Mesh Simplification

Michael Thomas

Berlin, 24. Januar 2011

Inhaltsverzeichnis

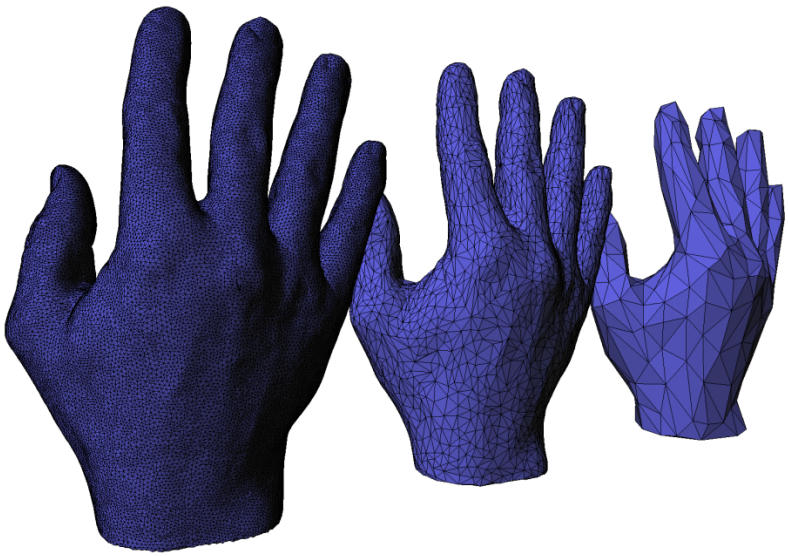


Illustration Mesh Simplfication. Quelle: [Cac06]

1 Einleitung

Die Vereinfachung von Oberflächen ist, gerade bei dem rasanten Fortschritt des digitalen Zeitalters eine zunehmend wichtige Anwendung der Computational Geometry geworden. Digitale Repräsentation von detaillierten Oberflächen und Objekten, 3-dimensionale Abbilder unserer realen Welt bis hin zu auschließlich am Computer modellierter virtueller Landschaften und Szenarien für Computerspiele und Filme sind heutzutage fester Bestandteil unseres Alltags.
Dabei sind die Möglichkeiten solche Daten von Oberflächenmodellen zu gewinnen oder durch den Computer zu generieren, sowie die Qualität und Auflösung dieser in den letzten Jahren immer weiter gestiegen. Wichtige Disziplinen sind z.B. Laserscanner zur räumlichen Erfassung von Objekten für die 3d-Modellierung, Range-Scanner in der Computer Vision zur Erkennung der Umgebung und Hindernissen, Satelliten zur Abtastung der Erdoberfläche und Erzeugung von Geländedaten, aber auch die Gewinnung von Oberflächeninformationen aus Volumen Daten, wie sie unter Anderem in der Medizin oder Materialforschung Anwendung finden. Verwendung finden diese Daten in vielen Anwendungsbereichen, die bekanntesten liegen wohl im virtuellen Design für Filme und Computerspiele, CAD, Kartographie und Scientific Visualization.
Bei der *Surface Simplification*, im Speziellen oft auch *Mesh Simplification* genannt,

handelt es sich um eine Klasse von Algorithmen, welche durch verschiedene Methoden versuchen, komplexe 3d-Modelle in Form von Polygonnetze durch einfachere Modelle anzunähern, wobei möglichst charakteristische Merkmale für die Erkennbarkeit erhalten bleiben sollen. Sie stellt somit das Gegenstück zur *Subdivision* dar, bei der einfache Oberflächen durch Unterteilung in komplexere, glättere Oberflächen überführt werden.
Warum sollten nun Oberflächen überhaupt vereinfacht werden? Die meisten Verfahren zur Erzeugung von Oberflächendaten lassen dies nur mit einer sehr hohen, fest gewählten Auflösung zu. Weiterhin ist der Grad der Details bei der Visualisierung von Oberflächen natürlich von der Anzeige sowie durch die Wahrnehmung des Betrachters begrenzt. Es ist offensichtlich, dass die Speicherung und Verarbeitung dieser Daten im direkten Zusammenhang mit ihrer Komplexität steht. In fast allen Anwendungsbereichen finden sich Algorithmen welche sich durch einen reduzierten Detailgrad der Oberflächen stark beschleunigen lassen. Beispiele dafür sind Rendering, Kollisionserkennung, Analyse von Strukturen und Formen. Oft macht das Arbeiten mit vereinfachten Oberflächen eine Anwendung in Echtzeit überhaupt erst möglich. Auch die Speicherung und Verbreitung der Daten, z.B. über das Internet ist einfacher wenn die Datenmenge klein ist.
Diese Arbeit soll als eine Art Einführung in den Bereich der Surface Simplification dienen. Es werden verschiedene Ideen zu möglichen Algorithmen betrachtet und mathematische Vorüberlegungen angestellt. Am Ende soll dann ein Algorithmus von Garland und Heckbert vorgestellt werden, welcher auf *Edge Contraction* basiert und sehr gute Ergebnisse liefert.

2 Anwendungsbereiche

Zur Veranschaulichung soll in diesem Kapitel noch einmal genauer auf verschiedene Anwendungsbereiche eingegangen werden. Eine gute Übersicht dazu ist in [HG97] zu finden.

Computergraphik Die Computergraphik hat heutzutage ein sehr breites Anwenwendungsspektrum, dazu gehören Scientific Visualization, Computer-Aided Design, Virtual Design / Virtual Reality. Gerade bei Computerspielen oder anderen 3D Animationen ist oft eine Berechnung in Echtzeit wichtig. Dabei lässt sich ausnutzen, dass der Fokus auf Objekten liegt, die sich im Vordergrund befinden, und es für die Darstellung von weiter entfernten Objekten einer weitaus geringeren Auflösung bedarf. Dafür werden in der Regel für

jedes Objekt mehrere Modelle mit verschiedenen Detailleveln bereitgestellt, wobei man eine Funktion definieren kann zwischen der Größe des Objektes in der Darstellung, und der Auflösung. Je kleiner ein Objekt dargestellt wird, desto geringer muss die Auflösung des Polygonnetzes sein. Man nennt dies auch *multiresolutional modelling*. Einige Simplification Algorithmen bieten die Möglichkeit eine Berechnung nur einmal durchzuführen, und sich die jeweiligen Operation zwischen drin zu merken. So können mit geringem Aufwand aus dem hochaufgelösten Modell, Modelle mit beliebigen geringeren Auflösungen erstellt werden. Darauf soll später genauer eingegangen werden.

Kartographie Durch die Möglichkeit mithilfe von Satelliten hochauflösende Bilder und Tiefen Informationen von Geländeoberflächen (*terrain fields*) zu erzeugen, macht die Surfaces Simplification zu einer wichtigen Anwendung in der modernen Kartographie. Die Daten weisen mitunter sehr hohe Auflösungen auf, mit viel Rauschen und redundanten Informationen, welche somit entfernt werden können. Auch bei der Erzeugung von Karten mit unterschiedlichen Maßstäben ist eine Simplifizierung von Küstenlinien, Straßen, Flüssen und Höheninformationen gefragt.

