Adaptive Vereinfachung von Dreiecksnetzen in Echtzeit

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig - Maximilians - Universität München

eingereicht von:

Thomas Odaker

am:

14.06.2016

1. Berichterstatter: Prof. Dr. Dieter Kranzlmüller
2. Berichterstatter: o.Univ.-Prof. Dr. Jens Volkert
3. Berichterstatter: Prof. Dr. Reinhard Klein

Tag der mündlichen Prüfung: 10.08.2016

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

ODAKER, Thomas

Name, Vorname

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2

Kurzfassung

Dreiecksnetze stellen in der Computergraphik eine häufig angewandte Repräsentation von 3-dimensionalen Objekten dar, indem die Objektoberfläche durch Dreiecke angenähert wird. Eine große Menge an Dreiecken erlaubt die Abbildung einer Vielzahl an Details, jedoch mit dem Nachteil eines hohen Berechnungsaufwandes bei der Bilderzeugung.

Verfahren zur Reduktion der Dreieckszahl werden seit langem erforscht. Sie kommen zum Einsatz, um Annäherungen von polygonalen Modellen zu errechnen, die weniger Zeit für die Bilderzeugung in Anspruch nehmen. Unter Zuhilfenahme der Kapazitäten moderner Graphikprozessoren werden Algorithmen entwickelt, die eine Vereinfachung eines Dreiecksnetzes zur Laufzeit, also vor der Bilderzeugung, berechnen.

In dieser Arbeit wird ein neuartiger, paralleler Ansatz zur Vereinfachung von Dreiecksnetzen präsentiert, der die notwendigen Operationen unter Berücksichtigung der Topologie und ohne Vorberechnung von Vereinfachungen ermittelt. Die Vereinfachungsoperatoren werden modifiziert, so dass eine große Menge von Operationen parallel auf einem Dreiecksnetz ausgeführt werden kann, ohne eine Kommunikation zwischen den individuellen Operationen zu erfordern und ohne Löcher und unerwünschte Faltungen auf der Oberfläche zu schaffen.

Der hohe Grad an Parallelität der Operationen erlaubt eine effiziente Implementierung auf moderner Hardware, insbesondere die Ausnutzung moderner Graphikprozessoren, was zu einer starken Reduktion der Berechnungszeit führt. Unter diesen Aspekten ist der Einsatz in Echtzeit möglich und somit eine Vereinfachung, die Position und Blickwinkel des Betrachters in die Berechnungen einfließen lässt, um erkennbare Auswirkungen der Vereinfachung zu reduzieren. Potentielle Erweiterungen sind die Extraktion und Berücksichtigung von markanten Merkmalen eines Objekts und eine verbesserte Oberflächenanalyse bei der Auswahl von Vereinfachungsoperationen.

Abstract

Triangle meshes are a well established representation of 3 dimensional models that approximate the surface of an object with triangles. A large number of triangles creates a detailed object at the cost of high processing time for image creation.

Mesh simplification is a well researched area. It is used to create approximations of polygonal models that require less processing time for image creation than the original. Using modern graphics processors, algorithms are developed to perform the necessary calculations at runtime, computing the simplification before rendering a frame.

This thesis presents a novel, parallel approach to the simplification of triangle meshes. It is designed to be topology-preserving and does not rely on precalculated operations. The simplification operators are adapted to enable the execution of a large number of operations in parallel without having to rely on communication between the individual operations and without causing holes or unwanted foldovers in the simplified mesh.

The parallel approach of the algorithm allows an efficient implementation on modern hardware and especially the use of modern graphics processors. This can be used to reduce the necessary processing time which enables real-time application of the simplification process and the adaptive simplification using the position and viewing vector of the camera to minimize the visible artifacts. Potential future work includes the integration of feature extraction and refinement of the surface analysis to improve the selection of simplification operations.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die mich direkt oder indirekt bei der Entstehung dieser Dissertation unterstützt haben.

Mein besonderer Dank gilt Herrn Prof. Dr. Dieter Kranzlmüller, der während der Betreuung dieser Arbeit mit seinen Anregungen und seiner Expertise wesentlich zur Entstehung meiner Dissertation beigetragen hat. Ich möchte ihm auch dafür danken, dass er mir die Arbeit in Projekten außerhalb meiner Dissertation ermöglicht hat und mir stets mit gutem Rat zur Seite gestanden ist.

Herrn Prof. Dr. Jens Volkert danke ich herzlich für die langjährige Unterstützung und seine Ratschläge sowie seine konstruktive Kritik. Ohne ihn wäre das Zustandekommen dieser Arbeit nicht möglich gewesen.

Vor allem aber danke ich meinen Eltern für die uneingeschränkte Unterstützung während meines Studiums und während der Entstehung dieser Arbeit, ohne die diese Dissertation nicht möglich gewesen wäre.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Darstellung von 3-dimensionalen Objekten durch Methoden der Computergraphik unter Einsatz von moderner Hardware ist weit verbreitet. Die Anwendungsgebiete reichen von der Visualisierung von Objekten über die virtuelle Verdeutlichung der Ergebnisse von Manipulationen an Objekten bis hin zum Einsatz in der Medizin, um die Analyse von Datensätzen aus medizinischen Geräten zu ermöglichen oder zu erleichtern. Auch in der Unterhaltung finden sich derartige Verfahren wieder, etwa in Spielen oder bei der Schaffung digitaler Welten in der Filmindustrie.

Als Ausgangspunkt für die Anwendungsgebiete der Computergraphik dient ein Datensatz. Dieser wird mit Hilfe eines Darstellungsverfahrens in ein Rasterbild umgewandelt, wofür über einen gewissen Zeitraum Rechenleistung notwendig ist. Das Anwendungsgebiet kann für die Anforderungen an die benötigte Rechenzeit für die Darstellung ausschlaggebend sein. Oftmals ist eine wiederholte, rasche Darstellung nötig, etwa um eine Interaktion durch den Benutzer zu ermöglichen.

Die erforderliche Zeit für die Darstellung wird unter anderem durch den Aufwand des Darstellungsverfahrens und die Komplexität des Datensatzes beeinflusst. Bei Echtzeitdarstellung ist die Komplexität des Datensatzes oftmals ein limitierender Faktor. Ein weit verbreiteter Ansatz zur Reduzierung des Darstellungsaufwandes liegt in der Erstellung von Repräsentationen von Objekten mit geringerer Komplexität als das Original. Solche Vereinfachungen führen meist zu einer Reduktion des Darstellungsaufwandes, jedoch häufig auch zu einer Verringerung der Qualität sowie dem Entfallen von Details auf dem dargestellten Objekt.

Obwohl diese Vereinfachung von einem Benutzer durchgeführt werden kann, ist dieser Prozess meist sehr aufwändig und nimmt oft lange Zeit in Anspruch. Vielmals werden automatisierte Verfahren angewandt, deren Ergebnisse zum Einsatz kommen, um den Darstellungsaufwand zu reduzieren. Während alle derartigen Verfahren zum Ziel haben, die Komplexität eines Datensatzes zu verringern und redundante Daten zu eliminieren, können sie sich drastisch in benötigter Berechnungszeit sowie in der Qualität des vereinfachten Resultats unterscheiden.

In dieser Arbeit wird ein Ansatz präsentiert, der eine Vereinfachung eines Dreiecksnetzes berechnet. Ein Hauptziel ist die Durchführung der Vereinfachung in möglichst geringer Zeit, wobei die sichtbaren Auswirkungen der Reduktion der Komplexität des Datensatzes minimiert werden sollen.

1.1 Motivation

In den letzten Jahren ist die Leistung von Graphikprozessoren (engl. „Graphics Processing Unit“, kurz GPU) massiv gestiegen. So stehen heutzutage bis zu mehreren tausend Rechenkerne (z. B. Nvidia Geforce GTX 780: 2304 Kerne [NVG14]) in einer einzelnen GPU zur Verfügung, die nicht mehr auf die Durchführung von graphischen Berechnungen beschränkt sind, sondern frei programmiert werden können [Nvi13]. Abbildung 1.1 zeigt diesen Leistungsanstieg am Beispiel von Nvidia GPUs zwischen 2001 und 2014.

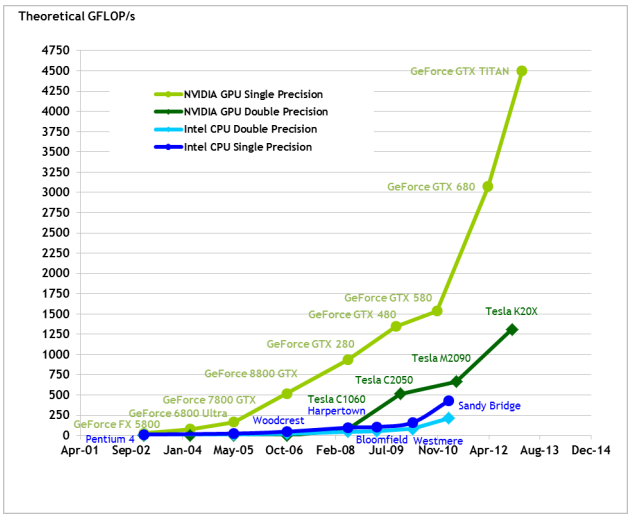

Abbildung 1.1: Leistungsvergleich von Nvidia GPUs [Nvi13]

Trotz dieser verfügbaren Leistung ist die maximale Zahl der darzustellenden Dreiecke immer noch ein limitierender Faktor für die Qualität einer Szene [LE97]. Eine hohe Anzahl an Dreiecken ist für die Wiedergabe von unebenen Oberflächen nötig, so dass gleichmäßige, runde Formen entstehen können und nicht durch kantige Objekte ersetzt werden [BS96]. Eine derart detailreiche Repräsentation wird zwar mit modernen Prozessoren und GPUs ermöglicht, die Verwendung in einer komplexen Szene mit mehreren Instanzen eines Dreiecksnetzes bzw. einer Vielzahl an anderen Objekten erreicht jedoch die Grenze der verfügbaren Leistung [Can11].

Die Entwicklung hardwareunterstützter Verfahren zur dynamischen Anpassung von

Dreiecksnetzen hat einen höheren Detailgrad von Objekten ermöglicht [ESV99b]. Diese beschränken sich allerdings weitgehend auf das Hinzufügen von Dreiecken in Bereichen, die eine präzisere Darstellung benötigen. In anderen Bereichen einer Szene kann die Darstellung von Objekten berechnet werden, die mehr Dreiecke als erforderlich enthalten und somit Rechenzeit in Anspruch nehmen, die für zusätzliche Operationen bei anderen Berechnungen des Darstellungsvorganges zum Einsatz kommen könnte [DX113] [Bou10].

Verfahren für die Vereinfachung von Dreiecksnetzen werden seit langer Zeit erforscht, um die Anzahl von darzustellenden Dreiecken zu reduzieren. Schon 1976 wurde dieses Problem erstmals in [Cla76] beschrieben. Heutzutage findet sich in der Literatur eine Reihe von Ansätzen wie etwa in [Hop97], [SZL92] oder [DT07]. Die vorhandenen Algorithmen lassen sich in 2 Kategorien einteilen [LE97]:

1. Verfahren, die eine vereinfachte Version eines Dreiecksnetzes in einem Vorberechnungsschritt erzeugen [Lin00].

2. Verfahren, die vor jedem Darstellungsvorgang erneut angewandt werden.

Viele Algorithmen sind in der ersten Kategorie zu finden (z. B. in [SZL92], [GH97] oder [LT00]). Auf einem Dreiecksnetz werden so lange Vereinfachungsoperationen durchgeführt, bis eine gewünschte Abbruchbedingung erfüllt ist. Dies resultiert in einer generischen, vereinfachten Version des originalen Objekts. Diese Art von Verfahren wurde entwickelt, um die Vereinfachung in einem Vorberechnungsschritt vor Beginn des Darstellungsvorganges zu berechnen. Die Laufzeit für die Vereinfachung ist somit nicht durch eine mögliche Zeitgrenze für die Darstellung eingeschränkt, und für die Berechnung des neuen Datensatzes können aufwändige Verfahren zum Einsatz kommen, um eine möglichst exakte Annäherung an das originale Netz zu erreichen [LE97]. Verfahren der ersten Kategorie können eingesetzt werden, um mehrere Detailstufen für ein Objekt zu erzeugen. Zur Laufzeit wird eine solche Vereinfachung ausgewählt und anstelle des ursprünglichen Objekts dargestellt. Diese Technik hat allerdings den Nachteil, dass sprunghafte Übergänge bei Wechseln zwischen den einzelnen Auflösungsstufen entstehen können. Des Weiteren kann es zu sichtbaren Einschränkungen bei der Qualität des Objekts kommen [LE97].

Verfahren der zweiten Kategorie werden zur Laufzeit durchgeführt, um unmittelbar vor Beginn eines Darstellungsvorganges die Anzahl an Dreiecken zu verringern. Die Objekte werden manipuliert, so dass ein verminderter Berechnungsaufwand erreicht wird, wobei die Reduktion der Anzahl der Dreiecke des Objekts aus der Position des Betrachters weniger stark bemerkbar sein soll [XV96]. Eine Vereinfachung eines Objekts zur Laufzeit bietet somit mehrere Vorteile [LE97]:

- Adaptive Anpassung erlaubt eine Minimierung sichtbarer Artefakte

- Skalierbare Dreiecksauflösung eines Objekts

- Berücksichtigung von Position und Blickrichtung des Betrachters

Verfahren der zweiten Kategorie haben den Nachteil, dass die für die Vereinfachung benötigte Laufzeit in Grenzen gehalten werden muss, da sie in den Darstellungsaufwand miteinbezogen ist. Dies führt zu Einschränkungen bezüglich der Wahl der durchzuführenden Vereinfachungen bzw. der resultierenden Qualität [LE97] [Hop97] [DT07]. Ein Beispiel für ein Verfahren mit Einschränkungen bei der Freiheit der Wahl der Vereinfachungen ist bei Hoppe [Hop97] zu finden: In einem Vorberechnungsschritt wird eine Hierarchie an möglichen Vereinfachungen definiert und zur Laufzeit daraus eine Menge von Operationen festgelegt und angewandt. Alle möglichen Vereinfachungen sind vorbestimmt, und nur die Selektion der durchgeführten Manipulationen erfolgt zur Laufzeit. Dieser Ansatz hat zwei Vorteile:

- **Fließende Übergänge** Die gewünschte Abbruchbedingung bzw. Anzahl an Dreiecken kann variabel adaptiert werden. Es existieren keine vorbestimmten Auflösungsstufen, und sprunghafte Übergänge werden vermieden.

- **Adaptive Anpassung** Vereinfachungen können so ausgewählt werden, dass sie für die Sicht aus der Position des Betrachters gut geeignet sind und manche Teilbereiche eines Objekts mit mehr Details dargestellt werden.

Der Ansatz in [Hop97] basiert jedoch auf einer Liste vorberechneter Vereinfachungsoperationen. Im Gegensatz dazu beschreiben DeCoro und Tatarchuk [DT07] ein hardwareunterstütztes Verfahren, das beliebige Objekte ohne Vorberechnung vereinfachen kann. Um eine schnelle Berechnungszeit zu erreichen, werden hier Kompromisse bei der Qualität des vereinfachten Dreiecksnetzes eingegangen.

Während mehrere Algorithmen eine Vereinfachung in Echtzeit berechnen können und durch parallele Berechnungen einen Leistungsgewinn auf moderner, paralleler Hardware erzielen, unterliegen sie verschiedenen Beschränkungen:

- **Freiheit der Vereinfachungsoperationen** Oftmals kommt eine vorberechnete Menge an Vereinfachungsoperationen zum Einsatz. Durch Abhängigkeiten zwischen Operationen (z. B. [Hop97]) kann eine optimale Vereinfachung verhindert werden.

- **Qualität der Vereinfachung** Manche Algorithmen (z. B. [DT07]) priorisieren eine geringe Laufzeit über die Qualität der Vereinfachung. Während dadurch eine Ausführung in Echzeit möglich ist, sinkt die Qualität des vereinfachten Objekts.

Im Rahmen dieser Arbeit soll ein Verfahren beschrieben werden, das diesen Beschränkungen nicht unterliegt.

1.2 Zielsetzung

In diesem Abschnitt werden Anforderung und Beitrag dieser Arbeit beschrieben:

- **Fragestellung** Erläuterung der wissenschaftlichen Fragestellung dieser Arbeit.
- **Wissenschaftlicher Beitrag** Präsentation des wissenschaftlichen Beitrags dieser Arbeit.

1.2.1 Fragestellung

Ein Graphikprozessor ist eine Vereinigung von bis zu mehreren tausend individuellen Rechenkernen. Eine Vereinfachung ist das Resultat einer Menge von Vereinfachungsoperationen, die auf einem Dreiecksnetz ausgeführt werden. Abhängigkeiten zwischen diesen Operationen sind oftmals ein beschränkender Faktor für die Parallelität eines Vereinfachungsalgorithmus.

Mehrere Ansätze erzielen eine Beschleunigung von Vereinfachungen durch die parallele Ausführung mehrerer Vereinfachungsoperationen (z. B. [DT07], [HSH09], [PP15], [SN13]). Bei vielen Ansätzen wurde die gleichzeitige Ausführung von zwei Vereinfachungsoperationen auf benachbarten Elementen eines Dreiecksnetzes als einschränkender Faktor identifiziert, die unerwünschte Auswirkungen (z. B. Faltungen der Oberfläche) bewirken kann. Mehrere Ansätze wurden präsentiert, um diese Auswirkungen zu vermeiden bzw. zu korrigieren. [PP15] zerlegt ein Dreiecksnetz in unabhängige Bereiche, um die Interaktion zweier Vereinfachungsoperationen zu verhindern. [HSH09] führt Vereinfachungsoperationen aus und korrigiert anschließend schrittweise die unerwünschten Auswirkungen.

Diese Arbeit beschäftigt sich mit der Fragestellung, wie ein Vereinfachungsoperator definiert werden kann, um eine Vielzahl von Operationen gleichzeitig auf einem Dreiecksnetz auszuführen. Durch die gleichzeitige Anwendung auf benachbarten Elementen eines Dreiecksnetzes dürfen keine unerwünschten Auswirkungen entstehen. Des Weiteren beschäftigt sich diese Arbeit mit dem Problem, wie mit einem derartigen Ansatz ein Verfahren entstehen kann, das unter der Ausnutzung einer modernen GPU bis zu mehreren tausend Vereinfachungsoperationen parallel ausführt und durch den Leistungsgewinn einen Einsatz in Echtzeit praktikabel macht.

Die Definition von Vereinfachungsoperationen als unabhängige Operationen auf

einem Dreiecksnetz bietet in diesem Fall großes Potential für eine Reduktion der Berechnungszeit für die Vereinfachung. Die Ausführung von beliebigen Kombinationen von Operationen wird ermöglicht. Dies erleichtert deren Auswahl und kann die Qualität des vereinfachten Netzes verbessern, da keine Abhängigkeiten beachtet werden müssen.

Im Anschluss wird der Beitrag zu dieser Problemstellung dargelegt.

1.2.2 Wissenschaftlicher Beitrag

Im Zuge dieser Arbeit wurde ein neuartiger Ansatz - die Vertexgruppeneliminierung (Kapitel 3) - entwickelt, der unter Berücksichtigung der Topologie eine Menge von Vereinfachungsoperationen gleichzeitig auf einem Dreiecksnetz ausführt. Dabei wird keine Vorberechnung für die Operationen durchgeführt. Stattdessen soll ein Dreiecksnetz unter Berücksichtigung der Position des Betrachters analysiert und aufgrund der Resultate eine Menge an Vereinfachungsoperationen vorgenommen werden.

Ein wichtiges Ziel der Vertexgruppeneliminierung ist, die große Anzahl an Rechenkernen moderner GPUs auszunutzen, um eine hohe Leistung bei der Vereinfachung zu ermöglichen. Ein Vereinfachungsoperator wird definiert, der auf Vertizes eines Dreiecksnetzes ausgeführt werden kann, um diese zu entfernen (Unterabschnitt 3.4.1). Es wird darauf geachtet, keine Löcher in der Oberfläche zu schaffen und unerwünschte Faltungen zu vermeiden, ohne dass Kommunikation zwischen einzelnen Operationen notwendig ist. Durch das Fehlen von Abhängigkeiten zwischen Operationen kann deren Auswahl zur Laufzeit erfolgen, und die Implementierung auf einer GPU erzielt einen großen Leistungszuwachs.

Neben dem Vereinfachungsoperator wird ein Ansatz beschrieben, um Vertizes eines Dreiecksnetzes für ein Entfernen zu selektieren. Auch hier stehen eine eingeschränkte Datenabhängigkeit und eine Berechnung ohne erforderliche Kommunikation im Vordergrund, um eine effiziente parallele Implementierung zu ermöglichen.

Mit Hilfe der Vertexgruppeneliminierung können in kurzer Zeit große, komplexe Datensätze vereinfacht und die Grundlage für eine rasche Darstellung geschaffen werden (Kapitel 5). Durch den Einfluss der Position des Betrachters in die Berechnungen werden sichtbare Auswirkungen der Vereinfachung minimiert. Die Kombination dieser Faktoren kann eingesetzt werden, um eine Darstellung von hochkomplexen Datensätzen zu ermöglichen oder den Darstellungsvorgang von Objekten zu beschleunigen.

Die Vertexgruppeneliminierung wurde für folgende Ziele entwickelt:

- **Generische Eingabedaten** Ein beliebiges, darzustellendes Dreiecksnetz kann als Eingabe für das Verfahren dienen.

- **Echtzeitverarbeitung** Die Vereinfachung eines Dreiecksnetzes wird in Echtzeit ermittelt. Hierfür kommen keine vorberechneten Operationen zum Einsatz, sondern eine dynamische Bearbeitung unter Einbeziehung der Position des Betrachters.

- **Leistung** Durch die Berücksichtigung der Anzahl der Rechenkerne von moderner Hardware und Ausnutzung der Parallelität soll ein Verfahren mit hoher Leistung entstehen.

- **Portabilität** Das Verfahren erfordert keine speziellen Bedingungen oder Umgebungen und soll einfach in bestehende Systeme einzubinden sein.

- **Optische Qualität** Das vereinfachte Objekt soll den ursprünglichen Datensatz möglichst genau wiedergeben und die erkennbaren Unterschiede zwischen den Darstellungen des vereinfachten und ursprünglichen Objekts minimieren.

1.3 Publikationen

Die nachfolgende Liste gibt einen Überblick über alle Publikationen, die im Bereich dieser Arbeit veröffentlicht wurden. Zuerst werden die Publikationen aufgelistet, bei denen ich Hauptautor bin, danach jene, zu denen ich einen Beitrag geleistet habe:

T. Odaker, D. Kranzlmüller, J. Volkert, *View-dependent Triangle Mesh Simplification using GPU-accelerated Vertex Removal*, WSCG 2016, in: WSCG 2016 Short Paper Proceedings, Pilsen, Tschechische Republik: Vaclav Skala - Union Agency, pp. 51 - 58, 2016

Diese Veröffentlichung beschäftigt sich mit Verbesserungen der Vertexgruppeneliminierung. Es wurden Modifikationen an den Schritten der Klassifizierung und der Verschiebung vorgenommen. Die präsentierten Änderungen dienen dazu, die Parallelität des Verfahrens zu erhöhen und somit die Laufzeit für die Vereinfachung zu reduzieren. Überdies können sie zu einer Verbesserung der Qualität des vereinfachten Dreiecksnetzes führen. Die in dieser Veröffentlichung vorgestellten Entwicklungen wurden erst nach Fertigstellung dieser Dissertation abgeschlossen und sind demnach nicht in dieser Arbeit enthalten.

T. Odaker, D. Kranzlmüller, J. Volkert, *GPU-Accelerated Triangle Mesh Simplification Using Parallel Vertex Removal*, ICCGIP 2016, in: International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol. 10 (1), Zürich, Schweiz: World Academy of Science, Engineering and Technology, pp. 160 - 166, 2016

Die Veröffentlichung setzt sich mit der Relevanz dieser Dissertation abseits des Echtzeiteinsatzes auseinander. Dafür werden Modifikationen an den Verschiebungs-

tests vorgenommen. Damit kann die Vertexgruppeneliminierung eingesetzt werden, um generische (nicht blickwinkelabhängige) Vereinfachungen von hoher Qualität zu berechnen. Da bei diesem Vorgehen der Fokus vor allem auf der Qualität des vereinfachten Dreiecksnetzes liegt, werden längere Laufzeiten in Kauf genommen und das präsentierte Verfahren ist nicht für den Echtzeiteinsatz geeignet. Die vorgestellten Ergebnisse zeigen, dass die Vertexgruppeneliminierung damit auch angewandt werden kann, um Vorberechnungen von Vereinfachungen zu beschleunigen.

T. Odaker, D. Kranzlmüller, J. Volkert, *GPU-accelerated Real-time Mesh Simplification Using Parallel Half Edge Collapses*, MEMICS 2015, in: Mathematical and Engineering Methods in Computer Science: 10th International Doctoral Workshop, MEMICS 2015, Telč, Tschechische Republik: Springer International Publishing, pp. 107 - 118, 2016

Der Schwerpunkt dieser Veröffentlichung liegt auf der Umsetzung (Kapitel 4) und den Ergebnissen (Kapitel 5) der Vertexgruppeneliminierung. Es wird ein Überblick über den Ablauf des Verfahrens geboten. Im Anschluss werden Details der Umsetzung diskutiert. Die Ergebnisse der Leistungsmessungen werden in dieser Veröffentlichung im Detail präsentiert und daraus Erkenntnisse über das Laufzeitverhalten gezogen sowie Vor- und Nachteile des Verfahrens dargelegt.

T. Odaker, D. Kranzlmüller, J. Volkert, *View-dependent Simplification using Parallel Half Edge Collapses*, WSCG 2015, in: Proceedings of WSCG 2015, Pilsen, Tschechische Republik: Vaclav Skala - Union Agency, pp. 63 - 72, 2015

In dieser Veröffentlichung wird der Ansatz der Vertexgruppeneliminierung präsentiert. Es werden die Teilschritte des Verfahrens erläutert und deren Zusammenhang beschrieben. Hier werden auch die für diese Arbeit entwickelten Verschiebungstests vorgestellt. Neben dem Ablauf und der detaillierten Beschreibung der Teilschritte werden erste, einfache Ergebnisse präsentiert, die das Potential der Vertexgruppeneliminierung aufzeigen. Diese Ergebnisse wurden im Zuge dieser Arbeit um mehrere, zusätzliche Testfälle und detaillierte Messungen erweitert und analysiert, um das Laufzeitverhalten sowie Vor- und Nachteile der Vertexgruppeneliminierung aufzuzeigen (Kapitel 5).

Nachfolgende Veröffentlichungen wurden vor dem Entstehen dieser Dissertation präsentiert. Sie beschäftigen sich mit demselben Fachbereich, sind jedoch nicht in dieser Arbeit enthalten:

T. Odaker, J. Volkert, *Raytracing in Grid Environments*, 3rd Austrian Grid Symposium, in: Proceedings of 3rd Austrian Grid Symposium, Linz, Österreich: Austrian Computer Society (OCG), pp. 53 - 62, 2009

T. Odaker, J. Volkert, *Raytracing in the Austrian Grid* (Poster), SC08 Austrian Grid exhibition. 15. - 21. November 2008, Austin, Texas

T. Odaker, *GPU basierte Rendering Engine für natürliche Umgebungen*, Diplomarbeit, Johannes Kepler Universität Linz, 2008

H. Meyer, T. Odaker, K. A. Hummel, *OMVis - A 3D Network Protocol Visualization*

Tool for OMNet++, SIMUTools 2010, in: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, Malaga, Spanien: ICST, 2010

U. Langer, T. Odaker, H. Yang, W. Zulehner, *Fluid-Structure Interaction (FSI) Simulation under the Grid Environment* (Poster), SC08 Austrian Grid exhibition. 15. - 21. November 2008, Austin, Texas

1.4 Aufbau der Arbeit

Die Kapitel dieser Arbeit behandeln im Einzelnen folgende Themen:

- **Kapitel 2, Grundlagen und verwandte Arbeiten** erläutert Grundlagen der Vereinfachung von Dreiecksnetzen. Es werden verschiedene Lösungsansätze beschrieben und auf vorhandene Algorithmen eingegangen. Dieses Kapitel liefert Informationen über den Ablauf und die Probleme derartiger Algorithmen und dient dem Verständnis der späteren Kapitel, inklusive des im Rahmen dieser Arbeit entwickelten Verfahrens.

- **Kapitel 3, Vertexgruppeneliminierung** beschreibt die einzelnen Aspekte und Abläufe des im Zuge dieser Arbeit entworfenen Verfahrens. Dabei wird auf Ziele und Schwierigkeiten eingegangen. Die Teilkomponenten werden aufgelistet und deren Zusammenhang sowie Interaktion erläutert. Im Anschluss daran sind die einzelnen Komponenten im Detail dargelegt. Die Gründe und Folgen für die Entscheidungen bei der Entwicklung werden erörtert und die Vor- und Nachteile sowie Einschränkungen der Komponenten diskutiert.

- **Kapitel 4, Umsetzung der Vertexgruppeneliminierung** beschäftigt sich mit der Realisierung des in Kapitel 3 beschriebenen Verfahrens. Es erklärt sowohl generische Implementierungsaspekte und Designentscheidungen als auch die Implementierung des Testsystems. Des Weiteren wird die Umsetzung der Teilkomponenten der Vertexgruppeneliminierung im Detail erörtert. Die Implementierung auf einer modernen GPU und die Probleme und Einschränkungen, die eine derartige Vorgehensweise birgt, werden vorgestellt.

- **Kapitel 5, Leistungs- und Ergebnisanalyse** beschreibt die Resultate der Vertexgruppeneliminierung. Hierbei werden die Ergebnisse der Effizienz- und Leistungsanalyse des implementierten Testsystems diskutiert und Probleme und Schwachstellen behandelt.

- **Kapitel 6, Zusammenfassung und Ausblick** fasst die Ergebnisse dieser Arbeit zusammen, erläutert die daraus gewonnenen Erkenntnisse und beschreibt mögliche weitere Entwicklungen und Verbesserungen für die Vertexgruppeneliminierung.

1.5 Terminologie

In diesem Abschnitt ist eine Menge an Begriffen erklärt, die im weiteren Verlauf ohne nähere Erläuterung verwendet werden. Die Erklärungen sollen dazu dienen, Missverständnissen vorzubeugen.

- **Objekt** Ein Objekt im Kontext dieser Arbeit bildet einen Teil der Szene. Es ist zumindest aus einem, meist aber mehreren Grundelementen, das sind Punkte und Polygone (häufig Dreiecke), aufgebaut [FVD82]. Besonderer Fokus liegt auf Objekten aus Dreiecken, da das hier vorgestellte Verfahren für Dreiecksnetze entwickelt wurde.

- **Szene** Eine Szene ist die Anordnung einer Menge von Objekten, die von der Lage einer Kamera bzw. eines Betrachters dargestellt wird. Für jedes Objekt sind eine Position sowie eine Rotation definiert, wodurch sich die Zusammenstellung der Szene ergibt. Für die Kamera bestimmt man einen Punkt im Raum sowie eine Blickrichtung. In interaktiven Szenen unterliegen die Daten der Kamera der Kontrolle des Benutzers, der sich zwischen den Objekten der Szene bewegen kann.

- **Blickpyramide** In der Computergraphik bezeichnet dies den Bereich vor der Kamera, in dem sich Objekte befinden können, so dass sie von Position und Blickwinkel der Kamera aus sichtbar sind. Die Blickpyramide wird bestimmt durch den Öffnungswinkel der Kamera sowie eine vordere und hintere Begrenzungsebene. Während Form und Größe konstant sind, können in interaktiven Szenen Position und Rotation durch Benutzerinteraktion manipuliert werden.

- **Vertex** Ein Vertex ist ein Punkt auf der Oberfläche eines Objekts. Vertizes bilden die Eckpunkte von Polygonen (häufig Dreiecke), mit deren Hilfe die Oberfläche des Objekts definiert ist.

- **Polygonnetz** Ein Polygonnetz ist eine Menge durch Kanten verbundener Vertizes, so dass Polygone gebildet werden. Diese bestimmen die Form eines dargestellten Objekts. Ein Vertex muss in zumindest einem Polygon enthalten sein, um einen Teil des Netzes darzustellen.

- **Dreiecksnetz** Eine Untergruppe der Polygonnetze sind Dreiecksnetze. Hier wird ein Objekt ausschließlich durch eine Menge an Dreiecken repräsentiert. Im Zuge dieser Arbeit wird vor allem auf Dreiecksnetze eingegangen, da das entwickelte Verfahren auf diesen beruht.

- **Benachbarte Vertizes** Zwei Vertizes eines Polygonnetzes werden im Zuge dieser Arbeit als benachbarte Vertizes (kurz: Nachbarn) bezeichnet, wenn eine Kante des Netzes existiert, die beide Vertizes verbindet.

- **Pixel** Pixel ist ein Begriff der Computergraphik, der aus den englischen Begriffen „picture“ und „element“ kreiert wurde und ist die kleinste Einheit eines Rasterbildes, also ein „Bildpunkt“. Im Gegensatz zu einem Punkt im mathematischen Sinn besitzt ein Pixel jedoch eine quadratische Fläche.

- **Vereinfachung** Eine Vereinfachung ist eine Repräsentation des Originals durch einen weniger komplexen Datensatz. In der Regel wird diese mit Hilfe eines Vereinfachungsalgorithmus, der auf das als Ausgangspunkt dienende Objekt angewandt wird, errechnet.

- **Visuelle Qualität der Vereinfachung** Die visuelle Qualität einer Vereinfachung bezeichnet hier die erkennbaren Unterschiede zwischen einem vereinfachten Objekt und dem Original. Je weniger die Unterschiede von einem Betrachter wahrgenommen bzw. als störend empfunden werden, desto besser ist die visuelle Qualität der Vereinfachung.

- **Detailgrad** Der Detailgrad (engl. „Level of detail“) beschreibt eine Menge von Detailstufen bei der Darstellung eines Objekts oder einer Szene. Hierbei können je nach Ansatz diskrete Detailstufen oder fließende Übergänge zum Einsatz kommen. Der Detailgrad verfolgt den Ansatz, die für die Darstellung benötigte Leistung auf Kosten der Qualität des resultierenden Bildes zu reduzieren.

- **Topologie** Ein Dreiecksnetz ist aufgebaut aus Vertizes, Kanten und Dreiecken. Zwei Vertizes bilden eine Kante, und ein Dreieck enthält drei Kanten. Neben den Vertexdaten muss für ein Dreiecksnetz also auch definiert sein, welche Vertizes eine Kante bilden und wie die Kanten zu Dreiecken verbunden sind. Im Rahmen dieser Arbeit bezeichnet der Begriff Topologie wie das Netz aus Kanten bzw. Dreiecken aufgebaut ist.

Kapitel 2

Grundlagen und verwandte Arbeiten

Dieses Kapitel beschäftigt sich zunächst mit den Grundlagen der Vereinfachung von Dreiecksnetzen. Anschließend werden Arten von unterschiedlichen Operationen bzw. Verfahren vorgestellt und beschrieben, in welchen Punkten sich diese unterscheiden. Im Zuge dieser Arbeit wurde ein Verfahren entwickelt, das Vereinfachungen von Dreiecksnetzen erstellt. Techniken zur Vereinfachung von anderen Datensätzen (etwa generischen Polygonnetzen) wurden somit als für diese Arbeit nicht relevant erachtet und nicht berücksichtigt.

In diesem Kapitel wird auf folgende Bereiche eingegangen:

- **1. Grundlagen** Beschreibt grundlegende Konzepte im Bereich der Vereinfachung von Dreiecksnetzen.
- **2. Grundlegende Techniken und Verfahren** Erläuterung einiger Techniken bzw. Verfahren, die häufig in Verbindung mit Vereinfachungen zum Einsatz kommen.
- **3. Vereinfachungsoperatoren** Gibt einen Überblick über eine Auswahl von weit verbreiteten Vereinfachungsoperatoren.
- **4. Vereinfachungsverfahren** Beschreibt mehrere häufig angewandte Verfahren basierend auf den zuvor präsentierten Vereinfachungsoperatoren.

2.1 Grundlagen

In der Computergraphik sind sehr detaillierte Modelle weit verbreitet. Die Komplexität der Geometrie erhöht jedoch nicht nur die Qualität der Darstellung. Jeder Vertex eines

Dreiecksnetzes fließt in die Berechnungen für die Darstellung mit ein. Komplexe Modelle mit einer großen Zahl an Vertizes erhöhen folglich die aufzuwendende Rechenzeit [XESV97].

Ansätze, um den Detailgrad von Dreiecksnetzen zu verringern und so den Rechenaufwand für die Darstellung zu reduzieren, sind schon seit langem im Einsatz und fallen unter die Gruppe der Systeme für den Detailgrad (engl. „Level of Detail“, kurz LoD). Vereinfachungen sind ein Ansatz, um Systeme für den Detailgrad einzusetzen. Dabei wird die Zahl der Grundelemente (Vertizes und Dreiecke), aus denen ein Objekt aufgebaut ist, reduziert. Je nach Art des Systems wird besonderer Wert gelegt auf:

- eine starke Reduktion der Vertizes bzw. Dreiecke
- einen Erhalt der Topologie
- einen Kompromiss aus beiden obigen Faktoren

Im Anschluss wird auf folgende grundlegende Konzepte, Verfahren und Techniken näher eingegangen:

- **1. Systeme zur Steuerung des Detailgrades** Präsentiert die Grundidee von Systemen zur Steuerung des Detailgrades.
- **2. Vereinfachung** Legt Grundlagen für die Vereinfachung von Dreiecksnetzen dar.
- **3. Topologie** Erklärung der Topologie eines Dreiecksnetzes sowie wichtiger Eigenschaften der Topologie und deren Manipulation durch Vereinfachungsoperatoren.

2.1.1 Systeme zur Steuerung des Detailgrades

Systeme zur Steuerung des Detailgrades werden heutzutage vielfach eingesetzt. Trotz der steigenden Rechenleistung der Hardware ist der Darstellungsaufwand zu reduzieren, da moderne Beleuchtungsverfahren und andere Berechnungen bei der Visualisierung von Szenen zu einem Anstieg der benötigten Berechnungszeit für die Darstellung führen. Die Reduktion der Komplexität der Geometriedaten bietet einen guten Ansatz zur Beschleunigung der Darstellung [COM98].

Der Sinn der Steuerung des Detailgrades besteht darin, Einfluss auf die benötigte Rechenleistung zur Darstellung einer Szene zu nehmen. Oftmals wird dies eingesetzt, um eine Obergrenze bei der Berechnungszeit einzuhalten. Die Steuerung des Detailgrades versucht, einen Kompromiss zwischen dem Aufwand für die Darstellung und der resultierenden Qualität zu finden. Viele Systeme haben zum Ziel, die Reduktion der Qualität

so zu steuern, dass sichtbare Auswirkungen minimiert werden [Eri00]. Der Detailgrad ist dabei nicht ausschließlich an eine Vereinfachung der Geometrie gebunden, sondern kann jede Art der Berechnung für die Darstellung beeinflussen (z. B. Beleuchtung, Animation, ...). Im Rahmen dieser Arbeit wird ausschließlich auf Vereinfachungen von Dreiecksnetzen als Methode zur Steuerung des Detailgrades eingegangen.

Der Ansatz zur Steuerung des Detailgrades beruht auf der Erzeugung eines Rasterbildes aus der gegebenen Szene. Bei der Berechnung können Details, die durch eine sehr hohe Zahl an Dreiecken gebildet werden, in der erzeugten Rasterisierung nicht repräsentiert werden. So lässt sich eine reduzierte Qualität der Geometriedaten einer Szene einsetzen, die jedoch durch die spätere Rasterisierung nur wenige wahrnehmbare Auswirkungen auf das Ergebnis hat [Cla76].

Abbildung 2.1: Beispiel mehrerer Detailstufen [CG09]

Abbildung 2.1 zeigt ein Beispiel für den Detailgrad. Dasselbe Objekt wird durch drei verschiedene Geometrien repräsentiert (mit sinkender Anzahl an Dreiecken von links nach rechts). Während links Details auf der Oberfläche erkennbar sind, gehen diese bei den vereinfachten Darstellungen verloren. Rechts ist nur eine kantige Repräsentation des originalen Objekts verblieben.

Systeme zur Steuerung des Detailgrades lassen sich grob in drei Gruppen einteilen [LWC+02] [ZMK02], die sich wesentlich in visueller Qualität und benötigtem Rechenaufwand unterscheiden:

- **1. Diskrete Systeme** Einfachste, statische Systeme zur Steuerung des Detailgrades.

- **2. Kontinuierliche Systeme** Verbesserter Ansatz für fließende Übergänge und höhere Qualität.

- **3. Blickwinkelabhängige Systeme** Aufwändigste Art der Systeme durch Einfluß der Position und Blickrichtung des Betrachters in die Steuerung des Detailgrades.

2.1.1.1 Diskrete Systeme

Diskrete Systeme zur Steuerung des Detailgrades gehören zu den am längsten eingesetzten Algorithmen. Hier werden mehrere diskrete Stufen eingeführt und mehrere Instanzen eines Dreiecksnetzes - eine Instanz per Stufe - generiert [Cla76]. Es wird jene Stufe mit der höchsten Dreiecksauflösung eingesetzt, die einen gewünschten, maximalen Darstellungsaufwand nicht überschreitet [ZMK02].

Im Detail bedeutet dies, dass in einem Vorberechnungsschritt eine Reihe von Vereinfachungen eines Objekts berechnet werden. Die Vereinfachung vor dem Darstellungsvorgang reduziert die benötigte Rechenleistung zur Laufzeit drastisch. Bei der Darstellung wird eine der gespeicherten Auflösungsstufen des Objekts ausgewählt und anstelle des Originals in die Szene eingefügt. Die Selektion erfordert nur minimale Rechenleistung und kann auch während der Erstellung des Rasterbildes ausgeführt werden. Somit wird die Anzahl der zu berechnenden Dreiecke reduziert [RLB10].

Die Vorteile der einfachen Umsetzung und geringen Rechenleistung stehen allerdings auch mehreren Problemen gegenüber. So ist zur Zeit der Vereinfachung des Dreiecksnetzes nicht bekannt, aus welcher Richtung bzw. Distanz ein Objekt zum Darstellungszeitpunkt betrachtet wird. Die durchgeführte Vereinfachung ist generisch und berücksichtigt nicht, welche Details vom Beobachter wie stark wahrgenommen werden können [LWC+02]. Ein weiteres Problem entsteht beim Übergang zwischen verschiedenen Auflösungsstufen des Dreiecksnetzes. Wird eine Grenze überschritten, so kommt eine andere Stufe zum Einsatz. Der Betrachter erkennt in interaktiven Szenen möglicherweise einen plötzlichen, sprunghaften Übergang zwischen Auflösungsstufen, der oft als störend wahrgenommen wird [RB92]. Ein weiterer Nachteil liegt in der Speicherung der unterschiedlichen Stufen. Ein Netz mit einer großen Anzahl an Vertexdaten kann zu erheblichem Speicherbedarf führen, wenn eine große Anzahl an Auflösungsstufen generiert wird, die getrennt abgespeichert werden muss.

Um sprunghafte Übergänge zwischen Detailstufen zu vermeiden, kommen kontinuierliche Systeme zum Einsatz. Sie werden im nächsten Abschnitt beschrieben.

2.1.1.2 Kontinuierliche Systeme

Kontinuierliche Ansätze beruhen nicht auf statisch vorberechneten Auflösungsstufen. Sie setzen auf einen fließenden Übergang der Anzahl an Vertizes bzw. Dreiecken eines Netzes - von einer minimalen Auflösung bis hin zu dem originalen Dreiecksnetz. Daher müssen die Berechnungen für die Vereinfachung dynamisch durchgeführt werden - also erst zur Laufzeit [Zhu01].

Diese Verfahren haben den Vorteil, dass im Vergleich zu diskreten Ansätzen ein gleichmäßiger Verlauf in interaktiven Szenen stattfindet [RCRG06]. Der Betrachter nimmt

keine störenden, sprunghaften Übergänge wahr, und die Anzahl der Vertizes kann besser an die Szene angepasst werden, was in einer höheren visuellen Qualität resultiert [dB00].

Der Nachteil besteht in einer längeren Laufzeit während des Darstellungsvorganges im Vergleich zu einer statischen Vorberechnung. Die am weitesten verbreiteten Algorithmen dieser Art beruhen auf der Vorberechnung einer Reihe von Vereinfachungsoperationen auf einem Dreiecksnetz. Zur Laufzeit werden unter Berücksichtigung der visuellen Auswirkungen mehrere dieser Operationen angewandt und somit eine Auflösungsstufe für das Dreiecksnetz berechnet, die den Anforderungen entspricht [LWC+02].

Die aufwändigste Art von Systemen zur Steuerung des Detailgrades sind blickwinkelabhängige Systeme.

2.1.1.3 Blickwinkelabhängige Systeme

Blickwinkelabhängige Systeme gehen im Vergleich zu kontinuierlichen Ansätzen noch einen Schritt weiter. Hier werden die Auswirkungen einer Vereinfachung unter Berücksichtigung der Position und Blickrichtung des Betrachters miteinbezogen [SM05]. Diese Systeme erlauben eine starke Reduktion der Vertexzahl in „unwichtigen“ Bereichen eines Dreiecksnetzes, während andere mit einer hohen Dreieckszahl repräsentiert werden [Sau01]. Auch innerhalb eines einzelnen Netzes können mehrere Detailgrade enthalten sein. Bereiche, die sich näher beim Betrachter befinden, werden durch mehr Dreiecke dargestellt, während weiter entfernte niedriger aufgelöst sind [LKH+00]. Der Blickwinkel des Betrachters wird in die Vereinfachungen miteinbezogen [LP02]. So werden etwa die Silhouette und gut sichtbare Bereiche besser aufgelöst, um die visuelle Qualität zu erhöhen [Hop97].

Man setzt diese Technik einerseits ein, um bessere Resultate zu erzielen, andererseits ist sie nötig, da in manchen Szenarien, sowohl diskrete als auch kontinuierliche Verfahren scheitern. Werden sehr große und schlecht segmentierte Objekte dargestellt (z. B. Terrains), erstrecken sich diese von der Nähe zum Betrachter bis hin zu großen Distanzen [Blo00a], [RHS98]. Diskrete und kontinuierliche Verfahren würden Operationen auf das gesamte Dreiecksnetz anwenden. Das Resultat wäre eine Vereinfachung, die nur für einen Teilbereich angemessen ist [Blo00b]. Mit Hilfe der blickwinkelabhängigen Systeme lassen sich Vereinfachungen erstellen, die den Anforderungen der Szene entsprechen [LWC+02].

Das bietet den weiteren Vorteil, dass nur jene Bereiche von Objekten berücksichtigt werden, die tatsächlich darzustellen sind. So kann eine Bearbeitung der Rückseiten von Objekten oder von Bereichen außerhalb der Blickpyramide übersprungen werden, um ein besser angepasstes Resultat zu erzielen und den Darstellungsaufwand zu reduzie-

ren, während der Ressourcenaufwand für das System zur Steuerung des Detailgrades in Grenzen gehalten wird (siehe Unterabschnitt 2.2.2).

Der nächste Abschnitt beschäftigt sich mit den Grundlagen der Berechnung einer Vereinfachung eines Objekts.

2.1.2 Vereinfachung

Ein wichtiger Faktor in der Computergraphik ist der benötigte Leistungsaufwand für die Darstellung eines Objekts. Überschreitet der gegebene Datensatz eine gewisse Komplexität, so ist eine interaktive Darstellung nicht mehr möglich, da sie sehr zeitaufwändig wird [XESV97]. Im Rahmen der Forschung im Bereich der Computergraphik spielen Vereinfachungsalgorithmen eine wichtige Rolle. Schon [Cla76] befasste sich mit diesem Problem und damit, wie man den benötigten Ressourcenaufwand reduzieren kann.

Um eine Vereinfachung eines Objekts zu berechnen, sind eine Fehlermetrik und ein oder mehrere Vereinfachungsoperatoren erforderlich [GWH01]. Eine Fehlermetrik bewertet die Auswirkung eines oder mehrerer Vertizes - bzw. deren Verlust - auf die Topologie des gesamten Dreiecksnetzes. Sie kann sich auf einzelne oder Gruppen von Vertizes beziehen, nur direkte Nachbarn oder aber großflächigere Auswirkungen berücksichtigen [Lue01]. Bei Vereinfachungen werden Fehlermetriken eingesetzt, um festzustellen, auf welchen Vertizes Vereinfachungsoperationen durchgeführt und welche Vertizes bevorzugt entfernt werden können, um die Oberfläche bestmöglich zu erhalten. Sie können auch festlegen, wann eine maximal gewünschte Vereinfachung erreicht und wie detailgetreu die vereinfachte Version eines Netzes in Bezug auf das Original ist.

Die Vereinfachungsoperation bestimmt, wie Vertizes aus dem Dreiecksnetz entfernt werden und wo Ersatzpositionen für einen oder mehrere Vertizes liegen können. Die Art der Vereinfachungsoperation hat Auswirkungen auf die Effizienz des Algorithmus und kann limitierender Faktor für die Qualität des finalen Netzes bzw. den Grad der Vereinfachung sein. Manche Vereinfachungsoperatoren können etwa die Topologie eines Objekts ignorieren und somit einen stärkeren Grad der Vereinfachung erreichen.

Vereinfachungen zur Laufzeit zu berechnen erfordert zusätzliche Rechenleistung. Es ist möglich, dass solch ein Algorithmus im Vergleich zur einfachen Darstellung des ursprünglichen Objekts mehr Ressourcen in Anspruch nimmt. Hier ist zu berücksichtigen, dass der Aufwand komplexer Animationen, Beleuchtungsverfahren, etc. im Anschluss an eine Vereinfachung durch eine verringerte Zahl an Vertizes bzw. Dreiecken reduziert wird.

Wichtige Faktoren sind der erreichte Grad der Vereinfachung und die Qualität des Resultats. Es muss davon ausgegangen werden, dass eine stärkere Vereinfachung auch eine höhere Beeinträchtigung der Qualität mit sich bringt. Je nach Abbruchkriterium bei Berechnung einer Vereinfachung kann man unterscheiden zwischen [LWC+02]:

- **Qualitätsbasierter Vereinfachung** Bei einer qualitätsbasierten Vereinfachung beabsichtigt man, die Vertex- bzw. Dreieckszahl eines Netzes zu reduzieren und dabei eine gewisse Mindestqualität des resultierenden Objekts beizubehalten. Es gilt, ein Maß für diese Qualität zu finden, mit Hilfe dessen eine Grenze für die maximale Vereinfachung definiert werden kann. Ein Algorithmus dieser Art führt Vereinfachungsoperationen durch. Nach jeder Operation wird überprüft, ob eine weitere Vereinfachung möglich ist, ohne das gewählte Qualitätskriterium zu verletzen. Diese Algorithmen eigenen sich vor allem für Szenarien, in denen gute visuelle Qualität vonnöten ist und weniger auf die erforderliche Leistung für die Darstellung geachtet wird [LE97].

- **Kostenbasierte Vereinfachung** Kostenbasierte Vereinfachungen sollen vorwiegend die Anzahl der darzustellenden Dreiecke reduzieren. Auch bei diesen Ansätzen wird ein Maß für den Fehler, der durch eine Vereinfachung entsteht, ermittelt. Es ist festzulegen, auf welchen Vertizes eine Vereinfachungsoperation durchgeführt wird. Das Hauptkriterium für das Beenden der Vereinfachung stellt eine vom Benutzer gewünschte, maximale Anzahl an Dreiecken dar. Das Ziel dieser Art der Vereinfachung ist, ein Dreiecksnetz so lange mit einem möglichst geringen Fehler zu vereinfachen, bis die geforderte Dreieckszahl erreicht bzw. unterschritten ist. Algorithmen dieser Kategorie eignen sich für Szenarien, in denen besonders auf Effizienz Wert gelegt wird. Sie führen jedoch meist zu stärker veränderten Dreiecksnetzen, da hierbei die Qualität des resultierenden Netzes nicht an erster Stelle steht, sondern vor allem auf die benötigte Leistung für die Darstellung geachtet wird [LWC+02].

Im nächsten Abschnitt wird auf die Topologie und den Zusammenhang mit den Vereinfachungsoperatoren eingegangen.

2.1.3 Topologie

Die Topologie eines Dreiecksnetzes beschreibt, wie die einzelnen Vertizes miteinander zu Kanten und die Kanten zu Dreiecken verbunden werden, also die Struktur eines Dreiecksnetzes [LLM+10]. Weit verbreitet kommt hierfür eine Liste zum Einsatz, die für jedes Dreieck die enthaltenen Vertizes enthält. Die lokale Topologie eines Dreiecks, einer Kante oder eines Vertex bezeichnet die Konnektivität in der unmittelbaren Nachbarschaft dieses Elements. Sie stellt einen wichtigen Faktor bei Manipulationen dar [LWC+02]. Wird eine Vereinfachungsoperation auf einem Dreiecksnetz durchgeführt, so kann die Topologie berücksichtigt oder ignoriert werden. Dies ist ebenfalls ein wichtiges Unterscheidungsmerkmal für Vereinfachungsoperatoren.

In dieser Eigenschaft liegt gleichzeitig ein limitierender Faktor. Da Löcher auch De-

tails in einem Objekt darstellen, die von manchen Algorithmen nicht entfernt werden können, sind der möglichen Vereinfachung Grenzen gesetzt, die bei einem Ignorieren der Topologie nicht existieren [SZL92][ESV98].

Auf folgende Konzepte wird näher eingegangen:

- **1. 2D-Mannigfaltigkeit** Eigenschaft der Topologie, die einschränkender Faktor bei der Auswahl von Vereinfachungsoperatoren sein kann.

- **2. Topologieerhaltung** Eigenschaft von Vereinfachungsoperatoren, die bestimmt, ob die 2D-Mannigfaltigkeit bei Vereinfachungen erhalten bleibt.

2.1.3.1 2D-Mannigfaltigkeit

Eine wichtige Eigenschaft für ein Dreiecksnetz ist die 2D-Mannigfaltigkeit. Ein Dreiecksnetz gilt als 2D-mannigfaltig, wenn für jeden Vertex ein geschlossener Ring aus Dreiecken gebildet werden kann, der eine einzelne Oberfläche darstellt. Ein Dreiecksnetz ist somit 2D-mannigfaltig, wenn jede Kante von zwei Dreiecken enthalten ist und jedes Dreieck drei benachbarte Dreiecke mit je einer gemeinsamen Kante besitzt [LWC+02].

Ein 2D-mannigfaltiges Netz mit Grenzen erlaubt Begrenzungskanten, die nur Teil eines Dreiecks sind [LWC+02]. Abbildung 2.2 zeigt ein Beispiel für ein 2D-mannigfaltiges Dreiecksnetz mit Grenzen. Kanten, die eine Grenze bilden, sind gestrichelt dargestellt. Jede Kante ist entweder eine Grenze oder in 2 Dreiecken enthalten.

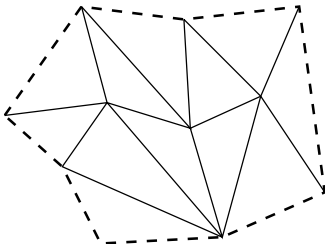

Abbildung 2.2: Beispiel eines 2D-mannigfaltigen Dreiecksnetzes mit Grenzen

Bei der Entfernung einer Kante aus einem Dreiecksnetz werden in einem 2D-mannigfaltigen Netz mit Grenzen immer ein oder zwei Dreiecke gelöscht [Gar99].

Nicht 2D-mannigfaltige Netze (mit Grenzen) sind jedoch nicht für alle Vereinfachungsverfahren geeignet, da manche die Topologie des Netzes in Betracht ziehen. Dann ist für diese Algorithmen nicht mehr bestimmt, wie die Vereinfachung durchgeführt werden soll. Ein weit verbreitetes Beispiel für einen Algorithmus, der nur für

2D-mannigfaltige Dreiecksnetze mit Grenzen definiert ist, findet sich bei Hoppe et al. [HDD+93]. Hierbei werden Vereinfachungen nur entlang von Kanten durchgeführt. Hoppe et al. definieren, dass das Verschmelzen zweier Vertizes eine Kante und ein oder zwei Dreiecke entfernt. Werden bei dieser Operation mehrere Dreiecke entfernt, so ist das Ergebnis nicht festgelegt. Bei jeder Vereinfachungsoperation wird darauf geachtet, dass die neu erzeugte Geometrie 2D-mannigfaltig (mit Grenzen) ist und somit bei späteren Vereinfachungen keine Fehler im Netz verursacht.

Die Topologieerhaltung bestimmt, ob ein Vereinfachungsoperator die Mannigfaltigkeit eines Dreiecksnetzes erhält.

2.1.3.2 Topologieerhaltung

Topologieerhaltende Verfahren berücksichtigen die Topologie eines Dreiecksnetzes bei der Manipulation der Oberfläche und führen nur Operationen aus, welche die Mannigfaltigkeit des Netzes erhalten [Tur92]. Derartige Verfahren können keine Löcher in der Oberfläche von Objekten schließen oder getrennte Oberflächen verschmelzen.

Im Gegensatz dazu ignorieren nicht-topologieerhaltende Algorithmen die Struktur eines Netzes. Sie können Löcher verschließen und im Extremfall sogar mehrere Objekte miteinander verschmelzen. Dies hat den Vorteil, dass sie eine sehr starke Vereinfachung erreichen, jedoch meist auf Kosten der resultierenden Qualität [LE97]. Ein Beispiel für ein derartiges Verfahren ist der in [LT97] beschriebene Algorithmus. Auf die bestehenden Dreiecke eines Netzes wird keine Rücksicht genommen. Vertizes werden ausschließlich aufgrund von räumlichen Gegebenheiten zusammengefasst. Die veränderte Oberfläche wird anschließend aus den neu erzeugten Vertizes aufgebaut.

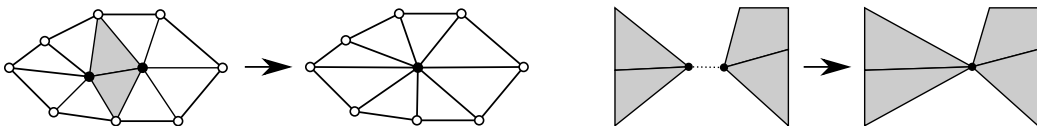

Abbildung 2.3: Beispiele für Topologieerhaltung

Abbildung 2.3 zeigt zwei unterschiedliche Vereinfachungsoperatoren. Auf beiden Seiten werden zwei Vertizes zu einem einzelnen Punkt verschmolzen. Auf der linken Seite geschieht dies entlang einer bestehenden Kante des Dreiecksnetzes. Diese Operation ist topologieerhaltend. Im Gegenzug dazu ist auf der rechten Seite eine nicht-topologieerhaltende Operation zu sehen. Hierbei werden zwar ebenfalls zwei Vertizes zu einem verschmolzen, der Unterschied besteht jedoch darin, dass diese nicht durch eine

Kante miteinander verbunden waren. Somit sind zwei getrennte Oberflächen zu einer einzelnen vereint.

Im Anschluss werden einige grundlegende Techniken für die Optimierung von Vereinfachungsverfahren erläutert.

2.2 Grundlegende Techniken und Verfahren

Neben grundlegenden Konzepten existieren auch einige Verfahren und Techniken, die verbreitet Anwendung im Bereich der Vereinfachung finden. In diesem Abschnitt werden folgende Ansätze präsentiert:

- **1. Fließende Interpolation** Beschreibt einen Ansatz für fließende Übergänge bei der Berechnung von Vereinfachungen.

- **2. Dreiecksfilterung** Erläutert das Konzept der Filterung von Dreiecken und seinen Einsatz bei Vereinfachungen.

- **3. Hierarchische Datenstrukturen** Grundlagen über Datenstrukturen, die bei Vereinfachungsalgorithmen weit verbreitet zum Einsatz kommen.

2.2.1 Fließende Interpolation

Die fließende Interpolation (engl. „geomorphing“) ist eine Technik, um einen fließenden Übergang zwischen einzelnen Stufen einer Vereinfachung eines Objekts zu ermöglichen. Bei einem Wechsel zu einer stärker vereinfachten Version kann es zu sprunghaften Übergängen kommen, die vom Betrachter in einer interaktiven oder animierten Szene erkennbar sein können.

Bei fließenden Interpolationen wird eine vereinfachte Version eines Dreiecksnetzes berechnet. Anstatt das Modell in einer Szene sprunghaft durch die vereinfachte Version zu ersetzen, werden Vertizes schrittweise auf die in der Vereinfachung für sie gewählten Ersatzpositionen verschoben, indem die Vertexpositionen zwischen jenen im originalen und vereinfachten Dreiecksnetz interpoliert werden [Hop97]. Der Vorteil dieses Ansatzes besteht in einem stufenlosen Übergang zwischen dem originalen Netz und der vereinfachten Version bei interaktiven Vereinfachungen. Aufgrund der Interpolation der Positionen der Vertizes ist es jedoch vonnöten, dass die zusätzlichen Vertizes des originalen Netzes während der Verschiebung mit manipulierten Positionen beibehalten werden.

Die verbesserte Qualität der Übergänge erfordert somit zusätzliche Vertizes und einen höheren Darstellungsaufwand [Zac02].

Neben der Verbesserung der optischen Qualität kann die Zahl der durchzuführenden Vereinfachungsoperationen durch Berücksichtigung der Dreiecksfilterung reduziert werden.

2.2.2 Dreiecksfilterung

Im Zuge von Vereinfachungsoperationen ist zu unterscheiden zwischen Verfahren, die statische, vorberechnete Vereinfachungen eines Objekts generieren und jenen, die den Blickwinkel bzw. die Position des Betrachters berücksichtigen. Letztere können die benötigten Vereinfachungsoperationen bei zur Laufzeit berechneten Verfahren optimieren bzw. deren Anzahl minimieren, indem die Sichtbarkeit von Dreiecken in Betracht gezogen wird.

Bei der Darstellung von Dreiecksnetzen wird zur Reduktion der Anzahl der Dreiecke eine Filterung angewandt. Es werden nur jene Teile der Geometrie berücksichtigt, die in der aktuellen Szene sichtbar sind [DT07]. Wird ein Dreieck durch die Filterung entfernt, so muss es bei der Berechnung einer Vereinfachung zur Laufzeit möglicherweise nicht berücksichtigt werden. Bei der Berechnung von Vereinfachungen ist jedoch zu beachten, dass manche Fehlermetriken und Vereinfachungsoperatoren nicht nur auf isolierten Vertizes bzw. Dreiecken operieren, sondern größere Bereiche einer Oberfläche berücksichtigen. Nicht jedes durch die Filterung entfernte Dreieck ist somit für ein Vereinfachungsverfahren wirklich irrelevant.

Es wird zwischen drei verschiedenen Arten der Filterung unterschieden:

- Blickpyramidenfilterung
- Filterung von Rückseiten
- Filterung von Verdeckungen

Die Blickpyramidenfilterung bezeichnet das Entfernen von Dreiecken, die sich außerhalb der aktuellen Blickpyramide der Kamera befinden. Diese Teile der Geometrie sind für den Betrachter nicht sichtbar und können somit verworfen werden [GKM93]. Beim zweidimensionalen Beispiel in Abbildung 2.4 befindet sich Objekt O_1 in der Blickpyramide und wird dargestellt. O_2 liegt außerhalb, und die Dreiecke, die O_2 bilden, können laut Blickpyramidenfilterung für die Darstellung ignoriert werden.

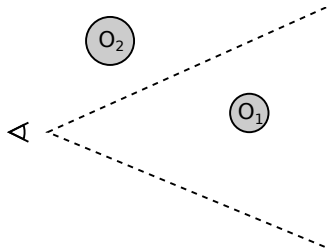

Abbildung 2.4: Beispiel Blickpyramidenfilterung

Die Filterung von Rückseiten basiert auf der Idee, dass alle Dreiecke eine Vorder- und Rückseite besitzen [JRSW03]. Die Vorderseite eines Dreiecks wird über dessen Normalvektor bestimmt. Die Ausrichtung der Dreiecksnormalen wird mit dem Blickvektor des Betrachters verglichen. Die Filterung von Rückseiten definiert, dass ein Dreieck, dessen Vorderseite der Kamera zugewandt ist, dargestellt werden muss. Blickt der Betrachter auf die Rückseite, so kann es verworfen werden [FvDFH90]. Die Idee hinter diesem Ansatz besteht darin, dass die dem Betrachter abgewandte Seite eines Objekts nicht sichtbar und somit für die Darstellung in vielen Bereichen irrelevant ist. Daher kann die zu berechnende Dreieckszahl einer Szene reduziert werden, weil man die Rückseite von Objekten vom Darstellungsvorgang ausnimmt. Abbildung 2.5 zeigt ein zweidimensionales Beispiel für die Filterung der Rückseiten. Dreiecke mit dem Normalvektor n_1 sind dem Betrachter zugewandt und werden dargestellt. Dreiecke mit den Normalvektoren n_2, n_3 und n_4 sind vom Betrachter abgewandt. Sie können laut Definition der Filterung der Rückseiten bei der Darstellung ignoriert werden.

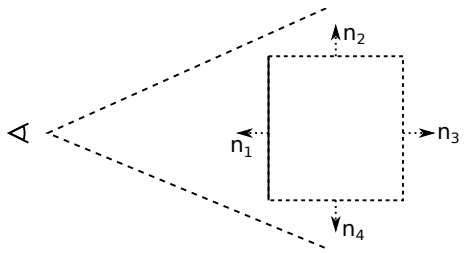

Abbildung 2.5: Beispiel Filterung von Rückseiten

Die Filterung von Verdeckungen kommt zum Einsatz, wenn ein Teil einer Szene aus Sicht des Betrachters von einem Objekt verdeckt wird. In diesem Fall können alle nicht sichtbaren Dreiecke vom Darstellungsprozess ausgeschlossen werden [FvDFH90]. Ein zweidimensionales Beispiel für diesen Ansatz ist in Abbildung 2.6 dargestellt. Objekt O_1 verdeckt aus Sicht des Betrachters O_2 vollständig. Die Dreiecke in O_2 werden bei Einsatz der Filterung von Verdeckungen nicht dargestellt.

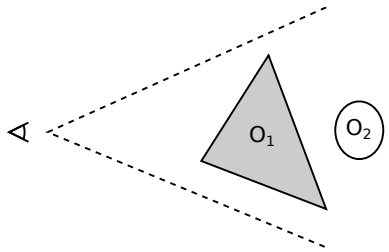

Abbildung 2.6: Beispiel Filterung von Verdeckungen

Unter Berücksichtigung dieser Methoden muss eine Vereinfachung auf einem Bereich eines Dreiecksnetzes nur berechnet werden, wenn dieser:

- dem Betrachter zugewandt ist.
- sich innerhalb der Blickpyramide befindet.
- nicht von einem Objekt verdeckt ist.

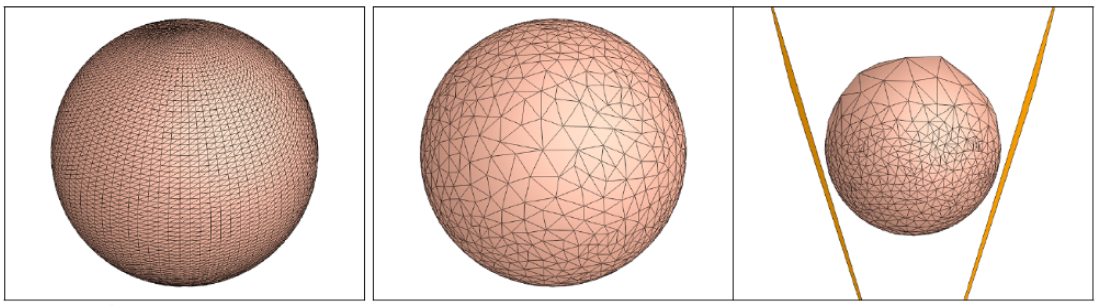

Abbildung 2.7: Auswirkung der Filterung auf Vereinfachungen [Hop97]

Abbildung 2.7 zeigt ein Anwendungsbeispiel für den Einsatz der Filterung anhand des Algorithmus der schrittweisen Dreiecksnetze ([Hop97], genauere Erklärung des Algorithmus in Unterabschnitt 2.4.1). Bei diesem Ansatz wird ein Objekt (links in Abbildung 2.7) in einem Vorberechnungsschritt durch eine Reihe von Vereinfachungsoperationen stark vereinfacht. Zur Laufzeit werden manche der ausgeführten Vereinfachungsoperationen reversiert, um den Detailgrad und somit die Qualität zu verbessern. Das Beispiel in Abbildung 2.7 stellt eine Sphäre aus Sicht des Betrachters (Mitte) dar. In der rechten Abbildung ist die Geometrie auf der Rückseite der Sphäre erkennbar. Der Benutzer betrachtet die Sphäre von unten, die beiden gelben Flächen zeigen die Blickpyramide. Auf der dem Betrachter zugewandten Seite wurde die Granularität der Oberfläche verbessert, um ein optisch ansprechenderes Resultat zu erhalten. Auf der

vom Betrachter abgewandten Seite hingegen wurden keine zusätzlichen Dreiecke eingefügt, da dieser Bereich aufgrund der Definition der Filterung von Rückseiten bei der Darstellung nicht berücksichtigt wird. Die stark vereinfachte Version des Dreiecksnetzes ist dort zu erkennen.

