten können. Soll keine Randschicht auf dem inneren Rand erzeugt werden, muss der Gittergenerator die Seitenelemente der Randschicht korrekt in die Triangulation des inneren Randes integrieren und dabei die Konnektivität der Randelemente des jeweils benachbarten Würfels beachten.

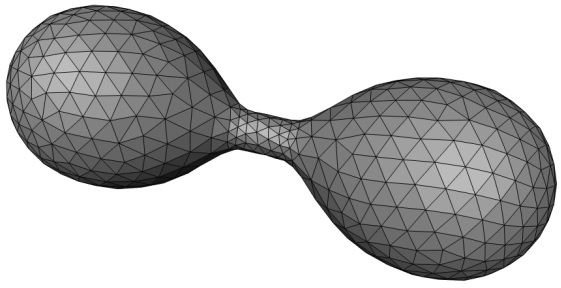
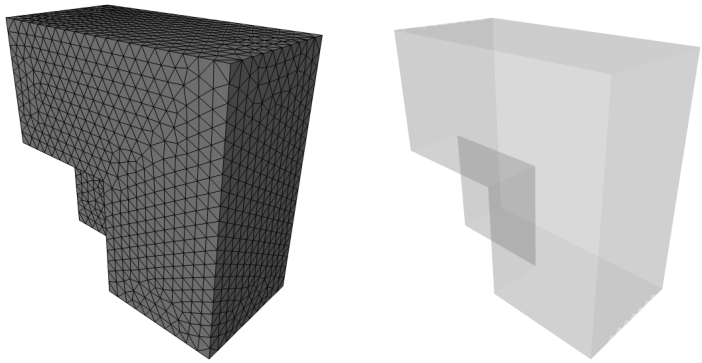


Abbildung 5.2: Globale Überschneidungen können bei Unterschieden im Geometriedurchmesser auftreten. Hier muss der Gittergenerator die Randschicht in der Höhe anpassen, damit es im Bereich der Verschmelzung der beiden Metaballs zu keinen Überschneidungen kommt.



(a) (b)

Abbildung 5.3: Die Geometrie beinhaltet drei Kantenzüge, an denen die inzidenten Dreiecke einen Dihedralwinkel von 270° bilden. Hier kann der Gittergenerator die leicht verzerrten Randelemente glätten oder auch Übergangselemente einfügen. Eine Behandlung des Kantenbereichs ist jedoch nicht notwendig, um ein valides Gitter zu erhalten - die Geometrie soll die Demonstration von Übergangselementen ermöglichen.

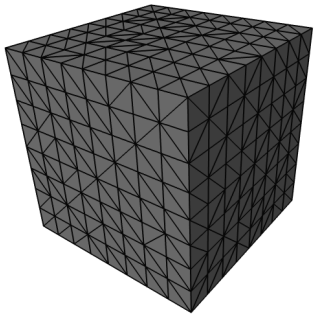
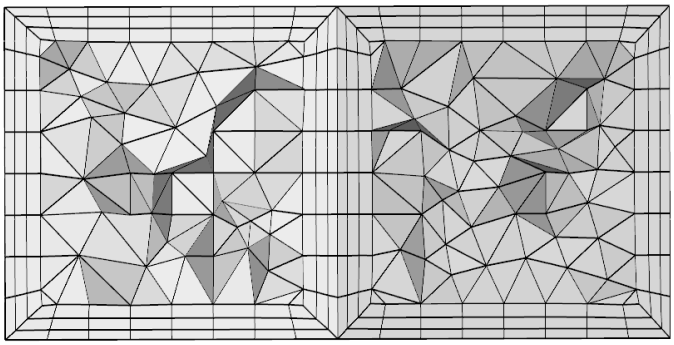


Abbildung 5.4: Der Gittergenerator muss die Seitenflächen der Prismenrandschicht in die Dreiecksrandflächen der Geometrie integrieren, falls auf der Geometrierandfläche keine Randschicht erzeugt werden soll. Dies wird an einem Würfel gezeigt, bei welchem fünf Seitenflächen mit einer Randschicht versehen werden sollen, während die verbleibende Seitenfläche keine Randschicht erhält.

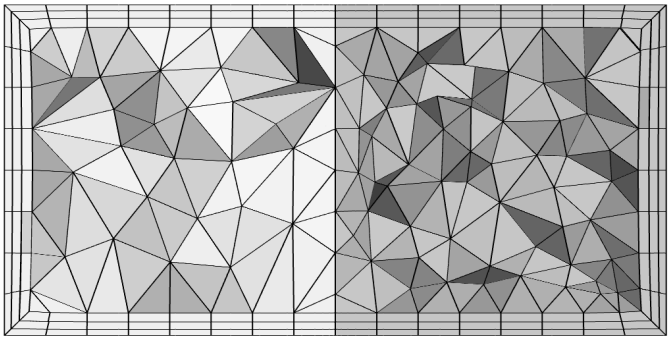
### 5.1.1 Hybride Gitter für die Testgeometrien

Für die Testgeometrien wurden hybride Gitter erstellt:

- Zwei Würfel (Abbildung 5.5)
- Metaballs (Abbildung 5.6)
- Ecke (Abbildung 5.7)
- Würfel (Abbildung 5.8)



(a)



(b)

Abbildung 5.5: (a) Der innere Rand wurde vom Gittergenerator berücksichtigt. Es wurden auf dem inneren Rand für beide Seiten Prismenschichten erzeugt. (b) In diesem Fall sollte keine Prismenschicht auf dem inneren Rand erstellt werden. Die Seitenflächen der Tetraeder- und Prismenelemente wurden korrekt miteinander verbunden.

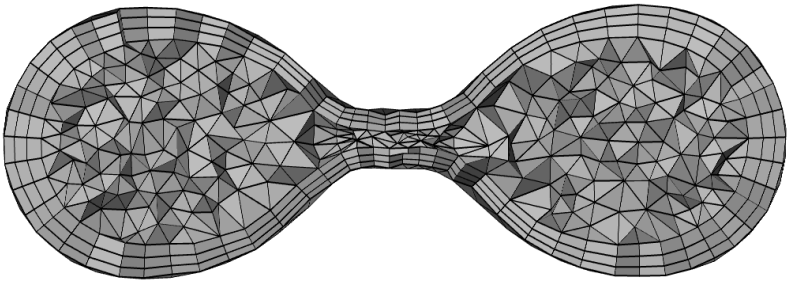


Abbildung 5.6: Im Bereich des geringen Durchmessers wurde die Randschichtdicke verringert, um Überschneidungen zu vermeiden.

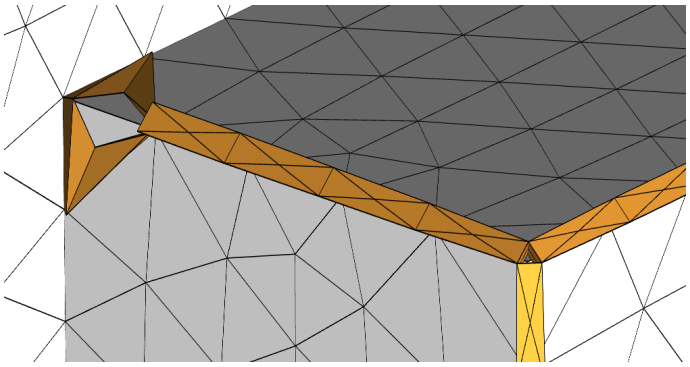
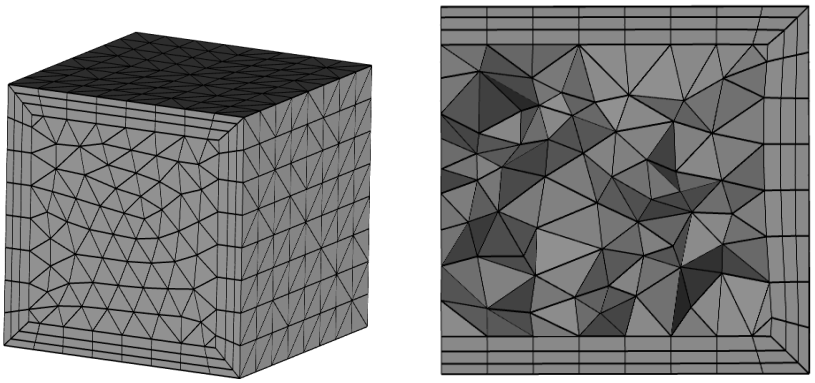


Abbildung 5.7: Die Übergangselemente wurden an den drei scharfen Kantenzügen platziert. Neben der Dreiecksoberfläche werden die Pyramiden und Hexaeder der Übergangselemente dargestellt.