Ingenieurwesen Bei Planung von Brücken, Autos, Flugzeugen etc. wird oft das physikalische Verhalten wie Aerodynamik oder Krafteinwirkung der zu konstruierenden Objekte an Simulationen getestet und optimiert.. Auch hier liegen die Modelle als Polygonnetze vor und müssen entsprechend vereinfacht werden, um Simulationsalgorithmen, für z.B. die Luftströmung um ein Flugzeugflügel.

3 Taxonomie

In den letzten 20 Jahren gab es viel Forschungsarbeit auf dem Gebiet der Surface Simplification. Wir betrachten dabei Algorithmen welche ausschließlich auf triangulierten Oberflächen arbeiten. Wichtige Kriterien für den Vergleich sind Effizienz, Qualität und Generalizität. Effizienz bezeichnet die Geschwindigkeit der Algorithmen im Verhältnis zur Größe der Ein- und Ausgabe Modelle, Qualität die Abweichung des vereinfachten Modells zum Original in Abhängigkeit von der Anzahl der reduzierten Facetten, sowie die Erhaltung der Topologie. Mit Generalizität wird unterschieden auf welchen Klassen von Objekten die Algorithmen funktionieren, wobei hier verschiedene topologische Typen differenziert werden,

angefangen bei einfachen Mannigfaltigkeiten, über Mannigfaltigkeiten mit Rändern, bis zu generellen 2-Komplexen. Eine sehr gute Übersicht über die verschiedenen Publikationen zu dem Thema Surface Simplification liefern Garland und Heckbert in [HG97] sowie in [GH97].
Fast alle heute relevanten Algorithmen lassen sich grob in die drei folgenden Kategorien unterteilen.

Vertex Clustering Vertex Clustering ist ein sehr naiver Ansatz, der zwar hohe Performance liefert jedoch nur sehr geringe Qualität aufweist. Der Raum, in dem das Modell liegt wird basierend auf einem Eingabeparameter in ein 3-dimensionales Gitternetz unterteilt. Alle Knoten die in einer Zelle liegen werden dabei zu einem Knoten zusammengefasst. Da allein die Zellgröße des Gitters entscheidend für den Grad der Vereinfachung ist, wird klar dass dieser Algorithmus sehr wenig Kontrolle darüber bietet. Weiter kann es zu Verzerrungen des Modells kommen, da Dichte der Vertices sehr ungleich verteilt sein kann, wohingegen das Gitter eine statische Größe hat.

Vertex Decimation Dies ist ein weiterer Ansatz, zuerst entwickelt von Schroeder et al. [SZL92]. Grundlegend bei diesem Konzept ist, dass bestimmte Vertices und seine Facetten (anliegende Kanten und Dreiecke) entfernt werden. Dadurch entstehen Löcher, welche dann neu trianguliert werden.
Unterschiede zwischen den verschiedenen Algorithmen dieser Klasse, sind die Wahl der Fehlermetriken zur Klassifizierung der zu eliminierenden Vertices sowie die Algorithmen die zur Triangulierung verwendet werden. Gemein ist ihnen, dass sie in der Regel eine annehmbare Effizienz und Qualität liefern, sowie eine gute Erhaltung der Topologie bieten. Sie sind jedoch beschränkt auf Mannigfaltigkeiten mit Rändern.

Edge Contraction Algorithmen basierend auf Edge Contraction vereinfachen Polygonnetze indem sie die zwei Endpunkte einer Kante durch einen einzelnen Vertex ersetzen. Dabei wird die Kante entfernt und die restlichen Kanten der beiden Vertices mit dem neuen Punkt verbunden. Alle degenerierten Facetten, also Dreiecke, die zu Linien kollabieren, sowie Kanten welche zwei gleiche Vertices miteinander verbinden werden entfernt. Die Algorithmen arbeiten iterativ, wobei in jedem Schritt ein Kante kontrahiert wird. Dabei verschwinden in der Regel zwei Dreiecke, nämlich diejenigen welche die kontrahierte Kante als Facette hatten.
Auch hier unterscheiden sich die Algorithmen in der Wahl der Fehlermetrik,

die bestimmt, welche Kanten kontrahiert werden, und wie der neue Vertex gewählt wird. Edge Contraction Algorithmen bieten eine sehr gute Qualität mit weichen Übergängen, bei angemessener Effizienz. Problematisch ist die Erhaltung der Topologie. Die erste Idee zur Edge Contraction wurde in einem Artikel von Hoppe at al. [Hop96] veröffentlicht. Garland und Heckbert lieferten einen weiteren wichtigen Beitrag durch ihre auf der Summe quadratischer Distanzen beruhende Fehlermetrik sowie die Verallgemeinerung des Algorithmus auf generelle 2-Komplexe [GH97]

4 Bestandteile eines Surface Simplification Algorithmus

Mit Ausnahme des *Vertex Clustering* Algorithmus laufen alle Algorithmen nach dem gleichen, einfachen Schema ab:

```
REPEAT
    wähle Element mit dem kleinsten Fehler
    führe eine Vereinfachungsoperation aus (Removal/Contraction)
    berechne den neuen Fehler
UNTIL Abbruchskriterium erreicht
```

4.1 Fehlerkontrolle

Der Fehler bezeichnet hierbei, inwieweit sich das vereinfachte Polygonnetz von seinem Original unterscheidet. Der oben genannte Algorithmus arbeitet iterativ und führt pro Schritt eine Dezimierungsoperation aus. Dabei ist es wünschenswert diejenigen Elemente (Kanten oder Vertices, je nach Algorithmus) auszuwählen, welche bei der Dezimierung den kleinsten Fehler verursachen. Würde durch die Vereinfachung ein wichtiges Detail, wie z.B. ein Horn einer Kuh verloren gehen, so sollte dies als sehr teuer bewertet werden. Bei einer relativ ebenen Oberfläche hingegen würde durch Entfernung von Dreiecken ein geringer Informationsverlust auftreten.
Man unterscheidet hier grob zwischen *globaler* und *lokaler* Fehlerkontrolle.

globale Fehlerkontrolle Hier wird die Differenz des zu dezimierenden Ausschnitts zum Original betrachtet. Dies liefert natürlich die bestmögliche Fehlerkontrolle, ist in der Regel jedoch sehr rechen- und speicheraufwendig, und wird daher selten als Fehlermaß in iterativen Algorithmen angewendet, kann

jedoch gut eingesetzt werden um verschiedene Algorithmen in ihrer Qualität zu vergleichen.

lokale Fehlerkontrolle Anstatt eine Region mit dem ursprünglichen Netz zu vergleichen, kann sie auch einfach mit dem Netz nache letzten Iterationsschritt verglichen werden. Das ist wesentlich effizienter da man nun mit einer schon vereinfachten Oberfläche vergleicht. Nachteil besteht hierin, dass sich die Fehler akkumulieren.

Als Maß für den Fehler, die sogenannte Fehlermetrik werden in der Regel euklidische Abstände von Vertices zu der ursprünglichen Oberfläche verwendet. Auch hier lässt sich der Rechenaufwand reduzieren, indem der Abstand zu einer Durchschnittsebene gemessen wird, welche den Mittelwert der Ebenen in der Region des Vertex bildet. Eine sehr gute Methode beschreiben hier Garland und Heckbert, auf welche später genauer eingegangen werden soll.

4.2 Abbruchskriterium

Der Algorithmus läuft so lang, bis ein bestimmtes Abbruchskriterium erreicht wird. Das heißt, die gewünschte Anzahl von Facetten, oder eine bestimmte Qualität wurde erreicht.