Im nächsten Abschnitt wird auf hierarchische Datenstrukturen eingegangen, die bei Vereinfachungsalgorithmen häufig zum Einsatz kommen.

2.2.3 Hierarchische Datenstrukturen

Eine hierarchische Datenstruktur dient dazu, Abhängigkeiten zwischen den Knoten der Datenstruktur zu definieren. Ausgehend von einem Wurzelknoten können - je nach Art der Datenstruktur - jedem Knoten ein oder mehrere Kindknoten zugeordnet werden (z. B. um eine Reihenfolge von auszuführenden Vereinfachungsoperationen zu definieren [Hop97]). Andere Algorithmen wie etwa jener von Rossignac und Borrell [RB92] verwenden diese nicht für die Speicherung von Vereinfachungsoperationen, sondern greifen auf sie zurück, um den Objektraum in eine Menge von Teilbereichen zu unterteilen.

Im Zuge der Zerlegung des Objektraumes mit Hilfe von hierarchischen Datenstrukturen wird auf folgende Konzepte näher eingegangen:

- **Hierarchische Objektraumunterteilung** Beschreibt die Zerlegung des Objektraumes mit Hilfe einer hierarchischen Datenstruktur.

- **Arten von Begrenzungsvolumen** Beschreibt wie Teilräume gespeichert werden können.

2.2.3.1 Hierarchische Objektraumunterteilung

Bei der Teilung des Objektraumes in eine Menge von Unterräumen kommt häufig eine hierarchische Zerlegung zum Einsatz. Ein Bereich des Raumes wird in einem Knoten einer hierarchischen Datenstruktur gespeichert. Kindknoten werden für eine weitere Zerlegung eingesetzt. Ein Kindknoten enthält einen Teilraum des Vaterknotens. Je nach Zerlegung und Definition der Unterteilung können Kindknoten exklusive Teilräume speichern oder sich überlappen. Auch die vollständige Abdeckung eines Teilraumes durch die Kindknoten ist abhängig vom Anwendungsgebiet.

Der Algorithmus in [SW03] beginnt mit der Berechnung einer Menge von Bereichen, die im Anschluss immer weiter zerlegt werden. Für die Speicherung der einzelnen Teilräume kommt eine Baumstruktur zur Anwendung. In [SW03] ist dies ein „Octree“ (lat.

„octo“: acht, engl. „tree“: Baum), also ein Baum bei dem jeder Knoten bis zu acht Kindknoten besitzt. Im Fall von Vertexgruppierungs-Algorithmen wie etwa [RB92], [SW03] oder [DT07] werden diese Teilbereiche eingesetzt, um Gruppen von Vertizes zu definieren, die auf einen Punkt reduziert werden sollen.

Die Speicherung der Bereiche erfolgt durch die Definition eines Raumes, dem sogenannten Begrenzungsvolumen (engl. „bounding box“, „bounding volume“). Das Volumen eines Knotens umgibt eine gewisse Region im Raum und somit auch die der Kindknoten. Ein Bereich des Objektraumes, der einem Knoten zugeteilt wird, kann rekursiv so lange erneut unterteilt werden, bis eine gewünschte Granularität erreicht ist.

Im nächsten Abschnitt wird auf Arten von Begrenzungsvolumen näher eingegangen.

2.2.3.2 Arten von Begrenzungsvolumen

Bei den Begrenzungsvolumen existieren verschiedene Varianten. Diese unterscheiden sich vor allem in der Form des Volumens und in seiner Orientierung im Raum. Bei Vereinfachungsalgorithmen kommen vor allem quaderförmige (z. B. [RB92]) und sphärische Volumen (z. B. [LT97]) zum Einsatz. Quaderförmige Begrenzungsvolumen teilt man erneut in achsparallele und objektorientierte Volumen ein [Gre94]. Die Arten unterscheiden sich einerseits darin, wie effizient das Objekt vom Volumen umfasst wird, andererseits in der benötigten Rechenleistung für die Generierung und der Aktualisierung bei Veränderungen in der Szene.

- **Achsparallele Begrenzungsvolumen** Diese sehr einfachen Begrenzungsvolumen bestimmen einen Quader, der um ein Objekt gelegt wird und parallel zu den Achsen des Raumes ausgerichtet ist [Arv90]. Die Länge einer Quaderseite entspricht der Ausdehnung eines Objekts entlang dieser Achse. Diese Art von Volumen ist sehr einfach zu berechnen und zu implementieren, sie besitzt jedoch den Nachteil, dass bei einer Rotation des Objekts das Begrenzungsvolumen nicht mehr korrekt ist und erneut bestimmt werden muss. Daher ist der Einsatz bei dynamischen Szenen nicht vorteilhaft. Des Weiteren können in achsparallelen Begrenzungsvolumen unnötig große Bereiche enthalten sein.

- **Objektorientierte Begrenzungsvolumen** Wie bei den achsparallelen Volumen wird ein Quader um das Objekt gelegt. Der Unterschied besteht jedoch darin, dass hier keine Achsparallelität vorliegt, sondern darauf geachtet wird, dass das Volumen des Quaders möglichst gering ausfällt. Dieses Verfahren bietet den Vorteil, dass bei einer Rotation des Objekts keine erneute Berechnung des Begrenzungs-

volumens durchgeführt werden muss, sondern das Volumen mit Hilfe derselben Transformation angepasst werden kann.

- **Sphärische Begrenzungsvolumen** Im Gegenzug zu den oben beschriebenen Volumen kommt hier eine Kugel anstelle eines Quaders zum Einsatz. Das Volumen ist durch einen Punkt und einen Radius bestimmt. Der Vorteil besteht in der effizienten Speicherung, der Unabhängigkeit von Rotationen und den einfachen Tests auf Distanz bzw. Schnittpunkte.

Bei Objektraumhierarchien haben quaderförmige Begrenzungsvolumen den Vorteil, dass sie die gesamte Region ohne Überlappungen abdecken können. Wird ein Volumen in einer hierarchischen Struktur weiter unterteilt, so ist es bei sphärischen Volumen nicht möglich, diese so zu bestimmen, dass keine Bereiche des Raumes in zwei oder mehreren enthalten sind und gleichzeitig der gesamte Raum repräsentiert wird. Trotz dieses Nachteils kommen sphärische Ansätze in Vereinfachungsalgorithmen zur Anwendung ([LT97]), um Bereiche des Objektraumes zu bestimmen.

Die Abbildungen 2.8 bis 2.10 zeigen zweidimensionale Beispiele für diese Arten von Begrenzungen, um die Ausrichtung und Effizienz der Eingrenzung zu verdeutlichen.

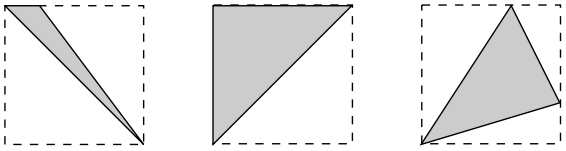

Abbildung 2.8: Achsparallele Ausrichtung

In Abbildung 2.8 sind Beispiele für die achsparallele Ausrichtung dargestellt. Wie hier ersichtlich ist, sind manche Dreiecke nicht optimal von dem Rechteck umgeben, so dass die Begrenzung größer ausfällt als für das Dreieck nötig wäre.

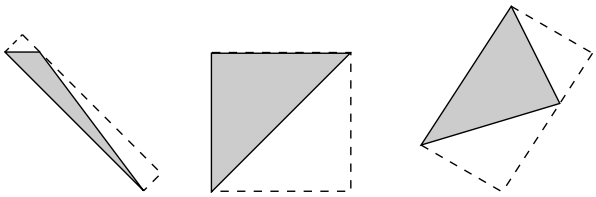

Abbildung 2.9: Objektorientierte Ausrichtung

Abbildung 2.9 visualisiert die objektorientierte Ausrichtung. Hier sind die Rechtecke teilweise besser an die zu umfassenden Dreiecke angepasst als ihre achsparallelen Äquivalente.

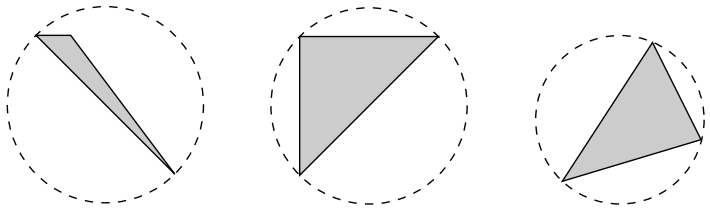

Abbildung 2.10: Sphärische Begrenzungen

Abbildung 2.10 zeigt zweidimensionale Beispiele für die sphärischen Begrenzungen. Hier ist erkennbar, dass die Effizienz stark von der Form des eingefassten Objekts abhängt. So kann bei sphärischen Begrenzungsvolumen in ungünstigen Fällen ein sehr großes Volumen erforderlich sein, um ein Objekt zu umfassen.

Im Zuge der Forschung über die Vereinfachung von Dreiecksnetzen wurde eine Reihe von Vereinfachungsoperationen entwickelt, die festlegen, wie Elemente aus einem Dreiecksnetz entfernt werden. An dieser Stelle werden einige weit verbreitete Vereinfachungsoperatoren vorgestellt. Dies dient dem Verständnis der später in diesem Kapitel beschriebenen Verfahren und Algorithmen für die Vereinfachung von Dreiecksnetzen.

2.3 Vereinfachungsoperatoren

Ein Vereinfachungsoperator ist eine Manipulation eines Dreiecksnetzes, um ein oder mehrere Vertizes bzw. Dreiecke aus dem Netz zu entfernen. Das Ergebnis der Ausführung solcher Operationen ist eine vereinfachte Version des ursprünglichen Objekts jedoch meist mit weniger Details. Vereinfachungsoperatoren unterscheiden sich einerseits darin, wie Vertizes bzw. Dreiecke entfernt werden, andererseits wie die neue Oberfläche ermittelt wird.

Nicht jeder Operator kann auf alle Arten von Dreiecksnetzen angewendet werden. Manche der hier vorgestellten Verfahren basieren auf der Topologie eines Netzes und können somit nur bei 2D-mannigfaltigen Netzen durchgeführt werden.

Im Anschluss werden einige, in der Literatur häufig erwähnte Operatoren näher beschrieben:

- **1. Kantenreduktion** Ersetzen einer Kante durch einen Vertex.

- **2. Eingeschränkte Kantenreduktion** Spezialfall der Kantenreduktion.
- **3. Vertexpaarverschmelzung** Ersetzen zweier nicht verbundener Vertizes durch einen Vertex.
- **4. Dreiecksreduktion** Ersetzen eines Dreiecks durch einen Vertex.
- **5. Zellenreduktion** Zusammenfassen aller Vertizes innerhalb eines Bereichs.
- **6. Vertexeliminierung** Löschen von einzelnen Vertizes.
- **7. Polygonverschmelzung** Löschen von mehreren Vertizes.
- **8. Vergleich der präsentierten Vereinfachungsoperatoren** Zeigt eine Gegenüberstellung der wichtigsten Eigenschaften der präsentierten Vereinfachungsoperatoren.

2.3.1 Kantenreduktion

Die Kantenreduktion (engl. „edge collapse“) wurde erstmals in Hoppe et al. [HDD+93] vorgestellt. Eine Kantenreduktion entfernt zwei durch eine Kante verbundene Vertizes und ersetzt sie durch einen einzelnen Vertex [Pup98]. Diese Manipulation der Topologie entfernt die Kante aus dem Dreiecksnetz. Das Entfernen der Vertizes des ursprünglichen Netzes macht die Bearbeitung aller mit ihnen gebildeten Dreiecke erforderlich. Beinhaltet ein Dreieck beide entfernten Vertizes, so wird es auf eine Linie reduziert und gänzlich aus dem Dreiecksnetz entfernt. Ist nur ein entfernter Vertex in einem Dreieck enthalten, muss das Dreieck angepasst werden, so dass es den neu geschaffenen Vertex enthält.

Gleichzeitig mit der Kantenreduktion wird in [HDD+93] auf die inverse Operation zurückgegriffen - die Vertexteilung (engl. „vertex split“). Sie macht die Veränderung einer Kantenreduktion rückgängig, ersetzt also einen einzelnen Vertex durch zwei Vertizes, eine Kante und ein oder zwei zusätzliche Dreiecke. Dies erlaubt es, eine durch Kantenreduktionen erreichte Vereinfachung umzukehren und somit wieder das originale Dreiecksnetz zu erhalten.

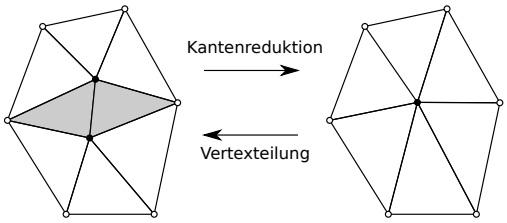

Abbildung 2.11: Vertexteilung und Kantenreduktion

Abbildung 2.11 zeigt ein Beispiel für die Kantenreduktion sowie die Vertexteilung. Die beiden grau markierten Dreiecke im Dreiecksnetz auf der linken Seite sind im rechten Dreiecksnetz durch eine Kantenreduktion entfernt und durch einen einzelnen Vertex ersetzt worden. Die Vertexteilung (von rechts nach links in Abbildung 2.11) kehrt diese Operation um.

Dieser Operator ist in mehreren Algorithmen als Basisoperation zu finden, so z. B. in [Hop97], [GH97] oder [XV96]. Die Kantenreduktion hat jedoch den Nachteil, dass nicht jedes Vertexpaar durch einen beliebigen Punkt ersetzbar ist, da es zu Faltungen im resultierenden Netz kommen kann [XESV97]. Bei einer Faltung wird ein Dreieck manipuliert, so dass es eine „Falte“ auf der Oberfläche bildet.

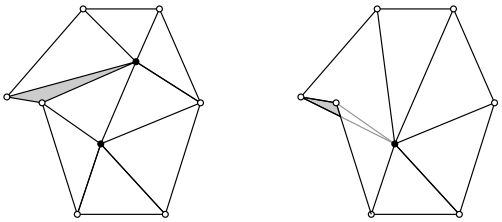

Abbildung 2.12: Beispiel einer Faltung

Abbildung 2.12 zeigt ein Beispiel für eine Faltung. Wird auf den beiden Vertizes in der Mitte des linken Dreiecksnetzes eine Kantenreduktion ausgeführt, kehrt sich die Ausrichtung des grau markierten Dreiecks um und eine „Falte“ wird geschaffen.

Eine Kantenreduktion kann auch eine inkonsistente Topologie verursachen. Das bedeutet, dass zwei Dreiecke mit identen Eckpunkten gebildet werden, die Reihenfolge der Eckpunkte jedoch vertauscht wird. Eines der beiden Dreiecke ist „gekippt“, die Normalvektoren der beiden überlappenden Dreiecke sind also gegenläufig.

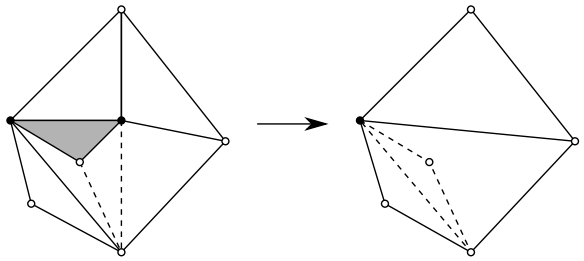

Abbildung 2.13: Beispiel einer inkonsistenten Topologie

Abbildung 2.13 zeigt ein Beispiel für eine inkonsistente Topologie. Werden das grau markierte Dreieck sowie der mittlere Vertex auf der linken Seite entfernt, so wird das gestrichelt gekennzeichnete Dreieck rotiert und zwei identische Dreiecke mit gegenläufigen Normalvektoren geschaffen.

Faltungen und überlappende Dreiecke dürfen beim Einsatz von Kantenreduktionen nicht entstehen, da einerseits redundante Dreiecke erzeugt würden und andererseits isolierte, aus dem Objekt ragende Dreiecke die Folge wären. Ein entsprechender Algorithmus muss derartige, illegale Operationen erkennen und deren Ausführung verhindern.

Die Kantenreduktion ist ein topologieerhaltender Vereinfachungsoperator. Sie agiert nur auf Vertexpaaren, die durch Kanten verbunden sind. Es kommt also zu keinem Entfernen von Löchern im Netz, wodurch die maximale Vereinfachung potentiell begrenzt wird. Dieser Operator ist nur für 2D-mannigfaltige Dreiecksnetze definiert [LWC$^+$02].

2.3.2 Eingeschränkte Kantenreduktion

Die Kantenreduktion definiert das Ersetzen einer Kante (und somit der beiden Endpunkte) durch einen einzelnen Vertex. Der neue Vertex ist frei plazierbar. Die eingeschränkte Kantenreduktion (engl. „half edge collapse“) ist eine Sonderform der Kantenreduktion. Hier wird keine freie Auswahl der Ersatzposition erlaubt, sondern einer der beiden an der Operation beteiligten Vertizes bestimmt. Als Folge entspricht die eingeschränkte Kantenreduktion einer Verschiebung eines Punktes entlang einer Kante auf die Position eines benachbarten Vertex [DDFM$^+$05].

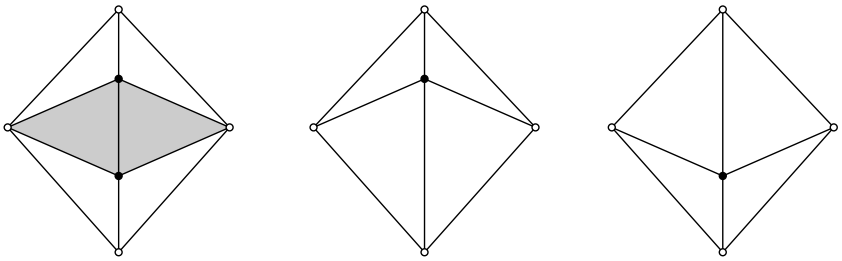

Abbildung 2.14: Mögliche Ergebnisse einer eingeschränkten Kantenreduktion

Abbildung 2.14 zeigt ein Beispiel für eine eingeschränkte Kantenreduktion. Auf der linken Seite ist das ursprüngliche Dreiecksnetz. Die gemeinsame Kante der beiden grau markierten Dreiecke soll durch eine eingeschränkte Kantenreduktion entfernt werden. Die anderen beiden Skizzen zeigen die möglichen Resultate nach Anwendung der eingeschränkten Kantenreduktion.

Die eingeschränkte Kantenreduktion kann dieselben Probleme mit invertierten Dreiecken und verletzter Netzintegrität verursachen wie die Kantenreduktion.

2.3.3 Vertexpaarverschmelzung

Eine Vertexpaarverschmelzung (engl. „vertex pair collapse“) funktioniert ähnlich der Kantenreduktion. Hier werden zwei Vertizes aus dem Netz entfernt und durch einen einzelnen Vertex ersetzt [Sch97]. Während die Kantenreduktion nur Vertexpaare betrachtet, die durch eine Kante miteinander verbunden sind, verschmilzt die Vertexpaarverschmelzung zwei nicht verknüpfte Vertizes (Abbildung 2.15).

Eine Kantenreduktion ist ein Verfahren, das die Topologie eines Netzes berücksichtigt und aus diesem Grund Operationen nur entlang von Kanten durchführt. Die Oberfläche wird zwar vereinfacht, Löcher, Tunnel und ähnliche Ausprägungen in einem Objekt bleiben jedoch erhalten. Im Gegenzug verfolgt die Vertexpaarverschmelzung die Idee, dass Vereinfachungen auch das Schließen von Löchern und sogar das Verschmelzen von Objekten beinhalten darf. Durch das Entfernen von nicht verbundenen Vertizes werden keine Dreiecke aus dem Netz gelöscht, sondern nur ein einzelner Vertex. Eine Vertexpaarverschmelzung kann geschlossene Flächen erzeugen. In Abbildung 2.15 ist dieses Vorgehen sichtbar. Die beiden eigentlich unabhängigen Flächen auf der linken Seite werden zu einer gemeinsamen Oberfläche verbunden.

Da mittels dieser Operation Löcher und Tunnel in einem Dreiecksnetz eliminiert werden, ist dieses Verfahren nicht topologieerhaltend. Diese Eigenschaft erlaubt es der Vertexpaarverschmelzung, sehr starke Vereinfachungen zu berechnen.

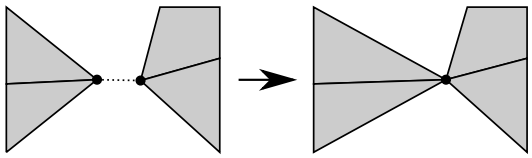

Abbildung 2.15: Vertexpaarverschmelzung

Wird bei dieser Operation ein Vertex entfernt, werden alle damit gebildeten Dreiecke analog zur Kantenreduktion manipuliert, um den neu geschaffenen Vertex anstelle des entfernten Vertex zu beinhalten. Dieser ist Kandidat für weitere Vereinfachungsoperationen. Durch die Vertexpaarverschmelzung kann eine Reihe zusätzlicher Kanten geschaffen werden, die im Anschluss für eine mögliche Kantenreduktion zum Einsatz kommt [LWC+02].

2.3.4 Dreiecksreduktion

Eine Dreiecksreduktion (engl. „triangle collapse“) funktioniert - ähnlich der Kantenreduktion und der Vertexpaarverschmelzung - durch das Verschmelzen von mehreren Vertizes zu einem einzelnen Punkt. Während bei den oben vorgestellten Techniken immer ein Paar durch einen Vertex ersetzt wird, reduziert die Dreiecksreduktion drei Vertizes, die gemeinsam ein Dreieck bilden, auf einen einzelnen Punkt [ZJK01]. Dieser Operator entspricht somit einer Ausführung von zwei Kantenreduktionen in einem Schritt und ist analog zur Kantenreduktion topologieerhaltend, da nur durch Kanten verbundene Vertizes für eine Manipulation in Frage kommen. Der Vorteil der Dreiecksreduktion liegt in der stärkeren Vereinfachung, da in einem Schritt mehrere Vertizes entfernt werden können. Durch die gröbere Granularität dieses Operators ist eine Bewertung der Auswirkungen auf das Dreiecksnetz schwieriger.

2.3.5 Zellenreduktion

Eine Zellenreduktion (engl. „cell collapse“) ist ein Vereinfachungsoperator, der eine Menge von Vertizes durch einen einzelnen Punkt ersetzt. Analog zur Vertexpaarverschmelzung muss dabei keine direkte Verbindung zwischen den einzelnen Vertizes bestehen [RB92].

Bei der Zellenreduktion wird ein Bereich festgelegt, der eine Menge von Vertizes enthält. Im Rahmen der Forschung wurden verschiedene Verfahren zur Bestimmung dieser Bereiche definiert, um die Qualität der Vereinfachung zu verbessern und die Verfahren zu optimieren. Manche Algorithmen verwenden regelmäßige Zellen ([RB92],

[DT07]), andere greifen auf Sphären zurück ([LT97]) und in weiteren Entwicklungen kommen Bereiche ungleicher Größe in Baumstrukturen zum Einsatz ([SW03]). Letztere erlauben die Erzeugung verschiedener Auflösungsstufen für die resultierende Vereinfachung. Sie haben auch den Vorteil, im Gegensatz zu regelmäßigen Gittern auf die Vertexverteilungen des Dreiecksnetzes einzugehen und so bessere Resultate zu erzielen ([LE97], [LT97]).

Für jede Zelle wird ein Ersatzvertex im Raum bestimmt. Dieser muss nicht mit einem Vertex in der Zelle identisch sein. Alle Vertizes innerhalb einer Zelle werden durch diesen einzelnen Vertex ersetzt. Das Ergebnis der Vereinfachung ist durch die Anzahl und Größe der Zellen steuerbar.

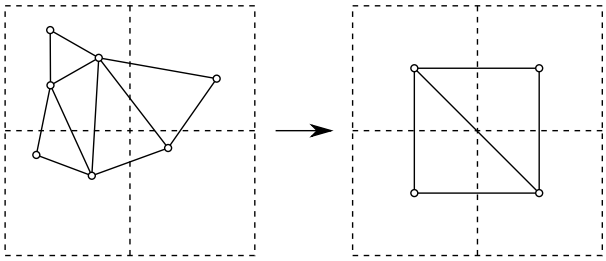

Abbildung 2.16: Beispiel Zellenreduktion

Abbildung 2.16 zeigt ein Beispiel einer Vereinfachung mittels Zellenreduktion. Auf der linken Seite ist das originale Dreiecksnetz dargestellt, das mit mehreren, regelmäßigen Zellen überlagert wird. Alle Vertizes innerhalb einer Zelle werden zu einem einzelnen Vertex zusammengefasst. Auf der rechten Seite ist das Ergebnis dieser Operation dargestellt. Als Ersatzposition wurde in diesem Beispiel das Zentrum der Zellen gewählt.

Das Problem dieser Operation liegt vor allem darin, dass sich die Anzahl der Vertizes des vereinfachten Objekts steuern lässt, dabei die Topologie aber nicht berücksichtigt wird. Die Verteilung der Vertizes im Raum beeinflusst den Aufbau der Zellen, die Form der Oberfläche hat jedoch keine Auswirkung auf die Vereinfachung. Die Zellenreduktion kommt zum Einsatz, wenn sehr starke Vereinfachungen erwünscht sind. Durch das Zusammenfassen einer großen Menge an Vertizes innerhalb eines Arbeitsschrittes ist eine effiziente Implementierung einer Vereinfachung umsetzbar.

Da dieses Verfahren keinerlei Rücksicht auf die Topologie nimmt, kann es auch bei nicht 2D-mannigfaltigen Netzen zur Anwendung kommen und ist nicht topologieerhaltend [LWC+02].

2.3.6 Vertexeliminierung

Bei der Vertexeliminierung (engl. „vertex removal“) werden einzelne Vertizes eines Dreiecksnetzes bestimmt und anschließend entfernt. Das Entfernen einzelner Vertizes erzeugt Löcher in der Oberfläche. Nach jedem Löschvorgang kommt es zu einer Retriangulierung (der Generierung neuer Dreiecke, um Löcher im Dreiecksnetz zu schließen) des Dreiecksnetzes, um eine geschlossene Oberfläche wiederherzustellen. Die neu geschaffenen Dreiecke bestehen aus jenen Vertizes, die eine Kante zu dem gelöschten Punkt besitzen. Dieses Vorgehen ist in Abbildung 2.17 dargestellt. Der Vertex in der Mitte des grau markierten Bereichs soll gelöscht werden. In der mittleren Abbildung markiert das graue Polygon die fehlende Triangulierung der Oberfläche. Auf der rechten Seite wurde dieser Bereich retrianguliert und die Oberfläche wieder geschlossen [SZL92].

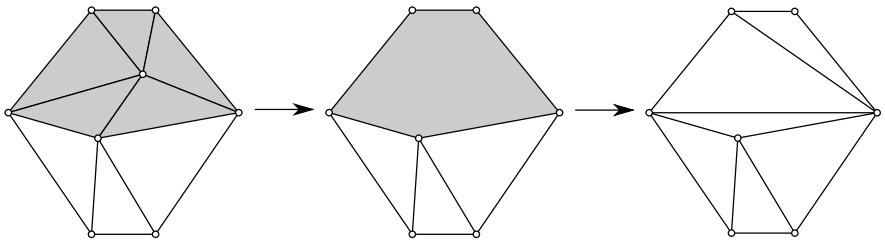

Abbildung 2.17: Vertexeliminierung mit Retriangulierung des Loches [LWC+02]

Vertexeliminierung basiert auf dem Ansatz, jeder einzelnen Löschoperation umgehend eine Retriangulierung folgen zu lassen. Das Entfernen von multiplen Vertizes bzw. das Verschließen der daraus resultierenden Löcher ist bei einfacher Vertexeliminierung nicht vorgesehen. Dies hat den Vorteil, dass es einfacher ist, die Auswirkung des Löschens eines Vertex zu bestimmen und die Komplexität der Retriangulierung wird reduziert [SZL92].

Es existieren mehrere Verfahren, um diese Löcher zu füllen. Eines davon ist die eingeschränkte Kantenreduktion. Es ist also möglich, die Vertexeliminierung als Spezialfall der Kantenreduktion anzusehen. Tatsächlich steht jedoch für sich ergebende Löcher eine Vielzahl möglicher Triangulierungen zur Verfügung. Daher wird die Vertexeliminierung separat behandelt und ist ein von der eingeschränkten Kantenreduktion getrennter Operator. Eine einfache eingeschränkte Kantenreduktion hat den Nachteil, nur beschränkte Freiheit bei der Retriangulierung zu bieten. Andere Verfahren können somit mehr Rücksicht auf die Form der Oberfläche nehmen und bessere Resultate bei den entstandenen Oberflächen liefern [Kle97].

Bei der Vertexeliminierung werden die neu erzeugten Dreiecke aus allen jenen Vertizes gebildet, die eine Kante zu dem gelöschten Punkt besitzen. Demnach gehört dieser Algorithmus zu den topologieerhaltenden Operationen.

2.3.7 Polygonverschmelzung

Polygonverschmelzung (engl. „polygon merging“) ist eine Technik, die ähnlich der Vertexeliminierung funktioniert. Bei letzterer wird ein einzelner Vertex gelöscht und das Loch verschlossen. Polygonverschmelzung erweitert diesen Ansatz, indem sie das Entfernen von mehreren Vertizes auf einmal zulässt. Die Idee hinter der Polygonverschmelzung basiert darauf, dass gemeinsam mit einem Vertex auch eine Reihe von Dreiecken entfernt wird, die mit dem Vertex gebildet wurde. Polygonverschmelzung fasst alle diese Dreiecke zu einem einzelnen Polygon zusammen. Dieses resultierende Polygon hat mehr Eckpunkte als Dreiecke, die für die Darstellung zum Einsatz kommen [HH93].

Werden mehrere Vertizes gelöscht, so verschmelzen alle zusammenhängenden Polygone zu einem einzigen. Vertexeliminierung würde hier nur das Entfernen einzelner Vertizes erlauben. Sobald die Vertizes eliminiert und damit die größeren Polygone gebildet wurden, findet eine Retriangulierung statt. Hierbei wird jedes dieser Polygone betrachtet und trianguliert, so dass sich die vereinfachte Oberfläche ergibt.

Polygonverschmelzung hat gegenüber Vertexeliminierung den Vorteil, dass eine verbesserte retriangulierte Oberfläche entstehen kann. Das Verschließen mit Hilfe eines Polygons gestattet einen höheren Anpassungsgrad der neuen Ausprägung an die Form des Objekts. Bei der Vertexeliminierung ist dies nur innerhalb jenes sehr begrenzten Bereichs möglich, der nach dem Entfernen eines einzelnen Vertex geschlossen wird [LWC+02].

Der Nachteil dieses Verfahrens liegt darin, dass sich die Retriangulierung eines komplexen Polygons wesentlich aufwändiger gestaltet als das Schließen der Löcher bei der Vertexeliminierung.

2.3.8 Vergleich der präsentierten Vereinfachungsoperatoren

Tabelle 2.1 gibt eine Übersicht über die präsentierten Vereinfachungsoperatoren mit ihren wichtigsten Eigenschaften. Neben dem Namen des Operators wird angeführt, ob dieser topologieerhaltend ist. Diese Eigenschaft ist ein wichtiges Kriterium für den Grad der maximalen Vereinfachung. Des Weiteren ist aufgelistet, welche Elemente eines Dreiecksnetzes durch die Operatoren entfernt werden, was Einfluss auf die Qualität der Vereinfachung und die Effizienz der Implementierung haben kann. Die letzten beiden Spalten nennen die Quelle, in der ein Operator vorgestellt wird, in der weitere Informationen gefunden werden können und - sofern vorhanden - die Abbildung, in der ein Beispiel für die Ausführung gezeigt wird.

Name	Topologie-erhaltend	Entfernt	Quelle	Abbildung
Kantenreduktion	Ja	Kante	Hoppe et al. [HDD+93]	2.11
Eingeschränkte Kantenreduktion	Ja	Kante	Hoppe et al. [HDD+93]	2.14
Vertexpaar-verschmelzung	Nein	Vertex	Garland und Heckbert [GH97]	2.15
Dreiecksreduktion	Ja	Dreieck	Zhieng et al. [ZJK01]	
Zellenreduktion	Nein	Vertizes	Rossignac und Borrell [RB92]	2.16
Vertexeliminierung	Ja	Vertex	Schroeder et al. [SZL92]	2.17
Polygon-verschmelzung	Ja	Dreiecke	Hinker et al. [HH93]	

Tabelle 2.1: Vergleich der präsentierten Vereinfachungsoperatoren

Die Vereinfachungsoperatoren bilden die Grundlage von Verfahren für die Vereinfachung von Dreiecksnetzen. Im nächsten Abschnitt wird eine Reihe von Vereinfachungsverfahren präsentiert.

2.4 Vereinfachungsverfahren

Die hier vorgestellten Operatoren dienen dazu, Teile von Dreiecksnetzen zu vereinfachen. Um eine vollständige Vereinfachung zu ermitteln, benötigt man zusätzliche Verfahren.

Im Anschluss werden einige weit verbreitete Verfahren und Algorithmen beschrieben. Aufgrund des Umfangs der Forschung auf diesem Gebiet wird kein Anspruch auf Vollständigkeit erhoben. Es wurden in der Fachliteratur häufig erwähnte Verfahren gewählt sowie neuere Forschungsergebnisse beschrieben, deren Ansatz sich mit Beschleunigung der Vereinfachung durch Parallelisierung der Berechnung beschäftigt.

Die hier vorgestellten Verfahren greifen auf die präsentierten Vereinfachungsoperationen zurück und kombinieren sie mit einer Zahl verschiedener Fehlermetriken und Bedingungen, um eine Vereinfachung zu berechnen. Die Ergebnisse solcher Verfahren unterscheiden sich einerseits in Qualität der Berechnung, andererseits in Effizienz und benötigtem Rechenaufwand.

Die vorgestellten Verfahren sind:

- **1. Schrittweise Dreiecksnetze** Vereinfachung aufgrund einer Liste von vorberechneten Kantenreduktionen.

- **2. Quadrik-Fehlermetrik** Optimierung der Vereinfachung mittels Vertexpaarverschmelzung durch Berechnung einer optimalen Ersatzposition.

- **3. Vertexgruppierung** Beschreibung von Verfahren der Vereinfachung basierend auf Zellenreduktion.

- **4. Vertexdezimierung** Vereinfachungsverfahren basierend auf Vertexeliminierung.

- **5. Bildbasierte Vereinfachung** Verfahren zur Reduktion der Auswirkungen von Vereinfachungsoperationen mittels Darstellung von Rasterbildern.

- **6. Parallele Kantenreduktion durch individuelle Bereiche** Paralleles Verfahren zur parallelen Ausführung mehrerer Kantenreduktionen.

- **7. Parallele Kantenreduktion durch Sperren von Bereichen** Paralleles Verfahren, das einen Sperrmechanismus auf eine von einer Kantenreduktion beeinflussten Kante anwendet.

- **8. Vereinfachung mittels Morton Integralen** Paralleles Verfahren, das Vertizes sortiert, in einem Baum organisiert und nach einem maximalen Fehlerwert zusammenfasst.

- **9. Vereinfachung durch Reduktion von Teilbäumen** Vereinfachung mit parallel ausgeführten eingeschränkten Kantenreduktionen, die in einer Menge von Bäumen repräsentiert werden.

- **10. Vergleich der präsentierten Verfahren** Bietet eine Gegenüberstellung der wichtigsten Eigenschaften der präsentierten Verfahren.

2.4.1 Schrittweise Dreiecksnetze

Der Algorithmus der schrittweisen Dreiecksnetze (engl. „progressive meshes“, abgekürzt PM) wurde erstmals vorgestellt von Hoppe [Hop96], basierend auf den Ansätzen aus [HDD+93] und vom Autor mehrmals weiterentwickelt.

Schrittweise Dreiecksnetze beruhen auf der Idee, dass ein Dreiecksnetz nicht direkt

gespeichert, sondern durch eine vereinfachte Version und eine Reihe von Verfeinerungsoperationen dargestellt wird. Ein beliebiges Netz M ist repräsentiert als eine gröbere Version M^0 und eine Reihe von n Verfeinerungen [Hop98a]. Jede dieser Operationen enthält Informationen über eine Vertexteilung. Ein schrittweises Dreiecksnetz ist also eine Sequenz aus Dreiecksnetzen $M^0, M^1, ..., M^n$ mit steigender Vertexzahl bzw. steigendem Detailgrad [Hop96].

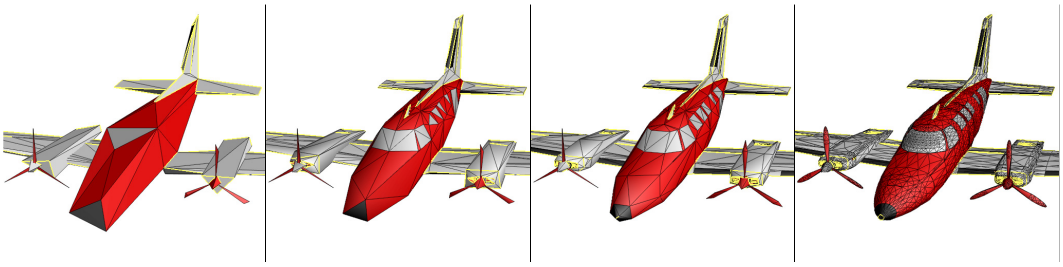

Abbildung 2.18: Beispiel von schrittweisen Dreiecksnetzen [Hop96]

Abbildung 2.18 zeigt ein Beispiel für mehrere Repräsentationen desselben Objekts. Auf der linken Seite ist eine sehr grobe Repräsentation (M^0) zu sehen. Hier sind nur wenige Dreiecke vorhanden, wodurch der Rumpf des Flugzeuges sehr kantig wird. Nach rechts werden zusätzliche Dreiecke in das Netz eingefügt. Die Darstellung außen rechts (M^n) zeigt das am feinsten aufgelöste Dreiecksnetz. Hier ist die runde Form des Rumpfes am besten wiedergegeben.

Der Vereinfachungsoperator Kantenreduktion und die inverse Form Vertexteilung wurden in Unterabschnitt 2.3.1 beschrieben. Gemäß der Definition in [Hop96] lässt sich eine vereinfachte Version des Dreiecksnetzes durch eine Reihe von Kantenreduktionen („ecol“) erstellen:

$$(M = M^n) \xrightarrow{ecol_{n-1}} ... \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 \tag{2.1}$$

Jede Kantenreduktion resultiert in einer vereinfachten Version des originalen Dreiecksnetzes M, wobei der Grad der Vereinfachung von der Anzahl der durchgeführten Kantenreduktionen abhängig ist. Diese Operationen sind durch eine Vertexteilung („vsplit“) umkehrbar. Im Gegenzug zu 2.1 lässt sich ein Dreiecksnetz M also ausdrücken durch:

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} ... \xrightarrow{vsplit_{n-1}} (M^n = M) \tag{2.2}$$

Mit Hilfe der Kantenreduktion und der Vertexteilung lassen sich eine Reihe von Auflösungsstufen eines Dreiecksnetzes darstellen. Mittels fließender Interpolation (siehe

Unterabschnitt 2.2.1) wird ein gleichmäßiger Übergang zwischen den Vereinfachungsstufen erreicht [Hop98b].

Für eine fließende Interpolation M^G zweier Auflösungsstufen des Dreiecksnetzes M^i und M^{i+1} wird ein Faktor $0 \leq \alpha \leq 1$ definiert, so dass gilt $M^G(0)$ entspricht M^i und $M^G(1)$ ist M^{i+1}. Die zwischen M^i und M^{i+1} durchgeführten Kantenreduktionen bzw. Vertexteilungen werden anhand dieses Faktors interpoliert, und damit entsteht ein stufenloser Verlauf anstatt einer sprunghaften Vereinfachung.

Neben den multiplen Auflösungsstufen und der fließenden Interpolation bietet dieser Algorithmus einen weiteren Vorteil: Man kann die zwischen den einzelnen Stufen durchgeführten Operationen auswählen und nur jene vollziehen, die wirklich benötigt werden. Dieses als selektive Verfeinerung bekannte Verfahren überprüft vor einer Vertexteilung, ob der Vertex innerhalb der Blickpyramide der Kamera liegt und ob das Resultat der Operation von der Position des Betrachters aus sichtbar ist. Somit ist die Anzahl der zu berechnenden Operationen reduzierbar (siehe auch Unterabschnitt 2.2.2 und Abbildung 2.7) [Hop97].

Für die Berechnung der Vereinfachung werden sämtliche Vereinfachungsschritte zwischen dem originalen Dreiecksnetz und der maximal vereinfachten Version bestimmt, anhand einer Funktion bewertet und jeder Operation wird ein Fehlerwert zugewiesen. Mit Hilfe dieser Abweichung werden alle gelisteten Schritte gereiht und schwerwiegendere Operationen später vollzogen. Eine Vereinfachung erfolgt durch die iterative Ausführung der gespeicherten Vereinfachungen. Ist die Liste abgearbeitet, so liegt das maximal vereinfachte Dreiecksnetz vor.

In [Hop97] wurde dieses Verfahren erweitert. Anstatt einer Liste der gespeicherten Manipulationen kommt eine Menge von Baumstrukturen zum Einsatz. Alle Wurzelknoten dieser Bäume stehen für einen Vertex des maximal vereinfachten Dreiecksnetzes. Jeder Baum enthält eine Menge an Vertexteilungen, die ausgehend von diesem Knoten vollzogen werden kann. Um festzustellen, wann die entsprechende Operation erfolgen darf, ist es nötig, die gespeicherten Daten so zu erweitern, dass auch alle beteiligten Dreiecke bekannt sind. Um diesen Algorithmus zu berechnen, muss jeder Knoten die Information besitzen, welche Vertizes an der Operation beteiligt sind und welche Dreiecke erzeugt bzw. manipuliert werden. Auf diese Weise definiert man, wann es möglich ist, eine Vertexteilung bzw. eine Kantenreduktion durchzuführen.

Für eine Vertexteilung gilt:

- Der Ausgangsvertex der Operation muss im gegenwärtigen Netz existieren.
- Alle referenzierten Dreiecke müssen im gegenwärtigen Netz existieren.

Für die Umkehrung (Kantenreduktion) gilt:

- Die beiden Endpunkte der zu entfernenden Kante müssen vorliegen.

- Die in der Operation referenzierten Dreiecke müssen vorliegen.

Aufgrund dieser Baumstruktur ist es möglich, gezielte Operationen auf einfachen Punkten oder Kanten durchzuführen. Eine effiziente Behandlung von selektiven Vereinfachungen bzw. deren Umkehrung wird realisierbar. Soll eine Operation auf einem bestehenden Vertex abgearbeitet werden, so erlauben es die zusätzlich gespeicherten Abhängigkeiten, alle benötigten Vertizes zu bestimmen. Existieren geforderte Vertizes oder Dreiecke nicht, so werden anhand der gespeicherten Bäume die Operationen zurückverfolgt, bis ein im Baum höher liegender Knoten gefunden wird, dessen Abhängigkeiten im gegenwärtigen Dreiecksnetz vorhanden sind. Die benötigten Vertexteilungen bzw. Kantenreduktionen werden ausgeführt, um alle Abhängigkeiten der Operation zu erfüllen. Erst dann kann sie ausgeführt werden, ohne dass die Hierarchie der gespeicherten Operationen verletzt wird.

Zusätzlich wird in [Hop97] die Auswahl der Operationen verbessert, um sie unter Berücksichtigung der Position und Blickrichtung des Betrachters durchzuführen: Es wird argumentiert, dass Bereiche eines Dreiecksnetzes außerhalb der Blickpyramide über keinen hohen Detailgrad verfügen müssen, da sie nicht dargestellt werden. Dabei ist darauf zu achten, dass sich nicht nur aktuell im Dreiecksnetz enthaltene Dreiecke außerhalb der Blickpyramide befinden, sondern auch jene, die durch mögliche Vertexteilungen entstehen können. Es wird auf die Baumstruktur zurückgegriffen und für jeden Vertex V_i eine Begrenzungssphäre ermittelt. Für die Blattknoten wird ihre Größe so gewählt, dass alle Nachbarn des im Knoten repräsentierten Vertex darin enthalten sind. Bei Vaterknoten wird eine Sphäre berechnet, welche die Sphären der Kindknoten beinhaltet. Ein Vertex kann bei Vertexteilungen ignoriert werden, wenn sich seine Begrenzungssphäre vollständig außerhalb der Blickpyramide befindet.

Ein ähnliches Verfahren kommt zum Einsatz, um dem Betrachter abgewandte Bereiche von Vertexteilungen auszunehmen. Für jeden Vertex V wird ein Kegel gespeichert (bestimmt durch die Position des Vertex v, den Normalvektor n_v des Vertex und den halben Öffnungswinkel α_v des Kegels), der als Begrenzungsvolumen für die Normalvektoren der Oberfläche im Bereich von V dient. Für eine Position des Betrachters e kann ein Vertex von Vertexteilungen ausgenommen werden, wenn beide, folgenden Bedingungen gelten:

$$\begin{aligned} (v-e) \bullet n_v &> 0 \\ ((v-e) \bullet n_v)^2 &> |v-e|^2 sin^2 \alpha_v \end{aligned} \tag{2.3}$$

Für jeden Vertex wird die Abweichung zwischen den Oberflächen vor und nach einer Vertexteilung vorberechnet und abgespeichert. Zur Laufzeit wird die Abweichung mit Hilfe des Winkels zwischen Blickvektor und Normalvektor des Vertex skaliert und mit einer Toleranzgrenze verglichen. Ein Vertex darf geteilt werden, wenn er sich innerhalb

der Blickpyramide befindet, dem Betrachter nicht abgewandt ist und die Abweichung der Oberfläche eine definierte Toleranzgrenze überschreitet.

In [HSH09] wurde eine parallele Implementierung aus [Hop97] vorgestellt, die eine weitere Beschleunigung des Verfahrens erreicht. Auch hier findet keine Bestimmung der Kantenreduktionen zur Laufzeit statt. Der modifizierte Ansatz ermöglicht die parallele Durchführung der in der Baumstruktur gespeicherten, vorberechneten Operationen.

Wird in [Hop97] eine Operation aus dem Baum gewählt, so müssen unter Umständen zuerst sämtliche in Vaterknoten gespeicherten Operationen durchgeführt werden, so dass alle für die Vereinfachung benötigten Vertizes und Dreiecke vorhanden sind. Die Autoren von [HSH09] identifizieren dieses Vorgehen als limitierenden Faktor für eine parallele Ausführung, da die Abarbeitung der Hierarchie keine bzw. eine nur sehr beschränkte Parallelität zulässt.

In [HSH09] wird die Abarbeitung der Hierarchie durch eine kaskadierende Aktualisierung ersetzt. Der Ansatz wählt eine Operation aus der Baumstruktur und führt diese durch, ohne auf die gespeicherten Voraussetzungen zu achten. Somit kann es temporär zu Faltungen oder einer inkonsistenten Topologie kommen. Erst im Anschluss werden schrittweise die benachbarten Vertizes anhand der in den Vaterknoten gespeicherten Operationen bearbeitet. Laut des Ansatzes in [HSH09] kann es mehrere Darstellungsvorgänge dauern, bis alle Abhängigkeiten abgearbeitet wurden und die Netzintegrität wieder garantiert ist.

In [DG12] wird ein Ansatz beschrieben, der die parallele Bearbeitung von schrittweisen Dreiecken weiter beschleunigt. Bei [HSH09] sind Abhängigkeiten zwischen Operationen vorhanden, die im Laufe mehrerer Arbeitsschritte erfüllt werden. Im Gegensatz dazu wird bei [DG12] eine Menge von Operationen auf benachbarten Vertizes ausgeführt. Abhängigkeiten werden vermieden und nur ein einzelner Arbeitsschritt ist erforderlich. Hierfür kommt eine Datenstruktur zum Einsatz, die für jede Operation die beteiligten Dreiecke speichert und es ermöglicht, effizient zu bestimmen, welche Vertizes gegenwärtig im Dreiecksnetz vorhanden sind. Durch diese Eigenschaft kann eine Vertexteilung oder Kantenreduktion auf einem Vertex ausgeführt werden, ohne Nachbarvertizes in einem nachfolgenden Arbeitsschritt zu manipulieren, um Abhängigkeiten aufzulösen. Für den Fehlerwert eines Vertex kommt das Maximum der Fehler aller benachbarten Vertizes zum Einsatz. Damit wird sichergestellt, dass sich benachbarte Vertizes in derselben Auflösungsstufe befinden und Fehler im Dreiecksnetz werden vermieden.

Die Autoren geben an, dass Faltungen und inkonsistente Topologie nicht auftreten können, wenn der Detailgrad für das gesamte Objekt anhand einer einzelnen Fehlertoleranz bestimmt wird. Bei blickwinkelabhängigen Einsatzbereichen wird diese Bedingung nicht erfüllt. Es wird jedoch davon ausgegangen, dass eine Behandlung von Faltungen nicht erforderlich ist, da sich die erlaubten Fehlertoleranzen nicht sprunghaft ändern, sondern fließend verlaufen und somit zwischen benachbarten Vertizes nur gering abweichen.

Die Autoren von [DGG14] beschäftigen sich mit dem Problem von Manipulationen eines Dreiecksnetzes (Transformationen) bei schrittweisen Dreiecksnetzen. Bei diesem Ansatz kommen die Vertexteilungen des schrittweisen Dreiecksnetzes zum Einsatz, um Transformationen der Geometrie, die auf einer groben Auflösungsstufe angewandt werden, weiterzugeben und bei der Rekonstruktion von Auflösungsstufen mit vielen Dreiecken zu berücksichtigen.

Als Basis dient die Datenstruktur aus [DG12], um die Vertexteilungen bzw. Kantenreduktionen zu speichern. Bei schrittweisen Dreiecksnetzen werden für eine Vertexteilung die Attribute (Position, Normalvektor, ...) der erzeugten Vertizes gespeichert. Die Autoren von [DGG14] argumentieren, dass eine Repräsentation dieser Daten im lokalen Koordinatensystem des zu teilenden Vertex häufig verwendet wird, in diesem Fall jedoch nicht sinnvoll ist. Jede Transformation eines Vertex bei einer Vertexteilung wird an die beiden erzeugten Vertizes weitergegeben. Um bei Vertexteilungen von benachbarten Vertizes eine gleichmäßigere Weitergabe der Transformationen zu ermöglichen, wird im Falle von [DGG14] das lokale Koordinatensystem mittels gewichtetem Durchschnitt der Daten aller benachbarten Vertizes aufgebaut. Die Attribute der Vertizes, die durch eine Vertexteilung erzeugt werden, sind darin repräsentiert. Findet eine Transformation auf einer groben Auflösungsstufe statt, so ist das lokale Koordinatensystem für diese Vertizes neu zu berechnen. Durch die Vertexteilungen werden diese Änderungen dann auf den feineren Auflösungsstufen angewandt.

Mit Hilfe dieses Ansatzes ist es auch möglich, Transformationen auf einem hoch aufgelösten schrittweisen Dreiecksnetz an eine grobe Auflösungsstufe weiterzugeben. Hierfür müssen die Kantenreduktionen für Kanten, die modifizierte Vertizes enthalten, neu berechnet und die Vertexattribute in den modifizierten lokalen Koordinatensystemen repräsentiert werden.

Neben der Beschleunigung der Berechnung wurde auch die Fehlermetrik für die Auswahl der Kantenreduktionen weiterentwickelt. Einen weit verbreiteten Ansatz, um die Auswirkungen einer Vereinfachungsoperation zu ermitteln, stellt das Bestimmen der lokalen Unterschiede, die durch eine Vereinfachung verursacht werden, dar.

Um die Selektion von Kantenreduktionen zu verbessern, wird in [WWLH11] argumentiert, dass nicht nur die von einer Vereinfachungsoperation verursachten Veränderungen bewertet werden sollen, sondern auch die geometrischen Charakteristika der Oberfläche des originalen Netzes in die Metrik mit einfließen sollen. Es wird ein Ansatz präsentiert, der versucht, geometrische Eigenschaften (z. B. Ecken und Kanten des Objekts) zu erkennen und beizubehalten. Aus einem Vertex und seinen Nachbarn wird eine durchschnittliche Krümmung der Oberfläche berechnet. Aus dieser ermitteln die Autoren eine Reihe von markanten Punkten, die genutzt werden, um Formen des Objekts wiederzugeben. Ziel dieses Algorithmus ist eine Vereinfachung, die markante geometrische Eigenschaften des originalen Netzes beibehält und so die visuelle Qualität verbessert.

Auch in [LCW+14] wird argumentiert, dass der Erhalt markanter Oberflächeneigenschaften einen wichtigen Faktor für die Qualität einer Vereinfachung darstellt. Die Autoren versuchen, die Metrik für die Auswahl der Kantenreduktionen zu verbessern, greifen hierzu ebenfalls auf die Krümmung der Oberfläche zurück und berechnen diese für einen Vertex und die benachbarten Vertizes.

Hier ziehen die Autoren die Veränderung der Krümmung der Oberfläche durch eine Vereinfachungsoperation in Betracht. Mit Hilfe eines Vertex und seiner Nachbarn wird eine durchschnittliche Krümmung der Oberfläche in diesem Bereich bestimmt und diese mit der durchschnittlichen Krümmung, die nach einer Vereinfachungsoperation vorliegen würde, verglichen. Diese Abweichung fließt in die Fehlermetrik mit ein, um so geometrische Ausprägungen der Oberfläche besser zu erhalten.

Das nächste vorgestellte Verfahren für die Vereinfachung von Dreiecksnetzen ist die Quadrik-Fehlermetrik.

2.4.2 Quadrik-Fehlermetrik

Die Quadrik-Fehlermetrik (engl. „quadric error metric“, abgekürzt QEM) ist ein Verfahren, das von Garland und Heckbert in [GH97] vorgestellt wurde. Sie führen die Vereinfachung mit Hilfe einer Reihe von Vertexpaarverschmelzungen aus. Die Besonderheit dieses Algorithmus liegt in der Selektion der zu entfernenden Kanten und der Bestimmung der Ersatzpositionen.

Für dieses Verfahren ist jedes Vertexpaar v_1, v_2 eines Dreiecksnetzes ein Kandidat für eine Vertexpaarverschmelzung, sofern:

- v_1, v_2 eine Kante bildet.
- $|v_1 - v_2| < t$, wobei t ein Grenzwert ist.

Für die Vereinfachung mittels Quadrik-Fehlermetrik soll iterativ eine Verschmelzung ausgewählt und durchgeführt werden, bis der gewünschte Grad der Vereinfachung erreicht ist. Für jede zur Wahl stehende Reduktion werden die Kosten der Vereinfachung ermittelt. Hierfür wird eine 4x4 Matrix Q für jeden Vertex erstellt. Für eine mögliche Reduktion $v_1, v_2 \rightarrow v$ wird eine kombinierte Matrix $Q = Q_1 + Q_2$ gebildet.

Um eine Vereinfachungsoperation zu bewerten, muss auch die Ersatzposition bekannt sein.

Der Algorithmus besteht somit aus folgenden Schritten:

1. Berechnung der Matrizen Q für alle Vertizes

2. Bestimmung aller möglichen Vertexpaare für Verschmelzungen

3. Berechnung der bestmöglichen Ersatzposition für alle Vertexpaare

4. Sortierung aller Vertexpaare nach den Kosten

5. Iterative Durchführung der Verschmelzungen mit den geringsten Kosten und Aktualisierung der Beurteilung aller beteiligten Vertizes.

Zunächst wird die Matrix für jeden Vertex kalkuliert. Es wird davon ausgegangen, dass jeder Vertex der Schnittpunkt einer Reihe von Ebenen ist. Diese Ebenen werden durch die Dreiecke, die in den Vertex einfallen, bestimmt. Der Fehler eines Punktes in Bezug auf die originale Oberfläche ist die Summe der quadrierten Normalabstände zu diesen Ebenen. Der Fehlerwert $\triangle(v)$ wird berechnet durch:

$$\triangle(v) = \sum_{p \in Ebenen(v)} (p^T v)^2 \tag{2.4}$$

Der Faktor $p = [a, b, c, d]^T$ ist definiert durch die Ebenengleichung $ax+by+cz+d = 0$, wobei gilt $a^2 + b^2 + c^2 = 1$. Diese Fehlermetrik kann umgeschrieben werden:

$$\begin{aligned} \triangle(v) &= \sum_{p \in Ebenen(v)} (v^T p)(v p^T) \\ &= \sum_{p \in Ebenen(v)} v^T (p p^T) v \\ &= v^T (\sum_{p \in Ebenen(v)} K_p) v \end{aligned} \tag{2.5}$$

Wobei K_p folgende Matrix darstellt:

$$K_p = p p^T = \begin{pmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{pmatrix} \tag{2.6}$$

Durch die Aufsummierung der einzelnen Matrizen K_p ergibt sich für einen Vertex die Matrix Q. Die Ebenen sind durch die Dreiecke, die in einen Vertex einfallen, bestimmt. Somit hat jeder Vertex des originalen Netzes den Fehler 0. Das Zusammenfassen der einzelnen Ebenen in nur eine Matrix führt allerdings eine Ungenauigkeit in die Berechnung ein. Das nehmen die Autoren jedoch für den stark reduzierten Speicherbedarf bei der Vorberechnung in Kauf.

Um die Ersatzposition einer Verschmelzung der Vertizes V_1 und V_2 mit den Matrizen Q_1 und Q_2 zu finden, wird zuerst die Matrix für das Vertexpaar durch $Q = Q_1 + Q_2$ bestimmt. Im Anschluss daran sucht man eine Ersatzposition v, die den Fehler $\triangle(v)$ minimiert. Sie wird durch $\partial \triangle / \partial x = \partial \triangle / \partial y = \partial \triangle / \partial z = 0$ ermittelt.

Um die Auswirkung einer Verschmelzung zu bewerten, vergleicht man die Oberflächen vor und nach der Manipulation. Hierzu wird auf den Oberflächen eine Reihe von Abtastpunkten definiert und für jeden dieser der Minimalabstand zu einem Dreieck auf der anderen Oberfläche bestimmt. Daraus berechnet sich der Fehlerwert, anhand dessen die Vereinfachungsoperationen gereiht werden. So können iterativ die Manipulationen mit den geringsten Auswirkungen durchgeführt werden, bis die geforderte Vereinfachung erreicht ist.

In [JLG14] wird eine Weiterentwicklung des Ansatzes von Garland und Heckbert präsentiert mit dem Ziel, die Qualität der Vereinfachung zu verbessern. Dafür wurde die Selektion der zu entfernenden Vertizes modifiziert. Die Autoren haben drei Faktoren bestimmt, die sie als wichtig für die Qualität der Vereinfachung erachten und in die Berechnung einfließen lassen:

- **Lokale Unebenheit** Für einen Vertex v_i mit Normalvektor n_i werden alle Dreiecke gesucht, die v_i enthalten. Für jedes Dreieck wird die Abweichung zwischen Dreiecksnormalen und n_i ermittelt. Die durchschnittliche Abweichung aller gefundenen Dreiecke bildet den ersten Faktor. Je größer der Durchschnitt dieser Abweichungen, umso bedeutender wird der Vertex eingestuft.

- **Dreiecksgröße** Die Autoren gehen davon aus, dass eine große Menge kleiner Dreiecke dazu dient, Details auf der Oberfläche auszubilden. Analog zur Berechnung der lokalen Unebenheit werden alle Dreiecke gesucht, die einen Vertex v_i enthalten, und ihre Flächen berechnet. Der Durchschnitt der Flächen kommt als zweiter Faktor zum Tragen.

- **Kantenlänge** Es wird die Länge von Kanten, die entfernt werden sollen, berücksichtigt. Die Autoren definieren, dass das Entfernen einer Kante mit großer Länge einen stärkeren Einfluss auf die Vereinfachung hat, als das Entfernen einer kurzen Kante.

Während Garland und Heckbert versuchen, eine optimale Ersatzposition zu ermitteln, wird in [JLG14] argumentiert, dass nur Vertizes des originalen Netzes zwingend auf der Oberfläche des Objekts liegen und folglich die beste Repräsentation des Objekts darstellen. Es wird daher nicht wie bei Garland und Heckbert eine Vertexpaarverschmelzung ausgeführt, sondern auf die eingeschränkte Kantenreduktion zurückgegriffen.

In [WWZJ13] wird ebenfalls versucht, die Resultate der Vereinfachung mittels Quadrik-Fehlermetrik zu verbessern. Hierzu setzen die Autoren eine zweigeteilte Metrik ein: Zum einen sollen geometrische Ausprägungen bewertet, zum anderen visuelle Merkmale erkannt und berücksichtigt werden.

- **Geometrische Metrik** Der geometrische Teil dieser Metrik analysiert die Abweichungen der Normalvektoren zwischen einem Vertex und seinen Nachbarn ähnlich wie in [JLG14]. Allerdings wird in [WWZJ13] nicht auf die Abweichung zwischen dem Normalvektor eines Vertex und der umliegenden Dreiecke geachtet, sondern der Normalvektor des Vertex mit den Normalvektoren der benachbarten Vertizes verglichen.

- **Visuelle Metrik** Die visuellen Ausprägungen werden mit Hilfe einer Torsions-Metrik beurteilt. Sie ermittelt die Torsion aller Dreiecke, die mit einem Vertex gebildet werden. Es wird argumentiert, dass eine Vereinfachungsoperation, die kleine und flache Dreiecke entfernt, gegenüber dem Löschen von großen Dreiecken mit hoher Torsion bevorzugt werden soll.

Ein weiterer Ansatz wird in [SLA15] beschrieben. Die Idee von Garland und Heckbert wird aufgegriffen, [SLA15] beschäftigt sich aber vor allem mit sehr starken Vereinfachungen. Die Autoren argumentieren, dass bei diesem Anwendungsfall der Erhalt der groben Formgebung bzw. Struktur eines Objekts im Vordergrund steht und kleinere geometrische Details in den Hintergrund treten.

Es wird davon ausgegangen, dass ein Objekt aus einer Menge von annähernd planaren Bereichen besteht, die erkannt werden können. Ein Vorberechnungsschritt wird eingeführt, der eine Menge von „Planaren Stellvertretern“ bestimmt, um die annähernd planaren Bereiche zu repräsentieren. Ein Stellvertreter besteht somit aus einer Ebene sowie einer Menge von dem Stellvertreter zugeordneten Vertizes des Dreiecksnetzes und dient dazu, die groben Strukturen des Objekts zu erhalten. Jedes Dreieck des Netzes wird einem oder mehreren Stellvertretern zugeordnet.

In weiterer Folge wird eine Reihe von Kantenreduktionen durchgeführt, wobei die Stellvertreter in die Fehlermetrik einfließen, um bei starken Vereinfachungen den Erhalt dieser Strukturen zu ermöglichen. Das Ergebnis der Vereinfachung hängt daher stark von der Erkennung der Stellvertreter und der Zuweisung der Dreiecke zu den Stellvertretern ab.

In [GDG11] wird der Ansatz von Garland und Heckbert aufgegriffen und für eine parallele Ausführung modifiziert, um eine Reduktion der Berechnungszeit zu erreichen. Als Ausgangspunkt wird ein Dreiecksnetz, das vereinfacht werden soll bis der erzeugte Fehler eine Toleranzgrenze überschreitet, angenommen. In der iterativen Variante von Garland und Heckbert würden hierfür so lange einzelne Kantenreduktionen aus-

geführt, bis keine Vereinfachungsoperation mehr möglich ist, ohne die Toleranzgrenze zu überschreiten. Die Autoren von [GDG11] geben an, dass bei Einsatz der Quadrik-Fehlermetrik der Fehler durch eine Kantenreduktion immer erhöht wird. Somit können alle Vereinfachungsoperationen, deren erzeugte Abweichung unterhalb der Toleranzgrenze liegt, gleichzeitig ausgeführt werden. Dabei muss jedoch Rücksicht auf Wettbewerbsbedingungen genommen werden.

In [GDG11] wird der minimale Fehler eines Vertex v_i definiert. Eine Menge an Kanten $E = \{e_1, e_2, ..., e_n\}$ wird mit v_i gebildet. Laut Garland und Heckbert kann für jede dieser Kanten ein Fehlerwert ermittelt werden. Die Autoren von [GDG11] bestimmen den Fehlerwert von v_i als das Minimum der Fehlerwerte aller $e_j \in E$. Alle jene Kanten, deren Endpunkte ein lokales Minimum des per-Vertex Fehlers beinhalten, können gleichzeitig durch eine Kantenreduktion entfernt werden.

Für die Vereinfachung werden vier Schritte ausgeführt:

1. Berechnen der per-Vertex Quadriken.

2. Finden der Ersatzposition und des Fehlerwertes für jede Kante.

3. Finden der lokalen Minima der Fehlerwerte und paralleles Ausführen der entsprechenden Kantenreduktionen.

4. Erneute Iteration (beginnend mit Schritt 2) bis keine weiteren Kantenreduktionen ausgeführt werden können, ohne die Toleranzgrenze für den Fehlerwert zu überschreiten.

Gegenüber der iterativen Variante von Garland und Heckbert kann mit dieser Parallelisierung eine starke Reduktion der Laufzeit erzielt werden (die Autoren geben eine Beschleunigung bis zu Faktor 43 gegenüber dem Verfahren von Garland und Heckbert an). Es ist anzumerken, dass bei dieser Berechnung keine Rücksicht auf etwaige Faltungen oder inkonsistente Topologie genommen wird.

Der Ansatz in [OKK15] beschäftigt sich mit der Vereinfachung von sehr großen Modellen, bei denen viele Verfahren aufgrund des hohen Speicherbedarfs nicht mehr angewandt werden können. Stattdessen werden in solchen Fällen Teilbereiche eines Dreiecksnetzes geladen und bearbeitet (engl. „out of core“). [OKK15] beschreibt einen Ansatz, der - basierend auf [GDG11] - die Vereinfachung von sehr großen Modellen beschleunigt. Zuerst wird ein Dreiecksnetz in eine Reihe von Teilbereichen zerlegt. Jeder dieser Teilbereiche kann individuell vereinfacht werden. Im Anschluss müssen die vereinfachten Bereiche wieder zusammengesetzt werden. Da jeder Teilbereich unabhängig vereinfacht wird, ist sicherzustellen, dass entlang der Grenze zweier Teilbereiche dieselben Ver-

einfachungsoperationen zum Einsatz kommen, um ein fehlerloses Zusammenfügen der Geometrie zu gewährleisten.

In [GDG11] werden lokale Minima des Fehlerwertes gesucht, um eine parallele Vereinfachung ohne Wettbewerbsbedingungen zu ermöglichen. Dieser Ansatz wird in [OKK15] genutzt, um die einheitliche Vereinfachung entlang der Grenzen von Teilbereichen zu garantieren: Für die beiden Endpunkte einer Kante gibt es einen inneren (1-Ring) und einen äußeren (2-Ring) Ring an Vertizes. Abbildung 2.19 zeigt ein Beispiel hierfür. Die gestrichelte Kante wird betrachtet. Hellgraue Dreiecke sind aus den Endpunkten der betrachteten Kante und ihren Nachbarn (innerer Ring) gebildet. Dunkelgraue Dreiecke beinhalten Vertizes des inneren und äußeren Ringes.

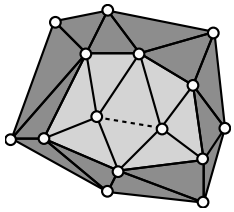

Abbildung 2.19: Innerer und äußerer Ring von Vertizes

Die Autoren von [OKK15] argumentieren, dass bei Einsatz einer gewissen Toleranzgrenze immer dieselben Kanten entfernt werden, sofern für alle Kanten die Vertizes des inneren und des äußeren Ringes bekannt sind. Werden bei Vereinfachung eines Teilbereichs für jede im Bereich enthaltene Kante der innere und äußere Ring berücksichtigt, so wird die Geometrie entlang der Grenzen zweier Teilbereiche ident vereinfacht. Beim Zusammensetzen der Teilbereiche kommt es zu keinen Fehlern in der Geometrie.

Um die Vereinfachung zu berechnen, werden in [OKK15] folgende Schritte ausgeführt:

1. Zerlegung eines Dreiecksnetzes in Teilbereiche.

2. Erweiterung der Teilbereiche, um für alle Kanten den inneren und äußeren Ring zu umfassen.

3. Vereinfachung der Teilbereiche mit dem Verfahren aus [GDG11].

4. Zusammensetzen der vereinfachten Teilbereiche.

Die Resultate in [OKK15] beschreiben eine sehr gute Laufzeit. Diese lässt sich auf die hohe Anzahl an gleichzeitigen Vereinfachungsoperationen durch die große Menge an

Dreiecken zurückführen. Die Autoren argumentieren jedoch, dass die Laufzeit von der Anzahl der gleichzeitig ausführbaren Kantenreduktionen abhängt und somit nur schwer vorhersagbar ist.

Eine Variante des Algorithmus von Garland und Heckbert, die Animationen berücksichtigt, wird in [MG03] vorgestellt. Durch die Transformation bei einer Animation kann es zu einer Deformierung des Dreiecksnetzes kommen, welche durch eine statische Fehlermetrik nicht repräsentiert wird. Der Algorithmus in [MG03] versucht, die Qualität der Vereinfachung bei animierten Objekten zu verbessern.