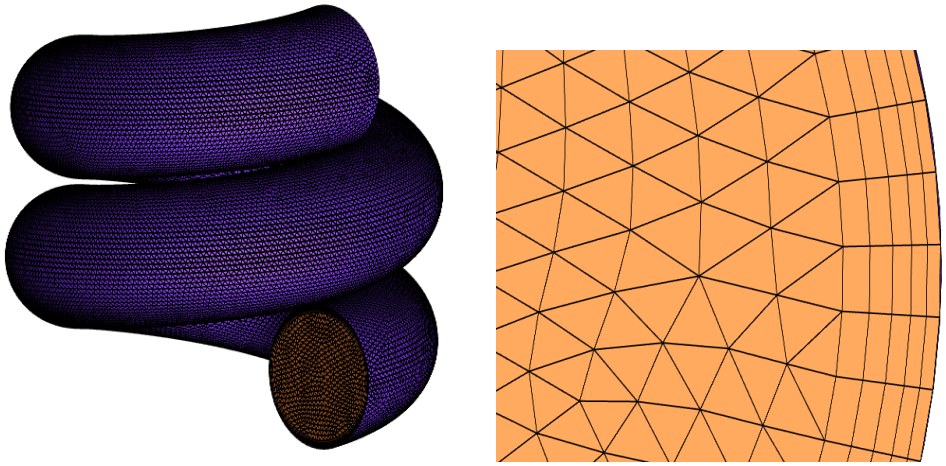


(a) (b)

Abbildung 5.8: (a) Die Viereckssseiten der Prismen wurden korrekt in den äußeren Rand integriert. (b) Schnitt durch das hybride Gitter

## 5.2 Vergleich zwischen analytischem und numerischem Ergebnis

Um zu verifizieren, ob das Ergebnis einer numerischen Strömungssimulation den wahren Strömungsverhältnissen ähnlich ist, kann ein Vergleich der Simulationsergebnisse mit analytisch ermittelten Werten oder mit experimentellen Daten, die auf einem Modell z.B. in einem Windkanal gemessen wurden, durchgeführt werden. Die Simulationseigenschaften der Gitter des implementierten Gittergenerators wurden mit den analytisch gewonnenen Ergebnissen einer Rohrwendel (Abbildung 5.9) verglichen.



(a) (b)

Abbildung 5.9: (a) Es wird die Rohrwendel gezeigt, mit welcher der Vergleich zwischen numerischen und analytisch ermittelten Simulationswerten durchgeführt wurde. Es wurden fünf Prismenrandschichten mit Anstiegsfaktor 1.2 erzeugt. In Bild (b) ist der Ausströmungsbereich im Detail zu sehen, der sich unten an der Rohrwendel befindet.

Die analytischen Ergebnisse stammen aus der Doktorarbeit von Baurmeister und Brauer [1979]. Die Strömung in Rohrwendeln für große Werte des bezogenen Krümmungsradius $R_k^*$

$$R_k^* = \frac{R_k}{R}$$

sind durch die Dean-Zahl $De$ gekennzeichnet.

$$De = Re\sqrt{\frac{R}{R_k}} \quad mit \quad Re = \frac{\overline{w}_x 2R}{\nu}$$

Dabei ist die Reynolds-Zahl $Re$ durch den Rohrinnendurchmesser $d = 2R$, der mittleren Axialgeschwindigkeit in der Rohrwendel $\overline{w}_x$ und der kinematischen Viskosität $\nu$ definiert.

Die Rohrwendel in diesem Versuch besaß einen $R_k^*$ Wert von zwei und einen Durchmesser von zwei Metern, während für die Strömung eine Dean-Zahl von 20 und 80 festgesetzt wurde. Als Fluid wurde, wie auch in allen anderen Strömungssimulationen in den folgenden Abschnitten, ein dem Blut ähnliches verwendet, für welches eine Dichte von 1000 $\frac{Kg}{m^3}$ und

eine dynamische Viskosität von 3,5 $mPa \cdot s$ definiert wurde. Die Strömung war in allen Fällen laminar, stationär, inkompressibel und reibungsbehaftet.

Numerisch berechnet wurden die Geschwindigkeit des Fluids eine halbe Rohrlänge vor dem Ausgang der Rohrwendel - um numerische Effekte bei direkter Messung am Ausgang zu vermeiden - und die Wandschubspannung (WSS) auf der Rohrwendeloberfläche. Die Wandschubspannung wurde auch in den folgenden Versuchen auf Geometrien von Aneurysmen, sackförmigen Ausweitungen an arteriellen Blutgefäßen, berechnet, da sie für die Aneurysmenforschung eine entscheidene Größe darstellt. Durch die Wandschubspannung können Veränderungen am Aneurysma prognostiziert werden, wie die Wahrscheinlichkeit des Wachstums, der Thrombenbildung und des Ruptierens. Alle Strömungssimulationen wurden mit dem Löser Ansys Fluent in der Version 6.3.26 von dem Labor für Biofluidmechanik der Charité Berlin durchgeführt.

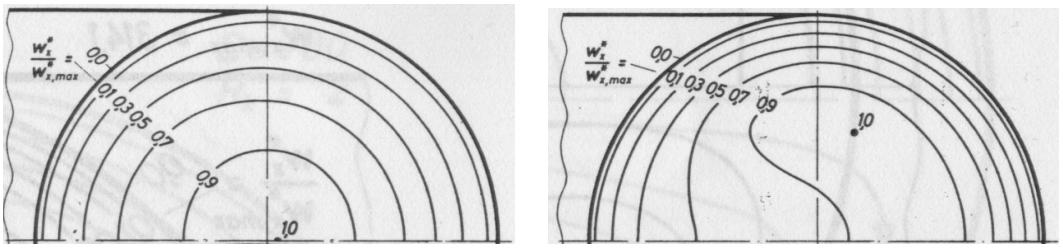
Es wurde ein Tetraedergitter mit Gambit erstellt und ein hybrides Gitter mit dem Gittergenerator dieser Diplomarbeit. Die Gesamtanzahl der volumetrischen Zellen des hybriden Gitters lag bei 812000 und die des reinen Tetraedergitters bei 892000. Auf dem hybriden Gitter mit fünf Prismenrandschichten und einem Anstiegsfaktor von 1,2 wurden Simulationen bei einer Dean-Zahl von 20 und 80 durchgeführt, während auf dem Tetraedergitter eine Simulation mit der Dean-Zahl 80 vorgenommen wurde.

Somit lassen sich das Tetraedergitter und das hybride Gitter bei einer Dean-Zahl von 80 miteinander und mit dem analytischen Ergebnis vergleichen. Die Simulation bei einer Dean-Zahl von 20 auf dem hybriden Gitter kann nur der analytischen Lösung gegenübergestellt werden.

Zu erwarten ist, dass das Tetraedergitter verglichen mit dem hybriden Gitter einen ähnlichen Strömungsverlauf im Querschnitt zeigt, da das Volumen des hybriden Gitters auch überwiegend mit Tetraedern vergittert ist. Da jedoch eine erhöhte Auflösung im Randbereich vorliegt, müssten Werte wie die Wandschubspannung genauer erfasst werden.

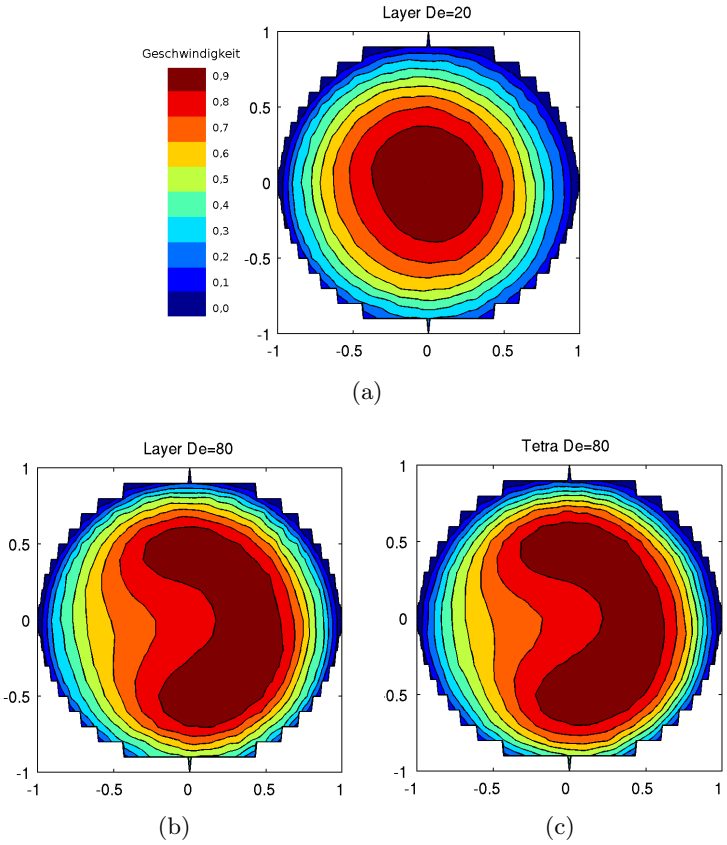
Abbildung 5.10 stellt die analytisch ermittelte Strömungsgeschwindigkeit und Abbildung 5.11 die numerische dar. Die Wandschubspannung wird auf den Bildern unter 5.12 gezeigt und die Anzahl der Iterationen in Abhängigkeit des Restfehlers (engl.: Residual) auf Abbildung 5.13.

Es zeigt sich eine Bestätigung der theoretischen Überlegungen. Die Geschwindigkeitswerte der drei Querschnitte unterscheiden sich kaum von den analytischen Werten und auch die Strömungsgeschwindigkeit des hybriden Gitters verglichen mit dem Tetraedergitter ist fast identisch. Bei der Wandschubspannung ist das hybride Gitter dem Tetraedergitter überlegen: Es treten bei den Wandschubspannungswerten des Tetraeders Unstetigkeiten auf - die Wandschubspannungswerte sollten glatt entlang der Oberfläche verlaufen. Die verrauschten Wandschubspannungswerte lassen auf eine Störung durch die Diskretisierung schließen. Bei der Anzahl der Iterationen zur Konvergenz der Lösung benötigte die Simulation auf dem hybriden Gitter mit 350 Schritten weniger Zeit als auf dem Tetraedergitter, welches 375 Iterationen erforderte.