4.3 Implementierungsdetail

In der Implementierung wird hier meistens eine Prioritätswarteschlänge, z.B. ein *Heap* verwendet, wobei alle Knoten bzw. Kanten nach ihrem Fehlermaß geordnet werden. So wird vorne immer das Element entnommen, welches bei Dezimierung den kleinsten Fehler verursacht.

Eine gut illustrierte Einführung befindet sich sich in den Vorlesungsfolien von A. Sheffer [SG08]

5 Edge Contraction Algorithmus

Wie bereits erwähnt, vereinfacht der *Edge Contraction* Algorithmus Polygonnetze indem schrittweise Vertexpaare die mit einer Kante verbunden sind zu einem einzelnen Vertex zusammengezogen werden. In diesem Kapitel wollen wir nun genauer

auf den Algorithmus eingehen. Zunächst soll versucht werden, den Algorithmus formal durch mathematische Operationen zu beschreiben. Im Anschluss daran wird ein gutes Konzept zur Fehlermessung basierend auf der Summe quadratische Distanzen vorgestellt und zum Ende soll noch auf die Problematik der Topologieerhaltung eingegangen werden.

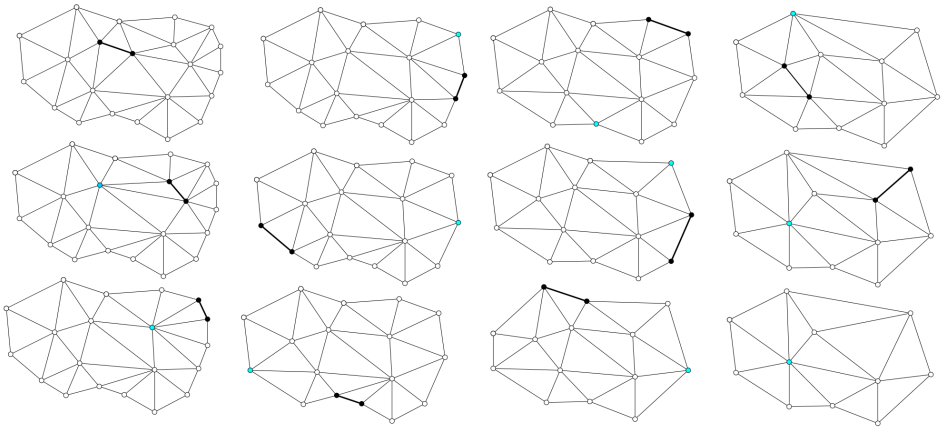


Abbildung 1: *Sequenz von Edge Contraction, von oben nach unten und rechts nach links. Die fetten Kanten markieren eine Kante die entfernt wird, die blauen Punkte den neuen Vertex*

5.1 Mathematische Definition

5.1.1 Wiederholung kombinatorische Topologie

Wir erinnern uns zurück an des vorherige Thema, und fassen unsere zu vereinfachende triangulierte Oberfläche als 2-Komplex K auf, bestehend aus k-Simplices. Ein 0-Simplex bezeichnen wir als Vertex (Punkt), 1-Simplex als Kante und ein 2-Simplex als Dreieck. Die konvexe Hülle einer nichtleeren Teilmenge von Vertices eines k-Simplex heißt Facette, mit einer Dimension $l \leq k$. Die Facetten eines Dreiecks sind also das Dreieck selbst, seine Kanten und seine Eckpunkte. Simplex τ ist Facette von σ wird notiert als $\sigma \geq \tau$. Der Stern eines Simplex $\sigma \in K$ sind all jene Simplices, von denen σ eine Facette ist. Bei einem Vertex also sind dies alle Kanten und Dreiecke von denen er Eckpunkt ist. Der Abschluss einer Menge von Simplices L ist der kleinste Subkomplex, der L enthält. Als Link eines Simplex σ bezeichnet man alle Facetten τ des abgeschlossenen Sterns, die nicht σ nicht schneiden. Anschaulich betrachtet wäre das im bei einem Vertex seine Nachbarn, sowie die Kanten, welche sie verbinden.

Der zugrunde liegende Raum eines Simplizialkomplexes notieren wir als $|K|$ wobei wir hier zunächst nur von 2-Mannigfaltigkeiten ausgehen. D.h. für alle Vertices des Komplex gilt, dass sein Link seiner Nachbarn einen geschlossenen Kreis bildet. Formal gesehen ist die Nachbarschaft eines Vertex homöomorph zu einer offen Teilmenge von R^2, es gibt also keinen Vertex der an einem Rand liegt. Dies wird später wichtig, wenn wir uns mit der Erhaltung der Topologie beschäftigen.

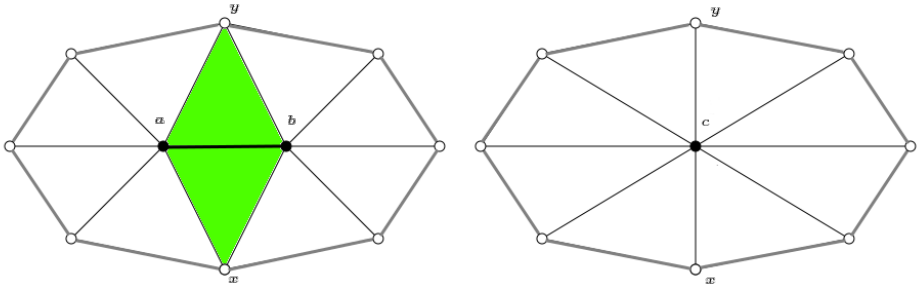


Abbildung 2: *Kontraktion einer Kante ab zu dem Vertex c, es verschwinden die zwei grünen Dreiecke. Die graue Kontur inkl. der weißen Vertices ist der Link von $\overline{ab} = \{a, b, ab\}$ (links) und Link von c (rechts).*

5.1.2 Formale Beschreibung einer *Edge Contraction*

Wir betrachten uns nun eine *Edge Contraction* Operation auf einer Kante $ab \in K$ wobei K ein wie oben definiertes 2-Komplex ist. Die Kante verschwindet und wird durch einen neuen Punkt c ersetzt. Dabei verschwinden die zwei an die Kante angrenzenden Dreiecke. Der neue Vertex c wird nun mit den Kanten verbunden, die vorher Kanten von a und b waren. Wie interessieren uns nun dafür, wie sich dieser Sachverhalt formal beschreiben lässt und wie der neu entstandene Komplex aussieht.