Zuerst wird aus der Animation eine Reihe von Beispielschritten erstellt, die das Netz in verschiedenen Phasen der Bewegung bzw. Deformierung repräsentiert. In weiterer Folge findet bei Mohr und Gleicher eine Aufsummierung der Fehlermetrik über alle diese Phasen der Animation statt. Während der Ansatz bei Garland und Heckbert die Kosten einer Vereinfachungsoperation für ein Dreiecksnetz errechnet, soll das Vorgehen bei Mohr und Gleicher die Kosten während mehrerer Phasen der Animation zusammenfassen und so zu einer verbesserten Vereinfachung führen. Die Berücksichtigung der Animationsschritte kann sich auch nachteilig auf die Qualität der Vereinfachung auswirken. Da mehrere Animationsphasen berücksichtigt werden und ein durchschnittlicher Fehler gebildet wird, ist die Metrik für jeden einzelnen Animationsschritt nicht optimal.

In [Tse14] wird argumentiert, dass der Ansatz in [MG03] Probleme mit homogenen Koordinaten nicht ausreichend berücksichtigt.

Hier bestimmt man ebenfalls eine Reihe von Beispielschritten der Animation. Der Autor zieht in Betracht, dass sich das lokale Koordinatensystem eines Vertex in den einzelnen Beispielschritten verändert. Aus diesem Grund transformiert der Autor einen Vertex, der in mehreren Beispielschritten existiert, in einen gemeinsamen lokalen Koordinatenraum bevor die Fehlerberechnung durchgeführt wird, um das Resultat der Vereinfachung zu verbessern.

Im Anschluss wird das Verfahren der Vertexgruppierung erläutert.

2.4.3 Vertexgruppierung

Das Verfahren der Vertexgruppierung (engl. „vertex clustering“) wurde erstmals 1992 in [RB92] vorgestellt. Die Grundidee basiert auf einem relativ simplen und leistungsfähigen Verfahren, um mehrere Vertizes zusammenzufassen, das auf der Zellenreduktion basiert (Unterabschnitt 2.3.5). Sie wurde seit der ersten Präsentation immer weiter entwickelt und im Zuge der Einführung der freieren Programmierbarkeit moderner Graphikhardware auch für eine effiziente Implementierung auf GPUs angepasst ([DT07]).

Der ursprüngliche Algorithmus aus [RB92] beruht darauf, jeden Vertex eines Dreiecksnetzes zu analysieren und ihm ein Gewicht zuzuweisen. Im Anschluss daran wird ein regelmäßiges 3-dimensionales Gitter über das Objekt gelegt. Jede Zelle davon steht für einen einzelnen Vertex des vereinfachten Netzes. In der ursprünglichen Version dieses Algorithmus werden alle Punkte innerhalb einer Zelle durch jenen Vertex mit dem höchsten Gewicht ersetzt. Die Auflösung des finalen Netzes wird durch die Granularität des 3-dimensionalen Gitters, das über das Objekt gelegt wird, bestimmt.

Das Gewicht eines Vertex wird in [RB92] durch die maximale Länge der einfallenden Kanten und den größten Winkel dazwischen bestimmt. Diese Faktoren sind maßgeblich für die Qualität der Vereinfachung und wurden im Zuge weiterer Entwicklungen mehrmals verändert und auch erweitert, um zusätzliche Vertexparameter zu berücksichtigen.

Der Ansatz in [RB92] beruht auf einem regelmäßigen 3-dimensionalen Gitter. Dies hat den Nachteil, dass in manchen Zellen des Dreiecks ausschließlich Vertizes mit geringem Gewicht zu finden sind. Um dies zu vermeiden, wird in [LT97] eine Technik mit frei platzierbaren Zellen präsentiert. Die Zellen des Rasters werden in [LT97] durch Sphären ersetzt, um die Effizienz der Berechnungen zu erhöhen. Diese Sphären werden nicht in einem Gitter angeordnet: Eine Sphäre wird definiert, die den Vertex mit dem gegenwärtig höchsten Gewicht im Zentrum hat. Die übrigen Vertizes innerhalb der Sphäre werden in diesem Punkt zusammengefasst. Der Vorgang wiederholt sich, bis alle Vertizes des Netzes abgearbeitet sind.

Der Vorteil dieser Idee besteht darin, dass das Gewicht der Vertizes besser in den Algorithmus einfließt und man somit gleichmäßigere Ergebnisse erzielt. Artefakte, die in der ursprünglichen Version des Algorithmus auftreten, werden demnach weitgehend vermieden [LT97].

In [SG01] wird der Ansatz des Gitters beibehalten, allerdings von der regelmäßigen Form abgewichen. Anstelle dessen kommt ein BSP (Binäre Raumunterteilung, engl. „Binary Space Partitioning“) Baum zum Einsatz. Jeder Knoten des Baumes beinhaltet einen Unterraum des Objektraumes. Die Kinder eines Knotens werden erzeugt, indem der im Knoten enthaltene Raum durch eine Ebene in zwei Teilräume getrennt wird. Der Vorgang wiederholt sich, bis in jedem Knoten nur mehr eine minimale Anzahl an Vertizes zu finden ist.

Diese Technik bietet zwei Vorteile:

- **Optimierte Gruppierung** Durch die Anwendung einer Fehlermetrik, mit deren Hilfe die Trennungsebene durch eine Zelle gelegt wird, ist es möglich zu steuern, welche Vertizes zusammengefasst werden. Dies erlaubt eine Optimierung des Algorithmus und eine Verbesserung der Resultate.

- **Dynamischer Detailgrad** Mit der Baumstruktur müssen nicht die untersten Knoten des Baumes für die Gruppierung zum Einsatz kommen. Vielmehr kann man auch höher im Baum liegende Zellen zusammenfassen. Somit lässt sich die Auflösung des resultierenden Dreiecksnetzes steuern.

Der Nachteil dieses Verfahrens liegt in der größeren Komplexität des Algorithmus und damit verbunden einem höheren Speicher- und Rechenaufwand [LWC+02].

In [SW03] kommt anstelle eines BSP Baumes ein Octree zum Einsatz. Jeder Knoten hat also acht Kindknoten. Der Algorithmus bestimmt eine maximale Anzahl an Knoten, die der Baum enthalten darf. Dies ist nötig, um einerseits den Speicherbedarf zu limitieren und andererseits, um die minimale Zahl an Vertizes festzulegen. Der Baum ist nicht gleichmäßig oder vollständig, also nicht alle Kinder müssen vorhanden sein.

Die Vertizes des originalen Netzes werden Knoten des Octrees zugewiesen, wobei der Baum zu diesem Zeitpunk nicht vollständig existiert, sondern erst mit der festgelegten Knotenanzahl durch die Daten der Vertizes generiert wird. Im Optimalfall würde jedes Blatt des Octrees einen Vertex des originalen Netzes enthalten, dies ist jedoch aufgrund des großen Speicherbedarfs bei Netzen mit hoher Vertexanzahl nicht praktikabel.

Abbildung 2.20 zeigt einen Vergleich verschiedener Ansätze, die eine Vereinfachung mittels Vertexgruppierung durchführen. Auf der linken Seite ist das originale Objekt abgebildet, die drei Darstellungen rechts daneben zeigen einen Ausschnitt des Objekts, der mit Hilfe von drei verschiedenen Ansätzen vereinfacht wurde. Beim linken Detailbild wurde ein einfaches, regelmäßiges Gitter eingesetzt. In der Mitte wurde mit Hilfe eines BSP Baumes die Vereinfachung errechnet. Hier sind weichere Rundungen zu erkennen, es wurden allerdings Stege geschaffen. Auf der rechten Seite ist die Vereinfachung mittels Octree zu sehen, die sich durch die beste Wiedergabe des originalen Objekts auszeichnet.

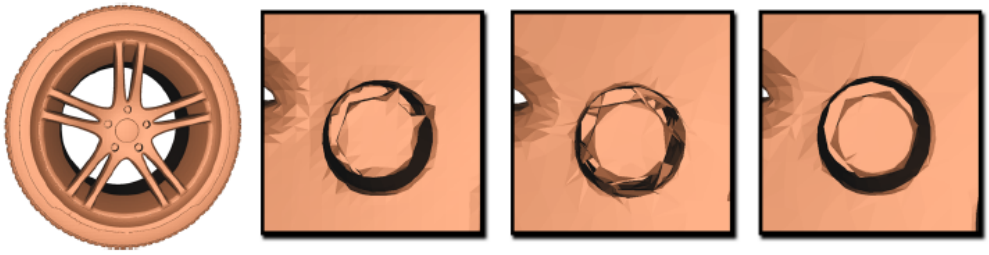

Abbildung 2.20: Vergleich: Regelmäßiges Gitter (links), BSP Baum (Mitte) und Octree (rechts) [SW03]

Eine weitere Methode, die auf Vertexgruppierung beruht, ist RSimp [BW99]. Diese Vorgehensweise beginnt mit der Generierung einer Reihe von Zellen innerhalb des Begrenzungsvolumens, die im Anschluss anhand einer Fehlermetrik unter Berücksichtigung der Normalvektoren verfeinert wird [CW02]. Ziel dieses Algorithmus ist, gekrümmte Oberflächenteile zu erkennen und diese mit mehr Vertizes darzustellen als gerade Flächen.

Dafür werden folgende Schritte ausgeführt:

1. Berechnung des Begrenzungsvolumens

2. Iterative Zerlegung der Zellen anhand der Oberflächenkrümmung innerhalb einer Zelle

3. Aufbau der neuen Oberfläche

In Schritt 1 berechnet RSimp das Begrenzungsvolumen eines Objekts. Dieses wird in regelmäßige Zellen unterteilt, und alle Vertizes des Objekts werden Zellen zugewiesen. Im Gegensatz zu anderen Algorithmen beruht die hier verwendete Fehlermetrik nicht auf Position oder Abstand der Vertizes, sondern das entscheidende Kriterium bildet der Normalvektor. Er dient dazu, die Krümmung von Oberflächen zu bestimmen.

In Schritt 2 werden alle Zellen analysiert und anhand der Abweichungen der Normalvektoren der Dreiecke in der Zelle sortiert und in einer Liste gespeichert [BW99], [Bro00]. Zellen mit den stärksten Abweichungen werden geteilt, die neuen Zellen analysiert und entsprechend in die Liste eingetragen. Dieser Vorgang wiederholt sich so lange, bis eine gewünschte Anzahl an Zellen erreicht ist.

Der dritte und letzte Schritt besteht darin, die neue Oberfläche analog zu anderen Vertexgruppierungs-Verfahren aufzubauen.

Die Fehlermetrik beruht ausschließlich auf zwei Faktoren: Der Fläche a_i eines Dreiecks und dem Normalvektor n_i. Die Fehlermetrik für ein Netz mit M Dreiecken und eine Zelle mit N Dreiecken wird berechnet durch:

$$\frac{\sum_{i=1}^{N} a_i}{\sum_{i=1}^{M} a_i}(1 - \frac{\sum_{i=1}^{N} a_i n_i}{\sum_{i=1}^{N} a_i}) \tag{2.7}$$

Auch die Unterteilung von Zellen wird anhand der Normalvektoren vollzogen. So kann es zu drei unterschiedlichen Trennungen kommen (Abbildung 2.21):

- Die Zelle wird mit Hilfe von drei Ebenen in 8 neue Zellen zerlegt. Dieser Fall tritt ein, wenn extreme Abweichungen in den Normalvektoren gefunden werden (links in Abbildung 2.21).

- Zwei Ebenen kommen zum Einsatz, die vier neue Zellen generieren (Mitte in Abbildung 2.21).

- Bei einer mäßigen Abweichung der Normalvektoren erzeugt eine einzelne Ebene zwei neue Zellen (rechts in Abbildung 2.21).

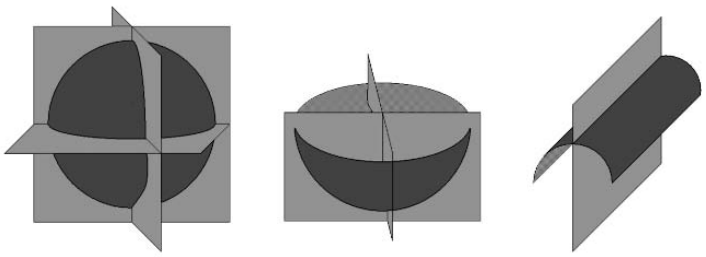

Abbildung 2.21: Unterteilungen in RSimp [LWC+02]

RSimp stellt einen Kompromiss zwischen Qualität und Leistung dar. Durch die Unterteilung der Zellen anhand der Krümmung werden gute Ergebnisse erzielt [BW00a]. Zusätzlich lässt sich mittels Partitionierung des Dreiecksnetzes in mehrere Zellen die weitere Verfeinerung der einzelnen Bereiche parallelisieren und somit eine Beschleunigung erreichen [BW00b].

In [DT07] wurde das Prinzip der Vertexgruppierung aufgegriffen und ein Algorithmus für die Ausführung auf einer GPU vorgestellt. Die Daten des Dreiecksnetzes werden durch die Stufen der Graphikverarbeitung abgewickelt. Um die Ersatzposition für eine Gruppe von Vertizes zu bestimmen, wird in [DT07] auf die Idee der Quadrik-Fehlermetrik von Garland und Heckbert [GH97] zurückgegriffen.

Für diese Berechnung sind drei Schritte notwendig:

- **Generierung der „Gruppierungs-Quadrik Abbildung“** Aus dem originalen Objekt und dem Begrenzungsvolumen definiert der Benutzer durch eine spezifizierte Unterteilungszahl die Menge der Zellen. Ein Oberflächenmuster wird erstellt, das als Datenspeicher fungiert und für jede Zelle eine Fehlermatrix und die durchschnittliche Vertexposition ablegt. Der Vertex-Shader errechnet für jeden Punkt, welcher Zelle der Vertex zugewiesen wird und der Geometrie-Shader bestimmt aus dem Dreieck die Matrix ([DT07], [GH97]). Der Pixel-Shader schreibt die Information in das Muster.

- **Berechnung der optimalen Ersatzpositionen** Aufgrund der Ergebnisse der im vorigen Schritt errechneten Daten wird für jede Zelle mit Hilfe der für den gesamten

Bereich resultierenden Matrix eine Ersatzposition festgelegt. Hierfür wird auf den Ansatz von Garland und Heckbert aus [GH97] zurückgegriffen und eine Position für den Ersatzvertex errechnet, die den Fehler anhand der Quadrik-Fehlermetrik minimiert.

- **Erzeugung des vereinfachten Netzes** Das originale Netz wird erneut durch die Stufen der Graphikverarbeitung behandelt. Der Vertex-Shader errechnet, zu welcher Zelle der Vertex gehört und findet somit die Ersatzposition. Der Geometrie-Shader generiert die neuen Dreiecke des vereinfachten Netzes und verwirft solche, deren Vertizes sich innerhalb einer Zelle befinden.

Dieser Algorithmus lässt sich noch weiter optimieren, indem vor der Zuweisung von Vertizes zu den Zellen eine Transformation der Vertizes in den Bildraum ausgeführt wird. Somit ist das originale Netz verzerrt und eine vom Blickwinkel abhängige Vereinfachung erreicht. Abbildung 2.22 zeigt die Auswirkungen dieser Optimierung. Auf der linken Seite sind das Gitter sowie das vereinfachte Objekt ohne diese Verzerrung dargestellt. Hier sind alle Zellen gleich groß. In der zweiten Abbildung wurde die Verzerrung angewandt, wodurch die Zellen auf der rechten Seite gestreckt wurden, was sich auch auf die Form des Objekts auf der rechten Seite auswirkt. Hier ist eine einfachere Geometrie beim Schwanz des Drachen erkennbar.

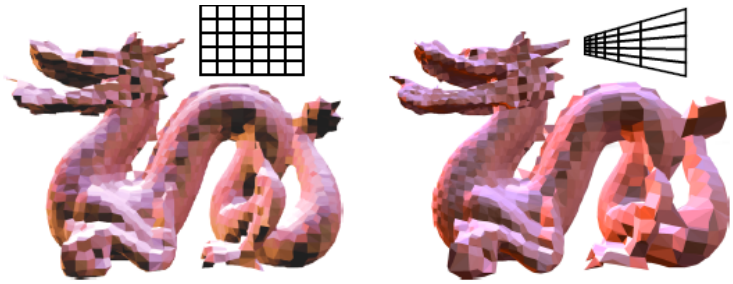

Abbildung 2.22: Vergleich: Vereinfachung mit und ohne Transformation [DT07]

Der Vorteil von Vertexgruppierungs-Verfahren besteht darin, dass sie einfach und effizient zu implementieren sind. Bei der Umsetzung auf moderner Hardware eignen sie sich auch gut für den Einsatz in Echtzeit.

Diese Algorithmen haben den Nachteil, dass die Topologie eines Dreiecksnetzes ignoriert wird. Alle Vertizes in einem Bereich werden ohne Rücksicht auf die visuellen Auswirkungen zusammengefasst. Während andere Techniken das Entfernen einzelner Vertizes bewerten und die Folgen berücksichtigen, kann solch ein Vorgehen hier nicht durchgeführt werden. Besonders der Einsatz von regelmäßigen Gittern bringt Probleme, da Bereiche eines Objekts mit sehr vielen Details in gleichem Maße reduziert werden wie Zellen, in denen kaum Details gespeichert sind [LWC+02].

Eine zusätzliche Schwierigkeit tritt auf, weil die Ergebnisse der Vereinfachung nicht vorhersagbar sind. Da kein Maß für die Auswirkungen der Operationen zum Einsatz kommt, wird die Dreieckszahl ausschließlich durch die Zahl der erzeugten Zellen, nicht jedoch durch eine Metrik für die Auswirkungen der Vereinfachungsoperationen gesteuert.

Das nächste Verfahren, das beschrieben wird, ist die Vertexdezimierung.

2.4.4 Vertexdezimierung

Vertexdezimierung ist ein Verfahren, das erstmals 1992 in [SZL92] vorgestellt wurde und basiert auf Vertexeliminierung (Unterabschnitt 2.3.6). Es werden mehrere Iterationen ausgeführt. In jeder Iteration werden die Vertizes eines Dreiecksnetzes bewertet, Vertizes deren Bewertung unterhalb eines gewissen Grenzwertes liegt, entfernt und die resultierenden Löcher verschlossen. Mit jeder Iteration kommt eine höhere Toleranz für das Entfernen von Vertizes zum Einsatz. Dies wiederholt sich, bis der gewünschte Grad der Vereinfachung erreicht ist.

Im ersten Schritt führt der Algorithmus für die Vertexdezimierung eine Klassifizierung ein. Jeder Vertex wird einer Kategorie zugeordnet (die Kategorien sind in Abbildung 2.23 dargestellt):

- **Einfach** Der Vertex ist von einem kompletten Ring aus Dreiecken umgeben.

- **Begrenzung** Der den Vertex umgebende Ring aus Dreiecken ist unvollständig.

- **Innere Kante** Eine Kante wird abhängig von dem Winkel zwischen den einfallenden Dreiecken als markante Kante bewertet, wenn dieser eine bestimmte Größe übersteigt. Ein Vertex, der in zwei markanten Kanten enthalten ist, wird als innere Kante klassifiziert.

- **Ecke** Analog zu inneren Kanten, allerdings finden sich hier 3 oder mehr markante Kanten für einen Vertex.

- **Komplex** Der Vertex fällt in keine der obigen Kategorien und wird nie entfernt.

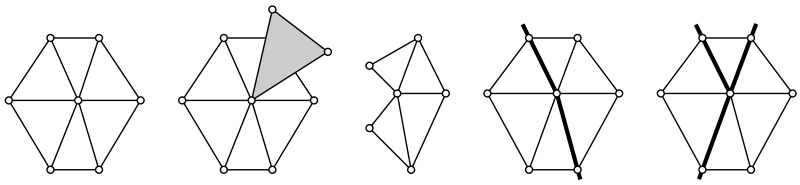

Abbildung 2.23: Kategorien der Vertizes (Einfach, Komplex, Begrenzung, Innere Kante und Ecke)

Je nach Kategorie eines Vertex kommt eine eigene Fehlermetrik zum Einsatz. Multiple Fehlermetriken erlauben eine verbesserte Auswahl der Vereinfachungen, die Formen des Objekts genauer zu erhalten und Fehler im Netz bei unbekannten Konstellationen zu vermeiden. Einzig Vertizes der Kategorie Komplex sind bei diesem Algorithmus von der Vereinfachung immer ausgenommen.

Die Fehlermetrik für die Kategorie Einfach in [SZL92] wird mit Hilfe einer durchschnittlichen Ebene berechnet, die durch jene Dreiecke, welche den Vertex enthalten, gebildet wird. Der Normalvektor der Ebene ist der Durchschnitt aller Dreiecksnormalen der mit dem Vertex gebildeten Dreiecke. Die Flächen der Dreiecke fließen als gewichtender Faktor in die Berechnung des Mittelwertes ein. Der Punkt für die Bestimmung der Lage der durchschnittlichen Ebene wird mit Hilfe des Durchschnittswertes der Mittelpunkte aller Dreiecke bestimmt. Auch hier findet eine Gewichtung mittels der Dreiecksflächen statt.

Der Fehlerwert des Vertex ist der Normalabstand zwischen Vertex und der durchschnittlichen Ebene. Ist diese Distanz kleiner als ein vom Benutzer gewählter Grenzwert, so entfernt man den Vertex.

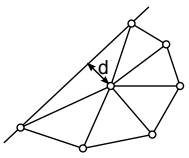

Abbildung 2.24: Fehlermetrik bei „Begrenzung“ Vertizes

Die Fehlermetrik für Begrenzungen und innere Kanten berechnet eine Gerade, welche beide Vertizes, die eine äußere Begrenzung oder innere Kante bilden, beinhaltet. Abbildung 2.24 zeigt ein Beispiel mit einer Geraden für eine äußere Begrenzung. Der Normalabstand d zwischen der Geraden und dem zentralen Vertex ist der Fehlerwert des Vertex (d in Abbildung 2.24). Analog zum einfachen Vertex kann der Punkt entfernt werden, wenn dieser ermittelte Normalabstand unterhalb des vom Benutzer gewählten Grenzwertes liegt.

Vertizes der Kategorie Ecke werden als wichtig erachtet. Sie enthalten topologische Informationen und werden generell nicht gelöscht. In [SZL92] wird allerdings darauf hingewiesen, dass bei sehr unruhigen Dreiecksnetzen die inneren Ecken- und Kanten-Vertizes mit der gleichen Fehlermetrik behandelt werden können wie Vertizes der Kategorie Einfach. Ansonsten würden nur sehr wenige Punkte für eine Vereinfachung zur Verfügung stehen, und das Verfahren könnte nur eine geringe Zahl an Vertizes entfernen.

In [SZL92] wird auf einen rekursiven Algorithmus zur Retriangulierung der Löcher zurückgegriffen. Es wird davon ausgegangen, dass die den ursprünglichen Vertex umgebenden Punkte einen Ring bilden. Der Algorithmus sucht zwei nicht benachbarte Punkte des Ringes und trennt ihn anhand dieser in zwei Abschnitte. Der Vorgang wird so lange wiederholt, bis ein Teil nur mehr aus drei Vertizes besteht und somit ein Dreieck bildet.

Das Problem ist, dass die Vertizes nicht in einer Ebene liegen. Für den gesamten Vertexring wird eine durchschnittliche Ebene gebildet. Eine Teilung findet statt, indem eine weitere Ebene, die orthogonal auf dieser durchschnittlichen Ebene steht, durch zwei Vertizes des Kreises gelegt wird. Im Anschluss überprüft man, ob sich alle Vertizes eines abgetrennten Segments des Ringes auf einer Seite der Trennungsebene befinden. Nur wenn diese Bedingung erfüllt ist, wird die Teilung als gültig erachtet. Abbildung 2.25 zeigt ein Beispiel für die durchschnittliche Ebene und die Trennungsebene.

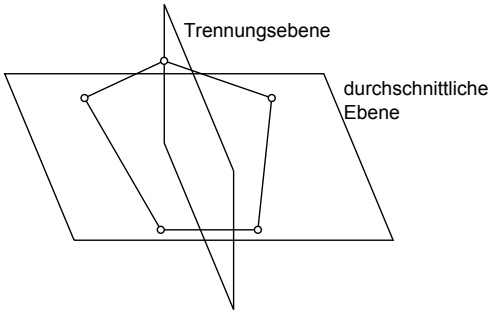

Abbildung 2.25: Berechnung der Retriangulierung

Sollte keine gültige Trennungsebene gefunden werden und die Retriangulierung schlägt fehl, so wird das Entfernen des Vertex abgebrochen. Er bleibt im Dreiecksnetz erhalten.

Ein großer Vorteil dieses Algorithmus ist die lineare Berechnungszeit in Abhängigkeit von der Anzahl der Vertizes. Sie ermöglicht eine effiziente Implementierung, wodurch das Verfahren oft Anwendung findet [LWC+02].

Der Algorithmus erlaubt, redundante Daten in Dreiecksnetzen zu erkennen und zu entfernen. Das resultierende, vereinfachte Dreiecksnetz besteht aus einer Teilmenge der Vertizes des originalen Netzes. Diese Tatsache beeinträchtigt allerdings in manchen Si-

tuationen auch die Qualität des Ergebnisses, da nur Daten des ursprünglichen Objekts gelöscht werden.

Ein weiterer Algorithmus - die bildbasierte Vereinfachung - greift auf einen bildbasierten Ansatz zurück, um die Auswirkungen von Vereinfachungen zu bestimmen.

2.4.5 Bildbasierte Vereinfachung

Diese in [LT00] vorgestellte Technik beruht auf der Verwendung der Kantenreduktion als Vereinfachungsoperator. Die Kosten einer Vereinfachung werden im Gegenzug zu anderen Algorithmen nicht mit Hilfe von geometrischen Mitteln, sondern anhand des dargestellten Bildes bewertet.

Ähnlich dem Ansatz der Quadrik-Fehlermetrik wird eine Menge von potentiellen Vereinfachungsoperationen auf dem Dreiecksnetz definiert. Diese werden gespeichert. Im Anschluss führt man eine mögliche Kantenreduktion durch, und das resultierende Dreiecksnetz wird aus mehreren Blickwinkeln dargestellt. Die ermittelten Bilder werden mit ihren Äquivalenten des originalen Netzes verglichen. Somit lässt sich entscheiden, wie stark die sichtbare Abweichung durch die Vereinfachung ausfällt und welche Vereinfachung bevorzugt auf dem Objekt ausgeführt werden soll.

Das Ergebnis dieser Technik sind sehr gute visuelle Resultate, da die sichtbaren Auswirkungen direkt bewertet werden (im Gegensatz zu einer versuchten Bestimmung dieser Veränderungen anhand der manipulierten Geometrie). Der Ansatz erfordert nach jeder Vereinfachungsoperation eine wiederholte Darstellung des gesamten Objekts. Dieses Verfahren stellt eine hohe Anforderung an die Rechenleistung und ist somit für den Echtzeiteinsatz nicht praktikabel. Eine mehrfache Berechnung eines Rasterbildes nach jedem Vereinfachungsschritt, um die Abweichung zu bestimmen, führt zu einem Leistungsbedarf, der durch eine vereinfachte Version bei der finalen Darstellung nicht gerechtfertigt ist.

Die Anzahl der zu analysierenden Dreiecke bei jeder Vereinfachung wird reduziert, indem nicht das gesamte Objekt berücksichtigt wird, sondern nur jene Teile, die direkt manipuliert wurden. Somit lässt sich eine starke Reduktion des Arbeitsaufwandes erreichen. Um die vollständigen Auswirkungen der Vereinfachungen zu bestimmen, wird das Ergebnis der Vereinfachung in [LT00] aus 20 verschiedenen Perspektiven mit dem Original verglichen. Abbildung 2.26 zeigt alle Perspektiven, aus denen das Modell des Stanford Bunny in [LT00] dargestellt wurde, um die Auswirkungen von Vereinfachungsoperationen zu bestimmen. Die Anzahl der gewählten Blickwinkel ist dabei nicht festgelegt. Vielmehr ist sie ein Kompromiss aus Leistung und Qualität. Je mehr Blickwinkel bei diesem Algorithmus berücksichtigt werden, desto besser ist die Qualität des Resultats. Die aufzuwendende Rechenleistung steigt mit der Anzahl der Perspektiven.

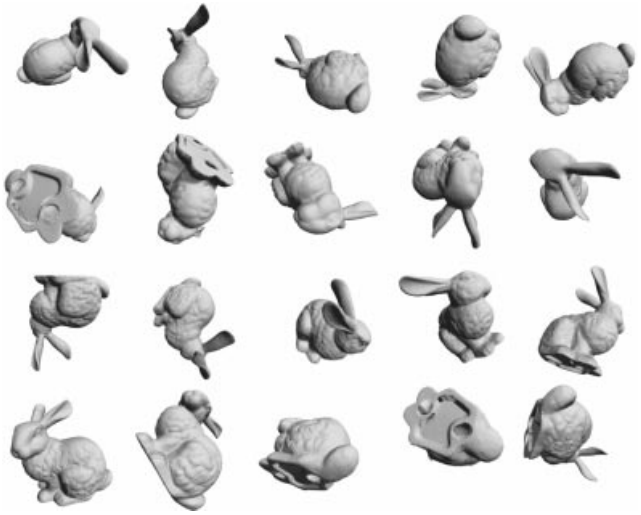

Abbildung 2.26: Dargestellte Blickwinkel [LT00]

Um die Abweichung zwischen zwei Darstellungen zu vergleichen, wird in [LT00] zuerst für das gesamte Bild die Luminanz jedes einzelnen Bildpunktes errechnet. So wird einerseits der Aufwand für den Vergleich auf einen einzelnen Wert reduziert und andererseits erkennt die menschliche Wahrnehmung Unterschiede vor allem durch die Helligkeit.

Im Anschluss daran wird über alle Pixel zweier zu vergleichender Bilder iteriert. Dabei erfolgt die Bestimmung des quadratischen Mittelwertes aller Abweichungen der berechneten Luminanzen. Betrachtet man zwei Luminanzbilder Y^0 und Y^1 mit der Dimension m x n, so ist die Abweichung zwischen diesen Bildern bestimmt durch:

$$d_{RMS}(Y^0, Y^1) = \sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}^0 - y_{ij}^1)^2} \tag{2.8}$$

In [LT00] wird argumentiert, dass diese sehr einfache bildbasierte Fehlermetrik nicht immer optimal ist und in manchen Fällen durch Verzerrungen und Transformationen keine perfekten Resultate liefert. Sie wurde aufgrund der effizienten Berechnung und des relativ geringen Leistungsaufwandes gewählt und hat in Testfällen zufriedenstellende Ergebnisse erzielt. Der Algorithmus und somit die Qualität der Vereinfachung lassen sich aber durch den Einsatz einer überarbeiteten Fehlermetrik mit dem möglichen Nachteil einer längeren Berechnungszeit verbessern.

Ein anderer Ansatz wird in [KWA12] verfolgt. Im Gegensatz zu den Luminanzen werden hier die Tiefenwerte des Rasterbildes, das unter Einsatz einer Orthogonalprojektion berechnet wurde, verglichen. Für jeden Bildpunkt wird ein Tiefenwert errechnet. Die Unterschiede zwischen den Tiefenbildern des originalen und modifizierten Dreiecksnetzes werden ermittelt und aufsummiert. Der Vorgang wird für alle zum Einsatz kommen-

den Perspektiven wiederholt. Folglich stellt die kumulierte Abweichung den gesamten Fehler einer Vereinfachung dar.

Um die Darstellungszeit der einzelnen Blickwinkel zu reduzieren, wird in [LT00] ein Verfahren angewandt, das die Anzahl der zu berechnenden Pixel verringert. Im ersten Ansatz kommt ein Begrenzungsvolumen im Bildraum zum Einsatz, das alle beteiligten Dreiecke enthält. Nur diese zu berücksichtigen wäre allerdings nicht ausreichend, da durch die Veränderungen der Silhouette und der Tiefenwerte auch andere Dreiecke betroffen wären und sich auf das Rasterbild auswirkten.

Um dieses Problem zu lösen, kommt in [LT00] eine Anzahl von Hashtabellen zum Einsatz. Jedem Pixel werden eine Reihe und eine Spalte zugewiesen, und die Hashtabellen speichern, welche Dreiecke in einer Zeilen-/Spaltenkombination eines Pixels liegen.

Demnach können die betroffenen Pixel ermittelt, alle nötigen Dreiecke anhand der Tabellen bestimmt und im Anschluss berechnet werden. So lässt sich vermeiden, in jedem Bild zur Kostenbestimmung der Vereinfachung das gesamte Objekt mehrmals darzustellen. Stattdessen reduziert man den Aufwand auf eine überschaubare Menge von Dreiecken.

[QM06] beschäftigt sich mit der Auswirkung von „Verdeckungen“ auf der Oberfläche. Die Autoren argumentieren, dass durch Berechnungen - wie etwa der Manipulation von Normalvektoren oder der Projektion von Oberflächenmustern - der Verlust von Details bei der Geometrie „verdeckt“ wird. Somit kann eine zusätzliche Reduktion der Vertexzahl vorgenommen werden.

Während die bildbasierte Vereinfachung gute Resultate liefert, besteht der Nachteil in einem großen Berechnungsaufwand. Grund dafür ist die hohe Anzahl an notwendigen Blickwinkeln und somit Darstellungsvorgängen für die Berechnung der Fehlermetrik [PJ03].

Im Anschluss wird ein Verfahren beschrieben, das versucht, eine Menge von Kantenreduktionen parallel auszuführen.

2.4.6 Parallele Kantenreduktion durch individuelle Bereiche

Einen Ansatz, um die Vereinfachung von Dreiecksnetzen zu beschleunigen, stellt die gleichzeitige Ausführung mehrerer Vereinfachungsoperationen dar. Dieses Vorgehen kann die parallelen Kapazitäten moderner GPUs nutzen und die Berechnungszeit für die Vereinfachung verkürzen.

Der Ansatz in [PP15] greift auf die Kantenreduktion zurück. Anstelle einer Sequenz von Kantenreduktionen soll eine Reihe von Kantenreduktionen parallel ausgeführt wer-

den. Die Problematik hierbei besteht im Auffinden von Kombinationen von Kantenreduktionen, die sich nicht gegenseitig beeinflussen. Um dies zu erreichen, wird eine Menge von unabhängigen Bereichen des Dreiecksnetzes gesucht. Laut Definition in [PP15] sind zwei Bereiche dann unabhängig, wenn eine in einem Bereich ausgeführte Kantenreduktion den zweiten Bereich nicht beeinflusst. Die Autoren beschreiben einen Bereich als einen Vertex inklusive aller ihn umgebenden Vertizes. Zwei Bereiche sind als unabhängig definiert, wenn ihre zentralen Vertizes „super-unabhängig“ sind. Dies bedeutet, dass sie auf einem Graphen (mit den Vertizes des Dreiecksnetzes als Knoten des Graphen und den Kanten des Netzes als Kanten des Graphen) einen Mindestabstand von 3 besitzen.

Die Vereinfachung erfolgt, indem zuerst für jeden Vertex ein Fehlerwert berechnet, mit dem über den Verbleib einer Kante im Netz entschieden wird. Im Anschluss ist eine Menge an individuellen Bereichen zu ermitteln. In jedem Bereich kann eine Kantenreduktion ausgeführt werden. Anhand des berechneten Fehlerwertes wird eine Kante ausgewählt, die Kante sowie die betroffenen Dreiecke entfernt und die Topologie innerhalb des Bereichs angepasst.

Das Bestimmen unabhängiger Bereiche sowie die anschließende Ausführung der Kantenreduktion wiederholen sich, bis der gewünschte Grad der Vereinfachung erreicht ist.

Ein zweiter Ansatz für die parallele Ausführung von Kantenreduktionen wird im nächsten Abschnitt beschrieben.

2.4.7 Parallele Kantenreduktion durch Sperren von Bereichen

In [SN13] wird anstatt des Einsatzes individueller Bereiche versucht, parallel eine Menge von Kantenreduktionen zu bestimmen. Sobald eine mögliche Reduktion selektiert wurde, werden alle von der Ausführung betroffenen Vertizes gesperrt, so dass keine zweite, gleichzeitig ablaufende Kantenreduktion zu unerwünschten Resultaten führen kann.

Für diesen Algorithmus wird auf Hauptprozessor und GPU zurückgegriffen. In einem ersten Schritt wird die Arbeitslast - in diesem Fall die Anzahl der Vertizes im Dreiecksnetz - per benutzerdefiniertem Prozentsatz auf Hauptprozessor und GPU verteilt.

Jeder Recheneinheit werden ein oder mehrere Vertizes zugewiesen. Jede Recheneinheit sucht für die zugewiesenen Vertizes eine Kantenreduktion. Ist eine mögliche Kantenreduktion selektiert, so werden mittels einer atomaren Operation, die Wettbewerbsbedingungen durch parallele Zugriffe ausschließt, alle von der Operation beeinflussten Vertizes blockiert. Schlägt dies für einen oder mehrere Vertizes fehl - da ein Vertex bereits von einer anderen Kantenreduktion blockiert ist - so ist die Kantenreduktion nicht realisierbar. Nur wenn alle benötigten Vertizes gegenwärtig von keiner anderen

Operation blockiert werden, kann die Kantenreduktion durchgeführt und die Vertizes wieder freigegeben werden.

Dieses Vorgehen wird wiederholt, bis der gewünschte Grad der Vereinfachung erreicht ist.

Der nächste Abschnitt beschäftigt sich mit der Vereinfachung mittels Morton Integralen.

2.4.8 Vereinfachung mittels Morton Integralen

Das in [LB15] präsentierte Verfahren sortiert alle Vertizes eines Dreiecksnetzes. Dafür kommt eine Z-Kurve zum Einsatz: Für jeden Vertex wird ein Z-Wert der Position im Raum durch Verschränken der binären Repräsentationen der X-, Y- und Z-Koordinate berechnet. Die Sortierung aufgrund des Z-Wertes gruppiert die einzelnen Vertizes nach ihrer räumlichen Kohärenz. Im Anschluss wird eine Liste mit Blattknoten erzeugt. Jedem Blattknoten ist ein Vertex zugewiesen. Die Blattknoten sind - ebenso wie die Vertizes - anhand des Z-Wertes des enthaltenen Vertex sortiert. Blattknoten, die Vertizes mit identischem Z-Wert enthalten, werden in einem Knoten zusammengefasst.

Für jeden Blattknoten wird die symmetrische, quadratische Matrix Q nach Garland und Heckbert [GH97] bestimmt: Für jedes Dreieck t wird die Matrix Q_t aufgestellt und zur Matrix aller jener Blattknoten, die einen Vertex in t enthalten, addiert. Um eine schnelle Berechnung zu ermöglichen, wird für jeden Knoten die kumulative Summe der Attribute (Matrix Q, durchschnittliche Position, ...) aller Blattknoten mit kleinerem Z-Wert ermittelt. Außerdem wird für jeden Blattknoten ein zusätzliches Attribut Q_{scan} berechnet. In Q_{scan} werden die Quadriken aller vorhergehenden Blattknoten aufsummiert („Morton Integral“). Dieser Schritt erlaubt es, die Summe eines Attributes für eine beliebige Menge aufeinanderfolgender Blattknoten effizient zu ermitteln.

Aus den Blattknoten wird eine hierarchische Datenstruktur aufgebaut. Für jeden Knoten des Baumes wird ein Fehlerwert nach Garland und Heckbert berechnet, der angibt, wie stark die Abweichung von der originalen Geometrie ausfällt, wenn alle Kindknoten in einem Vertex zusammengefasst würden. Je mehr Vertizes in einem Knoten enthalten sind, desto größer wird der Vertexfehler. Da in jedem Knoten nur aufeinanderfolgende Blattknoten enthalten sind, ist dies durch die Morton Integrale effizient berechenbar.

Mit Hilfe dieses Baumes wird bestimmt, welche Vertizes zusammengefasst werden und wo sich ihre Ersatzposition befindet. Gegeben ist eine Toleranzgrenze für den Vertexfehler. Es wird ein Schnitt durch den Baum gesucht, der die höchsten Knoten selektiert, welche die Toleranzgrenze nicht überschreiten. Sie bestimmen die Vertizes des vereinfachten Dreiecksnetzes.

Die Autoren geben an, dass neben der Vertexposition noch weitere Attribute - wie

etwa Normalvektor oder per-Vertex Farbwerte - in die Morton Integrale einfließen können und somit die Qualität der Vereinfachung verbessert wird.

Die Vereinfachung mittels Morton Integralen zeichnet sich zwar durch sehr gute Laufzeiten aus, sie basiert jedoch auf der Idee der Vertexgruppierung. Dementsprechend weist die Vereinfachung nicht dieselbe, hohe Qualität auf, wie sie bei anderen Verfahren erzielt werden kann.

Anschließend wird ein Verfahren beschrieben, das mit Hilfe einer Menge von Bäumen ein Modell vereinfacht.

2.4.9 Vereinfachung durch Reduktion von Teilbäumen

In [LK16] ist ein Verfahren beschrieben, das durch eingeschränkte Kantenreduktionen eine Vereinfachung berechnet. Es wird definiert, dass für jede Kante zwei eingeschränkte Kantenreduktionen existieren. Für eine Kante e, gebildet mit den Vertizes v_1 und v_2, gibt es eine Kantenreduktion e_0, die v_1 durch v_2 und ihren Zwilling e_1, der v_2 durch v_1 ersetzt. Es wird eine Liste L mit allen eingeschränkten Kantenreduktionen eines Dreiecksnetzes gespeichert (zwei Einträge pro Kante). Bei Ausführung einer Kantenreduktion werden die Einträge in der Liste L aktualisiert, um die veränderten Kanten zu repräsentieren: Für jeden Eintrag wird bestimmt, ob er unverändert bestehen bleibt, mit einer anderen eingeschränkten Kantenreduktion kombiniert wird oder ob er entfernt werden kann. Mit Hilfe der in L gespeicherten Information können Manipulationen der Geometrie bestimmt und das vereinfachte Dreiecksnetz aufgebaut werden.

Um auszuwählen, welche Kantenreduktionen ausgeführt werden sollen, wird in [LK16] eine Menge von Bäumen bestimmt. Für jede eingeschränkte Kantenreduktion wird der Fehlerwert nach Garland und Heckbert [GH97] berechnet. Es wird davon ausgegangen, dass ein Grenzwert für den maximalen Fehler der Vereinfachung bestimmt ist. Für jeden Vertex wird die Kantenreduktion - die den Vertex auf die Position eines Nachbarn verschiebt - mit dem kleinsten Fehler gesucht und gespeichert (sofern dieser kleiner als der erlaubte maximale Fehler ist). Auf diese Weise entsteht eine Menge von Bäumen: Vertizes sind die Knoten, die ausgewählten eingeschränkten Kantenreduktionen die Kanten der Bäume (Kindknoten werden auf die Position von Eltern verschoben). Alle Knoten eines Baumes werden im Wurzelknoten vereint. Um eine Überschreitung der erlaubten Toleranzgrenze zu vermeiden, werden diese Bäume geteilt. Beginnend bei den Blattknoten wird für jeden Knoten K berechnet, wie groß der Fehlerwert nach Garland und Heckbert ausfällt, wenn alle Kinder von K durch eingeschränkte Kantenreduktionen auf die Position von K verschoben werden. Überschreitet dieser Fehlerwert den definierten Grenzwert, so wird der Baum an dieser Stelle geteilt und die Kinder von K werden zu Wurzelknoten von getrennten Bäumen. Die Wurzelknoten der so entstehenden, finalen Menge an Bäumen repräsentieren die Vertizes des vereinfachten Dreiecksnetzes.

Um die Vereinfachung auszuführen, wird jeder Baum durch eine Menge von eingeschränkten Kantenreduktionen auf den Vertex im Wurzelknoten reduziert. Die Teilschritte dieses Verfahrens können gut parallelisiert werden, und die Berechnung auf einer GPU erzielt eine Reduktion der Laufzeit. Weiters erlauben der Aufbau der Bäume sowie der Einsatz der Liste der Kantenreduktionen ein gleichzeitiges Entfernen von benachbarten Vertizes.

Im Anschluss wird ein Überblick über die präsentierten Verfahren gegeben.

2.4.10 Vergleich der präsentierten Verfahren

In Tabelle 2.2 wird ein Überblick über die präsentierten Verfahren geboten. Dabei wird für jedes Vereinfachungsverfahren in der ersten Spalte der Name angegeben. In der zweiten Spalte ist der eingesetzte Vereinfachungsoperator genannt. Dieser ist ausschlaggebend, ob ein Verfahren die Mannigfaltigkeit eines Objekts verändern kann oder beibehält. Weiters hat der Vereinfachungsoperator oft Auswirkung auf die Effizienz des Verfahrens. In der dritten Spalte ist das Ziel des Verfahrens angegeben. Dies entscheidet weitgehend über den Einsatzbereich, also ob eine starke Vereinfachung, eine hohe Qualität oder eine schnelle Berechnungszeit im Vordergrund stehen. In der letzten Spalte ist angeführt, ob sich das Verfahren für die Berechnung einer Vereinfachung in Echtzeit eignet.

Name	Operator	Fokus	Echtzeitanwendung möglich
Schrittweise Dreiecksnetze	Kantenreduktion	Geringe Vertexzahl	Ja
Quadrik-Fehlermetrik	Vertexpaar-verschmelzung	Bildqualität	Ja ([GDG11])
Vertexgruppierung	Zellenreduktion	Berechnungszeit der Vereinfachung	Ja ([DT07])
Vertexdezimierung	Vertexeliminierung	Bildqualität	Nein
Bildbasierte Vereinfachung	Kantenreduktion	Bildqualität	Nein
Papageorgiou und Platis	Kantenreduktion	Bildqualität/ Berechnungszeit	Nein
Shontz und Nistro	Kantenreduktion	Bildqualität/ Berechnungszeit	Nein
Vereinfachung mittels Morton Integralen	Gruppieren von Vertizes	Berechnungszeit	Ja
Vereinfachung durch Reduktion von Teilbäumen	Kantenreduktion	Bildqualität/ Berechnungszeit	Ja

Tabelle 2.2: Vergleich der präsentierten Verfahren

2.5 Zusammenfassung

Dieses Kapitel erläutert grundlegende Begriffe sowie Konzepte, die für das Verständnis von Vereinfachungen sowie Vereinfachungsverfahren von Dreiecksnetzen von Bedeutung sind. Ansätze von Systemen zur Steuerung des Detailgrades, die in der Computergraphik weit verbreitet Einsatz finden, um den Aufwand für die Darstellung von Szenen zu regulieren, werden diskutiert.

Grundlage für eine Vereinfachung sind Vereinfachungsoperatoren. In diesem Kapitel werden mehrere Vereinfachungsoperatoren präsentiert, die genutzt werden können, um Elemente aus einem Dreiecksnetz zu entfernen. Ein Vergleich der Operatoren wird gezogen. Diese Operatoren finden in einer Reihe von Verfahren Anwendung, die sich durch die Qualität der Vereinfachung, die Berechnungszeit und die eingesetzten Vereinfachungsoperatoren unterscheiden. Nach Diskussion der Operatoren wird eine Menge von Verfahren zur Vereinfachung von Dreiecksnetzen präsentiert, ihre Vor- und Nachteile dargelegt und Erweiterungen sowie Verbesserungen dieser Verfahren miteinbezogen und ihre Anwendungsgebiete vorgestellt.

Im nächsten Kapitel wird ein neuartiges Verfahren zur Vereinfachung von Dreiecksnetzen präsentiert, das die Vereinfachung dynamisch zur Laufzeit errechnet und nicht auf vorberechneten Operationen beruht. Ziel bei der Entwicklung des Verfahrens war eine parallele Ausführung, die eine effiziente Implementierung auf modernen GPUs ermöglicht und so die Laufzeit für die Berechnung verkürzt.

Kapitel 3

Vertexgruppeneliminierung

Dieses Kapitel beschreibt das Verfahren der Vertexgruppeneliminierung, das im Zuge dieser Arbeit entwickelt wurde. Die Vertexgruppeneliminierung berechnet die Vereinfachung eines Dreiecksnetzes dynamisch zur Laufzeit. Das bedeutet, dass vor jeder Bilderzeugung die Vereinfachung erneut berechnet wird. Dieses Vorgehen ermöglicht es, die Position sowie die Blickrichtung des Betrachters in Bezug auf das Objekt einfließen zu lassen und bei der Auswahl der zu entfernenden Details zu berücksichtigen. Eine Minimierung der visuellen Artefakte, die vom Betrachter wahrgenommen werden können, ist die Folge [Lue01].

Für die Durchführung der Vereinfachung dynamisch zur Laufzeit ist eine effiziente Berechnung nötig, um die Berechnungszeit zu minimieren. Die Vertexgruppeneliminierung wurde entwickelt, um einen hohen Grad an Parallelität zu erlauben und moderne Prozessoren und insbesondere GPUs effizient ausnutzen zu können. Die resultierende Beschleunigung der Berechnung erlaubt, dass die aufwändigen Arbeitsschritte zur Laufzeit durchgeführt werden können. Neben der Auswahl der zu entfernenden Details und der Vereinfachung des Dreiecksnetzes stellt der Erhalt der Netzintegrität eine Herausforderung dar (Vermeidung von Faltungen und inkonsistenten Topologien, siehe Unterabschnitt 2.3.1).

Dieses Kapitel beschäftigt sich mit folgenden Bereichen:

- **1. Überblick über den Ablauf der Vertexgruppeneliminierung** Bietet einen Überblick über den Ablauf des Verfahrens.

- **2. Allgemeines** Beschreibt allgemeine Informationen, Probleme und Ansätze der Vertexgruppeneliminierung.

- **3. Klassifizierung** Beschäftigt sich mit den Teilschritten der Klassifizierung und Reklassifizierung.

- **4. Verschiebung** Beschreibt den Teilschritt der Verschiebung.

3.1 Überblick über den Ablauf der Vertexgruppeneliminierung

Die Vertexgruppeneliminierung führt mehrere Berechnungsschritte aus, um die Vereinfachung durchzuführen (siehe Abbildung 3.1). Als Eingabe kommt ein Dreiecksnetz zum Einsatz. Dieses speichert eine Menge an Vertizes, die neben der Position auch einen Oberflächennormalvektor in diesem Punkt enthalten. Des Weiteren ist eine Liste von Dreiecken vorhanden, die mit den Vertizes gebildet werden.

- **1. Klassifizierung** Der erste Teilschritt besteht in einer Analyse aller Vertizes des gegebenen Dreiecksnetzes. Für jeden Punkt wird bestimmt, ob er beibehalten oder in Bezug auf seine gegenwärtigen Nachbarn für die Vereinfachung aus dem Netz entfernt werden soll. Jedem Vertex wird ein Fehlerwert zugeordnet, der mit einer vom Benutzer definierten Toleranzgrenze verglichen wird. Das Ergebnis dieser Klassifizierung ist eine Menge an Vertizes, die in den weiteren Schritten des Verfahrens aus dem Dreiecksnetz entfernt werden soll und somit als Ausgangsbasis für die Vereinfachungsoperationen dient.

- **2. Bestimmung der Verschiebungskandidaten** Um einen in Schritt 1 für ein Entfernen markierten Vertex aus dem Dreiecksnetz zu löschen, wird er auf die Position eines beizubehaltenden Nachbarn verschoben. Diese Operationen finden ausschließlich entlang bestehender Kanten des Dreiecksnetzes statt. In diesem Teilschritt (Schritt 2) werden alle jene zu entfernenden Vertizes bestimmt, die eine Kante zu einem beizubehaltenden Nachbarn besitzen - diese Vertizes werden in weiterer Folge als Verschiebungskandidaten bezeichnet.

- **3. Verschiebung und Bestimmung neuer Verschiebungskandidaten** In Schritt 3 werden alle gefundenen Verschiebungskandidaten gleichzeitig bearbeitet und - sofern unter Erhaltung der Netzintegrität möglich - aus dem Dreiecksnetz entfernt. Durch die veränderte Position der modifizierten Vertizes werden auch die Kanten des Dreiecksnetzes manipuliert. Somit ergibt sich eine neue Menge an Verschiebungskandidaten. Ist diese Menge leer, so ist die Vereinfachung abgeschlossen.

- **4. Reklassifizierung** In Schritt 4 werden die veränderten Nachbarn der beibehaltenen Vertizes neu klassifiziert, um zu bestimmen, ob nach der Veränderung der Topologie die Einstufung der aktualisierten Verschiebungskandidaten noch gültig

ist oder ob die Vertizes durch die veränderte Topologie beibehalten werden sollen. Diese Reklassifizierung erlaubt eine dynamische Anpassung der Zahl der Vertizes des finalen, vereinfachten Dreiecksnetzes.

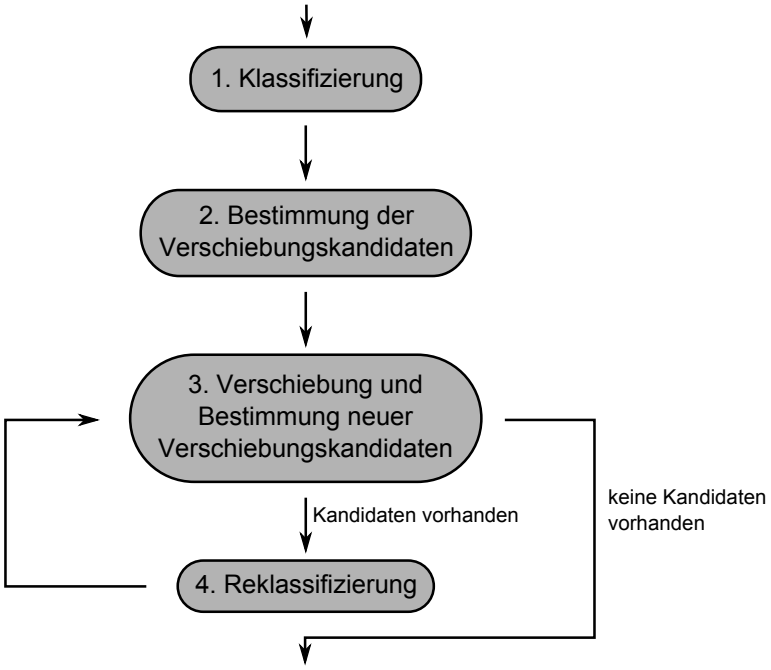

Abbildung 3.1: Ablauf der Vertexgruppeneliminierung

Die Schritte 3 (Verschiebung und Bestimmung neuer Verschiebungskandidaten) und 4 (Reklassifizierung) werden iterativ wiederholt, bis alle initial für ein Entfernen markierten Vertizes bearbeitet und somit entweder verschoben oder nachtragend für ein Beibehalten markiert wurden. Abbildung 3.1 zeigt den Ablauf des Verfahrens und den Zusammenhang der einzelnen Teilschritte.

Im nächsten Abschnitt werden allgemeine Informationen über den Ablauf der Vertexgruppeneliminierung beschrieben.

3.2 Allgemeines

Um eine Reduktion der zur Laufzeit notwendigen Operationen zu erzielen, wird der erste Arbeitsschritt - die Klassifizierung (Abbildung 3.1) - in zwei Teile zerlegt:

- **1. Vorberechnung** Berechnungen, die ausschließlich auf statischen Vertexdaten beruhen, werden in einem Vorberechnungsschritt durchgeführt und die Ergebnisse per Vertex abgespeichert.

- **2. Berechnung zur Laufzeit** Abschluss der Vertexanalyse basierend auf den Daten der Vorberechnung und Ermittlung der initialen Klassifizierung.

Im ersten Teilschritt - der Vorberechnung - wird der Oberflächenverlauf eines Objekts analysiert. Jeder Vertex eines Dreiecksnetzes speichert durch seine Position eine Ausprägung des Objekts. Mit Hilfe einer Fehlermetrik kann für jeden Vertex ein Fehlerwert errechnet werden. Dieser Fehlerwert (Vertexfehler) gibt eine statische Angabe über einen Vertex eines Dreiecksnetzes, die besagt, wie stark die Ausprägung der Oberfläche durch diesen beeinflusst wird. Der Fehlerwert dient im weiteren Verlauf als Ausgangspunkt für Berechnungen.

Der zweite Teilschritt der Klassifizierung ist die Berechnung zur Laufzeit. Die Daten des Betrachters (Blickrichtung und Distanz zwischen Betrachter und Objekt) sind hier bekannt. Der vorberechnete, statische Fehlerwert wird anhand der Position des Betrachters manipuliert. Aufgrund des Resultats dieser Berechnung wird jeder Vertex des Dreiecksnetzes bewertet. Dies führt zu einer Klassifizierung, die für jeden Vertex des Dreiecksnetzes besagt, ob er beibehalten und somit in der finalen, vereinfachten Version des Dreiecksnetzes enthalten sein oder ob er im weiteren Verlauf durch Operationen zur Reduktion der Vertizes aus dem Dreiecksnetz entfernt werden soll.

Im Verschiebungsschritt (Schritt 3 in Abbildung 3.1) sollen Vertizes, die nicht für ein Beibehalten markiert sind, im Verlauf einer Reihe von Iterationsschritten aus dem Dreiecksnetz entfernt werden. Das Löschen von Vertizes resultiert in einer Menge von Löchern in dem darzustellenden Objekt [SZL92]. Neue Dreiecke werden für das Netz generiert, so dass eine geschlossene Oberfläche wiederhergestellt wird [CG09]. Für die Berechnung dieser Triangulierung greift die Vertexgruppeneliminierung auf die Idee der Kantenreduktion zurück (siehe Unterabschnitt 2.3.1) und verschiebt Vertizes, die im vorigen Schritt für ein Entfernen markiert wurden, durch eine eingeschränkte Kantenreduktion (Unterabschnitt 2.3.2) auf die Position eines bestehen bleibenden, benachbarten Vertex.

Während ein weit verbreiteter Ansatz bei Verwendung von Kantenreduktionen darauf beruht, Kanten zu finden, die durch einen Vertex ersetzt werden [Hop97], werden im Verschiebungsschritt zu entfernende Vertizes gesucht. Jede Kante zu einem bestehen bleibenden Vertex wird als mögliche eingeschränkte Kantenreduktion mit dem benachbarten Vertex als Ziel der Verschiebung angesehen. Um die Berechnungszeit zu minimieren, werden alle möglichen eingeschränkten Kantenreduktionen parallel vollzogen. Bei dem hier vorgestellten Ansatz werden nur Verschiebungen auf die Position von nicht zu entfernenden Punkten durchgeführt. Folglich ist eine Reihe von Iterationsschritten vonnöten. In jedem Schritt werden alle zu entfernenden Vertizes, die eine Kante zu einem beibehaltenen Punkt besitzen, bearbeitet.

Eine Verschiebung schließt entstandene Löcher in einem Dreiecksnetz bzw. verändert bestehende Dreiecke, so dass diese keine zu entfernenden Vertizes mehr enthalten. Das erzeugt die Problematik, dass keine Dreiecke entstehen dürfen, die eine Verletzung der Integrität der Oberfläche des dargestellten Objekts verursachen (siehe Unterab-

schnitt 2.3.1).

Es wird versucht, eine Ersatzposition, die keine Fehler im Dreiecksnetz sowie möglichst geringe optische Auswirkungen verursacht, zu ermitteln und den Vertex auf eine gefundene Position zu verschieben. Bei Verschiebungen werden die Kanten zu den benachbarten Vertizes entsprechend angepasst.

Durch die Verschiebungen von Vertizes werden Kanten modifiziert, und die beibehaltenen Vertizes bekommen andere Vertizes als Nachbarn. Der initial errechnete Fehlerwert beruht ausschließlich auf den statischen Daten eines Vertex und seiner Nachbarn. Es ist zu diesem Zeitpunkt nötig, den Fehlerwert für die veränderten Nachbarn neu zu berechnen.

Aufgrund der statischen Fehlermetrik kann eine große Anzahl benachbarter Vertizes für ein Entfernen markiert werden. Auf der Oberfläche des Objekts können somit große Bereiche ohne Details entstehen. Durch die Neuberechnung des Vertexfehlers im Reklassifizierungsschritt (Schritt 4 in Abbildung 3.1) wird verifiziert, ob die initiale Klassifizierung auch in Bezug auf die modifizierten Nachbarvertizes noch Gültigkeit besitzt oder ob Nachbarn erneut klassifiziert werden müssen. Es ist somit einerseits möglich, den Detailgrad des finalen Netzes dynamisch zu bestimmen, andererseits erlaubt dieses Verfahren eine gleichmäßigere Triangulierung.

Abbildung 3.2 zeigt ein Beispiel für die Unterschiede zwischen einer Reklassifizierung (rechts) und einer Entfernung aller markierten Vertizes (Mitte). Auf der linken Seite ist das ursprüngliche Dreiecksnetz dargestellt. In der Mitte wurden mehr Vertizes entfernt, wodurch eine fächerförmige Triangulierung aufgetreten ist. Im Vergleich dazu ist auf der rechten Seite eine gleichmäßigere Triangulierung erkennbar.

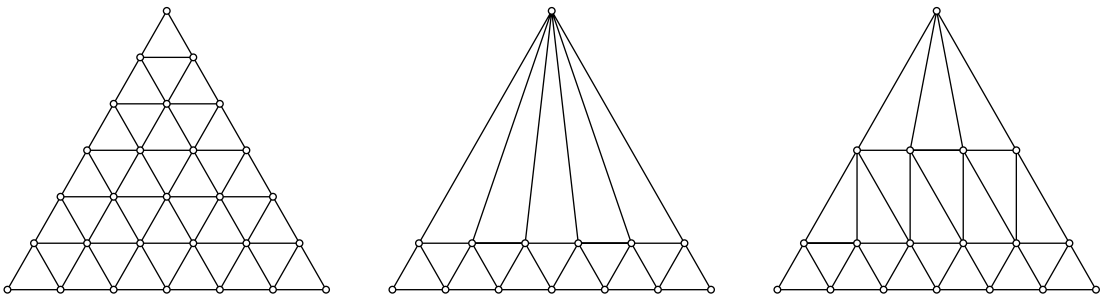

Abbildung 3.2: Vergleich einer Vereinfachung eines Dreiecksnetzes (links) durch statische Klassifizierung (Mitte) mit jener der Reklassifizierung (rechts)

Im Anschluss werden die Teilschritte der Vertexgruppeneliminierung im Detail beschrieben, beginnend mit der Klassifizierung der Vertizes.

3.3 Klassifizierung

Schritt 1 (in Abbildung 3.1) analysiert alle Vertizes eines Dreiecksnetzes, um zu bestimmen, wie stark ein Vertex zu den Ausprägungen der darzustellenden Oberfläche beiträgt und wie sich das Entfernen des Vertex auswirkt. Aus den Ergebnissen dieser Analyse wird eine Klassifizierung für die Punkte ermittelt. Ein Vertex kann entweder beibehalten werden und ist im Resultat der Vereinfachungsoperationen enthalten oder wird für ein Löschen aus dem Netz markiert.

Die Auswahl beruht auf den statischen Vertexdaten eines Dreiecksnetzes, aus denen für jeden Vertex ein Fehlerwert ermittelt wird. Zur Laufzeit wird aus Position und Blickrichtung des Betrachters ein Skalierungsfaktor errechnet, aus dem gemeinsam mit dem Vertexfehler die Klassifizierung bestimmt wird.

Ein Vertex speichert eine Position und bildet gemeinsam mit den benachbarten Vertizes eine Menge an Dreiecken. Wird ein Punkt gelöscht, so werden gleichzeitig auch die Dreiecke, welche diesen Punkt enthalten, aus dem Netz entfernt, da sie nicht mehr gebildet werden können [CRS96]. Die Oberfläche wird aus den bestehen bleibenden Vertizes neu aufgebaut. Dabei gehen die in dem entfernten Vertex gespeicherten Ausprägungen verloren.

Abbildung 3.3 illustriert diesen Verlust. Wird ein Vertex (V) entfernt, so verläuft die Oberfläche durch die beiden Nachbarvertizes (N_1, N_2) entlang der gestrichelten Linie. Die verlorene Ausprägung a ist in Abbildung 3.3 ebenfalls gekennzeichnet. Der erste Teil der Vertexklassifizierung beschäftigt sich damit, eine Fehlermetrik zu finden, mit der sich bestimmen lässt, wie stark der Verlust der Ausprägungen ausfällt, um daraus einen Fehlerwert zu errechnen, der für jeden Vertex gespeichert wird.

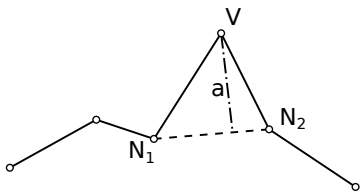

Abbildung 3.3: Verlust von Ausprägungen durch das Entfernen von Vertizes

Dieser Fehler wird bei der statischen Berechnung per Vertex ermittelt und berücksichtigt nur die lokal begrenzten Auswirkungen des Entfernens eines einzelnen Vertex in Bezug auf seine Nachbarn. Zu diesem Zeitpunkt ist nicht bestimmt, welche Vertizes tatsächlich für ein Entfernen markiert werden. Die Daten des Betrachters sind nicht bekannt. Es ist nicht vorhersagbar, wie stark die Auswirkungen der Vereinfachung wahrgenommen werden können. In Bereichen, in denen ein Objekt nur wenige Ausprägungen besitzt, kann eine hohe Anzahl an benachbarten Vertizes für ein Entfernen bestimmt

werden. Die Skalierung des Fehlers hat bei großen Distanzen zum Betrachter ein ähnliches Phänomen zur Folge und kann eine Mehrheit der Vertizes des Dreiecksnetzes für eine Entfernungsoperation selektieren. Dies hat zum einen den Nachteil, dass sehr große Flächen erzeugt werden, zum anderen wird die Zahl der Vertizes mit Kanten zu bestehen bleibenden Vertizes und folglich die Menge an möglichen Verschiebungen reduziert. Die Parallelität des Verfahrens wird eingeschränkt. In Extremfällen kann die Zahl der beizubehaltenden Vertizes so gering sein, dass es zu einer Serialisierung der Vereinfachungsoperationen kommt und das Verfahren nur mehr sehr ineffizient funktioniert. Überdies ist eine Vereinfachung nicht mehr durchführbar, wenn alle Vertizes entfernt werden sollen.

Um dieses Problem zu vermeiden, muss stets eine Mindestanzahl an Vertizes im Dreiecksnetz verbleiben. Sie dienen zum einen als Zielpunkt für Vertexverschiebungen, um die Funktionsfähigkeit des Verfahrens zu ermöglichen, indem stets potentielle Ziele für Verschiebungen zur Verfügung stehen, und zum anderen als Hilfspunkte, um die Parallelität zu erhöhen.

Die Idee ist, die Fehlerwerte einer Reihe von Vertizes künstlich so zu vergrößern, dass diese Punkte bei der Klassifizierung nie für ein Entfernen markiert werden. Dieser Ansatz lässt sich noch erweitern, indem die Fehlerwerte von Vertizes verschieden stark manipuliert werden: Eine geringe Menge an Vertizes erhält einen sehr großen Vertexfehler. Dies garantiert, dass in jedem Fall mögliche Ziele für Vertexverschiebungen zur Verfügung stehen und die Funktionsfähigkeit des Verfahrens gewährleistet ist. Bei einer anderen Menge an Vertizes wird der Fehler weniger stark erhöht, so dass diese bei steigender Entfernung zum Betrachter länger beibehalten werden, ihr Verbleib im Dreiecksnetz jedoch nicht garantiert ist.

Auf diese Weise lässt sich eine Reihe von „Auflösungsstufen“ für ein Dreiecksnetz festlegen. Für jede Stufe wird eine Menge an Vertizes bestimmt, deren Vertexfehler in Abhängigkeit von der Stufe angepasst wird. Durch die Skalierung des Fehlerwertes mit Hilfe der Distanz zum Betrachter bei der Auswahl der zu entfernenden Vertizes ist es möglich, eine passende Auflösungsstufe zu „selektieren“ und eine Mindestauflösung für bestimmte Entfernungen vom Betrachter festzulegen.

Abbildung 3.4 präsentiert einen Vergleich zwischen der Klassifizierung von Vertizes mit bzw. ohne Manipulation des Vertexfehlers. Die linke Seite zeigt das Beispiel eines Dreiecksnetzes, bei dem nur die Randvertizes verbleiben sollen. Durch die Manipulation der Fehlerwerte auf der rechten Seite sind zusätzliche, beizubehaltende Vertizes geschaffen worden, die als Ziele für Verschiebungen zur Verfügung stehen. Während auf der linken Seite nur jene Vertizes verschoben werden können, die eine Kante zu einem Randvertex besitzen, ist es auf der rechten Seite möglich, alle nicht markierten (zu entfernenden) Vertizes in einem Schritt zu verschieben.

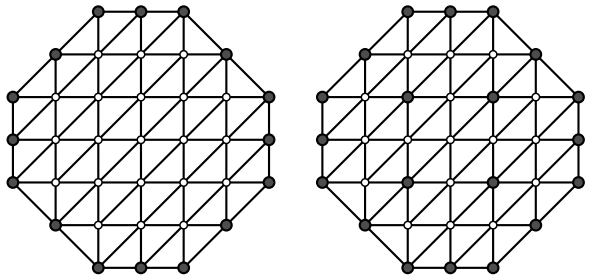

Abbildung 3.4: Beibehaltene Vertizes ohne und mit Manipulation des Fehlerwertes

Die Ergebnisse dieser Berechnung werden für jeden Vertex abgespeichert und können zur Laufzeit als Ausgangspunkt für die Klassifizierung herangezogen werden.

Um das Bestimmen von zu löschenden Vertizes zu verbessern, fließen Position und Blickrichtung des Betrachters in die erforderlichen Berechnungen ein. Somit lässt sich die sichtbare Auswirkung des Entfernens eines Vertex genauer festlegen. Bei einem Objekt, das sich in sehr großer Distanz zum Betrachter befindet, werden weniger Details wahrgenommen und sie können somit aus dem Dreiecksnetz entfernt werden. Im Gegenzug dazu benötigt ein Objekt nahe an der Position des Betrachters viele Details - nur wenige Vereinfachungen sollten vorgenommen werden.