(a) (b)

Abbildung 5.10: Die Grafiken zeigen die analytisch bestimmte Geschwindigkeit eine halbe Rohrlänge vor dem Ausgang der Wendel. Die Strömungsgeschwindigkeit des unteren Halbkreises ist identisch zur oberen. Der $R_k^*$ von 2 und der Durchmesser von 2 m bleiben für beide Dean-Zahlen identisch. Die Geschwindigkeitswerte $w_x^*$ sind in Bezug auf die maximale Geschwindigkeit $w_{xmax}^*$ normiert. (a) Dean-Zahl = 20; (b) Dean-Zahl = 80




(a) (b) (c)

Abbildung 5.11: Numerisch ermittelte Geschwindigkeiten der Rohrwendel für: (a) ein Prismengitter bei der Deanzahl von 20, (b) für ein Prismengitter bei der Deanzahl von 80 und (c) für ein Tetraedergitter bei der Deanzahl von 80. Auch in diesen Abbildungen sind die Geschwindigkeiten auf die maximale Geschwindigkeit normiert.

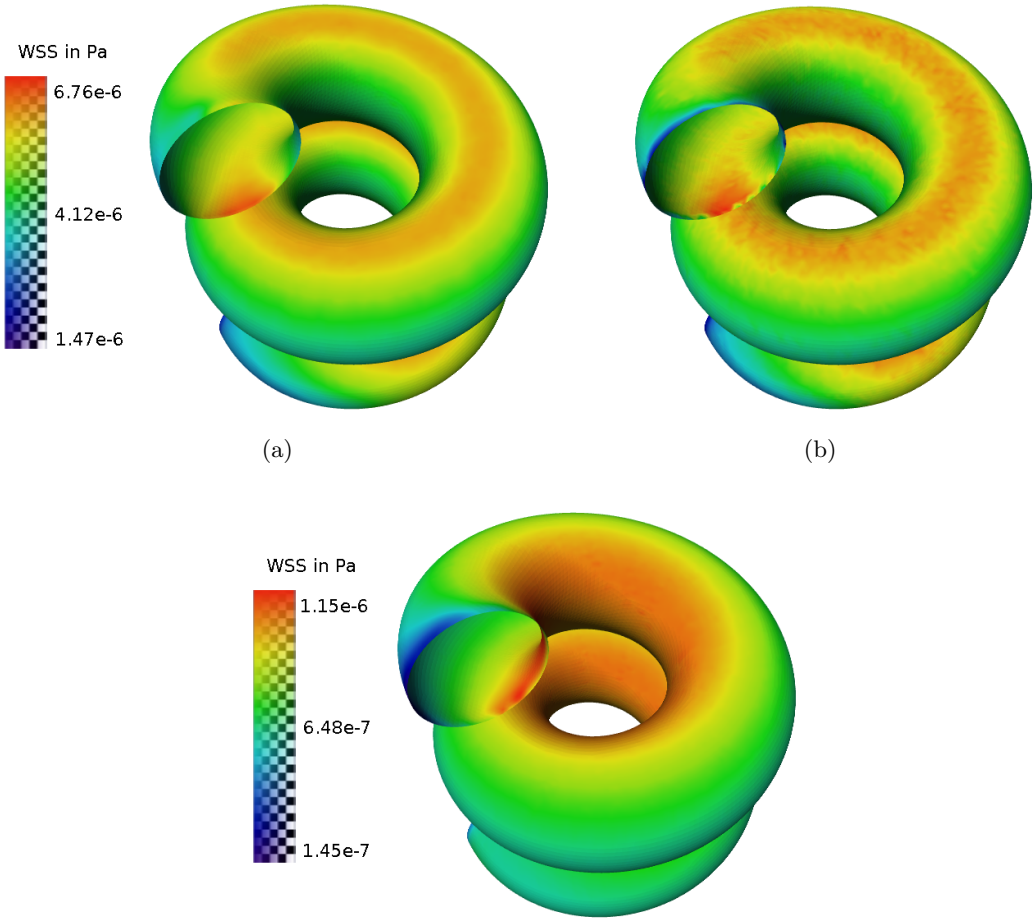



Abbildung 5.12: Die Wandschubspannung wurde bei unterschiedlichen Dean-Zahlen als Falschfarben auf die Rohrwendeloberfläche des jeweiligen Gitters aufgetragen. (a) Dean-Zahl von 80 beim hybriden Gitter; (b) Dean-Zahl von 80 beim Tetraedergitter; (c) Dean-Zahl von 20 beim hybriden Gitter

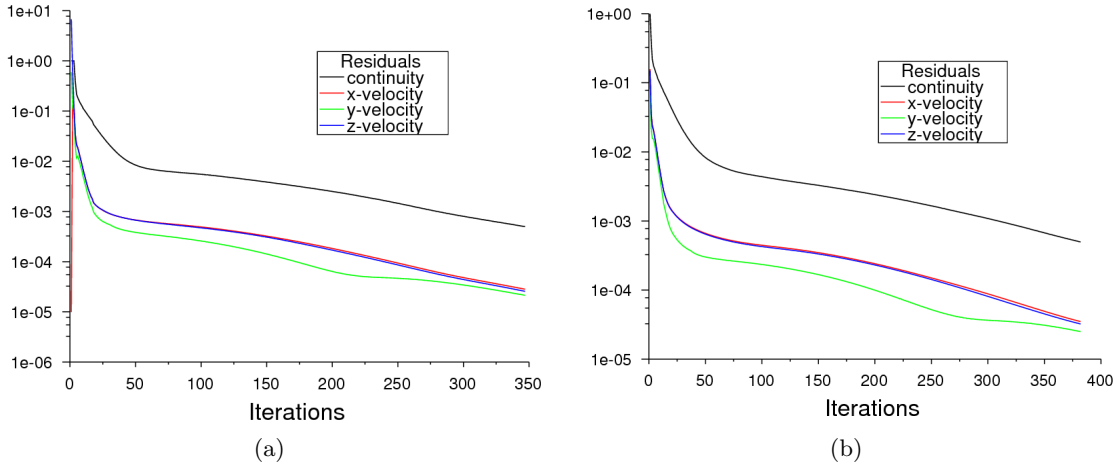



Abbildung 5.13: (a) Residuals für das hybride Gitter (Dean-Zahl 80); (b) Residuals für das Tetraedergitter (Dean-Zahl 80)

## 5.3 Vergleich mit kommerziellen Gittergeneratoren

### 5.3.1 Elementqualität

Für die Aneurysmengeometrie in Abbildung 5.14 wurden drei hybride Gitter mit unterschiedlichen Gittergeneratoren erstellt. Es wurden drei Prismenschichten mit einem Anstiegsfaktor von 1,2 generiert. Zum Einsatz kamen die kommerziellen Programme vom Marktführer Ansys mit den Bezeichnungen Ansys ICEM 11 und Gambit 2.4.6. Ein Gitter wurde mit dem Gittergenerator dieser Diplomarbeit erzeugt. Alle Gittergeneratoren benötigten ungefähr 20 Minuten zur Generierung der hybriden Gitter auf einem Laptop mit Intel Core 2 Duo Prozessor mit 1,83 GHz und 2 GB RAM. Die Elementqualitäten der Gitter sind in Tabelle 5.1 aufgelistet.

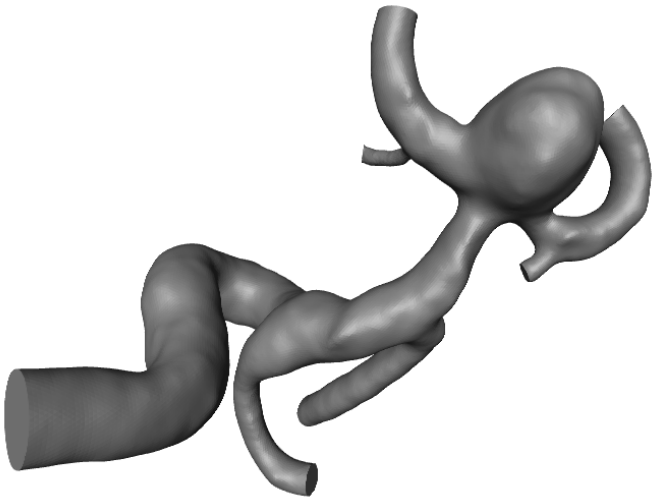


Abbildung 5.14: Geometrie eines Aneurysmas

Die durchschnittliche Elementqualität ist für alle drei Gitter identisch, während die minimale Tetraederqualität für den implementierten Gittergenerator leicht unter den anderen Werten der kommerziellen Gittergeneratoren liegt.

| | Anzahl | minimale Qualität | durchschn. Qualität |
|---|---|---|---|
| implementierter Gittergenerator | | | |
| Prismen | 306150 | 0.2524 | 0.9754 |
| Tetraeder | 891425 | 0.1557 | 0.6634 |
| Vierecke | 990 | 0.6980 | 0.9702 |
| Dreiecke | 104932 | 0.4364 | 0.9048 |
| Ansys ICEM 11 | | | |
| Prismen | 306150 | 0.2519 | 0.9751 |
| Tetraeder | 852260 | 0.2965 | 0.6738 |
| Vierecke | 990 | 0.7431 | 0.9707 |
| Dreiecke | 104806 | 0.3783 | 0.9022 |
| Gambit 2.4.6 | | | |
| Prismen | 306150 | 0.2460 | 0.9724 |
| Tetraeder | 850569 | 0.2158 | 0.6878 |
| Vierecke | 990 | 0.7997 | 0.9649 |
| Dreiecke | 104900 | 0.4362 | 0.9058 |

Prismenqualität = min(Determinante,Windschiefe); Tetraederqualität = Aspektverhältnis;
Viereckqualität = Determinante; Dreieckqualität = Aspektverhältnis

Tabelle 5.1: Aufgelistet ist der Elementqualitätsvergleich zwischen drei hybriden Gittern, welche mit unterschiedlichen Gittergeneratoren erzeugt wurden. Alle Qualitätswerte sind auf einen Zahlenbereich zwischen Null und Eins normiert, wobei ein Wert von Eins das Optimum darstellt.