Dazu definieren wir uns zunächst einen Kegel von einem Punkt x zu einem k-Simplex σ, als ein Simplex $x \cdot \sigma$, welches entsteht, wenn man von jedem Vertex $p \in \sigma$ eine Kante zu x zieht. Das neu entstandene Simplex hat dabei die Dimension $k + 1$, unter der Annahme, dass x in einer neuen Dimension liegt, also affin unabhängig zu dem von dem Simplex σ aufgespannten Unterraum ist. Der Kegel einer Kante ab und einem Punkt x, wobei der Punkt nicht auf der Geraden durch Kante liegt, ist nun also ein Dreieck abx. Wir notieren das ganze wie folgt: Sei x ein Punkt und σ ein Simplex, dann ist der *Kegel* $x \cdot \sigma = conv(\{x\} \cup \sigma)$. Analog dazu können wir nun auch einen Kegel von einem Punkt x zu einer Menge von Simplices T

bilden, in dem wir einen Kegel von x zu jedem Simplex $\tau \in T$ aufspannen, also $x \cdot T = \{x \cdot \tau | \tau \in T\}$. Weiter definieren wir uns auch den *Stern* und *Link* einer Menge von Simplices, wobei $\overline{T} = ClT - \emptyset$ als der Abschluss der Menge T ohne das (-1)-Simplex definiert ist:

$$St\, T = \{\sigma | \sigma \geq \tau \in T\}$$

$$Lk\, T = Cl\, St\, T - St\, \overline{T}$$

Anschaulich betrachtet ist der Stern einer Menge T von Simplices, die Menge aller Kanten und Dreiecke die an die Simplices der Menge T grenzen, inklusive T selbst. Der Abschluss des Sterns ist der Stern selbst plus sein Rand, also alle Vertices und Kanten die noch fehlen um die Menge zu einem vollständigen Komplex zumachen. Der Link der Menge T ist die Differenz der von dem abgeschlossenen Stern und dem Stern von T, es bleiben also nur die Vertices und Kanten am Rand des abgeschlossenen Sterns. Ein Beispiel für den Link von $\overline{ab} = \{a, b, ab\}$ ist in **Abb.** 2 gegeben. Der Stern in diesem Beispiel wären alle inneren Dreiecke und Kanten, der abgeschlossene Stern das ganze Komplex.
Mit diesen Definitionen lässt sich nun die Kontraktion einer Kante ab als eine mathematische Operation, welche einen Komplex K auf einen L abbildet, wie folgt beschreiben:

$$L = K - St\, \overline{ab} \cup c \cdot Lk\, \overline{ab}$$

D.h. wir gehen von K aus, entfernen alle Simplices welche im Stern der Menge $\overline{ab} = \{a, b, ab\}$ enthalten sind, das sind also alle Kanten und Dreiecke, die an a, b und ab angrenzen. Anschließend fügen wir einen neuen Vertex c ein und ziehen Kanten von c zu allen Vertices im Link der Menge $\overline{ab}$ ein.

5.2 Multiresolutional Modelling

In vielen Anwendungsbereichen ist es erforderlich, dass ein darzustellendes Modell in mehreren Auflösungen vorliegt, ein Konzept, dass sich *multresoltional modelling* nennt. Dabei ist es wünschenswert die Auflösung möglichst kontinuierlich anzupassen. Man stelle sich z.B. ein Computerspiel vor: Die Kamara nähert sich einem entfernten Objekt und es wird beinahe stufenlos immer detailreicher dargestellt.
In diesem Abschnitt soll nun untersucht werden, wie der *Edge Contraction* Algorithmus dazu verwendet werden kann.

5.2.1 Inkrementelle Repräsentation

Wir haben gesehen, wie sich die Kontraktion von genau einer Kante in einem Simplizialkomplex als eine mathematische Operation darstellen lässt, welche aus dem Komplex K einen neuen, vereinfachten Komplex L erzeugt.
Wie schon erwähnt arbeitet der Algorithmus iterativ, d.h. er führt in jedem Schritt eine solche Operation aus. Uns interessiert nun, wie sich das Endergebnis und alle Zwischenergebnisse des Algorithmus formal darstellen lassen.
Sei K_1 also der Komplex, den man erhält, wenn man aus dem ursprüngliche Komplex $K_0 = K$ die Kante ab kontrahiert, wobei die Vertices a und b auf den neuen Vertex c abgebildet werden. Diese Aktion lässt sich mit einer *Vertex Map* beschreiben, welche die Vertices aus K_0 auf die Vertices aus K_1 abbildet. Genau genommen ist dies eine Funktion $g_1 : \ Vert\, K_0 \to Vert\, K_1$ die wie folgt definiert ist:

$$g_1(u) = \begin{cases} u, if\ u \notin \{a, b\}, \\ c, if\ u \in \{a, b\} \end{cases} .$$

Man beachte, dass diese Abbildung surjektiv ist, da sie auf allen Vertices von K_0 definiert ist: ein Vertex wird $u \in K_0$ entweder auf sich selbst oder auf den neuen Vertex c abgebildet.
Der *Edge Contraction* Algorithmus liefert also formal betrachtet eine Folge der Abbildungen g_i, welche K_{i-1} auf das um eine Kante dezimierte Komplex K_i abbildet, genau genommen auf seine Vertices. Man erhält so das vereinfachte Polygonnetz $L = K_{n-m}$ durch die Anwendung des *Contraction* Algorithmus auf K, in dem man die einzelnen Aktionen $g_i : \ Vert\, K_{i-1} \to Vert\, K_i$ für alle $1 \leq i \leq n - m$ nacheinander ausführt, wobei n und m jeweils für die Anzahl der Vertices in K und L stehen, $n = |Vert\, K|$ und $m = |Vert\, L|$. Die resultierende Abbildung ist also die Verkettung der einzelnen Operationen g_i:

$$g = g_{n-m} \circ \cdots \circ g_2 \circ g_1 \ : Vert\, K \to Vert\, L$$

Speichert man nun alle Operationen g_i, welcher der Algorithmus erzeugt, so lassen sich ausgehend von dem ursprünglichen Netz K_0 auch das vereinfachte Netz L und alle dazwischenliegenden Annäherungen K_i reproduzieren:

$$K_0 \overset{g_1}{\to} K_1 \overset{g_2}{\to} K_2 \overset{g_3}{\to} \cdots \overset{g_{n-m}}{\to} L$$

Dies nennt man auch die *inkrementelle Repräsentation* von K_0. Dieses Vor-

gehen ist jedoch für die meisten Anwendungen nicht praktikabel, da jeweils das hochauflösende Originalmodell gespeichert werden muss plus alle Operationen, siehe [Gar99].

5.2.2 Progressive Meshes

Eine einzelne *Contraction* Operation, welche eine Kante kontrahiert ist auch invertierbar. Wir definieren uns zu jeder *Contraction* Operation ϕ_i, die der Algorithmus ausgibt eine inverse Operation ψ_i welche einen sogenannten *vertex split* ausführt, und wie folgt definiert werden kann: $\psi_i(K_i) = K_{i-1}$. Jede dieser *Split* Operation verzeichnet den Vertex, der geteilt werden soll, sowie die Position der zwei neuen Vertices und Dreiecke die entstehen. Nun können wir unser Modell quasi *bottom up* rekonstruieren und erhalten ein *Progressive Mesh*

$$K_{n-m} \overset{\psi_{n-m}}{\rightarrow} \cdots \overset{\psi_2}{\rightarrow} K_2 \overset{\psi_1}{\rightarrow} K_1 \overset{\psi_0}{\rightarrow} K_0$$

Die Idee für *Progressive Meshes* stammt ursprünglich von Hoppe [Hop96], und bietet eine sehr effiziente Möglichkeit viele verschiedene Auflösungen eines Modells zu rekonstruieren. Weiter Optimierungen dieser Methode sind in [Gar99] sehr gut beschrieben.