Ein weiterer Faktor ist der Blickwinkel des Betrachters. Ausprägungen entlang der Silhouette eines Objekts werden wesentlich deutlicher wahrgenommen als Höhenunterschiede von Flächen und Details, auf die man direkt von oben blickt [ESV99a]. Dementsprechend sind Vertexfehler entlang von sichtbaren Rändern stärker zu gewichten und bei der Klassifizierung zu berücksichtigen.

Zur initialen Klassifizierung kommt die Reklassifizierung von Vertizes als weitere Anforderung an die Fehlermetriken. Vertizes werden in Bezug auf Nachbarn und die Position des Betrachters bewertet. Als Folge kann eine große Anzahl von benachbarten Vertizes für ein Entfernen markiert werden. Jede Verschiebung erzeugt eine Abweichung von der ursprünglichen Oberfläche. Werden alle diese Vertizes gelöscht, kann aus der Summe der Verschiebungen ein großer, sichtbarer Unterschied bei der Darstellung des Objekts entstehen. Um dieses Phänomen zu vermeiden, können Vertizes, die für ein Entfernen markiert wurden, während der dynamischen Berechnung reklassifiziert werden. Wird eine eingeschränkte Kantenreduktion durchgeführt, findet eine Manipulation von Kanten statt und dem Verschiebungsziel werden neue Nachbarvertizes anstelle des gelöschten Vertex zugeordnet [DDFM$^+$05]. Diese manipulierten Kanten verändern den Vertexfehler von zu entfernenden Punkten und invalidieren die initiale Klassifizierung.

Nach jeder Verschiebung eines Vertex wird ein neuer Vertexfehler für die modifizierten Nachbarvertizes berechnet. Es ist zu beachten, dass die Ergebnisse dieser Reklassifizierungsoperation mit den Resultaten der initialen Bewertung konvergieren sollten. Andernfalls könnte ein Reklassifizierungsschritt eine unnötige Neueinstufung eines Vertex

verursachen. Um dies zu vermeiden, kommt in der Klassifizierung und Reklassifizierung dieselbe Fehlermetrik zum Einsatz. Aus der hohen Anzahl an notwendigen Reklassifizierungsoperationen zur Laufzeit ergibt sich, dass der Zeitaufwand für die Ermittlung des Vertexfehlers in Grenzen gehalten werden sollte. Eine Abhängigkeit von einer großen Menge an Daten und ein hoher Berechnungsaufwand sind folglich zu vermeiden.

Im Anschluss werden die Komponenten und Einsatzgebiete der Fehlermetrik, die bei Klassifizierung (Schritt 1 in Abbildung 3.1) und Reklassifizierung (Schritt 4 in Abbildung 3.1) zum Einsatz kommt, vorgestellt. Diese Bereiche sind:

- **1. Initiale Fehlermetrik** Berechnung des per-Vertex Fehlerwertes aus den statischen Vertexdaten
- **2. Selektion mittels regelmäßigem Gitter** Manipulation der Fehlerwerte anhand eines regelmäßigen Gitters
- **3. Kameraabhängigkeit** Einfluss der Daten des Betrachters bei der Klassifizierung
- **4. Reklassifizierung** Berechnung und Einsatz der Fehlerwerte bei der Reklassifizierung

3.3.1 Initiale Fehlermetrik

Die initiale Fehlermetrik, die im Rahmen dieser Arbeit zum Einsatz kommt, ermittelt für jeden Vertex einen Fehlerwert. Es ist zu berücksichtigen, dass sie nicht nur bei der initialen, statischen Klassifizierung berechnet wird, sondern auch bei der erneuten Bewertung von Vertizes nach Verschiebungsoperationen. Zu diesem Zeitpunkt wird potentiell nur ein Teil der Nachbarvertizes beibehalten, während der andere für ein Entfernen markiert ist und somit keine relevanten Daten für die Reklassifizierung liefert. Die Fehlermetrik muss daher in der Lage sein, auch auf einer Teilmenge der Nachbarvertizes zu operieren. Dementsprechend soll eine Fehlermetrik gefunden werden, die folgende Anforderungen erfüllt:

- Zuweisen eines einzelnen Fehlerwertes zu jedem Vertex
- Abhängigkeit ausschließlich von den Nachbarn
- effiziente Berechenbarkeit

Eine Oberfläche wird repräsentiert durch eine Menge von Vertizes. Jeder Vertex speichert neben diversen Daten eine Position im Raum und den Normalvektor der Tangentialebene der Oberfläche in diesem Punkt. Durch die beiden Werte wird der Verlauf der Oberfläche definiert. Das Entfernen eines Vertex aus dem Dreiecksnetz hat zur Folge, dass diese gespeicherten Daten nicht zur Verfügung stehen und die Oberfläche ausschließlich aus den Nachbarvertizes des entfernten Vertex gebildet wird. Der Oberflächenverlauf verliert somit an Ausprägung.

Der errechnete Fehler für einen Vertex repräsentiert diesen Verlust der Ausprägung. Allerdings wird bei der Berechnung des Fehlerwertes für die initiale Klassifizierung ausschließlich auf die Unterschiede von Vertizes in Bezug auf ihre Nachbarn Rücksicht genommen. Etwaige Detailverluste durch die Entfernung mehrerer benachbarter Vertizes werden bei der Vorberechnung des Vertexfehlers ignoriert.

Der Ansatz für die Vorberechnung des Fehlers basiert auf der Annahme, dass der für einen Vertex gespeicherte Normalvektor den Verlauf (also die Tangentialebene) der Oberfläche in diesem Punkt angibt. Benachbarte Vertizes stellen eine Abweichung von dieser Tangentialebene dar und definieren somit den Verlauf der Oberfläche.

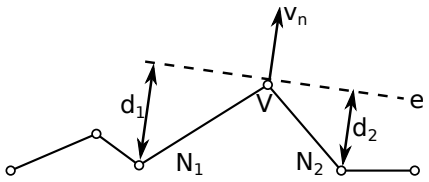

Abbildung 3.5: Fehlermetrik für die Bestimmung des Vertexfehlers

Abbildung 3.5 zeigt diese Idee auf. Die gekennzeichnete Tangentialebene (e), die durch den im Vertex (V) gespeicherten Normalvektor (v_n) definiert wird, enthält den Oberflächenverlauf im Punkt des Vertex. Die Nachbarn (N_1, N_2) verbleiben bei einer Entfernung dieses Vertex und bilden die neue Ausprägung der Oberfläche.

Für jeden Vertex V eines Dreiecksnetzes wird aus der Position und dem dazugehörigen Normalvektor die Tangentialebene errechnet. Anschließend wird die Menge $N(V) = \{P_1, P_2, ...P_n\}$ bestimmt. Sie enthält alle jene Vertizes, die gemeinsam mit V ein Dreieck bilden. Für jeden Vertex in $N(V)$ wird der Normalabstand (d_1, d_2 in Abbildung 3.5) zwischen der Position im Raum und der Tangentialebene von V ermittelt. Diese Distanz ist die Abweichung des Nachbarvertex von einem durch den Normalvektor angegebenen Oberflächenverlauf. Je größer dieser Abstand ausfällt, desto stärker ist der Einfluss von V auf den Oberflächenverlauf. Jeder Vertex in $N(V)$ enthält eine Abweichung von dieser Tangentialebene und folglich auch eine Abweichung vom Verlauf der Oberfläche. Der Fehlerwert, der für einen Vertex gespeichert wird, ist dementsprechend der Durchschnitt aller so ermittelten Fehlerwerte der benachbarten Vertizes.

Gegeben für jeden Vertex V sind die Position v und der Normalvektor v_n. Aus diesen Daten wird die Tangentialebene der Oberfläche in V aufgestellt.

$$\begin{aligned} ax + by + cz - d = 0 \\ t = a, b, c, d \end{aligned} \tag{3.1}$$

Für jeden benachbarten Vertex P mit der Position $p = [p_x, p_y, p_z, 1]$ kann der Normalabstand zu dieser Tangentialebene berechnet werden, der als Vertexfehler $\triangle(V, P)$ dient. Dieser Fehler wird quadriert, um negative Vertexfehler zu vermeiden und einzelne, große Fehler stärker zu gewichten.

$$\triangle(V, P) = (t \bullet p)^2 \tag{3.2}$$

Die so errechneten Werte repräsentieren die Abweichungen zwischen dem Oberflächenverlauf in einem Vertex und seinen Nachbarn. Der Wert der finalen Fehlermetrik $e(V)$ für einen Vertex wird definiert als der Mittelwert aller Abweichungen eines Vertex in Bezug auf seine Nachbarn. Für jeden Vertex V existiert eine Menge an Nachbarn $N(V) = \{P_1, P_2, ...P_n\}$. Jeder Punkt P_i in $N(V)$ besitzt eine Kante zu V.

$$e(V) = \frac{\sum_{i=1}^{n}(\triangle(V, P_i))}{n} \tag{3.3}$$

Der Fehlerwert repräsentiert den Fehler im Dreiecksnetz, den die Entfernung des Vertex verursacht. Dieser wird für jeden Vertex gespeichert und dient als Ausgangspunkt für weitere Berechnungen.

Die Eigenschaft, den Vertexfehler ausschließlich in Bezug auf Nachbarvertizes zu ermitteln, erlaubt eine effiziente Berechnung und ermöglicht durch die eingeschränkte Abhängigkeit von Daten auch eine schnelle Neuberechnung zur Laufzeit.

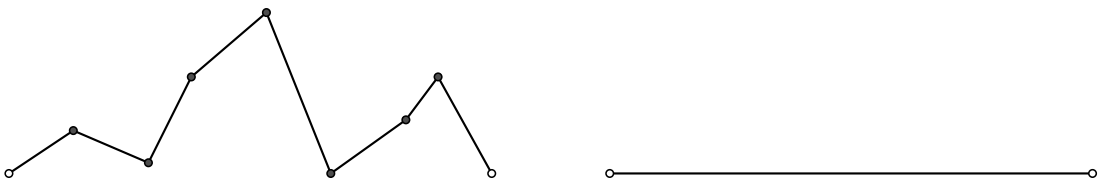

Abbildung 3.6: Entfernen von benachbarten Vertizes führt zu größeren Flächen

Dieser Vorteil wird jedoch durch das Phänomen eingeschränkt, dass viele benachbarte Vertizes entfernt werden und große, ebene Flächen eine zuvor unruhige Oberfläche des

Objekts ersetzen können. Abbildung 3.6 zeigt ein Beispiel hierfür. Dies wird hervorgerufen, da der Vertexfehler benachbarter Vertizes bei der initialen Klassifizierung keine Berücksichtigung findet. So kann eine Menge an Vertizes mit ähnlichem Vertexfehler entstehen, die somit alle bei entsprechendem Schwellenwert für ein Entfernen markiert werden.

Im nächsten Abschnitt wird die Selektion mittels regelmäßigem Gitter präsentiert, die dieses Problem vermeidet.

3.3.2 Selektion mittels regelmäßigem Gitter

Um die Funktionsfähigkeit der Vereinfachung zu gewährleisten, muss garantiert werden, dass bei der Auswahl zu entfernender Vertizes immer ein oder mehrere Vertizes verbleiben, die als Ziele für Verschiebungsoperationen zum Einsatz kommen. Eine weitere Überlegung betrifft die Anzahl der benötigten Iterationsschritte, um alle zu entfernenden Vertizes zu behandeln. Im Zuge der Vertexgruppeneliminierung wird eine eingeschränkte Kantenreduktion nur auf Kanten zwischen einem beizubehaltenden und einem zu entfernenden Vertex ausgeführt. Ergibt die Klassifizierung eine geringe Menge von verbleibenden Vertizes, so steht einer großen Zahl an zu entfernenden Vertizes eine geringe Menge an Verschiebungszielen gegenüber. Dies kann in einer kleinen Menge von eingeschränkten Kantenreduktionen, die gleichzeitig in einem Iterationsschritt ausgeführt werden können, resultieren und somit eine hohe Anzahl an Iterationsschritten nach sich ziehen, was zu einer Einschränkung der Parallelität führt.

Der Verbleib einer Mindestanzahl von Vertizes im Dreiecksnetz muss garantiert werden, um die Funktionalität der Vertexgruppeneliminierung zu gewährleisten. Außerdem können zusätzliche Verschiebungsziele geschaffen werden, um die Anzahl der benötigten Iterationsschritte zu senken.

Ein weit verbreitetes Verfahren zur Vereinfachung von Dreiecksnetzen projiziert ein 3-dimensionales Gitter über das Netz ([RB92] [DT07] [SW03], siehe Unterabschnitt 2.3.5). Alle Vertizes innerhalb einer Zelle dieses Gitters werden auf einen Punkt reduziert. Durch die Auflösung des Gitters kann der Detailgrad des dargestellten Objekts festgelegt werden. Diese Idee wird hier aufgegriffen. Anstatt Vertizes innerhalb eines Bereichs zusammenzufassen, kommt das Gitter zur Anwendung, um eine Menge D an Vertizes zu finden, deren gespeicherter Vertexfehler künstlich erhöht wird. Bei einer Teilmenge $D_0 \subseteq D$ wird ein so großer Fehlerwert eingesetzt, dass sie immer beizubehalten ist. Die Manipulation des Fehlers der Vertizes in $D \backslash D_0$ fällt weniger stark aus. Die Wahrscheinlichkeit, dass sie für ein Entfernen markiert werden, wird somit reduziert.

Bei der Auswahl der zu entfernenden Vertizes fließt die Distanz zum Betrachter als Skalierungsfaktor ein. Um dies zu berücksichtigen, findet die Manipulation des Vertexfehlers gestaffelt statt, so dass bei größeren Entfernungen die Zahl der beizubehaltenden

Vertizes mit manipuliertem Fehler sinkt und keine unnötig große Menge an verbleibenden Punkten erzeugt wird.

Die Vertizes in D werden folgendermaßen ermittelt:

1. Bestimmen des Begrenzungsvolumens des Objekts und Erstellung eines regelmäßigen Gitters G_0 mit den Knotenpunkten $GP(G_0) = \{P_0^1, P_0^2, ...P_0^n\}$.

2. Erzeugen von zusätzlichen Gitterstufen G_i durch Halbierung der Abstände $d(G_i)$ zwischen den Knotenpunkten. Somit gilt: $d(G_i) = \frac{d(G_{i-1})}{2}$ (i > 0, getrennt für jede Dimension, Anzahl der Stufen vom Benutzer vorgegeben).

3. Iteration über alle G_i und Zuweisung eines um P_i^j zentrierten Quaders $B(P_i^j)$ zu jedem Punkt P_i^j in $GP(G_i)$. Die Seitenlängen von $B(P_i^j)$ sind identisch zu $d(G_i)$.

4. Auffinden der Menge aller Vertizes $V(P_i^j) = \{V_1, V_2, ..., V_n\}$ innerhalb des Volumens $B(P_i^j)$ und Verwerfung aller P_i^j mit $|V(P_i^j)| = 0$.

5. Selektion eines Vertex $C(P_i^j)$ aus $V(P_i^j)$ für alle verbleibenden P_i^j. Hierfür wird für jeden Vertex V_k in $V(P_i^j)$ der gespeicherte Vertexfehler mittels der Distanz zwischen P_i^j und V_k gewichtet und der Vertex mit dem maximalen gewichteten Fehler gewählt.

6. Manipulation des Vertexfehlers aller $C(P_i^j)$ mittels eines Wertes der Stufe G_i (jeder Gitterstufe wird vom Benutzer ein Wert zugewiesen).

Abbildung 3.7 zeigt die Idee hinter dieser Technik. In diesem Beispiel wird ein Gitter über das dargestellte Dreiecksnetz gelegt. Für jeden Knoten einer Stufe wird ein ihm nahegelegener Vertex des Dreiecksnetzes ausgewählt. Wie im Beispiel in Abbildung 3.7 gezeigt müssen die Punkte des Gitters und des Dreiecksnetzes nicht übereinstimmen und Zellen ohne Vertizes können existieren. Außerdem können unter Umständen nicht für alle Punkte des Gitters Vertizes ermittelt werden.

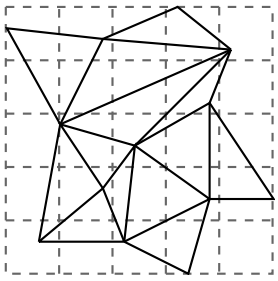

Abbildung 3.7: Überlagerung des Dreiecksnetzes mit einem Gitter

Zunächst sind die äußeren Grenzen, innerhalb derer das Gitter berechnet werden soll, zu ermitteln. Hier kommt das achsparallele Begrenzungsvolumen des darzustellenden Objekts zum Einsatz. Anschließend wird eine Anzahl von Punkten mit regelmäßigen Abständen im Bereich des Begrenzungsvolumens errechnet. Die Anzahl der Punkte ist für jede Dimension identisch.

In der gröbsten Stufe G_0 ist eine Menge an Punkten $GP(G_0)$ vorhanden. Bei weiteren Auflösungsstufen werden zusätzliche Punkte in das existierende regelmäßige Gitter eingeführt. Die Positionen der Knoten der zu G_i gehörenden Menge $GP(G_i)$ $(i > 0)$ werden bestimmt, indem die Abstände zwischen den in der Menge $GP(G_{i-1})$ gespeicherten Punkten halbiert werden. Dieser Vorgang wird wiederholt, bis die vom Benutzer definierte Zahl an Stufen erreicht ist.

Wie in Abbildung 3.8 ersichtlich, werden, ausgehend von der minimalen Auflösung (links), die weiteren Knoten der Gitter durch das Einfügen von zusätzlichen Punkten erzeugt. Es ist anzumerken, dass für die Abbildung die einzelnen Stufen getrennt dargestellt wurden, diese sich jedoch innerhalb des Begrenzungsvolumens überlagern.

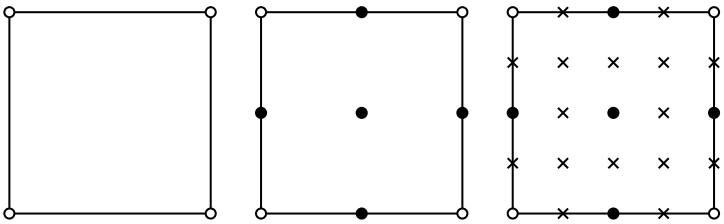

Abbildung 3.8: Erstellung der Gitter in mehreren Auflösungsstufen (Abbildung 2-dimensional für verbesserte Übersichtlichkeit, tatsächlich sind Gitter 3-dimensional)

Die daraus resultierenden Punkte befinden sich innerhalb des Begrenzungsvolumens des darzustellenden Objekts, korrelieren aber meistens nicht mit den Vertizes des Dreiecksnetzes. Es ist für jeden Punkt in $GP(G_0), GP(G_1), ...GP(G_n)$ ein entsprechender Vertex zu ermitteln bzw. der Punkt zu verwerfen, falls kein passender Vertex gefunden werden kann.

Für jeden Punkt P_i^j in $GP(G_i)$ wird - zentriert um P_i^j - ein quaderförmiger Bereich festgelegt. Die Seitenlänge entspricht dem Abstand zu einem benachbarten Punkt in der Menge der Punkte $GP(G_i)$ derselben Stufe. Alle Vertizes des Dreiecksnetzes, die sich innerhalb dieses Bereichs befinden, werden bestimmt (Abbildung 3.9). Ziel ist es, einen dieser Vertizes für die Manipulation des Fehlerwertes auszuwählen. Befindet sich kein Vertex des Dreiecksnetzes innerhalb dieses Bereichs, so wird der Punkt des Gitters verworfen.

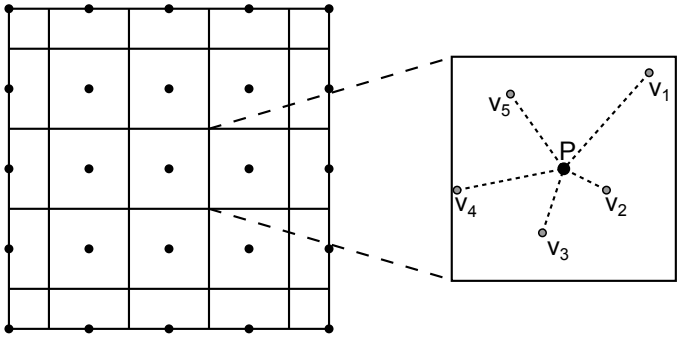

Abbildung 3.9: Bestimmung des Vertex

Für jeden Punkt P_i^j mit der Position p_i^j wird die Menge $V(P_i^j) = \{V_1, V_2, ...V_n\}$ ermittelt. $V(P_i^j)$ enthält alle Vertizes, die sich innerhalb des definierten Bereichs um den Punkt P_i^j befinden. Jeder Vertex V_k speichert die Position v_k und den Vertexfehler $e_k = e(V_k)$ des Vertex.

Die Auswahl eines einzelnen Vertex aus der Menge $V(P_i^j)$ erfolgt mit Hilfe des Vertexfehlers. Damit werden jene bevorzugt, die von der initialen Fehlermetrik als wichtig eingestuft wurden. Eine direkte Wahl des Vertex mit dem größten Fehler innerhalb des Bereichs um P_i^j ist jedoch nicht vorteilhaft: In unruhigen Bereichen des Dreiecksnetzes - in denen sich Gruppen von Vertizes mit verhältnismäßig großem Fehler befinden - können benachbarte Vertizes, die entlang der Grenze zweier Bereiche liegen, mit geringem Abstand zueinander selektiert werden. Die Idee des regelmäßigen Gitters sieht jedoch vor, isolierte Vertizes in regelmäßigen Abständen auszuwählen und so die Anzahl der Iterationsschritte zu reduzieren. Dies lässt sich vermeiden, indem der Vertexfehler anhand der Distanz zwischen P_i^j und V_k gewichtet wird.

Der erste Schritt ermittelt die Distanz $d(V_k, P_i^j)$ zwischen dem Punkt und einem Vertex.

$$d(V_k, P_i^j) = |p_i^j - v_k| \tag{3.4}$$

Der finale Wert $m(V_k, P_i^j)$ ergibt sich mit Hilfe der Gewichtung durch die maximale Seitenlänge l des Quaders um P_i^j und $d(V_k, P_i^j)$.

$$m(V_k, P_i^j) = (l - d(V_k, P_i^j))^2 * e_k \tag{3.5}$$

Aus den Vertizes in $V(P_i^j)$ wird jener selektiert, der den höchsten, gewichteten Vertexfehler $m(V_k, P_i^j)$ aufweist. Diese Vorgehensweise dient dazu, Vertizes, die mit höherer Wahrscheinlichkeit nicht gelöscht werden, zu bevorzugen.

Der gespeicherte Fehlerwert der ausgewählten Vertizes wird überprüft und gegebenenfalls verändert. Für jede Stufe des Gitters legt der Benutzer einen minimalen Fehlerwert

f_i fest. Wird ein Vertex V mit Hilfe eines Punktes in $GP(G_i)$ der Stufe G_i ausgewählt und der gespeicherte Vertexfehler $e(V)$ ist kleiner als f_i, so wird er durch f_i ersetzt.

Bei der Klassifizierung der Vertizes wird der gespeicherte Vertexfehler mittels der Distanz zwischen Beobachter und Vertex skaliert und mit einem vom Benutzer bestimmten Grenzwert verglichen. Somit kann festgelegt werden, ab welcher Entfernung alle Vertizes einer Stufe G_i bei der Klassifizierung beibehalten werden sollen.

Für die Stufe G_0 wird der Fehler f_0 so groß gewählt, dass die durch Punkte in $GP(G_0)$ gewählten Vertizes immer im Dreiecksnetz verbleiben. Mögliche Zielpositionen für Vertexverschiebungen stehen somit zur Verfügung, und die Vereinfachung bleibt funktionsfähig.

Dieses Verfahren kann mehrere Auswirkungen haben:

- **Reduktion von Iterationsschritten** Durch das Bestimmen von zusätzlich beizubehaltenden Punkten, die als Ziele für Vertexverschiebungen dienen, kann eine größere Anzahl an Vereinfachungsoperationen parallel ausgeführt werden. Daraus folgt eine Reduktion von Iterationsschritten und eine Erhöhung der Leistung des Algorithmus.

- **Mindestanzahl an beibehaltenen Vertizes** Es kann garantiert werden, dass in jeder Situation eine gewisse Mindestanzahl an Vertizes beibehalten wird. Diese dient als Ziel für Verschiebungen. Ein Objekt kann nicht vollständig entfernt und eine Reduktion eines Dreiecksnetzes auf einzelne oder sehr wenige Vertizes vermieden werden.

- **Vordefinierte Repräsentationen** Durch den gezielten Einsatz dieser Manipulation wird für jede Stufe G_i eine minimale Repräsentation des Dreiecksnetzes festgelegt. Bei der Klassifizierung findet eine Skalierung des Fehlerwertes mit der Distanz zum Betrachter statt. Das Ausmaß der Manipulation wird für die einzelnen Stufen so gewählt, dass die entsprechenden Vertizes erst ab einer gewissen Entfernung als zu löschen gewählt werden und für jede Distanz eine minimale Zahl an Punkten im Dreiecksnetz verbleibt.

Im Gegensatz dazu kann die Anwendung des regelmäßigen Gitters bei diesem Verfahren auch den Nachteil haben, dass bei der Vereinfachung das Objekt auf die Vertizes mit manipuliertem Vertexfehler reduziert wird. Dies kann eine gitterartige Vernetzung erzeugen, die bei der Darstellung negativ auffallen kann.

Die letzten Faktoren, die in die Klassifizierung von Vertizes einfließen, sind Position und Blickrichtung des Betrachters. Diese Kameraabhängigkeit wird im nächsten Abschnitt erläutert.

3.3.3 Kameraabhängigkeit

Der vorberechnete, statische Vertexfehler speichert die Relevanz eines Vertex in Bezug auf seine Nachbarn. Wie stark die Ausprägung von einem Betrachter wahrgenommen wird, ist abhängig von der Distanz zwischen Beobachter und Objekt und dem Winkel, aus dem das Objekt dargestellt wird [Red96]:

- **Distanz** Je größer der Abstand zwischen Betrachter und dargestelltem Objekt, desto mehr Details werden bei der Rasterisierung entfernt, und es ist nicht nötig, sie bei der Bilderzeugung zu berücksichtigen.

- **Blickwinkel** In welchem Winkel eine Oberfläche betrachtet wird, hat starke Auswirkung auf die Wahrnehmung von Details. Während Ränder des Objekts deutlicher zu sehen sind, fallen Ausprägungen, die sich in Bereichen des Objekts befinden, auf die der Blickvektor in einem Winkel von 90 Grad einfällt, weniger auf [LE97]. Dieses Phänomen lässt sich nutzen, um die Selektion zu entfernender Vertizes zusätzlich zu verfeinern und deutlicher wahrnehmbare Einzelheiten zu erkennen und beizubehalten.

Der vorberechnete Vertexfehler dient als Grundlage für die Entscheidung, ob ein Vertex im Dreiecksnetz verbleibt oder entfernt werden soll. Eine Toleranzgrenze wird bestimmt und mit dem Vertexfehler verglichen. Liegt dieser unterhalb der Toleranzgrenze, so ist die Ausprägung zu gering und der Vertex wird für ein Entfernen markiert. Bei Überschreitung der Toleranz wird der Vertex als relevant eingestuft und verbleit im Dreiecksnetz. Die Position des Betrachters und der Blickwinkel auf das Objekt fließen als Skalierung der Faktoren in die Berechnung ein.

Die Toleranzgrenze für den Vergleich bestimmt, wie stark die in einem Vertex gespeicherte Ausprägung der Oberfläche - repräsentiert durch den gespeicherten Vertexfehler - ausfallen darf, bevor ein Vertex beibehalten werden muss. Je größer die Entfernung zwischen dem dargestellten Objekt und dem Betrachter ist, desto weniger Details werden wahrgenommen. Infolge wird auch die erlaubte Toleranz erhöht und größere Details aus dem Dreiecksnetz entfernt. Die Distanz zwischen Kamera und Objekt fließt als Skalierung der Toleranzgrenze in die Klassifizierung ein.

Die Berechnung des Skalierungsfaktors für die Toleranzgrenze eines Vertex V erfolgt über die Distanz zwischen der Position c des Betrachters und der Position v des Vertex V. Um mehr Freiheit bei der Berechnung zu gewähren, wird ein vom Benutzer gewählter Faktor u eingeführt. Mit diesem lässt sich steuern, wie stark die Distanz in die Klassifizierung der Vertizes einfließt. Der finale Skalierungsfaktor s_t für die Toleranzgrenze wird berechnet durch:

$$s_t = |v - c| * u \tag{3.6}$$

Der Blickwinkel, aus dem ein Betrachter das dargestellte Objekt sieht, ist ein weiterer, wichtiger Faktor. Der Winkel zwischen dem Normalvektor des Vertex und dem einfallenden Blickvektor wird eingesetzt, um den gespeicherten Vertexfehler zu skalieren. Damit wird repräsentiert, wie gut die in einem Vertex gespeicherte Ausprägung der Oberfläche vom Betrachter wahrgenommen werden kann.

Der Skalierungsfaktor s_e für den gespeicherten Fehler wird ausgehend vom einfallenden Blickvektor d und dem Normalvektor v_n des Vertex ermittelt durch:

$$s_e = 1 - (d \bullet v_n)^2 \tag{3.7}$$

Für die Berechnung der finalen Klassifizierung wird ein Vertex V mit dem vorberechneten Vertexfehler e_v betrachtet. Die Toleranzgrenze t bestimmt, ab welchem Fehler der Vertex beibehalten werden muss. Aufgrund der Abhängigkeit von der Größe eines Objekts und der gewünschten Skalierung des finalen Detailgrades wird diese Toleranzgrenze vom Benutzer festgesetzt.

Die finale Ungleichung für die Klassifizierung eines Vertex ist gegeben durch:

$$e_v * s_e < t * s_t \tag{3.8}$$

Ist diese Ungleichung erfüllt, so liegt der gewichtete Vertexfehler unterhalb der skalierten Toleranzgrenze. Die im Vertex gespeicherte Ausprägung der Oberfläche wird bei der gegebenen Distanz von der Kamera und dem Winkel, aus dem das Objekt betrachtet wird, als nicht relevant erachtet und für ein Entfernen aus dem Dreiecksnetz markiert. Liegt der gewichtete Fehler über der skalierten Toleranzgrenze, so wird der Fehler der gespeicherten Ausprägung als zu groß gewertet und der Vertex bleibt erhalten.

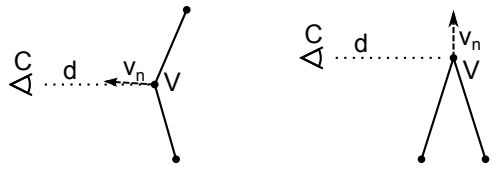

Abbildung 3.10: Beispiele für blickwinkelabhängige Gewichtung

Abbildung 3.10 zeigt auf der linken Seite das Einfallen des Blickvektors auf eine Oberfläche in einem Winkel von annähernd 90 Grad. Hier verringert sich der Skalierungsfaktor für e_v. Im Gegensatz dazu trifft der Blickvektor in der rechten Darstellung auf eine Silhouette des Objekts, wodurch der Faktor gegen 1 geht.

Diese Klassifizierung der Vertizes wird durchgeführt, um eine initiale Menge an Vertizes zu bestimmen, auf der die weiteren Operationen der Vereinfachung ausgeführt werden. Allerdings muss nach Verschiebungsoperationen diese Klassifizierung invalidiert werden, um veränderte Nachbarschaftsbeziehungen zu berücksichtigen. Somit ist nach jeder Verschiebungsoperation eine Reklassifizierung der Nachbarn des entfernten Vertex nötig.

3.3.4 Reklassifizierung

Bei der initialen Klassifizierung wird jeder Vertex ausschließlich in Bezug auf seine direkten Nachbarn betrachtet, und sie ist somit nur für das ursprüngliche Dreiecksnetz vor den Vereinfachungsoperationen gültig. Zusätzlich berechnet die initiale Fehlermetrik die Bewertung, ohne zu berücksichtigen, dass benachbarte Vertizes ebenfalls für ein Entfernen markiert werden könnten. Selbst bei geringen Abständen zur Kamera kann somit eine große Menge von Vertizes innerhalb eines Bereichs des Dreiecksnetzes gelöscht werden und sich eine mindere visuelle Qualität ergeben. Weiters verändern die Verschiebungsoperationen die Nachbarschaftsbeziehungen von Vertizes. Solche, die zur Berechnung der initialen Klassifizierung herangezogen werden, wurden durch Verschiebungen unter Umständen aus dem Dreiecksnetz entfernt. Wird ein Vertex V aus dem Dreiecksnetz entfernt, folgt eine Neuberechnung des Vertexfehlers für alle zu entfernenden Nachbarn von V. Der aktualisierte Fehlerwert kann eine Reklassifizierung dieser Nachbarn verursachen.

Während die Reklassifizierung zu einer Erhöhung des Leistungsaufwandes führt und das Beibehalten zusätzlicher Vertizes eine Einschränkung beim Grad der Vereinfachung darstellen kann, ergeben sich durch die wiederholte Berechnung und Evaluierung des Vertexfehlers (Schritt 4 in Abbildung 3.1) auch mehrere Vorteile:

- **Vermeiden von großen Flächen** Die initiale Klassifizierung kann eine große Anzahl an Vertizes zum Entfernen aus dem Netz bestimmen. Würden diese ohne weitere Überprüfung verschoben, könnten sehr große Flächen entstehen, die sich negativ auf die visuelle Qualität auswirken.

- **Fächerförmige Vernetzung** Durch das Löschen einer großen Anzahl an Vertizes können mit einem einzelnen Punkt ungewöhnlich viele Dreiecke gebildet werden. Diese ungünstige Vernetzung, die zu Einbußen bei der visuellen Qualität des dargestellten Objekts führen kann, wird durch die Reklassifizierung vermieden.

- **Adaptive Auflösung** Durch die Skalierung der Toleranzwerte bei der Neuklassifizierung mit der Position des Betrachters lässt sich der Detailgrad des vereinfachten

Dreiecksnetzes beeinflussen, und die Bestimmung der zu entfernenden Einzelheiten erfolgt stets in Abhängigkeit von deren Auffälligkeit.

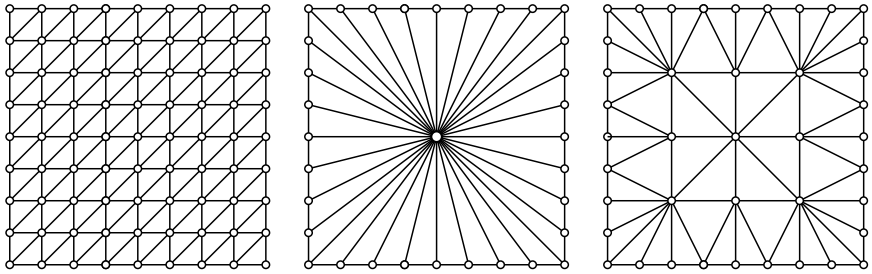

Abbildung 3.11: Beispiel: Vergleich originales Netz (links), Vereinfachung ohne (Mitte) und mit Reklassifizierung (rechts)

Abbildung 3.11 illustriert dieses Phänomen. Ausgehend von einem beliebigen Dreiecksnetz (links) ist der Vergleich einer Entfernung der meisten inneren Vertizes (Mitte) mit einem Ansatz mit Reklassifizierung (rechts) ersichtlich. Während das linke Bild eine stark fächerförmige und ungleichmäßige Triangulierung aufweist, ist auf der rechten Seite durch die Reklassifizierung eine gleichmäßigere Triangulierung entstanden.

Die Ausführung einer eingeschränkten Kantenreduktion auf einem Dreiecksnetz modifiziert Kanten und somit Nachbarschaftsbeziehungen zwischen Vertizes und invalidiert den gespeicherten Fehlerwert noch zu entfernender Nachbarn. Eine erneute Berechnung des Fehlers muss durchgeführt werden. Die modifizierten Distanzen zwischen den Kanten haben zur Auswirkung, dass ein veränderter Vertexfehler ermittelt wird und eine Reklassifizierung vonnöten sein kann, um das Entfernen zu vieler Details aus dem Dreiecksnetz zu vermeiden. Auch bei dieser Operation fließen die Position und Blickrichtung des Betrachters in die Berechnung ein. Das resultierende Netz des vorhergehenden Vereinfachungsschrittes wird als Ausgangspunkt herangezogen.

Die Fehlermetrik bei der Reklassifizierung soll dem gleichen Prinzip folgen wie jene bei der statischen Berechnung, da die Verwendung von abweichenden Fehlermetriken das Risiko birgt, unterschiedliche Resultate zu erhalten. Es besteht die Gefahr, dass die von der statischen Fehlermetrik für ein Entfernen markierten Vertizes durch die Reklassifizierung mit einer zweiten Fehlermetrik einen stark abweichenden Vertexfehler zugeteilt bekommen. Die Folge wäre eine ständige Reklassifizierung. Eine identische oder sehr ähnliche Fehlermetrik ermöglicht es, einheitliche Resultate zu garantieren.

Bei der statischen Vorberechnung beruht die Fehlermetrik auf allen Nachbarn eines Vertex. Zum Zeitpunkt der Reklassifizierung sind die Vertizes des Dreiecksnetzes jedoch als beizubehaltend oder zu entfernend eingestuft. Die Reklassifizierung wird unter der Annahme berechnet, dass zu entfernende Nachbarn im finalen Dreiecksnetz nicht enthalten und für die Berechnung des aktualisierten Fehlerwertes nicht relevant sind. Die

Reklassifizierung findet dementsprechend nur unter Rücksichtnahme auf jene Nachbarvertizes statt, die gegenwärtig als beizubehaltend markiert sind.

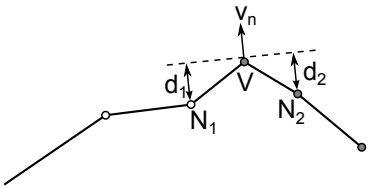

Abbildung 3.12: Modifizierte Fehlermetrik nach Verschiebungen

Abbildung 3.12 zeigt den Ansatz der Fehlermetrik für die Reklassifizierung. Für einen Vertex V soll der aktualisierte Fehlerwert berechnet werden. Analog zur statischen Berechnung wird auch hier auf die Tangentialebene der Oberfläche in V (gebildet aus der Position von V und dem Normalvektor v_n) zurückgegriffen. Es werden die Abstände der Nachbarvertizes (N_1, N_2) zur Tangentialebene ermittelt. Dabei finden nur jene Vertizes Berücksichtigung, die beizubehalten sind. In Abbildung 3.12 ist N_1 ein beizubehaltender Punkt und d_1 somit relevant. N_2 ist noch zu entfernen, d_2 wird somit ignoriert.

Bei der statischen Klassifizierung wird der Mittelwert aller Abstände errechnet. Bei der Reklassifizierung kann die Anzahl der benachbarten Vertizes allerdings stark schwanken und unter Umständen auch auf nur einem beizubehaltenden Nachbarn beruhen. Daher wird für die Reklassifizierung der Maximalwert eingesetzt. Der Fehlerwert für die Reklassifizierung wird berechnet mittels des maximalen Abstandes eines benachbarten Vertex, der beizubehalten ist.

Weiters ist zu berücksichtigen, dass durch die Verwendung einer leicht unterschiedlichen Fehlermetrik die Toleranzgrenze der initialen Klassifizierung bei der Reklassifizierung keine Gültigkeit besitzt. Es muss also eine separate Toleranz definiert werden, die bei der Reklassifizierung angewandt wird. Die Gewichtung des neu errechneten Fehlers durch Blickwinkel und Skalierung der Toleranzgrenze kommt analog zur initialen Klassifizierung zum Einsatz.

Die hier beschriebene Technik erzeugt solche Bereiche, in denen das originale Netz durch die initiale Klassifizierung beibehalten und andere, in denen eine Retriangulierung durchgeführt wird. Während der Detailgrad des finalen Dreiecksnetzes durch die Reklassifizierung dynamisch bestimmt wird, können Teile des originalen Netzes und retriangulierte Bereiche mit wesentlich größeren Dreiecken direkt aneinander grenzen. Es ergibt sich die Problematik, dass mehrere Punkte auf die Position eines einzelnen Zielvertex verschoben werden und eine fächerförmige Triangulierung auftritt. Ein fließender Übergang der Dreiecksgrößen würde ein optisch besseres Resultat erzielen.

Um diesen gleichmäßigeren Übergang zu erreichen, wird zwischen Bereichen des Dreiecksnetzes, die von der initialen Klassifizierung beibehalten, und jenen, die durch die

Vereinfachung neu erstellt wurden, unterschieden. Eine zusätzliche Skalierung der Toleranzgrenze bei der Reklassifizierung von Vertizes wird eingeführt, so dass der erlaubte, maximale Vertexfehler in den Übergangszonen dieser Bereiche verringert wird und die Dreiecksgrößen reduziert werden. Diese Skalierung ist abhängig von der „Entfernung“ zwischen einem zu reklassifizierenden Vertex und einem solchen, der durch die initiale Klassifizierung für ein Beibehalten markiert wurde. Je größer diese ausfällt, desto höher ist auch der tolerierte Fehlerwert. So kann die Größe der Dreiecke in den äußeren Regionen von retriangulierten Bereichen geringer ausfallen als in der Mitte und ein fließender Übergang der Dreiecksgrößen wird erreicht.

Dieser Ansatz wirft zwei Probleme auf: Einerseits muss zwischen der Menge I der beizubehaltenden Vertizes, die durch die initiale Klassifizierung bestimmt wurde, und jener, die bei der Reklassifizierung erzeugt wurde, unterschieden werden. Andererseits kann nicht auf den Abstand zweier Vertizes im Raum für die Skalierung zurückgegriffen werden, da dieser aufgrund der Krümmung der Oberfläche kein zuverlässiges Maß für die Skalierung liefert.

Anstelle einer Distanz im Raum wird die minimale Anzahl an Kanten zwischen einem zu reklassifizierenden Vertex und einem Vertex in I bestimmt. Dieser in weiterer Folge als Sprungzahl bezeichnete Wert wird eingesetzt, um die Toleranzgrenze zu skalieren. Je geringer die Anzahl der Sprünge, desto kleiner ist der erlaubte Fehlerwert.

Besitzt ein zu reklassifizierender Vertex einen Nachbarn, der in I vorhanden ist, so beträgt die Sprungzahl 1 (eine Kante zu einem initial als beizubehaltend eingestuften Vertex), und die Toleranzgrenze für den neu ermittelten Fehlerwert wird stark skaliert. Werden hingegen ausschließlich Nachbarn gefunden, die nicht in I vorhanden sind, so liegt eine Sprungzahl größer 1 vor, anhand der die Skalierung angepasst wird.

Der exakte Skalierungsfaktor für eine Anzahl an Sprüngen ist mitentscheidend für die Qualität des finalen Dreiecksnetzes und wird für eine bessere Anpassung an einen Datensatz vom Benutzer festgelegt.

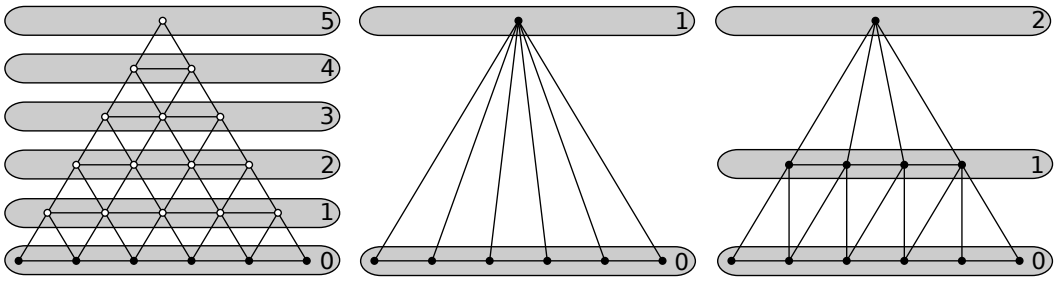

Abbildung 3.13: Vergleich: Reklassifizierung ohne und mit Skalierung durch Sprungzahl

Abbildung 3.13 zeigt einen Vergleich zweier retriangulierter Dreiecksnetze und der ermittelten Sprungzahlen. Links ist das Original zu sehen. Alle nicht markierten Vertizes sollen entfernt werden. In der Mitte wird die Version ohne Skalierung und rechts das

vereinfachte Netz mit Skalierung dargestellt. In der rechten Abbildung wurde durch die Sprungzahl die Toleranzgrenze skaliert, wodurch zusätzliche Dreiecke entstanden sind.

Nach einem Reklassifizierungsschritt erfolgt eine Menge an Vertexverschiebungen. Hierbei werden alle jene Vertizes, die auch nach der Reklassifizierung zu entfernen sind und eine Kante zu einem beizubehaltenden Vertex besitzen, verschoben.

3.4 Verschiebung

Im Anschluss an die Klassifizierung und Bestimmung der initialen Verschiebungskandidaten findet das Entfernen der Verschiebungskandidaten (Schritt 3 in Abbildung 3.1) statt. Dies geschieht durch die parallele Ausführung einer Menge von eingeschränkten Kantenreduktionen (Unterabschnitt 2.3.2).

Die Verschiebung setzt sich aus mehreren Teilbereichen zusammen:

- **1. Netzintegrität** Bei Verschiebungsoperationen darf es zu keiner Verletzung der Netzintegrität kommen.

- **2. Parallele Vertexverschiebung** Beschreibt die Beiträge, die eine parallele Ausführung von eingeschränkten Kantenreduktionen erlauben.

- **3. Erkennung und Auflösung von gesperrten Verschiebungen** Beschäftigt sich mit Verschiebungen, die sich durch die parallele Ausführung gegenseitig blockieren.

- **4. Abbruch von Verschiebungen** Behandlung von Vertizes ohne legale Ersatzposition

- **5. Verschiebungsbewertung** Bewertung und Selektion von Verschiebungszielen

Ein weit verbreiteter Einsatz der Kantenreduktion bzw. eingeschränkten Kantenreduktion basiert auf der Idee, eine Kante zu suchen, die aus dem Dreiecksnetz entfernt werden soll (z. B. Hoppe [Hop97]). Der hier präsentierte Ansatz verwendet zwar die eingeschränkte Kantenreduktion, verfolgt allerdings bei der Durchführung der Verschiebung einen anderen Weg, der auf einen parallelen Vertexeliminierungs-Ansatz zurückgreift.

Zunächst wird die Menge R, die alle zu löschenden Vertizes enthält, bestimmt. Anschließend wird mit K eine Teilmenge von R gesucht, die alle jene Vertizes aus

R enthält, die mindestens eine Kante zu einem beizubehaltenden Vertex besitzen. Jede dieser Kanten dient als mögliche eingeschränkte Kantenreduktion. In einem Iterationsschritt wird versucht, alle Vertizes in K gleichzeitig zu verschieben, indem für jeden Vertex in K eine Kante zu einem bestehen bleibenden Nachbarn ausgewählt wird, entlang der eine Verschiebung stattfindet.

Für jeden Vertex in K stehen mindestens ein, möglicherweise mehrere Verschiebungsziele zur Verfügung. Es ist zu überprüfen, welche dieser Operationen die Integrität des darzustellenden Dreiecksnetzes gefährden und ob eine mögliche Verschiebung ausgeführt werden darf. Diese Überprüfung auf eine Verletzung der Netzintegrität muss immer stattfinden, auch wenn nur eine einzige mögliche Zielposition für einen Vertex vorhanden ist. Von Verschiebungen, die die Netzintegrität gefährden, ist abzusehen [ESV99a].

3.4.1 Netzintegrität

Verschiebungsoperationen modifizieren nicht nur Form und Größe eines Dreiecks, sondern haben auch zur Folge, dass die Rotation des Normalvektors, die einen wichtigen, einschränkenden Faktor bei der Auswahl möglicher Ersatzpositionen darstellt, verändert wird. Die Dreiecksnormale kommt zum Einsatz, um die Vorder- und Rückseite eines Dreiecks zu definieren. Dem Betrachter „abgewandte“ Dreiecke werden bei der Bilderzeugung häufig nicht berücksichtigt (siehe Unterabschnitt 2.2.2). Bei Vereinfachungen des Dreiecksnetzes muss die Ausrichtung zum Betrachter erhalten bleiben, um zu vermeiden, dass durch bei der Filterung entfernte Dreiecke Löcher in der Oberfläche entstehen.

Da die Vertexgruppeneliminierung die eingeschränkte Kantenreduktion als Vereinfachungsoperator einsetzt, muss die Problematik von Faltungen und inkonsistenter Topologie (siehe Unterabschnitt 2.3.1) berücksichtigt werden. Bei der Auswahl von möglichen Ersatzpositionen für einen Vertex müssen alle Verschiebungen, die derartige Artefakte auslösen, erkannt und blockiert werden.

Eine weitere Einschränkung ist bedingt durch die parallele Bearbeitung von benachbarten Vertizes. Es können Fälle auftreten, in denen eine einzelne Verschiebungsoperation legal ist - also eine intakte Oberfläche ergibt und keine Verletzung der Netzintegrität verursacht. Die individuell legalen, gleichzeitigen Verschiebungen von benachbarten Vertizes verletzen jedoch die Integrität der Oberfläche. Auch diese Situationen müssen verhindert werden. Dies soll ohne Kommunikation zwischen benachbarten Operationen möglich sein, so dass Verschiebungen getrennt behandelt werden können und keine Leistungsverluste durch aufwändige Kommunikation entstehen.

Abbildung 3.14 zeigt ein Beispiel für dieses Phänomen. Ausgehend von dem Dreiecksnetz auf der linken Seite sind die Vertizes V_1 und V_2 zu löschen. Die mittlere Spalte zeigt

die Ergebnisse der einzelnen Verschiebungen. Hier werden Form und Größe des Dreiecks modifiziert, die Verschiebung erzeugt jedoch weder Faltungen noch gekippte Dreiecke. Rechts ist das Ergebnis einer parallelen Bearbeitung ersichtlich. Zwei gleichzeitig ausgeführte, individuell erlaubte Operationen haben den Normalvektor des Dreiecks um 180 Grad rotiert und somit die Netzintegrität verletzt. Diese Kombinationen von Verschiebungen müssen erkannt und gesperrt werden.

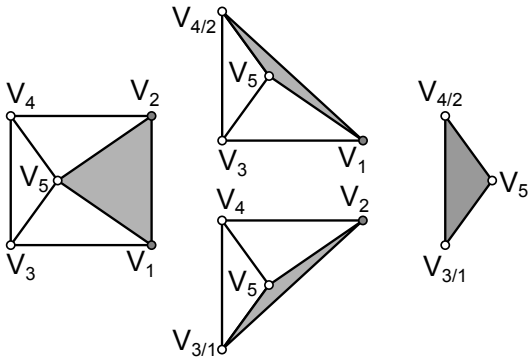

Abbildung 3.14: Vergleich einzelne und parallele Verschiebung

Sowohl für Faltungen als auch für inkonsistente Topologie kann die Ausrichtung der Dreiecksnormalen in Bezug auf den Blickvektor des Betrachters als kritischer Faktor erkannt werden. Gemäß der Definition einer Vorder- bzw. Rückseite eines Dreiecks (siehe Unterabschnitt 2.2.2) wird in beiden Fällen ein Dreieck so rotiert, dass sich die Ausrichtung der Dreiecksnormalen in Bezug auf die Blickrichtung des Betrachters umkehrt.

Durch die Definition der Position eines Betrachters ist feststellbar, wie die Dreiecksnormale in Bezug auf den Blickvektor liegt. Als Folge lässt sich das Kippen eines Dreiecks in Bezug auf eine Betrachterposition erst zur Laufzeit bestimmen. Daraus ergibt sich die Möglichkeit des Festlegens einer maximalen Rotation der Dreiecksnormalen, die ein Kippen von Dreiecken und somit das Entstehen von Faltungen und inkonsistenten Topologien vermeidet. Durch sie lässt sich feststellen, ob eine Verschiebung die Integrität eines Dreiecksnetzes verletzt.

Abbildung 3.15 zeigt diese erlaubte Grenze. In Ausgangslage deutet der Normalvektor (n) des Dreiecks in Richtung des Betrachters (C). Es wird also die Vorderseite des Dreiecks dargestellt. Verschiebt man den Punkt V des Dreiecks, so darf die gestrichelte Linie nicht überschritten werden, da sonst der Normalvektor so weit rotiert, dass die Rückseite des Dreiecks dem Betrachter zugewandt ist.

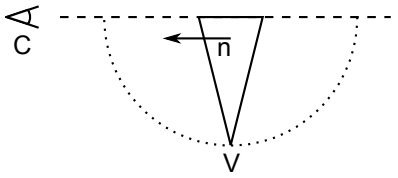

Abbildung 3.15: Maximale erlaubte Rotation

Der erforderliche Test, um die Netzintegrität zu erhalten, ist abhängig von individuellen Dreiecken. Soll ein Vertex V verschoben werden, so ist für alle Dreiecke, die V enthalten zu überprüfen, ob sich die Ausrichtung der Vorderseite des Dreiecks in Bezug auf den Betrachter verändert.

Ein einfacher Ansatz für diesen Test besteht in einem simplen Vergleich der Ausrichtung der Dreiecksnormalen hinsichtlich des Blickvektors vor und nach einer möglichen Verschiebung. Diese Berechnung ist zwar effizient und liefert korrekte Resultate, sie hat jedoch den Nachteil, ausschließlich für die Verschiebung von nur einem Vertex eines Dreiecks einsetzbar zu sein. Eine parallele Bearbeitung benachbarter Vertizes kann mit diesem Verfahren nicht überprüft werden, da etwaige Zielpositionen der Nachbarn vor Abschluss der Verschiebungsoperation - und somit zum Zeitpunkt des Tests - nicht bekannt sind.

Im Zuge dieser Arbeit wurde eine Vorgangsweise entwickelt, die erweitert werden kann, um parallele Verschiebungen ebenfalls zu berücksichtigen. Es werden drei unterschiedliche Tests angewandt, je nachdem wie viele Punkte eines Dreiecks parallel verschoben werden. Im nächsten Abschnitt wird die einfachste Variante dieser Tests erläutert, die zum Einsatz kommt, wenn nur ein Vertex eines Dreiecks ein Verschiebungskandidat ist.

Als Ausgangspunkt dient die Annahme, dass die Verschiebung eines Punktes V_1 eine Rotation des Normalvektors n eines Dreiecks, bestehend aus V_1, V_2 und V_3, darstellt. Die verbleibende Kante, gebildet durch V_2 und V_3, ist die Rotationsachse. Um den Test durchzuführen, wird ein Bereich definiert, in dem die Verschiebung erlaubt ist. Liegt eine Ersatzposition für V_1 außerhalb dieses Bereichs, so kehrt sich die Ausrichtung von n in Bezug auf die Blickrichtung des Betrachters um. Alle möglichen Verschiebungsziele innerhalb des definierten Bereichs stellen legale Ersatzpositionen dar.

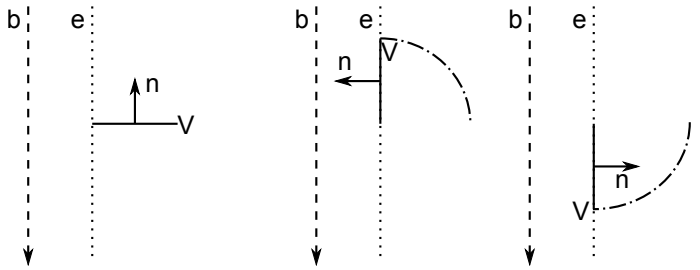

Abbildung 3.16: Grenzen für erlaubte Verschiebung

Abbildung 3.16 zeigt ein zweidimensionales Beispiel für diesen erlaubten Bereich. Der Normalvektor n darf durch eine Verschiebung von V nur so weit rotiert werden, dass der Winkel zwischen n und Blickvektor b minimal 90 Grad beträgt. Eine weitere Rotation negiert die Ausrichtung von n in Hinsicht auf b und vertauscht die Vorder- und Rückseite des Dreiecks. Auf der linken Seite ist die Ausgangsposition des Normalvektors und Vertex zu sehen. Die Ebene e stellt die Begrenzung für den erlaubten Bereich einer Verschiebung von V dar. Jeder Punkt, der sich in diesem Beispiel auf der rechten Seite von e befindet, ergibt einen Normalvektor mit der korrekten Ausrichtung. Ersatzpositionen auf e ergeben einen Normalvektor, der im rechten Winkel zum Blickvektor steht.

Für die Verschiebung eines einzelnen Vertex im Dreieck, also die Rotation um eine definierte Achse, ist dieser Bereich durch eine Ebene abgegrenzt. Eine Verschiebung des Vertex darf nur auf einer Seite der Ebene durchgeführt werden, diese jedoch nie überschreiten. Die Ebene e wird durch die verbleibende Kante des Dreiecks V_2 und V_3 aufgestellt. Jeder Punkt auf e muss mit V_2 und V_3 ein neues Dreieck bilden, dessen Normalvektor genau im rechten Winkel - also der maximal erlaubten Rotation - zum Blickvektor der Kamera steht. Somit kann garantiert werden, dass eine Verschiebung des Vertex kein gekipptes Dreieck erzeugt, solange sie nicht die Ebene schneidet. Abbildung 3.17 zeigt ein Beispiel für eine Ebene, die eine Vertexverschiebung limitiert.

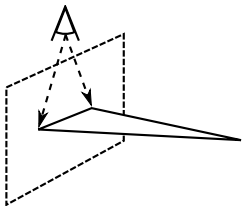

Abbildung 3.17: Ebene für maximal erlaubte Verschiebung

Ausgehend von der Kante zwischen V_2 und V_3 sind ein Punkt und ein Normalvektor zu bestimmen, aus denen e aufgestellt wird. Findet eine Verschiebung statt, so bildet

diese Kante die Rotationsachse der Dreiecksnormalen. Daraus ergibt sich, dass diese Gerade in der zu findenden Ebene liegt. Laut obiger Definition müssen alle Dreiecke, die auf e liegen, einen Normalvektor im Winkel von 90 Grad zum Blickvektor haben.

Für die Berechnung der Ebene eines Dreiecks, bestehend aus den Vertizes V_2, V_3 und dem Ausgangspunkt für die Verschiebung V_1, wird der Vektor zwischen den Positionen v_2 und v_3 der Vertizes V_2 und V_3 bestimmt. Als weiterer Faktor muss die Position des Betrachters c bekannt sein.

Das Kreuzprodukt zwischen $(v_3 - v_2)$ und dem Blickvektor des Betrachters ermittelt den Normalvektor n der Ebene:

$$n = (v_3 - v_2) \times (c - v_2) \tag{3.9}$$

Durch den Punkt v_2 und den Normalvektor n kann die gesuchte Ebene aufgestellt werden. Der Blickvektor der Kamera liegt in dieser Ebene. Daraus resultiert: Jedes Dreieck aus v_2, v_3 und einem Punkt p auf der Ebene erfüllt die Bedingung:

$$\frac{(v_3 - v_2) \times (p - v_2)}{|(v_3 - v_2) \times (p - v_2)|} \bullet \frac{(c - v_2)}{|(c - v_2)|} = 0 \tag{3.10}$$

Diese Ebene e dient als Grenze, um den Raum in zwei Bereiche zu teilen. Solange eine Verschiebung die Grenze nicht überschreitet, bleibt die Orientierung des Dreiecks in Bezug auf die Blickrichtung der Kamera erhalten.

Ausgehend von einem Winkel ϕ zwischen dem Blickvektor und der Dreiecksnormalen muss für die Ebene e gelten:

$$\begin{aligned} \forall p = X + r * n | X \in e, r > 0 : cos(\phi) > 0 \\ \forall p = X + r * n | X \in e, r < 0 : cos(\phi) < 0 \end{aligned} \tag{3.11}$$

Das Skalarprodukt zwischen dem Normalvektor und dem Blickvektor muss für alle Dreiecke, die mit einem Punkt auf einer Seite der Ebene gebildet werden, dasselbe Vorzeichen aufweisen.

Ein Punkt x kann beschrieben werden durch:

$$\begin{aligned} n = (v_3 - v_2) \times (c - v_2) \\ x = t + s * n \end{aligned} \tag{3.12}$$

t stellt einen beliebigen Punkt auf e dar.

Der Winkel ϕ zwischen dem Blickvektor des Betrachters und dem Normalvektor des Dreiecks wird berechnet durch:

$$cos(\phi) = \frac{((x - v_2) \times (v_3 - v_2)) \bullet (c - v_2)}{|(x - v_2) \times (v_3 - v_2)| * |(c - v_2)|} \tag{3.13}$$

Das Spatprodukt ist zyklisch vertauschbar:

$$\begin{aligned} l &= |(x - v_2) \times (v_3 - v_2)| * |(c - v_2)| \\ cos(\phi) &= \frac{((v_3 - v_2) \times (c - v_2)) \bullet (x - v_2)}{l} \\ cos(\phi) &= \frac{n \bullet (x - v_2)}{l} \\ cos(\phi) &= \frac{n \bullet (t + s * n - v_2)}{l} \\ cos(\phi) &= \frac{n \bullet ((t - v_2) + s * n)}{l} \\ cos(\phi) &= \frac{n \bullet (t - v_2) + s * (n \bullet n)}{l} \end{aligned} \tag{3.14}$$

Die Punkte t und v_2 liegen laut obiger Definition beide auf der Ebene. Das Skalarprodukt von $n \bullet (t - v_2)$ ergibt also 0. Außerdem gilt $l > 0$ und $(n \bullet n) > 0$. Das Vorzeichen ergibt sich somit aus:

$$cos(\phi) = \frac{s * (n \bullet n)}{l} \tag{3.15}$$

Das Vorzeichen von $cos(\phi)$ - und somit die Ausrichtung der Vorderseite des Dreiecks - ist nur mehr abhängig von s. Daher gilt:

$$\begin{aligned} \forall t \in e | s > 0 : cos(\phi) > 0 \\ \forall t \in e | s < 0 : cos(\phi) < 0 \end{aligned} \tag{3.16}$$

Es wurde somit gezeigt, dass alle Dreiecke, deren dritter Punkt sich auf einer Seite der Ebene befindet, dieselbe Ausrichtung zum Betrachter haben.

Um die Legalität einer eingeschränkten Kantenreduktion für ein Dreieck zu überprüfen, wird die Gerade g durch den Ausgangspunkt V_1 und das Ziel T einer Verschiebung aufgestellt. Im Anschluss wird g mit e geschnitten. Liegt ein gefundener Schnittpunkt auf der Kante - also zwischen V_1 und T - so ist die zu überprüfende Ersatzposition außerhalb des ermittelten Bereichs. Die definierte Einschränkung ist verletzt, und die Operation darf nicht durchgeführt werden. Andernfalls liegt eine legale Vereinfachung vor.

Abbildung 3.18 zeigt ein Beispiel für einen Verschiebungstest. V_1 stellt den Ausgangspunkt dar, V_4 die Ersatzposition. Die Ebene für das Dreieck V_1, V_2, V_3 wird mit dem Vektor der Kante zwischen V_1 und V_4 geschnitten. Für die gefundenen Schnittpunkte wird überprüft, ob die Verschiebung die Grenze überschreitet. In diesem Beispiel befinden sich sowohl V_1 als auch V_4 auf derselben Seite der Ebene, es existiert also kein Schnittpunkt auf der Kante. Die Verschiebung kann folglich durchgeführt werden.

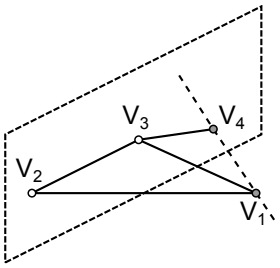

Abbildung 3.18: Beispiel für einen Verschiebungstest

Dieser Test bewertet die Gültigkeit einer Verschiebung für ein einzelnes Dreieck. Bei einer Verschiebungsoperation wird allerdings der gesamte Dreiecksfächer, gebildet mit einem zu entfernenden Vertex, manipuliert. Der Test muss für jedes dieser Dreiecke wiederholt werden.

Um den Bereich zu definieren, in dem eine Verschiebung durchgeführt werden darf, wird für jedes Dreieck, das den Ausgangsvertex V enthält, eine Ebene e aufgestellt. Daraus ergibt sich die zu V gehörige Menge an Ebenen $E(V) = \{e_1, e_2, ...e_n\}$, welche die Außengrenzen des Bereichs, innerhalb dessen ein Vertex verschoben werden darf, bestimmt.

Um die Legalität einer Ersatzposition von V zu bestimmen, muss der oben beschriebene Test für alle Ebenen in $E(V)$ ausgeführt werden. Wird eine Ebene in $E(V)$ gefunden, die einen Schnittpunkt mit der Geraden auf der Kante zwischen V und der Zielposition besitzt, so ist eine alternative Ersatzposition zu suchen. Eine Verschiebung kann nur gefahrlos erfolgen, wenn die Operation anhand aller Ebenen in $E(V)$ erlaubt ist.

Abbildung 3.19 zeigt dieses Vorgehen in zwei Beispielen. Die gestrichelten Linien repräsentieren die Ebenen in $E(V)$. Hellgrau markierte Kanten stellen gültige Verschiebungen dar. Der zentrale Vertex ist jeweils zu entfernen, die umliegenden sind mögliche Ersatzpositionen. Im linken Fall ist ersichtlich, dass keine der Kanten eine Ebene schneidet. Somit sind alle äußeren Vertizes gültige Ersatzpositionen für den mittleren Vertex. Im rechten Fall existieren mehrere Kanten, die Schnittpunkte mit einer Begrenzungsebene aufweisen. Jede dieser Kanten stellt eine ungültige Verschiebung dar, die nicht durchgeführt werden darf.

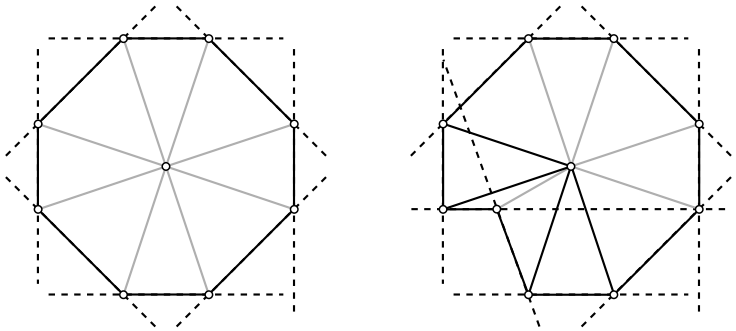

Abbildung 3.19: Bereich für erlaubte Verschiebungen

Bei unabhängigen Verschiebungen einzelner Vertizes sichert dieser erste, einfache Test den Erhalt der Netzintegrität. Kommt es jedoch zu parallelen Verschiebungen von benachbarten Punkten, so deckt dieser Test nicht alle Fehlerfälle ab. Stattdessen sind restriktivere Tests einzusetzen, welche die Netzintegrität bei gleichzeitiger Manipulation von Nachbarvertizes erhalten. Diese Tests werden im nächsten Abschnitt beschrieben.

3.4.2 Parallele Vertexverschiebungen

Der einfache Test sieht die Daten der Nachbarvertizes als gegeben und unveränderlich. Findet jedoch die gleichzeitige Verschiebung eines zweiten Vertex innerhalb eines Dreiecks statt, so ist das Dreieck, auf dem der einfache Test basiert, nicht mehr in seiner ursprünglichen Form vorhanden. Die Rotationsachse, die als Basis für das Aufstellen der Ebene dient, wird in demselben Arbeitsschritt verändert. Die Resultate des einfachen Tests sind nicht aussagekräftig, und der Erhalt der Netzintegrität kann somit nicht garantiert werden.

Als Folge muss ein alternativer Test für die Gültigkeit einer Verschiebung eingesetzt werden, der die Anzahl der zu entfernenden Punkte eines Dreiecks berücksichtigt. Das Kriterium der Gültigkeit wird so erweitert, dass eine Ersatzposition für einen Vertex nur legal ist, wenn:

- Die Verschiebung des Vertex für den gegenwärtigen Dreiecksfächer keine gekippten Dreiecke erzeugt.

- Die Verschiebung auch nach der parallel ablaufenden Manipulation des Nachbarvertex keine gekippten Dreiecke erzeugt.

Ein weiteres Kriterium der Berechnung von erlaubten Bereichen für die gleichzeitige Bearbeitung ist die Anforderung an die Parallelität des Verfahrens. Benachbarte Verti-

zes sollen ohne einen Informationsaustausch oder eine Serialisierung verschoben werden können, um eine effiziente Realisierung zu ermöglichen. Ziel ist es, eine Bedingung zu finden, durch die eine potentielle Ersatzposition so weit eingegrenzt wird, dass keine gekippten Dreiecke erzeugt werden, sofern dieselbe Bedingung ebenfalls für einen zu entfernenden Nachbarn zum Einsatz kommt. Als Folge dieser Anforderung müssen unterschiedliche Tests festgelegt werden, in Abhängigkeit, ob zwei oder alle drei Vertizes eines Dreiecks gleichzeitig bearbeitet werden sollen. Die restriktivste Einschränkung wird definiert, wenn alle drei Vertizes eines Dreiecks gleichzeitig verschoben werden.

Der dritte, restriktivste Test wäre auch für die parallele Verschiebung von ein oder zwei Punkten eines Dreiecks gültig. Durch die starke Begrenzung würde jedoch der Bereich für erlaubte Ersatzpositionen eines Vertex kleiner ausfallen als benötigt. Das Ergebnis läge in der Sperrung von legalen Ersatzpositionen. Um eine größtmögliche Freiheit bei der Auswahl einer eingeschränkten Kantenreduktion zu ermöglichen, findet eine getrennte Behandlung der drei Fälle statt.