### 5.3.2 Strömungssimulation

Für die Aneurysmengeometrie in Abbildung 5.15 wurde ein hybrides Gitter mit dem implementierten Gittergenerator und mit Gambit 2.4.6 erzeugt. Auch hier wurden drei Prismenschichten mit einem Anstiegsfaktor von 1,2 verwendet. Auf beiden Gittern wurde der Blutfluss simuliert. Die Elementqualität der beiden Gitter ist in Tabelle 5.2 zu sehen.

| | Anzahl | minimale Qualität | durchschn. Qualität |
|---|---|---|---|
| implementierter Gittergenerator | | | |
| Prismen | 305655 | 0.5308 | 0.9539 |
| Tetraeder | 974827 | 0.1060 | 0.6711 |
| Vierecke | 765 | 0.6953 | 0.9461 |
| Dreiecke | 103714 | 0.4147 | 0.9094 |
| Gambit 2.4.6 | | | |
| Prismen | 305655 | 0.3119 | 0.9529 |
| Tetraeder | 547303 | 0.1943 | 0.6676 |
| Vierecke | 765 | 0.5820 | 0.9221 |
| Dreiecke | 103828 | 0.4147 | 0.9103 |

Prismenqualität = min(Determinante,Windschiefe); Tetraederqualität = Aspektverhältnis; Viereckqualität = Determinante; Dreieckqualität = Aspektverhältnis

Tabelle 5.2: Aufgelistet ist der Elementqualitätsvergleich zwischen dem Gitter, welches mit dem Gittergenerator dieser Diplomarbeit produziert wurde, und dem Gambit-Gitter. Alle Qualitätswerte sind auf einen Zahlenbereich zwischen Null und Eins normiert, wobei ein Wert von Eins das Optimum darstellt.

Das Gitter, welches mit dem Gittergenerator dieser Diplomarbeit erstellt wurde, besitzt bei den meisten Werten eine bessere Qualitätsausprägung als das Gambit-Gitter. Ausnahme ist - wie im Abschnitt zuvor - die minimale Qualität des Tetraedergitters, welche leicht unter dem Wert des Gambit-Gitters liegt. Der Unterschied in der Tetraederelementanzahl liegt daran, dass der Tetraedergenerator von Amira erst ab einer bestimmten Tetraederanzahl die vorgeschriebene Qualität erreichen konnte, während beim Gambit Tetraedergenerator eine gute Elementqualität bei einer geringeren Tetraederzahl möglich war. Die Anzahl der Iterationen (Abbildung 5.16) zeigt, dass das hybride Gitter dieser Diplomarbeit zur Konvergenz 1300 Iterationen benötigte, während das Gambit Gitter 950 Iterationen brauchte. Die Wandschubspannungen in den Bildern unter 5.17 sind ungefähr gleich, wobei die Werte der Wandschubspannung für das Gitter des implementierten Gittergenerators leicht höher sind. Dies ist auch aus der Betrachtung der Wandschubspannung an drei im Gitter liegenden Punkten ersichtlich (Abbildung 5.18).

Als Ergebnis lässt sich sagen, dass im Verhältnis zum Gitter dieser Diplomarbeit das Gambit Gitter nur 73 % der Iterationen zur Konvergenz benötigt und sich die Werte für die Wandschubspannung leicht unterscheiden. Ob beide Beobachtungen durch die unterschiedliche Anzahl der Zellen bedingt sind, ist durch weitere Versuche zu klären.

Welches der Gitter näher an der tatsächliche Strömung liegt, kann durch einen Vergleich mit der Strömung auf einem Modell in Erfahrung gebracht werden. Das Institut für Biofluidmechanik der Charité müsste also für die untersuchte Geometrie ein Modell erstellen. Ein analytischer Vergleich ist nicht möglich, da die Geometrie und die Strömungsphänomene zu komplex sind.

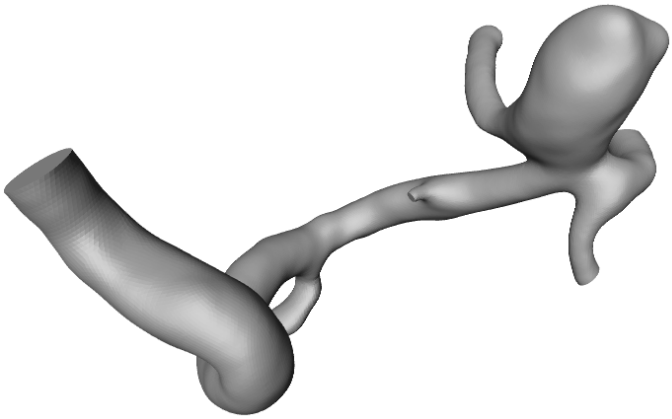
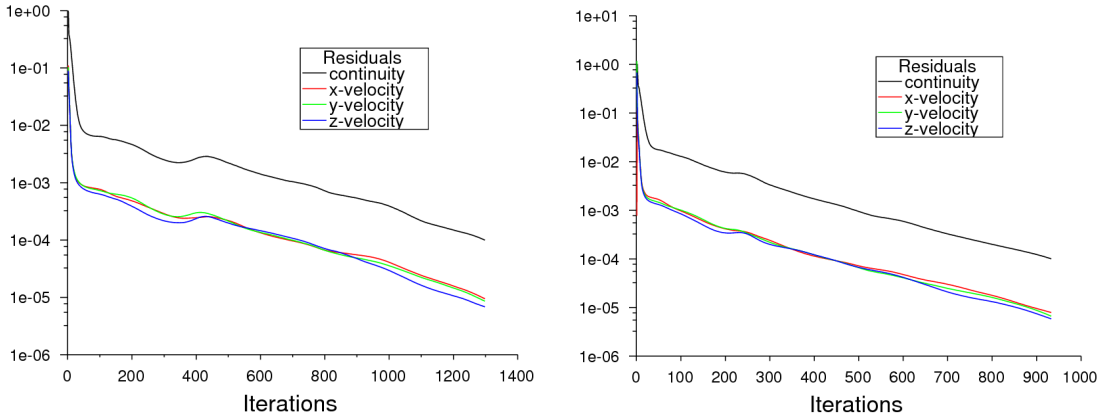
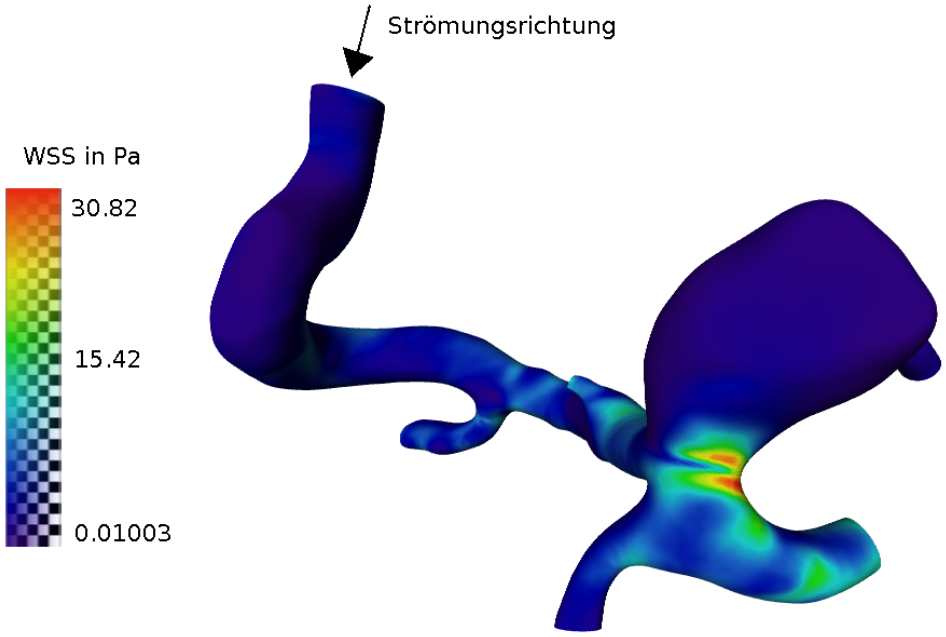