5.3 Fehlerberechung durch Quadriken

Nun bleibt noch die Frage zu klären, welche Kanten ab für eine Kontraktion ausgewählt werden, und wo der neue Vertex c platziert werden soll. Nicht jede Kante ist eine gute Wahl, es könnte u.U. die Geometrie stark verändert werden. Wir brauchen also ein Art Kostenfunktion welche die Kosten für eine Kontraktion liefert. Für die Wahl des Punktes liegt der naive Ansatz nahe, ihn genau in der Mitte der kontrahierten Kante zu positionieren, also $c = 0.5(a + b)$. Die Qualität der Approximierung bei einer statischen Wahl des Punktes lässt jedoch zu Wünschen übrig. Viel besser wäre es, wenn wir ein adaptives Maß finden könnten, welche den entstehenden Fehler minimieren würde.
In diesem Abschnitt werden wir eine interessante Methode kennenlernen, mit der es möglich ist die beiden oben genannten Probleme gemeinsam zu lösen. Die Idee dieses Verfahrens ist, für jeden neuen Vertex den Abstand zu der Menge von Ebenen zu approximieren, welche durch die an ihn angrenzenden Dreiecke aufgespannt werden. Sie stammt ursprünglich Garland und Heckbert [GH97].

5.3.1 Abstand von einem Punkt zur Ebene

Zunächst wollen wir eine kleine Wiederholung in linearer Algebra machen und uns ansehen, wie man den Abstand eines Punktes zu einer Ebene berechnet.
Eine Ebene in $\mathbb{R}^3$ ist definiert durch ihren Normalenvektor $\vec{n}$ und einem Ortsvektor $\vec{a}$. Jeder Punkt $x = \{x_1, x_2, x_3\}$ liegt in der Ebene wenn er die Gleichung $(x^T - \vec{a}) \cdot \vec{n} = 0$ erfüllt. Wobei x^T die Transponierung von x ist und den Vektor zu x beschreibt. Zur Abstandsberechnung ziehen wir die Ebenengleichung in Hessescher Normalform heran. Im Folgenden werde ich aus Gründen der Übersichtlichkeit auf die Vektorpfeile verzichten. Sei $n_0 = (a, b, c)^T$ der normalisierte Normalenvektor der Ebene, dann lässt sich die Ebene H wie folgt darstellen:

$$\begin{aligned} & H : (x^T - a) \cdot n_0 = 0 \\ \Leftrightarrow\ & x^T \cdot n_0 + d = 0 \\ \Leftrightarrow\ & ax_1 + bx_2 + cx_3 + d = 0 \end{aligned}$$

Der skalare Wert d berechnet sich dabei aus dem Skalarprodukt von n_0 und a: $d = -n_0 \cdot a$. Der Abstand eines Punkt x zur Ebene H lässt sich dann sehr einfach berechnen:

$$\begin{aligned} d(x, H) &= x^T \cdot n_0 + d \\ &= ax_1 + bx_2 + cx_3 + d \cdot 1 \\ &= (x_1, x_2, x_3, 1)^T \cdot (a, b, c, d)^T \\ &= \mathbf{v}^T \cdot \mathbf{p} \end{aligned}$$

Wie man sieht, lässt sich die Abstandsberechnung in $\mathbb{R}^3$ als ein Skalarprodukt in $\mathbb{R}^4$ auffassen. Der Punkt $\mathbf{v} \in \mathbb{R}^4$ ist definiert als $\mathbf{v} = (x, 1) = (x_1, x_2, x_3, 1)$ und $\mathbf{p} = (n_0, d) = (a, b, c, d)^T$ ist der Vektor der sich aus den Koeffizienten der Ebene in Hessescher Normalform ergibt.

5.3.2 Fundemental Quadric

Wie oben erwähnt, brauchen wir zur Berechnung der Kosten einer Kontraktion eine Fehlermetrik $E(x)$. Sie soll uns ein Maß für den Fehler liefern, der entsteht, wenn eine Kante $\mathbf{ab}$ durch die Operation $(\mathbf{ab}) \rightarrow \mathbf{c}$ zu dem neuen Vertex $\mathbf{c}$ kontrahiert wird.
Wir nehmen dazu die Summe der quadratischen Distanzen des Vertex v zu den

Ebenen $planes(\mathbf{v})$, welche durch die Dreiecke aufgespannt werden, an die er grenzt:

$$\begin{aligned} E(\mathbf{x}) &= \sum_{\mathbf{p} \in planes(\mathbf{v})} d(\mathbf{v}, \mathbf{p})^2 \\ &= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{v}^T \cdot \mathbf{p})^2 \end{aligned}$$

Warum hier die Summe der quadratischen Distanzen genommen wird, anstatt die einfache Summe werden wir gleich sehen. Der neue Vertex $\mathbf{c}$ ist zum Zeitpunkt der Fehlerberechnung noch nicht bekannt, daher approximieren wir die an ihn angrenzenden Ebenen durch die Vereinigung der Ebenen von $\mathbf{a}$ und $\mathbf{b}$: $planes(\mathbf{c}) = planes(\mathbf{a}) \cup planes(\mathbf{b})$. Dazu müssten jedoch im laufe des Algorithmus alle Ebenen mitgeführt werden; die Menge würde sich demnach bei jeder Iteration vergrößern, und für die Berechnung müssten alle Ebenen einzeln herangezogen werden. Dies ist offensichtlich sehr speicher- und rechenaufwändig. Nun können wir die quadratische Summe ausnutzen und die Formel derart umschreiben, dass wir den Teil, der die Ebenen beschreibt extrahieren:

$$\begin{aligned} E(\mathbf{x}) &= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{v}^T \cdot \mathbf{p})^2 \\ &= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{v}^T \cdot \mathbf{p}) \cdot (\mathbf{v}^T \cdot \mathbf{p}) \\ &= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{v}^T \cdot \mathbf{p}) \cdot (\mathbf{p}^T \cdot \mathbf{v}) \\ &= \sum_{\mathbf{p} \in planes(\mathbf{v})} \mathbf{v}^T \cdot (\mathbf{p} \cdot \mathbf{p}^T) \cdot \mathbf{v} \\ &= \mathbf{v}^T \cdot \left(\sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{p}\mathbf{p}^T) \right) \cdot \mathbf{v} \\ &= \mathbf{v}^T \cdot \left(\sum_{\mathbf{p} \in planes(\mathbf{v})} \mathbf{K_p} \right) \cdot \mathbf{v} \end{aligned}$$

Das Produkt eines Vektors p mit seiner transponierten Darstellung p^T ist das Matrixprodukt. In diesem Fall ist $\mathbf{p}\mathbf{p}^T = \mathbf{K_p}$ eine 4x4 Matrix, da $\mathbf{p} = (a, b, c, d) \in \mathbb{R}^4$. Sie wird als *fundamental quadric* bezeichnet, und beschreibt die Abstandsfunktion eines Punktes zu einer einzelnen Ebene.

$$\mathbf{K_p} = (a,b,c,d) \cdot (a,b,c,d)^T = \begin{pmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{pmatrix}$$

Matrizen besitzen die wunderbare Eigenschaft, dass sie beliebig addierbar sind, und wieder eine Matrix der selben Größe ergeben. Wir können nun den Abstand eines Punktes zu einer beliebigen Anzahl von Ebenen durch eine einzige Matrix $\mathbf{Q}$ ausdrücken, wobei $\mathbf{v} = (x_1, x_2, x_3, 1)$:

$$E(x) = \mathbf{v}^T \mathbf{Q} \mathbf{v}$$

Die initiale Matrix $\mathbf{Q}$ wird am Anfang des Algorithmus für jeden Vertex v im Polygonnetz berechnet, indem alle *fundamental quadrics* $\mathbf{K_p}$ für die an v angrenzenden Dreiecksebenen aufsummiert werden:

$$\mathbf{Q} = \sum_{\mathbf{p} \in planes(\mathbf{v})} \mathbf{K_p}$$

Die Vereinigung der Ebenen $planes(\mathbf{a}) \cup planes(\mathbf{b})$ lässt sich nun als Summe der zwei Matrizen $(\mathbf{Q_1} + \mathbf{Q_2})$ darstellen wobei $\mathbf{Q_1}$ und $\mathbf{Q_2}$ die Abstandsmatrizen für die Punkte a und b definieren.