Neben dem einfachen Test ist im zweiten Fall die parallele Verschiebung von zwei Punkten eines Dreiecks zu betrachten und alle möglichen Verschiebungskombinationen, die ein gekipptes Dreieck erzeugen, sind zu identifizieren.

Der einfachere Test basiert auf dem Ansatz, dass zwei Vertizes des Dreiecks in ihrer gegenwärtigen Position erhalten bleiben. Sie können somit als Datenbasis für die Aufstellung der Ebene herangezogen werden. Durch die parallele Verschiebung wird einer der verbleibenden Vertizes im selben Iterationsschritt entfernt. Die Lage dieser Kante im Raum wird manipuliert, und die neue Ausrichtung ist zum Zeitpunkt der Verschiebung nicht bekannt.

Abbildung 3.20 soll dieses Problem verdeutlichen. V_2 und V_4 sollen verschoben werden. Der Vertex V_2 greift beim einfachen Verschiebungstest auf die Daten des Dreiecks V_2, V_5, V_4 zurück. Wird V_4 parallel verschoben, so wird die Kante V_4, V_5 modifiziert und kann bei der Verschiebung von V_2 nicht verwendet werden. Es muss ein neuer Weg gefunden werden, um die entsprechende Begrenzungsebene für diesen Fall zu konstruieren.

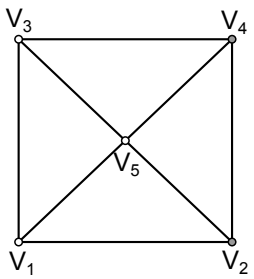

Abbildung 3.20: Wettbewerbsbedingung bei paralleler Verschiebung

Während beim einfachen Test der Normalvektor nur um eine Achse rotieren kann - verursacht durch die Manipulation der Position eines einzelnen Vertex des Dreiecks - ist nun zusätzlich die aus Betrachtersicht gegenläufige Verschiebung der beiden Kandidaten eines Dreiecks zu berücksichtigen. In Abbildung 3.21 sollen die zwei markierten Vertizes des linken Dreiecks gleichzeitig verschoben werden. Würden die beiden Vertizes entlang der mit Pfeilen markierten Kanten verschoben, so käme es zu einer nicht erlaubten Rotation des Dreiecks.

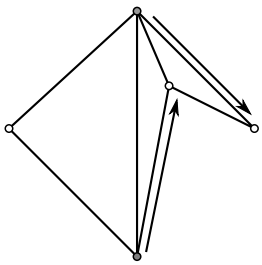

Abbildung 3.21: Gegenläufige Verschiebung

Bei der Definition der Begrenzungen für ein Dreieck mit zwei Verschiebungskandidaten sind folgende Kriterien zu beachten:

- Die gegenläufige Verschiebung von zwei benachbarten Verschiebungskandidaten muss beschränkt werden.

- Die Adaption der Begrenzungsebene aus dem einfachen Test, so dass die parallele Manipulation der Nachbarvertizes mit dem verbleibenden Punkt des Dreiecks keine Fehler verursacht.

Um die gegenläufige Verschiebung zu vermeiden, wird eine Ebene bestimmt, welche die Kante zwischen den beiden zu verschiebenden Vertizes in einem Punkt schneidet. Sie stellt eine Barriere zwischen den zwei Vertizes dar, die von keiner Verschiebungsoperation überschritten werden darf. Somit kann das Problem der gegenläufigen Verschiebungen unterbunden werden. Die gewählte Ebene muss für die beiden zu verschiebenden Punkte ident ausfallen.

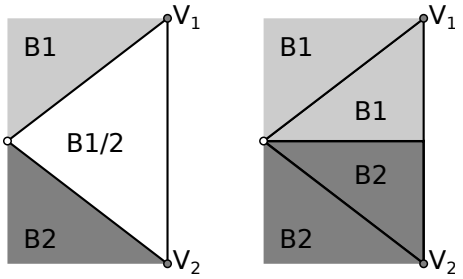

Abbildung 3.22: Erschaffung exklusiver Bereiche

Abbildung 3.22 verdeutlicht dieses Problem. Es ist anzumerken, dass das Ausmaß der Bereiche auf den Ausschnitt von Abbildung 3.22 reduziert wurde und in Wirklichkeit größer ist. Die Bereiche B_1 beinhalten alle gültigen Ersatzpositionen des Vertex V_1. Selbes gilt für die Bereiche B_2 und den Vertex V_2. Die Überschneidung von B_1 und B_2 (links) bedeutet, dass die möglichen Zielpositionen beider, zu löschender Vertizes in einem gemeinsamen Bereich liegen können. Das Ergebnis solcher Verschiebungen wäre nicht vorhersagbar. Rechts wurde dieses Problem durch die zusätzliche Ebene - und somit durch die Trennung von B_1 und B_2 in exklusive Bereiche - beseitigt.

Bei Einführung einer Ebene als Grenze zwischen zwei parallel zu verschiebenden Vertizes wird jedoch nicht die tatsächlich vollzogene Verschiebung des Nachbarn berücksichtigt. Kombinationen von Vertexverschiebungen, die keine Verletzung der Netzintegrität verursachen, können als ungültig erklärt werden. Diese Einschränkung wird akzeptiert, da eine Überprüfung der ausgeführten Verschiebungen eines Nachbarn eine Kommunikation zwischen den parallelen Verschiebungsoperationen erfordern würde. Durch die große Zahl der zu verschiebenden Vertizes und der möglichen Verschiebungskombinationen wäre diese Art der Überprüfung aufwändig und zeitintensiv und somit für die Berechnung in Echtzeit von Nachteil.

Der beibehaltene Vertex des Dreiecks und die Position des Betrachters sind gegeben. Um die Ebene aufzustellen, muss noch ein dritter Punkt gefunden werden, mit der Anforderung, dass die resultierende Ebene die Kante zwischen den beiden parallel zu verschiebenden Punkten schneidet. Jeder Punkt auf dieser Kante ist eine mögliche und gültige Wahl. Im Rahmen dieser Arbeit wurde der Mittelpunkt der Kante zwischen den beiden Verschiebungskandidaten (vor Ausführung einer Operation auf einem der Kandidaten) gewählt. Der resultierende Vektor ist somit die Schwerelinie des Dreiecks, das als Ausgangspunkt der Verschiebung dient. Abbildung 3.23 zeigt ein Beispiel für eine so definierte Ebene.

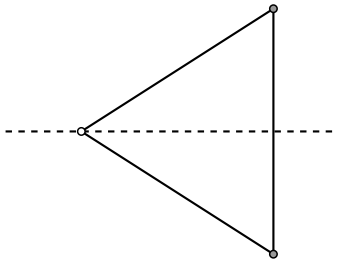

Abbildung 3.23: Zusätzliche Begrenzungsebene

Diese Ebene des zweiten Tests wird analog zu der Ebene des simplen Tests aufgestellt. Im ersten Schritt wird sie durch die Schwerelinie des zu testenden Dreiecks, bestehend aus den Vertizes V_1, V_2 und V_3 mit den Positionen v_1, v_2 und v_3, gelegt. V_1 und V_2 sind die Verschiebungskandidaten. Ausgehend von einem Betrachter in Punkt c wird der Normalvektor n der Ebene e ermittelt.

$$\begin{aligned} s &= v_3 - \frac{v_1 + v_2}{2} \\ t &= c - \frac{v_1 + v_2}{2} \\ n &= \frac{s \times t}{|s \times t|} \end{aligned} \tag{3.17}$$

Die Ebene e mit dem Normalvektor n wird durch den Schwerpunkt des Dreiecks gelegt.

Diese erste Ebene für den zweiten Verschiebungstest löst das Problem der überlappenden Verschiebungsbereiche und begrenzt die gegenläufige Verschiebung zweier Vertizes. Bei der parallelen Verschiebung von benachbarten Punkten ist auch die Kante zwischen den beiden Kandidaten zu berücksichtigen. Durch die Verschiebung beider Endpunkte wird die Lage der Kante im Raum verändert - sie kann aus Betrachtersicht auf die andere Seite des verbleibenden Punktes transferiert werden.

In Abbildung 3.24 ist die Kante (gebildet durch V_1 und V_2), deren Lage sich verändert, markiert. Das auftretende Problem ist die Distanz zu dem verbleibenden Vertex V_3 des Dreiecks. Werden V_1 bzw. V_2 in der Abbildung so weit nach links verschoben, dass die Kante V_3 passiert, so tritt ein gekipptes Dreieck auf.

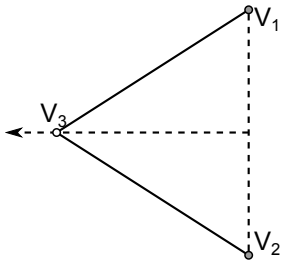

Abbildung 3.24: Verschiebung einer Kante

Betrachtet man die nach der oben definierten Ebene erlaubte Verschiebung von V_1 und V_2, so ist ersichtlich, dass bei der Verschiebung der Kante der bestehen bleibende Punkt (V_3) nicht berücksichtigt wird. Der tolerierte Bereich einer Verschiebung bedarf einer Anpassung, so dass die individuelle Verschiebung einzelner Punkte die maximale Verschiebung der Kante zwischen den Vertizes miteinbezieht. Diese maximale Verschiebung der Kante muss so begrenzt werden, dass sie den verbleibenden Punkt des Dreiecks respektiert.

Wie bereits bei der Definition der ersten Ebene des zweiten Verschiebungstests würde die Berücksichtigung der tatsächlich durchgeführten Verschiebung eine Kommunikation erfordern. Dieser Umstand soll jedoch vermieden werden. Stattdessen kommt auch hier die Definition eines Bereichs, der die Erfüllung dieser Bedingung (maximale Verschiebung der Kante) garantiert, zum Einsatz. Die Einschränkung bei der Wahl potentieller Ersatzpositionen und die damit verbundene Sperrung legaler Verschiebungen wird akzeptiert, um eine effiziente Umsetzung und einen hohen Grad an Parallelität zu ermöglichen.

Der Problemfall tritt ein, wenn die Kante aus Sicht des Betrachters über den verbleibenden Vertex hinaus verschoben wird. Eine zweite, gemeinsame Ebene e wird bestimmt, die für beide zu verschiebende Vertizes als Grenze gilt. Somit ist garantiert, dass die maximale Verschiebung beider Punkte eine Kante erzeugt, die auf e liegt, sie jedoch nie überschreiten kann. Wird e so definiert, dass sie durch den verbleibenden Punkt läuft und V_1 sowie V_2 auf derselben Seite von e liegen, kann sichergestellt werden, dass die Reihenfolge der Vertizes des Dreiecks - und somit die Ausrichtung der Dreiecksnormalen in Bezug auf eine Betrachterposition - erhalten bleibt.

Abbildung 3.25 zeigt ein Beispiel für eine derartige Ebene. Nimmt man die Ebene - die durch den verbleibenden Punkt liegt - als Begrenzung für die Verschiebung, so kann das Dreieck unter der Annahme, dass keine gegenläufige Verschiebung der Kandidaten stattfindet, nicht kippen.

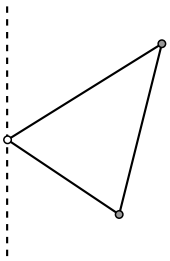

Abbildung 3.25: Maximal erlaubte Verschiebung einer Kante

Mit dem verbleibenden Vertex des Dreiecks ist der Punkt, durch den die Ebene gelegt wird, definiert. Es gilt, einen Normalvektor zu finden, der die Lage der Ebene bestimmt. Dieser Vektor muss für beide zu verschiebende Vertizes identisch und aus den Daten des Dreiecks vor der Verschiebungsoperation bestimmbar sein.

Als maximale Verschiebung bietet sich eine Ebene an, die parallel zu der Kante zwischen den Ausgangspositionen der beiden zu verschiebenden Vertizes liegt. Sie ist für beide Vertizes aus den Daten des ursprünglichen Dreiecks bestimmbar.

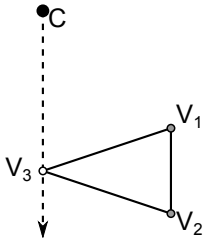

Abbildung 3.26: Berechnung der erlaubten Verschiebung

In Abbildung 3.26 wird für diesen Test die Ebene e durch den Punkt V_3 des Dreiecks aufgestellt. V_3 ist jener Punkt des Dreiecks, der nicht aus dem Netz entfernt wird. Der Normalvektor n von e wird in Bezug auf die Position des Betrachters c wie folgt berechnet:

$$\begin{aligned} s &= v_2 - v_1 \\ t &= c - v_3 \\ n &= \frac{s \times t}{|s \times t|} \end{aligned} \tag{3.18}$$

Der finale, erlaubte Bereich einer Verschiebung für ein Dreieck mit zwei parallel zu verschiebenden Punkten ist somit definiert durch zwei Ebenen:

- Ebene e_1: parallel zu der Kante zwischen den beiden zu verschiebenden Vertizes und durch den verbleibenden Punkt des Dreiecks
- Ebene e_2: durch die Schwerelinie des Dreiecks

Abbildung 3.27 zeigt die resultierenden Ebenen für die Verschiebung von zwei benachbarten Vertizes. Es existieren keine überlappenden Bereiche, in denen eine Verschiebung erlaubt wäre und durch den Einsatz der Daten des nicht modifizierten Dreiecks kann eine Wettbewerbsbedingung ausgeschlossen werden.

Wird der Verschiebungstest durchgeführt, muss die Kante, entlang welcher die Operation erfolgt, mit e_1 und e_2 geschnitten werden. Findet sich ein auf der Kante liegender Schnittpunkt mit einer oder beiden Ebenen, so entfällt die Verschiebung.

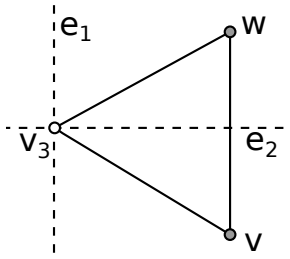

Abbildung 3.27: Erlaubte Bereiche für parallele Verschiebung von 2 Vertizes

Mit diesen zwei Ebenen ist für jeden Vertex ein exklusiver Bereich definiert, der die Ausrichtung des Dreiecks erhält: Gegeben sind die Ebene $e_1 : n_1 \bullet X - d_1 = 0$, die parallel zu der zu verschiebenden Kante verläuft und die Ebene $e_2 : n_2 \bullet X - d_2 = 0$, die mit der Schwerelinie aufgestellt wird. Ein Dreieck wird errichtet aus den Punkten w, v, v_3. w und v sind Verschiebungskandidaten. Sie liegen auf gegenüberliegenden Seiten von e_2. Es existiert also ein Schnittpunkt zwischen e_2 und der Kante zwischen v und w. Laut Definition des zweiten Verschiebungstests dürfen die Verschiebungen von v und w die Ebene e_2 nicht schneiden. Daraus resultiert, dass auch nach der Manipulation beider Punkte ein Schnittpunkt mit e_2 auf der veränderten Kante existiert.

Es wird ein Dreieck v_3, t, u betrachtet, für das gilt:

$$\begin{aligned} e_2 : n_2 \bullet u - d_2 = 0 \\ t = a + b * n_2 \\ n_2 \bullet a - d_2 = 0 \end{aligned} \tag{3.19}$$

a und u sind somit beliebige Punkte auf e_2. Für jedes derartige Dreieck wird der Winkel zwischen $(c - v_3)$ und dem Normalvektor des Dreiecks bestimmt:

$$\begin{aligned} cos(\phi) = \frac{(t - v_3) \times (u - v_3)) \bullet (c - v_3)}{|(t - v_3) \times (u - v_3)| * |(c - v_3)|} \\ l = |(t - v_3) \times (u - v_3)| * (c - v_3)| \\ cos(\phi) = \frac{((u - v_3) \times (c - v_3)) \bullet (t - v_3)}{l} \end{aligned} \tag{3.20}$$

Der Punkt u liegt laut obiger Definition auf e_2. Das Kreuzprodukt $(u - v_3) \times (c - v_3)$ ergibt einen Normalvektor der Ebene e_2. Länge und Orientierung des resultierenden Vektors können allerdings von n_2 abweichen. Es kann geschrieben werden $(u - v_3) \times (c - v_3) = s * n_2$.

$$\begin{aligned} cos(\phi) = \frac{s * n_2 \bullet ((a + b * n_2) - v_3)}{l} \\ cos(\phi) = \frac{s * n_2 \bullet (a - v_3) + s * b * (n_2 \bullet n_2)}{l} \end{aligned} \tag{3.21}$$

a und v_3 liegen laut obiger Definition auf der Ebene e_2. Das Skalarprodukt $n_2 \bullet (a - v_3)$ ist 0. Das Vorzeichen von $cos(\phi)$ - und somit die Ausrichtung des Dreiecks - hängt ab von $s * b$, also der Orientierung des Normalvektors (s) und auf welcher Seite von e_2 sich der Punkt t befindet.

Berechnet man nun den Vektor $n_3 = (c - v_3) \times n_2$, so liegt dieser parallel zu e_2. Die Ebenen e_1 und e_2 werden durch den Vektor $c - v_3$ aufgestellt, dieser ist folglich auch der Schnittvektor.

Man kann den Punkt u, der auf der Ebene e_2 liegt, definieren mit:

$$u = (v_3 + f * (c - v_3)) + g * n_3 \tag{3.22}$$

u repräsentiert einen beliebigen Schnittpunkt der Kante zwischen w und v auf e_2. Es soll gezeigt werden, dass die Ausrichtung des Normalvektors von e_2, errechnet mittels

des Kreuzprodukts von $u - v_3$ und $c - v_3$, identisch ist für alle $g > 0$ bzw. für alle $g < 0$ (also alle Punkte, die sich auf derselben Seite von e_1 befinden):

$$\begin{aligned} cos(\theta) &= \frac{((u - v_3) \times (c - v_3)) \bullet (n_2)}{|((u - v_3) \times (c - v_3))| * |n_2|} \\ m &= |((u - v_3) \times (c - v_3))| * |n_2| \\ cos(\theta) &= \frac{((c - v_3) \times n_2) \bullet (u - v_3)}{m} \\ cos(\theta) &= \frac{n_3 \bullet ((v_3 + f * (c - v_3)) + g * n_3 - v_3)}{m} \\ cos(\theta) &= \frac{n_3 \bullet f * (c - v_3) + g * (n_3 \bullet n_3)}{m} \end{aligned} \tag{3.23}$$

$c - v_3$ liegt normal zu n_3, das Skalarprodukt ist somit 0. Die Orientierung des Normalvektors aus $c - v_3$ und $u - v_3$ hängt demnach nur von g ab. Berechnet man also den Normalvektor n_2 von e_2, so ist die Ausrichtung von n_2 identisch für alle $g > 0$ bzw. $g < 0$.

Daraus ergibt sich, dass die Ausrichtung für alle Dreiecke, bestehend aus v_3, dem Schnittpunkt u auf e_2 und einem dritten Punkt t identisch ist, für alle t, u laut Definition 3.19 bzw. 3.22 mit:

$$\begin{aligned} &\forall b > 0, g > 0 : cos(\phi) > 0 \\ &\forall b > 0, g < 0 : cos(\phi) < 0 \\ &\forall b < 0, g > 0 : cos(\phi) < 0 \\ &\forall b < 0, g < 0 : cos(\phi) > 0 \end{aligned} \tag{3.24}$$

Da die Verschiebungskanten die Ebene e_1 nicht schneiden dürfen, befindet sich auch u immer auf derselben Seite von e_1 wie w und v. Somit besitzen alle Dreiecke, deren Schnittpunkt mit e_2 die geforderte Bedingung erfüllt, dass deren Verschiebungskandidaten vor sowie nach der Verschiebung auf unterschiedlichen Seiten von e_2 liegen und die Verschiebungen die Ebene e_1 nicht überschreiten, dieselbe Ausrichtung zur Kamera.

Dieser Test hat analog zum einfachen Verschiebungstest lediglich für ein Dreieck Gültigkeit. Wird nun bei einem Verschiebungstest der gesamte Dreiecksfächer betrachtet, so muss für jedes Dreieck die Anzahl der zu verschiebenden Vertizes bestimmt und der entsprechende Test durchgeführt werden (also der einfache Test bei einem zu verschiebenden Punkt, der zweite Test mit zwei Begrenzungsebenen bei zwei gleichzeitig zu verschiebenden Vertizes).

Obwohl der zweite Test auch bei Dreiecken mit nur einem zu verschiebenden Vertex die Netzintegrität bewahrt, wird dieser ausschließlich bei zwei zu verschiebenden

Vertizes in einem Dreieck angewandt, um größere Freiheiten bei der Verschiebung von Vertizes zu erlauben.

Der letzte Fall behandelt ein Dreieck, bei dem alle drei Eckpunkte gleichzeitig verschoben werden sollen. Wählt man als Ansatz die Ebenen aus dem zweiten Verschiebungstest, so werden 6 Ebenen pro Dreieck gebildet (2 pro Vertex). Drei liegen jeweils in einer Schwerelinie, die verbleibenden drei Ebenen verlaufen parallel zu einer Kante durch den gegenüberliegenden Eckpunkt des Dreiecks. Abbildung 3.28 zeigt die resultierenden, erlaubten Bereiche für Verschiebungen. Die Berücksichtigung der maximalen Verschiebung einer Kante ist in diesem Fall nicht gegeben. Werden die Vertizes V_1 und V_2 wie markiert verschoben, so schneidet die modifizierte Kante einen Bereich, in den der dritte Vertex des Dreiecks (V_3) verschoben werden darf. Somit könnte ein gekipptes Dreieck erzeugt werden.

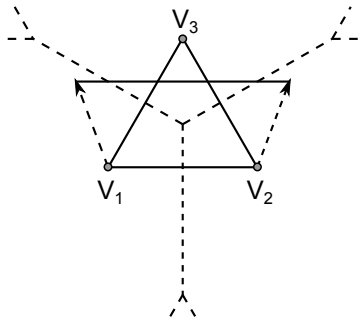

Abbildung 3.28: Verletzte Netzintegrität bei paralleler Verschiebung von 3 Vertizes

Als Folge dieser verletzten maximalen Verschiebung wird eine restriktivere (dritte) Version des Verschiebungstests eingesetzt. Hier kann sich die Lage jeder Kante des Dreiecks verändern. Diese zusätzliche Freiheit muss bei der Auswahl der erlaubten Bereiche berücksichtigt werden, so dass bei einer legalen Verschiebung zweier Vertizes keine Kante entstehen kann, die den erlaubten Verschiebungsbereich des dritten Punktes schneidet.

Diese Anforderung zieht eine weitere Einschränkung der möglichen Verschiebungsbereiche nach sich. Hierbei wird auf den Schwerpunkt des Dreiecks zurückgegriffen. Dieser liegt immer innerhalb des Dreiecks und kann als Mittelpunkt der drei Vertizes angesehen werden. Folglich ist der Verschiebungsbereich eines Vertex durch zwei Ebenen definiert. Diese verlaufen parallel zu den vom Vertex ausgehenden Kanten und werden durch den Schwerpunkt des Dreiecks aufgestellt.

Abbildung 3.29 zeigt die durch diese Ebenen erlaubten Verschiebungsbereiche für alle 3 Eckpunkte des Dreiecks. Ein Vertex darf nur innerhalb jenes Bereichs verschoben werden, in dem er sich befindet. Bei diesem Test wird für jede Kante eine maximale

Verschiebung festgelegt und der Bereich des dritten zu verschiebenden Vertex so eingegrenzt, dass dieser die Kante aus Sicht des Betrachters nicht kreuzen kann. Dieses Vorgehen resultiert in sehr restriktiven Grenzen.

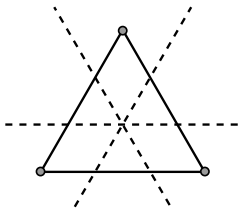

Abbildung 3.29: Erlaubte Verschiebungsbereiche für 3 Vertizes

Ein Dreieck besteht aus drei zu verschiebenden Punkten v_1, v_2 und v_3 mit dem Schwerpunkt s. Die beiden Ebenen für v_1 werden aufgestellt. Beide enthalten den Schwerpunkt s:

$$\begin{aligned} n_1 &= \frac{(v_2 - v_1) \times (c - s)}{|(v_2 - v_1) \times (c - s)|} \\ & e_1 : n_1 * X - d_1 = 0 \\ n_2 &= \frac{(v_3 - v_1) \times (c - s)}{|(v_3 - v_1) \times (c - s)|} \\ & e_2 : n_2 * X - d_2 = 0 \end{aligned} \tag{3.25}$$

Es gilt zu zeigen, dass alle Dreiecke, die durch diese Bereiche entstehen können, eine idente Ausrichtung zur Kamera haben.

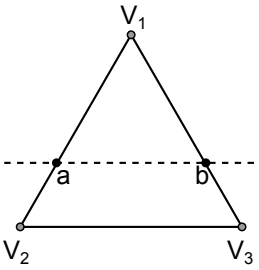

Abbildung 3.30: Schnittpunkte einer Begrenzungsebene mit den Kanten des Dreiecks

Anstelle des gesamten Dreiecks wird hier das Dreieck aus Abbildung 3.30, gebildet aus V_1 und den Schnittpunkten a, b auf e_1 - der gemeinsamen Begrenzungsebene für V_2

und V_3 parallel zur Kante zwischen V_2 und V_3 -, betrachtet. Dieses hat einen identischen Normalvektor zu dem Dreieck, bestehend aus V_1, V_2, V_3.

Beim einfachen Test wurde gezeigt, dass zwei festgelegte Punkte p_1, p_2 auf einer Ebene e, welche die Position des Betrachters enthält, mit allen Punkten, die sich auf einer Seite von e befinden, Dreiecke mit gleicher Ausrichtung zum Betrachter bilden.

Beim zweiten Test wurde dies erweitert, da nur p_1 eine festgelegte Position hat, p_2 zwar auf e liegt, seine Position sich jedoch verändert. Hier wurde bewiesen, dass die Ausrichtung des Dreiecks erhalten bleibt, sofern die Ausrichtung des Normalvektors aus $(c - p_1) \times (p_2 - p_1)$ für alle möglichen p_2 identisch ist.

In diesem Fall ergibt sich, dass die Ausrichtung des Vektors $n_2 = (b - a) \times (c - s)$ für alle möglichen a, b gleich sein soll. Ausgehend von der Begrenzungsebene e_1 (parallel zur Kante zwischen V_2 und V_3) mit Normalvektor n kann der Punkt b analog zu Gleichung 3.22 ausgedrückt werden durch:

$$\begin{aligned} n_1 &= n \times (c - s) \\ b &= (a + l * (c - s)) + g * n_1 \end{aligned} \tag{3.26}$$

In Gleichung 3.23 wurde gezeigt, dass die Ausrichtung von n_2 für alle $g > 0$ bzw. $g < 0$ erhalten bleibt.

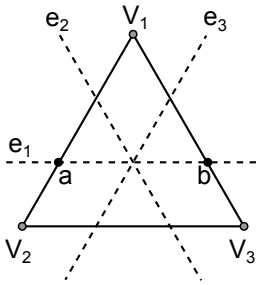

Abbildung 3.31: Ausrichtung des Dreiecks

Die Gerade $s + t * (c - s)$ stellt die Schnittgerade der drei Begrenzungsebenen dar. Keine der Ebenen (e_1, e_2 und e_3 in Abbildung 3.31) darf von Verschiebungen geschnitten werden. Es ergibt sich, dass der Vektor $(b - a)$ einen Schnittpunkt mit dieser Geraden hat, der zwischen a und b liegt.

Für den Schnittpunkt d zwischen $(b - a)$ und der Geraden $s + t * (c - s)$ gilt also: $|d - a| < |b - a|$ und $|d - b| < |b - a|$. Man kann a und b auch wie folgt ausdrücken:

$$\begin{aligned} a &= (s + o * (c - s)) + p * n_1 \\ b &= (s + q * (c - s)) + r * n_1 \end{aligned} \tag{3.27}$$

Hierbei muss gelten $p * r < 0$, da a und b niemals auf derselben Seite der trennenden Geraden liegen dürfen. Die Ebenen wurden so aufgestellt, dass sich die Ausgangspunkte der Verschiebung auf unterschiedlichen Seiten befinden und keine Verschiebungskante eine Ebene schneiden darf. Für 3.26 bedeutet dies, dass für alle möglichen Paare von a, b gilt, dass entweder $g > 0$ oder $g < 0$ und somit die Ausrichtung des Dreiecks erhalten bleibt.

Durch diese drei verschiedenen Testfälle wird garantiert, dass bei Verschiebungen keine gekippten Dreiecke entstehen. Je nach Test sind ein bzw. zwei Ebenen pro Vertex aufzustellen und zu überprüfen, ob ein Schnittpunkt zwischen der Kante, gebildet aus Ausgangspunkt V und Ziel T der Verschiebung und den Ebenen besteht. Existiert ein Schnittpunkt zwischen dem Vektor der Kante und einer Ebene und liegt dieser zwischen V und T, so ist die Verschiebung nicht legal und wird verworfen.

Die Ebenen für diese Tests wurden gewählt, um eine hohe Parallelität und eine effiziente Umsetzbarkeit zu ermöglichen. Des Weiteren bieten sie den Vorteil, dass keine Kommunikation zwischen den parallel zu verschiebenden Vertizes erforderlich ist. Ein Nachteil besteht in der hohen Restriktivität. Die selektierten Ersatzpositionen der Nachbarvertizes werden nicht berücksichtigt. Als Folge können durch die Begrenzungsebenen des zweiten bzw. dritten Tests Kombinationen, die keine gekippten Dreiecke erzeugen würden, verhindert werden.

Mit dem gegenseitigen Sperren von legalen Verschiebungen benachbarter Vertizes kann der Fall eintreten, dass für Nachbarn - und damit auch für Gruppen von Vertizes - keine gültige Ersatzposition gefunden wird. Derartige Punkte verbleiben potentiell über mehrere Iterationsschritte hinweg Verschiebungskandidaten und werden unter Umständen nie aus dem Netz entfernt, obwohl eine gültige Ersatzposition - jedoch blockiert durch den zweiten oder dritten Test - vorhanden wäre. In Teilbereichen kann die Vereinfachung gänzlich scheitern, da sich alle Vertizes aufgrund einer hohen Zahl an Verschiebungskandidaten, und den somit restriktiven Begrenzungen, gegenseitig blockieren.

Abbildung 3.32 zeigt diese Situation. Die beiden Vertizes V_1 und V_2 sollen parallel verschoben werden. Der einfache Verschiebungstest erlaubt für beide Vertizes eine Verschiebung auf die entsprechenden Zielpunkte. Führt man jedoch den Test für zwei parallel zu verschiebende Vertizes durch, so wird die Verschiebung von V_2 gesperrt, obwohl mit der gekennzeichneten Manipulation von V_1 keine Verletzung der Netzintegrität entstände. Tritt ein Fall ein, bei dem keine weiteren, möglichen Ersatzpositionen zur Verfügung stehen, so ist ein Entfernen von V_2 im gegenwärtigen Schritt nicht möglich.

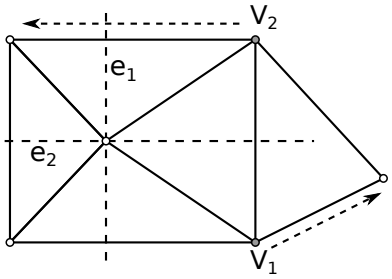

Abbildung 3.32: Beispiel einer Blockade

Um eine vollständige Vereinfachung zu berechnen, müssen diese gesperrten Verschiebungen erkannt und aufgelöst werden. Dabei ist zu unterscheiden, ob die parallele Ausführung (also der zweite oder dritte Test) eine Verschiebung verhindert oder auch die Begrenzungsebene eines einfachen Tests kein Entfernen des Vertex erlaubt. Wird aufgrund des zweiten oder dritten Tests keine legale Kantenreduktion gefunden, so wird der einfache Test ausgeführt.

Es muss unterschieden werden:

- **Gesperrte Verschiebung** Eine gesperrte Verschiebung tritt ein, wenn der einfache Verschiebungstest zwar eine gültige Ersatzposition findet, diese jedoch von den Tests für zwei bzw. drei parallel zu verschiebende Vertizes nicht erlaubt wird. Eine Vereinfachungsoperation ist somit unter Umständen möglich, allerdings nicht mit der aktuellen Konstellation zu löschender Nachbarn.

- **Unmögliche Verschiebung** Die Verschiebung wird nicht nur durch die Grenzen des zweiten oder dritten Tests verhindert. Der einfache Test liefert ebenfalls keine erlaubte Ersatzposition. Hier ist eine Verschiebung nicht möglich.

Bei gesperrten Verschiebungen wird der Ansatz verfolgt, Vertizes, die sich in einer Blockade befinden, temporär aus der Liste der Verschiebungskandidaten zu entfernen, um das Auflösen der Blockade zu ermöglichen. In einem späteren Iterationsschritt wird ein neuer Versuch einer Verschiebung unternommen.

Bei unmöglichen Verschiebungen werden die Nachbarvertizes überprüft: existiert ein benachbarter Verschiebungskandidat, so könnte durch dessen Entfernen eine neue Ersatzposition geschaffen werden, die keine Verletzung der Netzintegrität verursacht. Ein erneuter Versuch einer Verschiebung wird unternommen, sobald der Nachbar entfernt wurde. Liegt kein benachbarter Verschiebungskandidat vor, so kommt es zum Abbruch der Verschiebung und der Reklassifizierung des Vertex.

In den nächsten beiden Abschnitten wird die Behandlung von gesperrten und unmöglichen Verschiebungen genauer erklärt.

3.4.3 Erkennung und Auflösung von gesperrten Verschiebungen

In jedem Iterationsschritt wird versucht, alle Verschiebungskandidaten auf die Position eines im Dreiecksnetz verbleibenden Nachbarn zu verschieben. Hierzu werden die beschriebenen Tests zum Erhalt der Netzintegrität durchgeführt. Der zweite oder dritte Test kann legale Ersatzpositionen verhindern. Findet er keine gültige eingeschränkte Kantenreduktion, wird der einfache Verschiebungstest ausgeführt.

Gesperrte Verschiebungen können sich im Laufe mehrerer Iterationsschritte von selbst lösen, da Nachbarn von blockierten Vertizes entfernt und nur mehr der einfache Verschiebungstest benötigt würde. Die blockierten Vertizes würden somit erst nach mehreren Iterationsschritten verschoben. Das Resultat wäre eine Einschränkung der Parallelität bis hin zur Serialisierung von Verschiebungen.

Es können sich auch Bereiche ergeben, in denen ausschließlich Dreiecke mit multiplen Verschiebungskandidaten liegen und alle Verschiebungen gesperrt werden. In diesen Bereichen bleiben alle Vertizes des originalen Dreiecksnetzes erhalten, und die definierte Abbruchbedingung, dass alle aus dem Dreiecksnetz zu entfernenden Vertizes abgearbeitet werden, könnte nicht erreicht werden. Es ist somit unumgänglich, gesperrte Verschiebungen zu behandeln.

Da bei einer einzelnen Verschiebung nur ein Vertex und seine Nachbarn betrachtet werden, ist es unmöglich, innerhalb desselben Iterationsschrittes eine gegenseitige Blockade mehrerer Vertizes zu erkennen. Dazu muss das Ergebnis benachbarter Operationen vorliegen.

Das Entfernen eines einzelnen Vertex kann ausreichen, um entweder eine blockierende Ebene eines Verschiebungstests für irrelevant zu erklären oder eine neue, potentielle eingeschränkte Kantenreduktion zu schaffen. Das Problem besteht darin, dass die Verschiebungen immer von den Daten des letzten Iterationsschrittes ausgehen. Die von einem Nachbarn gewählte eingeschränkte Kantenreduktion ist zu diesem Zeitpunkt nicht bekannt.

Tritt eine gesperrte Verschiebung auf, wird im nächsten Iterationsschritt entschieden, ob ein erneuter Verschiebungsversuch erfolgt oder eine Auflösung der Sperrung nötig ist. Die Situation der Nachbarvertizes des letzten Iterationsschrittes wird mit der aktuellen Variante verglichen. Es wird festgestellt, welche zu verschiebenden Nachbarvertizes des vorherigen Iterationsschrittes noch vorhanden sind und ob sich die Zahl möglicher Verschiebungsziele im Vergleich zum letzten Iterationsschritt verändert hat:

- **Neue, mögliche Ersatzpositionen sind nicht vorhanden** Durch Verschiebungen von Nachbarvertizes oder deren Reklassifizierung können neue Zielpositionen entstan-

den sein, die selbst bei wiederholter Durchführung der zuvor blockierenden Tests eine Verschiebung zulassen und somit eine Sperrung auflösen.

- **Blockierende Nachbarvertizes sind noch vorhanden** Der zweite oder dritte Verschiebungstest sperrt die Operation. Solange diese blockierenden Nachbarn bestehen, kann keine Verschiebung durchgeführt werden und der neuerliche Test einer möglichen Verschiebung würde fehlschlagen, sofern die Ersatzpositionen identisch sind.

Ist maximal eine dieser beiden Bedingungen gegeben, so kommt es zu einem wiederholten Versuch, eine Ersatzposition zu finden. Sind beide Bedingungen erfüllt, so wird davon ausgegangen, dass sich dieser Bereich des Dreiecksnetzes in einer gegenseitigen Blockade befindet. Es gilt, diese aufzulösen, um die Funktionsfähigkeit der Vereinfachung zu garantieren und die Parallelität zu erhöhen. Das erfolgt durch das Einführen einer zusätzlichen Markierung von Vertizes, die es erlaubt, Verschiebungskandidaten zu ignorieren, so dass sie erst in einem späteren Iterationsschritt bearbeitet werden. Somit können Vertizes, die parallele Verschiebungen verhindern, aus der Liste der zu verschiebenden Vertizes entfernt, und es kann versucht werden, die Vereinfachung der entsprechenden Bereiche durchzuführen. Für die ignorierten Kandidaten wird eine Manipulation erst versucht, wenn zumindest ein Nachbarvertex verschoben wird oder neue, potentielle Verschiebungsziele geschaffen werden.

Zwar löst man mit diesem Vorgehen das Problem, dass alle Vertizes in einem Bereich des Dreiecksnetzes für eine Verschiebung gesperrt sind, es treten jedoch zwei neue Schwierigkeiten auf:

- **Serialisierung** Durch das Ignorieren von Vertizes für die Verschiebung wird in Bereichen des Dreiecksnetzes potentiell eine Serialisierung der Operationen durchgeführt. Dies reduziert die Parallelität und die Leistung des Verfahrens.

- **Fehlen erforderlicher Ausgangspunkte** Werden alle Vertizes eines isolierten Bereichs als gesperrt erkannt und als Verschiebungskandidaten ignoriert, ist eine Entfernung dieser Vertizes aus dem Dreiecksnetz nicht möglich.

Die Lösung dieser Probleme besteht darin, einzelne, ignorierte Kandidaten für Verschiebungen freizugeben - in weiterer Folge werden sie als Hilfspunkte bezeichnet. Ein Vertex wird nur dann von Operationen ausgenommen, wenn der einfache Test ein gültiges Verschiebungsziel gefunden hat und die Verschiebung ausschließlich durch den zweiten bzw. dritten Test verhindert wird. Werden die zu verschiebenden Nachbarn ignoriert, ist diese Situation nicht mehr gegeben und eine Verschiebung kann stattfinden.

Es ist darauf zu achten, dass die Hilfspunkte so gewählt werden, dass sie isoliert sind,

also keine zu verschiebenden Nachbarn besitzen. Daher werden die Ebenen des zweiten oder dritten Tests, die eine Verschiebung verhinderten, nicht mehr benötigt, und eine gültige Ersatzposition kann gefunden werden. Durch diese Bereitstellung von Vertizes als mögliche Verschiebungsziele bei Operationen in nachfolgenden Iterationsschritten wird einerseits die Funktionsfähigkeit des Verfahrens gesichert und andererseits die parallele Ausführung verbessert.

Diese Hilfspunkte können aufgrund der Anforderung, dass der einfache Verschiebungstest eine Ersatzposition zulässt, immer verschoben werden. Sie beschleunigen somit die Wiederaufnahme von ignorierten Nachbarn in die Liste der Verschiebungskandidaten, und ein höherer Grad an Parallelität wird ermöglicht. Bei der Auswahl der Hilfspunkte muss darauf geachtet werden, dass keine Kante zwischen zwei derartigen Vertizes besteht. Ansonsten könnte der zweite oder dritte Verschiebungstest, der ursprünglich die Problematik ausgelöst hat, wieder ausgeführt werden und eine neuerliche Blockierung verursachen.

Um die Hilfspunkte auszuwählen, betrachtet man den gespeicherten Vertexfehler. Dieser ist für jeden Vertex vorberechnet. Gesucht werden Vertizes mit lokalem Maximum des Vertexfehlers von ignorierten Vertizes, um zu garantieren, dass keine Nachbarn ebenfalls als Hilfspunkte selektiert werden.

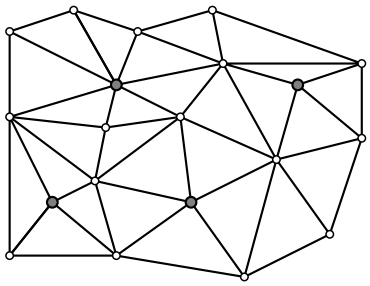

Abbildung 3.33: Ausgewählte Punkte ohne gemeinsame Kanten

Abbildung 3.33 zeigt ein Beispiel einer derartigen Auswahl. Alle inneren Vertizes wurden als gesperrte Verschiebungen erkannt. Die äußeren Punkte sind beizubehalten. Es wird eine Reihe von gesperrten Vertizes ausgewählt (grau). Diese besitzen keine gemeinsamen Kanten, und der einfache Verschiebungstest erlaubt eine Verschiebung. Somit werden für den nächsten Iterationsschritt Verschiebungsziele für die verbleibenden Punkte geschaffen und die Blockade kann aufgelöst werden.

Nicht nur der zweite oder dritte Test, auch der einfache kann eine Verschiebung verhindern. Wenn alle drei Verschiebungstests die Verschiebung blockieren, so muss dieser Zustand berücksichtigt werden. Dies wird im nächsten Abschnitt erklärt.

3.4.4 Abbruch von Verschiebungen

Während die erweiterten Begrenzungsebenen für zwei oder drei parallel zu verschiebende Vertizes restriktiv ausfallen und mögliche legale Verschiebungen sperren, ist der einfache Verschiebungstest immer gültig, da er nur Operationen blockiert, die auch bei einer Bearbeitung von einzelnen Vertizes gekippte Dreiecke erzeugen. Findet der einfache Test keine erlaubte Ersatzposition, ist mit der gegenwärtigen Situation an benachbarten Vertizes keine Verschiebung durchführbar.

Ein Verschiebungskandidat, für den keine Ersatzposition zur Verfügung steht, der jedoch in jedem späteren Iterationsschritt einen erweiterten Verschiebungstest für die Nachbarvertizes erzwingt und eine Blockade verursachen kann, ist aus der Liste der Verschiebungskandidaten zu entfernen. Schlägt der einfache Test fehl, so muss zwischen drei Fällen unterschieden werden:

- **1. Ein oder mehrere Nachbarn sind Verschiebungskandidaten** Im ersten Fall sind Nachbarvertizes vorhanden, die gegenwärtig als Verschiebungskandidaten markiert sind. Es besteht die Möglichkeit, dass durch das Finden einer Ersatzposition für diese Nachbarn ein blockierendes Dreieck entfernt oder eine potentielle Ersatzposition geschaffen wird. Der Vertex wird ignoriert, um Blockaden auszuschließen. Wird ein Nachbar verschoben, so wird der Vertex wieder in die Liste der Verschiebungskandidaten aufgenommen.

- **2. Keine zu entfernenden Nachbarn** Im zweiten Fall sind alle Nachbarvertizes beizubehalten. Aufgrund der gegenwärtigen Konstellation der Dreiecke ist keine Verschiebung möglich, ohne ein gekipptes Dreieck zu erzeugen. Der Vertex muss reklassifiziert werden und bleibt im finalen Resultat der Vereinfachung erhalten.

- **3. Zu entfernende Nachbarn ohne Verschiebungskandidaten** Im letzten Fall sind nicht alle Nachbarvertizes für ein Beibehalten markiert, unter ihnen finden sich jedoch auch keine Verschiebungskandidaten. Ein oder mehrere Nachbarn, die für ein Entfernen markiert sind, besitzen noch keine mögliche eingeschränkte Kantenreduktion. Es besteht die Option, dass diese in einem späteren Iterationsschritt verschoben werden und somit eine neue Ersatzposition für den aktuellen Vertex erzeugen. In diesem Fall wird der Vertex ebenfalls reklassifiziert.

Im dritten Fall wird die Möglichkeit einer potentiellen, späteren Verschiebung ignoriert. Dies hat mehrere Gründe:

- **Erhöhung der Parallelität** Durch die Schaffung eines neuen Verschiebungszieles können alle benachbarten, zu entfernenden Vertizes für den nächsten Iterationsschritt als Verschiebungskandidaten markiert werden.

- **Sicherstellung der Vereinfachung** Durch die Reklassifizierung wird sichergestellt, dass die benachbarten Vertizes eine Ersatzposition besitzen und das Verfahren funktionsfähig bleibt.

Nachdem die potentiellen Ersatzpositionen für die Verschiebungskandidaten bestimmt wurden, müssen alle eingeschränkten Kantenreduktionen bewertet und eine ausgewählte durchgeführt werden. Dazu kommt eine Bewertung aller möglichen Verschiebungen zum Einsatz.

3.4.5 Verschiebungsbewertung

Die parallele Bearbeitung von benachbarten Vertizes verhindert eine optimale Beurteilung der potentiellen Ersatzpositionen für Vertizes. Eine Zielposition muss ohne Kenntnis der ausgewählten Verschiebungen benachbarter Vertizes selektiert werden. So vermeidet man eine Kommunikation zwischen den einzelnen Verschiebungen, allerdings mit dem Nachteil, dass nicht optimale Ersatzpositionen zum Einsatz kommen können.

Für die Bewertung einer eingeschränkten Kantenreduktion wird auf die Quadrik-Fehlermetrik, vorgestellt von Garland und Heckbert [GH97] (siehe Unterabschnitt 2.4.2), zurückgegriffen. Der Grund für die Auswahl dieses Ansatzes besteht darin, dass er effizient berechenbar ist, ausschließlich auf Vertexdaten beruht, die bereits für die Verschiebungstests benötigt wurden und für die einzelnen Vereinfachungsschritte neu ermittelt werden kann.

Wird mit der Quadrik-Fehlermetrik ein Ersatzpunkt einer Kante errechnet, werden im originalen Ansatz die Matrizen der beiden Endpunkte einer Kante addiert und die resultierende Matrix als Grundlage für die Berechnung herangezogen. In dieser Arbeit wird der Ansatz adaptiert, und man benutzt anstelle des originalen Netzes das resultierende Dreiecksnetz des letzten Iterationsschrittes. Nach jedem durchgeführten Verschiebungsschritt muss die Quadrik-Fehlermetrik für einen Vertex neu berechnet werden. Diese Anpassung ist nötig, da der originale Ansatz für Vereinfachungsoperationen bei der Quadrik-Fehlermetrik eine Vertexpaarverschmelzung durchführt. Das hier eingesetzte Verfahren basiert im Gegensatz dazu auf der eingeschränkten Kantenreduktion. Ein Endpunkt der Kante und eine Teilmenge der Dreiecke, die mit diesem Punkt gebildet werden, bleiben erhalten. Die Summierung der Matrizen der beiden Vertizes - wie von Garland und Heckbert vorgestellt - wäre somit nicht zielführend. Zusätzlich kann ein einzelner Vertex als Ersatzposition für viele, entfernte Vertizes dienen, was eine Zusammenführung mehrerer Matrizen in einem Punkt nach sich zieht.

Verfolgt man den original vorgestellten Ansatz weiter, so können bei jedem späteren Iterationsschritt zusätzliche Vertizes auf dieselbe Ersatzposition verschoben werden. Dabei wird über multiple Iterationsschritte hinweg eine große Anzahl von Matrizen in einem einzelnen Vertex aufsummiert. Wie die Autoren von [GH97] besagen, ist diese

Aufsummierung nur eine Annäherung der Fehlermatrix eines so erzeugten Punktes. Durch die Menge an gebildeten Summen vergrößert sich der entstehende Fehler in der Matrix massiv, was diesen Ansatz nicht vorteilhaft macht.

Der in [GH97] vorgestellte Algorithmus wählt die Ersatzposition einer Kante nach dem minimalen Fehler $\triangle(\overline{v})$ für einen Ersatzvertex $\overline{v}$. Hierzu wird für die summierte Matrix einer Kante ein Punkt mit minimalem $\triangle(\overline{v})$ gesucht. Diese Position wird als Ersatzvertex für die Kante benutzt.

Das in dieser Arbeit vorgestellte Verfahren greift auf die Fehlermatrix eines Vertex zurück, es führt jedoch keine Summierung der Matrizen durch. Anstelle dieser Operation bestimmt man die Matrix für jeden Vertex, der in diesem Operationsschritt verschoben werden soll. Wird eine gültige eingeschränkte Kantenreduktion gefunden, so kann die Position des Ersatzvertex anhand der Matrix des zu entfernenden Punktes bewertet werden. Hierdurch wird für jede mögliche Verschiebung ein Wert ermittelt. Die Verschiebung mit dem geringsten berechneten $\triangle(\overline{v})$ wird bestimmt und ausgeführt.

3.5 Zusammenfassung

In diesem Kapitel wird die Vertexgruppeneliminierung - ein neues Verfahren zur Vereinfachung von Dreiecksnetzen - beschrieben. Im Gegensatz zu den meisten Algorithmen liegt der Fokus auf Echtzeitverarbeitung und Beschleunigung des Verfahrens durch hohe Parallelität. Vereinfachungsoperationen werden während der Laufzeit ausgewählt und durchgeführt. Die Vertexgruppeneliminierung wurde mit Hinblick auf die parallele Ausführung auf einer GPU entwickelt. Die Vereinfachungsoperatoren wurden angepasst, um für eine parallele Ausführung praktikabel zu sein.

Durch die Entwicklung dieses Verfahrens für die parallele Ausführung kann auch bei sehr großen Dreiecksnetzen eine Verkürzung der Rechenzeit erzielt werden. Aufgrund der gleichzeitigen Bearbeitung benachbarter Vertizes müssen Einschränkungen bei der Auswahl von Ersatzpositionen akzeptiert werden, um eine schnelle Berechnung der Vereinfachung zu gewährleisten.

Im Zuge dieser parallelen Verarbeitung wird ein neuer Ansatz zur Bestimmung der Auswirkungen der Vereinfachungsoperationen beschrieben. Dieser sichert den Erhalt der Vorder- bzw. Rückseite der Dreiecke und garantiert, dass die Integrität des Dreiecksnetzes erhalten bleibt. Das Verfahren berücksichtigt dabei die Position der Kamera und unterstützt somit den Erhalt der sichtbaren Oberfläche.

Es wurde eine Implementierung für die Ausführung der Vertexgruppeneliminierung auf einer GPU entwickelt. Das nächste Kapitel stellt Details dieser Implementierung vor.

Kapitel 4

Umsetzung der Vertexgruppeneliminierung

Es wurde eine Implementierung der Vertexgruppeneliminierung geschaffen, die als konzeptioneller Beweis dient und in weiterer Folge zur genaueren Analyse herangezogen wurde (Präsentation und Diskussion der Ergebnisse in Kapitel 5). In diesem Kapitel wird auf die Implementierung des Verfahrens genauer eingegangen. Es werden die eingesetzten Speicherstrukturen und Details zur parallelen Implementierung vorgestellt.

Folgende Punkte werden behandelt:

- **1. Überblick der benötigten Arbeitsschritte** Gibt einen Überblick über die Teilschritte der Implementierung
- **2. Anpassung der Vertexdaten und Hilfsdatenstrukturen** Präsentiert die in der Implementierung eingesetzten Datenstrukturen
- **3. Aufbau der Nachbarlisten** Beschäftigt sich mit Auffinden und Speicherung von Nachbarvertizes
- **4. Berechnung des statischen Vertexfehlers** Implementierungsdetails für die Berechnung des Vertexfehlers und dessen Manipulation
- **5. Klassifizierung und Bestimmung der initialen Verschiebungskandidaten** Beschreibung des ersten zur Laufzeit ausgeführten Teilschrittes
- **6. Vertexverschiebungen** Implementierungsdetails des Verschiebungsschrittes
- **7. Reklassifizierung** Ablauf des Reklassifizierungsschrittes in der Implementierung

- **8. Aufbau des vereinfachten Dreiecksnetzes** Generierung des finalen, vereinfachten Dreiecksnetzes

Die Implementierung wurde für die Ausführung auf einer GPU entwickelt, um die hohe parallele Rechenleistung moderner Graphikprozessoren für die Vereinfachung nutzen zu können. Allerdings ergeben sich daraus Einschränkungen bei der Umsetzung des Verfahrens. Die Aufteilung der nötigen Berechnungen auf die individuellen Prozessoren der GPU muss ohne Datenabhängigkeiten oder globale Barrieren zur Synchronisation innerhalb der einzelnen Arbeitsschritte erfolgen, da ein derartiger Mechanismus nicht zur Verfügung steht. Auf diese Problematik sowie die Teilschritte der Implementierung wird im nächsten Abschnitt genauer eingegangen.

4.1 Überblick der benötigten Arbeitsschritte

Die einzelnen Schritte der Implementierung der Vertexgruppeneliminierung (Abbildung 4.1) lassen sich in zwei Kategorien gliedern: eine Vorberechnung und die Berechnung des vereinfachten Dreiecksnetzes mit Hilfe der Ergebnisse der Vorberechnung.

- **Vorberechnung** Hier werden Berechnungen mit den statischen Vertexdaten eines Dreiecksnetzes ausgeführt. Es erfolgen der Aufbau und das Befüllen der benötigten Datenstrukturen und die Berechnung des Vertexfehlers. Die Ergebnisse dieser Operationen werden in den Datenstrukturen abgelegt und dienen in den späteren Teilen der Implementierung als Eingabe.

- **Berechnung des vereinfachten Dreiecksnetzes** Hier wird die Berechnung der Vereinfachung ausgeführt. Nach Abschluss der Manipulation wird der finale Datensatz für das resultierende Dreiecksnetz erstellt.

Die Liste der Teilschritte der Implementierung unterscheidet sich von der im vorigen Kapitel. Der Aufbau der Nachbarlisten wurde als zusätzlicher Schritt in die Vorberechnung aufgenommen. Dies dient zur Reduktion der Berechnungszeit, um eine Nachschlagetabelle der Nachbarvertizes bereitzustellen.

Bei der Berechnung des vereinfachten Dreiecksnetzes wird noch der Aufbau des vereinfachten Dreiecksnetzes als letzter Arbeitsschritt eingefügt. Hier werden redundante Daten aus dem Dreiecksnetz gelöscht. Dieser Schritt macht ein erneutes Aktualisieren der Nachbarlisten nach jeder Iteration unnötig.

Die Implementierung des Verfahrens wurde mit Nvidia CUDA entwickelt. Jeder Teilschritt wird individuell in einem CUDA Kernel ausgeführt. Es muss der Abschluss eines jeden Arbeitsschrittes abgewartet werden, bis mit der Ausführung des nächsten Teil-

schrittes begonnen werden kann. CUDA besitzt keinen Mechanismus für eine globale Barriere, die eingesetzt werden könnte, um die Ergebnisse aller gleichzeitig laufenden Berechnungen abzuwarten. Die Aufteilung der Teilschritte auf mehrere Kernels dient somit zur Vermeidung von Wettbewerbsbedingungen: Mit Abschluss der Berechnungen eines Teilschrittes und der erfolgreichen Beendigung des Kernels kann davon ausgegangen werden, dass alle Daten vollständig in den Speicher geschrieben wurden und der Datensatz kann als Eingabe für den nächsten CUDA Kernel dienen.

Im Anschluss werden die implementierten Teilschritte beschrieben (siehe auch Abbildung 4.1):

- **1. Aufbau der Nachbarlisten** Mehrere Arbeitsschritte greifen auf die Daten der Nachbarn eines Vertex zu. Im Rahmen des Darstellungsvorganges wird keine Information über die Nachbarn eines Vertex bereitgestellt, und eine Suche nach Nachbarvertizes in den Daten des Dreiecksnetzes zur Laufzeit würde eine große Menge an Ressourcen erfordern. Stattdessen wird eine Hilfsdatenstruktur erstellt, die schnellen Zugriff auf diese Information ermöglicht. Als Eingabe dieses Schrittes dient die Dreiecksliste des Objekts. Für jeden Vertex wird eine Liste der Nachbarn gespeichert. Sie fungiert als Nachschlagetabelle, wenn die Dreiecke gesucht werden.

- **2. Berechnung des statischen Vertexfehlers** Der initial berechnete Vertexfehler greift ausschließlich auf die statischen Daten des Dreiecksnetzes zurück. Aus ihnen - und unter Zuhilfenahme der Datenstruktur aus dem vorherigen Schritt - wird der Fehlerwert vorberechnet und in den Vertexdaten abgelegt. Diese Information stellt den Ausgangspunkt des Vereinfachungsprozesses dar.

- **3. Klassifizierung** Im ersten Schritt nach Abschluss der Vorberechnung findet eine Klassifizierung aller Vertizes anhand des in Abschnitt 3.3 beschriebenen Verfahrens statt. Die Vertizes werden mit Hilfe des Vertexfehlers und der Position bzw. Blickrichtung des Betrachters als beizubehaltend oder zu entfernend markiert.

- **4. Aufbau der Verschiebungslisten** Nach Abschluss der Klassifizierung aller Vertizes wird die Liste der initialen Verschiebungskandidaten - also aller Vertizes, die im ersten Iterationsschritt der Vereinfachung entfernt werden sollen - befüllt.

- **5. Verschiebungstests, Verschiebung und Aktualisierung der Verschiebungslisten** Es folgt die Erstellung einer Liste aller möglichen Ersatzpositionen. Im Anschluss wird überprüft, welche der darin gespeicherten Punkte nach den in Kapitel 3 beschriebenen Tests für den Erhalt der Netzintegrität gültig sind. Weiters bewertet dieser Schritt die erlaubten Ersatzpositionen und wählt daraus eine eingeschränk-

te Kantenreduktion. Im Anschluss kommt es zur eigentlichen Verschiebung, bei der die modifizierten Vertexdaten in den Ausgabespeicher geschrieben werden. Nach Abschluss dieser Operation muss eine Analyse der Nachbarn von entfernten Vertizes durchgeführt werden, um die Verschiebungskandidaten für den nächsten Iterationsschritt zu bestimmen.

- **6. Reklassifizierung** Jedem Verschiebungsschritt folgt eine Reklassifizierung. Dieser Vorgang wird auf allen in der aktualisierten Liste gespeicherten Verschiebungskandidaten ausgeführt. Neben der Überprüfung, ob die initiale Klassifizierung noch Gültigkeit besitzt, werden mögliche blockierte Verschiebungen erkannt und behandelt. Gleichzeitig mit der Reklassifizierung wird die Liste der Verschiebungskandidaten erneut aktualisiert, so dass in diesem Arbeitsgang reklassifizierte Vertizes im nächsten Iterationsschritt nicht fälschlicherweise bearbeitet werden.

- **7. Aufbau des vereinfachten Dreiecksnetzes** Um zu vermeiden, das Dreiecksnetz nach jedem Verschiebungsschritt zu aktualisieren und die Nachbarlisten (Schritt 1) neu zu generieren, operieren die beschriebenen Schritte auf den Daten des ursprünglichen Dreiecksnetzes. Wird ein Vertex verschoben, werden seine Daten (Position, Normalvektor, ...) durch jene des Verschiebungszieles ersetzt. Im vereinfachten Dreiecksnetz sind somit redundante Vertizes und Dreiecke enthalten, die nach Abschluss aller Iterationsschritte entfernt werden, um eine finale Version des resultierenden Dreiecksnetzes zu erstellen. Alle auf Linien oder Punkte reduzierten Dreiecke werden aus dem Datensatz entfernt.

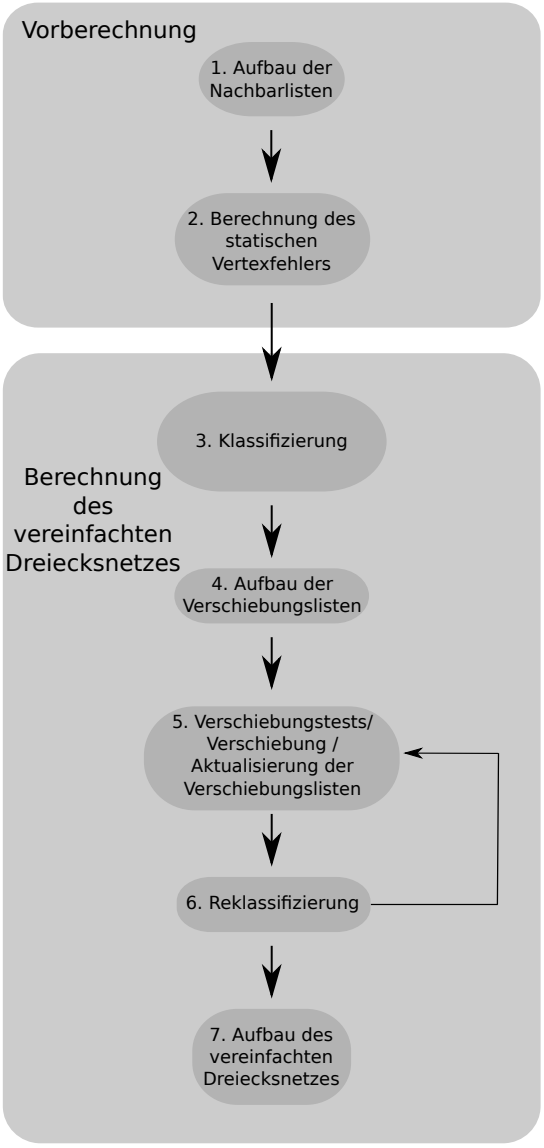

Abbildung 4.1: Ablauf der Implementierung

Im Anschluss werden die notwendigen Datenstrukturen sowie die Teilschritte der Implementierung im Detail beschrieben und die Gründe für Entscheidungen bzw. Probleme bei der Umsetzung dargelegt.

4.2 Anpassung der Vertexdaten und Hilfsdatenstrukturen

Neben den statischen Vertexdaten des darzustellenden Objekts kommen für die Vereinfachung noch weitere Informationen zum Einsatz:

- **1. Per-Vertex Daten** In den per-Vertex Daten des Dreiecksnetzes werden zusätzliche Informationen aus der Vorberechnung abgelegt. Dieser Abschnitt präsentiert die gespeicherten Daten.

- **2. Hilfsdatenstrukturen** Beschreibt eine Menge von Datenstrukturen, die für die Implementierung der Vereinfachung eingesetzt wird.

4.2.1 Per-Vertex Daten

Die für jeden Vertex gespeicherten Daten werden erweitert, um alle in Kapitel 3 beschriebenen Werte zu beinhalten:

- **Vertexfehler** Der in der Vorberechnung ermittelte Vertexfehler ist für jeden Vertex zu speichern. Er wird in einem Arbeitsschritt aus den statischen Daten berechnet und im Datensatz abgelegt. Dieser Vertexfehler kann während der Ausführung der Vereinfachung invalidiert und neu berechnet werden, so dass stets ein aktueller Wert im Speicher liegt.

- **Vertexklassifizierung** Die Klassifizierung gibt an, ob ein Vertex des Dreiecksnetzes beibehalten oder entfernt werden soll. Analog zum Fehlerwert kann dies durch spätere Schritte des Verfahrens verändert werden, um auf die Zwischenergebnisse bei der Vereinfachung einzugehen.

- **Sprungzahl** Wie in Unterabschnitt 3.3.4 erläutert kommt für die Skalierung des Vertexfehlers eine Sprungzahl zum Einsatz. Sie speichert die minimale Anzahl an Kanten zu einem original beibehaltenen Vertex. Die Sprungzahl muss ebenfalls für jeden Vertex gespeichert sein und wird nach einer Vertexverschiebung für alle Nachbarn ermittelt. Somit kann bei der Berechnung der Reklassifizierung auf diesen Wert zugegriffen werden.

- **Datenpuffer** Die letzte benötigte Information dient der Übergabe von Daten zwischen Iterationsschritten. Dieses Datenfeld wird bei der Vertexverschiebung mit Informationen über die Nachbarvertizes befüllt und dient der Erkennung von gesperrten Verschiebungen.

Neben den Vertexdaten sind noch zusätzliche Hilfsdatenstrukturen vonnöten.

4.2.2 Hilfsdatenstrukturen

Die zum Einsatz kommenden, zusätzlichen Datenstrukturen sind:

- Nachbarliste
- Liste der Verschiebungskandidaten
- Kopie der Vertexdaten

Die Nachbarliste speichert für jeden Vertex eine Liste mit den Indizes der benachbarten Vertizes. Die Berechnung fordert das Auffinden aller Dreiecke, die mit dem Vertex gebildet werden und kann daher einen erheblichen Leistungsaufwand bedeuten. Somit ist es praktikabel, dieses Vorgehen in einen Vorberechnungsschritt auszulagern und die Ergebnisse in einer Datenstruktur abzulegen.

Nach durchgeführten Verschiebungen und Reklassifizierungen wird auf diese Liste der Nachbarn zurückgegriffen, um neue Verschiebungskandidaten zu bestimmen und deren Daten wie etwa den Vertexfehler oder die Sprungzahl zu aktualisieren. Einen weiteren Einsatz findet diese Datenstruktur bei der Behandlung von sich gegenseitig sperrenden Verschiebungen. Hierbei ist Kenntnis über die Nachbarvertizes vonnöten, um diese zu finden und zu beseitigen.

Die Liste der Verschiebungskandidaten speichert die Indizes aller Vertizes, die gegenwärtig verschoben werden sollen. Diese Liste wird initial nach der Durchführung der Vertexklassifizierung befüllt und nach jeder Verschiebungsoperation und Reklassifizierung neu aufgebaut, so dass nur die aktuellen Kandidaten gespeichert sind. Nach jedem Verschiebungsschritt werden die hier abgelegten Vertizes auf mögliche Reklassifizierungen und gesperrte Verschiebungen überprüft. Im nächsten Verschiebungsschritt sind nur mehr die tatsächlich zu entfernenden Kandidaten enthalten.

Die Liste der Verschiebungskandidaten bildet gleichzeitig auch die Abbruchbedingung bei der Vereinfachung. Sobald sie leer ist, wird die Iteration der Verschiebungen beendet.

Die letzte zusätzliche Datenstruktur ist eine zweite Instanz der Vertexdaten. Die in Kapitel 3 vorgestellten Verschiebungstests zum Erhalt der Netzintegrität beruhen auf den aus dem vorigen Iterationsschritt resultierenden Dreiecken. Durch die parallele Abarbeitung der Verschiebungskandidaten ist Kenntnis über die Daten des vorherigen Iterationsschrittes notwendig. Um zu vermeiden, dass sie von Ergebnissen der gegenwärtig durchgeführten Verschiebungen überschrieben werden, kommen getrennte Speicherbereiche für die Ein- und Ausgabedaten zum Einsatz. Wettbewerbsbedingungen bei der Ausführung der individuellen Vereinfachungsoperationen werden so vermieden.

Somit dient eine Instanz der Vertexdaten als Eingabe für die Verschiebungstests, die andere kommt zum Einsatz, die Ausgaben zu schreiben und sie im nächsten Iterationsschritt als Eingabedaten wieder zu lesen. Ist ein Teilschritt abgeschlossen, werden die Eingabedaten durch die Ausgabedaten ersetzt. Dies ist erforderlich, da bei der Ausgabe nur ein Teil der Vertexdaten modifiziert wird, ein einfacher Wechsel der Ziel- und Quelldaten ist somit nicht möglich.

Im weiteren Verlauf werden die einzelnen Teilschritte der Implementierung im Detail erläutert, wobei auf Schwierigkeiten und Details eingegangen wird.

4.3 Aufbau der Nachbarlisten

Für die Vorberechnung des statischen Vertexfehlers und die meisten Berechnungen für die Vereinfachung des Dreiecksnetzes ist Kenntnis über die benachbarten Vertizes und die daraus gebildeten Dreiecke vonnöten. Aus diesem Grund findet der Aufbau der Nachbarlisten als erster Schritt statt.

Um während des dynamischen Teils der Vereinfachung eine aufwändige Suche nach den Nachbarvertizes zu vermeiden, wird ein Datenfeld angelegt, das für jeden Vertex eine Liste mit den Indizes der benachbarten Vertizes bereithält. Die Speicherung muss gleichzeitig die Information enthalten, wie Dreiecke mit den Nachbarvertizes gebildet werden. Beim Zugriff auf dieses Datenfeld kann der einen Vertex umgebende Ring aus Dreiecken rekonstruiert werden.
Im Gegensatz zur Bilderzeugung ist für die Vereinfachung die Ausrichtung der Dreiecke irrelevant, weil in den Berechnungen zwar auf eine Veränderung der Ausrichtung der Dreiecksnormalen in Bezug auf die Position des Betrachters geachtet wird, hierfür aber nur eine Änderung der Ausrichtung relevant ist, nicht jedoch ob die Vorder- oder Rückseite sichtbar ist. Aus diesem Grund kann die Reihenfolge der Vertizes bei der Speicherung der Nachbarliste ignoriert und so der benötigte Speicherplatz reduziert werden.