Abbildung 5.15: Geometrie eines Aneurysmas

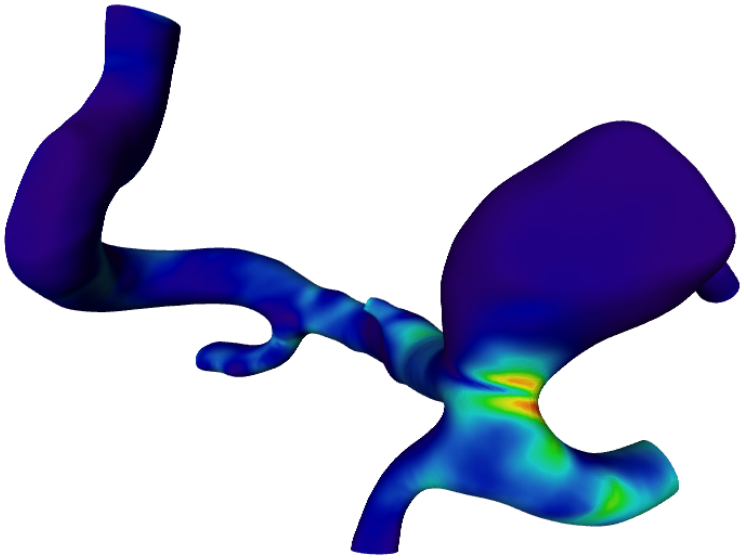



(a) (b)

Abbildung 5.16: (a) Residuals vom hybriden Gitter des implementierten Gittergenerators; (b) Residuals des hybriden Gitters, welches mit Gambit erzeugt wurde




(a)



(b)

Abbildung 5.17: Wandschubspannung als Falschfarben aufgetragen auf der Aneurysmenoberfläche: (a) Wandschubspannung für das hybride Gitter des implementierten Gittergenerators; (b) Wandschubspannung für das hybride Gambit-Gitter

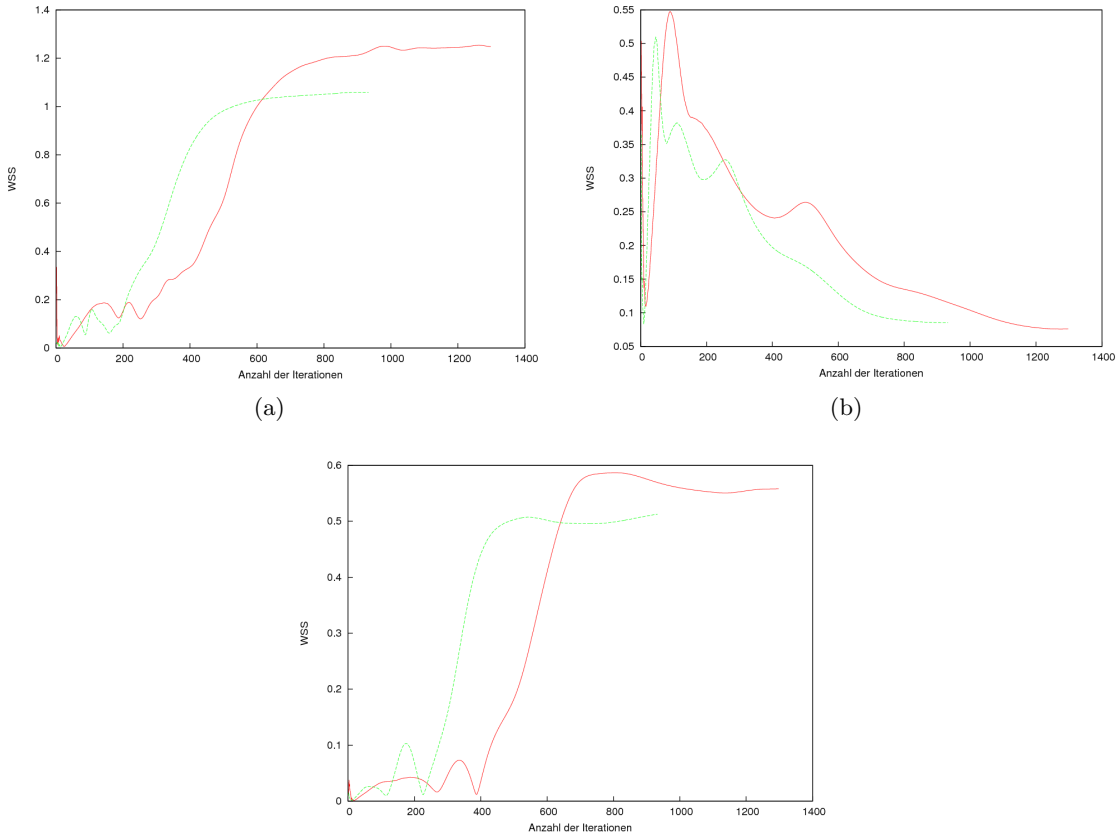
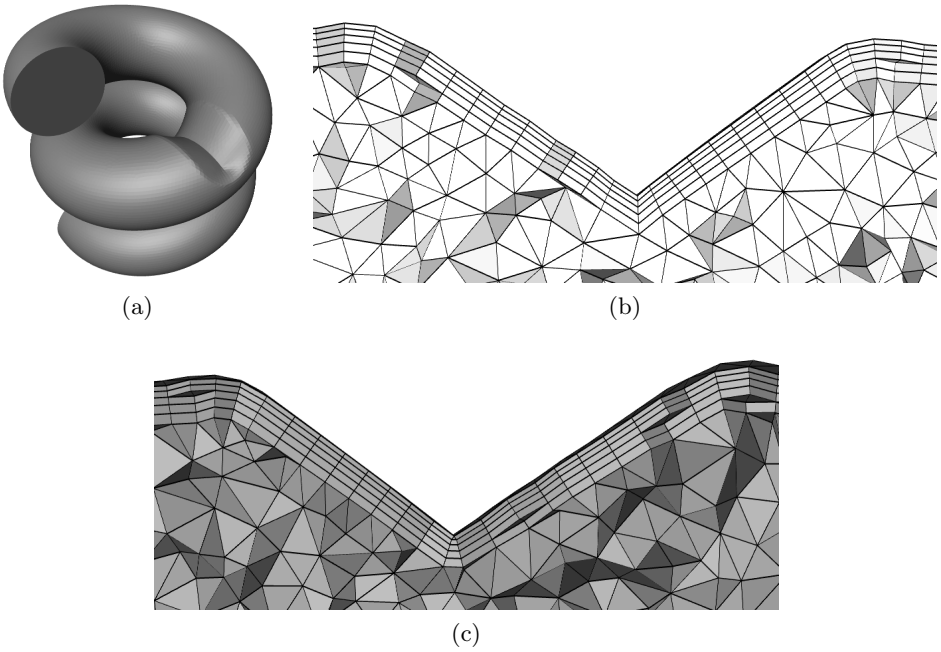



Abbildung 5.18: Die Graphen (a) bis (c) zeigen die errechnete Wandschubspannung (WSS) in Pa für das Gitter des implementierten Gittergenerators (rot) und das Gambit-Gitter (grün) in Abhängigkeit von der Anzahl der Iterationsschritte für drei Punkte im Gitter. Der Wert der Wandschubspannung am Ende aller Iterationsschritte stellt den ermittelten Wert der Wandschubspannung an dem betrachteten Punkt dar. Bei Graph (a) und (c) liegen die Werte des Gitters vom implementierten Gittergenerator über den Wandschubsspannungswerten vom Gambit-Gitter. Bei (b) sind die Werte fast identisch.

## 5.4 Einfluss der Übergangselemente

Um den Einfluss der Übergangselemente auf eine Strömungssimulation zu überprüfen, wurde in die Rohrwendel eine scharfe Kante eingefügt (Abbildung 5.19). Miteinander verglichen werden ein hybrides Gitter mit und eines ohne Übergangselemente. Beide wurden mit dem Gittergenerator dieser Diplomarbeit erzeugt. Beide Lösungen konvergierten nach 500 Iterationen (Abbildung 5.20). Bild 5.21 zeigt ein Schnitt mit einer Ebene, auf der die Strömungsgeschwindigkeit aufgetragen ist, durch den Bereich nahe der scharfen Kante.



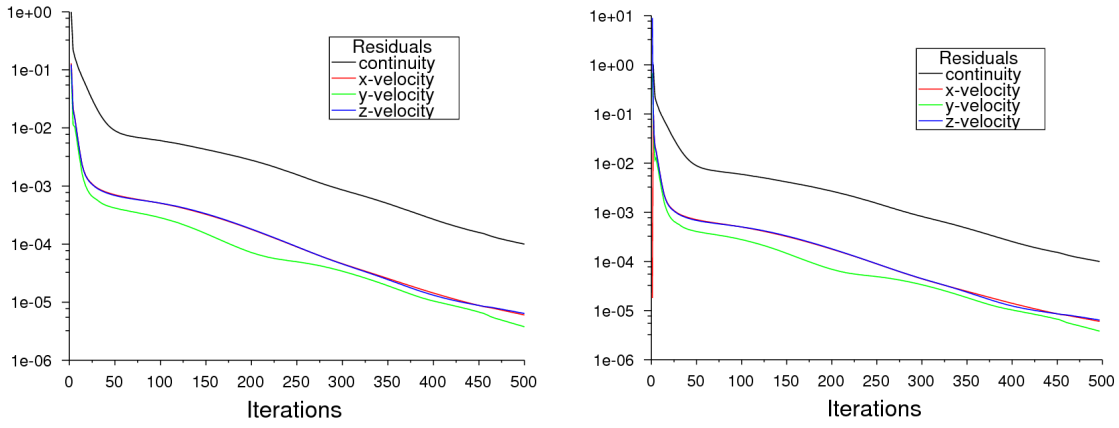
(a) (b)

(c)

Abbildung 5.19: In die Rohrwendelgeometrie (a) wurde eine scharfe Kante eingefügt. Danach wurden jeweils ein hybrides Gitter ohne (b) und mit Übergangselementen (c) erstellt.