5.3.3 Position des neuen Vertex

Die erste Frage nach einer effizienten Kostenfunktion haben wir geklärt. Jetzt müssen wir noch den neuen Vertex c geschickt positionieren. Die Beste Position für c ist offensichtlich an der Stelle, an der $E(x)$ minimal ist. Da $E(x)$ eine quadratische Funktion in $\mathbb{R}^4$ darstellt, können wir die Minimierung auf ein lineares Problem zurückführen, in dem wir den Gradienten $\nabla E = (\partial E/\partial x_1, \partial E/\partial x_2, \partial E/\partial x_3) = \vec{0}$ setzen. Die partiellen Ableitungen $\partial E/\partial x_i$ sind dabei lineare Funktionen.
Im folgenden wollen wir dies durch eine Rechnung demonstrieren. Multiplizieren

wir unsere Fehlerfunktion aus, erhalten wir:

$$\begin{aligned} E(x) &= \mathbf{v}^T \mathbf{Q} \mathbf{v} \\ &= q_{11}\, x_1^2 + 2q_{12}\, x_1 x_2 + 2q_{13}\, x_1 x_3 + 2q_{14}\, x_1 \\ &\quad + q_{22}\, x_2^2 + 2q_{23}\, x_2 x_3 + 2q_{24}\, x_2 \\ &\quad + q_{33}\, x_3^2 + 2q_{34}\, x_3 \\ &\quad + q_{44} \end{aligned}$$

Wenn wir diese Funktion nun jeweils nach x_i ableiten, erhalten wir den Gradienten ∇E wie folgt:

$$\begin{aligned} \partial E/\partial x_1 &= 2q_{11}\, x_1 + 2q_{12}\, x_2 + 2q_{13}\, x_3 + 2q_{14} \\ \partial E/\partial x_2 &= 2q_{12}\, x_1 + 2q_{22}\, x_2 + 2q_{23}\, x_3 + 2q_{24} \\ \partial E/\partial x_3 &= 2q_{13}\, x_1 + 2q_{23}\, x_2 + 2q_{33}\, x_3 + 2q_{34} \end{aligned}$$

Der Punkt $\mathbf{v} = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3})$, welcher $E(x)$ minimiert ist nun die Lösung eines linearen Gleichungssystem mit drei Gleichungen:

$$\begin{aligned} q_{11}\, x_1 + q_{12}\, x_2 + q_{13}\, x_3 + q_{14} &= 0 \\ q_{12}\, x_1 + q_{22}\, x_2 + q_{23}\, x_3 + q_{24} &= 0 \\ q_{13}\, x_1 + q_{23}\, x_2 + q_{33}\, x_3 + q_{34} &= 0 \end{aligned}$$

$$\Leftrightarrow$$

$$\begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{pmatrix} \cdot \mathbf{v}^T + \begin{pmatrix} q_{14} \\ q_{24} \\ q_{34} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Man beachte, dass dieses LGS nicht unbedingt eine eindeutige Lösung haben muss. Der Punkt kann entweder auf einer Geraden oder in einer Ebene liegen, in diesem Fall muss auf ein naives Verfahren zurückgegriffen werden.

5.3.4 Ablauf des Algorithmus

Wir haben gelernt wie man mit Hilfe von Quadriken eine effiziente Kostenfunktion für eine *Edge Contraction* erstellen kann, welche den Fehler recht gut approximiert. Auch haben wir gesehen, wie wir diese Kostenfunktion dazu nutzen können, eine

optimale Position für den neuen Vertex zu bestimmen. Nun wollen wir diese Methoden noch einmal zusammenfassen und uns einen Überblick darüber machen, wie der *Edge Contraction* Algorithmus im Ganzen abläuft, so wie er von Garland und Heckbert [GH97] beschrieben wurde. Wir gehen dabei von einem Polygonnetz K aus.

1. Berechne alle initialen Matrizen $\mathbf{Q}$ für jeden Vertex $v \in Vert\, K$
2. Berechne die optimale Position des neuen Vertex c der bei Kontraktion der Kante ab entstehen würde, für alle Kanten. Speichere für jede Kante die Kosten für die Kontraktion als $\epsilon = \mathbf{v^T}(\mathbf{Q_1} + \mathbf{Q_2})\mathbf{v}$, wobei $\mathbf{v} = (c, 1)$ die Koordinaten des neuen Vertex c um eine 1 ergänzt. $\mathbf{Q_1}$ und $\mathbf{Q_2}$ stehen dabei für die Abstandsmatrizen der Vertices a und b
3. Speichere alle Kanten in einer *priority queue*, z.B. einem Heap, wobei die Kosten ϵ als Ordnung herangezogen werden, d.h. die Kante mit den kleinsten Kosten ganz oben.
4. Entferne iterativ jeweils die Kante mit den geringsten Kosten aus der *queue* und führe eine *Edge Contraction* aus.
5. Aktualisiere die Abstandsmatrizen für den neuen Vertex c und alle Kanten, welche bei der Kontraktion verändert wurde.

Die einfache Aufsummierung der Abstandsmatrizen hat den Nachteil, dass eventuell manche Ebenen mehrfach gezählt werden, falls die Mengen der Ebenen in $\mathbf{Q_1}$ und $\mathbf{Q_2}$ nicht disjunkt sind, wodurch das Fehlermaß ungenau wird. Eine Verbesserung der Methode durch das Inklusion-Exklusion Prinzip wird in [Ede01] beschrieben.
Zu dem Laufzeitverhalten des Algorithmus wurden keine Angaben gefunden. Man kann davon ausgehen dass die Berechnung nicht trivial ist. In dem Beispiel in **Abb.** 7 wurden jeweils 15sec angegeben, was keine Berechnung in Echtzeit vermuten lässt. Wie in Kapitel 5.2 erläutert ist es jedoch möglich, die optimalen Kontraktionen vorberechnen zu lassen, wodurch eine Echtzeitanwendung garantiert ist.

5.4 Topologie Erhaltung

In den letzten Kapiteln haben wir uns mit der Funktionsweise des *Edge Contraction* Algorithmus befasst. Nun wollen wir untersuchen, welche Auswirkung die Kontraktion einer Kante auf die Topologie des Modells haben könnte.