Die Vertexgruppeneliminierung ist nur für die Vereinfachung von 2D-mannigfaltigen Dreiecksnetzen mit Grenzen bestimmt. Laut der Definition der 2D-Mannigfaltigkeit mit Grenzen (siehe Unterabschnitt 2.1.3.1) können bei einem derartigen Dreiecksnetz alle Dreiecke $T = \{T^1, T^2, ...T^n\}$, die einen Vertex V enthalten, auf eine Scheibe abgebildet werden. Jede Kante, die zwischen V und einem Nachbarvertex V' liegt, kann mit dieser Definition entweder in einem oder maximal zwei Dreiecken enthalten sein. Somit werden durch die Dreiecke, die V enthalten, ein oder mehrere Dreiecksfächer aufgespannt (Beispiel für Vertex mit mehreren Fächern in Abbildung 4.2).

Ein Dreieck dieses Fächers wird als inneres Dreieck bezeichnet, wenn seine beiden Kanten, die mit V gebildet werden, jeweils in exakt zwei Dreiecken enthalten sind. Im Gegensatz dazu kann eine Kante nur in einem einzelnen Dreieck enthalten sein. In die-

sem Fall handelt es sich um eine äußere Grenze. Enthalten die Dreiecke in T zumindest eine äußere Grenze, ist V nicht vollständig von Dreiecken umgeben und somit Teil der Begrenzung des Dreiecksnetzes.

Die Speicherung muss berücksichtigen, dass mehrere äußere Grenzen in einer dieser Scheiben existieren können und somit multiple Fächer für einen Vertex vorhanden sind. Abbildung 4.2 zeigt ein Beispiel für eine Scheibe mit mehreren äußeren Grenzen (hellgrau). Vier Kanten sind in diesem Beispiel nur in einem Dreieck enthalten.

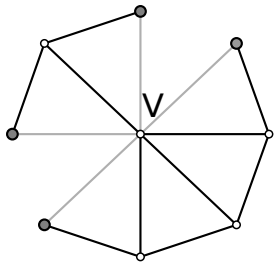

Abbildung 4.2: Fächer mit multiplen „äußeren Grenzen“

Die Speicherung der Nachbarn in Form eines Dreiecksfächers hat den Vorteil, dass jeder Vertex mit einer gemeinsamen Kante nur einmal enthalten sein muss und somit die Redundanz - und folglich auch der erforderliche Speicherplatz - reduziert wird. Für jeden Vertex wird das Datentupel $V = \{n, v_0, v_1, ..., v_i\}$ abgelegt, wobei n die Anzahl der benachbarten Vertizes und v_i den Index eines durch eine Kante verbundenen Vertex darstellt.

Nach Aufbau und Befüllung dieser Datenstruktur wird sie für die Ermittlung des statischen Vertexfehlers eingesetzt. Dieser Schritt erfolgt ebenfalls vor der dynamischen Berechnung, und die Ergebnisse werden in den Vertexdaten gespeichert.

4.4 Berechnung des statischen Vertexfehlers

Der statische Vertexfehler wird für jeden Vertex individuell berechnet und kann in den per-Vertex Daten gespeichert werden. Die Berechnung des Vertexfehlers setzt sich aus zwei Komponenten zusammen:

1. Berechnung des initialen Vertexfehlers

2. Berechnung des regelmäßigen Gitters und Manipulation des Vertexfehlers

Der statische Vertexfehler wird anhand des in Unterabschnitt 3.3.1 vorgestellten Verfahrens berechnet. Die Nachbarliste kommt bei diesem Arbeitsschritt zum Einsatz, um eine aufwändige Suche nach den Nachbarvertizes zu vermeiden und die Berechnungszeit zu reduzieren.

Anschließend an die Berechnung des Vertexfehlers findet die Manipulation mittels des regelmäßigen Gitters statt (siehe Unterabschnitt 3.3.2). Als Ausgangspunkt ist das Begrenzungsvolumen des Objekts zu berechnen. Das Volumen wird ausschließlich für diesen Vorberechnungsschritt benötigt und kommt für keine weiteren Berechnungen zum Einsatz. Die Ausrichtung zur Kamera ist irrelevant und eine Transformation zur Laufzeit nicht erforderlich. Aus diesen Gründen wird das einfachere, achsparallele Begrenzungsvolumen eingesetzt und nicht auf die objektorientierte Variante zurückgegriffen.

Die Anzahl j der gewünschten Unterteilungen und somit die Auflösung der feinsten Stufe wird vom Benutzer gewählt, um die Anpassung an das entsprechende Objekt zu gewährleisten. Zusätzlich kann die Stufe der gröbsten Auflösung i definiert werden, welche die unbedingt beizubehaltenden Vertizes bestimmt. Die zum Einsatz kommenden Punkte gehören somit zu den Stufen $G_i, G_{i+1}, ...G_j$. Die Knoten der Stufe G_0 liegen in den Eckpunkten des Begrenzungsvolumens. Die weiteren Stufen werden durch Halbierung der Abstände zwischen den Punkten erzeugt. Für jede Dimension wird die Distanz zwischen den Punkten getrennt berechnet, um die Ausdehnung des Volumens zu berücksichtigen. Quelltext 4.1 zeigt die Berechnung der Punkte in Pseudocode. In Zeile 6 wird die Anzahl der Punkte für die gegenwärtige Stufe berechnet. Zeilen 7 bis 9 bestimmen die Abstände zwischen zwei Punkten in den drei Dimensionen. In Zeilen 17 bis 22 werden die Punkte erzeugt, die Position im Raum und die Stufe zugewiesen und der Punkt gespeichert.

```
1   PointList CalculatePoints(bbX, bbY, bbZ, bbWidthx, bbWidthy, bbWidthz,
        numLevels)
2   {
3     numPoints = 2
4     for numSteps=0:numLevels
5     {
6       numPoints = (numPoints * 2)-1
7       widthX = bbWidthx / (numPoints-1)
8       widthY = bbWidthy / (numPoints-1)
9       widthZ = bbWidthz / (numPoints-1)
10
11      for x=0:(numPoints-1)
12      {
13        for y=0:(numPoints-1)
14        {
15          for z=0:(numPoints-1)
16          {
17            point(bbX, bbY, bbZ)
```

```
18              point.x += widthX * x
19              point.y += widthY * y
20              point.z += widthZ * z
21              point.level = numSteps
22              PointList.Add(point)
23            }
24          }
25        }
26      }
27    }
```

Quelltext 4.1: Berechnung der Gitterpunkte

Durch eine sehr geringe Ausdehnung einer Dimension könnten viele Punkte mit kleinem Abstand entstehen. Dadurch würde ein Großteil bis hin zu allen Vertizes eines Dreiecksnetzes für eine Manipulation des Vertexfehlers ausgewählt. Dies soll vermieden werden. Daher definiert der Benutzer einen minimalen Abstand zwischen den Knoten des Gitters. Beim Erschaffen einer neuen Stufe werden die Abstände zwischen den Punkten auf eine Unterschreitung der Untergrenze überprüft. Wird eine Unterschreitung festgestellt, kommt für diese Dimension der nicht halbierte Abstand zweier Punkte zum Einsatz.

Abbildung 4.3 zeigt ein Beispiel für ein Objekt (Kugel), das von dem achsparallelen Begrenzungsvolumen umgeben ist. Die Punkte für G_0 liegen in den Eckpunkten des Volumens. Durch die Halbierung der Distanz zwischen den Punkten wurden die zusätzlichen Punkte der Stufe G_1 eingefügt.

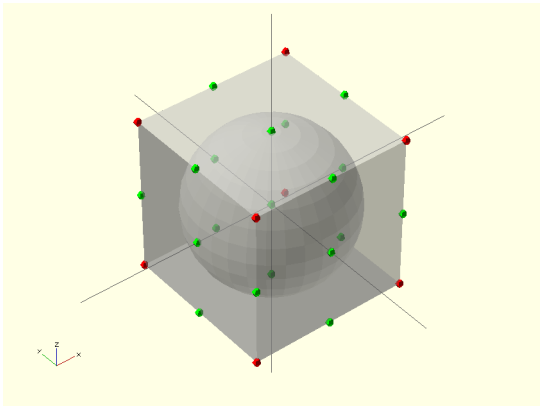

Abbildung 4.3: Objekt mit Begrenzungsvolumen und regelmäßigem Gitter (2 Stufen)

Dieser Arbeitsschritt generiert eine Menge von Stufen $G = \{G_i, G_{i+1}, ..., G_j\}$. Für jede Stufe ist eine Menge von Punkten erzeugt worden. Der Ansatz aus Unterabschnitt 3.3.2 versucht, für jeden dieser Punkte einen entsprechenden Vertex des Drei-

ecksnetzes zu ermitteln. Über alle Stufen wird iteriert und jedem Punkt ein Begrenzungsvolumen zugewiesen, innerhalb dessen ein Vertex des Dreiecksnetzes gesucht wird. Diese Volumen liegen parallel zum Begrenzungsvolumen B für das gesamte Objekt und sind somit ebenfalls achsparallel.

Die Seitenlängen der Teilvolumen entsprechen den Abständen zwischen den Punkten derselben Stufe. Die Volumen jener Knoten, die auf den Kanten von B liegen, beinhalten daher auch Bereiche außerhalb von B. Sie sind so zu modifizieren, dass alle Bereiche außerhalb von B verworfen werden, da sich hier keine Vertizes des Dreiecksnetzes befinden. Ein Beispiel für Volumen für G_0 und G_1 findet sich in Abbildung 4.4. Auf der linken Seite sind die Volumen für G_0 dargestellt. Die Punkte befinden sich hier in den Eckpunkten. Bei den Volumen für G_1 auf der rechten Seite ist das Verwerfen von Bereichen von B gut erkennbar. Die Volumen in den Randbereichen sind dadurch kleiner als jene in der Mitte.

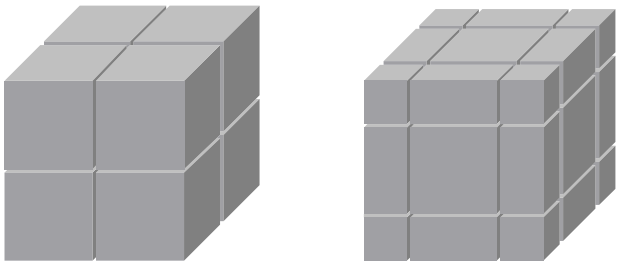

Abbildung 4.4: Aufbau der Teilräume für Stufen G_0 und G_1

Durch dieses Vorgehen ergibt sich in jeder Stufe $G_k \in G$ eine Unterteilung von B in eine Menge von Teilräumen. Für jeden Teilraum wird versucht, einen entsprechenden Vertex V des Dreiecksnetzes zu finden.

Anschließend folgt die Auswahl eines Vertex für die einzelnen Punkte des Gitters. Zunächst wird für jeden Teilraum B_i^k die Menge aller Vertizes des Dreiecksnetzes bestimmt, die sich in B_i^k befindet. Volumen, die eine leere Menge enthalten, werden an dieser Stelle verworfen. Anhand des in Unterabschnitt 3.3.2 beschriebenen Verfahrens wird für jedes verbleibende Volumen ein Vertex des Dreiecksnetzes ausgewählt. In Quelltext 4.2 wird über alle Vertizes innerhalb einer Zelle iteriert. In Zeilen 10 und 11 wird für jeden gefundenen Vertex der gewichtete Fehlerwert berechnet und in Zeilen 12 bis 15 der größte gewichtete Fehlerwert bestimmt.

```
1   FindVertex(VertexList, numVertizes, point, boundingBox)
2   {
3     minWError = MAX_FLOAT
4     selectedVertex = -1
5
6     for v=0:numVertizes
```

```
7       {
8         if IsIn(VertexList[v], boundingBox)
9         {
10          distance = sqrt((point - VertexList[v].position) * (point -
                VertexList[v].position))
11          weightedError = WeightedError(VertexList[v], distance, boundingBox)
12          if weightedError > minWError
13          {
14            selectedVertex = v
15            minWError = weightedError
16          }
17        }
18      }
19    }
```

Quelltext 4.2: Auswahl von Vertizes

Für jeden auf diese Weise selektierten Vertex wird der Fehlerwert manipuliert. Bei der Berechnung des Vertexfehlers findet keine Normalisierung der Daten statt. Die resultierenden Werte sind somit abhängig von der Skalierung des gewählten Dreiecksnetzes.

Ein weiteres Problem ergibt sich aus Dreiecksnetzen, die keine gleichmäßige Vernetzung aufweisen. Die Dreiecksgrößen bzw. Abstände zwischen benachbarten Punkten können in verschiedenen Bereichen des Objekts variieren und so auch die Größe des Vertexfehlers beeinflussen. Aus diesem Grund ist das Ausmaß der durchgeführten Manipulation des Vertexfehlers eine vom Benutzer gesteuerte Eingabe. Diese erlaubt eine genaue Anpassung an die gewünschten Ergebnisse. Zudem kann der Verlauf der Auswirkung der Manipulation zwischen den einzelnen Stufen frei gesteuert werden.

Diese Operation schließt die Vorberechnung ab. Der nächste Schritt ist die Klassifizierung.

4.5 Klassifizierung und Bestimmung der initialen Verschiebungskandidaten

Nach Abschluss der Vorberechnung ist die Klassifizierung der nächste Arbeitsschritt (Abbildung 4.1). Nach der Klassifizierung aller Vertizes sollen die Verschiebungskandidaten bestimmt werden.

Aufgrund der parallelen Bearbeitung können diese beiden Aufgaben nicht in einem Schritt erfolgen. Die Bestimmung der Verschiebungskandidaten erfordert Kenntnis über die Klassifizierung aller Nachbarn. Vor Abschluss der Klassifizierung ist nicht bestimmt, bei welchen Vertizes das Ergebnis der Berechnung bereits geschrieben wurde bzw. wel-

che Vertizes noch analysiert werden müssen. Folglich ist das Ergebnis der Vertexklassifizierung abzuwarten, bevor die Liste der Verschiebungskandidaten ermittelt werden kann. Die beiden Arbeitsschritte führen folgende Aufgaben durch:

1. Analyse der Vertizes und Bestimmung der Klassifizierung

2. Aufbau der Liste der initialen Verschiebungskandidaten

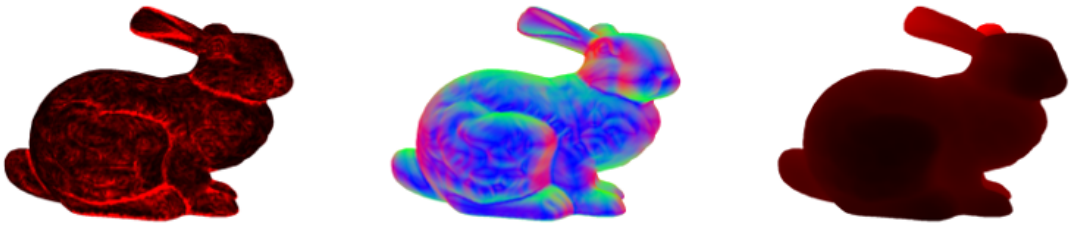

Abbildung 4.5: Eingabedaten für die Klassifizierung: Vertexfehler (links), Normalvektoren (Mitte) und Distanz zur Kamera (rechts)

Die benötigten Eingabedaten für die Klassifizierung sind der errechnete Vertexfehler, der Normalvektor und der Abstand zwischen Kamera und Vertex. Abbildung 4.5 zeigt Beispiele, in denen diese Daten visualisiert wurden. Auf der linken Seite ist der berechnete Vertexfehler. Rote Bereiche zeigen einen hohen Fehlerwert. Je dunkler die Farbe, desto niedriger ist der gespeicherte Vertexfehler. In der Mitte sind die gespeicherten Normalvektoren für das Objekt dargestellt und auf der rechten Seite die Distanz zwischen Betrachter und Teilen des Objekts (die Farbe rot signalisiert eine größere Distanz).

Anhand des in Unterabschnitt 3.3.3 beschriebenen Verfahrens wird ermittelt, ob ein Vertex zu entfernen oder beizubehalten ist. Der Pseudocode in Quelltext 4.3 zeigt diese Berechnung. In Zeile 1 wird der Skalierungsfaktor für die Distanz zwischen Betrachter und der Vertexposition berechnet. Zeilen 2 und 3 ermitteln die Skalierung mittels Winkel zwischen Normalvektor und Blickvektor. In Zeilen 5 bis 8 wird die Klassifizierung bestimmt und gespeichert.

```
1   float distance = sqrt((camera.position - vertex.position) *
        (camera.position - vertex.position))
2   float angle = dot(camera.viewDir, vertex.normal)
3   float scaleAngle = 1 - (angle * angle)
4
5   if(vertex.error * scaleAngle < threshold * distance)
6     vertex.classification = VERTEX_REMOVE
```

```
7     else
8       vertex.classification = VERTEX_KEEP
```

Quelltext 4.3: Berechnung der Klassifizierung

Das Ergebnis dieser Berechnung ist eine Reihe von Vertizes des Dreiecksnetzes, die für ein Entfernen markiert wurde.

Abbildung 4.6: Beispiel einer Menge von für ein Entfernen markierten Vertizes

Abbildung 4.6 zeigt ein Beispiel für ein Ergebnis der Analyse. Alle rot markierten Punkte sollen in den weiteren Arbeitsschritten aus dem Dreiecksnetz entfernt werden.

Nach dem Ende der initialen Bewertung und dem Schreiben aller Klassifizierungsdaten können die Kandidaten für die erste Iteration der Verschiebungen gesucht werden. Die Liste der Verschiebungskandidaten speichert die Indizes der Vertizes, die im nächsten Verschiebungsschritt auf die Position beizubehaltender Nachbarn verschoben werden sollen.

Bei der weiteren Verarbeitung wird davon ausgegangen, dass alle in der Liste gespeicherten Vertizes die beiden, folgenden Eigenschaften erfüllen:

- **Zu entfernender Vertex** Jeder Vertex in der Liste soll aus dem Dreiecksnetz gelöscht werden.

- **Beizubehaltende Nachbarn** Jeder Vertex in der Liste muss mindestens einen beizubehaltenden Nachbarn besitzen. Fehlt dieser, so kann der Vertex nicht verschoben werden.

Um zu vermeiden, beim Befüllen der Liste der initialen Verschiebungskandidaten erneut alle Vertizes des Dreiecksnetzes zu betrachten, wird bei der Klassifizierung die leere Kandidatenliste mit den Indizes aller beizubehaltenden Vertizes befüllt. Die Bestimmung der Verschiebungskandidaten erfordert somit nicht mehr die Analyse aller

Vertizes des Dreiecksnetzes. Es ist ausreichend, die Nachbarn der von der Klassifizierung in der Kandidatenliste abgelegten, beizubehaltenden Vertizes zu betrachten und zu entfernende Nachbarn in die Liste der Verschiebungskandidaten aufzunehmen.

Die Implementierung geht davon aus, dass ein Index nur einmal in der Liste der Kandidaten enthalten ist. Bei einer Verletzung dieser Bedingung würde es zu einer mehrmaligen Behandlung eines Vertex innerhalb desselben Iterationsschrittes kommen. Das Resultat einer derartigen Operation ist nicht definiert.

Ein Verschiebungskandidat kann mehrere beizubehaltende Nachbarn besitzen und somit mehrmals als Verschiebungskandidat identifiziert werden. Um sicherzustellen, dass der Vertex nur einmal als Kandidat selektiert wird, lässt sich die Sprungzahl einsetzen. In diesem Schritt werden nur Nachbarn von initial als beizubehaltend markierten Vertizes betrachtet. Die Sprungzahl muss für diese Vertizes laut Definition in Unterabschnitt 3.3.4 den Wert 1 betragen.

Soll ein Nachbar in die Liste der Verschiebungskandidaten aufgenommen werden, so kommt eine Operation zum Einsatz, die den Wert 1 in den Speicherbereich der Sprungzahl schreibt und dabei garantiert, dass bei der parallelen Ausführung keine gleichzeitigen Schreibzugriffe auf den Speicherbereich erfolgen. Dies ist notwendig, um Wettlaufsituationen auf den Ausgabedaten zu vermeiden. Das Ergebnis dieser Operation gibt an, ob der zu entfernende Nachbar bereits zuvor als Verschiebungskandidat gefunden wurde. Somit lässt sich feststellen, ob ein Vertex bereits in die Liste der Verschiebungskandidaten aufgenommen wurde. Nach Berechnung des ersten Schrittes der Klassifizierung wurde die Kandidatenliste mit den beizubehaltenden Punkten befüllt. Die Kandidatenliste funktioniert nach dem FIFO Prinzip („first in/first out“, zuerst hinzugefügte Werte werden als erste wieder entnommen).

An dieser Stelle werden die zu bearbeitenden Indizes aus der Kandidatenliste entnommen und auf die zur Verfügung stehenden Rechenkerne verteilt. Die Einträge der vergebenen Vertizes werden gleichzeitig aus der Liste gelöscht.

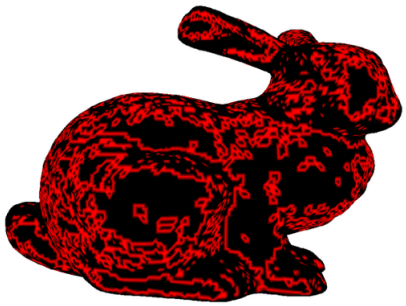

Abbildung 4.7: Beispiel für initiale Verschiebungskandidaten

Abbildung 4.7 zeigt einen Beispielfall für die bestimmten Verschiebungskandidaten. Alle rot markierten Vertizes sind Kandidaten und sollen im ersten Verschiebungsschritt entfernt werden.

Mit der Berechnung der initialen Verschiebungskandidaten werden die von der Klassifizierung in die Liste geschriebenen Indizes der beizubehaltenden Punkte entfernt und jene der korrekten, initialen Verschiebungskandidaten eingetragen.

Mit Abschluss dieser Arbeitsschritte ist die Vertexklassifizierung beendet. Somit kann die iterative Vereinfachung beginnen. An dieser Stelle des Verfahrens ist die Klassifizierung aller Vertizes bekannt und die Liste der Verschiebungskandidaten befüllt. Im nächsten Abschnitt wird die Implementierung der Verschiebungen behandelt.

4.6 Vertexverschiebungen

Die Vertexverschiebung setzt sich aus zwei Teilschritten zusammen:

- **1. Verschiebungstests** Hier werden alle beizubehaltenden Nachbarn gesucht, die mögliche Ersatzpositionen darstellen. Jede dieser Positionen wird mittels der Verschiebungstests aus Unterabschnitt 3.4.1 und 3.4.2 auf ihre Eignung als Verschiebungsziel analysiert. Illegale Positionen werden gesperrt, um Fehler im Dreiecksnetz auszuschließen.

- **2. Vertexverschiebung und Aktualisierung der Vertexlisten** Der zweite Teilschritt ist die eigentliche Verschiebung. Alle potentiellen Ersatzpositionen werden bewertet und die „beste“ wird ausgewählt. Im Anschluss werden die Daten im Ausgabespeicher angepasst, um die Vereinfachung des Dreiecksnetzes abzuspeichern.

4.6.1 Verschiebungstests

Die Eingabedaten für diesen Teilschritt sind die Liste der Verschiebungskandidaten und die statischen Vertexdaten. In diesem Arbeitsschritt werden die Kandidaten individuell behandelt.

Der erste Schritt besteht in der Betrachtung aller Nachbarvertizes. Beizubehaltende sind mögliche Ersatzpositionen, und es muss überprüft werden, ob eine Verschiebung auf ihre Position die Integrität des Netzes verletzen würde. Die Nachbarn werden überdies zum Aufstellen der Ebenen für die Verschiebungstests benötigt.

Wie in Unterabschnitt 3.4.1 beschrieben, wird für einen Verschiebungskandidaten V ein Bereich $B(V)$ definiert, innerhalb dessen eine Verschiebung stattfinden darf. $B(V)$

ist durch eine Menge an Ebenen begrenzt. Um die Menge an Ebenen für einen Verschiebungskandidaten V zu ermitteln, muss der gesamte Dreiecksfächer bestimmt und für jedes Dreieck die Anzahl der aktuell zu verschiebenden Eckpunkte festgelegt werden. Daraus ergibt sich, welcher Test - und folglich welche Ebenen - zum Einsatz kommt. Da der Dreiecksfächer bereits in den Vorberechnungen bestimmt und in der entsprechenden Datenstruktur abgelegt wurde, ist hier ein effizienter Zugriff möglich.

Jede Kante zwischen V und einem benachbarten, beizubehaltenden Vertex V' ist eine mögliche Ersatzposition. Es wird bestimmt, welche V' gültige Ziele darstellen und welche eine Verletzung der Netzintegrität verursachen. Wie in Unterabschnitt 3.4.1 beschrieben, werden Schnittpunkte zwischen der Kante, die V mit V' verbindet, und den Ebenen in $B(V)$ gesucht. Wird ein Schnittpunkt der Kante mit einer der Ebenen gefunden, so ist V' als Verschiebungsziel ausgeschlossen. Die Position eines Schnittpunktes ist für diese Operation irrelevant.

Der benötigte Leistungsaufwand für die Verschiebungstests lässt sich reduzieren, indem anstatt eines Schnittpunktes zwischen einer Geraden und einer Ebene nur die Lage der beiden Punkte (V und V') überprüft wird. Befinden sich beide auf derselben Seite einer Ebene, so liegt kein Schnittpunkt auf der Kante - also zwischen V und V' - vor. Finden sich V und V' jedoch auf gegenüberliegenden Seiten einer Ebene, so besteht ein Schnittpunkt auf der Kante und der Nachbar kommt nicht für eine Verschiebung in Frage. Abbildung 4.8 zeigt ein Beispiel für diesen Ansatz. Links befinden sich beide Punkte auf derselben Seite der Ebene. Es existiert kein Schnittpunkt zwischen Kante und Ebene. Rechts liegen die Punkte auf unterschiedlichen Seiten der Ebene. Hier existiert ein Schnittpunkt auf der Kante.

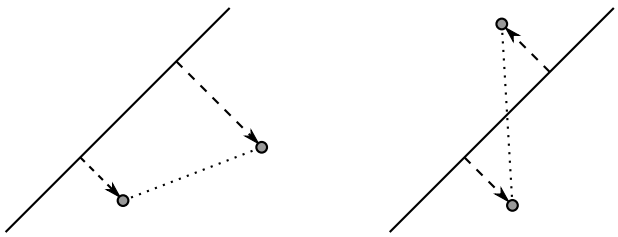

Abbildung 4.8: Punkte auf derselben (links) bzw. verschiedenen Seiten (rechts) einer Ebene

Dieser Test erfordert 2 Skalarprodukte und einen Vergleich für die Berechnung. In Quelltext 4.4 wird in Zeile 3 das Skalarprodukt zwischen Normalvektor der Ebene und Punkt für den ersten Vertex ermittelt. In Zeile 4 wird diese Berechnung für den zweiten Vertex durchgeführt. Zeile 5 multipliziert die beiden Skalarprodukte. Weisen beide dasselbe Vorzeichen auf, so ist das Ergebnis der Multiplikation größer als null und die beiden Vertizes liegen auf derselben Seite der Ebene:

```
1   bool IsAllowed(float4 source, float4 destination, float4 plane)
2   {
3     float p1 = dot(source, plane)
4     float p2 = dot(destination, plane)
5     return (p1 * p2 > 0)
6   }
```

Quelltext 4.4: Erkennen von illegalen Verschiebungen

Eine Optimierung lässt sich erzielen, indem die Ausrichtungen der Normalvektoren der Ebenen bei deren Erzeugung angepasst werden. Der Test in Quelltext 4.4 nimmt an, dass der Ausgangspunkt der Verschiebung auf einer beliebigen Seite der Ebenen liegen kann. Aus diesem Grund muss er - wie das Verschiebungsziel - in die Ebenengleichung eingesetzt werden, um die Überprüfung auszuführen. Ist die Lage von V bezüglich aller Ebenen definiert, so entfällt beim Test einer Ersatzposition eines der beiden Skalarprodukte.

Beim Laden des Dreiecksfächers und dem Aufbau der daraus resultierenden Ebenen wird das Skalarprodukt aus V und einer Ebene e errechnet. Ist $V \bullet e < 0$, so wird die Ausrichtung der Normalen von e negiert. Daraus folgt, dass alle möglichen Ersatzpositionen, für die $V' \bullet e > 0$ gilt, legal sind und die Netzintegrität nicht gefährden. Abbildung 4.9 zeigt ein Beispiel für diesen Ansatz. Der zentrale Vertex soll verschoben werden. Die Ebenen (gestrichelt gekennzeichnet) besitzen alle einen Normalvektor, der dieselbe Ausrichtung in Bezug auf den zentralen Vertex aufweist.

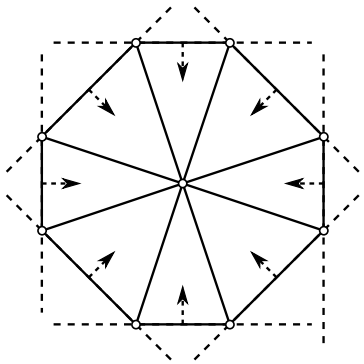

Abbildung 4.9: Identische Ausrichtung aller Normalvektoren

Durch dieses Vorgehen reduziert sich der Verschiebungstest für eine Ersatzposition und eine Ebene auf ein einzelnes Skalarprodukt zweier 4-Komponenten Vektoren, die Berechnung des eigentlichen Schnittpunktes wird vermieden. Quelltext 4.5 zeigt die optimierte Berechnung. In Zeile 3 wird das Skalarprodukt berechnet und verglichen, ob der Vertex auf der erlaubten Seite der Ebene liegt.

```
1    bool IsAllowed(float4 destination, float4 plane)
2    {
3      return (dot(destination, plane) > 0)
4    }
```

Quelltext 4.5: Erkennen von illegalen Verschiebungen

Bei der Überprüfung einer Ersatzposition muss für jede Ebene, die für den Dreiecksfächer erstellt wurde, der Test mit der Verschiebungskante ausgeführt werden. Für die Implementierung wird diese Anforderung jedoch abgeändert: Alle Ebenen, die durch ein Dreieck, das die zu überprüfende Ersatzposition V' enthält, aufgestellt wurden, werden übersprungen. Dies betrifft ausschließlich den ersten und zweiten Verschiebungstest, da beim dritten Fall alle drei Punkte Verschiebungskandidaten sind und somit nie eine mögliche Ersatzposition enthalten ist.

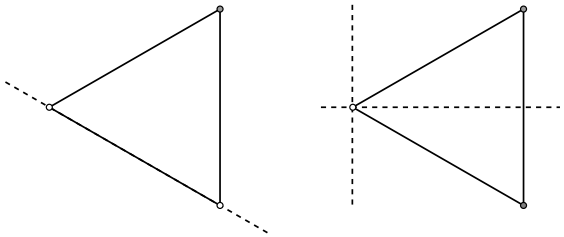

Abbildung 4.10: Ebenen des ersten und zweiten Verschiebungstests

Wie Abbildung 4.10 zeigt, liegen beizubehaltende Punkte bei diesen Tests immer auf den Begrenzungsebenen. Ein Vergleich der Lage dieser Punkte mit jener von V ist somit nicht aussagekräftig.

Überdies könnte durch den Test mit diesen Ebenen eine Verschiebung fälschlicherweise blockiert werden. Ein Einsetzen des Punktes V' sollte 0 ergeben. Aufgrund von Ungenauigkeiten bei der Berechnung von Gleitkommawerten können leichte Abweichungen entstehen. Diese könnten dazu führen, dass der Punkt als nicht auf der Ebene liegend erkannt wird. Die Folge wäre das fälschliche Bestimmen einer illegalen Verschiebung. Das Überspringen der entsprechenden Ebenen verhindert dieses Problem.

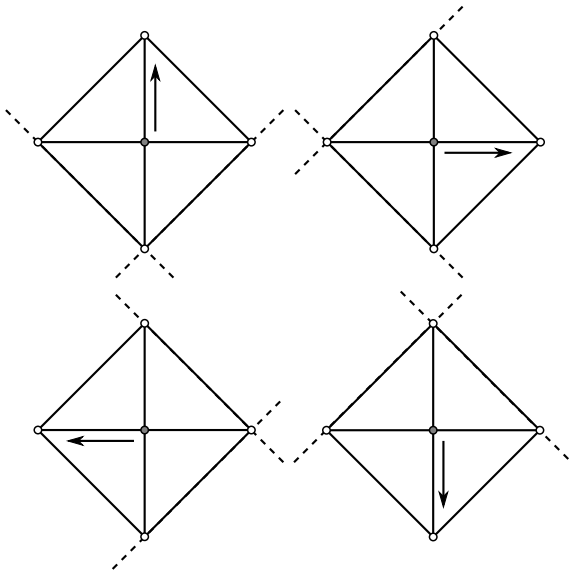

Abbildung 4.11: Einsatz einer Teilmenge der Begrenzungsebenen

Abbildung 4.11 zeigt diesen Ansatz. Jeder der äußeren Vertizes ist ein mögliches Verschiebungsziel. Dies hat zur Folge, dass vier Ersatzpositionen für den zentralen Vertex getestet werden müssen. Jede der vier Teilabbildungen stellt die Ebenen für den Test einer möglichen Ersatzposition dar. Der Pfeil markiert die Kante, für welche die Verschiebungstests ausgeführt werden. Während für den zentralen Vertex vier Ebenen aufgestellt wurden, kommen aufgrund des Auslassens der Ebenen, die das Verschiebungsziel enthalten, für die Tests immer nur zwei Ebenen zum Einsatz.

Das Speichern der Daten einer Ebene benötigt einen 4-Komponenten Gleitkommavektor. Für einen Vertex werden bis zu 2 Ebenen pro Dreieck erstellt. Der maximale Speicherbedarf liegt somit bei 8 Byte pro Dreieck und ist abhängig von der Anzahl der Dreiecke, aus denen der Fächer gebildet wird. Ausgehend von der Anzahl n der Dreiecke im Fächer um V beträgt der Speicherbedarf somit $n * 8$ Byte.

Die Anzahl der erforderlichen Verschiebungstests für ein mögliches Verschiebungsziel ist immer kleiner $n * 2$. Die genaue Zahl hängt ab von der Menge der zu verschiebenden Nachbarn und ob ein oder zwei Dreiecke mit V' gebildet werden. Der Pseudocode in Quelltext 4.6 führt den Verschiebungstest für eine Ersatzposition durch. In Zeile 4 wird über alle Ebenen für einen Verschiebungskandidaten V iteriert. In Zeile 6 wird überprüft, ob die gegenwärtige Ebene die Ersatzposition enthält und die Ebene gegebenenfalls übersprungen. In Zeilen 7 bis 10 wird überprüft, ob der Verschiebungstest die Ersatzposition zulässt und die Verschiebung wenn nötig blockiert.

```
1    //Überprüfen einer Verschiebung von V nach V1
2     list<planes> pList
3     float rating = 0
4     foreach plane p in pList
```

```
5       {
6         if(PlaneCreatedWith(p, V1)) continue
7         if(!IsAllowed(V, V1, p))
8         {
9           V1.blocked = true
10          break;
11        }
12      }
```

Quelltext 4.6: Überspringen von unnötigen Tests

Um eine Ersatzposition zu finden, ist für jede mögliche Verschiebung eine Bewertung gespeichert. Das Ergebnis des Verschiebungstests fließt ein, indem bei jedem Schnitttest mit einer Ebene ein Wert zur bestehenden Bewertung addiert wird. Existiert kein Schnittpunkt mit der Kante, beträgt dieser Wert 0. Schneidet die Kante die Ebene, so wird ein sehr hoher Fehlerwert addiert.

Dieser künstliche Fehler liegt über einer definierten Obergrenze für den Fehlerwert der Verschiebung, was garantiert, dass die gegenwärtig analysierte Verschiebung nicht durchgeführt wird und entweder eine andere Ersatzposition ausgewählt oder keine Verschiebung ausgeführt wird.

Im Anschluss erfolgt für die gültigen Ersatzpositionen eine Bewertung, mit deren Hilfe ein Ziel ausgewählt wird. Die Bewertung und Verschiebung wird im nächsten Abschnitt behandelt.

4.6.2 Vertexverschiebung und Aktualisierung der Vertexlisten

Von den Verschiebungstests wird eine Liste mit erlaubten Ersatzpositionen bestimmt. Die Vertexverschiebung selektiert aus den als legal eingestuften Ersatzpositionen ein Verschiebungsziel und führt die Vereinfachungsoperation durch. Nach Abschluss dieser Berechnungen beginnt die Reklassifizierung. Die Vertexverschiebung ist jener Arbeitsschritt, in dem die eigentliche Vereinfachung des Dreiecksnetzes erfolgt.

Die Vertexverschiebung fasst mehrere Teilschritte zusammen:

1. Bewertung der verbleibenden Verschiebungskandidaten
2. Schreiben der veränderten Vertexdaten
3. Behandlung von nicht durchführbaren Verschiebungen
4. Bestimmung veränderter Nachbarn

5. Aktualisierung der Liste der Verschiebungskandidaten

An dieser Stelle werden - sofern vorhanden - nur mehr die erlaubten Vertexverschiebungen betrachtet. Eine von ihnen soll als Ersatzposition für den Vertex ausgewählt werden. Wie in Unterabschnitt 3.4.4 präsentiert, wird hierfür auf die Idee der Quadrik-Fehlermetrik von Garland und Heckbert [GH97] zurückgegriffen.

Wie in Unterabschnitt 3.4.4 dargelegt, kommt an dieser Stelle ein modifizierter Ansatz der Quadrik-Fehlermetrik zur Anwendung, da die Ersatzposition nicht frei gewählt werden kann. Stattdessen werden die Fehlerwerte der beizubehaltenden Nachbarn laut Quadrik-Fehlermetrik ermittelt.

Für die Bewertung einer gefundenen, legalen Ersatzposition ist lediglich die Vertexposition mit der aufgestellten Matrix zu multiplizieren. Daraus ergibt sich der Fehler $\triangle$ für eine Zielposition V'. Im Anschluss kann die Ersatzposition mit dem geringsten Fehler ausgewählt werden. Quelltext 4.7 zeigt den Pseudocode für diese Operation. In Zeile 3 wird über alle legalen Ersatzpositionen n iteriert. Zeile 5 berechnet $\triangle(n)$ und in Zeilen 6 bis 9 wird die Ersatzposition mit dem minimalen Fehler bestimmt.

```
1       float rating = RATING_MAX
2       Vertex replacementPos = NULL
3       foreach Vertex n in keptNeighbours
4       {
5         float vRating = transpose(n.position) * v.Q * n.position
6         if(vRating < rating)
7         {
8           replacementPos = n
9           rating = vRating
10        }
11      }
```

Quelltext 4.7: Bewertung der Ersatzpositionen

Bei der Verschiebungsbewertung sind mögliche, starke Auswirkungen mancher Verschiebungen auf die Oberfläche zu berücksichtigen. Daher wird ein maximaler, erlaubter Fehler eingeführt, um unerwartete, starke Abweichungen von der Oberflächenform zu vermeiden. Es soll verhindert werden, Vereinfachungen um jeden Preis auszuführen. Stattdessen kann eine Operation nur dann in Erwägung gezogen werden, wenn sich die verursachte Abweichung innerhalb eines gewissen Rahmens bewegt. Eine bessere Qualität des resultierenden Netzes ist die Folge.

Der Benutzer legt die hier zum Einsatz kommende Toleranzgrenze fest. Überschreiten alle gefundenen Bewertungen erlaubter Ersatzpositionen diese, so findet keine Verschiebung statt. Der Vertex bleibt in seiner gegenwärtigen Form erhalten. Dies resultiert in einer Behandlung, als wäre keine gültige Ersatzposition vorhanden. Zeilen 3 bis 5 in Quelltext 4.8 ermitteln für jede legale Ersatzposition die Bewertung nach der Quadrik-

Fehlermetrik. In Zeile 7 wird überprüft, ob die gefundene, minimale Bewertung unterhalb der Toleranzgrenze liegt und die Verschiebung gegebenenfalls blockiert.

```
1   float rating
2   Vertex replacement
3   foreach Neighbour n in Neighbours
4   {
5     CalculateRating(V, n, &rating, &replacement)
6   }
7   if(rating >= MAX_TOLERANCE)
8     replacement = NULL
9   else
10    Simplify(V, replacement)
```

Quelltext 4.8: Finden der Ersatzposition für V

Ist die Bewertung der möglichen Verschiebungen abgeschlossen, muss zwischen drei Fällen unterschieden werden:

- **1. Gültige Ersatzposition wurde gefunden** Der erste Fall stellt das Finden einer gültigen Ersatzposition dar. Eine Vereinfachungsoperation ist möglich und die Verschiebung wird durchgeführt.

- **2. Keine gültige Ersatzposition, keine zu verschiebenden Nachbarn** Im zweiten Fall kann keine Verschiebung durchgeführt werden. Durch das Fehlen von zu verschiebenden Nachbarn wird angenommen, dass die Topologie bei Erhalt der Netzintegrität keine Vereinfachungen zulässt. Hier erfolgt eine Reklassifizierung.

- **3. Keine gültige Ersatzposition, zu verschiebende Nachbarn vorhanden** Im letzten Fall wird von durch die Tests gesperrten Verschiebungen ausgegangen, die im nächsten Reklassifizierungsschritt behandelt werden müssen.

Tritt der erste Fall ein - es wird eine gültige Verschiebung ausgewählt - so müssen die Vertexdaten im Ausgabespeicher angepasst und alle relevanten Daten des Verschiebungskandidaten wie etwa Position, Normalvektor, etc. durch den Datensatz des Verschiebungszieles ersetzt werden. Dieser Vorgang findet für alle Verschiebungen statt.

Eine Anpassung der Hilfsdaten entfällt. Ihre Aktualisierung würde nicht nur die Daten der gespeicherten Nachbarvertizes verändern, sondern auch deren Anzahl. Diese Operation wäre ineffizient, da die verwendeten Datenstrukturen keine dynamische Anpassung der Größe anbieten, wodurch eine aufwändige Behandlung nötig wäre. Eine derartige Vorgehensweise würde durch die parallele Verschiebung benachbarter Vertizes zusätzliche Wettbewerbsbedingungen verursachen. Stattdessen kommen eine Erkennung von duplizierten Vertexpositionen in den Datensätzen sowie deren Behandlung bei

Zugriff auf einen Dreiecksfächer zum Einsatz.

Im Anschluss an die Aktualisierung der gespeicherten Vertexdaten muss die Liste der Verschiebungskandidaten aktualisiert werden. Durch die Vereinfachung werden die Nachbarvertizes des Verschiebungszieles verändert. Alle Vertizes, die eine Kante zum gelöschten Punkt V besaßen, sind nun mit der Ersatzposition V' verbunden. Ein Beispiel hierfür ist in Abbildung 4.12 dargestellt. Die linke Seite zeigt die Kanten vor der Verschiebungsoperation. Rechts wurde der mittlere Vertex entfernt und somit die Kanten und Nachbarn verändert.

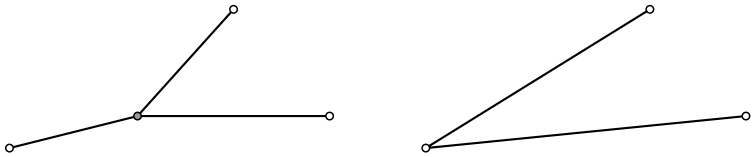

Abbildung 4.12: Veränderte Nachbarn nach Verschiebung

Unter den Nachbarn von V werden alle Punkte gesucht und als Verschiebungskandidaten selektiert, die:

- für ein Entfernen markiert wurden.
- noch keine Verschiebungskandidaten sind.

Gemeinsam mit der Erkennung der neuen Verschiebungskandidaten findet auch die Aktualisierung des gespeicherten Vertexfehlers der Nachbarn statt. Hierbei kommen die Daten des Verschiebungszieles bzw. der neuen Position des Vertex zum Einsatz. Quelltext 4.9 zeigt den Pseudocode, in dem alle Nachbarn betrachtet (Zeile 1), zu entfernende erkannt (Zeile 3) und die Vertexfehler aktualisiert werden (Zeile 4). Ist ein Vertex noch nicht in der Liste der Kandidaten enthalten (Bestimmung mittels Sprungzahl, Zeilen 5 und 6), so wird er in Zeile 8 in die Liste eingefügt.

```
1       foreach Vertex n in Neighbours
2       {
3         if(n.markedForRemoval == false) continue
4         float error = UpdateVertexError(replacement, n)
5         atomicMin(n.error, error)
6         int jumps = atomicMin(n.jumps, (replacement.jumps + 1))
7         if(jumps == 0)
8           candidateList.add(n.index)
9       }
```

Quelltext 4.9: Aktualisieren der Kandidatenliste

Falls keine gültige Ersatzposition existiert, wird - wie in Unterabschnitt 3.4.3 beschrieben - unterschieden, ob sich Verschiebungen gegenseitig sperren oder die Topologie keine Vereinfachungen zulässt.

Ist keine Verschiebung möglich und sind keine Nachbarvertizes vorhanden, die eine mögliche Verschiebung durch den zweiten oder dritten Verschiebungstest blockieren, wird eine Reklassifizierung des Vertex ausgelöst. Dies bedeutet, dass die Vertexdaten nicht manipuliert werden.

Da es zu einer Reklassifizierung des Punktes kommt, wird für alle zu entfernenden Nachbarn von V ein mögliches Verschiebungsziel geschaffen und eine Analyse aller Nachbarvertizes durchgeführt. Diese Analyse findet ähnlich zu dem in Quelltext 4.9 beschriebenen Verfahren statt. Im Gegensatz zu Quelltext 4.9 wurde hier jedoch keine Ersatzposition für V gefunden. Anstelle der Daten von V' wird hier auf jene von V bei der Bestimmung der Nachbarn zurückgegriffen. Ein weiterer Unterschied besteht in der Berechnung des aktualisierten Vertexfehlers, die hier übersprungen werden kann, da die Daten des Vertex nicht verändert wurden.

Eine andere Behandlung ist vonnöten, wenn:

- keine Verschiebung möglich ist.
- benachbarte Verschiebungskandidaten existieren.
- die Operation durch den zweiten oder dritten Test blockiert wird.

Sind alle diese drei Punkte erfüllt, findet keine Veränderung der Vertexdaten statt. Die Klassifizierung des Vertex wird beibehalten und keine Operation auf den Nachbarn von V durchgeführt. Stattdessen findet eine Kodierung mehrerer Werte in ein Datenfeld statt, um Informationen in den nächsten Iterationsschritt zu übertragen. Sie kommen zum Einsatz, um eine mögliche Blockade erkennen und gegebenenfalls behandeln zu können. Die gespeicherten Daten umfassen:

- **Mögliche Ersatzposition** Sie speichert, ob eine potentielle Ersatzposition gefunden wurde, die ausschließlich durch den zweiten oder dritten Verschiebungstest blockiert wird. Der einfache Verschiebungstest würde diese Position erlauben.

- **Anzahl benachbarter, zu verschiebender Vertizes** Diese Zahl muss bekannt sein, um festzustellen, ob benachbarte Punkte verschoben wurden oder ob eine Blockade eingetreten ist. Sind sie im nächsten Iterationsschritt identisch, wird von einer Blockade ausgegangen.

- **Anzahl benachbarter, beizubehaltender Vertizes** Sofern sich die Anzahl der benachbarten, beizubehaltenden Punkte zwischen den Iterationsschritten nicht ver-

ändert, liegen auch keine möglichen Verschiebungsziele vor. Tritt eine Änderung ein, kann eine erneute Verschiebung versucht werden.

Diese Daten werden gespeichert und dienen somit im nächsten Iterationsschritt als Eingabe.

Nach Durchführung dieser Operationen ist die Verschiebung des aktuellen Iterationsschrittes abgeschlossen und anhand der neu berechneten Daten und der aktualisierten Kandidatenliste zu überprüfen, ob eine Blockade vorliegt und ob manche der Verschiebungskandidaten reklassifiziert werden können.

4.7 Reklassifizierung

Die Reklassifizierung findet nach jeder Vertexverschiebung statt und erfüllt folgende Aufgaben:

- Berechnung der Reklassifizierung anhand des aktualisierten Vertexfehlers
- Aktualisierung der Kandidatenliste für die nächste Verschiebungsiteration auf Basis der Ergebnisse der Reklassifizierung
- Erkennung von gesperrten Verschiebungen und Auflösung von Blockaden

Die Reklassifizierung unterscheidet zwei Fälle:

- **1. Nachbar eines verschobenen Vertex** Ist ein Vertex Nachbar eines gelöschten Vertex, so ist durch das Entfernen des Nachbarn eine Veränderung der Topologie entstanden und die Klassifizierung ist neu zu berechnen.
- **2. Nachbar eines reklassifizierten Punktes** Wurde der Vertex nach einer Reklassifizierung als Verschiebungskandidat bestimmt, kann die Neuberechnung des Vertexfehlers - wie in Unterabschnitt 3.3.4 erläutert - übersprungen werden.

Tritt Fall 1 ein, wird der aktualisierte Vertexfehler analog zu der Vertexklassifizierung mit den Daten des Betrachters gewichtet und mit einer skalierten Toleranzgrenze verglichen (siehe Unterabschnitt 3.3.3). Das Ergebnis dieses Vergleiches entscheidet, ob ein Vertex reklassifiziert wird. Als zusätzliche Skalierung für die Toleranzgrenze kommt die in einem Vertex gespeicherte Sprungzahl zum Einsatz, um einen gleichmäßigeren Verlauf der Vereinfachung zu erzielen (siehe Unterabschnitt 3.3.4).

Wird ein Vertex an dieser Stelle reklassifiziert, wird sein Index aus der Liste der Verschiebungskandidaten entfernt. Anschließend sind alle Nachbarvertizes des soeben reklassifizierten Vertex zu betrachten. Nachdem der aktuelle Vertex beibehalten wird, sind alle noch zu löschenden Nachbarn potentielle Verschiebungskandidaten und können im nächsten Verschiebungsschritt behandelt werden. Sie werden daher in die Liste der Verschiebungskandidaten aufgenommen.

Das Finden von Verschiebungskandidaten unter den Nachbarvertizes verläuft analog zum Verfahren, das nach einer Reklassifizierung bei unmöglichen Verschiebungen zum Einsatz kommt. Eine neuerliche Berechnung des Vertexfehlers ist für diese Vertizes nicht nötig, da sie Nachbarn eines nicht manipulierten Vertex des originalen Netzes sind. Die initiale Klassifizierung hat somit für diese Vertizes Gültigkeit, und sie müssen verschoben werden.

Nach der Reklassifizierung anhand des Vertexfehlers werden in diesem Arbeitsschritt gesperrte Verschiebungen behandelt, sofern das Datenfeld eines Verschiebungskandidaten befüllt ist. Dies tritt ein, wenn der Punkt im letzten Iterationsschritt ein Verschiebungskandidat war, jedoch keine Ersatzposition gefunden wurde. Zusätzlich blockierte der zweite oder dritte Test die Verschiebung.

Die im Datenfeld gespeicherten Werte enthalten alle nötigen Informationen für die Behandlung. Sie werden mit dem gegenwärtigen Zustand der Nachbarvertizes verglichen. Hierzu wird auf die Daten in der Nachbarliste zurückgegriffen. Alle Nachbarn werden betrachtet und die Anzahl der zu verschiebenden Vertizes und möglichen Ersatzpositionen neu berechnet. Stimmen diese Daten mit den Werten aus dem Datenfeld überein, so konnte auch für die zu verschiebenden Nachbarn keine Ersatzposition gefunden werden bzw. diese waren keine Verschiebungskandidaten. In diesem Fall wird von einer gesperrten Verschiebung ausgegangen.

Die blockierenden Vertizes sind aus der Kandidatenliste zu entfernen - mit Ausnahme von nicht benachbarten Vertizes, die Ausgangspunkte für neuerliche Verschiebungen bilden. Die Bestimmung erfolgt anhand des gespeicherten Vertexfehlers und wurde in Unterabschnitt 3.3.4 beschrieben. Wird ein Vertex im Zuge der Behandlung von gesperrten Verschiebungen ignoriert und aus der Liste der Verschiebungskandidaten entfernt, so werden die Vertexdaten neu initialisiert und diese Vertizes so behandelt, als hätte mit ihnen nie der Versuch einer Vereinfachung stattgefunden.

Alle übrigen Vertizes bleiben ohne Veränderung in der Kandidatenliste erhalten und können im nächsten Iterationsschritt - sofern möglich - verschoben werden.

Verschiebung und Reklassifizierung werden iterativ so lange wiederholt, bis alle zu entfernenden Vertizes bearbeitet wurden. Diese Schritte ersetzen jedoch nur die Vertexdaten im Speicher. Sie verändern die Zahl der gespeicherten Dreiecke nicht. In einem finalen Berechnungsschritt werden auf Punkte oder Kanten reduzierte Dreiecke erkannt und aus dem Netz entfernt. Es verbleiben nur jene Dreiecke mit drei unterschiedlichen Eckpunkten.

4.8 Aufbau des vereinfachten Dreiecksnetzes

Nachdem alle Vereinfachungsoperationen beendet wurden und in der Kandidatenliste keine Vertizes mehr vorhanden sind, ist die Manipulation der Vertexdaten abgeschlossen. Das Problem liegt nun in der Darstellung des resultierenden Netzes.

Wird ein Vertex verschoben, so findet ein Überschreiben der Vertexdaten statt. Die Anzahl der Vertizes bzw. der Dreiecke des Netzes wird dabei nicht verändert. Würde eine direkte Darstellung dieses Dreiecksnetzes erfolgen, so wäre die Vereinfachung sichtbar, die Dreieckszahl jedoch identisch mit dem originalen Netz, da ein Teil der Dreiecke auf Punkte und Linien reduziert wurde, allerdings im Datensatz noch vorhanden ist.

Der Aufbau der finalen Daten erfolgt durch einen Vergleich der Eckpunkte der Dreiecke. Hierfür wird jedes Dreieck des originalen Netzes geladen und die Position von zwei Eckpunkten miteinander verglichen. Finden sich innerhalb eines Dreiecks zwei oder drei Vertizes mit identischen Positionen im Raum, so wird das Dreieck verworfen. Liegen alle drei Punkte auf unterschiedlichen Koordinaten, so hat das Dreieck Gültigkeit und ist im finalen Dreiecksnetz enthalten.

Nach Durchführung dieser Schritte ist die Vereinfachung abgeschlossen. Das Ergebnis der Operationen ist ein Satz aus Vertex- bzw. Indexdaten des vereinfachten Netzes. Diese Schritte haben keinen Einfluss auf die Funktionsfähigkeit eines nachfolgenden Darstellungsvorganges. Es ist daher sehr simpel, diesen Ansatz in ein bestehendes Verfahren für die Darstellung zu integrieren. Die Vertexdaten werden keiner Interpolation oder ähnlichen Operationen, die etwaige Vorberechnungen verfälschen könnten, unterzogen.

4.9 Zusammenfassung

Dieses Kapitel geht auf Details der Implementierung des in Kapitel 3 vorgestellten Verfahrens ein. Zuerst wird ein Überblick über die Teilschritte gegeben. Die Implementierung des Testsystems für eine Ausführung auf einer GPU führt zu Einschränkungen, bedingt durch das Vermeiden von Wettbewerbsbedingungen und das Schaffen von einheitlichen Datensätzen zwischen den Teilschritten des Verfahrens.

Für die Implementierung werden zwei zusätzliche Teilschritte eingeführt, die aufwändige Berechnungen zur Laufzeit vermeiden und somit die Leistung des Testsystems verbessern.

Nach der Auflistung der eingesetzten Speicherstrukturen werden Implementierungsdetails der einzelnen Teilschritte vorgestellt, wobei auf Schwierigkeiten und Einschränkungen eingegangen wird. Insbesondere die Abbildung der Teilschritte auf die Archi-

tektur der GPU und das damit zusammenhängende Vermeiden von Wettbewerbsbedingungen sind wichtige Punkte bei der Implementierung.

Diese Komponenten besitzen unterschiedliche Skalierbarkeit. Die Leistung der Vereinfachung ist in weiten Teilen abhängig von der Anzahl der durchzuführenden Iterationen. Auch die Anzahl der Vertizes des originalen Dreiecksnetzes ist nicht zu vernachlässigen, da sowohl die Klassifizierung als auch der Aufbau des finalen Netzes eine Iteration über alle diese Vertizes beinhalten. Diese Eigenschaften werden im nächsten Kapitel analysiert.

Kapitel 5

Leistungs- und Ergebnisanalyse

Dieses Kapitel beschäftigt sich mit der Analyse der Ergebnisse und Laufzeitmessungen der Vertexgruppeneliminierung. Abbildung 5.1 zeigt Beispiele von drei Modellen, die mit diesem Verfahren vereinfacht wurden.

Abbildung 5.1: Beispiele von vereinfachten Modellen

Die Vertexgruppeneliminierung wurde implementiert und die Ausgabe in ein einfaches Darstellungssystem integriert. Mit einer Reihe von ausgesuchten Szenen wurden verschiedene Tests basierend auf der Implementierung ausgeführt und anhand eines Testsystems die wichtigen Faktoren für die Skalierung des Verfahrens ermittelt. Einerseits wird die Leistung des Verfahrens bestimmt und andererseits die Qualität der Vereinfachung demonstriert.

In diesem Kapitel werden folgende Punkte präsentiert:

- **1. Allgemeine Analyse des Laufzeitverhaltens** Beschäftigt sich mit dem allgemeinen Laufzeitverhalten der Implementierung anhand des Umfangs der Eingabedaten.

- **2. Eingabedaten für die Laufzeitanalyse** Beschreibt die für die Laufzeitmessungen zum Einsatz kommenden Datensätze.

- **3. Testsystem** Listet die Eckdaten des verwendeten Testsystems auf.

- **4. Beschreibung der gemessenen Leistungsdaten** Beschreibt, welche Daten im Zuge der Laufzeitmessungen erfasst wurden.

- **5. Testfälle für die Leistungsmessungen** Gibt einen Überblick über die eingesetzten Testfälle.

- **6. Ergebnisse der Leistungsmessungen** Präsentation und Diskussion der Ergebnisse der Messungen, die mit Hilfe des implementierten Verfahrens durchgeführt wurden.

- **7. Visuelle Auswertung der Vereinfachungen** Präsentation und Diskussion der optischen Resultate der Vereinfachung.

- **8. Vergleich mit existierenden Algorithmen** Zieht einen Vergleich der Ergebnisse der Vertexgruppeneliminierung mit jenen anderer Algorithmen.

5.1 Allgemeine Analyse des Laufzeitverhaltens

Die Einzelschritte des Verfahrens sind eingeteilt in jene, die während der Vereinfachung nur ein Mal ausgeführt werden und solche, die iterativ wiederholt werden müssen, bis die entsprechenden Daten vollständig bearbeitet wurden. Demnach ist auch die Skalierung abhängig von mehreren Faktoren.

- **Einmalig ausgeführte Schritte** Die Klassifizierung und der damit verbundene Aufbau der initialen Kandidatenliste erfolgt während der gesamten Vereinfachung eines Dreiecksnetzes nur ein einziges Mal. Es findet eine Analyse per Vertex statt - also eine Iteration über jeden Vertex des Dreiecksnetzes. Die Laufzeit dieser Schritte ist daher abhängig vom Umfang des Datensatzes und skaliert mit der Anzahl der Vertizes, die zu betrachten sind.

- **Iterativ wiederholte Schritte** Die Schritte der Vertexverschiebung und Klassifizierung zeigen ein anderes Skalierungsverhalten. Hier sind zwei Faktoren ausschlaggebend: Die Anzahl der Vertizes, die in einem Iterationsschritt bearbeitet werden müssen und die Anzahl der Iterationsschritte. Diese Faktoren sind weitgehend abhängig von den Ergebnissen der Klassifizierung und der Topologie des Dreiecksnetzes. Eine hohe Zahl zu entfernender Vertizes ist keine Garantie, dass ein Großteil dieser Vertizes in einem Iterationsschritt behandelt werden kann. Der Aufbau der Dreiecke und die Kanten zwischen den Vertizes spielen hierbei ebenfalls eine Rolle. Ein einzelner beizubehaltender Vertex mit vielen zu entfernenden Nachbarn wirkt sich positiv auf die Parallelität aus. Im Gegensatz dazu trägt eine Gruppe von beizubehaltenden Vertizes mit nur wenigen zu löschenden Nachbarn geringer zur Parallelität bei.

Die direkte Abhängigkeit der Laufzeit von der Zahl der Vertizes betrifft lediglich den initialen Arbeitsschritt der Vertexklassifizierung. Dieser führt zudem nur Berechnungen von geringer Komplexität aus, da die aufwändige Berechnung der Fehlermetrik in einen Vorverarbeitungsschritt ausgelagert wurde. Somit ist zu erwarten, dass der Einfluss auf die Skalierung des gesamten Verfahrens geringer ausfällt als die Anzahl der Iterationen.

Die Menge der in einem Iterationsschritt zu bearbeitenden Vertizes benötigt während des Verschiebungsschrittes einen größeren Leistungsaufwand, da neben den arithmetischen Operationen auch eine Reihe von Speicherzugriffen erforderlich sind, um die Daten der Nachbarvertizes zu bestimmen. Dies kann - in Abhängigkeit von der Topologie des Netzes - einen großen Einfluss auf die Leistung des gesamten Verfahrens haben.

Die Zahl der Iterationsschritte ist eine weitere wichtige Komponente für die Skalierung. Nach jeder Iteration findet ein Kopieren der Daten zwischen Ein- und Ausgabespeicher statt. Die benötigte Zeit für diese Arbeit ist abhängig von der Größe der Speicher und somit von der Anzahl der Vertizes im Dreiecksnetz. Sie stellt eine konstante Arbeitslast dar.

Die Verteilung aller zu löschender Vertizes auf eine hohe Anzahl von Iterationsschritten kann dazu führen, dass die Menge der in einem Schritt zu löschenden Vertizes nicht groß genug ist, um alle Prozessoren der GPU zu nutzen. Es kommt zur Einschränkung der Parallelität und folglich zur Reduktion der Effizienz.

Um die Leistung des Verfahrens in einer Reihe von Tests zu bestimmen, müssen einheitliche Testfälle geschaffen werden. Hierfür wird ein Dreiecksnetz spezifiziert und die Parameter für die Vereinfachung so gewählt, dass ein aussagekräftiger Vergleich gezogen werden kann.

5.2 Eingabedaten für die Laufzeitanalyse

Als Eingabedaten wird ein 3D-Modell benötigt, das vereinfacht werden soll. Um repräsentative Resultate zu erzielen, kommen hierbei mehrere Szenarien zum Einsatz. Verschiedene Modelle werden durch das Verfahren vereinfacht, die sich durch die Anzahl an Vertizes bzw. Dreiecken unterscheiden. Die Anzahl an Dreiecken in den Datensätzen soll dabei verbreitet zum Einsatz kommende Modelle repräsentieren. Als weitere Besonderheit sind Löcher in der Oberfläche zu nennen. Nicht alle Modelle besitzen eine vollständig geschlossene Oberfläche, um die Funktionsfähigkeit der Vertexgruppeneliminierung bei derartigen Modellen zu zeigen.

Für die Daten, die für Testläufe zur Anwendung kommen, wurde das Stanford 3D Scanning Repository zu Hilfe genommen, das eine Reihe von 3D-Modellen zur Verfügung stellt. Die Modelle des Stanford 3D Scanning Repositories zeichnen sich durch eine unterschiedliche Anzahl an Dreiecken aus und nicht alle Modelle weisen eine vollständig geschlossene Oberfläche auf. Sie werden zudem in einer Vielzahl von wissenschaftlichen Beiträgen für Testfälle eingesetzt, was einen Vergleich mit anderen Verfahren und Algorithmen erleichtert.

Das erste für die Testfälle eingesetzte Modell ist der Stanford Bunny. Dieses Objekt wurde aus mehreren 3D-Scans mit insgesamt 362 272 Datenpunkten rekonstruiert. Die finale, hochauflösende Rekonstruktion besteht aus 35 947 Vertizes, die 69 451 Dreiecke bilden. Abbildung 5.2 zeigt den Stanford Bunny. Links ist die originale Tonfigur zu sehen, rechts eine Darstellung der digitalen Version.

Abbildung 5.2: Stanford Bunny

Der Stanford Bunny weist ein großes Spektrum an Oberflächeneigenschaften auf. Es existieren signifikante, glatte Flächen und große, unruhige Bereiche, sowie markante Ecken und Kanten.

Neben den detaillierten Analysen mit Hilfe des Stanford Bunny wurde auch eine star-

ke Vereinfachung auf Modellen mit mehr Dreiecken ausgeführt, um das Verhalten der Laufzeiten zu bestimmen. Auch für diese Modelle wurde auf das Stanford 3D Scanning Repository zurückgegriffen.
Die zusätzlichen Modelle sind Armadillo (172 974 Vertizes, 345 944 Dreiecke), Dragon (437 645 Vertizes, 871 414 Dreiecke) und Happy Buddha (543 652 Vertizes, 1 087 716 Dreiecke). Sie sind in Abbildung 5.3 dargestellt. Bei diesen Tests wurde nur die Laufzeit der Vereinfachung bestimmt.

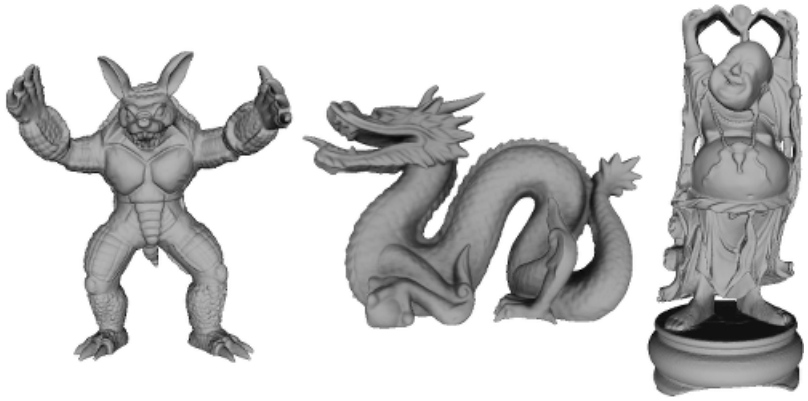

Abbildung 5.3: Weitere Modelle: Armadillo (links), Dragon (Mitte) und Happy Buddha (rechts)

Tabelle 5.1 gibt einen direkten Vergleich der Größe der Datensätze der für die Laufzeitanalyse eingesetzten Modelle.

Modell	Stanford Bunny	Armadillo	Dragon	Happy Buddha
Vertizes	35 947	172 974	437 645	543 652
Dreiecke	69 451	345 944	871 414	1 087 716

Tabelle 5.1: Vergleich der Testmodelle

5.3 Testsystem

Auf einem Testsystem wurde eine Reihe von Testfällen definiert, anhand derer mehrere Laufzeitmessungen ausgeführt wurden. Das Testsystem wurde gewählt, um die Laufzeit und Eigenschaften des Verfahrens auf zum Zeitpunkt der Ausführung der Testfälle weit verbreiteten Komponenten zu bestimmen.

- Prozessor: Intel Core i7 (4 x 3.4 Ghz)
- Graphikkarte: Geforce GTX 670
  - 1344 Kerne
  - 1059 Mhz Kerntakt
  - 2048 MB Graphikspeicher
  - 3004 Mhz Speichertakt
  - 256 bit Speicherinterface

Die Eckdaten der Hardware machen ersichtlich, dass die Vertexgruppeneliminierung von einer Implementierung auf einer GPU mit einer hohen Anzahl an Rechenkernen profitieren kann. Dies führt jedoch unter Umständen dazu, dass in manchen Iterationsschritten nicht für alle Prozessoren Verschiebungskandidaten bestimmt werden und somit ein Teil der GPU nicht benötigt wird. Dieser Leerlauf stellt verschwendete Rechenzeit dar und sollte vermieden werden.

Für die Implementierung der Software wurde das CUDA Toolkit Version 5.0 unter Windows 7 verwendet und die Software mit Hilfe von Visual Studio 2008 geschrieben.

Es ist anzumerken, dass für die Entwicklung in CUDA die Driver API zum Einsatz kam, somit nicht der gesamte Quellcode, sondern nur die CUDA Kernels durch den Nvidia Compiler nvcc kompiliert wurden. Für die restlichen Teile kam der Visual Studio Compiler zur Anwendung. Die Ausgabe erfolgte mittels DirectX 11.

Im Anschluss wird beschrieben, welche Leistungsdaten gemessen wurden.

5.4 Beschreibung der gemessenen Leistungsdaten

Bei den Messungen zur Analyse der Ergebnisse des Verfahrens wurden folgende Werte ermittelt:

- **Zeitaufwand für die Vereinfachung** Gibt die benötigte Zeit für die Vereinfachung eines Testfalles an. Dieser Parameter ist ein guter Indikator für die Gesamtleistung des Verfahrens.

- **Laufzeit der Teilkomponenten** Die absolute und relative Laufzeit der einzelnen Teilschritte der Vertexgruppeneliminierung. Durch diese Werte lassen sich Rückschlüsse auf Engpässe bei der Berechnung und die Auswirkung der Hardware ziehen.

- **Anzahl der markierten Vertizes** Die Zahl der Vertizes des Dreiecksnetzes, die für ein Entfernen markiert wurde. Diese hat starken Einfluss auf die benötigte Laufzeit.

- **Anzahl der Iterationsschritte** Die Zahl der Iterationsschritte, die durchgeführt wurde, bis die Abbruchbedingung erreicht war. Diese Zahl wird erheblich durch gesperrte Verschiebungen und Reklassifizierungen beeinflusst.