Es zeigt sich, dass die Übergangselemente keine Auswirkung auf die Konvergenz der Simulation haben. Das Einfügen von Übergangselementen setzt die minimale Tetraederqualität herunter (Tabelle 5.3). Alle anderen Qualitätswerte der beiden Gitter sind fast identisch. Die verringerte Elementqualität wird durch Tetraeder verursacht, die Teil von Übergangselementen sind. Um die Minderung der Tetraederqualität zu vermeiden, könnte ein neues Übergangselement konzipiert werden, welches die Einführung von beliebigen Diagonalen in einem Prisma bei Wahrung einer guten Elementqualität ermöglicht. Ein anderer Ansatz wäre die Verwendung von allgemeinen Polyedern, welches die Unterteilung des degenerierten Prismas überflüssig machen würde und damit auch die Integration von Diagonalen in benachbarten Elementen.

Die gemessenen Strömungswerte sind qualitativ gleichwertig, jedoch lässt sich ein Unterschied im Geschwindigkeitsfeld ausmachen, der durch eine minimale Korrektur der Schnittebene verschwindet. Ob dies durch die leicht unterschiedlichen Tetraedergitter oder die Übergangselemente verursacht wird, ist durch weitere Versuche zu überprüfen.



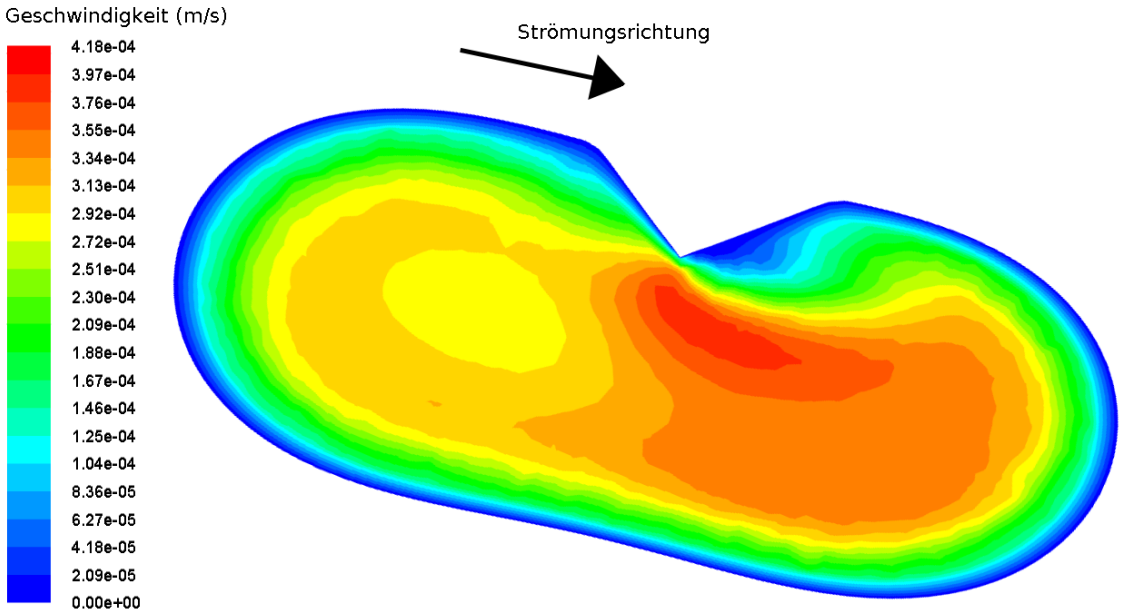

(a) (b)

Abbildung 5.20: (a) Residuals für das hybride Gitter ohne Übergangselemente; (b) Residuals für das hybride Gitter mit Übergangselementen

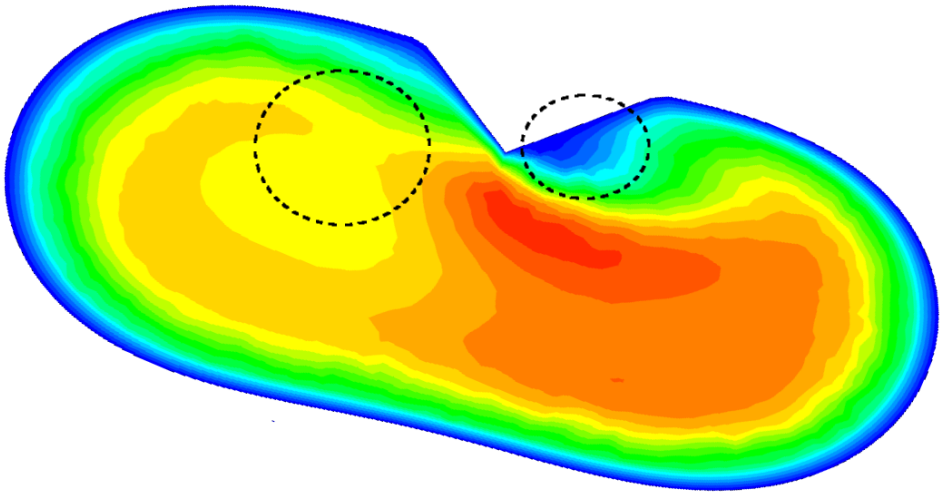
| | Anzahl | minimale Qualität | durchschn. Qualität |
|---|---|---|---|
| keine Übergangselemente | | | |
| Prismen | 301790 | 0.5173 | 0.9670 |
| Tetraeder | 506662 | 0.1380 | 0.6559 |
| Vierecke | 800 | 0.9583 | 0.9744 |
| Dreiecke | 62418 | 0.4010 | 0.8660 |
| mit Übergangselementen | | | |
| Prismen | 301819 | 0.4410 | 0.9674 |
| Tetraeder | 505740 | 0.0004638 | 0.6561 |
| Pyramiden | 133 | 0.7863 | 0.9002 |
| Hexaeder | 75 | 0.5957 | 0.6796 |
| Vierecke | 800 | 0.9583 | 0.9744 |
| Dreiecke | 62418 | 0.4010 | 0.8660 |

Prismenqualität = min(Determinante,Windschiefe); Tetraederqualität = Aspektverhältnis;
Pyramidenqualität = Determinante; Hexaederqualität = Determinante;
Viereckqualität = Determinante; Dreieckqualität = Aspektverhältnis

Tabelle 5.3: Aufgelistet ist der Elementqualitätsvergleich zwischen einem Gitter mit und einem ohne Übergangselementen. Alle Qualitätswerte sind auf einen Zahlenbereich zwischen Null und Eins normiert, wobei ein Wert von Eins das Optimum darstellt.




(a)

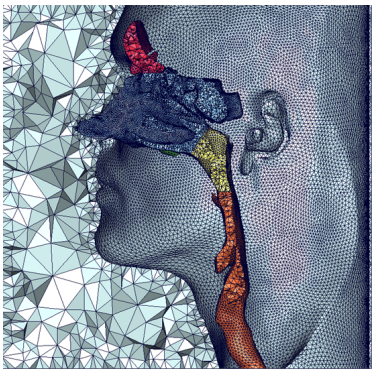


(b)

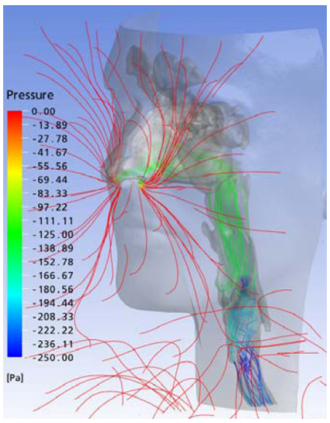
Abbildung 5.21: Es wurde eine Ebene, auf welcher der Geschwindigkeitsverlauf (m/s) aufgetragen ist, an der scharfen Kante durch das hybride Gitter gelegt. Bei Bild (a) handelt es sich um das hybride Gitter mit Übergangselementen und bei Bild (b) um das Gitter ohne Übergangselemente. Unterschiede im Geschwindigkeitsfeld zwischen (a) und (b) wurden in Abbildung (b) durch gestrichelte Kreise markiert.

## 5.5 Obere Atemwege

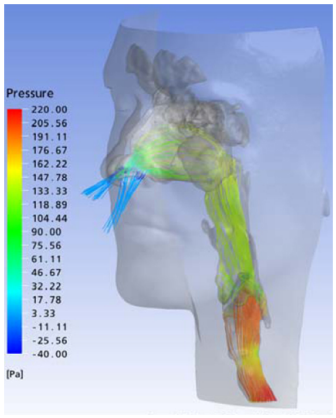
Mit dem Prototyp des implementierten Gittergenerators, der noch keine Übergangselemente besessen hat und am Anfang der Diplomarbeitszeit entstanden ist, wurde für die Geometrie der oberen Atemwege ein hybrides Gitter erstellt und eine Strömungssimulation in Kooperation mit der Ansys Niederlassung in Berlin berechnet. Die Simulationsergebnisse (Abbildung 5.22) werden in der Veröffentlichung von Steinmann u. a. [2008] diskutiert.