Wir werden sehen, dass der topologische Typ der Oberfläche in speziellen Fällen verändert werden kann, was in vielen Anwendungsfällen nicht gewünscht ist. Betrachtet man z.B. eine Oberfläche mit einem Loch wie in **Abb.** 3, so ist es möglich, dass im Laufe der Kontraktionen das Loch durch ein einziges Dreieck approximiert wird. Wird nun eine der Kanten von diesem Dreieck kontrahiert, so würde das Dreieck verschwinden. Der topologische Typ wäre verändert worden, da es keine innere Kante mehr gibt.
Im Folgenden werden wir eine Methode kennenlernen, wie man Kanten danach klassifizieren kann, ob Sie bei Kontraktion die Topologie verändern oder nicht.. Zunächst sehen wir uns dazu einfache Mannigfaltigkeiten an und anschließend Mannigfaltigkeiten mit Rändern, so wie das Beispiel mit dem Loch.

5.4.1 Mannigfaltigkeiten

Betrachten wir also 2-Komplexe welche eine Mannigfaltigkeit triangulieren. Wie weiter oben schon erwähnt, lassen sich Mannigfaltigkeiten dadurch charakterisieren, dass jeder Punkt der Oberfläche $x \in |K|$ eine Nachbarschaft homöomorph zu einer offenen Teilmenge von $\mathbb{R}^2$ ist. Anschaulich betrachtet bedeutet dies, dass jeder Vertex im Innern einer Scheibe liegt, nämlich den Raum, der von seinem Stern aufgespannt wird. Analog dazu ist der Link des Vertex, also das Liniensegment, was entsteht, wenn man alle seine Nachbarn verbindet, ein geschlossener Kreis. In **Abb.** 4 ist dies gut zu sehen.
Führen wir nun eine *Edge Contraction* auf K aus, so kann es u.U. passieren, dass diese Bedingung für Mannigfaltigkeiten verloren geht. In **Abb.** 5 ist ein Beispiel für so einen Fall demonstriert. Es handelt sich dabei um einen Tetraeder der aus einer Ebenen Fläche herausragt. Wird die Kante ab kontrahiert, bleibt nur noch ein einzelnes Dreieck übrig, welches senkrecht auf der Fläche steht. Die Nachbarschaftsbedingung für Mannigfaltigkeiten gilt hier offensichtlich nicht mehr: Die Spitze des Dreiecks, hier als y bezeichnet, ist ein Randpunkt. Seine Nachbarschaft ist ein Halbkreises. Auch für die Vertices z und c gilt diese Bedingung nicht mehr.
Ein weiteres Beispiel wäre die Triangulierung eines Torus, oder Hohlzylinders, deren Oberflächen zwar Mannigfaltigkeiten sind, jedoch nicht homöomorph zu einer Kugel. Analog zur der oben genannten Scheibe mit Loch ist es auch hier möglich, dass das Innere geschlossen wird, und so eine Oberfläche homöomorph zu einer Kugel entstehen würde.
In dem Beispiel in **Abb.** 4 hingegen bleibt die Topologie erhalten. Die Nachbarschaft des neuen Vertex c bildet eine Scheibe.

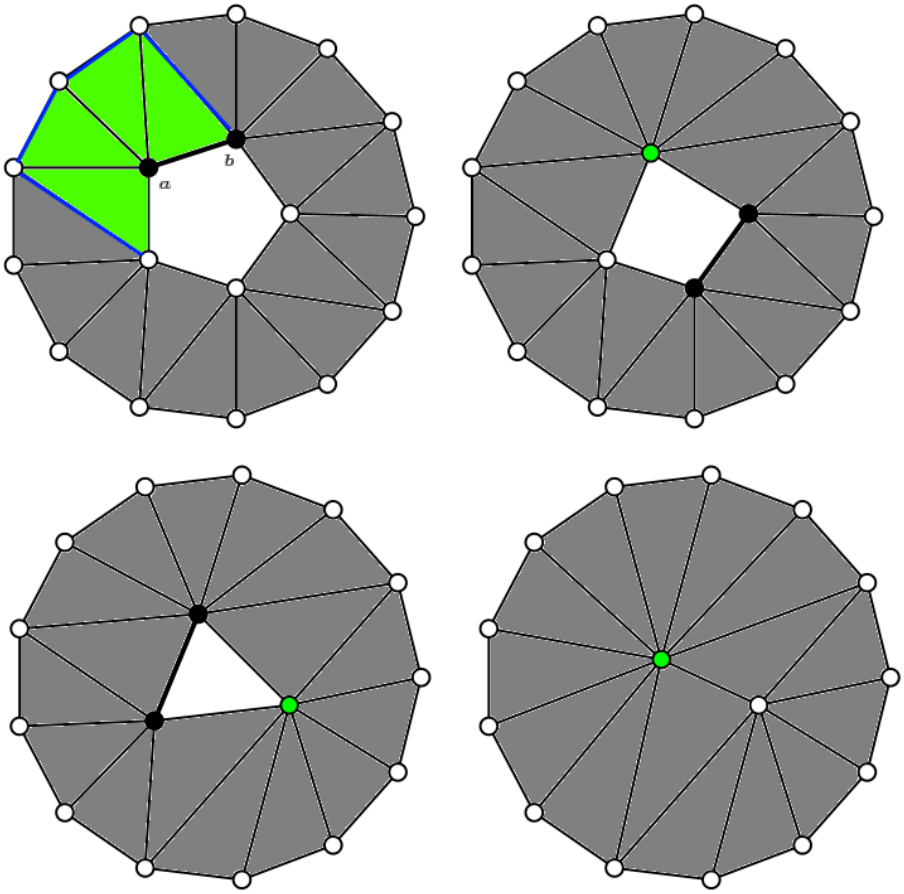


Abbildung 3: *Das Loch in der Mitte wird im Laufe der Kontraktionen zu einem Dreieck. Wird nun eine der Kanten kontrahiert, verschwindet es. Die grüne Fläche links oben ist die Fläche unter dem Stern $|St\, a|$, sie homöomorph zu einer Halbscheibe. Die blaue Kontur und ihre Ecken beschreiben den Link von a*

Wir interessieren uns nun dafür, wie man eine Topologie erhaltende Kontraktion von anderen unterscheiden kann. Sieht man sich die Beispiele genau an, stellt man fest, dass dies etwas damit zu tun hat, an welchen Punkten sich die Links von a und b treffen. Die Topologie bleibt genau dann erhalten, wenn die Schnittmenge der Links von a und b gleich dem Link der Kante ab ist:

Link Condition Lemma A Sei K die Triangulierung einer 2-Mannigfaltigkeit. Die Kontraktion von $ab \in k$ erhält den topologischen Typ dann und nur dann wenn $Lk\, a \cap Lk\, b = Lk\, ab$

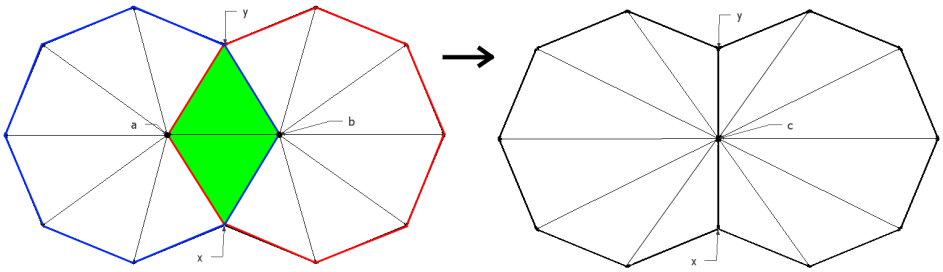


Abbildung 4: *Die Topologie bleibt bei der Kontraktion von ab erhalten. Die blaue Kontur kennzeichnet den Link von a, die rote den Link von b. Die grünen Dreiecke verschwinden.*