- **Finale Dreieckszahl** Die Anzahl der Dreiecke, aus denen das vereinfachte Objekt aufgebaut ist. Sie wird nicht vorherbestimmt, sondern ergibt sich durch die einzelnen Arbeitsschritte der Vertexgruppeneliminierung.

- **Zahl reklassifizierter Vertizes** Die Anzahl an Vertizes, die während der Iterationsschritte reklassifiziert wurde.

Das Ziel dieser Daten ist einerseits, die Leistungsmerkmale der Vertexgruppeneliminierung darzustellen, andererseits die entscheidenden Faktoren und möglichen Engstellen zu ermitteln. Durch den Vergleich verschiedener Testfälle lassen sich Schlüsse auf die Leistung der Teilkomponenten ziehen und die Auswirkungen von verschiedenen Parametern auf die Vereinfachung bestimmen.

Im nächsten Abschnitt werden die für die Messungen eingesetzten Testfälle verglichen.

5.5 Testfälle für die Leistungsmessungen

Für die Leistungsmessung kommen die Modelle Stanford Bunny, Armadillo, Dragon und Happy Buddha zum Einsatz. Auf allen vier Modellen wurde eine starke Vereinfachung ausgeführt, um den Einfluss der Komplexität des Datensatzes auf das Verfahren zu bestimmen.

Mit Hilfe des Modells Stanford Bunny wurden zusätzliche Testfälle definiert, die sich im Grad der Vereinfachung voneinander unterscheiden. Um vergleichbare Resultate für die Leistungsmessungen zu erhalten, wurde eine Kameraposition fixiert. Dies garantiert, dass keine Störung der ermittelten Daten durch einen variablen Blickvektor auftritt und die Resultate der Testfälle nur von den kontrollierten Parametern abhängen.

Die entscheidenden Faktoren für Klassifizierung und Reklassifizierung werden manuell gesteuert und nicht mittels der Position der Kamera errechnet. Somit kommt bei allen Testfällen eine vergleichbare Ausführung der Vertexgruppeneliminierung zur Anwendung und Ungenauigkeiten durch eine Benutzersteuerung der Position und Blickrichtung des Betrachters sind ausgeschlossen. Es ist anzumerken, dass für Toleranzgrenzen und Metrikmanipulation in dieser Arbeit keine direkten Werte genannt werden, da diese vom gewählten Objekt sowie von der Skalierung des Dreiecksnetzes abhängig und somit nicht repräsentativ sind.

Neben den generischen Testfällen mit allen vier Modellen wurden für die Detailanalyse 5 Testfälle mit dem Stanford Bunny definiert, auf die näher eingegangen wird. Die Grenzwerte für Klassifizierung und Reklassifizierung wurden so gewählt, dass die Testfälle mehrere Stufen zwischen einer geringen und einer starken Vereinfachung simulieren. So können die Auswirkungen des Ausmaßes der Vereinfachung auf die Laufzeit bestimmt, beschränkende Faktoren und Nachteile erkannt und die Stärken des Verfahrens aufgezeigt werden.

Als zusätzlicher Test wird die Vereinfachung einer Version des Stanford Bunny mit weniger Dreiecken (16 301 Dreiecke/8 171 Vertizes) errechnet, um die Vereinfachung eines Netzes mit geringerer Vertexzahl zu beobachten und die Auswirkungen auf den Ablauf der Vertexgruppeneliminierung zu analysieren. Der Einsatz des Modells des Stanford Bunny für diesen Test ermöglicht einen direkten Vergleich der gemessenen Daten mit den 5 Testfällen auf der hochauflösenden Version des Stanford Bunny.

Die ersten fünf Fälle simulieren die Berechnung der Vereinfachung mit zunehmender Distanz zwischen Objekt und Betrachter. Der erste geht von einem Objekt nahe an der Position des Betrachters aus. Diese Distanz wird erhöht. Beim letzten wird ein großer Abstand gewählt und eine wesentlich stärkere Vereinfachung durchgeführt.

Das Verhalten kann über nur zwei Parameter gesteuert werden: Die beiden Toleranzgrenzen für Klassifizierung und Reklassifizierung. Diese Werte werden unabhängig voneinander verändert. Für beide gilt jedoch, dass eine höhere Toleranz das Entfernen zusätzlicher Vertizes bewirkt und die finale Vertexzahl reduziert.

Abbildung 5.4 zeigt, aus wie vielen Dreiecken das vereinfachte Objekt in jedem Testfall besteht. Wie hier zu sehen ist, verläuft die Abnahme der Dreieckszahl annähernd linear. Ausgehend von den 69 451 Dreiecken des originalen Netzes werden in diesen Testfällen zwischen 20 620 und 62 392 Dreiecke gelöscht.

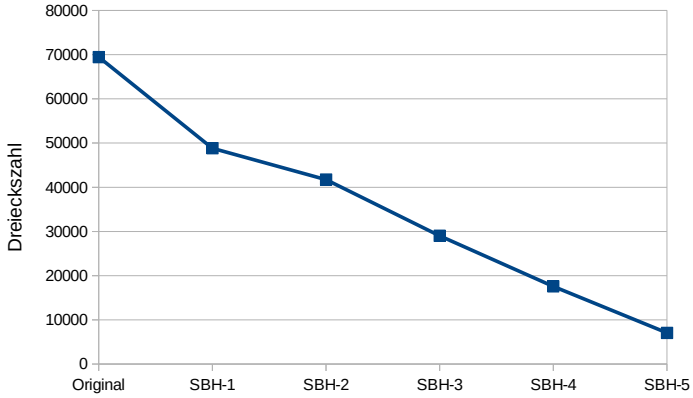

Abbildung 5.4: Dreieckszahl der vereinfachten Netze (ausgehend von hochauflösendem Stanford Bunny)

Mit zunehmendem Grad der Vereinfachung ist die Ausführung zusätzlicher Vereinfachungsoperationen nötig, und die Anzahl an verbleibenden Dreiecken sinkt. Überdies verringert sich die Zahl der beibehaltenen Vertizes, und weniger Vertizes werden reklassifiziert. Somit nimmt der Zeitaufwand für die Bearbeitung des Netzes bis zur finalen Version zu.

Der nächste Abschnitt präsentiert einen kurzen Überblick über die einzelnen auf dem hochauflösenden Modell ausgeführten Testfälle. Anschließend werden die Ergebnisse der mit diesen Fällen ausgeführten Messungen vorgestellt, die Daten der einzelnen Testfälle miteinander verglichen und daraus Schlüsse über die Leistung und das Verhalten der Vertexgruppeneliminierung gezogen. Für jeden der Testfälle (SBH-1 bis SBH-5) auf dem hochauflösenden Netz des Stanford Bunny wurden die Dreiecke im vereinfachten Netz, die Zahl der entfernten Dreiecke, die nötigen Iterationen und die Laufzeit der Vereinfachung bestimmt. Diese Daten sind für die 5 Testfälle in Tabelle 5.2 aufgelistet.

Testfall	SBH-1	SBH-2	SBH-3	SBH-4	SBH-5
Dreiecke im vereinfachten Netz	48 831	41 732	29 014	17 565	7 059
Entfernte Dreiecke	20 620	27 719	40 437	51 886	62 392
Iterationen	2	3	5	8	12
Laufzeit der Vereinfachung	1,94 ms	2,13 ms	3,36 ms	4,43 ms	5,76 ms

Tabelle 5.2: Ermittelte Werte für 5 Testfälle auf hochauflösendem Stanford Bunny

5.5.1 Testfall SBH-1

Dieser Testfall dient dazu, das Laufzeitverhalten der Vertexgruppeneliminierung bei einem niedrigen Vereinfachungsgrad zu bestimmen. Die geringen Toleranzgrenzen für Klassifizierung und Reklassifizierung bewirken eine große Anzahl an beizubehaltenden Vertizes. Dementsprechend stehen viele Verschiebungsziele für zu löschende Vertizes zur Verfügung und eine hohe Parallelität kann erreicht werden.

Der Testfall SBH-1 (Ergebnis in Abbildung 5.5) simuliert die Vereinfachung eines Objekts, das sich in geringer Distanz zum Betrachter befindet. Daher werden nur relativ wenige Vertizes aus dem Netz entfernt.

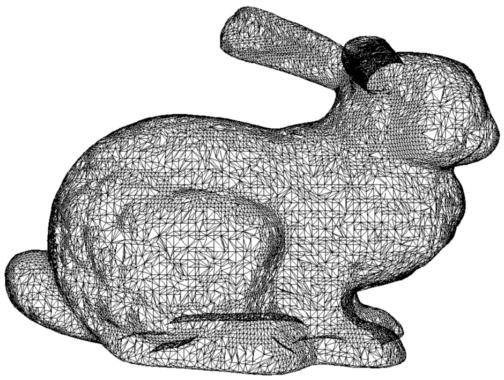

Abbildung 5.5: Ergebnis Testfall SBH-1

Abbildung 5.5 zeigt das Ergebnis der Vereinfachung (siehe auch Tabelle 5.2). Obwohl die Dreieckszahl bereits deutlich reduziert wurde, ist das Netz immer noch detailliert dargestellt. Die geringe Distanz zur Kamera resultiert in einer großen Anzahl an beibehaltenen Punkten bei der Klassifizierung. Unabhängig von der Topologie des Dreiecksnetzes können viele der zu löschenden Vertizes in einem Iterationsschritt bearbeitet werden.

Somit benötigt die gesamte Vereinfachung in diesem Testfall nur 2 Iterationsschritte. Dieser erste Testfall führt eine Reduktion der Dreieckszahl gegenüber dem originalen Netz um fast 30% durch. Begünstigt durch die geringe Anzahl an erforderlichen Iterationen werden hierfür nur knapp 2 ms benötigt.

5.5.2 Testfall SBH-2

Der zweite Testfall (Ergebnis in Abbildung 5.6) findet mit einem größeren Abstand zwischen Kamera und Betrachter statt. Die höhere Toleranz bei der Klassifizierung hat den

Effekt, dass eine andere Stufe des regelmäßigen Gitters ausgewählt wird. Dementsprechend stehen weniger Verschiebungsziele in gleichmäßigem Abstand zur Verfügung.

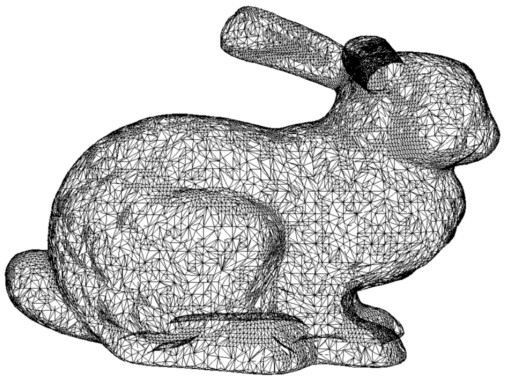

Abbildung 5.6: Ergebnis Testfall SBH-2

Abbildung 5.6 zeigt das resultierende Netz für die gewählten Parameter (siehe auch Tabelle 5.2). Die im Vergleich zu Testfall SBH-1 reduzierte Dreieckszahl ist deutlich erkennbar. Trotz der verminderten Auflösung sind Unterschiede in der Silhouette bzw. an Details auf dem Objekt nicht offensichtlich zu sehen.

Ungeachtet des Wechsels in eine Stufe des regelmäßigen Gitters mit weniger Punkten ist im Vergleich zu SBH-1 nur eine zusätzliche Iteration für das Entfernen aller Vertizes nötig. Dies hat zur Folge, dass für etwa 7 000 zusätzlich gelöschte Dreiecke - was einer um etwa 35% stärkeren Reduktion der Dreieckszahl entspricht - 0,2 ms mehr Laufzeit in Anspruch genommen wird.

5.5.3 Testfall SBH-3

Auch hier wird die Distanz zur Kamera erhöht. Im Gegensatz zum Sprung zwischen Testfall SBH-1 und SBH-2 ist jedoch die gewählte Stufe des regelmäßigen Gitters identisch wie bei SBH-2. Die Unterschiede im resultierenden Netz werden somit einerseits durch weniger beibehaltene, andererseits durch weniger reklassifizierte Vertizes verursacht. Abbildung 5.7 zeigt das Gitternetz des resultierenden Netzes.

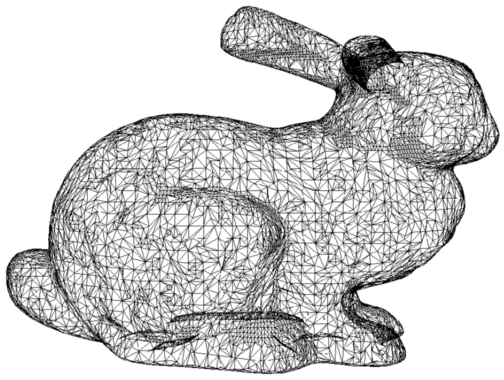

Abbildung 5.7: Ergebnis Testfall SBH-3

Abbildung 5.7 macht dieses Verhalten ersichtlich (siehe auch Tabelle 5.2). Vor allem in detailarmen Gebieten des Netzes ist der Unterschied zu erkennen. Hier wurden mehr Dreiecke entfernt. In manchen Bereichen zeichnet sich bereits eine rechteckige Vernetzung ab - hervorgerufen durch die Punkte des regelmäßigen Gitters.

Trotz der identischen Stufe des Gitters sind im Vergleich zum vorherigen Testfall (SBH-2) zusätzliche Iterationen erforderlich. Eine Reklassifizierung hat zur Folge, dass in einem Iterationsschritt ein zu löschender Vertex reklassifiziert und ihm benachbarte Verschiebungskandidaten entfernt werden. Die Reduktion der Reklassifizierungen kann somit als Faktor für die zusätzlichen Iterationen und die höhere Laufzeit genannt werden.

5.5.4 Testfall SBH-4

Dieser Testfall (Ergebnis in Abbildung 5.8) greift erneut auf eine andere Stufe des regelmäßigen Gitters als SBH-2 und SBH-3 zu. Daraus resultiert mitunter eine Reduktion der beibehaltenen Vertizes. Gemeinsam mit den durch die größere Distanz zum Betrachter gestiegenen Toleranzen wird eine deutliche Vereinfachung des Modells durchgeführt.

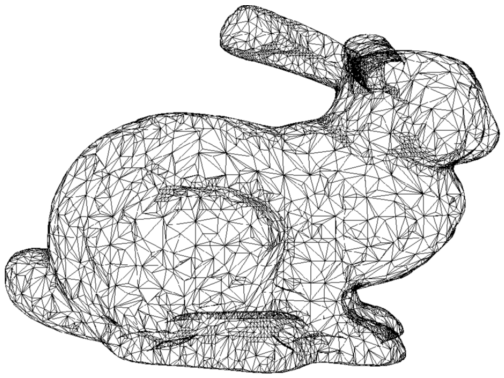

Abbildung 5.8: Ergebnis Testfall SBH-4

Abbildung 5.8 macht den Unterschied zu den vorherigen Testfällen deutlich (siehe auch Tabelle 5.2). Hier ist eine starke Verminderung der Dreieckszahl erkennbar. Trotz dieser Reduktion ist die Silhouette des Modells noch gut ausgebildet. Auch Ausformungen wie etwa der Oberschenkel werden sichtbar detailreicher wiedergegeben, wodurch die visuelle Qualität des resultierenden Netzes gut erhalten bleibt.

Die Vertexgruppeneliminierung geht von einem hochauflösenden Dreiecksnetz aus. Mit jedem Iterationsschritt werden Vertizes entfernt (engl. „top-down“). Der Ansatz zeigt hier in Form einer längeren Laufzeit Auswirkungen. Bei diesem Testfall werden fast 75% der Dreiecke aus dem Netz entfernt.

Die geringe Auflösungsstufe des regelmäßigen Gitters gemeinsam mit der gestiegenen Toleranzgrenze für die Klassifizierung hat die Konsequenz, dass die Zahl der beibehaltenen Vertizes - und somit der möglichen Verschiebungsziele - abnimmt. Zusammen mit der sinkenden Menge an Reklassifizierungen wirkt sich dies negativ auf die Anzahl der benötigten Iterationen aus. Bei diesem Fall sind 8 Schritte vonnöten, was für eine Laufzeit von 4,4 ms verantwortlich ist.

5.5.5 Testfall SBH-5

Dies ist der letzte Testfall auf dem hochauflösenden Modell. Dabei wird die stärkste Vereinfachung der hier vorgestellten Testfälle durchgeführt (Abbildung 5.9). Es kommt dieselbe Stufe des regelmäßigen Gitters wie in Testfall SBH-4 zum Einsatz. Dabei sind jedoch die Toleranzgrenzen für Klassifizierung und Reklassifizierung erneut gestiegen. Dies resultiert in einem Entfernen der meisten Vertizes des Modells.

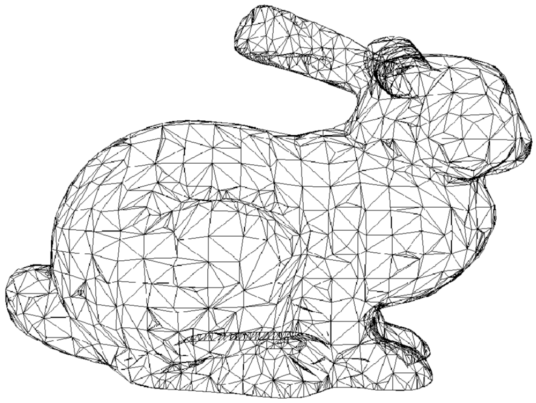

Abbildung 5.9: Ergebnis Testfall SBH-5

Abbildung 5.9 zeigt das Ergebnis dieses Testfalles (siehe auch Tabelle 5.2). Das Netz wurde weitgehend auf die Punkte des regelmäßigen Gitters reduziert. Nur in manchen Bereichen mit starken Ausformungen (Ohren, Oberschenkel) sind noch zusätzliche Dreiecke vorhanden, die eine detailreichere Darstellung ermöglichen.

Auch hier ist die Silhouette des Objekts trotz dieser starken Vereinfachung noch gut erhalten. Es sind leichte Qualitätsverluste und Kantenbildungen ersichtlich, diese liegen jedoch im marginalen Bereich. Details auf der Oberfläche (Augen, unruhiges Fell) sind allerdings bei dieser stärkeren Vereinfachung entfernt worden.

Erwartungsgemäß ist für diese Vereinfachung auch die größte Laufzeit erforderlich. Beim vorliegenden Testfall wird die Dreieckszahl des Objekts auf nur etwa 7 000 verringert. Dies entspricht einer Reduktion um fast 90%.

Die starke Vereinfachung bedarf jedoch auch einer hohen Iterationszahl. Verursacht wird dies durch die Stufe des regelmäßigen Gitters sowie die geringe Zahl an beizubehaltenden Vertizes, gemeinsam mit einer hohen Toleranzgrenze bei der Reklassifizierung. So werden insgesamt 12 Iterationsschritte benötigt, wodurch sich die im Vergleich zu den anderen Testfällen hohe Laufzeit von 5,7 ms erklärt.

Im nächsten Abschnitt werden die Ergebnisse der Leistungsmessungen präsentiert.

5.6 Ergebnisse der Leistungsmessungen

Dieser Abschnitt präsentiert die Ergebnisse der mit den Testfällen durchgeführten Messungen. Neben den bereits vorgestellten Resultaten werden die gemessenen Daten der Testfälle SBH-1 bis SBH-5 gegenübergestellt. Es werden die Aufgliederung der Laufzeit in die Teilkomponenten der Vertexgruppeneliminierung und die Analyse dieser Teillaufzeiten präsentiert. Des Weiteren wird auf die Skalierung der einzelnen Aspekte

eingegangen. Schließlich werden die Stärken und Probleme sowie etwaige Engpässe bei der Ausführung aufgezeigt.

Es werden folgende Punkte behandelt:

- **1. Laufzeiten** Präsentiert das Laufzeitverhalten der Vertexgruppeneliminierung bei den Testfällen SBH-1 bis SBH-5.

- **2. Vereinfachung Stanford Bunny mit geringer Dreieckszahl** Analyse der Vereinfachung einer Version des Stanford Bunny mit geringer Dreieckszahl.

- **3. Weitere Modelle** Präsentation der Laufzeit der Vereinfachung für die Modelle Armadillo, Dragon und Happy Buddha.

Der erste Punkt - einer der wichtigsten für die Resultate der Vertexgruppeneliminierung - ist die gemessene Laufzeit, die von der Vereinfachung in Anspruch genommen wird.

5.6.1 Laufzeiten

Die Laufzeit der Vertexgruppeneliminierung kann einerseits als Ganzes betrachtet werden - also jene Zeit, die zwischen Beginn der Klassifizierung und Abschluss der Vereinfachung liegt - oder genauer anhand der Teilschritte. Für die Messergebnisse bedeutet dies eine Unterteilung in:

- **Klassifizierung** Klassifizierung aller Vertizes sowie Bestimmung der initialen Verschiebungskandidaten

- **Reklassifizierung** Aufgewandte Zeit für einen bzw. alle Reklassifizierungsschritte

- **Verschiebungsoperationen** Zeit, erforderlich für das Entfernen der Vertizes - insgesamt oder per Iteration

Ein weiterer wichtiger Faktor ist die Anzahl der Iterationsschritte. Je stärker ein Objekt vereinfacht werden soll, desto mehr Iterationsschritte werden unter Umständen benötigt. Dieses Verhalten ist jedoch nicht exakt vorhersagbar, da der Faktor stark von der Topologie des Netzes sowie der Lage der beizubehaltenden Vertizes in der Topologie abhängig ist.

Der erste gemessene Wert beinhaltet die Zeit, die für die Vereinfachung benötigt wurde. Dabei sollen die Laufzeiten der einzelnen Testfälle direkt miteinander verglichen

werden. Die für die Darstellung des Objekts erforderliche Laufzeit ist in dieser Messung nicht berücksichtigt.

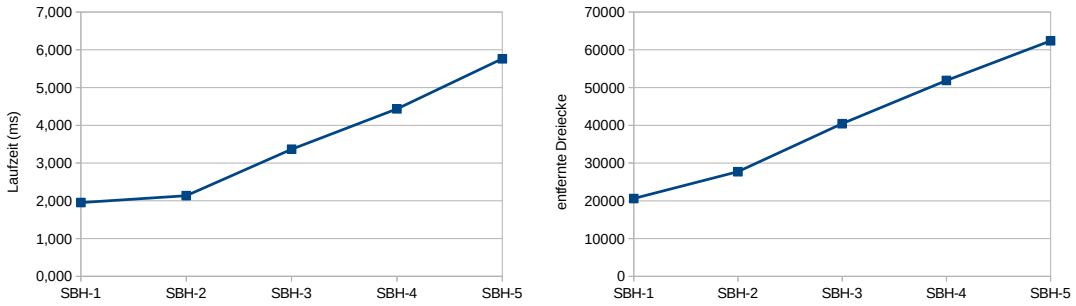

Abbildung 5.10: Gesamtlaufzeit der Vereinfachung (links) und Zahl der entfernten Dreiecke (rechts)

Abbildung 5.10 zeigt auf der linken Seite die ermittelten Laufzeiten der einzelnen Testfälle in Millisekunden. Das Diagramm rechts zeigt den Verlauf der entfernten Vertizes in den einzelnen Testfällen. Hier ergeben sich vor allem zwei Auffälligkeiten: Zum einen das Verhalten von Testfall SBH-2, zum anderen der Gesamtanstieg der Laufzeit im Vergleich zur Zahl entfernter Vertizes.

Betrachtet man den Anstieg der gelöschten Dreiecke, so verläuft dieser zwischen den Testfällen annähernd linear. Die einzelnen, eruierten Laufzeiten ergeben ein ähnliches Bild. Während Testfall SBH-2 bei den entfernten Vertizes und der Gesamtlaufzeit leicht von einem linearen Verlauf abweicht, ist der geringe Anstieg der Laufzeit im Vergleich zu SBH-1 deutlich. Dies ist vor allem bemerkenswert, da zwischen Testfall SBH-1 und SBH-2 auch ein Wechsel der Stufe des regelmäßigen Gitters stattfindet, was weniger beizubehaltende Vertizes als mögliche Verschiebungsziele nach sich zieht: Bei SBH-1 bleiben 24 760 Vertizes erhalten, bei SBH-2 19 366 und bei SBH-3 13 408.

Testfall SBH-1 führt 2 Iterationen durch, SBH-2 benötigt nur eine zusätzliche. Im Vergleich zu den darauffolgenden Testfällen ist dies ein geringerer Anstieg, was sich auf die Laufzeit auswirken kann. Verursacht wird die geringe Iterationszahl durch die kleinen Toleranzen bei Klassifizierung und Reklassifizierung. Dies hat zur Folge, dass einerseits eine große Anzahl an beibehaltenen Vertizes erkannt wird. Andererseits können relativ viele Reklassifizierungen vorgenommen werden, was die Iterationszahl niedrig hält.

Der Anstieg der Laufzeiten im Vergleich zu den entfernten Vertizes verläuft gleichmäßig. Dies ist der Fall, obwohl die Zahl der benötigten Iterationsschritte stark ansteigt (SBH-1: 2, SBH-5: 12). Somit nehmen die Iterationen in den Testfällen mit stärkerer Vereinfachung durchschnittlich weniger Zeit in Anspruch. Verursacht wird dieses Verhalten ebenfalls durch die beizubehaltenden Vertizes. Weniger mögliche Verschiebungsziele resultieren in weniger Verschiebungskandidaten. Die zu löschenden Vertizes

werden also auf eine höhere Anzahl an Iterationsschritten verteilt, und die Laufzeit für einen einzelnen Verschiebungsschritt sinkt.

In weiterer Folge werden die Laufzeiten der Teilaspekte der Vertexgruppeneliminierung näher betrachtet. Im ersten Schritt bedeutet dies die Messung der Laufzeit der Klassifizierung. Er enthält sowohl die Berechnungen der eigentlichen Klassifizierungsoperationen, als auch den Aufbau der initialen Kandidatenliste.

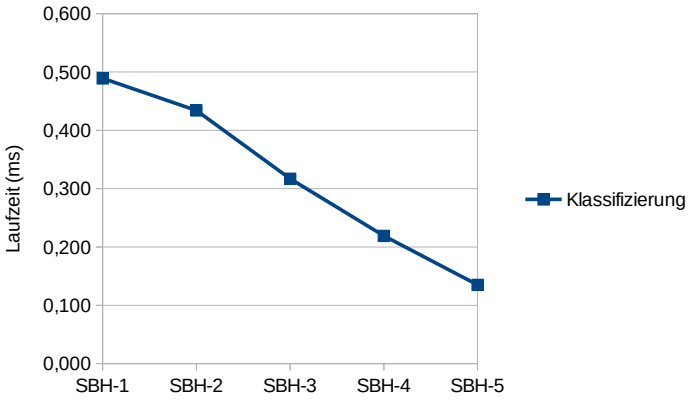

Abbildung 5.11: Laufzeit der Klassifizierung

Betrachtet man das in Kapitel 3 beschriebene Verfahren, wäre in diesem Fall eine konstante Laufzeit zu erwarten. Dies begründet sich darin, dass die Klassifizierung alle Vertizes eines Dreiecksnetzes analysiert und als beizubehaltend oder zu löschend einstuft.

Die Erklärung für das in Abbildung 5.11 gezeigte Laufzeitverhalten liegt in der Tatsache, dass der Aufbau der initialen Kandidatenliste in den Messungen inkludiert ist. Wie in Kapitel 4 beschrieben, wird in der Implementierung, die in dieser Arbeit für das Testsystem geschaffen wurde, der Aufbau der initialen Kandidatenliste im Anschluss an die Klassifizierung vorgenommen. Da diese beiden ersten Schritte der zur Laufzeit ausgeführten Berechnungen immer gemeinsam stattfinden, sind ihre Laufzeiten hier zusammengefasst.

Für das Auffinden der Kandidaten wird von den beizubehaltenden Vertizes ausgegangen. Die Klassifizierung bestimmt alle verbleibenden Vertizes. Nach Abschluss dieser Operation für sämtliche Vertizes des Dreiecksnetzes werden in einem zusätzlichen Schritt alle Nachbarn beizubehaltender Vertizes betrachtet. Wird dabei ein zu entfernender Vertex vorgefunden, gilt er als Verschiebungskandidat. Durch dieses Vorgehen erklärt sich auch das Verhalten der Laufzeit aus Abbildung 5.11.

In den ersten Testfällen werden viele Vertizes als beizubehaltend markiert. Da zum Zeitpunkt der eigentlichen Klassifizierung keine Information über die Nachbarn vorliegt, müssen alle diese Vertizes kontrolliert werden. Für das Auffinden der initialen

Kandidaten ist daher eine große Anzahl an Vertizes zu betrachten. Bei den späteren Testfällen vergrößern sich die Toleranzen für das Löschen von Vertizes. Weniger Vertizes sind beizubehalten und die Zahl der Vertizes, deren zu entfernende Nachbarn bestimmt werden, ist geringer. Daraus ergibt sich, dass die benötigte Laufzeit für die Berechnung dieses Arbeitsschrittes in den Testfällen mit stärkerer Vereinfachung abnimmt.

Der nächste wichtige Punkt ist die Laufzeit der Verschiebungsoperationen. In diesem Fall wird die gesamte Laufzeit betrachtet - also aufsummiert über alle Iterationsschritte.

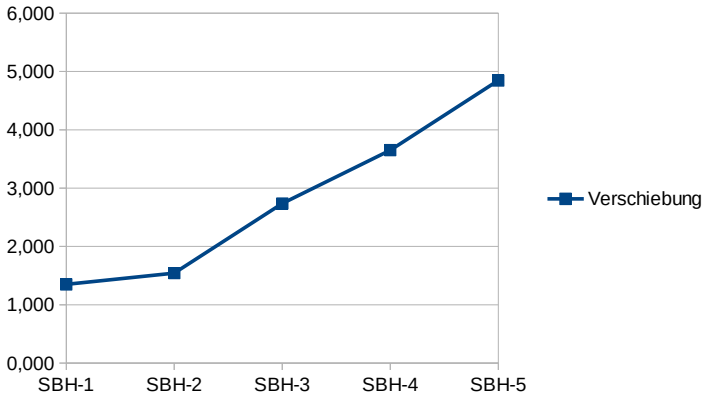

Abbildung 5.12: Laufzeit der Verschiebungsschritte

Abbildung 5.12 zeigt die gemessenen Werte für die Gesamtlaufzeit der Vertexverschiebungen der einzelnen Testfälle. In dieser Darstellung ist - ebenso wie in Abbildung 5.10 - die Abweichung von Testfall SBH-2 vom annähernd linearen Anstieg der Laufzeiten ersichtlich. Dieser tritt hier auf, da die summierte Laufzeit über alle Iterationsschritte gezeigt ist. Somit hat die Anzahl der durchgeführten Iterationen einen Einfluss auf dieses Ergebnis.

Ein weiterer Punkt ist der Anteil der Verschiebungen an der Gesamtlaufzeit.

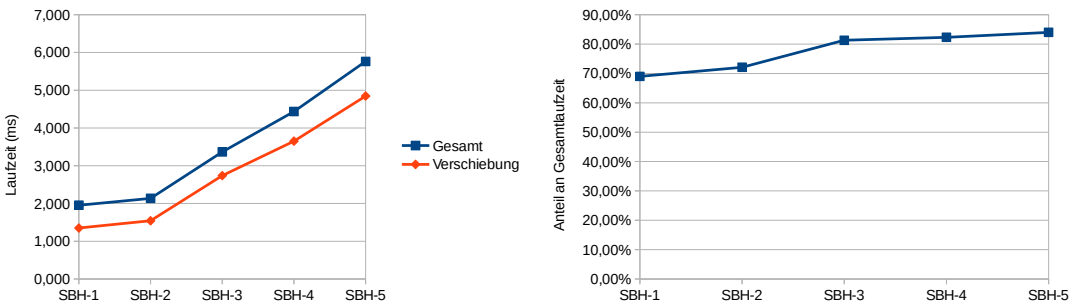

Abbildung 5.13: Anteil der Verschiebung an der Gesamtlaufzeit

Abbildung 5.13 zeigt auf der linken Seite den Vergleich der gesamten Laufzeit mit der kumulativen Laufzeit der Verschiebung. Es ist erkennbar, dass diese Operationen für den Hauptanteil der gesamten Laufzeit verantwortlich sind.

Rechts ist der prozentuelle Anteil direkt abgebildet. Selbst beim ersten Testfall mit nur zwei Iterationen nehmen die Verschiebungsoperationen bereits über 70% der Gesamtlaufzeit ein. Dieses Verhalten verdeutlicht auch, welcher Teilschritt einen effektiven Ansatzpunkt für Verbesserungen und Optimierungen bietet, um eine starke Auswirkung auf die Gesamtlaufzeit zu erzielen.

Das linke Diagramm in Abbildung 5.13 - es vergleicht die Gesamtlaufzeit mit der Laufzeit der Verschiebungen - veranschaulicht, dass der neben den Verschiebungen verbleibende Anteil der Laufzeit nur geringe Schwankungen zwischen den einzelnen Testfällen aufweist.

Wie bereits oben gezeigt wurde, ist die benötigte Zeit für den vollständigen Klassifizierungsschritt von Testfall SBH-1 zu SBH-5 leicht abfallend. Dies bedeutet, dass für die Reklassifizierungsoperationen ein entsprechender Anstieg der benötigten Laufzeiten zu erwarten ist.

Bisher wurde für die Laufzeit der Verschiebungen die gesamte Dauer der Operationen, also eine Aufsummierung der Zeiten der Vereinfachungen aller Iterationsschritte, betrachtet und analysiert. In weiterer Folge wird die Aufmerksamkeit auf die genauere Analyse des Verhaltens dieses Arbeitsschrittes während der einzelnen Iterationen gerichtet.

Als erste nähere Analyse findet die Betrachtung der durchschnittlichen Laufzeit der Vereinfachung statt.

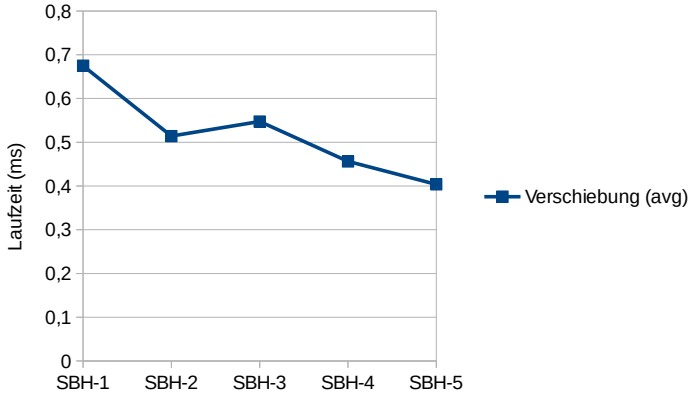

Abbildung 5.14: Durchschnittliche Laufzeit eines Verschiebungsschrittes

Abbildung 5.14 zeigt die durchschnittlichen Laufzeiten der Verschiebungsiterationen für die fünf Testfälle. Die Abbildung illustriert, dass ein genereller Abwärtstrend der durchschnittlichen Laufzeit für die Verschiebung vorliegt: Je stärker die durchgeführte

Vereinfachung und je mehr Iterationen somit vorhanden sind, desto geringer fällt die durchschnittliche Laufzeit aus. Dieses Verhalten kann über die Anzahl der zu bearbeitenden Vertizes erklärt werden. Bei den ersten Testfällen finden nur wenige Iterationen statt. Aufgrund der hohen Anzahl der beizubehaltenden Vertizes wird hier in jedem Iterationsschritt auch eine große Menge an Verschiebungskandidaten gefunden, die bearbeitet werden muss. Die Laufzeit einer einzelnen Verschiebung steigt durch die Mehrzahl an benötigten Operationen.

Im Gegenzug fehlen in den späteren Testfällen mit stärkeren Vereinfachungen die erforderlichen Verschiebungsziele. Hier können in einem einzelnen Iterationsschritt weniger Vertizes entfernt werden. Dies resultiert in einer geringeren Menge an Verschiebungen per Iteration, die benötigte Zahl an Iterationen erhöht sich jedoch.

Die einzige Ausnahme dieses Trends, der in Abbildung 5.14 aufscheint, stellt Testfall SBH-3 dar. Hier ist die durchschnittliche Laufzeit der Verschiebung länger als bei SBH-2. Dieses Phänomen wird durch zwei Faktoren hervorgerufen: Einerseits dem starken Abfall der durchschnittlichen Laufzeit zwischen SBH-1 und SBH-2 und andererseits einem nicht vorhersagbaren Verhalten aufgrund der Topologie des Dreiecksnetzes.

Das Diagramm in Abbildung 5.14 zeigt, dass Testfall SBH-2 im Vergleich zu SBH-1 den stärksten Abfall in der durchschnittlichen Laufzeit aufweist. Die Abweichung vom Gesamtverlauf wird somit nicht nur durch SBH-3 verursacht, sondern scheint wegen der geringen Dauer in Testfall SBH-2 und der etwas höheren in SBH-3 auf. Beide Fälle können mit der Topologie des Dreiecksnetzes und der daraus resultierenden Anzahl an Verschiebungskandidaten per Iterationsschritt erklärt werden.

Die Laufzeit für eine einzelne Vereinfachungsoperation setzt sich zusammen aus einem konstanten (Synchronisierung der Datenspeicher, Initialisierung der Operation) und einem dynamischen Anteil, der von der Anzahl der zu behandelnden Daten abhängig ist. Stehen in manchen Iterationen nur sehr wenige Verschiebungskandidaten zur Verfügung, so trägt der konstante Anteil gleichbleibend zur Laufzeit bei.

Im ersten Iterationsschritt von Testfall SBH-3 wird eine zu den anderen Tests vergleichsweise große Menge an Verschiebungskandidaten gefunden. Gegenläufig dazu ergeben sich durch die Topologie im letzten Schritt nur mehr sehr wenige Kandidaten. Dies führt zu einer langen Laufzeit der ersten Verschiebungsiteration, die durch den letzten Bearbeitungsvorgang nicht ausgeglichen werden kann und somit die etwas höhere Gesamtlaufzeit nach sich zieht.

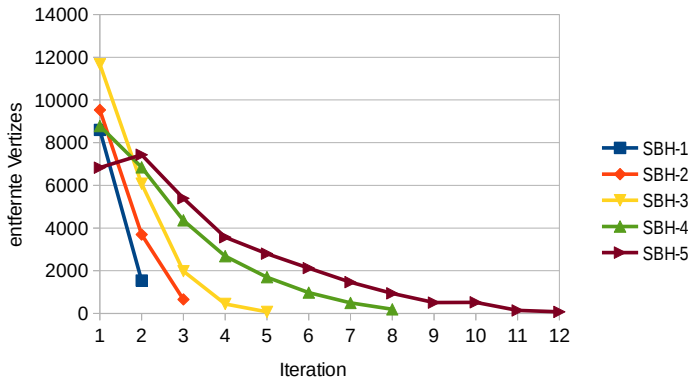

Abbildung 5.15: Anzahl entfernter Vertizes in den Iterationsschritten

Abbildung 5.15 zeigt dieses Verhalten. Hier sind die per Iterationsschritt bearbeiteten Vertizes für alle fünf Testfälle dargestellt. Bei Testfall SBH-3 ist die hohe Anzahl an Kandidaten im ersten Iterationsschritt auffällig. Im Gegensatz hierzu sind in Schritt 5 nur mehr sehr wenige Vertizes verfügbar.

Bei SBH-4 und SBH-5 sinkt die Anzahl der verfügbaren Verschiebungskandidaten drastisch ab. Besonders die letzte Iteration von SBH-5 ist in dieser Hinsicht hervorzuheben. Es sind nur mehr 74 zu löschende Vertizes vorhanden, die noch abgearbeitet werden müssen.

Dieses Verhalten verdeutlicht ein Problem der Vertexgruppeneliminierung: Die verfügbare Anzahl an Kandidaten ist stark abhängig von der Topologie des Netzes, der Lage der beizubehaltenden Vertizes und somit den Ergebnissen der Klassifizierung. Es wird keine minimale Anzahl an Verschiebungskandidaten, die in einem Schritt bearbeitet werden sollen, garantiert. Die Parallelität kann stark eingeschränkt werden. Bei einer ungünstigen Topologie ist eine Vereinfachung nur in vielen Iterationsschritten mit einer jeweils geringen Zahl an Verschiebungskandidaten möglich. Das Ergebnis wäre eine schlechte Leistung und unter Umständen eine nicht vorteilhafte Vereinfachung.

Eine geringe Anzahl an Verschiebungskandidaten kann eine effiziente Ausnutzung der verfügbaren Ressourcen verhindern. Sind in der eingesetzten GPU mehr Kerne als Verschiebungskandidaten vorhanden, bleiben Teile der Rechenleistung ungenützt.

Bei der Betrachtung der späteren Arbeitsschritte mancher Testfälle in Abbildung 5.15 wird dieses Problem verdeutlicht. Sie enthalten weniger zu verschiebende Vertizes als Prozessoren auf der GPU (im Falle des in Abschnitt 5.3 vorgestellten Testsystems 1344 Kerne) zur Verfügung stehen. Die parallele Abarbeitung ist zwar noch ausführbar, das Potential der Architektur wird jedoch nicht mehr zur Gänze genutzt, wodurch sich Leerlaufzeiten der Hardware ergeben.

Abbildung 5.15 zeigt auf, dass dieses Problem in den ersten Testfällen weniger deutlich auftritt. Je mehr Vertizes jedoch aus dem Netz entfernt werden sollen, desto gravierender wirkt sich diese Eigenschaft auf die Vereinfachung aus.

Somit kann die Anzahl der Iterationsschritte, die für die gesamte Vereinfachung durchgeführt werden muss, als wichtiger Faktor für die Leistung der Vertexgruppeneliminierung und die Nutzung der bereitstehenden Hardware gesehen werden. Je weniger Iterationsschritte ausgeführt werden, desto effizienter verläuft die Vereinfachung.

Das regelmäßige Gitter (Unterabschnitt 3.3.2) hat zum Ziel, die Iterationszahl zu reduzieren. Es schafft eine Menge von beizubehaltenden Vertizes, die über das gesamte Dreiecksnetz verteilt sind und nicht ausschließlich von den Ausprägungen des Objekts abhängen. Große Bereiche ohne mögliche Verschiebungsziele, die für eine hohe Iterationszahl verantwortlich sein können, werden somit weitgehend vermieden.

Dies wirkt sich besonders bei Testfall SBH-5 aus. Hier wird ein Großteil der Vertizes des Netzes für ein Entfernen markiert. Ein Testlauf mit den Parametern von SBH-5 ergab, dass die Anzahl der benötigten Iterationen durch den Einsatz des regelmäßigen Gitters um etwa den Faktor 4 reduziert wurde.

Der letzte Teil der Vertexgruppeneliminierung, der zur Laufzeit beiträgt, ist die Reklassifzierung.

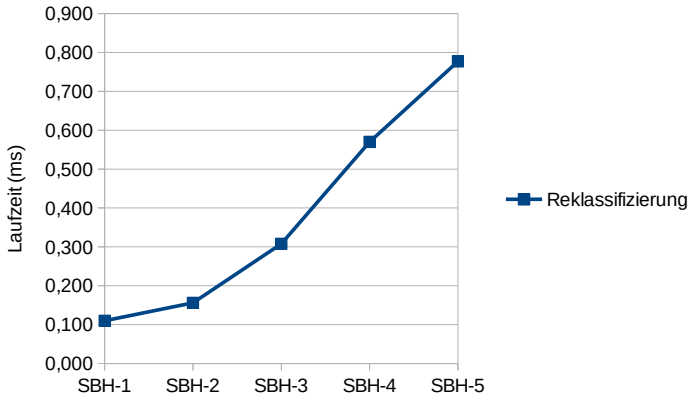

Abbildung 5.16: Laufzeit der Reklassifizierung

Abbildung 5.16 zeigt die aufsummierte Laufzeit aller Reklassifizierungen pro Testfall und den Anstieg der hierfür benötigten Laufzeit. Es ist anzumerken, dass in diesem Schritt auch die Behandlung von gesperrten Verschiebungen durchgeführt wird und diese somit in den gezeigten Laufzeitmessungen enthalten ist.

Trotz des Anstiegs der benötigten Zeit ist der Anteil der Reklassifizierung an der Gesamtlaufzeit der Vereinfachung relativ gering. In Testfall SBH-5 nimmt sie etwa 13% der Gesamtlaufzeit ein. Bemerkenswert ist jedoch der starke Anstieg der benötigten Zeit für die Durchführung aller Reklassifizierungen.

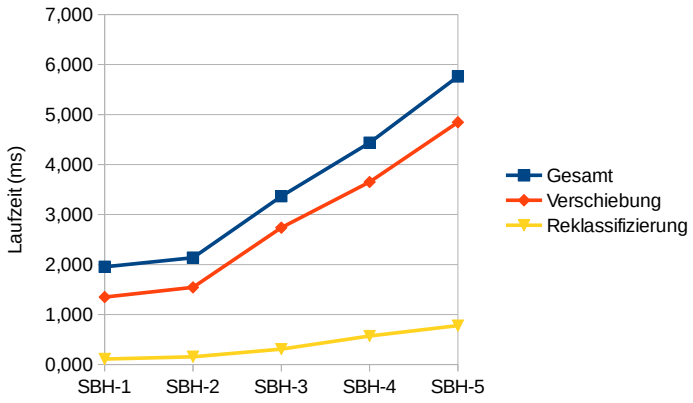

Abbildung 5.17: Vergleich der Laufzeiten

Abbildung 5.17 zeigt die kumulative Laufzeit der Reklassifizierung im Vergleich zur Gesamtlaufzeit und der kumulativen Dauer der Verschiebungen.

Während der für die Reklassifizierung erforderliche Zeitaufwand mit dem Grad der Vereinfachung ansteigt, zeigt Abbildung 5.17, dass diese Zunahme geringer ausfällt als dies bei der Verschiebung der Fall ist.

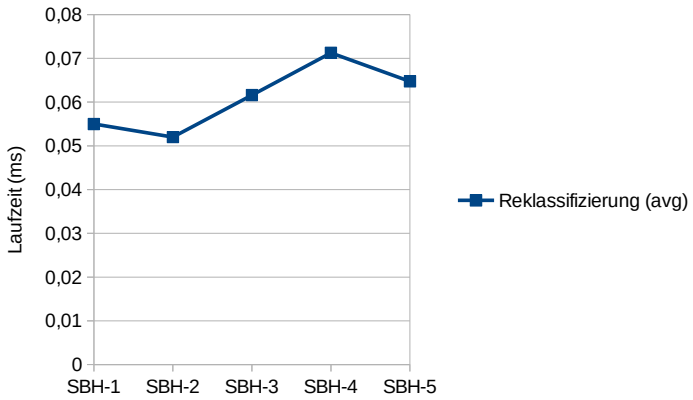

Abbildung 5.18: Durchschnittliche Laufzeit der Reklassifizierung

Abbildung 5.18 zeigt die durchschnittliche Laufzeit eines Reklassifizierungsschrittes. Ihr Verlauf ist relativ konstant. Die Schwankungsbreite dieser Messung bewegt sich über alle Testfälle in einem Bereich von 0,02 ms. Dies bedingt, dass schon geringe Ungenauigkeiten bei den Messungen und Schwankungen in den Bearbeitungszeiten ausreichen, um die Daten zu verfälschen. Es ist somit davon auszugehen, dass die durchschnittliche Dauer der Reklassifizierung eines Iterationsschrittes konstant ist.

Neben der Laufzeit der Reklassifizierung werden an dieser Stelle auch die Auswirkungen bzw. Resultate dieses Schrittes betrachtet. Ziel der Reklassifizierung ist, als zu

löschend markierte Vertizes beizubehalten, wenn diese einen zu großen Fehler verursachen würden. Wird ein Vertex vom Reklassifizierungsschritt weiterhin als zu löschend eingestuft, so bildet dieser einen Verschiebungskandidaten für den nächsten Iterationsschritt. Wird er jedoch aufgrund der veränderten Topologie beibehalten, so ist die Bearbeitung dieses Vertex abgeschlossen und alle seine zu löschenden Nachbarn werden zu Verschiebungskandidaten. Reklassifizierungen können sich auf die Zahl der benötigten Iterationsschritte auswirken, da innerhalb einer Iteration zusätzliche, zu entfernende Vertizes abgearbeitet werden.

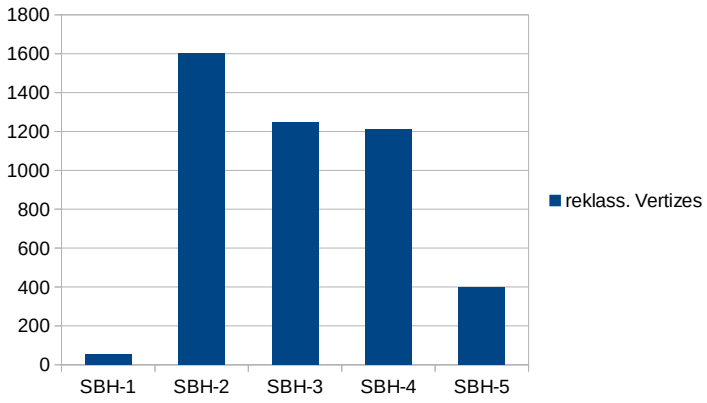

Abbildung 5.19: Anzahl der reklassifizierten Vertizes

Abbildung 5.19 zeigt die Anzahl an Vertizes, die während der gesamten Vereinfachung reklassifiziert wurden. Die stark schwankenden Ergebnisse zwischen den einzelnen Testfällen sind auffällig. Auch hier existiert eine Abhängigkeit von der Topologie und den gewählten Verschiebungen. Damit ist dieses Verhalten nicht exakt vorhersagbar.

Bei Testfall SBH-1 ist nur eine kleine Anzahl an Vertizes reklassifiziert worden. Die geringe Toleranz bei der Klassifizierung hat die Folge, dass wenige Vertizes gelöscht werden und keine starken Abweichungen von den original ermittelten Vertexfehlern auftreten. Zusätzlich wird bei diesem Testfall - in Abbildung 5.15 erkennbar - ein Großteil der zu löschenden Vertizes schon im ersten Verschiebungsschritt behandelt. Die Zahl der potentiell zu reklassifizierenden Vertizes ist demnach geringer, und in beiden durchgeführten Iterationsschritten dieses Testfalles finden wenige Reklassifizierungen statt.

Der zweite auffällige Testfall ist SBH-5. Auch hier ist eine geringe Menge an Reklassifizierungen vorhanden. Das Verhalten begründet sich jedoch anders. Die Ursache liegt in der hohen Toleranzgrenze für die Reklassifizierung. Das Ergebnis dieses Testfalles (Abbildung 5.9) zeigt, dass das Dreiecksnetz weitgehend auf die Punkte des regelmäßigen Gitters reduziert wird. Lediglich in Bereichen mit starker Krümmung finden sich zusätzliche Vertizes. Aufgrund der hohen Distanz zwischen Betrachter und Objekt ist der tolerierte Fehler bei Klassifizierung und Reklassifizierung so groß, dass dieser von

den Vertizes bei der gewählten Stufe des regelmäßigen Gitters kaum erreicht wird. Als Folge kommt es zu einer geringen Zahl an Reklassifizierungen.

Die verbleibenden drei Testfälle zeigen ein einheitliches, leicht abfallendes Verhalten bei der Anzahl der reklassifizierten Vertizes, da die Toleranzgrenze für die Reklassifizierung mit der zunehmenden Distanz zwischen Kamera und Betrachter - wie in den Tests simuliert - ansteigt.

Diese Leistungsmessungen ergeben, dass der Hauptanteil der Laufzeit von den iterativen Schritten der Vereinfachung verursacht wird. Die Klassifizierung hat aus oben genannten implementierungstechnischen Gründen eine leicht abfallende Laufzeit. Besonders der Anstieg der Iterationszahl bei den Testfällen mit stärkerer Vereinfachung beeinflusst die Leistung und Effizienz des Verfahrens. Daher wird hier diese Zunahme näher betrachtet.

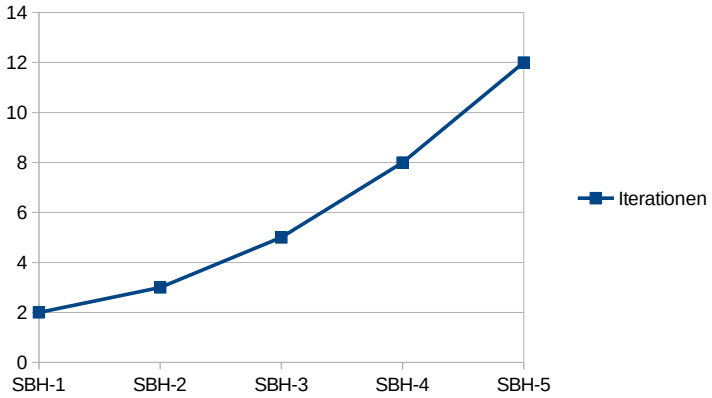

Abbildung 5.20: Benötigte Iterationsschritte

Abbildung 5.20 zeigt den Anstieg der Zahl der notwendigen Iterationen in den einzelnen Testfällen. Bei stärkeren Vereinfachungen kommt eine starke Erhöhung der Zahl der Iterationsschritte zustande. Wie oben gezeigt, ist der Anstieg der erforderlichen Berechnungszeit trotz des gezeigten Verlaufs annähernd linear (Abbildung 5.10).

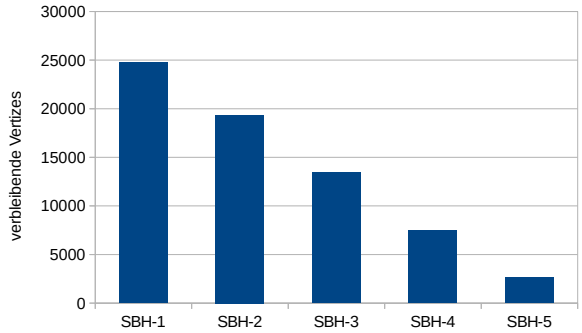

Abbildung 5.21: Beizubehaltende Vertizes (initiale Klassifizierung)

Abbildung 5.21 zeigt, wie viele Vertizes in den einzelnen Testfällen durch die initiale Klassifizierung als beizubehaltend ausgewählt wurden. Dies umfasst sowohl Vertizes, die durch das regelmäßige Gitter bestimmt, als auch jene, die mit dem korrekten Vertexfehler berechnet wurden. Hier fällt im Vergleich zu den benötigten Iterationsschritten auf, dass ein annähernd linearer Verlauf der Anzahl von Vertizes auftritt. Das nicht lineare Verhalten der Zahl an Iterationsschritten erklärt sich also nicht durch diese initiale Menge an Verschiebungszielen.

Besonders anzumerken ist an dieser Stelle der Bedarf zusätzlicher Iterationen zwischen den Testfällen SBH-2 und SBH-3 bzw. SBH-4 und SBH-5 in Abbildung 5.20. Diese greifen auf dieselbe Stufe des regelmäßigen Gitters zurück und haben somit eine gemeinsame Grundlage an beizubehaltenden Punkten. Trotz dieses Verhaltens ist ein deutlicher Anstieg der Iterationszahl zwischen den einzelnen Testfällen zu erkennen. Ein Faktor hierfür ist die geringere Anzahl an beizubehaltenden Vertizes, die ohne das regelmäßige Gitter bestimmt werden. Durch sie entfallen manche Verschiebungsziele, was die höhere Iterationszahl nach sich zieht.

Wie in Abbildung 5.19 ersichtlich ist, tritt zwischen den genannten Testfällen ein Abfall der reklassifizierten Vertizes auf - verursacht durch die größeren Toleranzen bei der Reklassifizierung. Diese Eigenschaft reduziert die möglichen Verschiebungsziele, die zwischen Iterationsschritten geschaffen werden, woraus sich die Zunahme der Iterationsschritte begründet.

Im nächsten Abschnitt sollen die Ergebnisse der Testfälle, die auf dem hochauflösenden Modell des Stanford Bunny ausgeführt wurden, mit der Vereinfachung einer Version desselben Objekts mit weniger Vertizes verglichen werden.

5.6.2 Vereinfachung Stanford Bunny mit geringer Dreieckszahl

Diese Version des Stanford Bunny ist aus wesentlich weniger Dreiecken aufgebaut als jene, die in den bereits vorgestellten Testfällen zum Einsatz kommt. Mit diesem Modell soll das Verhalten der Vereinfachung bei einer geringeren Problemgröße (Modell mit weniger Vertizes, Testfall SBN) ermittelt werden. Die Ergebnisse werden mit den gemessenen Daten aus den Testfällen, die auf dem hochauflösenden Modell ausgeführt wurden (SBH-1 bis SBH-5), verglichen.

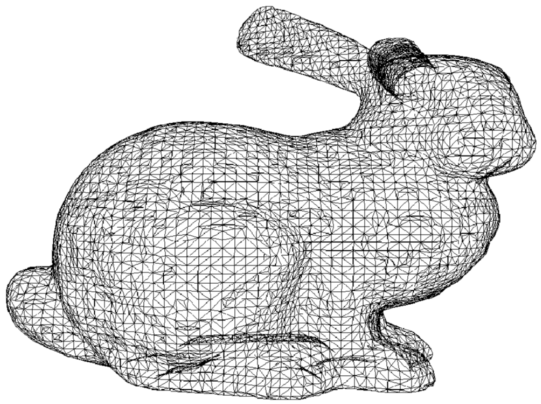

Abbildung 5.22: Modell mit geringerer Auflösung

Während die hochauflösende Version aus 69 451 Dreiecken besteht, sind hier nur mehr 16 301 Dreiecke vorhanden, die aus 8 171 Vertizes aufgebaut werden. Abbildung 5.22 zeigt das Dreiecksnetz dieses Modells. Es verdeutlicht, dass schon im Vergleich zum vereinfachten Dreiecksnetz von Testfall SBH-1 (Abbildung 5.5) eine geringere Zahl an Dreiecken vorhanden ist. Dies stellt eine drastische Reduktion der Vertexzahl dar, von der die Vereinfachung ausgeht.

Die Stufen des 3-dimensionalen Gitters werden in diesem Fall unter anderen Aspekten gewählt. Eine sehr hohe Auflösung - wie sie etwa bei Testfall SBH-1 des hochauflösenden Netzes zum Einsatz kommt - ist nicht sinnvoll, da ein Großteil der existierenden Vertizes für ein Beibehalten markiert würde.

Bei einem Modell mit weniger Vertizes sind meist auch weniger Vertizes zu entfernen, wodurch die Bearbeitungszeit für die gesamte Vereinfachung absinkt. Da weniger Vertizes gleichzeitig gelöscht werden können, kommt es unter Umständen zu einer Beschränkung der Parallelität, was - wie bereits oben beschrieben - zu Problemen bei der Auslastung der verfügbaren Hardware führen kann.

Bei der Vereinfachung des Stanford Bunny mit geringer Vertexzahl wurde ein Großteil der Dreiecke entfernt. Die ermittelten Werte für diesen Test sind in Tabelle 5.3 aufgelistet.

Dreiecke im vereinfachten Netz	5 900
Entfernte Dreiecke	10 401
Iterationen	4
Laufzeit der Vereinfachung	1,578 ms

Tabelle 5.3: Ermittelte Werte für Testfall SBN

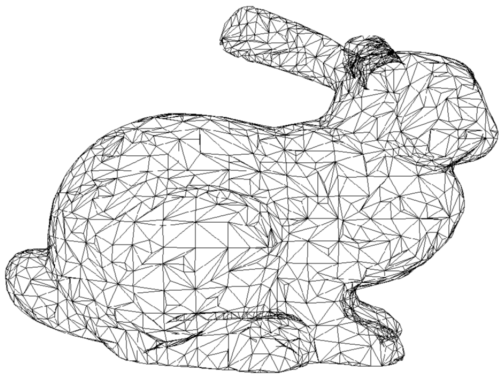

Abbildung 5.23: Vereinfachtes Modell

Abbildung 5.23 zeigt das resultierende Dreiecksnetz, das im Verlauf dieser Vereinfachung erzeugt wurde. Bei der Vereinfachung in Testfall SBN wurden etwa 64% der Dreiecke aus dem Netz entfernt. Dies entspricht etwa dem Grad der Vereinfachung, der bei Testfall SBH-3 zum Einsatz kommt. Es ist auch die Gegenüberstellung mit SBH-1 sinnvoll, um die Zahl an entfernten Dreiecken in Bezug auf die Laufzeit zu vergleichen.

Mit 10 401 entfernten Dreiecken wurden hier nur etwa halb so viele Dreiecke aus dem Netz gelöscht wie bei SBH-1, bei dem die Berechnung in knapp 1,9 ms abgeschlossen wurde. Allerdings waren bei SBH-1 lediglich 2 Iterationsschritte nötig - im Gegensatz zu den 4 bei Testfall SBN.

Durch die niedrigere Anzahl an Vertizes, aus denen das Dreiecksnetz aufgebaut ist, werden bei der Klassifizierung auch weniger zu löschende Vertizes selektiert. Bei den Verschiebungsschritten ist eine kleinere Menge an Verschiebungskandidaten zu bearbeiten. Somit nimmt die durchschnittliche Dauer der Verschiebungen ab, woraus sich trotz zusätzlicher Iterationsschritte die kürzere Laufzeit erklärt.

In Abbildung 5.24 werden die Laufzeiten der Teilschritte des Verfahrens betrachtet.

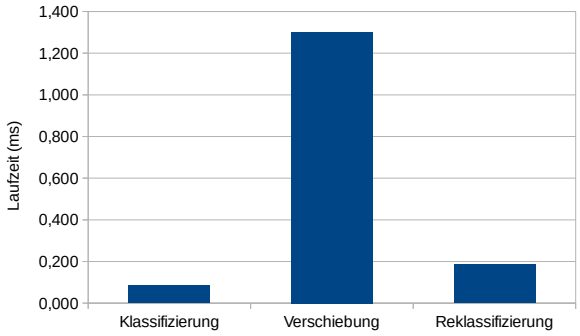

Abbildung 5.24: Laufzeiten der Teilschritte

Hier zeigt sich ein vergleichbares Bild zu den Tests mit dem hochauflösenden Modell. Die Verschiebungen benötigen einen Großteil der Laufzeit. Im Gegensatz dazu nimmt die Klassifizierung mit 0,086 ms nur sehr wenig Zeit in Anspruch. Dieses Verhalten ist durch die niedrige Anzahl an zu bearbeitenden Vertizes erklärbar. Gleichzeitig sinkt die absolute Zahl an beizubehaltenden Vertizes, deren Nachbarn betrachtet werden.

Auch die Reklassifizierung belegt nur einen relativ kleinen Teil der Laufzeit, das sind in diesem Fall weniger als 0,2 ms. Durch die geringere Auflösung des Netzes - und daraus resultierend die kleinere Anzahl der zu entfernenden Vertizes und somit auch Verschiebungskandidaten - sind in den Reklassifizierungsschritten verhältnismäßig wenige Vertizes zu bearbeiten bzw. zu reklassifizieren. Dies ergibt die kurze Laufzeit dieses Arbeitsschrittes.

Die vergleichsweise hohe Laufzeit für die Verschiebungsoperationen ist in der Zahl der Iterationen begründet. Beim Testfall SBH-1 mit dem hochauflösenden Modell wurden im Vergleich etwa doppelt so viele Vertizes entfernt, allerdings in nur 2 Iterationsschritten. Je mehr Vertizes in einem einzelnen Iterationsschritt aus dem Dreiecksnetz gelöscht werden können, desto effizienter verläuft das Verfahren. Die Zeit für einen Verschiebungsschritt steigt an, die Anzahl der Iterationsschritte wird jedoch reduziert und die konstante Laufzeit für Synchronisierungen und das Ausführen der Arbeitsschritte entfällt.

Neben dem Vergleich mit Testfall SBH-1 aufgrund der Zahl der entfernten Dreiecke wird auch der Vergleich mit SBH-3, der eine ähnliche prozentuelle Reduktion aufweist, gezogen.

Bei Testfall SBN werden etwa 64% der Dreiecke gelöscht. Dies entspricht einer stärkeren Vereinfachung als in Testfall SBH-3, bei dem für die gesamte Vereinfachung eine Laufzeit von 3,36 ms in Anspruch genommen wurde und 5 Iterationen vonnöten waren. Die kleine Zahl an entfernten Vertizes ist auf die geringere Problemgröße zurückzuführen. Somit hat eine Reduktion der Problemgröße um etwa 77% und eine annähernd gleich starke Vereinfachung nur eine Verminderung der Laufzeit von etwa 56% zur Folge.

Diese geringere Problemgröße wirft Schwierigkeiten auf, die bereits mit diesem Test gezeigt werden können. Durch die kleinere Vertexzahl lässt sich die notwendige Parallelität nicht garantieren. In jedem Schritt ist nur eine geringe Anzahl an Verschiebungskandidaten vorhanden.

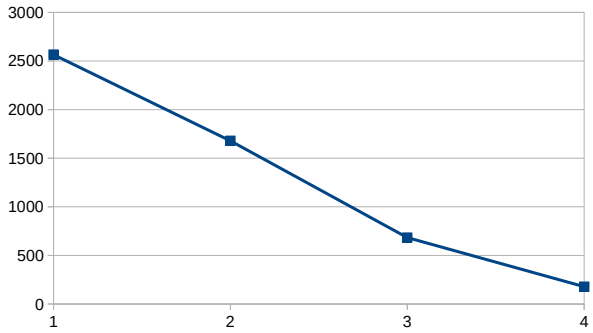

Abbildung 5.25: In den Iterationsschritten entfernte Vertizes

Abbildung 5.25 zeigt, wie viele Vertizes in jeder Iteration entfernt wurden. Hier wird das Problem der Vereinfachung eines Dreiecksnetzes mit geringer Auflösung deutlich. Während die Berechnungen in der Gegenüberstellung zum Netz mit mehr Vertizes schneller abgearbeitet werden, fällt der Laufzeitgewinn im Vergleich zur Verringerung der Problemgröße weniger stark aus.

Abbildung 5.25 zeigt die Anzahl der Verschiebungskandidaten. Schon in der ersten Iteration existieren nur etwa 2 500 zu löschende Vertizes mit beizubehaltenden Nachbarn. In weiterer Folge fällt diese Zahl noch weiter ab, bis in der letzten Iteration nur mehr 178 zu verschiebende Vertizes vorhanden sind.

Im Gegensatz zu den Iterationsschritten der Testfälle auf dem hochauflösenden Netz ist der Abfall der Zahl an Verschiebungskandidaten zwischen einzelnen Iterationsschritten weniger stark ausgeprägt. Bei den Testfällen SBH-1 bis SBH-5 wurde vor allem in den ersten Iterationen die Verschiebung einer großen Zahl an Vertizes beobachtet. Diese Kurve der entfernten Vertizes flacht bei späteren Iterationsschritten zunehmend ab.

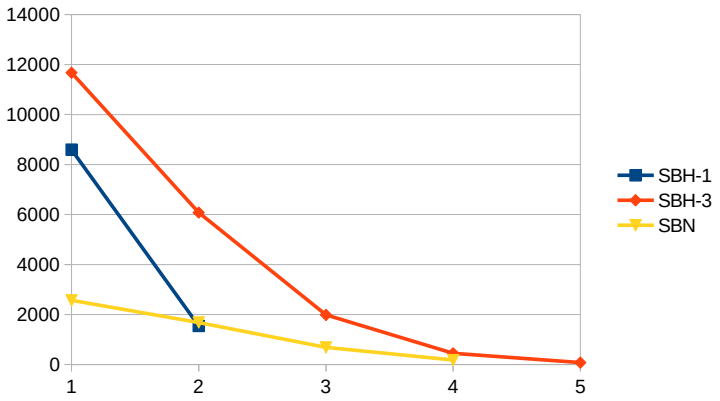

Abbildung 5.26: Vergleich von in den Iterationsschritten entfernten Vertizes

Abbildung 5.26 verdeutlicht diese Problematik. Der große Vorteil des hochauflösenden Modells liegt vor allem im ersten Iterationsschritt, wogegen beim Testfall SBN eine konstant geringere Vertexzahl für Verschiebungen zur Verfügung steht. Gleichzeitig tritt in diesem Diagramm auch das stärkere Sinken der Verschiebungskandidaten zwischen den einzelnen Iterationen beim hochauflösenden Modell hervor. In späteren Iterationen gleicht sich die Anzahl der Verschiebungskandidaten zwischen dem hochauflösenden und dem Dreiecksnetz von Testfall SBN an. Bei Testfall SBH-1 bzw. SBH-3 fällt die Zahl der Verschiebungskandidaten in den ersten Iterationsschritten schneller ab.

Abbildung 5.26 zeigt zusätzlich auch den Testfall SBH-3 des hochauflösenden Netzes. Dieser wurde in die Darstellung einbezogen, da er vom Grad der Vereinfachung mit dem Test auf dem Netz mit weniger Vertizes vergleichbar ist. Auch hier tritt ein ähnliches Verhalten auf. Die Mehrzahl an Vertizes ermöglicht bei den meisten Iterationsschritten eine wesentlich höhere Parallelität, wovon die Ausführung der Vereinfachung profitiert.

In späteren Iterationsschritten zeigt sich bei den drei Vereinfachungen ein ähnliches Verhalten. Die Zahl der Verschiebungskandidaten sinkt stark ab, so dass die Parallelität eingeschränkt wird und die Auslastung der Hardware nicht mehr optimal erfolgen kann.

Im Anschluss werden die Ergebnisse von Vereinfachungen, die aus komplexeren Modellen erstellt wurden, präsentiert.