(a)




(b)




(c)

Abbildung 5.22: (a) Schnitt durch ein numerisches Volumengitter der oberen Atemwege; (b) Strömung beim Einatmen; (c) Strömung beim Ausatmen

## 5.6 Zusammenfassung

Das aus der Theorie zu erwartende Ergebnis, dass hybride Gitter zu besseren Resultaten als reine Tetraedergitter führen, konnte für die Gitter des implementierten Gittergenerators gezeigt werden. Die Simulationen auf einem Aneurysma verdeutlichten, dass der Gittergenerator ähnliche Qualitäten wie der kommerzielle Gittergenerator Gambit besitzt. Für eine Aneurysmengeometrie wurden drei Gitter mit unterschiedlichen Gittergeneratoren erzeugt. Verwendet wurden der implementierte Gittergenerator und die kommerziellen Generatoren Ansys ICEM und Gambit. Die volumetrischen hybriden Gitter wurden in Bezug auf die Elementqualität verglichen. Das Ergebnis war, dass sich alle Gitter sehr ähnlich sind. Eine Ausnahme bildeten die Tetraederelemente, die bei dem Gittergenerator dieser Diplomarbeit von leicht minderer Qualität waren. Der Einsatz von Übergangselementen hatte in der durchgeführten Strömungssimulation keine negativen Auswirkungen auf die Konvergenz und nur leichte Auswirkungen auf die Simulationsergebnisse. Um die Qualität des Gittergenerators gezielt zu optimieren, sind weitere Versuche durchzuführen.

# 6 Diskussion und Ausblick

Als Basis für den in dieser Diplomarbeit implementierten Gittergenerator diente der von Garimella [1999] in seiner Doktorarbeit entwickelte Ansatz zur Einführung von anisotropen Tetraedern im Randbereich eines reinen Tetraedergitters. In diesem Ansatz wird eine Reihe von Übergangselementen vorgeschlagen, die an scharfen Kanten platziert werden sollen. Diese Übergangselemente wurden jedoch nicht in dem zur Doktorarbeit gehörenden Gittergenerator implementiert. Im Rahmen dieser Diplomarbeit wurde die Idee der Übergangselemente aufgegriffen und bei hybriden Gittern eingesetzt, um auch komplexe Eingabegeometrien vergittern zu können. Der von Garimella [1999] vorgeschlagene Gittergenerierungprozess wurde überarbeitet und erweitert. Eine neue Menge an Übergangselementen wurde eingeführt, es wurden gekrümmte Extrusionsvektoren verwendet und es wurde die Auswertung der medialen Oberfläche vorgenommen, um Überschneidungen im hybriden Gitter zu vermeiden. Der Gittergenerator wurde als Modul in das Visualisierungs- und Analyseprogramm Amira implementiert und die erstellten hybriden Gitter wurden auf ihre Elementqualität und die Güte der Strömungssimulationsergebnisse hin überprüft.

Das aus der Theorie zu erwartende Ergebnis, dass hybride Gitter zu besseren Resultaten als reine Tetraedergitter führen, konnte für die Gitter des implementierten Gittergenerators gezeigt werden. Die Simulationen auf einem Aneurysma verdeutlichten, dass der Gittergenerator ähnliche Qualitäten wie der kommerzielle Gittergenerator Gambit besitzt. Für eine Aneurysmengeometrie wurden drei Gitter mit unterschiedlichen Gittergeneratoren erzeugt. Verwendet wurden der implementierte Gittergenerator und die kommerziellen Generatoren Ansys ICEM und Gambit. Die volumetrischen hybriden Gitter wurden in Bezug auf die Elementqualität verglichen. Das Ergebnis war, dass sich alle Gitter sehr ähnlich sind. Eine Ausnahme bildeten die Tetraederelemente, die bei dem Gittergenerator dieser Diplomarbeit von leicht minderer Qualität waren. Der Einsatz von Übergangselementen hatte in der durchgeführten Strömungssimulation keine negativen Auswirkungen auf die Konvergenz und nur leichte Auswirkungen auf die Simulationsergebnisse. Um die Qualität des Gittergenerators gezielt zu optimieren, sind weitere Versuche durchzuführen.

Es wurden Übergangselemente eingeführt, um sehr komplexe Geometrien mit einem hybriden Gitter zu versehen. Dadurch steigt auch der Automationsgrad der Gittergenerierung: Der Gittergenerierungsprozess ist robuster und der Nutzer kann Zeit bei der Vorbereitung der Geometrie sparen. Ein Nachteil der Übergangselemente ist, dass sie teilweise eine geringere Elementqualität und im Vergleich zu benachbarten Zellen meist ein kleineres Volumen besitzen. Die verminderte Elementqualität wird durch die nötige Unterteilung des degenerierten Prismas verursacht, da das CGNS-Dateiformat keine allgemeinen Polyeder unterstützt. Beschränkte man sich nicht auf Dreiecke, Vierecke, Tetraeder, Pyramiden, Prismen und Hexaeder, dann würde der Einsatz von Übergangselementen zu keiner Herabsetzung der Gitterqualität führen.

Der Zeitaufwand zur Vorbereitung der Geometrie und Generierung des volumetrischen Gitters ist meist minimal im Vergleich zur Dauer der eigentlichen Strömungssimulation.

So stellt sich die Frage, ob man ein Nichtkonvergieren oder Fehler in der Strömung, verursacht durch die geringe Elementqualität der Übergangselemente, riskieren möchte. Man muss abschätzen, wie viel Aufwand erforderlich ist, die problematischen Stellen, an denen Übergangselemente wirklich notwendig sind, manuell zu beseitigen. Die Geometrie der oberen Atemwege besitzt ungefähr zehn dieser Bereiche, so dass ein manuelles Bereinigen durch Vertexverschieben, Kantenkippen und Remeshing kein zu großes Problem darstellt. Ein Bestreben die Geometriebereinigung zu automatisieren ist dem Einsatz von Übergangselementen vorzuziehen.

## 6.1 Weiterentwicklung

Da der Gittergenerator weiterentwickelt werden soll, werden nun Punkte aufgezählt, an denen angesetzt werden kann, um die Leistung des Gittergenerators zu verbessern.

Die Qualität der Prismenrandschichten war vergleichbar mit den erreichten Qualitätswerten der kommerziellen Gittergeneratoren, während die minimale Elementqualität des Tetraedergitters, welches mit dem in Amira integrierten Gittergenerator erstellt wurde, geringer war. Die Tetraeder mit minderer Qualität traten jedoch zum Teil an Stellen auf, wo dies vermeidbar gewesen wäre. Deshalb wäre eine Überarbeitung des Tetraedergenerators bzw. der Tetraederglättungsalgorithmen sinnvoll. Des Weiteren könnte die Steuerbarkeit des Tetraedergenerierungsprozesses erhöht werden, indem z.B. die Feature Size als Eingabe akzeptiert wird, um die Größe der Tetraeder in Abhängigkeit des Geometriedurchmessers zu erzeugen.

Die Wahl der Bereiche, auf denen Prismenelemente erstellt werden können, ist beschränkt, da die Viereckssseiten der Prismen nicht in Dreiecke unterteilt werden. Dies erfolgt nur, wenn ein Prisma zu einem Übergangselement benachbart ist. Die Grundlage, um Prismenrandschichten auf einer Geometrie beliebig beginnen und enden zu lassen, ist jedoch gegeben. So existieren Übergangselemente, mit denen beliebige Seitendiagonalen in Prismen und auch Hexaeder eingeführt werden können.

Das fertige hybride Gitter wird direkt als CGNS-Datei gespeichert und kann nur als reines Tetraedergitter in Amira betrachtet werden. Die Visualisierung von hybriden Gittern in Amira ist seit kurzem möglich, so dass eine Erweiterung des Gittergenerators, um hybride Gitter anzuzeigen, leicht erreichbar ist. Das Anzeigen der Elementqualität nicht nur für Tetraeder würde das Überprüfen der Elementqualität des hybriden Gitters mit externen Tools überflüssig machen.