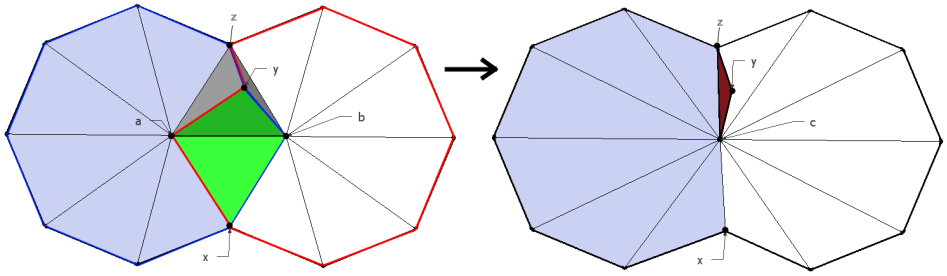


Abbildung 5: *Die Topologie bleibt bei der Kontraktion von ab nicht erhalten. Die grünen Dreiecke verschwinden, und es bleibt ein einzelnes Dreieck, welches senkrecht auf der Fläche steht (hier rot dargestellt). Die blaue Kontur ist der Link von a, die rote Link von b*

Das Lemma stammt aus [DEGN98]. Wer sich für den Beweis interessiert kann dort nachlesen.

In **Abb.** 4 ist die Gültigkeit des Lemmas gut erkennen: Der Link von a und b ist der rote bzw. blaue Kreis. Sie schneiden sich im Punkt x und y. Der Link der Kante ab sind genau diese Punkte, $Lk\, ab = \{x, y\} = Lk\, a \cap Lk\, b$, die Kontraktion von ab erhält die Topologie.

Im Beispiel aus **Abb.** 5 ist diese Bedingung nicht erfüllt. Der Link von a und b schneiden sich in den Punkten x, y und z wobei der Link der Kante ab nur x und y enthält. $Lk\, ab = \{x, y\} \neq Lk\, a \cap Lk\, b = \{x, y, z\}$, die Kontraktion von ab erhält die Topologie **nicht**.

5.4.2 Mannigfaltigkeiten mit Rändern

Die oben genannte Scheibe mit Loch ist ein gutes Beispiel für eine Mannigfaltigkeit mit Rand. Ihre Triangulierung besitzt neben den inneren Vertices auch Randvertices, also solche, die eine Nachbarschaft in Form einer Halbscheibe haben. Formal gesehen ist der Raum, welche dem Stern des Vertex zugrunde liegt homöomorph zu einer Halbscheibe: $|St\, u| \approx \mathbb{H}^2$.
Auch hier lässt sich das *Link Condition Lemma* anwenden, wenn wir unsere Komplex mit einem kleinen Trick modifizieren, in dem wir uns daraus eine Mannigfaltigkeit bauen.
Angenommen K enthält $l \geq 1$ Löche. Dann denken wir uns einen neuen Vertex ω und bilden einen Kegel zu jeden C_i welches das Loch umgibt, zu diesen Vertex ω, wie **Abb.** 6 in illustriert, und erhalten K^ω Offensichtlich gilt nun für die Vertices, welche vorher am Rand der Löcher lagen wieder die Bedingung für Mannigfaltigkeiten. Nur für ω selbst muss dies nicht unbedingt gelten.
Definieren wir jetzt unseren Link Lk^ω als den Link auf K^ω so lässt sich das oben genannte *Link Condition Lemma* analog zu Mannigfaltigkeiten definieren:

Link Condition Lemma B Sei K die Triangulierung einer 2-Mannigfaltigkeit. Die Kontraktion von $ab \in k$ erhält den topologischen Typ dann und nur dann wenn $Lk^\omega\, a \cap Lk^\omega\, b = Lk^\omega\, ab$

Auch dieses Lemma stammt aus [DEGN98]. In diesem Artikel wird gezeigt, dass sich auch für generelle 2-Komplexe eine entsprechende *Link Condition* formulieren lässt. Die Beweise dazu sind jedoch sehr umfangreich, daher soll an dieser Stelle darauf verzichtet werden.

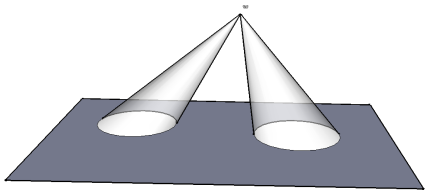
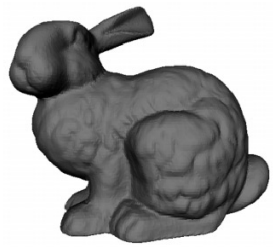


Abbildung 6: *Die Löcher werden geschlossen, indem ein Kegel von jedem Kreis zu dem Dummyvertex ω gespannt werden*

6 Fazit

Betrachtet wurde hier nur ein sehr kleiner Teil des gesamten Themengebietes der Surface Simplification. Die grundlegenden Ideen die beschrieben wurden sind schon mehr als 10 Jahre alt, inzwischen hat sich das Forschungsspektrum auf diesem Gebiet stark ausgeweitet. Die Algorithmen wurden verbessert, viele Modifikationen und Optimierungen für verschiedene Anwendungsfälle sind vorgenommen worden. Dennoch hoffe, dass die Idee der Surface Simplification und ihre Anwendung in dieser Arbeit gut vermittelt wurde. Wer sich darüber hinaus mehr mit dem Thema befassen möchte findet in der Bibliographie viele interessante Links mit freizugänglichen wissenschaftlichen Publikationen.

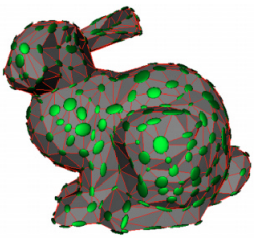
7 Beispiele



(a) Original mit 69,451 Dreiecken

(b) 1,000 Dreiecken

(c) 100 Dreiecken



(d) 1,000 Dreiecke, grüne Fehler Ellipsoide

Abbildung 7: *Ein Kaninchen Modell mit verschiedenen Approximierungen, generiert mit dem Garland und Heckbert Algorithmus. Für die Laufzeit wurden jeweils 15sec angegeben. Quelle: [GH97]*

Literatur

[Cac06] Fernando Cacciola. *Triangulated Surface Mesh Simplification*, 2006. http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Surface_mesh_simplification/Chapter_main.html.

[DEGN98] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S*, 66:23–45, 1998.

[Ede01] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, May 2001.

[Gar99] Michael Garland. Multiresolution modeling: Survey & future opportunities. In *EUROGRAPHICS '99*, 1999. http://mgarland.org/files/papers/STAR99.pdf.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, August 1997. http://www.cs.cmu.edu/~garland/quadrics/.

[HG97] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH 97 Proc.*, May 1997. http://www.cs.cmu.edu/~ph.

[Hop96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96 Proc.*, pages 99–108, August 1996. http://research.microsoft.com/~hoppe/.

[SG08] A. Sheffer and C. Gotsman. Surface simplification. Technical report, University of British Columbia, 2008. http://www.cs.ubc.ca/~sheffa/dgp/sylabus.html.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, July 1992.