5.6.3 Weitere Modelle

Neben den detaillierten Tests mit Hilfe des Stanford Bunny wurden noch weitere Modelle aus dem Stanford 3D Scanning Repository vereinfacht. Diese zeichnen sich durch eine größere Zahl an Vertizes bzw. Dreiecken aus und können somit herangezogen wer-

den, um die Laufzeit des Verfahrens bei Anwendung auf einen größeren Datensatz zu bestimmen. Für diese Tests wurde stets nur eine Vereinfachung ausgeführt, wobei jeweils ein Großteil der Dreiecke aus dem Netz entfernt wurde.

Das erste weitere Modell, das vereinfacht wurde, ist „Armadillo“ aus dem Stanford 3D Scanning Repository. Im Original ist es aus 172 974 Vertizes aufgebaut, die 345 944 Dreiecke bilden. Bei der Vereinfachung wurden 323 356 Dreiecke entfernt, womit sich das vereinfachte Modell nur mehr aus 22 588 Dreiecken zusammensetzt (entspricht einer Reduktion von über 93%). Hierfür wurde eine Laufzeit von 29,1 ms gemessen.

Abbildung 5.27: Armadillo: Vergleich originales Modell und vereinfachte Version

Abbildung 5.27 zeigt den Vergleich des originalen Modells (links) mit der vereinfachten Version (rechts). Das Modell zeichnet sich im Original durch eine detailreiche Oberfläche aus, die im Zuge der Vereinfachung entfernt wurde. Die Silhouette sowie die groben Ausformungen des Körpers blieben bei der Vereinfachung jedoch gut erhalten.

Das zweite zusätzliche Modell ist „Dragon“. Hier kommen im Original 437 645 Vertizes zum Einsatz, aus denen 871 414 Dreiecke gebildet werden.
Die Vereinfachung reduzierte die Anzahl der Dreiecke um 826 109. Dies entspricht einer Reduktion um 94,8%, womit sich das aus der Vereinfachung resultierende Modell aus 45 305 Dreiecken zusammensetzt. Dabei wurde eine Laufzeit von 80,1 ms gemessen.

Abbildung 5.28: Dragon: Vergleich originales Modell und vereinfachte Version

Der Vergleich zwischen dem Original und der vereinfachten Version ist in Abbildung 5.28 zu sehen. Ähnlich wie bei dem Modell Armadillo weist auch Dragon eine unruhige Oberfläche auf, die bei der Vereinfachung weitgehend entfernt wurde. Im Gegensatz dazu sind prägnante Details wie etwa die Zähne oder Zunge des Modells in nahezu unveränderter Form erhalten geblieben.

Das letzte Modell ist „Happy Buddha“. Der originale Datensatz enthält 543 652 Vertizes, die 1 087 716 Dreiecke bilden. Es ist somit das Modell mit den meisten Dreiecksdaten in diesen Tests. Bei der vereinfachten Version wurden 1 022 232 Dreiecke entfernt, womit dieses Modell von 65 484 Dreiecken gebildet wird (Reduktion um etwa 94%). Dementsprechend ergibt sich hier auch die längste für die Vereinfachung benötigte Laufzeit mit 96,1 ms.

Abbildung 5.29: Happy Buddha: Vergleich originales Modell und vereinfachte Version

Abbildung 5.29 zeigt den Vergleich zwischen dem Original und dem vereinfachten Modell. Während bei Armadillo und Dragon eine unruhige Oberfläche zu finden war, zeichnet sich das Modell Happy Buddha durch große Bereiche mit glatten Oberflächenverläufen (z. B. Kopf und Bauch) aus. Im Gegensatz dazu finden sich auch einige Kanten auf der Oberfläche wie etwa die Falten der Bekleidung.
Bei diesem Modell ist die Silhouette ebenfalls gut erhalten, und alle wesentlichen Details sind erkennbar. Die weichen Verläufe des Gewandes können allerdings durch die starke Reduktion der Anzahl der Dreiecke nicht mehr so exakt wiedergegeben werden.

Mit Stanford Bunny, Armadillo, Dragon und Happy Buddha wurden insgesamt vier Modelle für die Tests der Vertexgruppeneliminierung eingesetzt. Die Anzahl der Dreiecke der Modelle liegt zwischen 69 451 und 1 087 716.

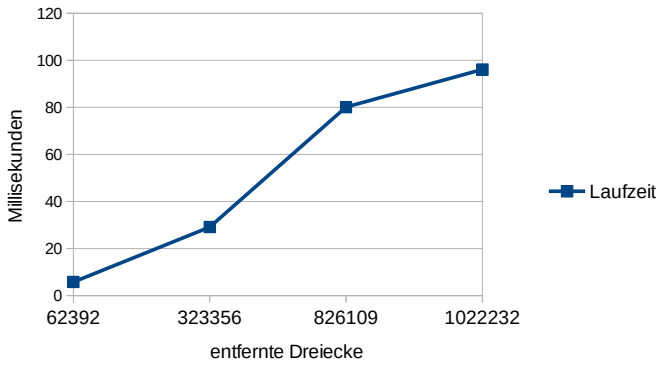

Abbildung 5.30: Vergleich Laufzeiten und entfernte Dreiecke

Abbildung 5.30 zeigt einen Vergleich der Laufzeiten für jedes Modell. Beim Stanford Bunny wurde die Laufzeit für Testfall 5 herangezogen, da bei den weiteren Modellen jeweils eine starke Vereinfachung durchgeführt wurde und dies somit den aussagekräftigsten Vergleich liefert. Die Laufzeiten sind stark von der Topologie der einzelnen Modelle sowie der Lage der beizubehaltenden Vertizes abhängig. Somit können Schwankungen beim Vergleich der einzelnen Laufzeiten entstehen.

Der nächste Abschnitt beschäftigt sich mit der Qualität der Vereinfachung.

5.7 Visuelle Auswertung der Vereinfachungen

Die Vertexgruppeneliminierung hat neben einem hohen Grad an Parallelität und damit einer möglichst kurzen Berechnungszeit auch die Ermittlung der Vereinfachung unter Einfluss der Daten des Betrachters zum Ziel. Es werden sowohl die Topologie und Ausformung der Oberfläche bei der Wahl von Vereinfachungen einbezogen als auch die Auswirkungen aus der Sicht des Beobachters.

Dieser Abschnitt präsentiert die optischen Resultate der Vereinfachungen. Hierbei wird auf die fünf Testfälle (SBH-1 bis SBH-5), die auf dem hochauflösenden Objekt durchgeführt wurden, zurückgegriffen. Das Modell mit weniger Vertizes findet keine Berücksichtigung: Es wurde durch eine Vereinfachung des hochauflösenden Stanford Bunny erstellt. Bei einer neuerlichen Vereinfachung durch die Vertexgruppeneliminierung könnte nicht bestimmt werden, welche etwaigen Qualitätseinbußen bereits durch die vorherige Vereinfachung verursacht wurden.

Die Qualität eines vereinfachten Objekts und das Ausmaß der Abweichung vom Original werden von Personen oft unterschiedlich stark bewertet. Es existiert keine einheitliche und umfassend aussagekräftige Metrik, um die Qualität einer Vereinfachung zu

bewerten. Aus diesem Grund wird weitgehend die Gegenüberstellung einer Darstellung des originalen Objekts und der vereinfachten Version eingesetzt, um die Auswirkungen der Vereinfachung zu demonstrieren. Auch hier wird diese Methode für die visuelle Auswertung der Ergebnisse eingesetzt.

Der erste Schritt stellt das originale Netz dem Ergebnis der Vereinfachung gegenüber. Hierfür kommt der Testfall SBH-5 zum Einsatz. Dieser repräsentiert die stärkste Vereinfachung, die in den Testfällen enthalten ist.

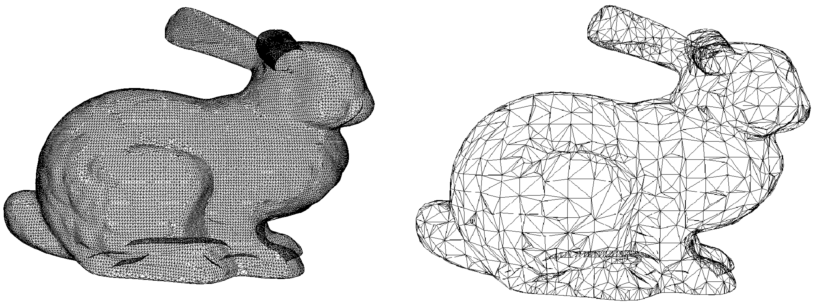

Abbildung 5.31: Vergleich originales Netz und stark vereinfachtes Resultat

Abbildung 5.31 zeigt die Gegenüberstellung des originalen Dreiecksnetzes (links) und der vereinfachten Version aus Testfall SBH-5 (rechts). Im Vergleich zum Original liegt eine sehr starke Vereinfachung des Modells vor. Dennoch sind die groben Züge des Objekts deutlich erkennbar. Auf größeren Flächen der Figur ist der Unterschied klar zu sehen. Durch die wesentlich gröbere Vernetzung entstehen glattere Flächen. Feine Details wie etwa Unebenheiten auf der Oberfläche können mit dem vereinfachten Netz nicht mehr dargestellt werden. Sie wurden bei der Vereinfachung entfernt.

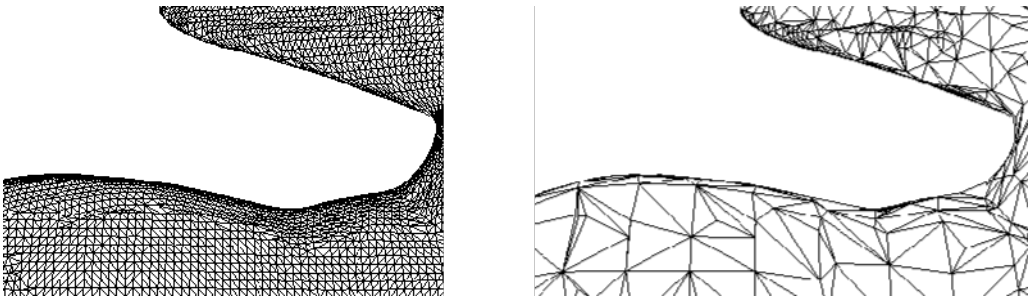

Abbildung 5.32: Detailvergleich: originale (links) und vereinfachte (rechts) Version

Abbildung 5.32 zeigt den detaillierten Vergleich eines Ausschnittes aus Abbildung 5.31. Hier sind erneut auf der linken Seite das originale Dreiecksnetz und rechts

die vereinfachte Version aus Testfall 5 abgebildet.

Die Silhouette ist zwar sehr gut erhalten geblieben, aufgrund der Vereinfachung kommt es jedoch zu einer etwas gröberen Darstellung des Umrisses. Alle Ausformungen und selbst Details wie die leichte Einbuchtung (Mitte) sind in der vereinfachten Version noch vorhanden.

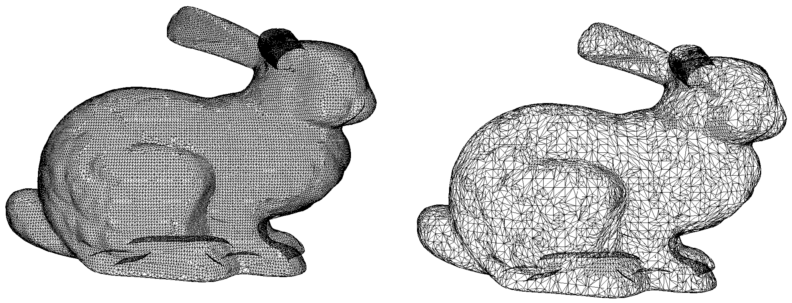

Abbildung 5.33: Vergleich: Original und Vereinfachung aus Testfall SBH-3

Abbildung 5.33 vergleicht das originale Modell mit Testfall SBH-3. Im Gegensatz zu Testfall SBH-5 findet bei SBH-3 eine wesentlich weniger starke Vereinfachung statt. Die Unterschiede werden deutlich und die Formen des Modells sind - bedingt durch die höhere Dreieckszahl - wesentlich besser erhalten als bei SBH-5.

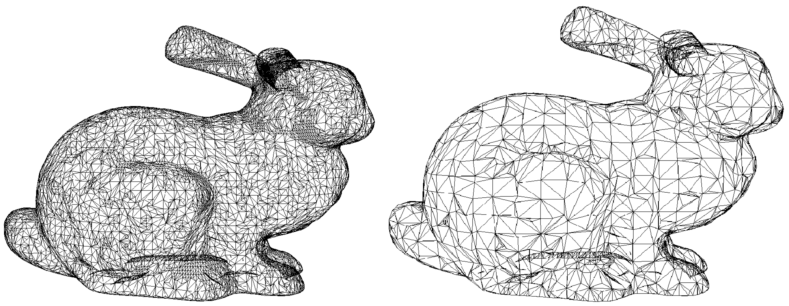

Abbildung 5.34: Vergleich: Vereinfachung aus Testfall SBH-3 und SBH-5

Abbildung 5.34 zeigt den direkten Vergleich zwischen Testfall SBH-3 und Testfall SBH-5. Im Bereich des Beines ist der Unterschied im Detailgrad deutlich sichtbar. Während auf der linken Seite (Testfall SBH-3) die Ausformungen sehr klar erhalten sind, ist rechts (SBH-5) nur mehr eine gröbere Struktur zu erkennen.

Ein weiteres, sehr gutes Beispiel ist der Kopf des dargestellten Hasen. Links sind hier noch Einzelheiten auf der Oberfläche wie etwa das Auge und die Nase zu sehen. Auf der rechten Seite ist die Form des Kopfes zwar trotz der starken Vereinfachung gut

erhalten geblieben, die Details sind jedoch großteils verloren gegangen und kaum mehr auszumachen.

Vergleicht man das originale Modell mit Testfall SBH-3 (Abbildung 5.33), so ist in der Silhouette des Objekts kaum ein Unterschied wahrnehmbar. Hier ist besonders die Abhängigkeit der Berechnung von der Position des Betrachters von Vorteil. Bei diesem Testfall wurde die Zahl der Dreiecke um über 58% reduziert. Trotz dieses Faktums sind die Ausformungen des Modells gut erhalten geblieben, das Objekt und die Formen sind leicht zu erkennen. Allerdings wurden auch bei diesem Vereinfachungsgrad schon Details und unruhige Oberflächenformen entfernt.

Im nächsten Abschnitt folgt eine Gegenüberstellung mit Beispielen für andere Algorithmen, die ebenfalls eine Vereinfachung in Echtzeit durchführen.

5.8 Vergleich mit existierenden Algorithmen

Dieser Abschnitt zieht einen Vergleich mit existierenden Algorithmen. Es wurden mehrere Verfahren ausgewählt:

- **1. Schrittweise Dreiecksnetze** ([Hop97]) siehe Unterabschnitt 2.4.1
- **2. Vertexgruppierung** ([DT07]) siehe Unterabschnitt 2.4.3
- **3. GPU-beschleunigte Quadrik-Fehlermetrik** ([GDG11]) siehe Unterabschnitt 2.4.2
- **4. Vereinfachung mittels Morton Integralen** ([LB15]) siehe Unterabschnitt 2.4.8
- **5. Vereinfachung durch Reduktion von Teilbäumen** ([LK16]) siehe Unterabschnitt 2.4.9

Diese Verfahren wurden aus mehreren Gründen ausgewählt: Die schrittweisen Dreiecksnetze und die Vertexgruppierung sind etablierte, weit verbreitete Vorgehensweisen, die schon seit geraumer Zeit existieren und weiterentwickelt wurden. Zusätzlich wurden für beide Algorithmen GPU-beschleunigte Versionen beschrieben, um die Leistung moderner Hardware zu nutzen und den Echtzeiteinsatz zu ermöglichen. Die Vereinfachung mittels Morton Integralen und die Vereinfachung durch Reduktion von Teilbäumen sind neuere Verfahren, die für eine Ausführung auf einer GPU entwickelt wurden und sich durch gute Laufzeiten auszeichnen.

Während in [PP15] (Unterabschnitt 2.4.6) und [SN13] ebenfalls Verfahren präsentiert werden, die auf GPU-Beschleunigung zurückgreifen, sind diese Algorithmen nicht für die Berechnung einer Vereinfachung in Echtzeit geeignet. Weiter bestätigt wird diese Einschätzung bei [PP15], da laut den Ergebnissen in [PP15] bis zu mehreren Sekunden für die Vereinfachung benötigt werden. Aus diesem Grund werden diese Algorithmen hier nicht berücksichtigt.

Der Ansatz in [OKK15] erzielt unter Berücksichtigung der Komplexität der Dreiecksnetze ebenfalls sehr gute Laufzeiten. Dieses Verfahren wird hier nicht einbezogen, da sich die guten Resultate auf die hohe Parallelität durch die sehr hohe Anzahl an Dreiecken in den Testfällen (etwa 28 000 000 bis 197 000 000 Dreiecke) erklären lässt. Überdies basiert die Vereinfachung der Teilbereiche bei [OKK15] auf dem Ansatz von [GDG11]. Ein Vergleich der Ergebnisse des Verfahrens aus [GDG11] mit jenen der Vertexgruppeneliminierung findet sich in Unterabschnitt 5.8.3.

Aus den Ergebnissen von [GDG11] und [OKK15] lässt sich schließen, dass der Vorteil paralleler Verfahren besonders bei der Vereinfachung sehr komplexer Modelle, bei denen eine große Zahl an Vereinfachungsoperationen gleichzeitig ausgeführt werden kann, zum Tragen kommt. Da die Vertexgruppeneliminierung ebenfalls durch die parallele Ausführung von Vereinfachungsoperationen eine Verkürzung der Laufzeit erzielt, ist zu erwarten, dass bei der Vereinfachung von komplexeren Modellen als jenen, die für die Testfälle in dieser Arbeit vereinfacht wurden, der Leistungsgewinn durch die Parallelität noch größer ausfällt. Hochkomplexe Modelle - wie etwa aus dem Digital Michelangelo Project ([LPC+00]) - wurden bei den Testfällen der Vertexgruppeneliminierung jedoch nicht berücksichtigt: Diese Modelle (wie sie auch bei [OKK15] zum Einsatz kommen - etwa 160 000 000 bis 200 000 000 Dreiecke) können aufgrund des Speicherbedarfs nicht mehr vollständig in den Speicher der Graphikkarte des Testsystems (siehe Abschnitt 5.3) geladen werden. Stattdessen würden Teilbereiche eines Modells in den Speicher geladen und vereinfacht. Ein Beispiel für ein derartiges Vorgehen findet sich in [OKK15]: Für die Zerlegung in Teilbereiche und die starke Vereinfachung des Modells „Atlas“ aus dem Digital Michelangelo Project wird eine Berechnungszeit von 2 233 Sekunden angegeben, wobei die Zerlegung in Teilbereiche über 1 000 Sekunden in Anspruch nimmt. Da sich diese Arbeit mit der Vereinfachung von Dreiecksnetzen in Echtzeit befasst, wurden derart komplexe Testfälle aufgrund der hohen Laufzeiten nicht als sinnvoll erachtet. Stattdessen wurden die Problemgrößen der Testfälle für die Vertexgruppeneliminierung so gewählt, dass Laufzeiten erzielt werden, die sich für den Echtzeiteinsatz eignen.

5.8.1 Schrittweise Dreiecksnetze

Zuerst wird der Vergleich mit schrittweisen Dreiecksnetzen gezogen. Wie in Kapitel 2 beschrieben, wurde für dieses Verfahren eine Erweiterung („Parallel View Depen-

dent Refinement of Progressive Meshes“, kurz PVDPM) entwickelt und in [HSH09] mit weiteren Details in [HSH10] präsentiert. Dieser Algorithmus stützt sich analog zu den früheren, die auf diesem Ansatz beruhen, auf eine Menge von vorberechneten Vereinfachungen, deren Auswahl allerdings die Position des Betrachters zu Grunde liegt.

Ein weiterer Unterschied zur Vertexgruppeneliminierung ist der Aufbau des Netzes. Während hier ein Verfahren beschrieben wurde, das auf ein hochauflösendes Netz als Datenbasis zurückgreift und aus diesem Vertizes entfernt, basiert PM auf der Idee, zuerst ein sehr stark vereinfachtes Netz zu berechnen, das anschließend - also zur Laufzeit - verfeinert wird.

Abbildung 2.18 zeigt ein Beispiel dieses Aufbaus in der klassischen Version der schrittweisen Dreiecksnetze, bei dem der Mangel der Berücksichtigung der Kameraposition klar hervortritt. Erst bei dem vollständig wiederhergestellten Netz ist eine runde Form der Silhouette gegeben. Den Qualitätsverlust der übrigen Versionen erkennt man deutlich. Die kameraabhängige Version des Algorithmus versucht, diese Einschränkung zu umgehen.

Abbildung 5.35: Beispiel: PVDPM [HSH09]

Abbildung 5.35 stellt eine Anpassung von zwei Dreiecksnetzen nach dem kameraabhängigen schrittweise Dreiecksnetze Verfahren dar. Hier ist der Detailgrad auf der linken Seite von links nach rechts und im rechten Bild von unten nach oben ansteigend.

Diese Abbildung verdeutlicht ein Problem des Verfahrens: Die geringe Anzahl an Vertizes im Netz, von dem ausgehend zusätzliche Dreiecke wieder eingefügt werden, ist ein limitierender Faktor für die Qualität. Eine deutliche Verschlechterung der Silhouette wird sichtbar. Kantige Formen und eine ungenaue Wiedergabe des originalen Netzes sind die Folge. Es bedarf der Wiederherstellung einer großen Anzahl an Vertizes, um die Silhouette zu rekonstruieren.

Im Gegensatz dazu hat PVDPM den Vorteil, dass die gewählten Vereinfachungen unter Umständen besser selektiert werden können. Die Vorberechnung der Vereinfachungen schränkt zwar die Freiheit bei der Rekonstruktion ein, erlaubt jedoch aufwändigere Verfahren für die Bestimmung möglicher Vereinfachungen.

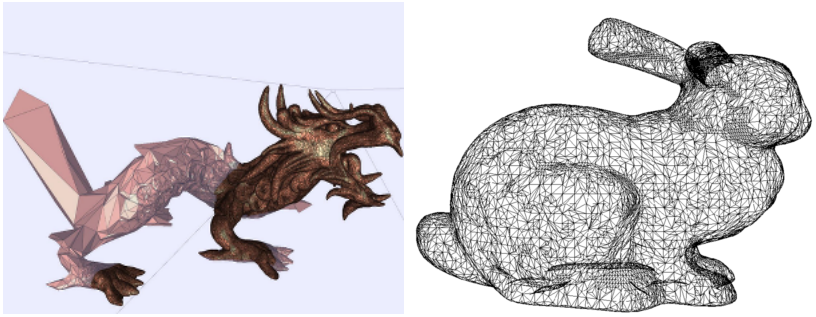

Abbildung 5.36: Vergleich: PVDPM ([HSH09]) mit Testfall 3

Abbildung 5.36 vergleicht die beiden Resultate. Es ist erkennbar, dass bei jedem Grad der Vereinfachung des im Zuge dieser Arbeit entwickelten Algorithmus die Silhouette und Ausformungen der Oberfläche gut erhalten bleiben. Die blickwinkelabhängige Variante der schrittweisen Dreiecksnetze hingegen weist in manchen Bereichen eine etwas gleichmäßigere Triangulierung auf.

In [HSH10] wird eine Reihe von Testläufen mit dem Algorithmus präsentiert. Es ist dabei anzumerken, dass PVDPM nicht die gesamte Aktualisierung des Dreiecksnetzes in einem Arbeitsschritt durchführt, sondern die Ausführung über mehrere Schritte verteilt, um den Einfluß auf die Darstellungszeit zu minimieren.

Die Autoren führen in [HSH10] an, dass die Testläufe mit Hilfe einer Nvidia GeForce 8800 GTX ausgeführt wurden. Die verwendeten Geometriedaten lagen im Bereich von 100 000 bis etwa 500 000 Dreiecken. Die Autoren geben die erforderlichen Zeiten für die Aktualisierung für diese Anwendung im Bereich von 10 bis etwa 70 ms an.

In [DG12] wurde die parallele Ausführung von schrittweisen Dreiecksnetzen im Vergleich zu [HSH09] stark beschleunigt. In den Testfällen von [DG12] wird eine Szene verwendet, die in der maximalen Auflösung etwa 980 000 000 Dreiecke enthält. Im Durchschnitt werden für die Geometrie nur etwa 5 500 000 Dreiecke eingesetzt. Bei einer vorbestimmten Kamerafahrt wurden bei Berechnung auf einer Nvidia Geforce GTX 580 Laufzeiten zwischen 10 und 30 Millisekunden für die Aktualisierung der Geometrie vor jedem Einzelbild gemessen.

Der Ansatz in [DG12] ist jedoch eine Implementierung der schrittweisen Dreiecksnetze und basiert daher auf einer Menge von vorberechneten Vereinfachungsoperationen. Im Gegensatz dazu wird bei der Vertexgruppeneliminierung keine Menge an Vereinfachungsoperationen vorbestimmt, sondern zur Laufzeit ermittelt.

Im nächsten Abschnitt wird die Vertexgruppeneliminierung mit der Vertexgruppierung verglichen.

5.8.2 Vertexgruppierung

Hier wird der Vergleich mit dem Vertexgruppierungs-Verfahren aus [DT07] gezogen. Dieser Algorithmus unterscheidet sich von der Vertexgruppeneliminierung, da er nicht topologieerhaltend vorgeht (siehe Unterabschnitt 2.3.5). Hier wird eine Reihe von Zellen über das Dreiecksnetz gelegt, und alle Vertizes innerhalb einer dieser Zellen werden zu einem einzelnen Vertex zusammengefasst.

Das Resultat ist ein sehr schneller und effizienter Algorithmus, die Qualität des vereinfachten Netzes leidet allerdings.

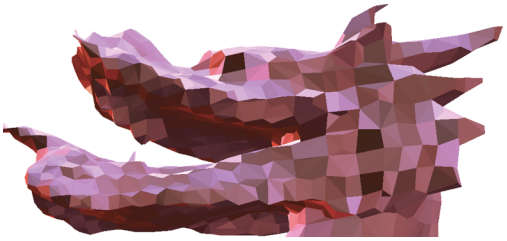

Abbildung 5.37: Vereinfachung nach [DT07]

Abbildung 5.37 zeigt das Beispiel eines mit diesem Ansatz vereinfachten Dreiecksnetzes. Die Nachteile dieses Algorithmus sind deutlich erkennbar. Er wurde für eine geringe Laufzeit entwickelt, die Qualität des finalen Dreiecksnetzes ist jedoch gravierend schlechter als andere Vereinfachungen.

Neben den Verlusten von Details und der kantigen Silhouette ist vor allem die deutliche Bildung von Kacheln - bedingt durch die regelmäßigen Zellen, die zum Einsatz kommen - erkennbar. Dies wird als störend wahrgenommen, und erst eine sehr große Anzahl dieser Zellen kann ein visuell ansprechendes Ergebnis erzeugen.

Der Vorteil dieses Algorithmus liegt jedoch in der Bearbeitungszeit. Während bei der Vertexgruppeneliminierung das Ausmaß der Vereinfachung - wie oben gezeigt - die Zahl der benötigten Iterationsschritte beeinflusst und diese sogar superlinear ansteigt, sind in [DT07] immer nur drei Arbeitsschritte vonnöten (siehe Unterabschnitt 2.4.3). Diese Eigenschaft eliminiert eine Abhängigkeit der Laufzeit vom Grad der Vereinfachung weitgehend und schafft somit eine stabilere und vorhersagbare Laufzeit.

Des Weiteren hat das Verfahren den Vorteil, dass es auch sehr starke Vereinfachungen in kurzer Laufzeit errechnen kann, ohne hierfür eine große Menge an Ressourcen zu verbrauchen.

Für die Testläufe kommt in [DT07] eine „DirectX 10 ATI Radeon GPU“ zum Einsatz, und es wird auf Modelle aus dem Stanford 3D Scanning Repository zurückgegriffen. Die Autoren berichten eine Laufzeit der Vereinfachung von 13 ms (Stanford Bunny), 55 ms

(Armadillo), 117 ms (Dragon) und 146 ms (Happy Buddha). Damit zeigt dieser Algorithmus einen ähnlichen Anstieg der Laufzeiten wie die Vertexgruppeneliminierung.

Anschließend wird der Vergleich zwischen der Vertexgruppeneliminierung und der GPU-beschleunigten Version der Quadrik-Fehlermetrik präsentiert.

5.8.3 GPU-beschleunigte Quadrik-Fehlermetrik

Das in [GDG11] präsentierte Verfahren (Unterabschnitt 2.4.2) basiert auf der Quadrik-Fehlermetrik und sucht eine Menge von Kanten, die gleichzeitig aus einem Dreiecksnetz entfernt werden können. Im Gegensatz zur Vertexgruppeneliminierung werden dabei nur Kantenreduktionen ausgeführt, die sich gegenseitig nicht beeinflussen. Da [GDG11] auf dem Verfahren nach Garland und Heckbert [GH97] beruht, wird eine hohe visuelle Qualität der Vereinfachung erreicht. Es ist jedoch - im Gegensatz zur Vertexgruppeneliminierung - kein blickwinkelabhängiges Verfahren, wodurch vor allem entlang der Silhouette die Auswirkungen der Vereinfachung bemerkbar sind.

Abbildung 5.38: Vereinfachung eines Modells durch GPU-beschleunigte Quadrik-Fehlermetrik [GDG11]

Abbildung 5.38 zeigt dieses Phänomen. Die Formgebung des Modells ist auch bei stärkeren Vereinfachungen gut erhalten. Vergleicht man jedoch die Ergebnisse mit jenen in dieser Arbeit (Abbildung 5.33 und Abbildung 5.34), so sind bei der Vertexgruppeneliminierung deutlich weniger Unterschiede zwischen originalem und vereinfachtem Dreiecksnetz entlang der Silhouette erkennbar.

In [GDG11] wird für die starke Vereinfachung eines Modells mit 807 365 Dreiecken eine Laufzeit von 0,29 Sekunden und für ein weiteres Modell mit 871 414 Dreiecken eine Berechnungszeit von 0,28 Sekunden angegeben. Die Resultate wurden mit einer Nvidia Geforce GTX 580 erzielt. Diese GPU verfügt über eine geringere Anzahl an Kernen als jene, die für die Testfälle in dieser Arbeit eingesetzt wurde (512 im Vergleich zu 1344, siehe Abschnitt 5.3). Es ist somit davon auszugehen, dass auf einem gemeinsamen Testsystem die Berechnungszeit des Verfahrens aus [GDG11] nicht wesentlich langsamer ist als jene der Vertexgruppeneliminierung.

Anschließend wird der Vergleich zwischen der Vertexgruppeneliminierung und der Vereinfachung mittels Morton Integralen gezogen.

5.8.4 Vereinfachung mit Morton Integralen

An dieser Stelle soll der Vergleich zwischen der Vertexgruppeneliminierung und der Vereinfachung mit Morton Integralen (Unterabschnitt 2.4.8, [LB15]) gezogen werden. Im Unterschied zu [DT07] werden Vertizes bei der Vereinfachung mit Morton Integralen nicht zusammengefasst, wenn sie sich innerhalb eines Bereichs befinden. Stattdessen werden Vertizes mit aufeinanderfolgendem Z-Wert zu einem Vertex verschmolzen. Dies resultiert in einer Vereinfachung mit besserer visueller Qualität als bei [DT07].

Trotz dieser Verbesserungen treten bei [LB15] ähnliche Probleme auf wie bei [DT07]: Das Verfahren wurde für kurze Berechnungszeiten entwickelt und nicht für eine hohe Qualität der Vereinfachung.

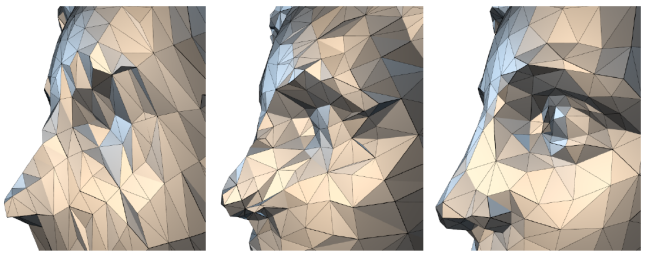

Abbildung 5.39: Vergleich: Vereinfachung nach [DT07] (links), mit Morton Integralen (Mitte) und iterative Kantenreduktion nach Garland und Heckbert ([GH97], rechts) [LB15]

Abbildung 5.39 zeigt den Vergleich mehrerer Vereinfachungsverfahren. Bei der Vertexgruppierung nach [DT07] (links) ist deutlich die mindere Qualität der Vereinfachung erkennbar. Details im Bereich des Auges sind verloren gegangen. In der Mitte wurde dasselbe Modell mit dem Verfahren aus [LB15] vereinfacht. Das Ergebnis zeigt eine bessere visuelle Qualität als die Vertexgruppierung. Im Vergleich zu einer Vereinfachung mittels Kantenreduktion (rechts, nach [GH97]) ist allerdings vor allem im Bereich der Nase und Augenbraue eine deutlich schlechtere Wiedergabe der Formen zu erkennen. Besonders im Bereich der Nase wurde bei der Vereinfachung nach [LB15] die Silhouette deutlich verändert. Betrachtet man die Ergebnisse der Vereinfachung mit Vertexgruppeneliminierung (siehe Abbildung 5.33 und Abbildung 5.34), so macht dieser Vergleich deutlich, dass das Verfahren aus [LB15] nur eine eingeschränkte visuelle Qualität erzielt. Besonders entlang der Silhouette ist die Formgebung bei der Vertexgruppeneliminierung besser erhalten.

Die Vereinfachung mit Morton Integralen kann vor allem durch sehr gute Laufzeiten überzeugen. Für die Testfälle in [LB15] kommt vergleichbare Hardware zum Einsatz, wie sie auch für die Laufzeitmessungen in dieser Arbeit (siehe Abschnitt 5.3) eingesetzt wurde. Die Autoren von [LB15] geben an, eine Vereinfachung des Stanford Bunny (ähnlich SBH-5, siehe Unterabschnitt 5.5.5) in 6,4 Millisekunden zu errechnen. Bei komplexeren Modellen zeichnet sich die geringe Laufzeit stärker ab. Eine Vereinfachung eines Modells mit etwa 1 700 000 Dreiecken (reduziert auf 44 500 Dreiecke) wird in 22,1 Millisekunden erstellt. Während die Laufzeiten der Vereinfachung mit Morton Integralen jenen der Vertexgruppeneliminierung bei komplexeren Modellen überlegen sind, ist die Qualität der Vereinfachung - wie Abbildung 5.3 zeigt - begrenzt. Die Vertexgruppeneliminierung hat hingegen zum Ziel, mittels Blickwinkelabhängigkeit eine hochqualitative Vereinfachung bei guten Laufzeiten zu berechnen.

Abschließend wird der Vergleich zwischen der Vertexgruppeneliminierung und der Vereinfachung durch Reduktion von Teilbäumen gezogen.

5.8.5 Vereinfachung durch Reduktion von Teilbäumen

Das letzte Verfahren, das mit der Vertexgruppeneliminierung verglichen wird, ist die Vereinfachung durch Reduktion von Teilbäumen ([LK16], siehe Unterabschnitt 2.4.9). Hier wird ein ähnlicher Ansatz wie bei der Vertexgruppeneliminierung verfolgt: ein paralleles Verfahren basierend auf eingeschränkten Kantenreduktionen, bei dem die Operationen zur Laufzeit ausgewählt und durchgeführt werden. Die bei [LK16] eingesetzten Datenstrukturen erlauben es, mehrere eingeschränkte Kantenreduktionen auf benachbarten Vertizes auszuführen, ohne Wettbewerbsbedingungen zu verursachen. Dies wird genutzt, um eine Menge von Bäumen, die mit Vertizes des Dreiecksnetzes gebildet werden, in einem Vertex zusammenzufassen. Faltungen und inkonsistente Topologie in den Bäumen werden erkannt und nicht ausgeführt. Im Gegensatz dazu setzt die Vertexgruppeneliminierung auf eine Menge von Begrenzungsebenen, die Kombinationen von eingeschränkten Kantenreduktionen, die Faltungen oder inkonsistente Topologie verursachen könnten, verhindert.

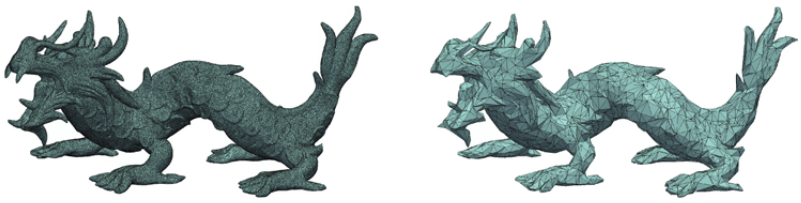

Abbildung 5.40: Ergebnisse der Vereinfachung bei [LK16]

Abbildung 5.40 zeigt ein Beispiel für die visuellen Ergebnisse der Vereinfachung in [LK16]. Das Modell links (etwa 7 000 000 Dreiecke) wurde stark vereinfacht (rechts, 6 400 Dreiecke). Ähnlich den Ergebnissen der Vertexgruppeneliminierung wurden die Silhouette und Details auf der Oberfläche gut erhalten.

Das Verfahren in [LK16] erreicht sehr gute Laufzeiten. Für die Reduktion der Dreieckszahl um 99% bei einem Modell mit 1 035 698 Dreiecken wird eine Laufzeit von 65 Millisekunden angegeben. Während diese Laufzeit deutlich kürzer ist als der vergleichbare Test für die Vertexgruppeneliminierung (Happy Buddha, etwa 96 Millisekunden, Vereinfachung um etwa 94%), ist anzumerken, dass in [LK16] für die Testläufe eine Graphikkarte zum Einsatz kommt, die über doppelt so viele Rechenkerne verfügt wie die GPU bei den Testfällen der Vertexgruppeneliminierung. Bei einem weiteren Testfall in [LK16] wird ein Modell mit 39 694 Dreiecken vereinfacht (Reduktion um 90% der Dreiecke). Diese Berechnung benötigt 7 Millisekunden. Der Testfall SBH-5 in dieser Arbeit (Stanford Bunny, 69 451 Dreiecke, Vereinfachung um etwa 90%, Unterabschnitt 5.5.5) benötigt trotz höherer Zahl an Dreiecken und leistungsschwächerer Hardware weniger als 6 Millisekunden.

Unter Berücksichtigung der Leistungsunterschiede der Hardware kann davon ausgegangen werden, dass die Vereinfachung durch Reduktion von Teilbäumen bei den vergleichbaren Testfällen keinen deutlichen Leistungsvorteil erzielen kann und die Vereinfachung von wenig komplexen Modellen im Vergleich zur Vertexgruppeneliminierung sogar wesentlich längere Berechnungszeiten benötigt.

5.9 Zusammenfassung

Dieses Kapitel präsentiert die Leistung und die Ergebnisse der Vertexgruppeneliminierung anhand mehrerer Testfälle, die ausgewählt wurden, um die Auswirkungen verschiedener Parameter und Szenarien auf den Ablauf des Verfahrens zu bestimmen. Diese Testfälle simulieren unterschiedliche Grade der Vereinfachung, die bei ansteigender Distanz zwischen Objekt und Betrachter zur Anwendung kommen. Somit kann das jeweilige Ausmaß der Vereinfachung betrachtet und seine Auswirkung auf die Komponenten des Verfahrens ausgewertet werden.

Die Analyse der Ausführung der Tests zeigt, dass durch den hohen Grad an Parallelität bei der Vereinfachung eine kurze Laufzeit erreicht wird. Die Ergebnisse weisen einen Anstieg der Laufzeit bei stärkeren Vereinfachungen auf und entsprechen somit dem typischen Verhalten von Ansätzen, die ein hochauflösendes Netz als Basis benutzen, das im Laufe der Berechnungen vereinfacht wird.

Die Testläufe zeigen auch, dass die Vertexgruppeneliminierung durch den Einsatz von moderner, paralleler Hardware stark profitieren kann, die mögliche Parallelität aber durch eine ungünstige Topologie des Dreiecksnetzes eingeschränkt wird. Trotz dieser teilweisen Einschränkung der Parallelität liegen die gemessenen Laufzeiten in einem

Zeitrahmen, der die Echtzeitverarbeitung ermöglicht.

Neben den Ergebnissen der Leistungsanalyse wird der Vergleich mit bestehenden Algorithmen gezogen. Hier erzielt die Vertexgruppeneliminierung durch die Einbeziehung der Kamera und die freie Wählbarkeit möglicher Vereinfachungen positive Resultate. Silhouette und Details des vereinfachten Objekts bleiben deutlich bestehen. Auch bei starker Reduktion der Dreieckszahl ist das dargestellte Objekt gut erhalten, selbst wenn manche Einzelheiten auf der Oberfläche bedingt durch die geringe Dreieckszahl verloren gehen.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Zuge dieser Arbeit wurde ein Verfahren entwickelt, das aus einem bestehenden Dreiecksnetz eine vereinfachte, weniger detaillierte Version errechnet. Die Vertexgruppeneliminierung kann während des Darstellungsvorganges eingesetzt werden, was durch einen hohen Grad an Parallelität der einzelnen Operationen und damit einer effizienten Ausführung auf paralleler Hardware ermöglicht wird. Dieser Aspekt erlaubt die Nutzung moderner, paralleler Prozessoren und GPUs, resultiert in einer Reduktion der Berechnungszeit und ermöglicht den Einsatz in Echtzeit.

Durch die Berechnung während des Darstellungsvorganges kann die Position des Betrachters bei der Durchführung der Vereinfachungsoperationen berücksichtigt werden. Unter Einbeziehung der Topologie eines Dreiecksnetzes können Vereinfachungen gezielter durchgeführt und die Oberflächenbeschaffenheit mit Hinblick auf die sichtbaren Auswirkungen manipuliert werden. Überdies kann diese Eigenschaft zu einer Reduktion der notwendigen Vereinfachungsoperationen herangezogen werden: Bei Einsatz der Dreiecksfilterung (Blickpyramidenfilterung, Filterung von Rückseiten oder Filterung von Verdeckungen, siehe Unterabschnitt 2.2.2) können Dreiecke, die durch diese Ansätze entfernt werden, von der Vereinfachung ausgenommen werden.

Die eingesetzten Vereinfachungsoperationen wurden so angepasst, dass sie eine parallele Abarbeitung von benachbarten Vertizes erlauben. Neben der Auswahl der zu entfernenden Vertizes findet auch die Bestimmung der durchzuführenden Vereinfachung unter Berücksichtigung der Position des Betrachters statt. Dies ermöglicht eine weitere Reduktion der sichtbaren Artefakte, die durch die Operationen verursacht werden.

Die Vertexgruppeneliminierung bietet dynamische Kontrolle des resultierenden Dreiecksnetzes. Anstatt einer Vereinfachung bis zu einer bestimmten Vertexzahl wird die Berechnung beendet, wenn eine gewünschte Qualität des Netzes erreicht ist.

Das Verfahren hat nur eine geringe Auswirkung auf die Daten der Vertizes und lässt

sich als zusätzlicher Verarbeitungsschritt in den Darstellungsvorgang einbinden. Es ist somit ohne großen Aufwand möglich, eine bestehende Darstellung so zu modifizieren, dass eine vereinfachte Version der Objekte berechnet wird.

6.2 Anwendungsmöglichkeiten

Die Vertexgruppeneliminierung kann eingesetzt werden, um Dreiecksnetze während der Darstellung zu vereinfachen und so den Darstellungsaufwand zu reduzieren. Dabei ist zu berücksichtigen, dass die für die Vereinfachung benötigte Rechenleistung in die Darstellungszeit miteinbezogen werden muss. Trotz des Mehraufwandes durch die Vereinfachung kann es zu einer Beschleunigung des Darstellungsvorganges kommen, da komplexe Darstellungsverfahren auf Modellen mit geringerer Komplexität ausgeführt werden können.

Bei den Darstellungsverfahren werden oft komplexe Berechnungen wie Beleuchtung, Animation oder Simulation von physikalischen Einwirkungen auf die Geometrie einer Szene eingesetzt. Der Aufwand dieser Berechnungen ist abhängig von der Anzahl der Dreiecke der dargestellten Objekte. Findet zuerst ein Vereinfachungsschritt statt, so kann sich der Aufwand für nachfolgende Berechnungen stark reduzieren.

Während die Vertexgruppeneliminierung entwickelt wurde, um Vereinfachungen zur Laufzeit zu berechnen, ist es nicht zwangsläufig nötig, die parallele Verarbeitung während der Darstellung durchzuführen. Dieser Vorgang erlaubt es, komplexe Szenen zu vereinfachen und somit eine Szene zu erstellen, die weniger Rechenaufwand für die Darstellung erfordert.

Durch die Entwicklungen der adaptiven Anpassung von Dreiecksnetzen in den letzten Jahren wurden hardwareunterstützte Verfahren entwickelt, die zusätzliche Dreiecke bzw. Details zu bestehenden Dreiecksnetzen hinzufügen. Die Vertexgruppeneliminierung lässt sich mit derartigen Algorithmen kombinieren. Ein Dreiecksnetz mittlerer Auflösung kann als Ausgangspunkt gewählt werden, so dass je nach Positionierung in der Szene entweder dynamisch vereinfacht oder verfeinert wird.

Des Weiteren lässt sich die Vertexgruppeneliminierung auch dazu einsetzen, statisch vorberechnete, vereinfachte Dreiecksnetze zu ermitteln und ermöglicht auch bei Modellen mit hoher Dreieckszahl kurze Berechnungszeiten.

6.3 Ausblick

Die Vertexgruppeneliminierung bietet eine Reihe von Erweiterungs- und Optimierungsmöglichkeiten, deren Umsetzung jedoch den Rahmen dieser Arbeit überschreiten würde. Hier wird kurz auf Ideen und Anregungen für Erweiterungen und Optimierungen

eingegangen. Diese werden getrennt nach den einzelnen Teilschritten des Verfahrens behandelt.

6.3.1 Klassifizierung

Die Klassifizierung wird in der gegenwärtigen Form des Verfahrens per Vertex bestimmt. Dies hat zur Folge, dass die Auswirkungen von entfernten Vertizes, die eine gemeinsame Kante besitzen, nicht berücksichtigt werden.

Der Ansatz lässt sich erweitern, indem nicht nur einzelne Vertizes, sondern Bereiche des Dreiecksnetzes betrachtet werden und beim Berechnen des Vertexfehlers auch das mögliche Entfernen von benachbarten Vertizes mit einfließt. Weist also ein benachbarter Vertex einen sehr großen Vertexfehler auf, so sollte der Algorithmus berücksichtigen, dass der Nachbar wahrscheinlich für ein Entfernen markiert wird und den eigenen Vertexfehler entsprechend anpassen.

Während ein derartiger Ansatz die Komplexität der Klassifizierung erhöhen würde und auch im Reklassifizierungsschritt Anwendung finden müsste, hätte er mehrere Vorteile:

- Reduzierte Vertexzahl für Verschiebungsoperationen
- Erhöhte Zahl an Verschiebungskandidaten
- Verbesserte Qualität des vereinfachten Netzes

Fließt das Entfernen von benachbarten Vertizes ein, kann eine Reduktion der initialen Menge an zu entfernenden Vertizes erzielt werden. Damit kann die Anzahl der Iterationsschritte reduziert und zusätzliche Zielpunkte für die Verschiebungen geschaffen werden. Es kommt zu einer Erhöhung der Zahl von Verschiebungskandidaten und einer Reduktion der nötigen Reklassifizierungen. Die Vertexgruppeneliminierung entfernt eine Reihe von Vertizes gleichzeitig und führt anschließend einen Reklassifizierungsschritt aus. Benachbarte Vertizes können entfernt werden und eine unerwünschte Abweichung vom originalen Dreiecksnetz kann entstehen. Durch eine verbesserte Klassifizierung könnte dieses Verhalten verringert und so die Qualität der Vereinfachung gesteigert werden.

Eine weitere Verbesserung der Klassifizierung lässt sich durch die Extraktion von Merkmalen der Oberflächen erzielen. Dieser Ansatz dient dazu, markante Ausformungen auf der Oberfläche eines Objekts zu bestimmen. Wird ein Objekt vereinfacht, so ist besonders der Erhalt dieser markanten Eigenschaften von Bedeutung. Kleinere Details können etwa bei größerem Abstand zur Kamera nur schlecht oder gar nicht wahrge-

nommen werden. Die markanten Ausformungen hingegen dienen dazu, die Form des Objekts wiedererkennen zu können.

Dementsprechend lässt sich die Klassifizierung verbessern, indem wichtige Eigenschaften der Oberfläche zuerst bestimmt werden und in die Berechnung des Vertexfehlers und somit in die Klassifizierung der Vertizes einfließen.

6.3.2 Vertexverschiebung

Die Vertexverschiebung greift auf eine Reihe von restriktiven Verfahren zurück, um eine parallele Bestimmung der Ersatzpositionen zu ermöglichen. Diese Vorgehensweise hat den Nachteil, dass durch die Tests gültige Ersatzpositionen gesperrt werden und Blockaden zwischen den Verschiebungskandidaten entstehen können. Die Behandlung dieses Zustandes resultiert in zusätzlich auszuführenden Iterationen.

Eine Verbesserung der parallelen Abarbeitung unter Berücksichtigung der benachbarten Verschiebungen bringt mehrere Vorteile. Einerseits kann die Behandlung von Blockaden weitgehend vermieden werden. Andererseits ermöglicht ein derartiger Ansatz zusätzliche Verschiebungskombinationen, die eine vorteilhaftere Vereinfachung erlauben.

Eine weitere Verbesserung lässt sich durch die Einbeziehung der benachbarten Verschiebungen bei der Auswahl des Verschiebungszieles erreichen. Werden benachbarte Vertizes auf mögliche Zielpositionen verschoben, ohne benachbarte Verschiebungen zu berücksichtigen, so kann das negative Auswirkungen auf die Qualität des vereinfachten Dreiecksnetzes haben, da etwa lange, schmale Dreiecke entstehen können. Des Weiteren ist die Bestimmung einer „optimalen“ Ersatzposition nicht möglich, wie sie bei iterativen Verfahren zum Einsatz kommt. Im Zuge dieser Arbeit wurde diese Einschränkung akzeptiert, um eine Reduktion der Laufzeit für die Vereinfachung zu erzielen. Eine Berücksichtigung von benachbarten Vertizes kann also eingesetzt werden, um die Qualität der Vereinfachung zu verbessern.

Die Parallelität des Verfahrens wird eingeschränkt durch die Bedingung, dass ein Vertex nur entfernt werden kann, wenn er einen beizubehaltenden Nachbarn besitzt. Der Grad der Parallelität ließe sich steigern, indem auch als zu entfernend markierte Vertizes behandelt werden könnten, die gegenwärtig keinen beizubehaltenden Nachbarn aufweisen. Die größere Anzahl an Verschiebungskandidaten könnte zu einer höheren Auslastung der Hardware führen und die Zahl der nötigen Iterationsschritte reduzieren.

Gemeinsam mit diesem Ansatz lässt sich die Freiheit von Verschiebungen verbessern, indem nicht nur beizubehaltende Vertizes als Verschiebungsziele in Frage kommen, sondern alle Nachbarn, die gegenwärtig kein Verschiebungskandidat sind. Für jeden Verschiebungskandidaten kann so die Zahl der möglichen Zielpositionen steigen. Durch die zusätzliche Freiheit für Verschiebungen kann es zu einer Reduktion der Zahl der Blocka-

den bzw. gesperrten Verschiebungen kommen. Mehr mögliche Ersatzpositionen bergen außerdem das Potential, die Qualität des vereinfachten Netzes zu verbessern, da unter Umständen Ersatzpositionen gewählt werden können, die eine genauere Repräsentation des originalen Netzes erlauben.

6.3.3 Reklassifzierung

Die Reklassifizierung ist weitgehend von den Metriken der Klassifizierung abhängig. Wie bereits oben erwähnt kann die Zahl der benötigten Reklassifizierungen drastisch reduziert werden, indem auf einen Algorithmus zurückgegriffen wird, der nicht nur einzelne Vertizes klassifiziert, sondern auch benachbarte Vertizes berücksichtigt.

Von einem derartigen Ansatz profitiert auch die Reklassifizierung. In der gegenwärtigen Form werden einzelne Vertizes reklassifiziert, wenn sie einen tolerierten Vertexfehler überschreiten. Benachbarte Vertizes werden entfernt, obwohl das gleichzeitige Löschen eine große Abweichung vom originalen Dreiecksnetz verursacht. Dieses Verfahren wurde gewählt, um eine hohe Leistung zu ermöglichen.

Eine gezielte Berücksichtigung der benachbarten Vertizes kann somit eine bessere Qualität des vereinfachten Netzes nach sich ziehen. Zusätzlich kommt es zu einer gleichmäßigeren Verteilung der Vertizes im finalen Dreiecksnetz. Der Nachteil dieses Ansatzes liegt in der erforderlichen Kommunikation zwischen den einzelnen Reklassifizierungen, was einen höheren Leistungsbedarf zur Folge hat.

Die Vertexgruppeneliminierung zeigt eine gute Laufzeit und Ausnutzung von moderner, paralleler Hardware und erfüllt damit die zu Beginn gestellten Anforderungen. Wie in Abschnitt 6.3 beschrieben, ist außerdem das Potential zur Verbesserung des Verfahrens vorhanden, um sowohl die Leistung als auch die Qualität der vereinfachten Objekte noch weiter zu steigern.

Abkürzungsverzeichnis

CUDA	Compute Unified Device Architecture
FIFO	First in/First out
GPU	Graphics processing unit
PM	Schrittweise Dreiecksnetze (engl. „progressive meshes“)
PVDPM	Parallele, blickwinkelabhängige Variante der schrittweisen Dreiecksnetze (engl. „Parallel View Dependent Refinement of Progressive Meshes“)

Abbildungsverzeichnis

Quelltext

Tabellenverzeichnis

Literaturverzeichnis

[Arv90] James Arvo. Transforming axis-aligned bounding boxes. In Andrew S. Glassner, Herausgeber, *Graphics Gems*, Seiten 548–550. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[Blo00a] Jonathan Blow. Terrain rendering at high levels of detail. In *Game Developers Conference Proceedings*, Seiten 903–912. CMP Media LLC, San Jose, CA, 2000.

[Blo00b] Jonathan Blow. Terrain rendering research for games, 2000. Course Notes.

[Bou10] Tamy Boubekeur. A view-dependent adaptivity metric for real-time mesh tessellation. In *IEEE International Conference on Image Processing (ICIP 2010)*, Seiten 3969 – 3972, 2010.

[Bro00] Dmitry Brodsky. Model simplification through refinement. In *Proceedings of Graphics Interface*, Seiten 221–228, 2000.

[BS96] Chandrajit L. Bajaj und Daniel R. Schikore. Error-bounded reduction of triangle meshes with multivariate data. In *Proceedings of SPIE Symposium on Visual Data Exploration and Analysis III*, Seiten 34–45, 1996.

[BW99] Dmitry Brodsky und Benjamin Watson. R-simp: Model simplification in reverse, a vector quantization approach, 1999. `www.cs.ubc.ca/~dima/pubs/wcgs99.pdf`.

[BW00a] Dimitry Brodsky und Benjamin Watson. Model simplification for interactive applications. In *Virtual Reality, 2000. Proceedings. IEEE*, Seiten 286–, 2000.

[BW00b] Dmitry Brodsky und Benjamin Watson. Parallelization, small memories and model simplification. In *Western Canadian Computer Graphics Symposium*, Seiten 75–83, 2000.

[Can11] Iain Cantlay. Directx 11 terrain tessellation, 2011.

[CG09] Chansophea Chuon und Sumanta Guha. Volume cost based mesh simplification. In *Proceedings of the 2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*, CGIV '09, Seiten 164–169, Washington, DC, USA, 2009. IEEE Computer Society.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, October 1976.

[COM98] Jonathan Cohen, Marc Olano, und Dinesh Manocha. Appearance-preserving simplification. In *PROC. SIGGRAPH 98*, Seiten 115–122, 1998.

[CRS96] Paolo Cignoni, Claudio Rocchini, und Roberto Scopigno. Metro: measuring error on simplified surfaces. Technical report, Consiglio Nazionale delle Ricerche, Pisa, Italy, 1996.

[CW02] Prasun Choudhury und Benjamin Watson. Completely adaptive simplification of massive meshes. Technical report, North Western University, 2002. `http://www.mccormick.northwestern.edu/eecs/documents/tech-reports/1999-2004/nwu-cs-02-091.pdf`.

[dB00] Willem H. de Boer. Fast terrain rendering using geometrical mipmapping, 2000. `www.flipcode.com/archives/article_geomipmaps.pdf`.

[DDFM+05] Emanuele Danovaro, Leila De Floriani, Paola Magillo, Enrico Puppo, Davide Sobrero, und Neta Sokolovsky. The half-edge tree: A compact data structure for level-of-detail tetrahedral meshes. In *Shape Modeling and Applications 2005*, Seiten 334–339. IEEE Computer Society, 2005.

[DG12] Evangij Derzapf und Michael Guthe. Dependency-free parallel progressive meshes. *Computer Graphics Forum*, 31(8):2288–2302, December 2012.

[DGG14] Evgenij Derzapf, Nico Grund, und Michael Guthe. Parallel Progressive Mesh Editing. In Margarita Amor und Markus Hadwiger, Herausgeber, *Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization*, Seiten 33–40. The Eurographics Association, 2014.

[DT07] Christopher DeCoro und Natalya Tatarchuk. Real-time mesh simplification using the GPU. In *Symposium on Interactive 3D Graphics (I3D)*,

volume 2007, Seite 6, April 2007.

[DX113] Microsoft DirectX 11 tesselation. `http://msdn.microsoft.com/en-us/library/windows/desktop/ff476340%28v=vs.85%29.aspx`, 2013. 2013.

[Eri00] Carl M. Erikson. *Hierarchical Levels Of Detail To Accelerate The Rendering Of Large Static And Dynamic Polygonal Environments.* PhD thesis, The University of North Carolina at Chapel Hill, 2000.

[ESV98] Jihad El-Sana und Amitabh Varshney. Topology simplification for polygonal virtual environments. *IEEE Trans. Vis. Comput. Graph.*, 4(2):133–144, 1998.

[ESV99a] Jihad El-Sana und Amitabh Varshney. Generalized view-dependent simplification. *Comput. Graph. Forum*, 18(3):83–94, 1999.

[ESV99b] Jihad El-Sana und Amitabh Varshney. View-dependent topology simplification. In *Virtual Environments '99*, 1999.

[FVD82] James D. Foley und Andries Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, und John F. Hughes. *Computer Graphics: Principles and Practice (2Nd Ed.).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Gar99] Michael Garland. *Quadric-based Polygonal Surface Simplification.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[GDG11] Nico Grund, Evgenij Derzapf, und Michael Guthe. Instant level-of-detail. In Peter Eisert, Joachim Hornegger, und Konrad Polthier, Herausgeber, *VMV*, Seiten 293–299. Eurographics Association, 2011.

[GH97] Michael Garland und Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, Seiten 209–216, 1997.

[GKM93] Ned Greene, Michael Kass, und Gavin Miller. Hierarchical z-buffer visibility. In *SIGGRAPH 93 Proceedings*, Seiten 231–240, 1993.

[Gre94] Ned Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In Paul S. Heckbert, Herausgeber, *Graphics Gems IV*, Seiten 74–82. Academic Press, 1994.

[GWH01] Michael Garland, Andrew Willmott, und Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, Seiten 49–58, New York, NY, USA, 2001. ACM.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John Alan McDonald, und Werner Stuetzle. Mesh optimization. In *ACM SIGGRAPH 1993 Proceedings*, Seiten 19–26. ACM, 1993.

[HH93] Paul Hinker und Charles D. Hansen. Geometric optimization. In Gregory M. Nielson und R. Daniel Bergeron, Herausgeber, *IEEE Visualization*, Seiten 189–195. IEEE Computer Society, 1993.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, Seiten 99–108, New York, NY, USA, 1996. ACM.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, Seiten 189–198, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Hop98a] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics*, 22(1):27–36, 1998.

[Hop98b] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the Conference on Visualization '98*, VIS '98, Seiten 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[HSH09] Liang Hu, Pedro V. Sander, und Hugues Hoppe. Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, Seiten 169–176, New York, NY, USA, 2009. ACM.

[HSH10] Liang Hu, Pedro V. Sander, und Hugues Hoppe. Parallel view-dependent level-of-detail control. *IEEE Trans. Vis. Comput. Graph.*, 16(5):718–728,

2010.

[JLG14] Qi Jia, Yongshan Liu, und Xiaoying Gu. Edge collapse mesh simplification algorithm based on detail features preserving. *Journal of Computational Information Systems 10*, 10(7):2883 – 2890, 2014.

[JRSW03] Justin Jang, William Ribarsky, Christopher D. Shaw, und Peter Wonka. Appearance-preserving view-dependent visualization. In Greg Turk, Jarke J. van Wijk, und Robert J. Moorhead II, Herausgeber, *IEEE Visualization*, Seiten 473–480. IEEE Computer Society, 2003.

[Kle97] Reinhard Klein. Multiresolution representations for surfaces meshes. In *Proceedings of the SCCG*, Seiten 57–66, 1997.

[KWA12] Maja Krivokuca, Burkhard Wünsche, und Waleed H. Abdulla. A new error metric for geometric shape distortion using depth values from orthographic projections. In *IVCNZ*, Seiten 388–393. ACM, 2012.

[LB15] Hélène Legrand und Tamy Boubekeur. Morton Integrals for High Speed Geometry Simplification. In Petrik Clarberg und Elmar Eisemann, Herausgeber, *Proceedings of the 7th Conference on High-Performance Graphics*, Seiten 105–112. ACM Siggraph, 2015.

[LCW+14] Wei Li, Yufei Chen, Zhicheng Wang, Weidong Zhao, und Lin Chen. An improved decimation of triangle meshes based on curvature. In *Rough Sets and Knowledge Technology*, volume 8818 of *Lecture Notes in Computer Science*, Seiten 260–271. Springer International Publishing, 2014.

[LE97] David Luebke und Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, Seiten 199–208. ACM Press/Addison-Wesley Publishing Co., 1997.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, Seiten 259–262, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LK16] Hyunho Lee und Min-Ho Kyung. Parallel mesh simplification using embedded tree collapsing. *The Visual Computer*, Seiten 1–10, 2016.

[LKH+00] Peter Lindstrom, David Koller, Larry F. Hodges, William Ribarsky, Nickolas Lea Faust, und Gregory Turner. Level-of-detail management for real-time rendering of phototextured terrain. Technical report, Visualization and Usability Center, Georgia Institute of Technology, 2000.

[LLM+10] Thomas Lewiner, Hélio Lopes, Esdras Medeiros, Geovan Tavares, und Luiz Velho. Topological mesh operators. *Computer Aided Geometric Design*, 27(1):1–22, 2010.

[LP02] Peter Lindstrom und Valerio Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, July 2002.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, und Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, Seiten 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LT97] Kok-Lim Low und Tiow Seng Tan. Model simplification using vertex-clustering. In Andy van Dam, Herausgeber, *SI3D*, Seiten 75–82, 188. ACM, 1997.

[LT00] Peter Lindstrom und Greg Turk. Image-driven simplification. *ACM Trans. Graph.*, 19(3):204–241, July 2000.

[Lue01] David Luebke. A developer's survey of polygonal simplification algorithms. *IEEE COMPUTER GRAPHICS AND APPLICATIONS*, 21:24–35, 2001.

[LWC+02] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, und Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[MG03] Alex Mohr und Michael Gleicher. Deformation sensitive decimation. Technical report, University of Wisconsin, 2003.

[NVG14] Nvidia Geforce GTX 780 Spezifikationen. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-780/specifications, 2014. 15.1.2014.

[Nvi13] Nvidia Corporation. *Nvidia CUDA C Programming Guide*, Juli 2013.

[OKK15] Hiromu Ozaki, Fumihito Kyota, und Takashi Kanai. Out-of-Core Framework for QEM-based Mesh Simplification. In C. Dachsbacher und P. Navrátil, Herausgeber, *Eurographics Symposium on Parallel Graphics and Visualization*, Seiten 87–96. The Eurographics Association, 2015.

[PJ03] Wouter Pasman und Frederik W. Jansen. Comparing simplification and image-based techniques for 3d client-server rendering systems. *IEEE Trans. Vis. Comput. Graph.*, 9(2):226–240, 2003.

[PP15] Alexandros Papageorgiou und Nikos Platis. Triangular mesh simplification on the GPU. *The Visual Computer*, 31(2):235–244, 2015.

[Pup98] Enrico Puppo. Variable resolution triangulations. *Computational Geometry*, 11:219–238, 1998.

[QM06] Lijun Qu und Gary W. Meyer. Perceptually driven interactive geometry remeshing. In Marc Olano und Carlo H. Séquin, Herausgeber, *SI3D*, Seiten 199–206. ACM, 2006.

[RB92] Jarek Rossignac und Paul Borrell. *Multi-resolution 3D Approximations for Rendering Complex Scenes*. IBM Research Division, T. J. Watson Research Center, 1992.

[RCRG06] Francisco Ramos, Miguel Chover, Oscar Ripolles, und Carlos Granell. Continuous level of detail on graphics hardware. In Attila Kuba, László G. Nyúl, und Kálmán Palágyi, Herausgeber, *DGCI*, volume 4245 of *Lecture Notes in Computer Science*, Seiten 460–469. Springer, 2006.

[Red96] Martin Reddy. Scrooge: Perceptually-driven polygon reduction. *Comput. Graph. Forum*, 15(4):191–203, 1996.

[RHS98] Stefan Röttger, Wolfgang Heidrich, und Hans-Peter Seidel. Real-time generation of continuous levels of detail for height fields. In *Proceedings of WSCG '98*, Seiten 315–322, 1998.

[RLB10] José Ribelles, Angeles López, und Oscar Belmonte. An improved discrete level of detail model through an incremental representation. In John P. Collomosse und Ian J. Grimstead, Herausgeber, *TPCG*, Seiten 59–66. Eurographics Association, 2010.

[Sau01] Stephen R. Saucier. View-dependent simplification for glyph-based visualizations of large datasets. In *2001 CADIP Research Symposium Proceedings*, 2001.

[Sch97] William J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization 97 Proceedings*, Seiten 205–212, 1997.

[SG01] Eric Shaffer und Michael Garland. Efficient adaptive simplification of massive meshes. In Thomas Ertl, Kenneth I. Joy, und Amitabh Varshney, Herausgeber, *IEEE Visualization 01 Proceedings*. IEEE Computer Society, 2001.

[SLA15] David Salinas, Florent Lafarge, und Pierre Alliez. Structure-aware mesh decimation. *Computer Graphics Forum*, Seite 20, January 2015.

[SM05] Pedro V. Sander und Jason L. Mitchell. Progressive buffers: View-dependent geometry and texture for lod rendering. In Mathieu Desbrun und Helmut Pottmann, Herausgeber, *Symposium on Geometry Processing*, volume 255 of *ACM International Conference Proceeding Series*, Seiten 129–138. Eurographics Association, 2005.

[SN13] Suzanne M. Shontz und Dragos M. Nistor. Cpu-gpu algorithms for triangular surface mesh simplification. In *Proceedings of the 21st International Meshing Roundtable*, Seiten 475–492, 2013.

[SW03] Scott Schaefer und Joe Warren. Adaptive vertex clustering using octrees. In *SIAM Geometric Design and Computing*, Seiten 491–500, 2003.

[SZL92] William J. Schroeder, Jonathan A. Zarge, und William E. Lorensen. Decimation of triangle meshes. *SIGGRAPH Comput. Graph.*, 26(2):65–70, July 1992.

[Tse14] Juin-Ling Tseng. Surface simplification of 3d animation models using robust homogeneous coordinate transformation. *Journal of Applied Mathematics*, 2014, 2014.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, Seiten 55–64, New York, NY, USA, 1992. ACM.

[WWLH11] Jiacai Wang, Lirong Wang, Jinzhu Li, und Ichiro Hagiwara. A feature preserved mesh simplification algorithm. In *Journal of Engineering and Computer Innovations*, volume 2, Seiten 98–105, 2011.

[WWZJ13] Jian Wang, Hai-Ling Wang, Bo Zhou, und Ni Jun. An efficient mesh simplification method in 3d graphic model rendering. *2013 Seventh International Conference on Internet Computing for Engineering and Science*, 0:55–59, 2013.

[XESV97] Julie C. Xia, Jihad El-Sana, und Amitabh Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Trans. Vis. Comput. Graph.*, 3(2):171–183, 1997.

[XV96] Julie C. Xia und Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th Conference on Visualization '96*, VIS '96, Seiten 327–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[Zac02] Christopher Zach. Integration of geomorphing into level of detail management for realtime rendering. In *Proceedings of the 18th Spring Conference on Computer Graphics*, SCCG '02, Seiten 115–122, New York, NY, USA, 2002. ACM.

[Zhu01] Yuanchen Zhu. Real-time continuous level-of-detail terrain rendering with nested splitting space, 2001. Intel International Science and Engineering Fair.

[ZJK01] Pan Zhigeng, Shi Jiaoying, und Zhou Kun. A new mesh simplification algorithm based on triangle collapses. *J. Comput. Sci. Technol.*, 16(1):57–63, January 2001.

[ZMK02] Christopher Zach, Stephan Mantler, und Konrad F. Karner. Time-critical rendering of discrete and continuous levels of detail. In *VRST*, Seiten 1–8. ACM, 2002.

Index