# Literaturverzeichnis


[Athanasiadis und Deconinck 2003] Athanasiadis, A. N. ; Deconinck, H.: Object-oriented three-dimensional hybrid grid generation. In: *International Journal for Numerical Methods in Engineering* 58 (2003), S. 301–318

[Baker 2005] Baker, T.J.: Mesh generation: Art or science? In: *Progress in Aerospace Sciences* 41 (2005), Nr. 1, S. 29–63

[Bauer 1994] Bauer, R.: *Numerische Berechnung von Kapazitäten in dreidimensionalen Verdrahtungsstrukturen*, Dissertation, 1994

[Baurmeister und Brauer 1979] Baurmeister, U. ; Brauer, H.: Laminare Strömung und Wärmeübergang in Rohrwendeln und Rohrspiralen. In: *VDI-Forschungsh.* 593 (1979), Nr. 1, S. 48

[Beall 1998] Beall, MW: *Framework for the reliable automated solution of problems in mathematical physics over arbitrary domains using scalable parallel adaptive techniques*, Rensselaer Polytechnic Institute, Troy, NY 12180, Dissertation, 1998

[Blacker 1996] Blacker, T.: The Cooper Tool. (1996), S. 10–11

[Böswirth 2007] Böswirth, Leopold: *Technische Strömungslehre.* Vieweg+Teubner, 2007

[Campagna u. a. 1998] Campagna, S. ; Kobbelt, L. ; Seidel, H.P.: Directed edges—A scalable representation for triangle meshes. In: *Journal of Graphics Tools* 3 (1998), Nr. 4, S. 1–11

[Connell und Braaten 1995] Connell, S.D. ; Braaten, M.E.: Semistructured mesh generation for three-dimensional Navier-Stokes calculations. In: *AIAA Journal* 33 (1995), Nr. 6, S. 1017–1024

[De Berg u. a. 2008] De Berg, M. ; Cheong, O. ; Kreveld, M. van: *Computational Geometry: Algorithms and Applications.* Springer, 2008

[Detomi 2004] Detomi, Davide: *Mesh Modifications for Finite Element Methods in Complex Three-Dimensional Domains*, Politecnico di Milano, Dipartimento di Ingegneria Aerospaziale, Italy, Dissertation, 2004

[Dyedov u. a. 2009] Dyedov, V. ; Einstein, D.R. ; Jiao, X. ; Kuprat, A.P. ; Carson, J.P. ; Pin, F. del: Variational generation of prismatic boundary-layer meshes for biomedical computing. In: *International Journal for Numerical Methods in Engineering* (2009)

[Edelsbrunner 2001] Edelsbrunner, Herbert: *Geometry and Topology for Mesh Generation.* Cambridge Univ. Press, 2001

[Erleben und Sporring 2007] Erleben, K. ; Sporring, J.: Line-Stepping for Shell Meshes. In: *Lecture Notes In Computer Science* 4522 (2007), S. 472

[Erleben u. a. 2005] Erleben, Kenny ; Dohlmann, Henrik ; Sporring, Jon: The Adaptive Thin Shell Tetrahedral Mesh. In: *The Journal of WSCG* 13 (2005)

[Frey und George 2000] Frey, P.J. ; George, P.L.: *Mesh Generation: Application to Finite Elements*. Hermes Publishing Company, 2000

[Garimella 1999] Garimella, Rao V.: *Anisotropic tetrahedral mesh generation*, Rensselaer Polytechnic Institute, Troy, New York, Dissertation, 1999

[George und Borouchaki 1998] George, P.L. ; Borouchaki, H.: *Delaunay Triangulation and Meshing: Application to Finite Elements*. Hermes, 1998

[Hassan u. a. 1996] Hassan, O. ; Morgan, K. ; Probert, EJ ; Peraire, J.: Unstructured tetrahedral mesh generation for three-dimensional viscous flows. In: *International journal for numerical methods in engineering* 39 (1996), Nr. 4, S. 549–567

[Hassan u. a. 1995] Hassan, O. ; Probert, EJ ; Morgan, K. ; Peraire, J.: Mesh generation and adaptivity for the solution of compressible viscous high speed flows. In: *International Journal for Numerical Methods in Engineering (ISSN 0029-5981)* 38 (1995), Nr. 7

[Herwig 2006] Herwig, Heinz: *Strömungsmechanik: Eine Einführung in die Physik und die mathematische Modellierung von Strömungen*. Springer Berlin Heidelberg, 2006

[Herwig 2008] Herwig, Heinz: *Strömungsmechanik: Einführung in die Physik von technischen Strömungen*. Vieweg+Teubner, 2008

[Ito u. a. 2007] Ito, Yasushi ; Smith, Alan M. ; Soni, Bharat K. ; Nakahashi, Kazuhiro: Multiple Marching Direction Approach to Generate High-Quality Hybrid Meshes. In: *AIAA Journal* 45 (2007), S. 162–167

[Kallinderis u. a. 1997] Kallinderis, Y. ; Khawaja, A. ; McMorris, H. ; Irmisch, S. ; Walker, D.: Hybrid Prismatic/Tetrahedral Grids for Turbomachinery Applications. (1997), S. 2132

[Kallinderis und Ward 1993] Kallinderis, Y. ; Ward, S.: Prismatic grid generation for three-dimensional complex geometries. In: *AIAA Journal* 31 (1993), S. 1850–1856

[Karamete u. a. 2001] Karamete, B.K. ; Garimella, R.V. ; Shephard, M.S.: Recovery of an arbitrary edge on an existing surface mesh using local mesh modifications. In: *Int. J. Numer. Meth. Engng* 50 (2001), S. 1389–1409

[Khawaja und Kallinderis 2000] Khawaja, A. ; Kallinderis, Y.: Hybrid grid generation for turbomachinery and aerospace applications. In: *Int. J. Numer. Meth. Engng* 49 (2000), S. 145–166

[Kinsey 1993] Kinsey, L. C.: *Topology of Surfaces*. Springer-Verlag, 1993

[Knupp 2007] Knupp, Patrick M.: Remarks on Mesh Quality. In: *45th AIAA Aerospace Sciences Meeting and Exhibit* (2007)

[Lee und Xu 2005] Lee, C. K. ; Xu, Q. X.: A new automatic adaptive 3D solid mesh generation scheme for thin-walled structures. In: *International Journal for Numerical Methods in Engineering* 62 (2005), S. 1519–1558

[Lohner 1993] Lohner, R.: Matching semi-structured and unstructured grids for Navier-Stokes calculations. In: *AIAA Paper* (1993), S. 93–3348

[Lohner 1999] Lohner, R.: Generation of unstructured grids suitable for rans calculations. In: *Aerospace Sciences Meeting and Exhibit, 37th, Reno, NV* (1999)

[Marcum und Weatherill 1995] Marcum, D.L. ; Weatherill, N.P.: Unstructured grid generation using iterative point insertion and local reconnection. In: *AIAA Journal* 33 (1995), Nr. 9, S. 1619–1625

[McMorris und Kallinderis 1997] McMorris, H. ; Kallinderis, Y.: Octree-Advancing Front Method for Generation of Unstructured Surface and Volume Meshes. In: *AIAA Journal* 35 (1997), S. 976–984

[Owen 1998] Owen, S.J.: A survey of unstructured mesh generation technology. In: *7th International Meshing Roundtable* 3 (1998), Nr. 6

[Park und Lee 1998] Park, S. ; Lee, K.: Automatic multiblock decomposition using hypercube++ for grid generation. In: *Computers & fluids* 27 (1998), Nr. 4, S. 509–528

[Pirzadeh 1993a] Pirzadeh, S.: Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method. In: *AIAA Journal* 31 (1993), Nr. 2, S. 257–265

[Pirzadeh 1993b] Pirzadeh, S.: Unstructured Viscous Grid Generation by Advancing-Front Method. (1993)

[Richardson 1922] Richardson, LF: Weather Prediction by Numerical Processes. (1922)

[Schneiders u. a. 1994] Schneiders, R. ; Oberschelp, W. ; Weiler, R. ; Kopp, R. ; Bünten, R. ; Franzke, M.: Automatic generation of hexahedral element meshes for the simulation of metal forming processes. In: *Proc. 4th Int. Conf. Num. Grid Gen. Comp. Fluid Dyn* (1994), S. 223–233

[Sharov u. a. 2003] Sharov, Dimitri ; Luo, Hong ; Baum, Joseph D. ; Löhner, Rainald: Unstructured Navier-Stokes grid generation at corners and ridges. In: *International Journal for Numerical Methods in Fluids* 43 (2003), S. 717–728

[Sharov und Nakahashi 1996] Sharov, Dimitri ; Nakahashi, Kazuhiro: Hybrid prismatic/tetrahedral grid generation for viscous flow applications. In: *Fluid Dynamics Conference, 27th, New Orleans, LA*, June 17-20 1996

[Sheehy u. a. 1995] Sheehy, DJ ; Armstrong, CG ; Robinson, DJ: Computing the medial surface of a solid from a domain Delaunay triangulation. In: *Proceedings of the third ACM symposium on Solid modeling and applications* (1995), S. 201–212

[Shewchuk 2002a] Shewchuk, J.R.: What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In: *11th International Meshing Roundtable* (2002), S. 115–126

[Shewchuk 2002b] Shewchuk, J.R.: *What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures (Preprint).* Dezember 2002

[Shimada 2006] Shimada, Kenji: Current Trends and Issues in Automatic Mesh Generation. In: *Computer-Aided Design & Applications* 3 (2006), Nr. 6, S. 741–750

[Steinmann u. a. 2008] Steinmann, A. ; Bartsch, P. ; Zachow, S. ; Hildebrandt, Th.: Breathing Easily: Simulation of airflow in human noses can become a useful rhinosurgery planning tool. In: *Ansys Advantage* 2 (2008), S. 30–31

[Stimpson u. a. 2007] Stimpson, CJ ; Ernst, CD ; Knupp, P. ; Pebay, PP ; Thompson, D.: The Verdict Geometric Quality Library. (2007)

[Thompson und Soni 1999] Thompson, J.F. ; Soni, B.: *Handbook of Grid Generation.* CRC Press, 1999

[Wang und Murgie 2006] Wang, Y. ; Murgie, S.: Hybrid Mesh Generation for Viscous Flow Simulation. (2006)

[Weiler 1988] Weiler, KJ: The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In: *Geometric Modeling for CAD Applications* (1988), S. 3–36

[Zilske u. a. 2008] Zilske, M. ; Lamecker, H. ; Zachow, S.: Adaptive Remeshing of Non-Manifold Surfaces. In: *Drettakis, G. und R. Scopigno (Herausgeber): Eurographics* 27 (2008)